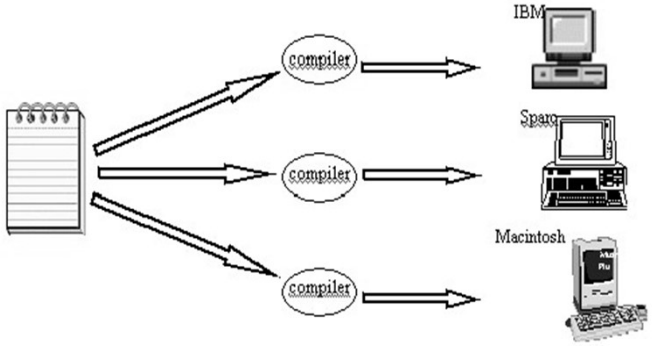
01. Lập Trình Java

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

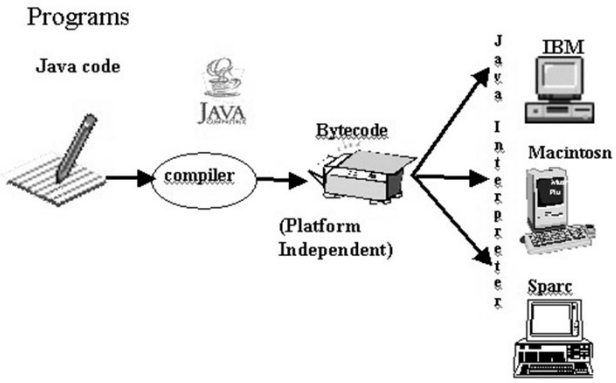
Mục Tiêu

- Tất cả những điều cần biết và không được quên về Java
  - Tại sao cần học Java?
  - Kiến trúc Java .
  - Chương trình Java làm việc như thế nào?
  - Java "bytecode"
- Sẽ học những gì?
  - Ngôn ngữ lập trình Java - Java programming language
  - Các lớp thư viện Java - Java class library (APIs)
- Chương trình Java được tạo (create), biên dịch compile) và chạy (run) như thế nào?
  - Java SE --> JDK tools
  - JRE.
- Chương trình Java đầu tiên

Các chương trình dịch truyền thống

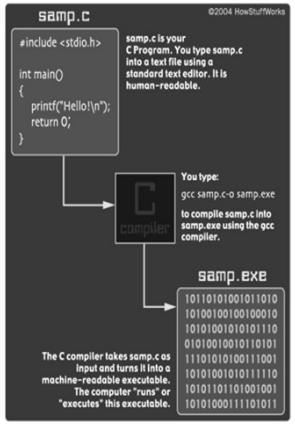


Chương trình dịch Java



…nhận thấy

- Chương trình viết bằng C, mã nguồn (source code) được biên dịch thành ngôn ngữ máy gốc (native) bào gồm những số 1 và 0
- Ngôn ngữ máy được xác định bởi HĐH - Operating System (Windows, Mac, UNIX or Linux, Androi, Window phone).
- Vậy, có thể có một module chương trình nào (đã được dịch) có thể chạy trên mọi nền HDH?



5

Các giải pháp của Microsoft (trước năm 2000)

- Công cụ:
  - Visual Studio 6.0.
- Ngôn ngữ lập trình:
  - Visual Basic (VB), Visual C++.
- Môi trường thực thi - Runtime environment
  - Windows Only.

6

JAVA, giải pháp của Sun Microsystems

- Cha đẻ của Java
  - PhD. James Gosling
  . CTO of Sun's Developer Products.

Sun microsystems
We make the net work.

Java

7

Java là cái gì?

- Tên thương mại do Sun đưa ra để nói đến các kỹ thuật để tạo và thực thi các chương trình phần mềm trên môi trường máy đơn và máy mạng một cách an toàn và hiệu quả

Sun microsystems
We make the net work.

Java

8

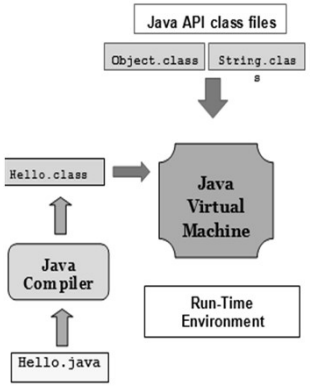
Kiến trúc của Java - Java architecture

1. Ngôn ngữ lập trình Java - Java Programming Language
2. Các file class của Java (các file dạng mã bycode)
3. Thự viện các lớp Java APIs
    1. API, Application Programming Interface
4. Máy ảo Java - Java Virtual Machine - JVM

9

Java làm việc như thế nào?

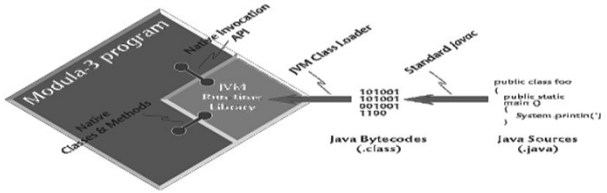
1. Chương trình nguồn (source code) được viết bằng ngôn ngữ Java
2. Các chương trình được biên dịch thành các file dạng lớp (*. Class)
3. Các file .class được nạp vào bộ nhớ và thực thi bởi máy ảo Java (JVM)



10

JVM và Java "bytecode"

- Chương trình Java không biên dịch mã nguồn thành ngôn ngữ máy đích mà biên dịch thành file dạng "bytecode" – file *.class
- Mỗi HĐH sẽ có thể hiện riêng của máy ảo Java –JVM
- Mã bytecode làm việc với JVM và JVM làm việc với HĐH



11

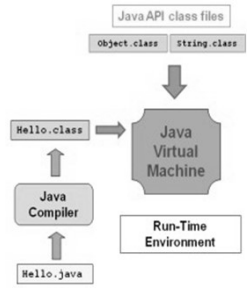
Máy ảo Java – Java virtual machine - JVM

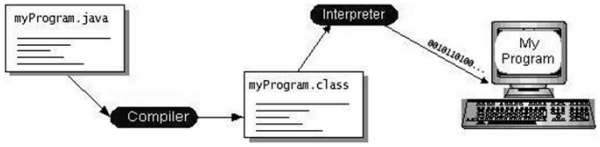
- Là 1 phần mềm – được xem là "bộ máy thực thi" (execution engine) – dùng để thực thi các mã bycode (*.class) trên mọi nền (platform) một cách an toàn và tương thích.

12

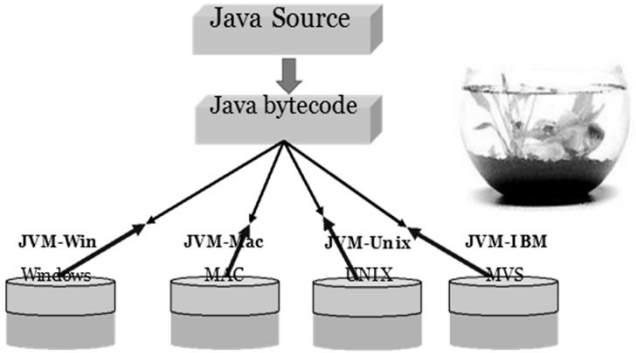
JVM làm việc như thế nào

- Chương trình được biên dịch thành dạng bytecode
- JVM thông dịch mã bytecode thành mã máy đích





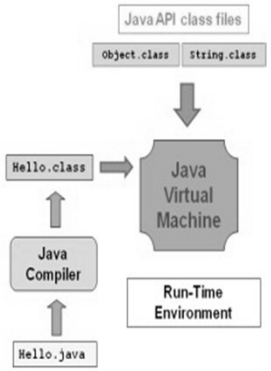
13



14

Chương trình Java được thực thi như thế nào?

- JVM là 1 phần của Sun Java Runtime Environment, Standard Edition (JRE)
- JVM không phải là một chương trình độc lập
- Để chạy ứng dụng Java, HĐH phải cài đặt JRE



15

Java Runtime Environment (JRE)

- Là 1 tập con của Java Development Kit (JDK)- bộ công cụ phát triển các ứng dụng Java
- JRE bao gồm JVM, các lớp căn bản (core Java) và các file hỗ trợ
- Lấy JRE ở đâu?
  - Access Sun website
  - Google it

16

Chương trình Java được xây dựng như thế nào?

17

Để tạo 1 chương trình Java

- Có thể sử dụng các công cụ soạn thảo văn bản đơn giản.
- File chương trình có phần mở rộng .java
  - HelloWord.java

18

Biên dịch, thử và kiểm lỗi

- Sử dụng môi trường lập trình Java và công cụ
- Phụ thuộc vào kiểu ứng dụng Java sẽ có môi trường lập trình tương ứng.

19

Môi trường lập trình Java (Java programming Environment) - JDK

- Java SE *(Java platform, Standard Edition)*
  - Là gói dùng để phát triển phần mềm của Sun
  - Cung cấp tập cơ bản các công cụ cần thiết để viết, test và kiểm lỗi các ứng dụng viết bằng Java (application và applet)
  - Phiên bản hiện hành là Java SE 7 (JDK 7u7)
- Java EE *(Java Platform, Enterprise Edition)*
  - Cho các ứng dụng enterprise server
- Java ME *(Java Platform, Micro Edition)*
  - Cho các ứng dụng trên các thiết bị điện tử gia dụng, các thiết bị nhúng…(for consumer and embedded servers and applications)

20

Bộ công cụ Java SE 7U7 (JDK 7U7)

- javac:
  - Trình biên dịch mã nguồn (*.java) thành mã bytecode (*.class)
- java:
  - Trình thông dịch được sử dụng để thực thi mã bytecode
- appletviewer:
  - Được sử dụng để xem và test applets
- javadoc:
  - Trình tạo tài liệu dạng HTML cho các chương trình nguồn và các gói

21

Các công cụ phát triển trực quan

- Java Studio Enterprise.
- Sun Java Studio Creator.
- Borland JBuilder
- NetBeans.
- JDeverloper.
- Eclipse
- Jcreator
- ….

22

Khi học lập trình Java, chúng ta sẽ học ?

1. Ngôn ngữ lập trình Java.
2. 2. Các lớp thư viện của Java (Java APIs)

23

Ngôn ngữ lập trình Java

- Java có cú pháp tương tự C.
  - Mọi điều học từ C đều rất hữu dụng trong Java.
- Rất quan trọng:
  - Java là ngôn ngữ hướng đối tượng (OOP)
  - Mọi thứ trong ngôn ngữ Java đều được xem như đối tượng
- Không bao giờ được quên điều này

24

Các tính năng của Java

- Hướng đối tượng
- Độc lập với mọi nền phần cứng - Platform-independent
- Mạnh mẽ
  - Tất cả dữ liệu đều phải được khai báo 1 cách tường minh
  - Kiểm code vào thời điểm biên dịch va thông dịch
  - Giới hạn các lỗi của chương trình
- Bảo mật
  - Xây dựng môi trường bảo mật cho việc thực thi chương trình
  - Có rất nhiều mức khác nhau cho việc điều khiển bảo mật
- Phân tán
  - Có thể chạy trên mọi nền phần cứng, mọi HĐH
  - Hỗ trợ ứng dụng chạy trên mạng
- Đa tuyến
  - Thực hiện nhiều nhiệm vụ đồng thời trong một ứng dụng

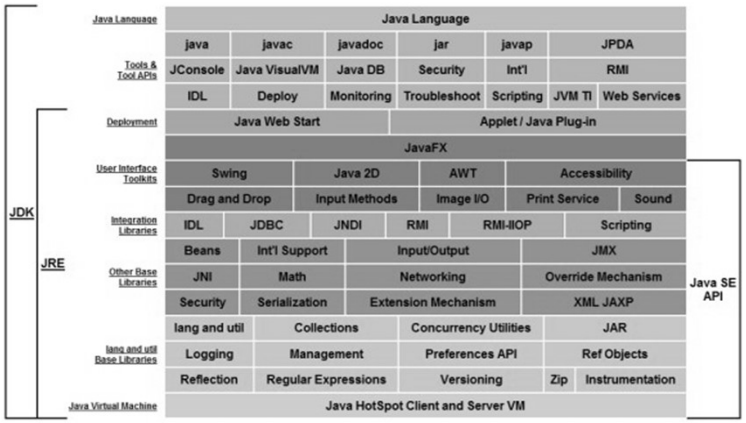
25

Các lớp thư viện Java (Java APIs)

- Java APIs (Application Programming Interface)
- Các lớp này làm đơn giản quá trình phát triển các ứng dụng Java cho người lập trình. Chúng giúp cho LTV viết các chương trình phức tạp một cách nhanh chóng
- Để làm chủ Java, ta phải hiểu rõ về các lớp thư viện này.

26

Các lớp thư viện của Java (Java SE)



27

Tóm lại!

- Để lập trình với Java, ta cần phải biết:
  1. Ngôn ngữ lập trình Java.
  2. Các lớp thư viện của Java (Java APIs)

OOP

28


29

01. Lập Trình Java

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

OOP overview …

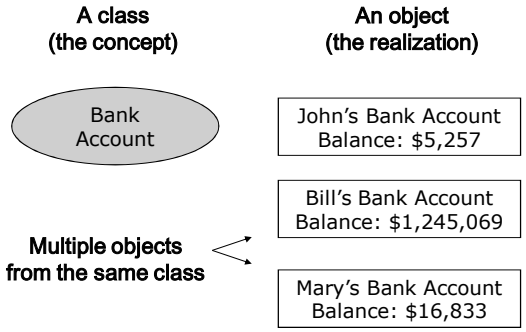
Nội Dung

- OOP overview
- Encapsulation
- Passing object as parameter
- Java interface
- Inheritance
- Polymorphism

Classes

- Object → Class
  - Một đối tượng được xác đ ịnh bởi một lớp. Lớp là một bảng thiết kế chi tiết (blueprint) của một đối tượng, là khuôn mẫu để sinh ra đối tượng.
  - Một lớp chứa định danh, thuộc tính (dữ liệu) và các hành vi (phương thức)
  - Đối tượng là thể hiện (instance) của mộ t lớp. Đối tượng phải được tạo ra một cách tường minh bằng toán tử new
- Attributes → Data declarations
  - Khai báo dữ liệu được đặt bên trong lớp, nhưng không được đặt bên trong bất kỳ phương thức nào.
- Behaviors → Method declarations

Objects and Classes example



Encapsulation
(Bao đóng /che dấu thông tin)

- Bao đóng là che đi phần hiện thực, và chỉ cung cấp cho bên ngoài các chức năng. Mục đích là làm cho việc thay đổi trong hiện thực của một module không ảnh hưởng tới các module đã sử dụng nó.
- Che dấu thông tin
  - Thao tác với dữ liệu thông qua các giao diện xác định.
  - Che dấu người lập trình khách (client programmer).

Encapsulation
(Bao đóng /che dấu thông tin)

- Bao đóng : che dấu mọi chi tiết hiện thực của đối tượng, không cho bên ngoài thấy và truy xuất ⇒ tạo độ độc lập cao giữa các đối tượng.
- Che dấu các thuộc tính dữ liệu : nếu cần cho phép bên ngoài truy xuất 1 thuộc tính, ta tạo 2 tác vụ get/set tương ứng để giám sát và kiểm soát việc truy xuất.
- Che dấu chi tiết hiện thực các tác vụ.

Từ khóa truy cập

- Java cung cấp các từ khóa truy cập private, protected và public để xác định tầm vực truy xuất từng thành phần của class, các từ khóa trên ta gọi là các từ khóa truy cập (visibility modifier).
- Một modifier là một từ khóa Java, xác định
- tính đặc thù của một phương thức hay dữ liệu

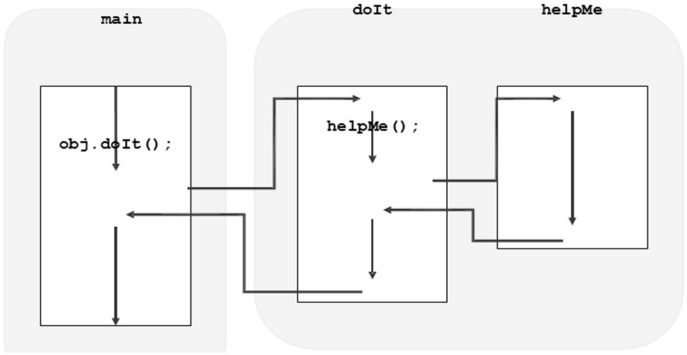
Từ khóa truy cập

- Các thành viên của một lớp được khai báo là public, nghĩa là có thể được tham chiếu bất cứ nơi nào.
- Các thành viên của một lớp đượ c khai báo là private, nghĩa là có thể được tham chiếu chỉ trong cùng lớp.
- Các thành viên của một lớp không được khai báo visibility modifier nào thì mặc đị nh là có thể được tham chiếu trong bất kỳ class nào trong cùng package.

Phương thức

- Một khai báo phương thức là xác định code mà sẽ được thực thi khi phương thứ được gọi.
- Khi một phương thức được gọi, luồng điều khiển sẽ nhảy đến phương thức và thực thi code.
- Khi hoàn thành, luồng điều khiển trả về nơi mà phương thức được gọi và làm tiếp tục

Luồng điều khiển phương thức



Phương thức

- Định nghĩa một phương thức:
- Modifier returnType methodName(parameterList)

```
{
    // Java statements
    return returnValue;
}
```

Overloading method
(Nạp chồng phương thức)

- Nạp chồng phương thức (Overloading method):
  - Các phương thức trong cùng class có cùng tên.
  - Danh sách tham số phải khác nhau: bao gồm số tham số, kiểu dữ liệu và thứ tự các tham số.
  - Trình biên dịch so sánh danh sách thông số thực để quyết định gọi phương thức nào.
  - Kiểu trả về của phương thức không được tính vào dấu hiệu của overloading method.

Overloading method
(Nạp chồng phương thức)

Version 1

```
float tryMe (int x)
{
   return x + .375;
}
```

Version 2

```
float tryMe (int x, float y)
{
   return x*y;
}
```

Invocation

```
result = tryMe (25, 4.32)
```


Phương thức toString

- Tất cả các class miêu tả cho đối tượng nên định nghĩa một phương thức toString.
- Phương thức toString trả về string dùng để đặc tả cho đối tượng theo một cách nào đó.

Constructors
(Phương thức khởi tạo)

- Một constructor là một phương thức đặt biệt, dùng để khởi tạo một đối tượng
- Một constructor có tên trùng với tên class.
- Một constructor không có kiểu trả về.
- Mỗi khi đối tượng được tạo ra bởi toán tử new, hệ thống sẽ tự động gọi phương thức khởi tạo.
  - Nếu không có constructor nào, hệ thống sẽ gọi constructor mặc định.

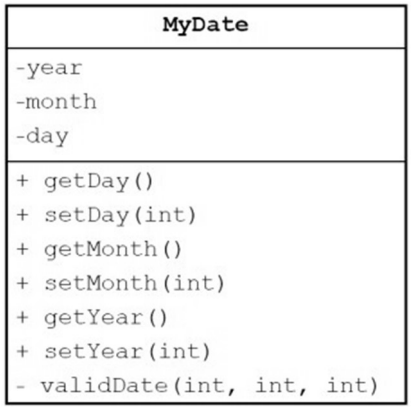
Constructors
(Phương thức khởi tạo)

- Constructor mặc định là constructor không có tham số.
- Các constructor còn lại gọi là copy constructor

Tham chiếu this

- Java cung cấp tham chiếu this để trỏ tới chính đối tượng đang hoạt động.
- this được sử dụng vào các mục đích như:
  - Tham chiếu tường minh đến thuộc tính và phương thức của đối tượng.
  - Truyền tham số và trả lại giá trị.
  - Dùng để gọi constructor.
- Khi tham số trùng tên thuộc tính thì nhờ từ khóa this chúng ta phân biệt rõ thuộc tính với tham số.

Mô hình hóa đối tượng

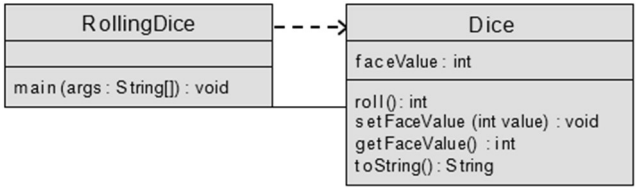


Mô hình Uml(Unified Modeling Language)

- Mô hình UML dùng để biểu diễn mối quan hệ giữa các lớp (class) và các đối tượng (object).
- UML class diagram chứa một hoặc nhiều lớp. Một lớp có tên lớp (class name), thuộc tính (attributes, data), phương thức (operations, methods).
- Các đường nối giữa các lớp gọi là sự kết hợp (associations)
- Mũi tên nét đứt thể hiện một lớp này thì dùng lớp khác.

Mô hình UMl

- Một UML class diagram

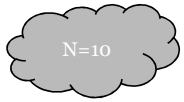


Passing objects to methods

- Trong Java, tham số được truyền bằng giá trị
  - Một bản sao của thông số thực (giá trị của thông số thực) được chép vào thông số hình thức (trong tiêu đề của phương thức)
- Khi tham số kiểu object, thực ra địa chỉ của
- object được truyền.
  - Thông số thực và thông số hì nh thứ c tr ở nên bí danh (aliases) của nhau
    - Nếu chúng ta thay đổi tra ng thái của object thông qua formal parameter reference bên trong phương thức, chính là thay đổi object được tham khảo bởi tham số thực.
    - Lưu ý sự khác nhau giữa thay đổi tham khảo (con trỏ) và thay đổi object mà tham khảo trỏ đến.

Example

```
class myClass {
public static void main( String [] agrs)
{
        int num = 10;
        System.out.println("Before  call setNum : " + num);
        setNum(num);
        System.out.println("After  call setNum" + num);
}
public static void setNum(int n) {
        n = 5;
}
```



Example: class HinhChuNhat

```
Class HinhChuNhat
{ private double dai, rong;
        public HinhChuNhat(double dai, double rong) {
        this.dai=dai;
        This.rong=rong;
        }
        public static void setDai (double dai) {
        this.dai=dai;
        }
        public double getDai() {
        return dai;
        }
}
```

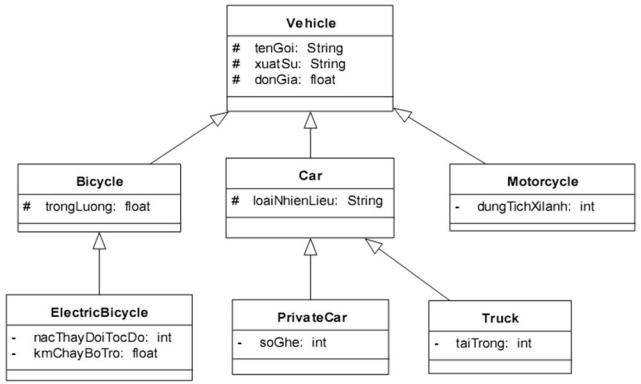
Example: testHCN

```
Class testHCN{
public static  void main( String [] agrs)
{
        HinhChuNhat   n = new HinhChuNhat(20,15);
        System.out.println   ("Before  call changeDai  : " + n.getDai());
        changeDai(n);
        System.out.println   ("After  call  changeDai:  " + n.getDai());
}
public static  void changeDai(  HinhChuN hat   x)
{
        x.setDai(30);
}
```


Tính thừa kế là gì? (Inheritance)

- Tính thừa kế là việc sử dụng lại các đặc tính của lớp cơ sở trong các lớp dẫn xuất. Với tính thừa kế, để xây dựng lớp mới, chỉ cần thêm các đặc tính riêng vào lớp dẫn xuất.
- Thừa kê trong lập trình hướng đối tượng cho phép ta có thể tạo ra 1 lớp mới có đầy đủ tất cả những thuộc tính và phương thức của 1 lớp đã có mà không phải viết lại mã lệnh.

Specialization to the Subclass



super

- Constructors không được thừa kế.
- Super trong constructors.
  - Sử dụng từ khóa super để xác định constructor nào trong lớp cơ sở sẽ được triệu gọi.
- Ngoài ra, từ khóa super còn được dùng để truy
- cập vào các members của lớp cơ sở.

Từ khóa final

- **final variables**: Hằng số
  - Hằng số là giá trị được gán cho biến tại thời điểm khai báo và sẽ không thay đổi về sau.
- **final class:** Là lớp không có lớp con
  - Là lớp không thể có lớp dẫn xuất từ nó
- **final method:** Là phương thức không được phép cài đè (override).
- Mọi phương thức của một lớp final là thì ngầm định là final

Lớp trừu tượng (abstract class)

- Chúng ta có thể tạo ra các lớp cơ sở để tái sử dụng mà không muốn tạo ra đối tượng thực của lớp.
  - Các lớp Point, Circle, Rectangle chung nhau khái niệm cùng là hình vẽ Shape
- → Giải pháp là khai báo lớp trừu tượng (abstract class)

Lớp trừu tượng (abstract class)

- Lớp trừu tượng dùng để tạo ra khung làm việc chung.
- Không thể tạo đối tượng từ lớp trừu tượng.
- Lớp trừu tượng định nghĩa các đặc tính chung cho các lớp con của nó.

Phương thức trừu tượng (abstract method)

- Một phương thức trừu tượng là một phương thức với các từ khóa abstract, và nó kết thúc bằng một dấu chấm phẩy.
  - Ví dụ: abstract double tienSauThue();
- Một lớp là trừu tượng nếu lớp có chứa một phương thức trừu tượng.
- Các lớp con của một lớp cha trừu tượng phải cài đặt tất cả các phương thức trừu tượng. Nếu không nó cũng sẽ trở thành lớp trừu tượng

Tính đa hình - Polymorphism

- Đa hình thái, nhiều cách phản ứng khác nhau cho cùng một hành vi
- Kỹ thuật cho phép nhiều phương thức khác nhau có cùng tên.
- Có hai cách thực hiện đa hình:

02. Graphic User Interface in Java

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Introduce
Graphic User Interface

- ✓ Giới thiệu AWT và Swing
- ✓ Xây dựng Java GUI cơ bản
- ✓ Cơ chế kiểm soát sự kiện người dùng

JFC (Java Foundation Classes)

- Gồm 5 phần chính:
  - AWT (Abstract Windows Toolkit): là thành phần công cụ thiết kế và lập trình giao diện cơ bản nhất trong Java
  - Swing
  - Accessibility API: Là bộ công cụ giúp người dùng kết nối với các thiết bị như bàn phím nổi, bộ đọc chữ tự động…cho phép truy xuất trực tiếp tới các thành phần Swing.
  - 2D API: chứa các lớp hiện thực nhiều kiểu vẽ, các hình phức tạp, fonts, colors. 2D API không phải là 1 phần của Swing
  - Drag and Drop: cho phép người dùng chọn giữ một đối tượngGUI rồi di chuyển qua các của sổ hoặc frame khác

Giới thiệu về AWT

- AWT (Abstract Window Toolkit) (java.awt.*) cung cấp một tập hợp các lớp dùng để viết giao diện người dùng dạng đồ họa.
- Bộ khung (framework) GUI cũ cho Java (Java 1.1)

Giới thiệu về AWT

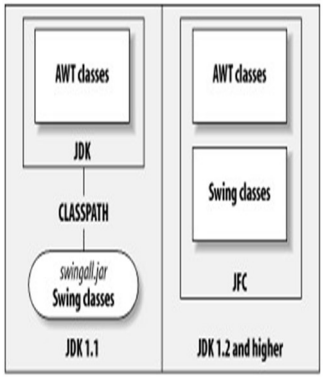
- Đặc điểm:
  - Bao gồm tập hợp các lớp ngang hàng, tức là giao diện lập trình ứng dụng cho các tính năng cửa sổ hiện có được cung cấp bởi hệ điều hành.
  - AWT cung cấp hai mô hình xử lý biến cố:
  - Mô hình thừa hưởng (mô hình phân cấp)
    - *Mô hình ủy quyền*
    - *AWT cung cấp các lớp chứa (container) và các thành phần*
  - (component) để đơn giản hóa việc xây dựng các chương trình.
  - AWT quản lý bố cục theo các sơ đồ tổ chức khác nhau

Giới thiệu về AWT

- Hạn chế:
  - Chiếm nhiều tài nguyên hệ thống (heavyweight object)
  - Khó mở rộng (không có các công nghệ hỗ trợ)
  - Một số dựa vào các bản sao mã bản ngữ (native code)
  - Gặp các vấn đề độc lập hệ nền
    - *Phụ thuộc vào các thành phần GUI của hệ điều hành*

Giới thiệu về SWING

- Swing (javax.swing.*)
- Bộ khung GUI mới được giới thiệu đầu tiên trong java 1.2
- Bao gồm tất cả những đặc tính của AWT cộng với nhiều đặc tính tiên tiến khác.
- Thuần Java, các thành phần nhẹ (lightweight) (không dựa vào mã bản ngữ)
- Kiến trúc cảm quan (Look and feel)



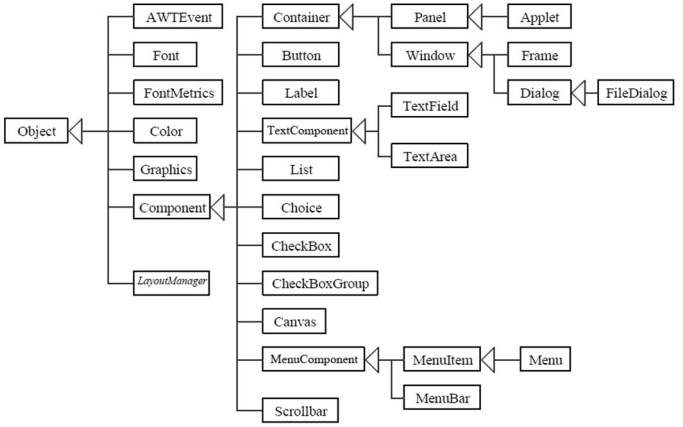
Giới thiệu về SWING

- Các ưu điểm của Swing:
  - Các thành phần của Swing chiếm ít tài nguyên hệ thống hơn vì chúng không ngang hàng riêng trong hệ điều hành.
  - Hỗ trợ khái niệm "pluggable look-and-feel", cung cấp thêm nhiều diện mạo để người dùng lựa chọn
  - Hỗ trợ các công nghệ nhập xuất mới: tiếng nói và thao tác không mouse
  - Dễ dàng mở rộng:
    - Button hỗ trợ cả văn bản và đồ họa
    - Sử dụng HTML trong Label
    - …

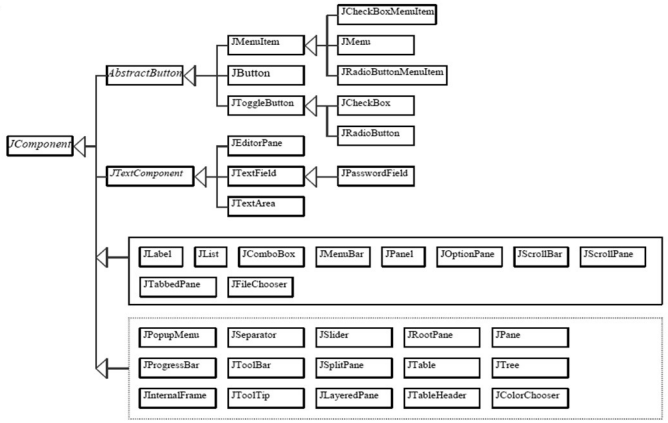
Giới thiệu Java GUI

- AWT và Swing cung cấp tập hợp các lớp Java cho phép tạo các giao diện đồ họa (GUI)
- Cung cấp các thành phần để tạo hoạt động và hiệu ứng GUI như
  - Container (bộ chứa)
  - Component (thành phần GUI)
  - Layout manager (bộ quản lý bộ cục)
  - Graphic và drawing capabilitie (vẽ đồ họa)
  - Font
  - Event

GUI Class Hierarchy (AWT)



GUI Class hierarchy (swing)



Ví dụ: Tạo cửa sổ với Swing

- Ứng dụng HelloWorld cơ bản
- Tạo một Cửa sổ với “HelloWorldString” trong thanh tiêu đề và hiển thị label “Hello World”

*HelloWorldSwing.java

```
import javax.swing.*;
public class HelloWorldSwing extends JFrame
{
    private  void createGUI() {
        JLabel label = new JLabel("Hello World");
        this.getContentPane().add(label);
    }
    public void ShowGUI()
    {
        createGUI();
        this.setTitle("HelloWorldSwing");//tieu de
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.pack();
        this.setSize(400,200);
        this.setLocationRelativeTo(null);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        HelloWorldSwing f=new HelloWorldSwing();
        f.ShowGUI();
    }
}
```

Cơ bản về thiết kế GUI

- Khái niệm xây dựng GUI rất đơn giản. Những thành phần (component) được bố trí trong một bộ chứa (container) theo cách thức có tổ chức nào đó.
- Những component có thể là các đối tượng (như Button, Menu, Label, Textbox, Slider, Checkbox, Radio button,…) hoặc có thể các bộ chứa lồng nhau,…
- Những thành phần được tổ chức trong những bộ chứa sử dụng bộ quản lý bố cục (Layout Manager)

Component

- Là các đối tượng có biểu diễn đồ họa được hiển thị lên màn hình mà người dùng tương tác được
- Ví dụ: nút nhấn, checkbox, scrollbar

Container

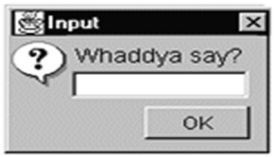
- Panel
  - Đối tượng khung chứa đơn giản nhất dùng để nhóm các đối tượng, thành phần con lại với nhau
  - Một Panel có thể chứa 1 Panel khác
- ScrollPanes
  - Tương tự Panel nhưng có thêm 2 thanh trượt giúp ta tổ chức và xem các đối tượng lớn
- Dialogs
  - Là một cửa sổ dạng hộp thoại dung để đưa ra các thông báo, lấy dữ liệu nhập từ người dùng.

Container

- Frame, Jframe
  - Là một cửa sổ Windows ở mức trên cùng gồm tiêu đề và đường biên như các ứng dụng Windows thông thường khác
  - Thường được dùng để tạo ra cửa sổ chính cho các ứng dụng khác.
- Applet: Web Applet
- JWindow: Không có thanh tiêu đề hay các nút điều khiển.

Container

- Top-level component: là thành phần trên cùng của bất kì Swing containment hierarchy nào



- Dialog Frame Applet

Container

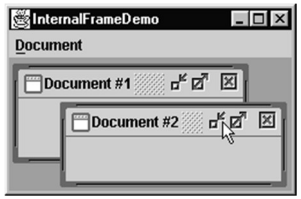
- Intermediate containers: là thành phần đơn thuần dùng để chứa các component khác

Panel

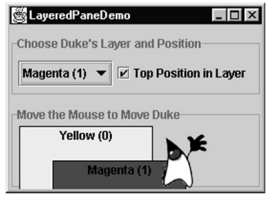
Scroll pane

Container

- Special-Purpose Containers là các thành phần chứa trung gian đặc biệt



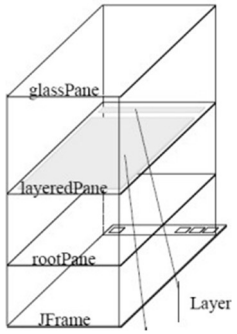
Internal Frame



Layered pane

Cấu trúc JFrame

- Khung chứa đa tầng
- Hầu hết mọi thứ đặt vào trong khung nội dung (content panel)
  - getContentPane()
- Sử dụng glassPane cho Popup menus, một số hoạt cảnh,…
- Các phương thức
  - getRootPane(); getLayeredPane()
  - getContentPane(); getGlassPane()



LayeredPane chứa contentPane

CÁC THÀNH PHẦN ĐIỀU KHIỂN CƠ BẢN

- Dùng để nhận dữ liệu từ người dùng

- **Buttons** **Combo Box** **List**

CÁC THÀNH PHẦN ĐIỀU KHIỂN CƠ BẢN

Menu Text fields Slide

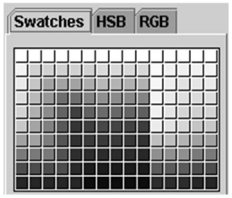
CÁC THÀNH PHẦN THUẦN HIỂN THỊ THÔNG TIN

- Dùng để hiển thị thông tin cho người sử dụng
- Không thể sửa đổi nội dung thông tin

Label Tool tip Progress Bar

CÁC THÀNH PHẦN SỬA CHỮA ĐỊNH DẠNG

- Dùng để hiển thị các thông tin định dạng
- Cho phép người dùng lựa chọn định dạng

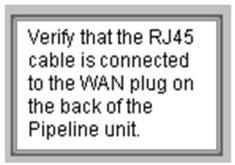


Color Chooser

File Chooser

CÁC THÀNH PHẦN HIỂN THỊ THÔNG TIN ĐÃ ĐỊNH DẠNG

Table



Text

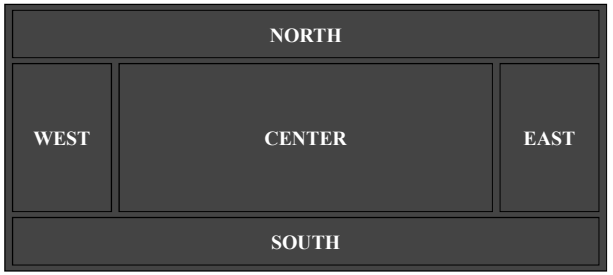
Tree

QUẢN LÝ BỐ CỤC (LAYOUT MANAGER)

- Dùng để xác định kích thước và vị trí của các thành phần GUI.
- Mỗi thành phần sẽ có 1 Layout manager mặc định.
- Các Layout manager Java hỗ trợ:
  - BorderLayout
  - BoxLayout
  - CardLayou
  - FlowLayout
  - GridBagLayout
  - GridLayout
  - OverlayLayout

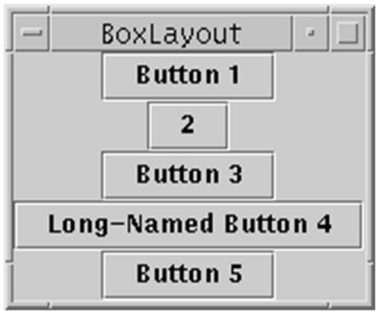
BORDERLAYOUT

- Chia thành 5 phần: NORTH, WEST,CENTER, EAST, SOUTH.



BOXLAYOUT

- Đưa các thành phần vào thành từng dòng hoặc từng cột.



CARDLAYOUT

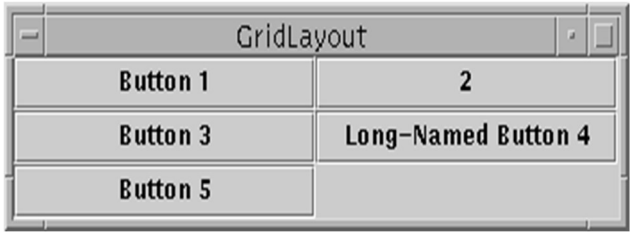
- Cho phép hiện thị nhiều component khác nhau tại nhiều thời điểm khác nhau.

FLOWLAYOUT

- Thêm các đối tượng tuần tự từ trái sang phải.

GRIDLAYOUT

- Thêm các đối tượng tuần từ trái sang phải, từ rên xuống dưới vào các ô đã định sẵn.



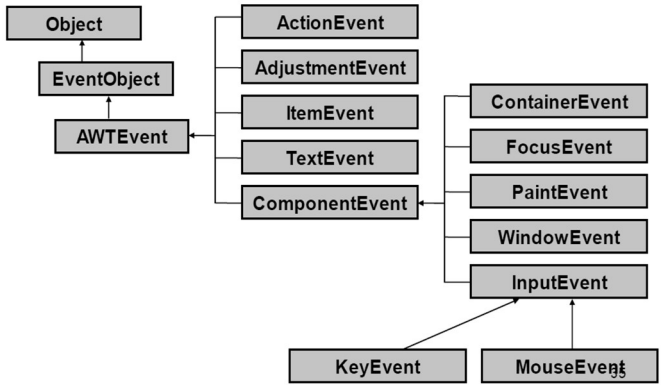
GRIDBAGLAYOUT

- Thêm các đối tượng vào các ô lưới đã định sẵn, nhưng cho phép người dùng mở rộng chỗ chứa cho các component (không chỉ 1 ô).

BỘ XỬ LÝ SỰ KIỆN (EVENT HANDLER)

- GUI là hệ thống hướng sự kiện (event-driven)
  - Chuột nhấn và chuyển động, nút nhấn và văn bản nhập thông qua bàn phím, nhấn vào các mục menu,…
  - Thao tác mong muốn sinh ra một hành động trên mỗi đối tượng này.
- Gói java.awt.event.*, java.swing.event.*

Gói java.awt.event.*



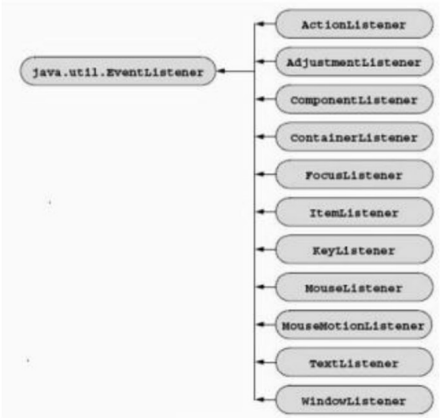
CÁC ĐỐI TƯỢNG TRONG XỬ LÝ SỰ KIỆN

- Nguồn sự kiện:
  - Các lớp thành phần GUI mà người sử dụng tương tác.
  - Bạn có thể đăng ký "Listener" đáp ứng với những sự kiện nhất định.
- Bộ lắng nghe (Listener):
  - Nhận đối tượng sự kiện khi được thông báo và thực hiện đáp ứng thích hợp.
  - Nhiều kiểu của bộ lắng nghe tồn tại cho các sự kiện cụ thể như MouseListener, ActionListener, KeyListener,…
  - Các giao tiếp được hiện thực và cài đặt các hành động.
- Đối tượng sự kiện (Event)
  - Đóng gói thông tin về sự kiện xuất hiện
  - Các đối tượng sự kiện được gửi tới bộ lắng nghe khi sự kiện xuất hiện trên thành phần GUI

MÔ HÌNH XỬ LÝ SỰ KIỆN

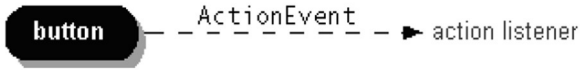
- Lớp hiện thực giao tiếp bộ lắng nghe sự kiện (bộ xử lý sự kiện).
  - Ví dụ:
  - *class Circle extends JFrame i mplements ActionListener () {...}*
- Đăng ký bộ lắng nghe sự kiện cho nguồn sự kiện
  - Ví dụ: *btCancel.addActionListener(handler);*
- Cài đặt phương thức xử lý sự kiện (các phương thức của giao tiếp bộ lắng nghe sự kiện)
  - Ví dụ: với bộ lắng nghe sự kiện ActionListener cần cài đặt phương thức
  - *public void actionPerformed(ActionEvent ev) { ...}*

MỘT SỐ BỘ LẮNG NGHE SỰ KIỆN



VÍ DỤ

- Một ActionListener được hiện thực và đối tượng lắng nghe được đăng ký với một thành phần Jbutton.
- Khi nút được nhấn, một sự kiện tự động được phát sinh và *phương thức tương ứng cài đặt trong ActionListener được gọi* (actionPerformed)



VÍ DỤ:

```
import javax.swing.*;
import javax.awt.event.*;

public class MyGUIEvent extends JFrame
                        implements ActionListener
{
   JButton btOk;
   public MyGUIEvent(){
      ....
      btOk.addActionListener(this);
      ....
   }
   public void actionPerformed(ActionEvent e){
      // Todo
   }
}
```

Ví dụ

```
import javax.swing.*;
import javax.awt.event.*;
public class MyGUIEvent extends JFrame
{
   JButton btOk;
   public MyGUIEvent(){
      ....
      btOk.addActionListener( new ActionListener() {
         public void actionPerformed(ActionEvent e){
            // Todo
         }
      });
      ....
   }
}
```


Determining event sources

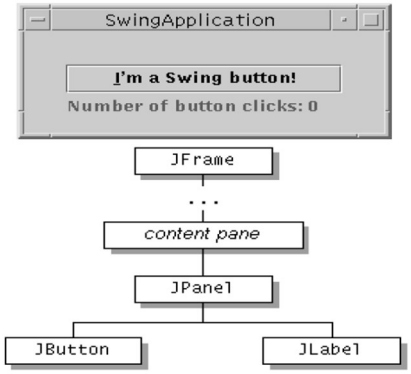
- One `listener` object can "listen" to many different `components`
  - The source of the event can be determined by using the `getSource` method of the `event` passed to the `listener`, it
  `component` that ge

```
Object source = e.getSource();
if (source.equals(component1) )
      // do something
else if (source.equals(component2))
      // do something
...
```


Painting

- Các component có thể cần hiển thị lại mình trên giao diện vì các lý do sau:
  - Thiết lập các trạng thái khác với mặc định
  - Phản ứng với các tương tác khác
- Các repaint: bắt đầu repaint với thành phần cao nhất cần repaint đi xuống cho tới hết cây phân cấp thành phần.
- Các thành phần thường sẽ repaint mỗi khi cần thiết
  - VD: khi gọi setText()

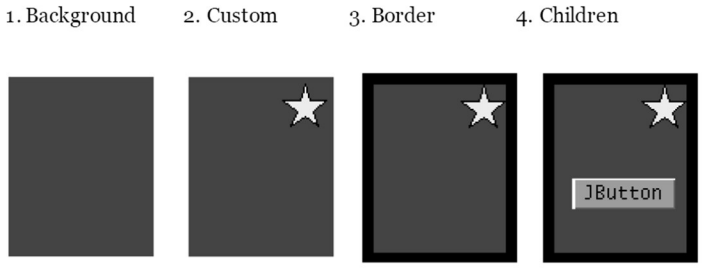
Ví dụ



VÍ DỤ

- JFrame sẽ repaint đầu tiên
- Content pane sẽ repaint background của nó rồi báo cho JPanel vẽ lại
- JPanel repaint background của nó sau đó vẽ lại đường biên → báo cho các thành phần con vẽ lại
- JButton sẽ vẽ lại background của nó rồi sửa lại đoạn text mà nó chứa
- JLabel sẽ repaint đoạn text của nó.

TUẦN TỰ TRIỆU GỌI CỦA REPAINT

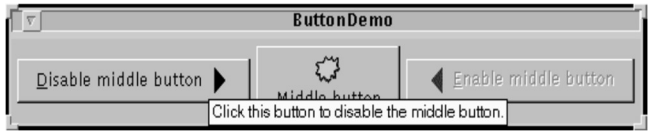


MỘT SỐ TÍNH NĂNG ĐẶC BIỆT

- Trừ top-level containers, các thành phần bắt đầu bằng chữ ***J*** đều có 1 số tính năng đặc biệt sau:
  - *Tooltips*
  - *Border*
  - *Look and fee*l

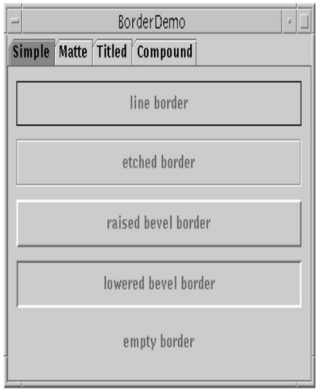
JCOMPONENT: TOOL TIPS

- Bằng cách dùng phương thức *setToolTipText*, bạn có thể cung cấp các gợi ý trợ giúp cho người sử dụng
- Khi con trỏ lướt qua vùng của component → t
- ooltip sẽ hiển thị



JComponent: border

- Phương thức *setBorder* cho phép người sử dụng chỉ định đường biên xung quanh component
- Bạn có thể sử dụng lớp *BorderFactory* để tạo ra 1 số border thường gặp




```
JPanel pane = new JPanel();
pane.setBorder(BorderFactory.createLineBorder(Color.black));
```


JComponent: look and feel

- Việc hiển thị của các component phụ thuộc vào ComponentUI bên dưới
- Bạn có thể dùng phương thức *UIManager.setLookAndFeel* để thay đổi cách hiển thị của các thành phần.

```
public static void main(String[] args) {
try {
UIManager.setLookAndFeel(
  UIManager.getCrossPlatformLookAndFeelClassName());
}
catch (Exception e) {}
new SwingApplication(); //Create and show the GUI.
}
```


JComponent: look and feel

- Bạn cũng có thể dùng các "look and feel" của các hệ nền khác sử dụng cú pháp sau

UIManager.setLookAndFeel(
"com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

- Nếu bạn thiết lập "look and feel" trước khi bất cứ thành phần UI nào được tạo ra thì chương trình sẽ cố gắng thiết lập "look and feel" theo thông số bạn truyền vào
  - Nếu không được sẽ lấy giá trị mặc định trong file swing.properties

JComponent: look and feel

- Nếu bạn thiết lập lại "look and feel" sau khi đã có thành phần UI tạo ra thì bạn sẽ làm như sau để cập nhật "look and feel" cho các thành phần nay:

```
UIManager.setLookAndFeel(lnfName);
SwingUtilities.updateComponentTreeUI(frame);
```

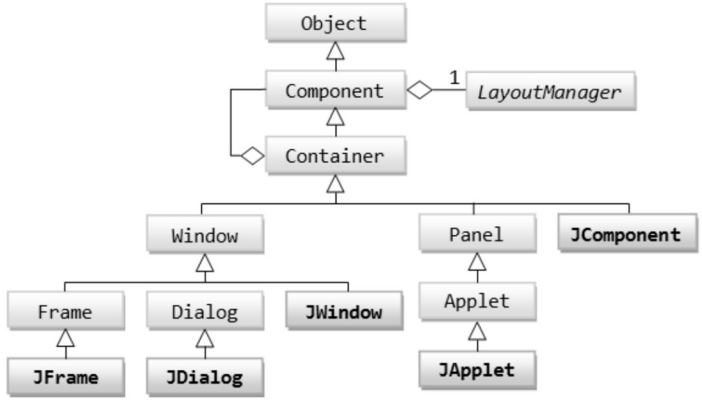
03. Graphic User Interface in Java

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

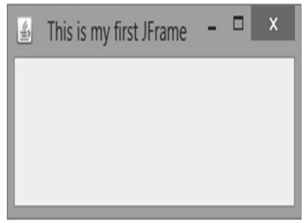
GUI components (p2)

- ✓ Top-level container
- ✓ Layout Manager
- ✓ Common Control
- ✓ Event Listener
- ✓ Dialogbox
- ✓ Advanced Control

Top-level container

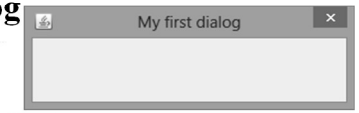


Top-level container: JFrame



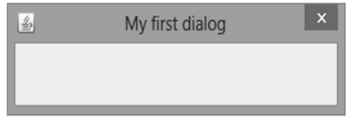

```
3  import javax.swing.JFrame;
4
5  public class DemoJFrame extends JFrame{
6      public DemoJFrame() {
7          setTitle("This is my first JFrame");
8          setSize(300, 200);//set the size of jframe
9          setDefaultCloseOperation(EXIT_ON_CLOSE);//exit program when click exit button
10         setLocationRelativeTo(null);//center off screen
11         setResizable(false);//do not allow resized
12         //other properties
13     }
14
15     public static void main(String[] args) throws Exception {
16         new DemoJFrame().setVisible(true);
17     }
18 }
```

Top-level container: JDialog




```
2
3  import javax.swing.JDialog;
4
5  public class DemoJDialog extends JDialog{
6      public DemoJDialog() {
7          setTitle("My first dialog");//set the title
8          //dispose the dialog when click exit button
9          setDefaultCloseOperation(DISPOSE_ON_CLOSE);
10         setSize(300, 200);
11         setModal(true);//dialog is a modal dialog
12         this.setAlwaysOnTop(true);//always on top
13         //...
14     }
15
16     public static void main(String[] args) throws Exception {
17         new DemoJDialog().setVisible(true);
18     }
19 }
```

Top-level container: Window




```
3  import javax.swing.JWindow;
4
5  public class DemoJWindow extends JWindow{
6      public DemoJWindow() {
7          setSize(300, 200);
8          this.setAlwaysOnTop(true);//always on top
9          //...
10     }
11
12     public static void main(String[] args) throws Exception {
13         new DemoJDialog().setVisible(true);
14     }
15 }
```

6

GUI Components - JPanel

- A `JPanel` is used to group other components into rectangular regions
- Properties:
  - A panel can be nested within another panel
  - The default layout manager is `FlowLayout`
- Usage:
  - Allocate JPanel, put other components in it, add to other container
- Constructors:
  - `JPanel()`
  - `JPanel(LayoutManager lm)`

7

GUI Components - JPanel


```
3  import javax.swing.BorderFactory;
4  import javax.swing.JFrame;
5  import javax.swing.JPanel;
6
7  public class DemoJPanel extends JFrame{
8      public DemoJPanel() {
9          setTitle("Demo JPanel");
10         setSize(400, 250);
11         setDefaultCloseOperation(EXIT_ON_CLOSE);
12
13         JPanel p=new JPanel();//create Jpanel
14         //set border of Jpanle
15         p.setBorder(BorderFactory.createTitledBorder("This is my panel"));
16
17         this.add(p);//put panel to JFrame
18     }
19
20     public static void main(String[] args) throws Exception {
21         new DemoJPanel().setVisible(true);
22     }
23 }
```

8

GUI Components - JLabel

- Labels are usually used to display information or identify other components in the interface
- A label can be composed of text, and image, or both at the same time
- The `ImageIcon` class is used to represent an image that is stored in a label
- The alignment of the text and image within the label can be set explicitly

GUI Components - JLabel

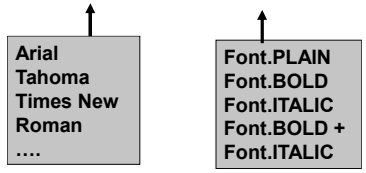
- Constructors:
  - JLabel()
    - Creates an empty label
  - JLabel (String labeltext)
    - Creates a label with a given text
  - JLabel (String labeltext, int alignment)
    - Creates a label with given alignment where alignment can be LEFT, RIGHT, CENTER, LEADING or TRAILING.
  - JLabel (Icon img)
    - Only Icon will be used for label
  - JLabel (String str, Icon img, int align)

GUI Components - JLabel

- Some methods:
  - String getText()
  - void setText(String text)
    - Gets or sets the text displayed on the label
  - Font getFont()
  - void setFont(Font font)
    - Gets or sets the current font of the label

Using special fonts for text

- To draw characters in a font, you must first create an object of the class Font
- Constructor:
  - **Font(String font_name, int font_style, int font_size)**

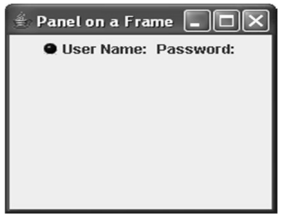


- Example:

```
Font  fo = new Font ("Times New Roman", Font.BOLD, 14);
```

JLabel Demo

```
public  class JLabelDemo extends JFrame  {
    public JLabelDemo()  {
        super("Panel on a Frame");
        JPanel jp = new JPanel();
        jp.add( new JLabel("User Name: ", new ImageIcon("blue-ball.gif"),
                                          SwingConstants.CENTER);
        jp.add( new JLabel("Password: "));
        add(jp);
        setSize(400, 400);
    }
    public static void main(String args[]){
        new JLabelDemo().setVisible(true);}
}
```



GUI Components - JTextField

- Purpose
  - to input text
  - display information
- The width of **`JTextField:`** is measured by column
  - The column width that you set in the `JTextField` constructor is not an upper limit on the number of characters the user can enter

JTextField - Constructors

- **JTextField()**
  - creates an empty textfield with 1 columns
- **TextField(String s)**
  - creates a new textfield with the given string
- **JTextField(int cols)**
  - creates an empty textfield with given number of columns
- **JTextField(String text, int cols)**
  - creates a new textfield with given string and given number of columns

Example:

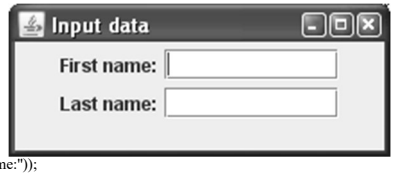
```
JTextField mmText = new JTextField(10);
JTextField txtName = new JTextField("Hello", 10);
```


JTextField - Methods

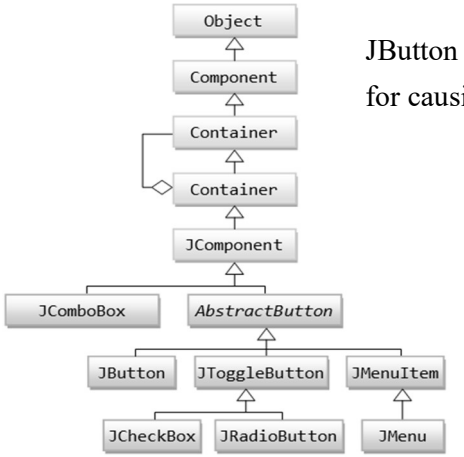
- **String getText()**
- **void setText(String t)**
  - gets or sets the text in text field
- **void setFont(Font font)**
  - sets the font for this text field
- **void setEditable(boolean b)**
  - determines whether the user can edit the content

JTextField Demo

```
public class JTextFieldDemo extends JFrame {
        JTextField textFN, textLN;
        public JTextFieldDemo()  {
                super("Input data");
                JPanel jp = new JPanel();
                jp.add(new JLabel("First name:"));
                jp.add ( textFN=new JTextField (10));
                jp.add(new JLabel("Last name:"));
                jp.add ( textLN=new JTextField (10));
                add(jp);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                setSize(250, 100);
        }
        public static void main(String[] args) { new JTextFieldDemo().setVisible(true); }
}
```



GUI Components - JButton



JButton is a clickable region for causing actions to occur

GUI Components - JButton

- Constructors:
  - JButton()
  - JButton(Icon icon)
  - JButton(String text)
  - JButton(String text, Icon icon)
    - Creates a button with the specified text or/and icon
- When the user click the button, JButton generates an Action event

JButton Demo

```
public class JButtonDemo2 extends JFrame  {
    JButton btn1, btn2;
    public JButtonDemo2()  {
            super("Button Test!");
            // Create the two buttons
            btn1 = new JButton ("First Button", new ImageIcon("red-ball.gif"));
            btn2 = new JButton ("Second button", new ImageIcon("blue-ball.gif"));
            JPanel  panel = new JPanel();
            // Add the two buttons to the panel
            panel.add(btn1);
            panel.add(btn2);
            add(panel);
            setSize(300,300);
    }
    public static void main(String[] args) { new JButtonDemo2().setVisible(true); }
}
```

03. Graphic User Interface in Java

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

GUI components (p2)

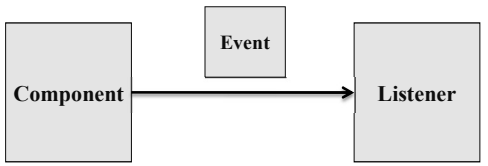
- ✓ Top-level container
- ✓ Layout Manager
- ✓ Common Control
- ✓ Event Listener
- ✓ Dialogbox
- ✓ Advanced Control

Event

- An `event` is an object that represents some activity to which we may want to respond, example:
  - the mouse is moved
  - a mouse button is clicked
  - a keyboard key is pressed
  - a timer expires,…
- Events often correspond to user actions, but not always
- Each type of event belongs to an Event class

Event handling

- `Components` generate `events`, and `listeners` handle the events when they occur
  - A *`listener`* object "waits" for an `event` to occur and responds accordingly



A component may generate an event

A listener responds to the event

When the event occurs, the appropriate method of the listener is called with an object that describes the event is passed

General process

- Determine what type of listener is of interest
  - ActionListener, ItemListener, KeyListener, MouseListener, WindowListener,…
    - If you have an Event class, you will have a listener
- Define a class of that type
  - Implement the interface
    - If you implement a listener, you must write all methods in that listener
- Register an object of your listener class with the component
  - component.**addXxxListener**(eventListenerObject);
  - E.g., addKeyListener, addMouseListener

Example: ActionListener interface

```
// ActionListener interface
public interface ActionListener {
        public void actionPerformed(ActionEvent event);
}

// ButtonHandlingDemo class
public class ButtonHandlingDemo implements ActionListener {
        public void actionPerformed(ActionEvent event){
                …
        }
}
```


Example: MouseListener interface

```
public interface MouseListener  {
        public void mouseClicked(MouseEvent e);
        public void mousePressed(MouseEvent e);
        public void mouseReleased(MouseEvent e);
        public void mouseEntered(MouseEvent e);
        public void mouseExited(MouseEvent e);
}
public class ButtonHandlingDemo implements MouseListener {
        public void mouseClicked(MouseEvent e) { …}
        public void mousePressed(MouseEvent e) {… }
        public void mouseReleased(MouseEvent e) { …}
        public void mouseEntered(MouseEvent e) {… }
        public void mouseExited(MouseEvent e) { …}
}
```


The ways to handle event

- Event-handling options
  - Handling events with separate listeners
  - Handling events by implementing interfaces in frame
  - Handling events with named inner classes
  - Handling events with anonymous inner classes

Handling events with separate listeners

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class HandlingButtonDemo11
                        extends JFrame  {
    JButton btn1, btn2;
    JPanel panel;
    public HandlingButtonDemo1 ()     {
       super("Button Test  with Handling!");
       panel = new JPanel();
       panel.add(btn1 = new JButton ("RED"));
       panel.add(btn2 = new JButton ("YELLOW"));
       add(panel);

       btn1.addActionListener(new HandlingButton1(panel));

       setSize(300,300);
}
    public static void main(String[] args) {new HandlingButtonDemo1().setVisible(true);}
}
```

```
class HandlingButton1 implements ActionListener
    {
    JPanel panel;
    public HandlingButton1(JPanel p)     {
        panel = p;
    }
    public void actionPerformed(ActionEvent e)
    {
        panel.setBackground(Color.red);
    }
}
```

How to do with YELLOW button?

Handling events with separate listeners

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class HandlingButtonDemo2  extends JFrame implements ActionListener  {
    JButton btn1, btn2;
    JPanel panel;
    public HandlingButtonDemo2()   {
        ….

        setSize(300,300);
    }
    public void actionPerformed(ActionEvent e)
    {
        panel.setBackground(Color.red);
    }
    public static void main(String[] args)   {
            new HandlingButtonDemo2().setVisible(true);
    }
}
```

How to do with YELLOW button?

Handling events with named inner classes

```
import ...
public class HandlingButtonDemo31extends JFrame  {
    JButton btn1, btn2;
    JPanel panel;
    public HandlingButtonDemo3 ()     {

        btn1.addActionListener(new HandlingButton());

        setSize(300,300);
```

How to do with YELLOW button?

```
    private class HandlingButton implements
        ActionListener {
        public void actionPerformed( ActionEvent e )  {
            panel.setBackground(Color.red);
        }
    }

    public static void main(String[] args) { new HandlingButtonDemo3().setVisible(true)
}
```


Handling events with anonymous inner classes

```
import …
public class HandlingButtonDemo4  extends JFrame  {
    JButton btn1, btn2; JPanel panel;
    public HandlingButtonDemo4()   {
        …
        btn1.addActionListener( new ActionListener() {
                                        public void actionPerformed(ActionEvent e)  {
                                                panel.setBackground(Color.red);
                                        }
                                } );
        btn2.addActionListener( new ActionListener() {
                                        public void actionPerformed(ActionEvent e)  {
                                                panel.setBackground(Color.yellow);
                                        }
                                } );
        setSize(300,300);
    }

    public static void main(String[] args)   {
            new HandlingButtonDemo4().setVisible(true);
    }
```

Determining event sources

- One `listener` object can "listen" to many different `components`
  - The source of the event can be determined by using the `getSource` method of the `event` passed to the `listener`, it returns a reference to the `component` that generated the `event`

```
Object source = e.getSource();
if (source.equals(component1) )
      // do something
else if (source.equals(component2))
      // do something
…
```


HandlingButtonDemo2.java

```
import …
public class HandlingButtonDemo2  extends JFrame implements ActionListener {
   JButton btn1, btn2;
   JPanel panel;
   public HandlingButtonDemo2 ()    {
    super("Button Test  with Handling!");
    panel = new JPanel();

    btn1 = new JButton("RED");
    btn2 = new JButton("YELLOW");
    btn1.addActionListener(this);
    btn2.addActionListener(this);

    panel.add(btn1);
    panel.add(btn2);

    add(panel);
    setSize(300,300);
   }

    public void actionPerformed(ActionEvent e)
    {
     Object source = e.getSource();
     if (source .equals(btn1) )
          panel.setBackground(Color.red);
     else if (source.equals(btn2))
          panel.setBackground(Color.yellow);
    }
    public static void main(String[] args)    {
      new HandlingButtonDemo2().
      setVisible(true);
    }
}
```

Events of top-level container (1)

```
 2 import javax.swing.JFrame;
 3
 4 public class DemoJFrame extends JFrame{
 5     public DemoJFrame() {
 6         setTitle("This is my first JFrame");
 7         setSize(300, 200);//set the size of jframe
 8
 9         setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
10
11         addWindowListener(new MyWindowListener());
12
13     }
14     public static void main(String[] args) throws Exception {
15         new DemoJFrame().setVisible(true);
16     }
17 }
```

Registe WindowListener

Events of top-level container (2)

```
 3 import java.awt.event.WindowEvent;
 4 import java.awt.event.WindowListener;
 5
 6 class MyWindowListener implements WindowListener{
 8     public void windowOpened(WindowEvent e) {
10     @Override
11     public void windowClosing(WindowEvent e) {
12         System.out.println("Do works before exit");
13         //without follow line of code, Jframe does not exist
14         System.exit(1);
15     }
17     public void windowClosed(WindowEvent e) {
20     public void windowIconified(WindowEvent e) {
23     public void windowDeiconified(WindowEvent e) {
26     public void windowActivated(WindowEvent e) {
29     public void windowDeactivated(WindowEvent e) {
31 }
```

Layout Manager

- ✓ **Flow Layout**
- ✓ **Border Layout**
- ✓ **Card Layout**
- ✓ Grid Layout
- ✓ **GridBag Layout**
- ✓ **Box Layout**
- ✓ **Overlay Layout**

Layout Managers

- ✓ Flow Layout
- ✓ Border Layout
- ✓ Card Layout
- ✓ Grid Layout
- ✓ GridBag Layout

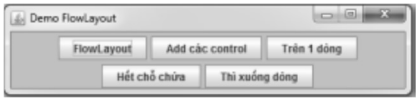
} Defined in the AWT

- ✓ Box Layout
- ✓ Overlay Layout

} Defined in Swing

FlowLayout

- Mặc định khi một JPanel được khởi tạo thì bản thân lớp chứa này sẽ có kiểu Layout là FlowLayout.



FlowLayout – Constructors

- **public FlowLayout()**
  - Centers each row and keeps 5 pixels between entries in a row and between rows
- **public FlowLayout(int align)**
  - Same 5 pixels spacing, but changes the alignment of the rows to *FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER*
- **public FlowLayout(int align, int hgap, int vgap)**
  - Specify the alignment as well as the horizontal and vertical spacing between components (in pixels)

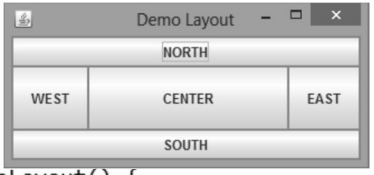
FLowLayout demo


```
3 import java.awt.FlowLayout;
4 import javax.swing.JButton;
5 import javax.swing.JFrame;
6
7 public class DemoLayout extends JFrame{
8     public DemoLayout() {
9         setTitle("Demo Layout");setSize(400, 250);
10        setDefaultCloseOperation(EXIT_ON_CLOSE);
11        setLayout(new FlowLayout(FlowLayout.CENTER,5,5));
12        for (int i = 0; i < 10; i++) {
13            this.add(new JButton("Button "+i));
14        }
15    }
16
17    public static void main(String[] args) throws Exception {
18        new DemoLayout().setVisible(true);
19    }
20 }
```


Border Layout

- Nếu như không có 4 vùng : North, West, South, East. Thì vùng Center sẽ tràn đầy cửa sổ.
- Thông thường khi đưa các control JTable, JTree, ListView, JScrollpane… ta thường đưa vào vùng Center để nó có thể tự co giãn theo kích thước cửa sổ giúp giao diện đẹp hơn.

Border Layout demo




```
public DemoLayout() {
    setTitle("Demo Layout");setSize(400, 250);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLayout(new BorderLayout());
    this.add(new JButton("NORTH"),BorderLayout.NORTH);
    this.add(new JButton("SOUTH"),BorderLayout.SOUTH);
    this.add(new JButton("WEST"),BorderLayout.WEST);
    this.add(new JButton("EAST"),BorderLayout.EAST);
    this.add(new JButton("CENTER"),BorderLayout.CENTER);
}
```

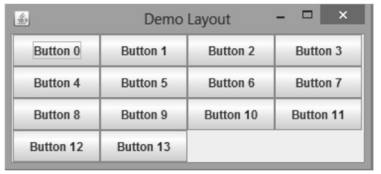

Grid Layout

- Hỗ trợ việc chia container thành một lưới
- Các thành phần được bố trí trong các dòng và cột
- Một ô lưới nên chứa ít nhất một thành phần
- Kiểu layout này được sử dụng khi tất cả các thành phần có cùng kích thước

GridLayout – Constructors

- **public GridLayout()**
  - Creates a single row with one column allocated per component
- **public GridLayout(int rows, int cols)**
  - Divides the window into the specified number of rows and columns
  - Either rows or cols (but not both) can be zero
- **public GridLayout(int rows, int cols, int hgap, int vgap)**
  - Uses the specified gaps between cells

Grid Layout demo




```
public DemoLayout() {
    setTitle("Demo Layout");setSize(400, 250);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLayout(new GridLayout(4,4));//rows x cols
    for (int i = 0; i < 14; i++) {
        this.add(new JButton("Button "+i));
    }
}
```


Box Layout

- BoxLayout cho phép add các control theo dòng hoặc cột, tại mỗi vị trí add nó chỉ chấp nhận 1 control, do đó muốn xuất hiện nhiều control tại một vị trí thì bạn nên add vị trí đó là 1 JPanel rồi sau đó add các control khác vào JPanel này.
  - **BoxLayout.X_AXIS** cho phép add các control theo hướng từ trái qua phải.
    - Box box = new Box(BoxLayout.X_AXIS);
    - Box box = Box.createHorizontalBox();
  - **BoxLayout.Y_AXIS** cho phép add các control theo hướng từ trên xuống dưới.
    - Box box = new Box(BoxLayout.Y_AXIS);
    - Box box = Box.createVerticalBox();
- **BoxLayout** sẽ không tự động xuống dòng khi hết chỗ chứa, tức là các control sẽ bị che khuất nếu như thiếu không gian chứa nó.

Box Layout

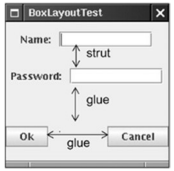
BoxLayout.X_AXIS

BoxLayout.Y_AXIS

No wrap row when resize dimension

Fillers in BoxLayout

- To space the components out, you add invisible fillers
  - Strut
    - Simply adds some space between components
  - Glue
    - Separates components as much as possible
  - Rigid area
    - Fixed-size rectangular space between two components



Strut

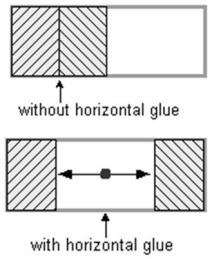
- Create a space with fixed width, fixed height
  - static Component createHorizontalStrut (int width)
  - static Component createVerticalStrut (int height)
- For example, add 10 pixels of space between two components in a horizontal box:

```
Box box = Box.createHorizontalBox();
box.add(label);
box.add(Box.createHorizontalStrut(10));
box.add(textField);
```


Glue

- To create an invisible component that can expand infinitely horizontally, vertically, or in both directions:
  - static Component createHorizontalGlue()
  - static Component createVerticalGlue()
  - static Component createGlue()
- For example:

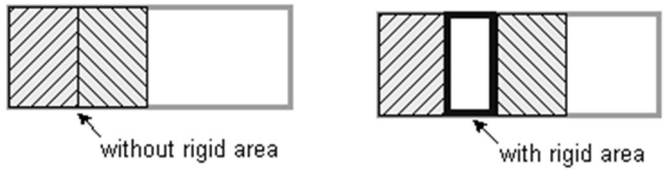
```
Box box = Box.createHorizontalBox();
box.add(firstComponent);
box.add(Box.createHorizontalGlue());
box.add(secondComponent);
```



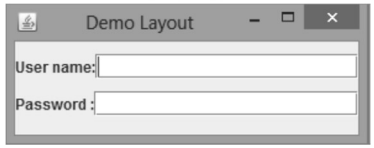
Rigid area

- Create a rigid area with fixed width and height:
  - static Component createRigidArea(Dimension d)
- For example, to put 5 pixels between two components in a vertical box

```
container.add(firstComponent);
container.add(Box.createRigidArea(new Dimension(5, 0)));
container.add(secondComponent);
```

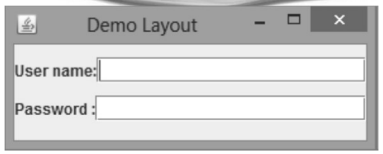


Box Layout demo




```
public DemoLayout() {
    setTitle("Demo Layout");setSize(400, 250);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JPanel p=new JPanel();JPanel p1=new JPanel();JPanel p2=new JPanel();
    p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
    p1.setLayout(new BoxLayout(p1, BoxLayout.X_AXIS));
    p2.setLayout(new BoxLayout(p2, BoxLayout.X_AXIS));

    p1.add(new JLabel("User name:"));p1.add(new JTextField(20));
    p2.add(new JLabel("Password :"));p2.add(new JPasswordField(20));
    p.add(Box.createRigidArea(new Dimension(10,10)));
    p.add(p1);
    p.add(Box.createRigidArea(new Dimension(10,10)));
    p.add(p2);
    this.add(p,BorderLayout.NORTH);
}
```


Box Layout demo




```
public DemoLayout() {
    setTitle("Demo Layout");setSize(400, 250);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    Box b=Box.createVerticalBox();//BoxLayout.Y_AXIS
    Box p1=Box.createHorizontalBox();//BoxLayout.X_AXIS
    Box p2=Box.createHorizontalBox();
    p1.add(new JLabel("User name:"));p1.add(new JTextField(20));
    p2.add(new JLabel("Password :"));p2.add(new JPasswordField(20));
    b.add(Box.createRigidArea(new Dimension(10,10)));
    b.add(p1);
    b.add(Box.createRigidArea(new Dimension(10,10)));
    b.add(p2);
    this.add(b,BorderLayout.NORTH);
}
```


Borders

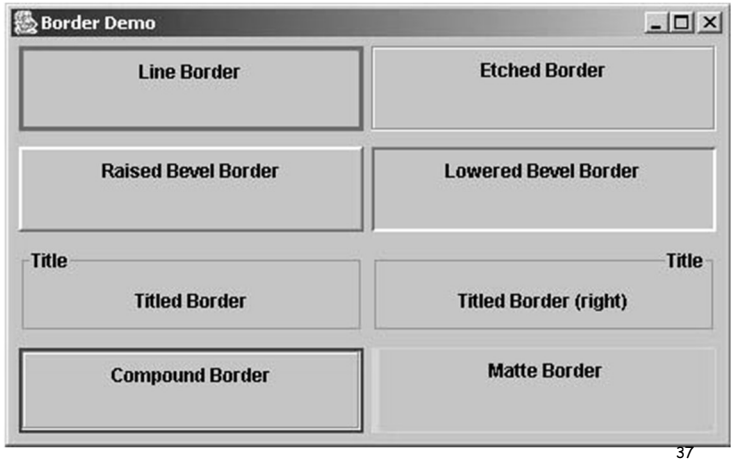
- Purpose
  - Create border to any component that extends JComponent
    - With JPanel, group components visually
- Usage
  - Create a Border object by calling BorderFactory.createXxxBorder(…)
  - To apply a border object to a component, using the component's `setBorder` method
- Example

  label.setBorder(BorderFactory.createLineBorder(Color.red));

Static methods in BorderFactory

- **createEmptyBorder**(int top, int left, int bottom, int right)
  - Simply adds space (margins) around the component
- **createLineBorder**(Color color)
- **createLineBorder**(Color color, int thickness)
  - Creates a solid-color border
- **createTitledBorder**(String title)
- **createTitledBorder**(Border border, String title)
  - The border is an etched line unless you explicitly provide a border style in second constructor
- **createEtchedBorder**()
- **createEtchedBorder**(Color highlight, Color shadow)
  - Creates a etched line without the label

Border Demo



37

38

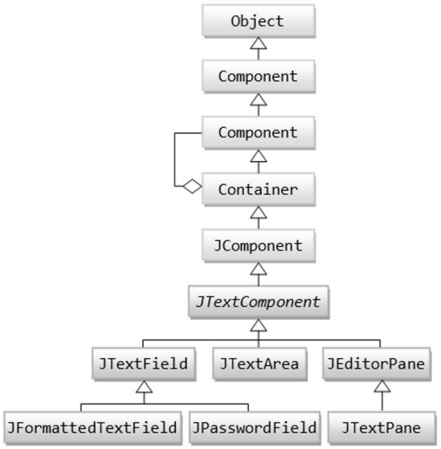
04. Graphic User Interface in Java

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

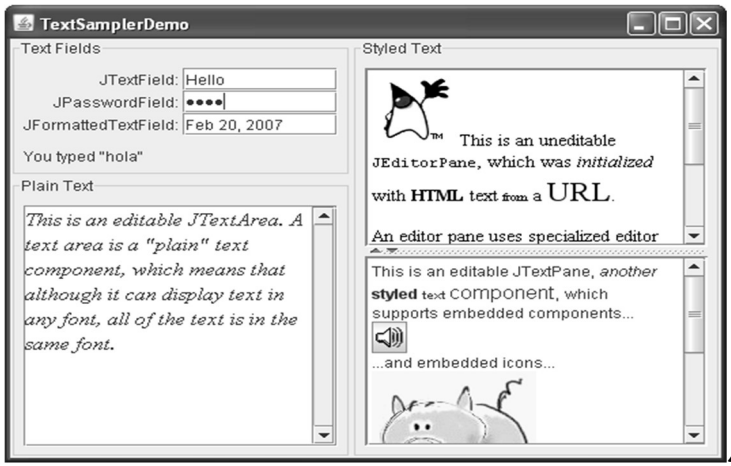
GUI components (p3)

- Text component
- Choice component
- Menu
- Mnemonic
- Toolbar
- Tooltip
- Tabbed pane
- Scroll pane
- Dialog box

Text component



Text component



Text component

Group	Description	Swing Classes
Text Controls	Also known simply as text fields, text controls can display only one line of editable text. Like buttons, they generate action events.	JTextField and its subclasses JPasswordField and JFormattedTextField
Plain Text Areas	JTextArea can display multiple lines of editable text. Although a text area can display text in any font, all of the text is in the same font.	JTextArea
Styled Text Areas	A styled text component can display editable text using more than one font. Some styled text components allow embedded images and even embedded components.	JEditorPane and its subclass JTextPane

JTextField

- If the cursor is in the text field, the user presses the *Enter* key, `JTextField` generates an Action event
  - Listener?
    - … implements ActionListener
  - Method in the listener?
    - public void actionPerformed(ActionEvent e)
  - Register listener to the text field?
    - void addActionListener(ActionListener listener)

JTextField Demo

```
public class JTextFieldDemo extends JFrame
            implements ActionListener
{
    JTextField mmText;
   JLabel resultLabel;
   public JTextFieldDemo()  {
      super("Chuyen doi don vi");
      setLayout(new GridLayout(2,2));

      add (new JLabel ("Nhap vao so millimet:"));
      add (mmText = new JTextField (10));
      add (new JLabel ("So centimet tuong ung:"));
      add (resultLabel = new JLabel ("---"));

     mmText.addActionListener (this);
     setDefaultCloseOperation(EXIT_ON_CLOSE);
     setSize(300,90);
}
```

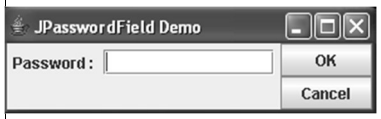

```
   public void actionPerformed(ActionEvent e)
   {
     double cm, mm;
     mm =
     Double.parseDouble(mmText.getText());
     cm = mm/10;
     resultLabel.setText
     (Double.toString(cm));
   }

   public static void main(String[] args)  {
     new JTextFieldDemo().setVisible(true);
   }
}
```


JPasswordField

- Purpose: used to enter a password
- A `JPasswordField` is similar to a JTextField except the characters typed in by the user are not echoed back to the user
  - Instead, an alternate character such as asterisks (*) is displayed
  - You can set the echo character using the method:
    - **`public void setEchoChar(char c)`**

JPasswordField Demo

```
public class JPasswordFieldDemo extends JFrame
            implements ActionListener
{
   JPasswordField txtPassword;
   JButton btnOk, btnCancel;

   public JPasswordFieldDemo()
   {
      super("JPasswordField Demo");
     JPanel pnlLeft, pnlRight;

      txtPassword = new JPasswordField(12);
      txtPassword.addActionListener(this);

      pnlLeft = new JPanel();
      pnlLeft.add(new JLabel("Password: "));
      pnlLeft.add(txtPassword);

      pnlRight = new JPanel(new GridLayout(0,1));
      pnlRight.add(btnOk = new JButton("OK"));
      pnlRight.add(btnCancel=new JButton("Cancel"));

      add(pnlLeft, BorderLayout.WEST);
      add(pnlRight, BorderLayout.CENTER);

      btnOk.addActionListener(this);
      btnCancel.addActionListener(this);

      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setsize(200, 200);
}
```



JPasswordField Demo (contd.)

```
public void actionPerformed(ActionEvent e)
{
     Object o = e.getSource();
     if (o == btnOk || o == txtPassword)      {
          char chPassword[] = txtPassword.getPassword();
          String strPassword = new String(chPassword);
          if(strPassword.trim().equals("pass")) {
               JOptionPane.showMessageDialog(this,"Correct Password");
               System.exit(0);
          }
          else {
               JOptionPane.showMessageDialog(this,"Incorrect Password",
                                   "Error Message", JOptionPane.ERROR_MESSAGE);
               txtPassword.selectAll();
               txtPassword.requestFocus();
              }
     }
     else  {
          System.exit(0);
     }
}
public static void main(String [] args){ new JPasswordFieldDemo().setVisible(true); }
}
```


JTextArea

- Purpose
  - For texts with more than one line long
- Constructors
  - **`JTextArea(int rows, int cols)`**
    - constructs a new text area with number of rows and columns
  - **`JTextArea(String text, int rows, int cols)`**
    - constructs a new text area with an initial text

JTextArea Demo

```
JPanel buttonPanel = new JPanel();
buttonPanel.add(insertButton=new JButton("Insert"));
buttonPanel.add(wrapButton = new JButton("Wrap"));
add(buttonPanel, BorderLayout.SOUTH);

textArea = new JTextArea(8, 40);
add(textArea, BorderLayout.CENTER);
```

Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

JCheckBox

- Purpose:
  - Used for multi-option user input that the user may select or deselect by clicking on them
- Constructors:
  - **JCheckBox()**
    - Creates an initially unchecked checkbox with no label
  - **JCheckBox(String text)**
    - Creates a checkbox (initially unchecked) with the specified text; see setSelected for changing it
  - **JCheckBox(String text, boolean selected)**
    - Creates a checkbox with the specified text
      - The initial state is determined by the boolean value provided
      - A value of true means it is checked

JCheckBox – Methods

- **boolean isSelected()**
  - returns the state of the checkbox
- **void setSelected(boolean state)**
  - sets the checkbox to a new state
- **String getText()**
- **void setText(String text)**
  - gets or sets the button's text
- **addItemListener**
  - Add an ItemListener to process ItemEvent in itemStateChanged

JCheckBox Demo

```
setLayout(new GridLayout(2, 1));
// set up JTextField and set its font
JPanel p1 = new JPanel();
p1.add(field = new JTextField("Watch the font style change", 20));
field.setFont(new Font("Serif", Font.PLAIN, 16));
add(p1);


// create checkbox objects
JCheckBox bold, italic;
JPanel p2 = new JPanel();
p2.add( bold = new JCheckBox("Bold"));
p2.add( italic = new JCheckBox("Italic"));
add(p2);
```

Handling JCheckBox event

- A JCheckBox generates an Item event whenever it changes state (checked/unchecked)
  - Handle the event
    - implements ItemListener
    - Method: **public void itemStateChanged(ItemEvent e)**
      - The ItemEvent class has a `getItem` method which returns the item just selected or deselected
    - Register: addItemListener
  - Ignore the event
    - With checkboxes, it is relatively common to ignore the select/deselect event when it occurs
    - Instead, you look up the state of the checkbox later using the **isSelected** method of JCheckBox

Handling JCheckBox event


```
@Override
public void itemStateChanged(ItemEvent e) {
   Font f= txtField.getFont();
   // process bold checkbox events
   if (e.getItem() == chkBold)
         txtField.setFont(new Font(f.getName(), f.getStyle() ^ Font.BOLD, f.getSize()));

   // process italic checkbox events
   if (e.getItem() == chkItalic)
         txtField.setFont(new Font(f.getName(), f.getStyle() ^ Font.ITALIC, f.getSize()));
}
```


JRadioButton

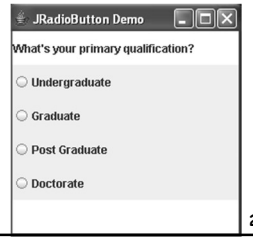
- Purpose: Used as option button to specify choices
- Only one radio button in a group can be selected at any given time
- To define the group of radio buttons, create a ButtonGroup object

```
ButtonGroup group = new ButtonGroup();
group.add(smallButton);
group.add(mediumButton);
```

- Note: the ButtonGroup controls only the behavior of the buttons; if you want to group the buttons for layout purposes, you also need to add them to a container
- When the user checks a radio button, JRadioButton generates an Action event (…)

JRadioButton Demo

```
setLayout(new GridLayout(6,1));
JRadioButton radUnder, radGra, radPost, radDoc;
add(new JLabel("What's your primary qualification?" ));
add(radUnder=new JRadioButton("Undergraduate"));
add(radGra=new JRadioButton("Graduate"));
add(radPost=new JRadioButton("Post Graduate"));
add(radDoc=new JRadioButton("Doctorate"));
ButtonGroup group = new ButtonGroup();
group.add (radUnder);
group.add (radGra);
group.add (radPost);
group.add (radDoc);
```



DemoCheckBoxRadio.java

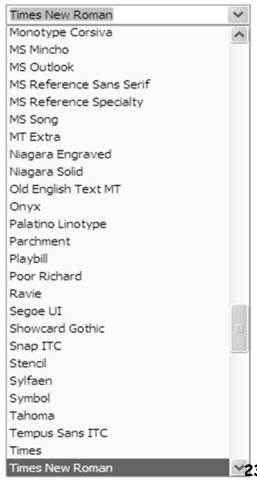
```
1  package gui.com.vovanhai;
2
3  import java.awt.BorderLayout;
17
18 public class DemoCheckBoxRadio extends JFrame
19     implements  ItemListener/*checkbox event*/,
20                 ActionListener/*radio event*/{
21
22     private JCheckBox italicCheckBox;private JLabel displatLabel;
23     private JRadioButton leftRadioButton,rightRadioButton;
24     public DemoCheckBoxRadio() {
25         setTitle("derived button");
26         setSize(300, 200);setDefaultCloseOperation(EXIT_ON_CLOSE);
27         JPanel pTop=new JPanel();this.add(pTop,BorderLayout.NORTH);
28         pTop.add(italicCheckBox=new JCheckBox("Italic"));
29         pTop.add(leftRadioButton=new JRadioButton("Left",true));
30         pTop.add(rightRadioButton=new JRadioButton("Right"));
31         ButtonGroup group=new ButtonGroup();/*group radios*/
32         group.add(leftRadioButton);group.add(rightRadioButton);
33         this.add(displatLabel=new JLabel("Hello world!"));
34         italicCheckBox.addItemListener(this);/*checkbox listener*/
35         leftRadioButton.addActionListener(this);/*radio listener*/
36         rightRadioButton.addActionListener(this);/*radio listener*/
37     }
38
```

JRadioButton & JCheckBox demo


```
39     @Override
40     public void itemStateChanged(ItemEvent e) {
41         Font f=displatLabel.getFont();
42         if(e.getStateChange()==ItemEvent.SELECTED)//if it was selected
43             displatLabel.setFont(new Font(f.getName(),Font.ITALIC,f.getSize()));
44         else
45             displatLabel.setFont(new Font(f.getName(),Font.PLAIN,f.getSize()));
46     }
47
48     @Override
49     public void actionPerformed(ActionEvent e) {
50         Object o=e.getSource();
51         if(o.equals(leftRadioButton))//determines the object
52             displatLabel.setHorizontalAlignment(SwingConstants.LEFT);
53         else if(o.equals(rightRadioButton))
54             displatLabel.setHorizontalAlignment(SwingConstants.RIGHT);
55     }
56     /*Main method*/
57     public static void main(String[] args) throws Exception {
58         new DemoCheckBoxRadio().setVisible(true);
59     }
60 }
```


JComboBox

- Purpose
  - To present a set of choices in a small space
- Current selection
  - item displaying
- Can be editable
  - A JComboBox can be either editable or uneditable (default)



JComboBox - Constructors

- JComboBox()
  - Creates a JComboBox with a default data model
- JComboBox(Object[] items)
  - Creates a JComboBox that contains the elements of the specified array
  - Example:

    String[] words= { "quick", "brown", "hungry", "wild", ... };
    JComboBox wordChoose = new JComboBox(words);
- JComboBox(ComboBoxModel asModel)
  - Creates a JComboBox that takes its items from an existing ComboBoxModel

JComboBox – Methods

- **void addItem(Object item)**
  - adds the specified item to the end of the combo box
- **Object getItemAt(int index)**
  - returns the item at the specified index
- **int getSelectedIndex()**
  - returns the position of selected item
- **void setSelectedIndex(int index)**
  - sets the selected index
- **Object getSelectedItem()**
  - returns the currently selected item
- **void setSelectedItem(Object item)**
  - sets the selected item

JComboBox – Methods (cont.)

- **void removeAllItems()**
  - removes all items from the combo box
- **void removeItem(Object item)**
  - removes an item from the combo box
- **void removeItemAt(int index)**
  - removes the item at an index
- **int getItemCount()**
  - returns the number of items in the list
- **void setEditable(boolean flag)**
  - sets whether this combo box is editable (default by false). Note that editing affects only the current item, it does not change the content of the list

Handle event in JComboBox

- When the user selects an item from a combo box, the combo box generates an Action event or Item event
  - implement?
  - method?

```
public void actionPerformed(ActionEvent e) {
    // sử dụng hàm getSelectedItem()
    // hoặc getSelectedIndex()
    // để lấy mục đang được chọn trên combo
}
```

  - register?

JComboBox Demo

```
public class JComboBoxDemo extends JFrame implements ActionListener {
  JComboBox faceCombo;
  JLabel label;
  public JComboBoxDemo()  {
     setTitle("JComboBox Demo");
     label = new JLabel("The quick brown fox jumps over the lazy dog.");
     label.setFont(new Font("Serif", Font.PLAIN, 12));
     add(label, BorderLayout.CENTER);
     // make a combo box
     faceCombo = new JComboBox();
     faceCombo.addItem("Serif");
     faceCombo.addItem("SansSerif");
     faceCombo.addItem("Monospaced");
     add(faceCombo, BorderLayout.SOUTH);
     faceCombo.addActionListener(this);
     setSize(300, 200);
  }
  public void actionPerformed (ActionEvent e)  {
    String fontName = (String)faceCombo.getSelectedItem();
    label.setFont (new Font(fontName, label.getFont().getStyle(), label.getFont().getSize()));
  }
```

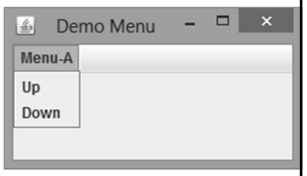
Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

JMenu

- A menu offers options to user
- Menu usually appears either in a menu bar or as a popup menu
- A JFrame often has a menu bar containing many menus; and each menu can contain many choices
- Menu bar can be added to a JFrame with the method setJMenuBar

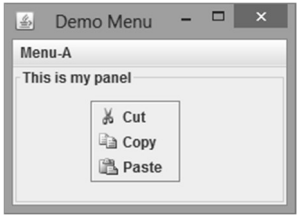
```
void createMenu(){
    JMenuBar menuBar = new JMenuBar();
    this.setJMenuBar(menuBar);//set menu to JFrame

    JMenu aMenu = new JMenu("Menu-A");
    menuBar.add(aMenu);

    JMenuItem upItem = new JMenuItem("Up");
    JMenuItem downItem = new JMenuItem("Down");

    aMenu.add(upItem);
    aMenu.add(downItem);
}
```



JPopupMenu

- A `pop-up menu` is a menu that is not attached to a menu bar but that floats somewhere
- It is also used as a *shortcut menu*, which is activated by the right click of the mouse
- To pop up a menu when the user clicks on a component, simply call the `setComponentPopupMenu` method

JPopupMenu Demo




```
void createPopupMenu(){
    JPopupMenu popup = new JPopupMenu();
    JMenuItem copyItem = new JMenuItem("Copy",new ImageIcon("images/CopyHS.png"));
    JMenuItem cutItem = new JMenuItem("Cut",new ImageIcon("images/CutHS.png"));
    JMenuItem pasteItem = new JMenuItem("Paste",new ImageIcon("images/PasteHS.png"));
    popup.add(cutItem);
    popup.add(copyItem);
    popup.add(pasteItem);
    p.setComponentPopupMenu(popup);//set popup menu for component
}
```

Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

Mnemonics

- Mnemonics (shortcut keys) provide quick access to menu commands or button commands through the keyboard
- Once the mnemonic has been established, the character in the label will appear underlined
- You can supply a mnemonic letter by calling the `setMnemonic` method:
- Example:

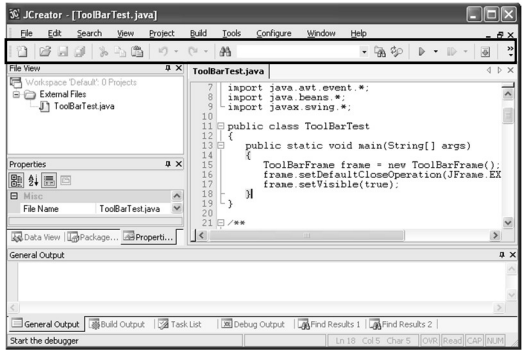
```
JMenu helpMenu = new JMenu("Help");
helpMenu.setMnemonic('H');
```


Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

Toolbar

- A toolbar is a button bar that gives quick access to the most commonly used commands in a program



JToolbar Demo


```
public class JToolbarDemo extends JFrame  implements ActionLi
    private JPanel panel;
    private JButton btnBlue, btnYell, btnRed, btnExit;
    public JToolbarDemo() {
        super("JToolBar Demo");
        JToolBar bar = new JToolBar();
        bar.add(btnBlue = new JButton(new ImageIcon("blue-ball.gif")));
        bar.add(btnYell = new JButton(new ImageIcon("yellow-ball.gif")));
        bar.add(btnRed = new JButton(new ImageIcon("red-ball.gif")));
        bar.addSeparator();
        bar.add(btnExit = new JButton(new ImageIcon("exit.gif")));
        add(bar, BorderLayout.NORTH);
        panel = new JPanel();
        add(panel);
        setSize(300, 200);
```


JToolbar Demo (contd.)

```
        btnBlue.addActionListener(this);
        btnYell.addActionListener(this);
        btnRed.addActionListener(this);
        btnExit.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)        {
        Object o = e.getSource();
        if (o.equals(btnBlue))      panel.setBackground(Color.BLUE);
        if (o.equals(btnYell))      panel.setBackground(Color.YELLOW);
        if (o.equals(btnRed))       panel.setBackground(Color.RED);
        if (o.equals(btnExit))      System.exit(0);
    }
    public static void main(String[] args)        {
        new JToolbarDemo().setVisible(true);    }
}
```


Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

Tooltip

- A `tooltip` represents a text tip that is displayed when the mouse cursor rests for a moment over a button and when the user moves the mouse away, the `tooltip` is removed
  - The `tooltip` text is displayed inside a colored rectangle
- Add `tooltip` by calling the `setToolTipText` method
- Example:

```
exitButton.setToolTipText("Bấm vào đây để thoát chương trình");
```

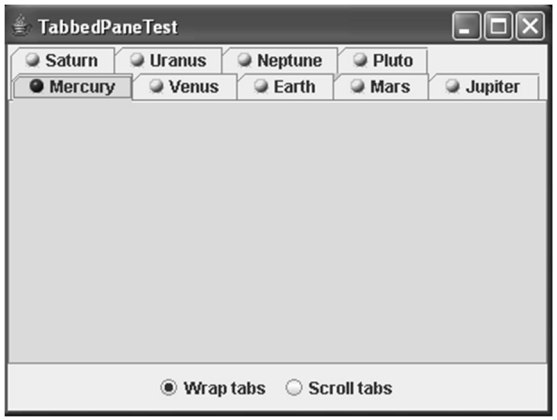
04. Graphic User Interface in Java

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

Tabbed panes



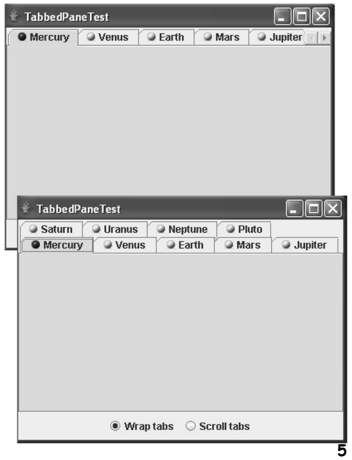
JTabbedPane

- Purpose
  - Break up a complex dialog box into subsets of related options
- A tabbed pane is defined by the `JTabbedPane` class
- To create a tabbed pane, you first construct a `JTabbedPane` object, then you add tabs to it
  - Example:

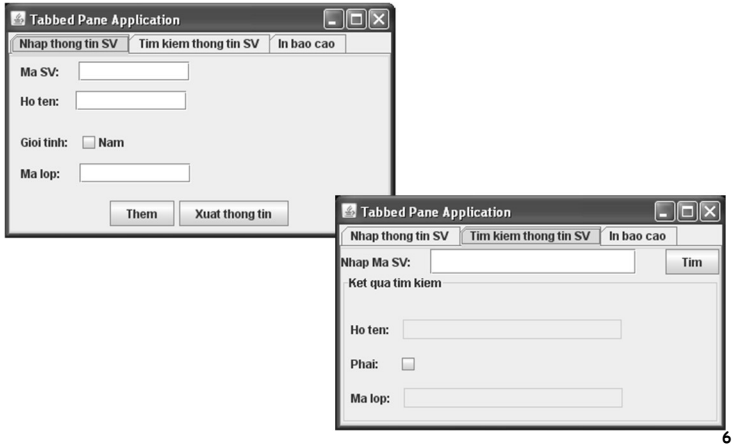
```
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.addTab("ScreenSaver", new ImageIcon("Ss.gif"), panel1);
```

JTabbedPane

- You set the tab layout to wrapped or scrolling mode by calling `setTabLayoutPolicy` method:
  - tabbedPane.setTabLayoutPolicy(JTabbedPane.WRAP_TAB_LAYOUT);
  - tabbedPane.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);



Example: JTabbedPaneExample.java



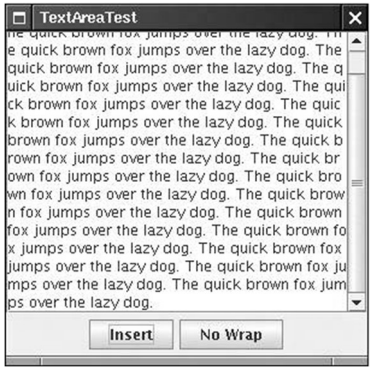
Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

JScrollPane

- Components do not automatically provide a scrollbar, such as: JTextArea, JList, JTable,…
- A `JScrollPane` object is used to provide the automatic scrolling capability for components
  - `JScrollPane` automatically appears if there is much data than the component can display, and they vanish again if text is deleted

TextAreaTest




```
textArea = new JTextArea(8, 40);
scrollPane = new JScrollPane(textArea);
add(scrollPane, BorderLayout.CENTER);
```

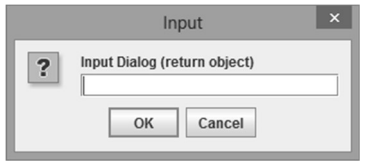

Outline

- **Text component**
- **Choice component**
- **Menu**
- **Mnemonic**
- **Toolbar**
- **Tooltip**
- **Tabbed pane**
- **Scroll pane**
- **Dialog box**

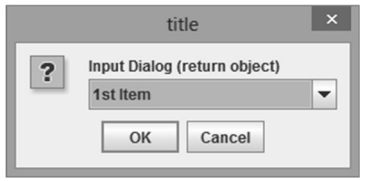
Dialog box

- A `dialog box` is a window that appears on top of any currently active window
- It may be used to:
  - Show message / confirm an action / input data
  - Display information
  - Choose a file
  - Pick a color
- Dialog box in Swing
  - JOptionPane
  - JDialog
  - JFileChooser
  - JColorChooser

Input Dialog




```
JOptionPane.showInputDialog(null, "Input Dialog (return object)");
```

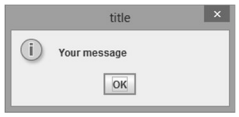



```
String []items={"1st Item","2nd Item","3rd Item" };
JOptionPane.showInputDialog(null, "Input Dialog (return object)", "title",
        JOptionPane.QUESTION_MESSAGE, null, items, "2nd Item");
```

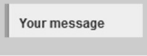
Confirm Dialog


```
int result= JOptionPane.showConfirmDialog(null,
        "Confirmation (return integer value)",
        "title", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null);
if(result==JOptionPane.YES_OPTION){//YES button is selected
    //do your works
}
```


Message Dialog




```
JOptionPane.showMessageDialog(null, "Your message", "title",
        JOptionPane.INFORMATION_MESSAGE, null);
```

	JOptionPane.INFORMATION_MESSAGE
	JOptionPane.ERROR_MESSAGE
	JOptionPane.PLAIN_MESSAGE
	JOptionPane.WARNING_MESSAGE
	JOptionPane.QUESTION_MESSAGE

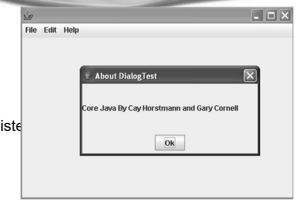
Option Dialog


```
String []buttons={"First Button","Second First Button","Third Button" };
int x=JOptionPane.showOptionDialog(null, "Option Dialog (return an integer)",
        "title",JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE,
        null, buttons, "");
if(x==0){/*First button is selected*/}
else if(x==1){/*Second button is selected*/}
else if(x==2){/*Third button is selected*/}
else /*if(x==JOptionPane.CLOSED_OPTION) */
{/*close button is selected*/}
```


JDialog

- To implement a dialog box, you derive a class from JDialog
- A modal dialog box won't let users interact with the remaining windows of the application until it is closed
  - You use a `modal` dialog box when you need information from the user before you can proceed with execution

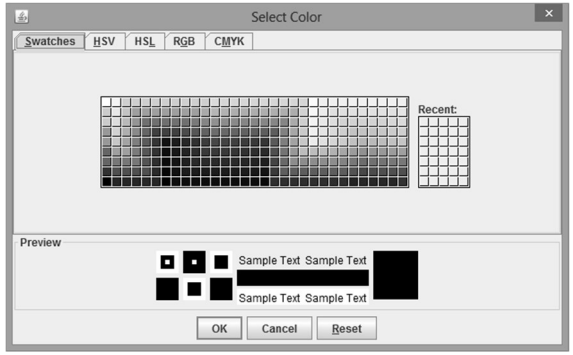
JDialog Demo




```
public class AboutDialog extends JDialog implements ActionListe
  JPanel panel;   JButton ok;   JLabel infor;
  public AboutDialog(JFrame owner) {
            super(owner,  "About DialogTest",  true);
            infor = new JLabel("Core Java By Cay Horstmann and Gary Cornell");
            add(infor, BorderLayout.CENTER);
            ok = new JButton("Ok");
            ok.addActionListener(this);
            panel = new JPanel();
            panel.add(ok);
            add(panel, BorderLayout.SOUTH);
            setSize(300, 150);
            setLocation(300,300);
  }
  public void actionPerformed(ActionEvent e)  { setVisible(false); }
}
```

```
AboutDialog dialog;
public void actionPerformed(ActionEvent e)
{
    if (dialog == null)
        dialog = new AboutDialog(this);
    dialog.setVisible(true);
}
```

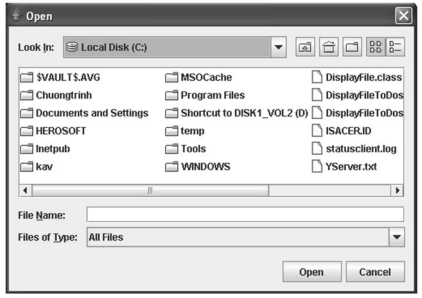

JColorChooser




```
Color selColor=JColorChooser.showDialog(this, "Select Color", Color.black);
//Do your works with your selected color
```


JFileChooser

- Purpose
  - display a dialog for opening a file or saving a file



JFileChooser

- To create a JFileChooser object:
  - **JFileChooser chooser = new JFileChooser();**
- To show the dialog box, calling the `showOpenDialog` or `showSaveDialog` method, return:
  - JFileChooser.APPROVE_OPTION: if approval (Yes, Ok) is chosen
  - JFileChooser.CANCEL_OPTION: if Cancel is chosen
  - JFileChooser.ERROR_OPTION: if an error occured
- To get the selected file or files:
  - **File f = chooser.getSelectedFile();**
  - **File[] f = chooser.getSelectedFiles();**
- To get path of the selected file:
  - **String filename = chooser.getSelectedFile().getPath();**

JFileChooser

- To set the current directory:
  - **setCurrentDirectory(new File("."));**
- To allow to select multiple files in the dialog
  - **setMultiSelectionEnabled(true);**
- To allow to select only directories, only files or files and directories:
  - setFileSelectionMode(JFileChooser.FILES_ONLY) (default)
  - setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY)
  - setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES)
- To restrict the display of files in the dialog to those of a particular type, you need to set a file filter

Dialog Boxes - JFileChooser

Choosing multiple file types

```
private void showOpenFileDialog() {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setCurrentDirectory(new File(System.getProperty("user.home")));
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    fileChooser.addChoosableFileFilter(
            new FileNameExtensionFilter("PDF Documents", "pdf"));
    fileChooser.addChoosableFileFilter(
            new FileNameExtensionFilter("MS Office Documents", "docx", "xlsx", "pptx"));
    fileChooser.addChoosableFileFilter(
            new FileNameExtensionFilter("Images", "jpg", "png", "gif", "bmp"));
    fileChooser.setAcceptAllFileFilterUsed(true);
    int result = fileChooser.showOpenDialog(this);
    if (result == JFileChooser.APPROVE_OPTION) {
        File selectedFile = fileChooser.getSelectedFile();
        System.out.println("Selected file: " + selectedFile.getAbsolutePath());
    }
}
```


Toolbar – Constructors and Methods

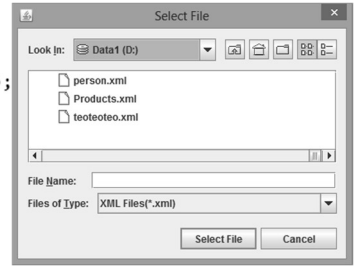
- JToolBar()
- JToolBar(String titleString)
- JToolBar(int orientation)
- JToolBar(String titleString, int orientation)
  - construct a toolbar with the given title string and orientation, orientation is one of SwingConstants.HORIZONTAL (the default) and SwingConstants.VERTICAL
- void add(Component comp)
  - add a component to toolbar
- void addSeparator()
  - adds a separator to the end of the toolbar

JScrollPane - Contructors

- **`JScrollPane(Component comp)`**
  - Creates a JScrollPane that displays the contents of the specified component, where both horizontal and vertical scrollbars appear whenever the component's contents are larger than the view
- **`JScrollPane(Component comp, int vsbPolicy, int hsbPolicy)`**
  - Creates a JScrollPane that displays the view component in a viewport whose view position can be controlled with a pair of scrollbars
    - JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
    - JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS
    - JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED
    - JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED

JFileChooser Demo

```
JFileChooser fc=new JFileChooser("C:/");
fc.setFileFilter(new FileFilter() {
    @Override
    public String getDescription() {
        return "XML Files(*.xml)";
    }

    @Override
    public boolean accept(File f) {
        String fileName=f.getName().toLowerCase();
        return f.isDirectory()||fileName.endsWith("xml");
    }
});

if(fc.showDialog(null, "Select File")==JFileChooser.APPROVE_OPTION){
    //do your works
}
```



Methods

- **`void addTab(String title, Component c)`**
- **`void addTab(String title, Icon icon, Component c)`**
- **`void addTab(String title, Icon icon, Component c, String tooltip)`**
  - add a tab to the end of the tabbed pane
- **`void insertTab(String title, Icon icon, Component c, String tooltip, int index)`**
  - inserts a tab to the tabbed pane at the given index
- **`void removeTabAt(int index)`**
  - removes the tab at the given index
- **`void setSelectedIndex(int index)`**
  - selects the tab at the given index
- **`int getSelectedIndex()`**
  - returns the index of the selected tab

JOptionPane

- Purpose: presenting information, confirming an action, or accepting an input value
- Four static methods to show these simple dialogs:
  - showMessageDialog(…)
    - Show a message and wait for the user to click OK
  - showConfirmDialog(…)
    - Show a message and get a confirmation (like OK/Cancel)
  - showOptionDialog(…)
    - Show a message and get a user option from a set of options
  - showInputDialog(…)
    - Show a message and get one line of user input

JOptionPane - Parameters

- Component parent
  - The parent component (this or null)
- Object message
  - The message to show on the dialog (can be a string, icon, component, or an array of them)
- String title
  - the string in the title bar of the dialog
- int messageType: can be one of ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE
- int optionType: can be one of DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, OK_CANCEL_OPTION
- Icon icon
  - an icon to show instead of one of the standard icons

JOptionPane – Message and Confirm


```
JOptionPane.showMessageDialog(this, "Message", "Title",
                              JOptionPane.WARNING_MESSAGE);
```

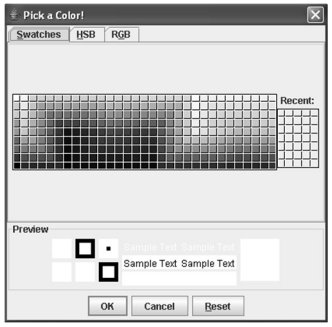

```
int selection = JOptionPane.showConfirmDialog(this, "Message", "Title",
                                              JOptionPane.OK_CANCEL_OPTION,
                                              JOptionPane.WARNING_MESSAGE);
if (selection == JOptionPane.OK_OPTION) . . .
```


JOptionPane Demo

```
import javax.swing.*;
public class JOptionPaneDemo {
  public static void main (String[] args)   {
    String numStr, result;
    int num, again;
    do {
          numStr = JOptionPane.showInputDialog ("Enter an integer: ");
          num = Integer.parseInt(numStr);
          result = "That number is " + ((num%2 == 0) ? "even" : "odd");
          JOptionPane.showMessageDialog (null, result);
          again = JOptionPane.showConfirmDialog (null, "Do Another?");
    } while (again == JOptionPane.YES_OPTION);
  }
}
```


JColorChooser

- Purpose
  - To allow the user to select a color
- A dialog box that lets the user click on a color of choice from a palette or specify the color using RGB values

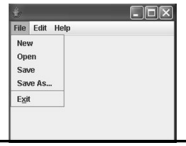


JColorChooser Demo

```
public class JColorChooserDemo extends JFrame
                        implements ActionListener {
   JMenuBar mnubar;
   JMenu mnuFile, mnuEdit;
   JMenuItem mnuFileExit;
   JMenuItem mnuEditColor;
   JPanel panel;

   public JColorChooserDemo()   {
     mnubar = new JMenuBar();
     mnuFile = new JMenu("File");
     mnubar.add(mnuFile);
     mnuEdit = new JMenu("Edit");
     mnubar.add(mnuEdit);

     // menu File
     mnuFileExit = new JMenuItem("Exit");
     mnuFile.add(mnuFileExit);

     // menu Edit
     mnuEditColor = new JMenuItem("Color");
     mnuEditColor.addActionListener(this);
     mnuEdit.add(mnuEditColor);

     setJMenuBar(mnubar);
     panel = new JPanel();
     add (panel);
     setLocation(200,100);
     setSize(500, 500);
     setVisible(true);
   }
   public void actionPerformed(ActionEvent e)   {
     Color mau= panel.getBackground();
      mau = JColorChooser.showDialog (this,
                        "Pick a Color!", mau);
     panel.setBackground (mau);
   }
   public static void main(String[] args)   {
     new JColorChooserDemo();
   }
}
```

JMenu Demo

```
public class JMenuDemo extends JFrame {
    JMenuBar mnubar;
    JMenu mnuFile, mnuEdit, mnuHelp;
    JMenuItem mnuFileNew, mnuFileOpen, mnuFileSave, mnuFileSaveAs, mnuFileExit;
    JMenuItem mnuEditCut, mnuEditCopy, mnuEditPaste, mnuEditOption;
    public JMenuDemo()      {
        mnubar = new JMenuBar();
        setJMenuBar(mnubar);

        mnuFile = new JMenu("File");     mnubar.add(mnuFile);
        mnuEdit = new JMenu("Edit");     mnubar.add(mnuEdit);
        mnuHelp = new JMenu("Help");     mnubar.add(mnuHelp);

        // menu File
        mnuFileNew = new JMenuItem("New");     mnuFile.add(mnuFileNew);
        mnuFileOpen = new JMenuItem("Open");     mnuFile.add(mnuFileOpen);
        mnuFileSave = new JMenuItem("Save");     mnuFile.add(mnuFileSave);
        mnuFileSaveAs = new JMenuItem("Save As...");     mnuFile.add(mnuFileSaveAs);
        mnuFile.addSeparator();
        mnuFileExit = new JMenuItem("Exit", 'x');     mnuFile.add(mnuFileExit);
```

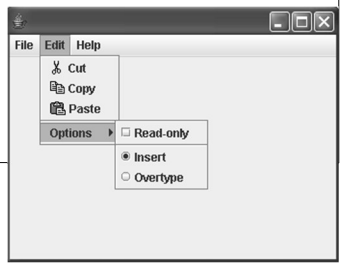


JMenu Demo (contd.)

```
        //menu Edit
        mnuEditCut = new JMenuItem("Cut", new ImageIcon("cut.gif"));
        mnuEditCopy = new JMenuItem("Copy", new ImageIcon("copy.gif"));
        mnuEditPaste = new JMenuItem("Paste", new ImageIcon("paste.gif"));
        mnuEditOption = new JMenu("Options");
        mnuEdit.add(mnuEditCut);
        mnuEdit.add(mnuEditCopy);
        mnuEdit.add(mnuEditPaste);
        mnuEdit.addSeparator();
        mnuEdit.add(mnuEditOption);
        //…
        setLocation(200, 200);
        setSize(500, 400);
        setVisible(true);
    }
    public static void main(String[] args)          {
                new JMenuDemo();
    }
}
```

JMenu Demo(contd.)

```
JCheckBoxMenuItem readonlyItem = new JCheckBoxMenuItem("Read-only");
optionsMenu.add(readonlyItem);

JRadioButtonMenuItem insertItem = new JRadioButtonMenuItem("Insert", true);
JRadioButtonMenuItem overtypeItem = new JRadioButtonMenuItem("Overtype");
optionsMenu.add(insertItem);
optionsMenu.add(overtypeItem);

ButtonGroup group = new ButtonGroup();
group.add(insertItem);
group.add(overtypeItem);
```

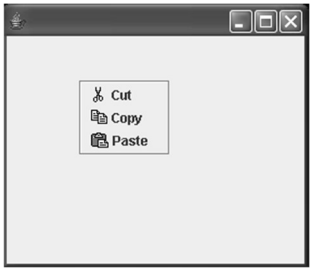


JMenuPopup Demo

```
public class JMenuPopupDemo extends JFrame implements ActionListener {
    JPopupMenu popup;
    JMenuItem mnuEditCut, mnuEditCopy, mnuEditPaste;
    public JMenuPopupDemo()          {
        popup = new JPopupMenu();
        mnuEditCut = new JMenuItem("Cut", new ImageIcon("cut.gif"));
        mnuEditCopy = new JMenuItem("Copy", new ImageIcon("copy.gif"));
        mnuEditPaste = new JMenuItem("Paste", new ImageIcon("paste.gif"));

        popup.add(mnuEditCut);
        popup.add(mnuEditCopy);
        popup.add(mnuEditPaste);
        mnuEditCut.addActionListener(this);
        mnuEditCopy.addActionListener(this);
        mnuEditPaste.addActionListener(this);
        JPanel panel = new JPanel();
        panel.setComponentPopupMenu(popup);
        panel.addMouseListener( new MouseAdapter() {} );
        add(panel);
```

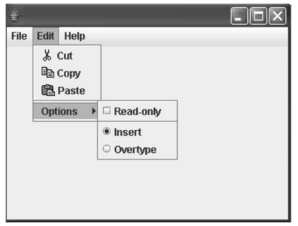
JMenuPopup Demo (contd.)

```
            setLocation(200, 200);
            setSize(500, 400);
            setVisible(true);
      }

      public void actionPerformed(ActionEvent e) {
            Object o = e.getSource();
            System.out.println(((JMenuItem)o).getText() + " selected.");
      }
      public static void main(String[] args)        {
            new JMenuPopupDemo();
      }
}
```

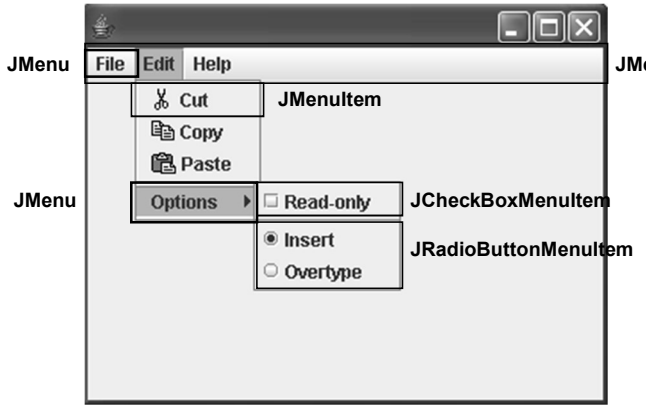


Menu

- Menus display several options that are broadly categorized
  - Menu bar (JMenuBar)
  - Menu (JMenu)
  - Menu item (JMenuItem)
- Clicking a JMenuItem, it generates an Action event



Components in the menu



JMenuBar

- JMenuBar is a container for JMenu
- Constructor:
  - JMenuBar()
- To set the menu bar for JFrame, using method:
  - void setJMenuBar(JMenuBar menubar)
  - To add a JMenu to the menu bar, using add method of JMenuBar
  - add(JMenu menu)

JMenu - Constructors and Methods

- JMenu(String label)
  - constructs a menu
- JMenuItem add(JMenuItem item)
  - adds a menu item (or a menu)
- JMenuItem add(String label)
  - adds a menu item to this menu
- void addSeparator()
  - adds a separator line to the menu
- void remove(int index)
  - removes a specific item from the menu
- void remove(JMenuItem item)
  - removes a specific item from the menu

JMenuItem – Constructors

- JMenuItem()
  - constructs an empty menu item
- JMenuItem(String label)
  - constructs a menu item with a given label
- JMenuItem(Icon icon)
  - constructs a menu item with the given icon
- JMenuItem(String label, Icon icon)
  - constructs a menu item with the given label and icon
- JMenuItem(String label, int mnemonic)
  - constructs a menu item with the given label and mnemonic

JPopupMenu – Methods

- **`JMenuItem add(JMenuItem menuItem)`**
  - appends the specified menu item at the end of the menu
- **`JMenuItem add(String s)`**
  - creates a new menu item with the specified text and appends it to the end of the menu
- **`void show(Component c, int x, int y)`**
  - displays the popup menu at the position (x,y) in the coordinate space of the component "c"
- **`boolean isPopupTrigger()`**
  - determines whether the mouse event is considered as the popup trigger for the current platform

05. Graphic User Interface in Java

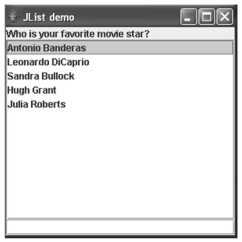
Faculty of Information Technologies
Industrial University of Ho Chi Minh City

GUI components (p4)

- JList
- JTable
- JTree
- JSplitPane
- Jslider
- MDI - multiple document interface

JList

- Purpose
  - To present a list of items from which the user can choose
- Behavior
  - Items in JList can be selected individually or in a group
  - A JList does not provide support for double-click action



JList – Constructors

- **JList()**
  - Constructs a JList with an empty model
- **JList(Object[] listData)**
  - Displays the elements of the specified array
  - Example:

```
String[] words= { "quick", "brown", "hungry", "wild", ... };
JList wordList = new JList(words);
```

- **JList (ListModel dataModel)**
  - Displays the elements in the specified, non-null list model

JList – Methods

- **int getSelectedIndex()**
- **void setSelectedIndex(int index)**
  - gets or sets the selected index
- **Object getSelectedValue()**
  - returns the first selected value or null if the selection is empty
- **Object[] getSelectedValues()**
  - returns the selected values or an empty array if the selection is empty
- **boolean isSelectedIndex(int index)**
  - returns true if the specified index is selected
- **boolean isSelectionEmpty()**
  - returns true if no item is currently selected

JList – Methods (contd.)

- **int getVisibleRowCount()**
- **void setVisibleRowCount(int height)**
  - get or set the number of rows in the list that can be displayed without a scroll bar
- **int getSelectionMode()**
- **void setSelectionMode(int mode)**
  - **ListSelectionModel.SINGLE_SELECTION**, *only one item can be selected at a time*
  - **ListSelectionModel.SINGLE_INTERVAL_SELECTION**, *multiple, contiguous items can be selected at a time*
  - **ListSelectionModel.MULTIPLE_INTERVAL_SELECTION**, *any combination of items can be selected.* By default, a user can select multiple items

Handle event of JList

- When the current selection changes, `JList` object generates a ListSelection event
  - Method:

```
public void valueChanged (ListSelectionEvent e) {
        Object value = list.getSelectedValue();
         //do something with value
}
```

```
public void valueChanged (ListSelectionEvent e) {
        Object[] items = list.getSelectedValues();
        for (Object value : items)
                //do something with value
}
```


JList with fixed set of choices

- Build JList:
  - The simplest way to use a JList is to supply an *array of strings* to the JList constructor. Cannot add or remove elements once the JList is created

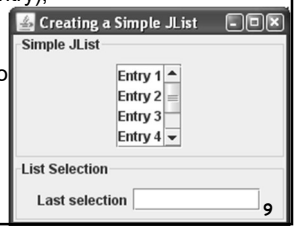
```
String[] options = { "Option 1", ... , "Option N"};
JList optionList = new JList(options);
```

- Set visible rows
  - Call **`setVisibleRowCount`** and drop JList into JScrollPane

```
optionList.setVisibleRowCount(4);
JScrollPane scrolList = new JScrollPane(optionList);
someContainer.add(scrolList);
```

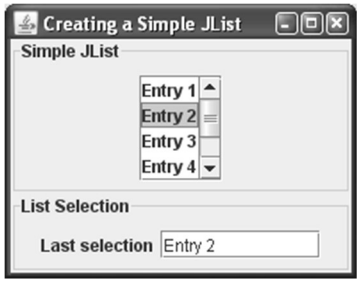
JList Demo

```
String[] entries = { "Entry 1", "Entry 2", "Entry 3",
                     "Entry 4", "Entry 5", "Entry 6" };
JList lstEntry;
lstEntry = new JList(entries);
lstEntry.setVisibleRowCount(4);
JScrollPane listPane = new JScrollPane(lstEntry);
JPanel pCen = new JPanel();
pCen.setBorder(BorderFactory.createTitledBo
pCen.add(listPane);
add(pCen, BorderLayout.CENTER);
```



JList Demo (contd.)

```
public void valueChanged(ListSelectionEvent e)
{
    txtSelected.setText(lstEntry.getSelectedValue().toString());
}
```



Editing JList

- To add or remove elements, you must access the `ListModel`
- `ListModel` is an interface. How do you obtain it?
  - Constructing your own list by creating a class that implements the `ListModel` interface
  - Using a `DefaultListModel` class

JList with changeable choices

- Build JList:
  - Create a `DefaultListModel`, add data, pass to constructor:

    ```
    DefaultListModel sampleModel = new DefaultListModel();
    JList optionList = new JList(sampleModel);
    ```
- Set visible rows
  - Same: Use `setVisibleRowCount` and a JScrollPane
- Add/remove elements
  - Use the model, not the JList directly

Methods in DefaultListModel

- **void addElement(Object obj)**
  - adds object to the end of the model
- **boolean removeElement(Object obj)**
  - removes the first occurrence of the object from the model. Return true if the object was contained in the model, false otherwise
- **void setElementAt(Object item, int index)**
  - sets item at index
- **Object getElementAt(int position)**
  - returns an element of the model at the given position
- **int getSize()**
  - returns the number of elements of the model

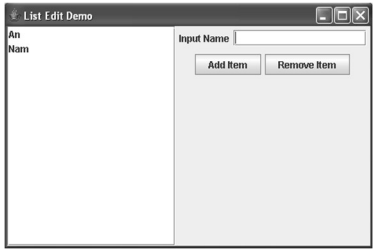
Traverse DefaultListModel

- To traverse all elements of the model, using:

```
Enumeration e = listmodel.elements();
while (e.hasMoreElements())
{
    Object o = e.nextElement();
    // process o
}
```


JList Demo (edittable)

```
public class ListEditDemo extends JFrame implements ActionListener {
    JButton btnAdd, btnRemove;
    JTextField txtName;
    DefaultListModel listmodelName;
    JList listName;

    public ListEditDemo() {
        super("List Edit Demo");

        // list
        listmodelName = new DefaultListModel();
        listName = new JList(listmodelName);
        add(new JScrollPane(listName), BorderLayout.CENTER);
```



JList Demo (edittable)

```
        JPanel pRight;
        JPanel pTop, pBottom;

        pTop = new JPanel();
        pTop.add(new JLabel("Input Name"));
        pTop.add(txtName = new JTextField(15));
        pBottom = new JPanel();
        pBottom.add(btnAdd = new JButton("Add Item"));
        pBottom.add(btnRemove = new JButton("Remove Item"));

        pRight = new JPanel(new BorderLayout());
        pRight.add(pTop, BorderLayout.NORTH );
        pRight.add(pBottom, BorderLayout.CENTER);

        add(pRight, BorderLayout.EAST);

        txtName.addActionListener(this);
        btnAdd.addActionListener(this);
        btnRemove.addActionListener(this);

        setSize(500, 320);
    }
```

JListEdit Demo (contd.)

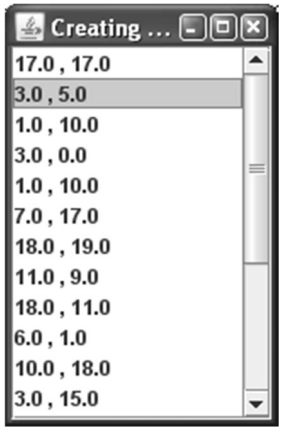
```
public void actionPerformed(ActionEvent e) {
        Object o = e.getSource();
        if ( o==btnAdd ) {
           String name = txtName.getText().trim();
           if ( name == "" )
             JOptionPane.showMessageDialog(this, "Please input name!");
             else {
                  listmodelName.addElement(name);
                  txtName.setText( "" );
             }
        }
        else if (o.equals (btnRemove))
            listmodelName.removeElement(listName.getSelectedValue());
        else if (o.equals (btnEdit))
            listmodelName.setElementAt( txtName.getText(),
                                        listName.getSelectedIndex() );
  }
  public static void main(String[] args){ new ListEditDemo().setVisible(true); }
}
```


JList with custom data model

- Build JList
  - Have existing data implement ListModel interface
    - `getElementAt`
      - Given an index, returns data element
    - `getSize`
      - Tells JList how many entries are in list
    - `addListDataListener`
      - Lets user add listeners that should be notified when an item is selected or deselected
    - `removeListDataListener`
  - Pass model to JList constructor
- Set visible rows & handle events: as before
- Add/remove items: use the model

Example: JList with custom data

- Rectangle.java
- RectangleCollection.java
- RectangleListModel.java
- JListWithRectangleListModel.java



Example: JList with custom data (contd.)

```
// Rectangle.java
public class Rectangle {
    public String toString(){ return width + " , " + height;  }  …
}
// RectangleListModel.java
public class RectangleListModel implements ListModel {
    private RectangleCollection collection;
    public RectangleListModel(RectangleCollection collection) {
          this.collection = collection;
    }
    public Object getElementAt(int index) {
          return collection.getElement(index);
    }
    public int getSize() {
          return collection.getSize();
    }
    public void addListDataListener(ListDataListener l) { }
    public void removeListDataListener(ListDataListener l) { }
}
```

Example: JList with custom data (contd.)

```
// JListRectangleGUI.java
RectangleCollection collection = new RectangleCollection();
Random gen = new Random();
for (int i = 0; i < 20; i++) {
   collection.addRectangle(gen.nextInt(20), gen.nextInt(20));
}
JList lstRect;
RectangleListModel lstModel;
lstModel = new RectangleListModel(collection);
lstRect = new JList(lstModel);
lstRect.setVisibleRowCount(6);
JScrollPane listPane = new JScrollPane(lstRect);
add(listPane, BorderLayout.CENTER);
```


JList with custom cell renderer

- Idea
  - Instead of predetermining how the JList will draw the list elements, Swing lets you specify what graphical component to use for the various entries
  - Attach a ListCellRenderer that has a getListCellRendererComponent method that determines the GUI component used for each cell
- getListCellRendererComponent arguments
  - JList: the list itself
  - Object: the value of the current cell
  - int: the index of the current cell
  - boolean: is the current cell selected?
  - boolean: does the current cell have focus?

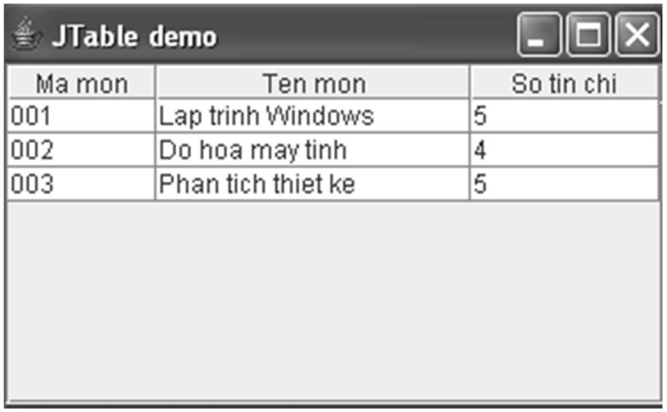
Example: JList with custom cell renderer

Outline

- **JList**
- **JTable**
- **JTree**
- **JSplitPane**
- **JSlider**
- **MDI - multiple document interface**

JTable

- A table displays a two-dimensional grid of objects



JTable

- A `JTable` consists of:
  - Rows of data
  - Columns of data
  - Column headers
  - An editor, if you want cells to be editable
  - A `TableModel`, usually a subclass of AbstractTableModel, which stores the table's data

Constructors - Methods of JTable

- **JTable(Object[][] entries, Object[] columnNames)**
  - constructs a table with a default table model
- **JTable(TableModel model)**
  - displays the elements in the specified, non-null table model
- **int getSelectedRow()**
  - returns the index of the first selected row, -1 if no row is selected
- **Object getValueAt(int row, int column)**
- **void setValueAt(Object value, int row, int column)**
  - gets or sets the value at the given row and column
- **int getRowCount()**
  - returns the number of row in the table

JTable with changeable choices

- Build JTable:
  - Create a columns name array, create a `DefaultTableModel`, pass to constructor

```
String[] cols= {"Ma mon", "Ten mon", "So tin chi"};
DefaultTableModel model=new DefaultTableModel(cols,0);
JTable table = new JTable(model);
JScrollPane pane = new JScrollPane(table);
```

- Add/remove elements
  - Use the model, not the JTable directly

Methods in DefaultTableModel

- **void addRow(Object[] rowData)**
  - add a row of data to the end of the table model
- **void insertRow(int row, Object[] rowData)**
  - adds a row of data at index row
- **void removeRow(int row)**
  - removes the given row from the model
- **void setValueAt(Object value, int row, int column)**
  - sets the value at the given row and column

Delete all rows in table

- while (model.getRowCount()>0) {

  model.remove(0);

  }

Or:

- DefaultTableModel dm =

  (DefaultTableModel)table.getModel();

dm.getDataVector().removeAllElements();

Create JTable

DemoJTable.java

```
1  package gui.com.vovanhai;
2
3  import javax.swing.JFrame;
4  import javax.swing.JScrollPane;
5  import javax.swing.JTable;
6  import javax.swing.table.DefaultTableModel;
7
8  public class DemoJTable extends JFrame{
9      private DefaultTableModel tableModel;//table model
10     private JTable table;//table
11
12     public DemoJTable() {
13         setTitle("Demo JTable");setDefaultCloseOperation(EXIT_ON_CLOSE);
14         setSize(400, 300);
15         createGUI();
16     }
17
18     private void createGUI() {
19         String []header={"Column #1","Column #2","Column #3","Column #4"};
20         //create Table Model with header and 0 rows
21         tableModel=new DefaultTableModel(header,0);
22         table=new JTable(tableModel);//create JTable based on model
23         this.add(new JScrollPane(table));table.setRowHeight(30);
24         //add one row to table (through model)
25         String []row={"St0110","Thân Thị Đẹt","12/12/2012","12 NVB"};
26         tableModel.addRow(row);//add row
27     }
```

Demo JTable

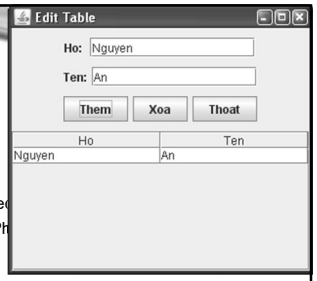
Column #1	Column #2	Column #3	Column #4
St0110	Thân Thị Đẹt	12/12/2012	12 NVB

Event of JTable

- We use MouseEvent to process event of choosing rows on JTable
  - Implement MouseListener (in *java.awt.event*)
  - Method:

```
public void mouseClicked (MouseEvent e) {}
public void mousePressed (MouseEvent e) {}
public void mouseReleased (MouseEvent e) {}
public void mouseEntered (MouseEvent e) {}
public void mouseExited (MouseEvent e) {}
```

Example: Edit JTable



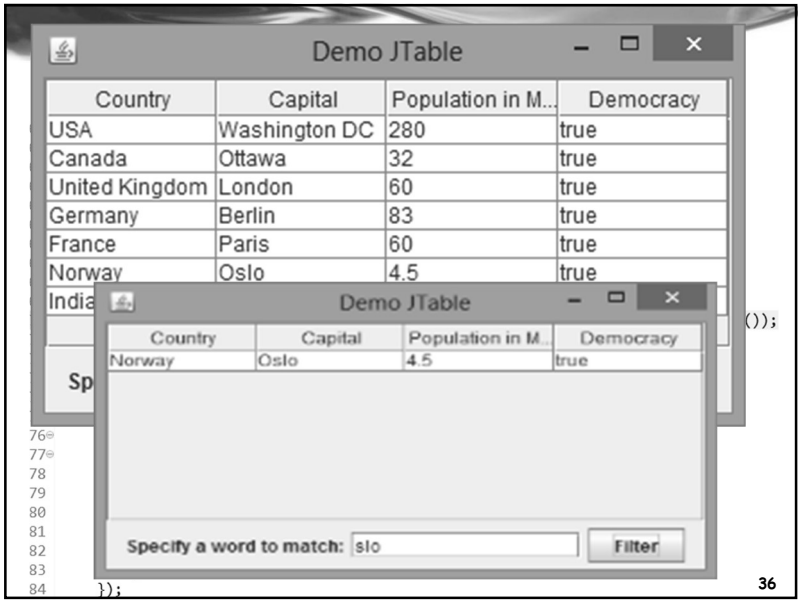

```
public void actionPerformed(ActionEvent e) {
    Object o = e.getSource();
    if (o.equals(btnAdd)) {
        if( txtHo.getText().equals("") || txtTen.getText().e
            JOptionPane.showMessageDialog(this, "Ph
        else {
            Object[] obj = new Object[2];
            obj[0] = txtHo.getText();
            obj[1] = txtTen.getText();
            model.addRow(obj);
        } }
    else if (o.equals(btnRemove)) {
        if (table.getSelectedRow() == -1)
            JOptionPane.showMessageDialog(this, "Phai chon dong can xoa.");
        else{
            if (JOptionPane.showConfirmDialog(this,"Ban co muon xoa dong nay
khong?","Canh bao",JOptionPane.YES_NO_OPTION)==JOptionPane.YES_OPTION
            model.removeRow(table.getSelectedRow());
        } }
```


Example: Edit JTable (contd.)


```
    else if (o.equals(btnEdit)) {
        if (table.getSelectedRow() == -1)
            JOptionPane.showMessageDialog(this,
                                "Phai chon dong can sua.");
        else {
            // lay dong dang chon tren table
            int row = table.getSelectedRow();
            model.setValueAt( txtHo.getText(), row, 0 );
            model.setValueAt( txtTen.getText(), row, 1 );
        }
}
```


Example: Edit JTable (contd.)

```
public void mouseClicked(MouseEvent e)
{
        // lay dong dang chon tren table
        int row = table.getSelectedRow();
        txtHo.setText(table.getValueAt(row, 0).toString());
        txtTen.setText(table.getValueAt(row, 1).toString());
}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
```

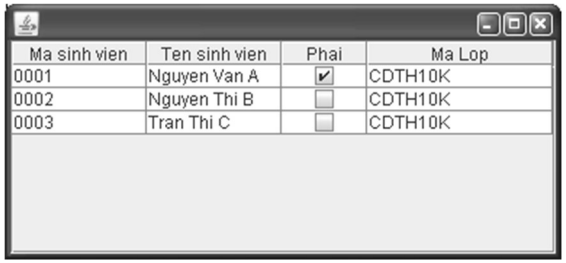



JTable with Custom Data Model

- Build custom JTable
  - Create a class has Vector field, this class extends AbstractTableModel
    - public int getColumnCount()
    - public int getRowCount()
    - public void setValueAt(Object value, int row, int col)
    - public Object getValueAt(int row, int col)
    - public String getColumnName(int col)
    - public Class getColumnClass(int c)
  - Pass model to JTable constructor
- Add/remove items: use the model
- Handle events: as before

Example: JTable with custom data

- Student.java
- StudentTableModel.java
- JTableWithStudentTableModelGUI.java



05. Graphic User Interface in Java

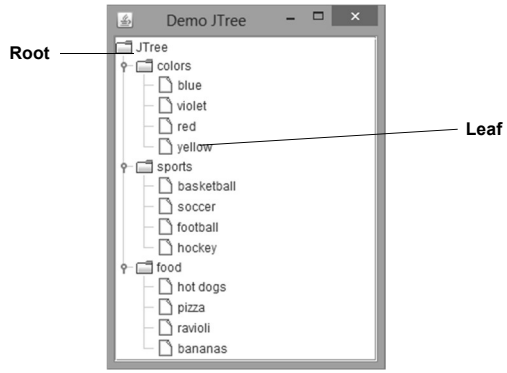
Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Outline

- JList
- JTable
- JTree
- JSplitPane
- JSlider
- MDI - multiple document interface

JTree

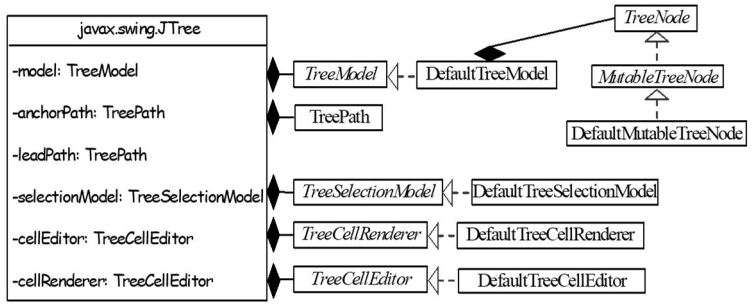
JTree is a Swing component that displays data in a treelike hierarchy



JTree

- While JTree displays the tree, the data representation of the tree is handled by TreeModel, TreeNode and TreePath
  - TreeModel represents the entire tree
  - TreeNode represents a node
  - TreePath represents a path to a node
- You can create a tree using its no-arg constructor, a tree model, a tree node, a Hashtable, an array, or a vector

JTree



The TreeSelectionModel interface handles tree node selection.

The DefaultTreeCellRenderer class provides a default tree node renderer that can display a label and/or an icon in a node.

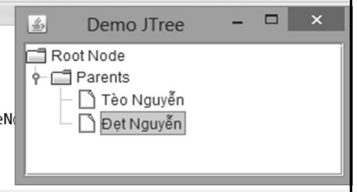
The DefaultTreeCellEditor can be used to edit the cells in a text field.

JTree

- While TreeModel represents the entire tree, TreeNode stores a single node of the tree
- MutableTreeNode defines a subinterface of TreeNode with additional methods for changing the content of the node, for inserting and removing a child node, for setting a new parent, and for removing the node itself
- DefaultMutableTreeNode is a concrete implementation of MutableTreeNode that maintains a list of children in a vector and provides the operations for creating nodes, for examining and modifying a node's parent and children, and also for examining the tree to which the node belongs
- Normally, you should use DefaultMutableTreeNode to create a tree node

DemoJTree.java

```
1  package gui.com.vovanhai;
2  import javax.swing.JFrame;
3  import javax.swing.JScrollPane;
4  import javax.swing.JTree;
5  import javax.swing.tree.DefaultMutableTreeNo
6
7  public class DemoJTree extends JFrame{
8      private JTree tree;
9      private DefaultMutableTreeNode root;
10     public DemoJTree() {
11         setTitle("Demo JTree");setDefaultCloseOperation(EXIT_ON_CLOSE);
12         setSize(400, 300);
13         root=new DefaultMutableTreeNode("Root Node");//define root node
14         tree=new JTree(root); //create tree with root node
15         this.add(new JScrollPane(tree));
16         //and an child node of the root
17         DefaultMutableTreeNode parents=new DefaultMutableTreeNode("Parents");
18         root.add(parents);//add to root
19         DefaultMutableTreeNode child1=new DefaultMutableTreeNode("Tèo Nguyễn");
20         DefaultMutableTreeNode child2=new DefaultMutableTreeNode("Đẹt Nguyễn");
21         parents.add(child1);//add 1 node to parents
22         parents.add(child2);//another ones
23     }
24
25     public static void main(String[] args) throws Exception {
28 }
```



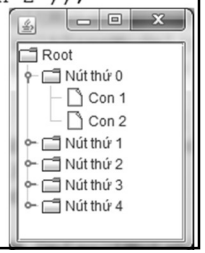
Demo JTree

Editing in JTree

- Using TreeNode and TreePath
- Using the `DefaultTreeModel` (in package javax.swing.tree)
- Constructing your own model by creating a class that implements the `TreeModel` interface

Using TreeNode

```
root = new DefaultMutableTreeNode("Root");// nút gốc
add(new JScrollPane(tree = new JTree(root)));
for (int i = 0; i < 5; i++) {
    DefaultMutableTreeNode node;
    node = new DefaultMutableTreeNode("Nút thứ " + i);
    root.add(node);

    node.add(new DefaultMutableTreeNode("Con 1"));
    node.add(new DefaultMutableTreeNode("Con 2"));
}
tree.expandRow(0);
```



Using DefaultTreeModel - Methods

- **`void insertNodeInto (MutableTreeNode newChild, MutableTreeNode parent, int index)`**
  - Inserts newChild as a new child node of parent at the given index and notifies the tree model listeners
- **`void removeNodeFromParent (MutableTreeNode node)`**
  - Removes node from this model
- **`Object getRoot()`**
  - Returns the root node of tree
- **`TreeNode[] getPathToRoot (DefaultMutableTreeNode node)`**
  - Returns a TreeNode array of all nodes from a node to the root node

Using DefaultTreeModel

- Build JTree
  - Create root node and child nodes

  DefaultMutableTreeNode root=new DefaultMutableTreeNode("…");

  (You can establish the parent/child relationships between the nodes by using the **add** method)
  - Construct a DefaultTreeModel with the root node

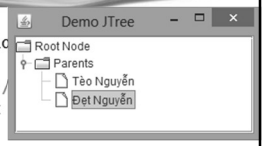
  DefaultTreeModel treeModel = new DefaultTreeModel (root);
  - Construct a JTree with the tree model

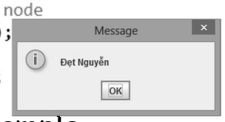
  JTree tree = new JTree (treeModel);
  - Add JTree to scrollpane

Events of JTree

- JTree can fire TreeSelectionEvent and TreeExpansionEvent, among many other events
  - Whenever a new node is selected, JTree fires a TreeSelectionEvent
    - valueChanged method
  - Whenever a node is expanded or collapsed, JTree fires a TreeExpansionEvent
    - treeCollapsed and treeExpanded methods for handling node expansion or node closing

```java
public DemoJTree() {
    setTitle("Demo JTree");setDefaultCloseOperatio
    setSize(400, 300);
    root=new DefaultMutableTreeNode("Root Node");/
    tree=new JTree(root); //create tree with root
    this.add(new JScrollPane(tree));
    //and an child node of the root
    DefaultMutableTreeNode parents=new DefaultMutableTreeNode("Parents");
    root.add(parents);//add to root
    DefaultMutableTreeNode child1=new DefaultMutableTreeNode("Tèo Nguyễn");
    DefaultMutableTreeNode child2=new DefaultMutableTreeNode("Đẹt Nguyễn");
    parents.add(child1);//add 1 node to parents
    parents.add(child2);//another ones
    tree.addTreeSelectionListener(new TreeSelectionListener() {
        @Override
        public void valueChanged(TreeSelectionEvent e) {
            //get the selection path
            TreePath selPath=e.getNewLeadSelectionPath();
            //get the Object bounded with selected node
            Object o=selPath.getLastPathComponent();
            //process o
            JOptionPane.showMessageDialog(null, o);
        }
    });
}
```

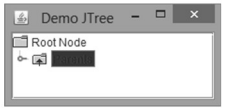


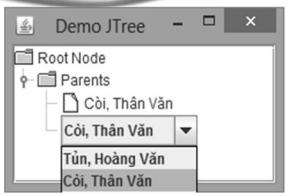


JTree event sample

JTree

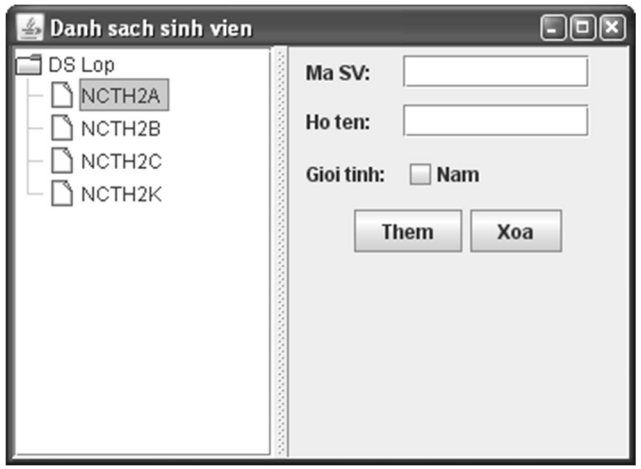
Tree Node Rendering and Editing






```java
void render_n_editor(){
    DefaultTreeCellRenderer renderer =
            (DefaultTreeCellRenderer)tree.getCellRenderer();
    renderer.setLeafIcon(MetalIconFactory.getTreeLeafIcon());
    renderer.setOpenIcon(MetalIconFactory.getTreeFolderIcon());
    renderer.setClosedIcon(MetalIconFactory.getFileChooserUpFolderIcon());
    renderer.setBackgroundSelectionColor(Color.red);

    // Customize editor
    JComboBox<String> nameComboBox = new JComboBox<>();
    nameComboBox.addItem("Tủn, Hoàng Văn");
    nameComboBox.addItem("Còi, Thân Văn");
    nameComboBox.addItem("Đẹt, Trương Thị");
    tree.setEditable(true);//allow edit
    tree.setCellEditor(new javax.swing.DefaultCellEditor(nameComboBox));
}
```


JTree Demo

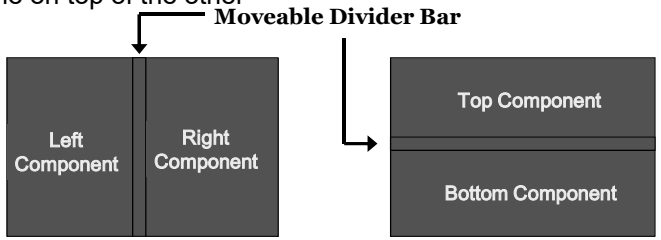


Outline

- JList
- JTable
- JTree
- JSplitPane
- JSlider
- MDI - multiple document interface

JSplitPane

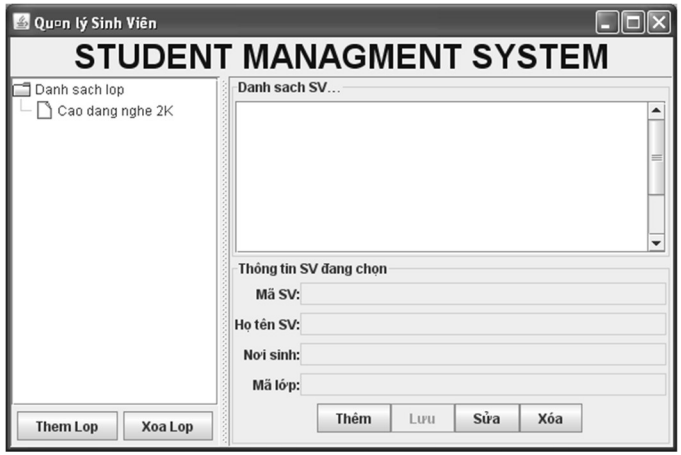
- JSplitPane is a container that displays two components separated by a moveable divider bar
- The two components can be displayed side by side, or one on top of the other



JSplitPane (contd.)

- Usage
  - The orientation of the split pane is set using the HORIZONTAL_SPLIT or VERTICAL_SPLIT constants
  - Split panes can be nested
  - Instead of adding the components of interest directly to a split pane, you often put each component into a scroll pane. You then put the scroll panes into the split pane

Example: Compound



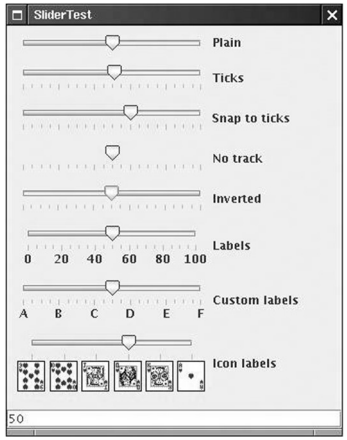
Outline

- **JList**
- **JTable**
- **JTree**
- **JSplitPane**
- **JSlider**
- **MDI - multiple document interface**

JSlider

- Purpose
  - allows the user to select a numeric value by sliding a knob within a bounded interval
- Usage
  - A JSlider can be oriented vertically or horizontally
  - A JSlider can have optional tick marks and labels
    - By default, tick marks and labels are invisible and spacing for major and minor tick marks is zero
      - To see tick marks, use method: setPaintTicks(true)
      - To display standard numeric labels at major tick mark locations, use method setPaintLabels(true)

JSlider



Event of JSlider

- When the slider is moved, a slider produces a ChangeEvent
  - implement ?
  - method ?
  - register ?

```
public void stateChanged (ChangeEvent event)
{
        JSlider slider = (JSlider) event.getSource();
        int value = slider.getValue();
        // do something with value. . .
}
```


JSlider Demo

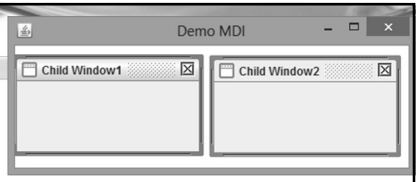
```
public class SliderDemo extends JFrame
                        implements ChangeListener
{
    JSlider jSlider;
    public SliderDemo()
    {
     super("Tick Slider");
     setDefaultCloseOperation(EXIT_ON_CLOSE);
     jSlider = new JSlider();

     // large tick marks every 25 units and small tick
     // marks every 5 units
     jSlider.setMinorTickSpacing(5);
     jSlider.setMajorTickSpacing(25);
     jSlider.setPaintTicks(true);
     jSlider.addChangeListener(this);

     add(jSlider, BorderLayout.NORTH);
     setSize(300, 200);
    }
    public void stateChanged(ChangeEvent e)
    {
     Object source = e.getSource();
     if (source.equals(jSlider))
          if (!jSlider.getValueIsAdjusting())
          System.out.println("Slider changed: "
                         + jSlider.getValue());
    }
    public static void main(String args[]) {
          new SliderDemo(). setVisible(true);
     }
}
```

Outline

- JList
- JTable
- JTree
- JSplitPane
- JSlider
- **MDI - multiple document interface**

MDI - multiple document interface

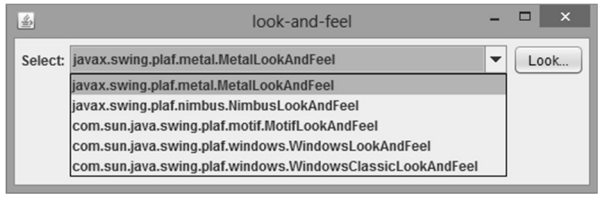
- Many applications present information in multiple windows that are all contained inside a large frame
- If you minimize the application frame, then all of its windows are hidden at the same time
- In the Windows environment, this user interface is sometimes called the multiple document interface or MDI
- In the world of Java, where you can't rely on a rich host windowing system, it makes a lot of sense to have your application manage its frames



DemoMDI.java

```
1  package gui.com.vovanhai;
2
3  import javax.swing.JDesktopPane;
4  import javax.swing.JFrame;
5  import javax.swing.JInternalFrame;
6
7  public class DemoMDI extends JFrame{
8      private JDesktopPane desktopPane;
9      public DemoMDI() throws Exception{
10         setTitle("Demo MDI");setSize(400, 400);
11         setDefaultCloseOperation(EXIT_ON_CLOSE);
12         desktopPane=new JDesktopPane();
13         this.setContentPane(desktopPane);
14         JInternalFrame iFrame1=new JInternalFrame("Child Window1", true, true);
15         JInternalFrame iFrame2=new JInternalFrame("Child Window2", true, true);
16
17         iFrame1.setSize(200, 200);iFrame1.setVisible(true);
18         iFrame2.setSize(200, 200);iFrame2.setVisible(true);
19         desktopPane.add(iFrame1);
20         desktopPane.add(iFrame2);
21     }
22     public static void main(String[] args) throws Exception {
23         new DemoMDI().setVisible(true);
24     }
25 }
```


Look-and-Feel




```
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
```

Somes installed look-n-feel:

- ✓ javax.swing.plaf.metal.MetalLookAndFeel
- ✓ javax.swing.plaf.nimbus.NimbusLookAndFeel
- ✓ com.sun.java.swing.plaf.motif.MotifLookAndFeel
- ✓ com.sun.java.swing.plaf.windows.WindowsLookAndFeel
- ✓ com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel

Mouse events

- The MouseMotionListener or MouseListener interface: respond to events when the mouse cursor is moved into or out of the area occupied by a component, or one of the mouse buttons is pressed, released, or clicked
- A class that implements MouseMotionListener or MouseListener must provide definitions some methods, each of which receives a MouseEvent as its argument

Example

```
public DefaultMutableTreeNode findUserObject (Object obj)
{
    Enumeration e = root.breadthFirstEnumeration();
    while (e.hasMoreElements())
    {
        DefaultMutableTreeNode node=(DefaultMutableTreeNode)e.nextElement();
        if (node.getUserObject().equals(obj))
                return node;
    }
    return null;
}
```


Editable ComboBox

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.text.*;
import java.util.*;
class DateDisplay extends JPanel implements ActionListener
{
    JLabel result;
    String curpat;
    DateDisplay ()
    {
        // Set the layout of the panel as BoxLayout
        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        // Declare a string array and store the various patterns in it.
        // Set the current pattern to the first element in the array

        String []patterns ={"dd MMMMM yyyy", "dd.MM.yy","MM/dd/yy", "yyyy.MM.dd G 'at' hh:mm:ss
z","yyyy.MMMMM.dd GGG hh.mm aaa"};
        curpat = patterns[0];
    // Create a combo box and initialize it with the String array that contains the pattern list
        JLabel patlab = new JLabel ("Enter the pattern or select from list : ");
```


Editable ComboBox Contd…

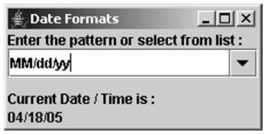
```
        JComboBox patlist = new JComboBox(patterns);
        patlist.setEditable(true);

        // Make the combo box editable
        patlist.addActionListener(this);
        JLabel reslabel = new JLabel("Current Date / Time is :", JLabel.LEADING);
        result = new JLabel(" ");
        result.setForeground(Color.red);
        // Add the label and the combo box to a panel
        JPanel patPanel = new JPanel();
        patPanel.setLayout(new
        BoxLayout(patPanel,BoxLayout.PAGE_AXIS));
        patPanel.add(patlab);
        patlist.setAlignmentX(Component.LEFT_ALIGNMENT);
        patPanel.add(patlist);
        // Add the label where the result will be displayed in the appropriate format
        JPanel resPanel = new JPanel(new GridLayout(0,1));
        resPanel.add(reslabel);
        resPanel.add(result);
        //Setting the alignment to the left
        patPanel.setAlignmentX(Component.LEFT_ALIGNMENT);
        resPanel.setAlignmentX(Component.LEFT_ALIGNMENT);
        add(patPanel);
        add(Box.createRigidArea(new Dimension(0,10)));
        add(resPanel);
        reformat();
    }
```

Editable ComboBox Contd…

```
// Check the currently selected item
    public void actionPerformed(ActionEvent e)
    {
        JComboBox cb = (JComboBox)e.getSource();
        String sel = (String)cb.getSelectedItem();
        curpat = sel;
        reformat();
    }
    // Get the current date and display it.
    // Set the date object to the appropriate format before displaying it.
    public void reformat()
    {
        Date today = new Date();
        SimpleDateFormat dateformat = new SimpleDateFormat(curpat);
        try
        {
            String dateString = dateformat.format(today);
            result.setForeground(Color.blue);
            result.setText(dateString);
        }
        catch(IllegalArgumentException e)
        {
            result.setForeground(Color.red);
            result.setText("Error " + e.getMessage());
        }
    }
```

```
public static void main ( String [] args)
    {
        JFrame frame = new JFrame("Date Formats");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent objDate = new DateDisplay();
        frame.setContentPane(objDate);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Output



Event of JTree

- When the user selects tree nodes, a tree produces a `TreeSelectionEvent`
  - implement TreeSelectionListener
  - method
    - `public void valueChanged(TreeSelectionEvent event)`

    → That method is called whenever the user selects or deselects tree nodes
  - register

Contructors of JTree

- JTree (TreeNode root)
  - construct a tree with a default tree model that displays the root
- JTree (TreeModel model)
  - constructs a tree from a tree model
- TreeNode is an interface. How do you obtain it?
  - Using the DefaultMutableTreeNode class
  - Constructing your own treenode by creating a class that implements the TreeNode interface

Methods of JTree

- **`void setEditable(boolean b)`**
  - If b is *true*, then a node on JTree can be edited
- **`void setRootVisible(boolean b)`**
  - If b is *true*, then the root node is displayed
- **`void makeVisible(TreePath path)`**
  - Expands all nodes along the path
- **`void scrollPathToVisible(TreePath path)`**
  - Expands all nodes along the path and, if the tree is contained in a scroll pane, scrolls to ensure that the last node on the path is visible
- **`Object getLastSelectedPathComponent()`**
  - Gets the node object that represents the currently selected node, or the first node if multiple nodes are selected. Returns *null* if no node is selected

JTree with Fixed Set of Nodes

- Build JTree
  - Create a root node and child nodes:

```
DefaultMutableTreeNode root=new DefaultMutableTreeNode("World");
DefaultMutableTreeNode country;
country = new DefaultMutableTreeNode("USA");
root.add(country);
country = new DefaultMutableTreeNode("Germany");
root.add(country);
```

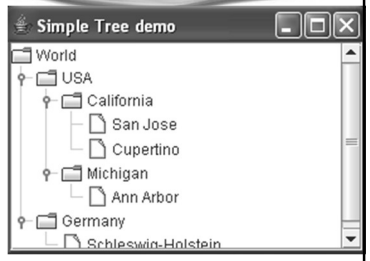
  - Pass root node in JTree's constructor:

```
JTree tree = new JTree(root);
```

  - Add JTree to scrollpane:

```
JScrollPane scrollTree = new JScrollPane(tree);
```


JTreeSimpleDemo.java




```
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
public class TreeSimpleDemo extends JFrame {
    JTree tree;
    public TreeSimpleDemo() {
        setTitle("Simple Tree demo");
        // set up tree model data
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");

        DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");
        root.add(country);
        DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");
        country.add(state);
        DefaultMutableTreeNode city = new DefaultMutableTreeNode("San Jose");
        state.add(city);
        city = new DefaultMutableTreeNode("Cupertino");
        state.add(city);
        state = new DefaultMutableTreeNode("Michigan");
        country.add(state);
```


JTreeSimpleDemo.java (cont.)

```
        city = new DefaultMutableTreeNode("Ann Arbor");
        state.add(city);
        country = new DefaultMutableTreeNode("Germany");
        root.add(country);
        state = new DefaultMutableTreeNode("Schleswig-Holstein");
        country.add(state);
        // construct tree and put it in a scroll pane
        tree = new JTree(root);
        JScrollPane scrollTree = new JScrollPane(tree);
        add(scrollTree);
        setSize(300, 200);
        setVisible(true);
    }
    public static void main(String[] args) {
        new TreeSimpleDemo();
    }
}
```


Methods of DefaultMutableTreeNode

- `int getChildCount(Object parent)`
  - returns the number of children of parent
- `TreeNode getParent()`
  - returns the parent node of the node
- `int getIndex(TreeNode item)`
  - return the index of the node
- `Object getUserObject()`
  - returns this node's user object
- `Enumeration breadthFirstEnumeration()`
- `Enumeration depthFirstEnumeration()`
  - return enumeration objects for visiting all nodes of the tree model in a particular order

Visit all nodes

- Sometimes you need to find a node in a tree by starting at the root (or any node) and visiting all children until you have found a match
  - The typical usage pattern (if beginning from root node)

```
DefaultMutableTreeNode root =
        (DefaultMutableTreeNode) tree.getModel().getRoot();
Enumeration e = root.breadthFirstEnumeration();
while (e.hasMoreElements()) {
    DefaultMutableTreeNode node =
                (DefaultMutableTreeNode) e.nextElement();
    // process node
}
```


Editing in JTree

- **`JTree`** doesn't actually store the data; it provides an organized **view** that allows the user to traverse the data
- So you edit its data from a `TreeModel`
- `TreeModel` is interface. How do you obtain a it?
  - Using the `DefaultTreeModel` (in package javax.swing.tree)
  - Constructing your own model by creating a class that implements the `TreeModel` interface

Editing in JTree with DefaultTreeModel

- Build JTree
  - Create root node and child nodes

DefaultMutableTreeNode root=new DefaultMutableTreeNode("…");

  - You can establish the parent/child relationships between the nodes by using the **add** method
  - Construct a `DefaultTreeModel` with the root node

DefaultTreeModel treeModel = new DefaultTreeModel (root);

  - Construct a `JTree` with the tree model

JTree tree = new JTree (treeModel);

  - Add JTree to scrollpane: same before

Methods of DefaultTreeModel

- **`void insertNodeInto (MutableTreeNode newChild, MutableTreeNode parent, int index)`**
  - Inserts newChild as a new child node of parent at the given index and notifies the tree model listeners
- **`void removeNodeFromParent (MutableTreeNode node)`**
  - Removes node from this model
- **`Object getRoot()`**
  - Returns the root node of tree
- **`TreeNode[] getPathToRoot (DefaultMutableTreeNode node)`**
  - Returns a TreeNode array of all nodes from a node to the root node

Constructors of JSplitPane

- **`JSplitPane()`**
  - constructs a split pane by the orientation: JSplitPane.HORIZONTAL_SPLIT
- **`JSplitPane(int direction)`**
  - constructs a split pane by specifying the orientation: JSplitPane.HORIZONTAL_SPLIT or JSplitPane.VERTICAL_SPLIT
- **`JSplitPane(int direction, Component comLeft, Component comRight)`**

Methods of JSplitPane

- **`void setLeftComponent(Component c)`**
- **`void setTopComponent(Component c)`**
  - These operations have the same effect, to set c as the first component in the split pane
- **`void setRightComponent(Component c)`**
- **`void setBottomComponent(Component c)`**
  - These operations have the same effect, to set c as the second component in the split pane

Constructors of JSlider

- **JSlider()**
  - Creates a horizontal slider with the range 0 to 100 and an initial value of 50
- **JSlider(int min, int max)**
  JSlider(int min, int max, int init_value)
  - Creates a horizontal slider with the specified minimum and maximum values. The third int argument, when present, specifies the slider's initial value
- **JSlider(int orientation)**
  JSlider(int orientation, int min, int max, int init_value)
  - Creates a slider with the specified orientation, which must be either JSlider.HORIZONTAL or JSlider.VERTICAL. The last three int arguments, when present, specify the slider's minimum, maximum, and initial values, respectively

Methods of JSlider

- **`void setMajorTickSpacing(int value)`**
- **`void setMinorTickSpacing(int value)`**
  - sets the major or minor tick spacing for this slider
- **`void setPaintTicks(boolean b)`**
  - sets whether tick marks are painted on the slider
- **`void setValue(int value)`**
- **`int getValue()`**
  - sets or gets the slider's current value
- **`void setOrientation(int orientation)`**
- **`int getOrientation()`**
  - sets or gets the orientation for the slider

Methods of JSlider (cont.)

- `setMinimum(int minimumValue)`
- `setMaximum(int maximumValue)`
  - sets the minimum or maximum value for this slider
- `int getMinimum()`
- `int getMaximum()`
  - returns the minimum or maximum value of this slider
- `void setPaintLabels(boolean b)`
- `boolean getPaintLabels()`
  - sets or gets whether labels are painted on the slider
- `boolean getValueIsAdjusting()`
  - returns *true* if the slider knob is being dragged

Outline

- **JList**
- **JTable**
- **JTree**
- **JSplitPane**
- **JSlider**
- **Key Event**
- **Mouse Event**
- **MDI - multiple document interface**

Key Event

- Key events are generated when keys on the keyboard are pressed and released
- Any component can generate these events
- To handle, write a class that implements KeyListener interface, with methods
  - keyPressed(KeyEvent event) -- any key pressed down
  - keyReleased(KeyEvent event) -- any key released
  - keyTyped(KeyEvent event) -- key for printable char released
- See KeyDemo.java

Outline

- **JList**
- **JTable**
- **JTree**
- **JSplitPane**
- **JSlider**
- **Key event**
- **Mouse event**
- **MDI - multiple document interface**

Mouse event

- A mouse event is generated when the mouse button is clicked or when the mouse moves into or out of a component's drawing area
- Java divides these events into two categories: mouse events and mouse motion events

Methods in MouseListener interface

Defined Methods	Description
mouseClicked(MouseEvent e)	Called when a mouse button is clicked on a component - that is, when the button is pressed and released
mousePressed(MouseEvent e)	Called when a mouse button is pressed on a component
mouseReleased(MouseEvent e)	Called when a mouse button is released on a component
mouseEntered(MouseEvent e)	Called when the mouse enters the area occupied by a component
mouseExited(MouseEvent e)	Called when the mouse exits the area occupied by a component

Methods in MouseMotionListener interface

- See MouseTracker.java

Defined Methods	Description
mouseMoved(MouseEvent e)	Called when a mouse button is moved on a component
mouseDragged(MouseEvent e)	Called when the user has pressed the mouse button down and moved the mouse without releasing the button

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

LẬP TRÌNH NHẬP XUẤT

GV: Nguyễn Thị Hoàng Khánh

Tài liệu tham khảo

1. Y. Daniel Liang; *Introduction to Java Programming, Comprehensive Version*; Tenth Edition; Prentice Hall; 2014; Chapter 17
2. https://docs.oracle.com/javase/tutorial/essential/io/streams.html

NỘI DUNG

- Luồng nhập xuất là gì?
- Các loại luồng.
- Phân cấp các lớp luồng.
- Qui trình điều khiển thao tác nhập xuất sử dụng luồng.
- Luồng byte.
- Luồng ký tự.
- Luồng đệm.
- Các luồng nhập xuất chuẩn.
- Luồng dữ liệu.
- Luồng đối tượng.
- Lớp File.

I/O stream

Luồng nhập xuất là gì?

- Luồng là một dòng lưu chuyển của dữ liệu, nhận thông tin từ nguồn và gửi thông tin tới đích.
- ***I/O Stream*** diễn tả cho một luồng nhập hoặc luồng xuất.
  - Luồng nhập *(input stream)*: Gắn với các thiết bị nhập như bàn phím, máy scan, file…
  - Luồng xuất *(output stream):* Gắn với các thiết bị xuất như màn hình, máy in, file…

5

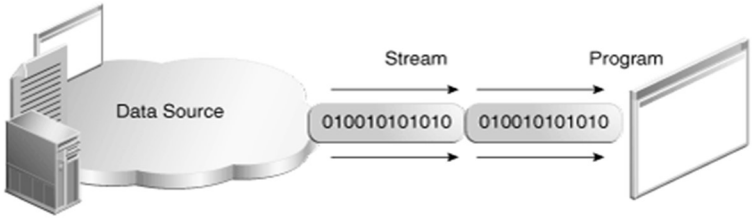
Luồng nhập xuất là gì?

- Nguồn và đích có thể là: tập tin, các thiết bị, các chương trình khác, kết nối mạng, và bộ nhớ.
- Luồng hỗ trợ nhiều loại dữ liệu khác nhau: byte, các ký tự, các kiểu dữ liệu cơ sở, các đối tượng.
- Package java.io
- Khi làm việc với luồng, chúng ta cần phải bẫy lỗi tường minh lỗi IOException bằng khối try - catch.

6

Luồng nhập

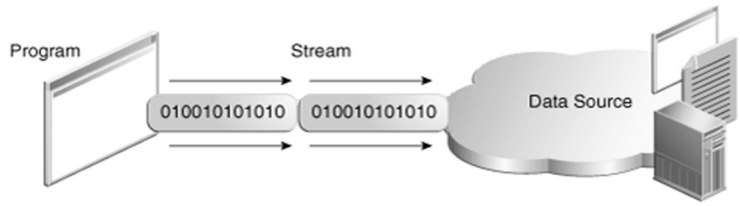
- Chương trình sử dụng ***luồng nhập*** để đọc dữ liệu từ nguồn *(mỗi unit tại một thời điểm)* đưa vào chương trình:



7

Luồng xuất

- Chương trình sử dụng ***luồng xuất*** để ghi dữ liệu xuống đích *(mỗi unit tại một thời điểm).*



8

Types of Streams

Các loại luồng

- Luồng ký tự và luồng byte *(Character và Byte Streams)*
  - Character vs. Byte
- Luồng nhập và luồng xuất *(Input và Output Streams)*.
  - Dựa trên nguồn hoặc đích.
- Luồng dữ liệu đích và luồng lọc *(Node và Filter Streams)*
  - Dữ liệu trên một luồng có thể được thao tác hoặc chuyển đổi hoặc không.

10

Luồng ký tự và luồng byte

- Luồng byte *(byte streams)*
  - Cho dữ liệu dạng nhị phân.
  - Các lớp gốc cho luồng byte:
    - InputStream
    - OutputStream
    - Cả 2 lớp là trừu tượng (abstract)
- Luồng ký tự *(character streams)*
  - Cho các ký tự Unicode.
  - Các lớp gốc cho luồng ký tự:
    - Reader
    - Writer
    - Cả 2 lớp là trừu tượng (abstract)

11

Luồng nhập và luồng xuất

- Luồng nhập *(input hoặc source streams)*
  - Có thể đọc từ những luồng này.
  - Lớp gốc của tất cả các luồng nhập:
    - InputStream
    - Reader
- Luồng xuất *(output hoặc sink (destination) streams)*
  - Có thể ghi từ những luồng này.
  - Các lớp gốc của tất cả các luồng xuất:
    - OutputStream
    - Writer

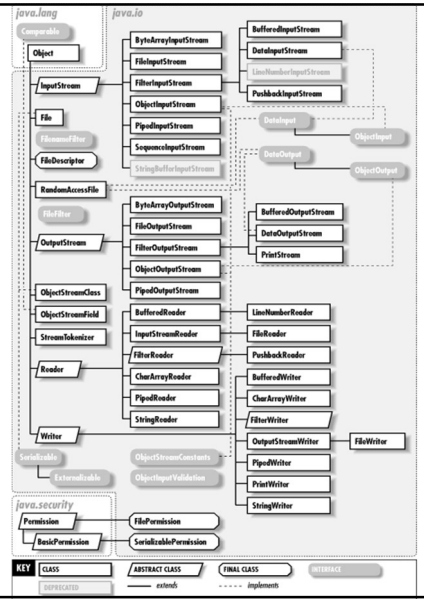
12

Luồng dữ liệu đích và luồng lọc

- Luồng dữ liệu đích *(Node streams / Data sink stream)*
  - Chứa những chức năng cơ bản cho việc đọc và ghi từ một vị trí xác định.
  - Các loại luồng node gồm: file, bộ nhớ và pipe.
- Luồng lọc *(Filter streams / Processing stream)*
  - Luồng lọc có khả năng kết nối với các luồng khác và xử lý dữ liệu "theo cách riêng" của nó.
  - FilterInputStream và FilterOutputStream là 2 lớp luồng lọc cơ bản.

13

Phân cấp lớp luồng I/O Java



14

Qui trình điều khiển thao tác nhập xuất

1. Tạo một luồng và liên kết nó với dữ liệu nguồn (hoặc dữ liệu đích).
2. Đưa "tools" chuyên dụng cho việc đọc ghi vào luồng.
3. while (vẫn còn thông tin).
4. read (write) dữ liệu kế tiếp từ (đến) luồng.
5. Đóng luồng.

15

Byte Stream

16

Luồng byte

- Chương trình sử dụng luồng byte *(byte stream)* để thực hiện nhập xuất theo đơn vị byte.
- Tất cả các lớp của luồng byte đều được dẫn xuất từ lớp InputStream và OutputStream.
- Có nhiều lớp để thao tác trên luồng byte:
  - FileInputStream
  - FileOutputStream
- Hầu hết cách thức sử dụng chúng là giống nhau, khác nhau chủ yếu là cách thức được khởi tạo.

17

Vài phương thức InputStream

- int **available**() throws IOException
  - Trả về số byte có thể đọc tiếp
- abstract int **read**() throws IOException
  - Đọc một byte từ luồng.
  - Nếu cuối luồng sẽ trả về -1
- int **read**(byte[] b) throws IOException
  - Đọc một mảng byte từ luồng và lưu vào mảng b.
- int **read** (byte[] b, int off, int len) throws IOException
  - Đọc len byte từ luồng và lưu vào mảng b, bắt đầu tại vị trí off.
- void **close**() throws IOException
  - Đóng luồng và giải phóng tất cả tài nguyên hệ thống nếu có liên kết với luồng này.
- **FileInputStream** dùng để đọc các byte từ một tập tin.

18

Vài phương thức OutputStream

- void abstract **write**(int b) throws IOException
  - Ghi byte dữ liệu b.
- void **write**(byte[] b) throws IOException
  - Ghi mảng byte b.
- void **write**(byte[] b, int off, int len)
  - Ghi len byte từ mảng byte b, bắt đầu tại vị trí off.
- void **close**() throws IOException
  - Đóng luồng xuất và giải phóng tất cả tài nguyên hệ thống liên quan đến luồng này.
- void **flush** () throws IOException
  - Đẩy dữ liệu còn trên luồng xuống đích.
- **FileOutputStream** dùng để ghi các byte xuống tập tin.

19

Ví dụ về luồng byte

```
2  import java.io.FileInputStream;
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  public class CopyBytes {
6      public static void main(String[] args) throws IOException{
7          FileInputStream in=null;
8          FileOutputStream out=null;
9          try{
0              in=new FileInputStream("Data\\src.txt");
1              out=new FileOutputStream("Data\\dest.txt");
2              int c;;
3              while((c=in.read())!=-1){
4                  out.write(c);
5              }
6          }finally{
7              if(in!=null) in.close();
8              if(out!=null) out.close();
9          }
0      }
1  }
```

Khi nào không sử dụng luồng byte?

- Luồng byte biểu diễn một loại nhập xuất ở mức thấp mà ta nên tránh.
  - Nếu dữ liệu là dữ liệu ký tự, thì phương pháp tốt nhất là sử dụng luồng ký tự.
  - Ngoài ra, còn có nhiều luồng khác thích hợp cho những kiểu dữ liệu phức tạp khác.
- Các luồng byte chỉ nên sử dụng nhập xuất cho các kiểu nhập xuất cơ bản.
- Tất cả các luồng khác đều dựa trên luồng byte.

21

Character Stream

22

Luồng ký tự

- Java hỗ trợ đọc và thao tác trên luồng đối với các ký tự unicode.
- Luồng ký tự *(character stream)*: Thực hiện các thao tác nhập xuất theo ký tự.
- Tất cả các lớp của luồng ký tự đều được dẫn xuất từ lớp Reader và Writer.
- Các lớp thao tác trên file của luồng ký tự:
  - FileReader
  - FileWriter.

23

Vài phương thức của lớp Writer

- void **write**(int c) throws IOException
  - Ghi ký tự c được biểu diễn bằng số nguyên.
- void **write**(char cbuf[]) throws IOException
  - Ghi mảng ký tự cbuf.
- void **write**(char[] cbuf, int off, int len) throws IOException
  - Ghi len ký tự trong mảng cbuf, bắt đầu tại vị trí off.
- void **write**(String str) throws IOException
  - Ghi một chuỗi ký tự str.
- void **write**(String str, int off, int len) throws IOException
  - Ghi len ký tự trong chuỗi str, bắt đầu tại vị trí off.
- void **flush**() throws IOException
  - Đẩy dữ liệu còn trên luồng xuống đích.
- void **close**() throws IOException
  - Đóng luồng.

24

Vài phương thức của lớp Reader

- int **read**() throws IOException
  - Đọc một ký tự.
  - Trả về -1 nếu cuối luồng.
- int **read**(char cbuf[]) throws IOException
  - Đọc các ký tự từ luồng và lưu vào mảng cbuf.
- int **read**(char[] cbuf, int off, int len) throws IOException
  - Đọc len ký tự từ luồng và lưu chúng vào mảng cbuf, bắt đầu tại vị trí off.
- boolean **ready**() throws IOException
  - Trả về true nếu số ký tự để đọc tiếp vẫn còn trong luồng.
- long **skip** (long *n*) throws IOException
  - Bỏ qua n ký tự *(để đọc các ký tự sau)*.
- void **close**() throws IOException
  - Đóng luồng.

25

Ví dụ về luồng ký tự

```
3  import java.io.FileReader;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  public class CopyCharacters {
7      public static void main(String[] args) throws IOException{
8          FileReader in=null;
9          FileWriter out=null;
10         try{
11             in=new FileReader("Data/src.txt");
12             out=new FileWriter("Data/dest.txt");
13             int c;
14             while((c=in.read())!=-1)
15                 out.write(c);
16         }
17         finally{
18             if(in!=null) in.close();
19             if(out!=null) out.close();
20         }
21     }
22 }
```

26

Luồng Byte và Luồng Ký Tự

- Luồng ký tự thường là "wrappers" của luồng byte.
- Luồng ký tự sử dụng luồng byte để thực hiện nhập xuất vật lý. Trong khi đó luồng ký tự xử lý chuyển đổi giữa ký tự và byte.
  - FileReader dùng FileInputStream
  - FileWriter dùng FileOutputStream
- Dùng luồng ký tự có thể thao tác được cho luồng byte.
- Có thể chuyển từ luồng byte sang luồng ký tự nhờ: InputStreamReader và OutputStreamReader.

27

Line-Oriented I/O

- Thông thường nhập xuất trên ký tự thường xảy ra là một chuỗi các ký tự hơn là các ký tự riêng lẻ.
  - Thông dụng nhất là một dòng: một chuỗi các ký tự với một tín hiệu kết thúc dòng.
  - Tín hiệu kết thúc dòng có thể là \r (carriage-return) hoặc \n (line-feed).

28

Ví dụ Line-Oriented I/O

```
3  import java.io.FileReader;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  import java.util.Scanner;
7  public class CopyLines {
8      public static void main(String[] args) throws IOException{
9          FileWriter out=null;
10         Scanner in=null;
11         try{
12             in=new Scanner(new FileReader("Data/src.txt"));
13             out=new FileWriter("Data/dest.txt");
14             String line=null;
15             while(in.hasNextLine()){
16                 line=in.nextLine();
17                 out.write(line+"\n");
18             }
19         }finally{
20             if(in!=null) in.close();
21             if(out!=null) out.close();
22         }
23     }
24 }
```

29

Buffered Stream

30

Sự cần thiết của luồng đệm

- Nếu một I/O không có bộ đệm, nghĩa là mỗi yêu cầu đọc hoặc ghi được xử lý trực tiếp trên thiết bị.
- Để giảm các chi phí trên, nền tảng Java hỗ trợ luồng nhập xuất có bộ đệm.
  - Luồng nhập có bộ đệm *(buffered input stream)* đọc dữ liệu từ một vùng nhớ được xem như một bộ đệm; chỉ ghi vào khi nào bộ đệm rỗng.
  - Luồng xuất có bộ đệm *(buffered output stream)* ghi dữ liệu tới bộ đệm; chờ cho đến khi bộ đệm đầy mới ghi tới đích.

31

Các lớp của luồng đệm

- Một chương trình có thể chuyển một luồng không bộ đệm thành luồng có bộ đệm *(buffered stream)*.
- Có 4 lớp luồng đệm dùng để "wrap" các luồng không bộ đệm:
  - BufferedInputStream và BufferedOutputStream là các luồng byte có bộ đệm.
  - BufferedReader và BufferedWriter là các luồng ký tự có bộ đệm.

32

Flushing Buffered Streams

- Vài trường hợp dữ liệu không chứa đủ bộ đệm. Vì vậy, phải dùng flush để ghi hết những gì còn lại trong bộ đệm ra.
- Một vài lớp luồng xuất có bộ đệm hỗ trợ autoflush.
  - Khi chức năng autoflush được thiết lập, cần phải thiết lập sự kiện cụ thể để bộ đệm ghi ra.
  - Ví dụ, autoflush trong đối tượng PrintWriter, bộ đệm ghi ra mỗi khi có lệnh println hoặc format.
- Muốn ghi ra tại thời điểm bất kỳ, ta dùng phương thức flush().

33

Ví dụ luồng đệm

```
 4 import java.io.BufferedOutputStream;
 5 import java.io.FileInputStream;
 6 import java.io.FileOutputStream;
 7 import java.io.IOException;
 8 import java.io.PrintWriter;
 9 import java.util.Scanner;
10 public class DemoBufferedStream {
11     private final String[] dsTenSV={"An", "Lan","Hoa","Hoàng","Nga","Tâm"};
12     private final float[] DSDiemGK={7,8.5f,9,10,4.5f,6};
13     private final float[] DSDiemCK={5,5.5f,6.5f,10,6,4.5f};
14     public void writeTo (String fileName) throws IOException{
15         PrintWriter out=null;
16         try{
17             out=new PrintWriter(new BufferedOutputStream(
18                                 new FileOutputStream(fileName)), true);
19             for (int i = 0; i < dsTenSV.length; i++) {
20                 out.println(String.format("%-10s %10s %10s",
21                         dsTenSV[i]+";",DSDiemGK[i]+";", DSDiemCK[i]));
22             }
23         }finally{
24             if(out!=null) out.close();
25         }
26     }
```

Ví dụ luồng đệm

```
29 public void readFrom(String fileName) throws IOException{
30     Scanner in=null;
31     try{
32         in=new Scanner(new BufferedInputStream(
33                         new FileInputStream(fileName)));
34         String line=null;
35         while(in.hasNextLine()){
36             line=in.nextLine();
37             String[] cols=line.split(";");
38             float diemGK=Float.parseFloat(cols[1]);
39             float diemCK=Float.parseFloat(cols[2]);
40             float diemTB=(diemGK+diemCK)/2;
41             System.out.println(String.format("%-25s %10s",
42                                              line+";",diemTB));
43         }
44     }finally{
45         if(in!=null) in.close();
46     }
47 }
```

35

Standard Streams

36

Các luồng chuẩn trên nền Java

- Có 3 luồng chuẩn:
  - Luồng nhập chuẩn - System.in
  - Luồng xuất chuẩn - System.out
  - Luồng xuất lỗi chuẩn - System.err
- System.out, System.err được định nghĩa như các đối tượng PrintStream.

37

Ví dụ luồng chuẩn

```
3  import java.util.Scanner;
4  public class DemoStandardStream {
5      public static void main(String[] args) {
6          int[] a = { 10, 4, 9 ,15 };
7          Scanner sc=null;
8          try {
9              sc = new Scanner(System.in);
10             System.out.println("Nhập vị trí: ");
11             int i = sc.nextInt();
12             System.out.println(
13                     String.format("Phần tử tại vi tri %d là: %s", i, a[i]));
14         }catch (ArrayIndexOutOfBoundsException ex) {
15             System.err.println("Lỗi:"+ex.toString());
16         }
17         catch (Exception ex) {
18             System.err.println("Lỗi:"+ex.getStackTrace()[2]);
19         }
20         finally{
21             if(sc!=null) sc.close();
22         }
23     }
24 }
```

Data Streams

39

Luồng dữ liệu

- Luồng dữ liệu *(data streams)* hỗ trợ nhập xuất nhị phân trên các kiểu dữ liệu cơ sở *(boolean, char, byte, short, int, long, float, và double)* và String.
- Tất cả các luồng dữ liệu hiện thực từ giao diện DataInput hoặc từ DataOutput.
- Hầu hết việc nhập xuất trên luồng dữ liệu thì dùng lớp DataInputStream và DataOutputStream.
- Những dữ liệu được ghi bởi DataOutputStream sẽ đọc được bởi DataInputStream

40

Vài phương thức DataOutputStream

Constructor: DataOutputStream(OutputStream out)

- public final void writeByte(int b) throws IOException
- public final void writeShort(int s) throws IOException
- public final void writeChar(int c) throws IOException
- public final void writeInt(int i) throws IOException
- public final void writeLong(long l) throws IOException
- public final void writeUTF(String s) throws IOException
- …

41

Ví dụ DataOutputStream

```
3  import java.io.BufferedInputStream;
4  import java.io.BufferedOutputStream;
5  import java.io.DataInputStream;
6  import java.io.DataOutputStream;
7  import java.io.FileInputStream;
8  import java.io.FileOutputStream;
9  import java.io.IOException;
10 public class DemoDataStream {
11     private final String[] DSTenSV={"An", "Lan","Hoa","Hoàng","Nga","Tâm"};
12     private final float[] DSDiemGK={7,8.5f,9,10,4.5f,6};
13     private final float[] DSDiemCK={5,5.5f,6.5f,10,6,4.5f};
14     public void writeTo (String fileName) throws IOException{
15         DataOutputStream out=null;
16         try{
17             out=new DataOutputStream(new BufferedOutputStream(
18                             new FileOutputStream(fileName)));
19             for (int i = 0; i < DSTenSV.length; i++) {
20                 out.writeUTF(DSTenSV[i]);
21                 out.writeFloat(DSDiemGK[i]);
22                 out.writeFloat(DSDiemCK[i]);
23             }
24         }finally{ if(out!=null) out.close(); }
25     }
```

Vài phương thức DataInputStream

Constructor: DataInputStream(InputStream in)

- public final byte readByte() throws IOException
- public final char readChar() throws IOException
- public final short readShort() throws IOException
- public final int readInt() throws IOException
- public final long readLong() throws IOException
- public final String readUTF() throws IOException
- …

43

Ví dụ DataInputStream

```
28 public void readFrom(String fileName) throws IOException{
29     DataInputStream in=null;
30     try{
31         in=new DataInputStream(new BufferedInputStream(
32                         new FileInputStream(fileName)));
33         while(in.available()!=0){
34             String tenSV=in.readUTF();
35             float diemGK=in.readFloat();
36             float diemCK=in.readFloat();
37             float diemTB=(diemGK+diemCK)/2;
38             System.out.println(String.format("%-10s %7s %7s %7s",
39                                     tenSV,diemGK,diemCK,diemTB));
40         }
41     }finally{
42         if(in!=null) in.close();
43     }
44 }
```

44

Object Streams

Tuần tự hóa dữ liệu

- Tính bền vững *(persistence)* là khả năng một đối tượng duy trì sự tồn tại độc lập sau thời gian sống của chương trình tạo ra nó.
- Java cung cấp cơ chế được gọi là tuần tự hóa đối tượng *(Object Serialization)* để tạo đối tượng bền vững.
- Khi một đối tượng được tuần tự hóa, nó sẽ được chuyển thành tuần tự các byte dạng thô, biểu diễn đối tượng.

Luồng đối tượng

- **Luồng đối tượng** *(Object Streams)* hỗ trợ việc đọc, ghi các đối tượng.
- Nếu đối tượng hiện thực giao diện Serializable thì ta có thể sử dụng luồng đối tượng để đọc, ghi đối tượng đó.
- Hai lớp hỗ trợ luồng đối tượng:
  - ObjectInputStream
  - ObjectOutputStream
- Hai lớp này tương ứng hiện thực các giao diện:
  - ObjectInput
  - ObjectOutput

Luồng đối tượng

- Bất kỳ đối tượng nào mà ta muốn tuần tự hóa *(serialize)* thì bắt buộc phải hiện thực giao diện Serializable.
- Để tuần tự hóa một đối tượng, gọi phương thức writeObject của lớp ObjectOutputStream.
- Để khôi phục lại đối tượng đã được tuần tự hóa trước đó (*deserialize)*, gọi phương thức readObject của lớp ObjectInputStream.
- Các đối tượng được tuần tự hóa có thể được ghi vào file, truyền qua mạng hoặc có thể chuyển sang các luồng khác.

Object serialization demo

ObjectSerialization.java

```
1  package myio;
2
3  import java.io.FileInputStream;
4  import java.io.FileOutputStream;
5  import java.io.ObjectInputStream;
6  import java.io.ObjectOutputStream;
7
8  public class ObjectSerialization {
9      public void serialize2file(Object obj, String dataFile)throws Exception{
10         FileOutputStream fos=new FileOutputStream(dataFile);
11         ObjectOutputStream oos=new ObjectOutputStream(fos);
12         oos.writeObject(obj);
13         oos.close();
14     }
15     public Object deserialize(String datafile)throws Exception{
16         Object obj=null;
17         FileInputStream fis=new FileInputStream(datafile);
18         ObjectInputStream ois=new ObjectInputStream(fis);
19         obj=ois.readObject();
20         ois.close();
21         return obj;
22     }
23 }
```

49

Lớp File

- Lớp File dùng cho việc thao tác trên file và thư mục.
- Một số phương thức của lớp File:
  - boolean exists(); // kiểm tra sự tồn tại của file
  - boolean isDirectory(); // kiểm tra xem file có phải là thư mục
  - String getParent(); // lấy thư mục cha
  - long length(); // lấy kích cỡ file (byte)
  - long lastModified(); // lấy ngày sửa file gần nhất
  - String[] list(); // lấy nội dung của thư mục
  - boolean createNewFile(); //tạo file

50

Ví dụ về File

1. Nhập một đường dẫn. Kiểm tra đường dẫn vừa nhập vào là file hay là thư mục.
   - Nếu là file, chép nội dung file vừa nhập sang file có tên là backup.dat.
   - Nếu là thư mục, hãy liệt kê nội dung của thư mục đó.
2. Giả sử có 2 folder chứa cùng số tập tin. Folder thứ nhất chứa câu hỏi trắc nghiệm, folder thứ 2 chứa đáp án của câu trắc nghiệm tương ứng cho folder thứ nhất *(mỗi câu là 1 file)*. Hãy trộn câu hỏi trắc nghiệm với đáp án tương ứng, và lưu vào file "TracNghiem_coDapAn.txt".

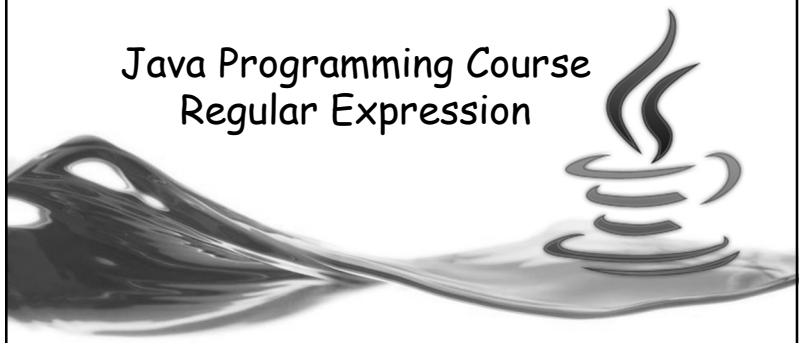
51

TỔNG KẾT

- Luồng nhập xuất là gì?
- Các loại luồng.
- Phân cấp các lớp luồng.
- Qui trình điều khiển thao tác nhập xuất sử dụng luồng.
- Luồng byte.
- Luồng ký tự.
- Luồng đệm.
- Các luồng nhập xuất chuẩn.
- Luồng dữ liệu.
- Luồng đối tượng.
- Lớp File.

52

Java Programming Course
Regular Expression



Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Regular Expression (Biểu thức chính quy)

- Một biểu thức chính quy (Regular expressions) định nghĩa một khuôn mẫu (pattern) tìm kiếm chuỗi
- Nó có thể được sử dụng tìm kiếm, sửa đổi, và thao tác trên văn bản. Khuôn mẫu được định nghĩa bởi biểu thức chính quy có thể khớp một hoặc một vài lần, hoặc không khớp với một văn bản cho trước.

  ΘViết tắt của biểu thức chính quy là regex

Regular Expression (Biểu thức chính quy)

- Biểu thức chính quy được sử dụng đặc biệt nhiều trong:
  * Phân tích cú pháp
  * Xác nhận tính hợp lệ của dữ liệu
  * Xử lý chuỗi
  * Tách dữ liệu và tạo báo cáo

Hỗ trợ các ngôn ngữ

- Biểu thức chính quy (Regular expression) được hỗ trợ bởi hầu hết các ngôn ngữ lập trình, ví dụ, Java,C#, Perl, Groovy, vv
- Tuy nhiên mỗi ngôn ngữ hỗ trợ biểu thức thông thường hơi khác nhau.

Các biểu thức chính quy trong JDK 1.4

- Toàn bộ sự hỗ trợ biểu thức chính quy được chứa trong gói java.util.regex và được tạo bởi 2 class chính sau
  - **java.util.regex.Pattern // Mẫu**
  - **java.util.regex.Matcher //so khớp**
- Một sự thi hành đặc trưng của việc tìm kiếm và/hoặc thao tác văn bản sử dụng gói java.util.regex được chia thành 3 bước.
  - Biên dịch biểu thức chính quy thành một instance của Pattern
  - Sử dụng object Pattern để tạo một object Matcher
  - Sử dụng object Matcher để tìm kiếm và/hoặc thao tác dãy kí tự

Tạo khuôn mẫu

- **static Pattern compile(String regex)**

  Biên dịch biểu thức chính quy được cung cấp regex vào trong một Pattern.
- **static Pattern compile(String regex, int flags)**

  Biên dịch biểu thức chính quy được cung cấp regex vào trong một khuôn mẫu với các cờ được cung cấp, ở đó các cờ là một bit mặt nạ của các cờ hành xử.

Quy tắc viết biểu thức chính quy

No	Regular Expression	Mô tả
1	.	Khớp (match) với bất kỳ ký tự nào
2	^regex	Biểu thức chính quy phải khớp tại điểm bắt đầu
3	regex$	Biểu thức chính quy phải khớp ở cuối dòng.
4	[abc]	Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c.
5	[abc][vz]	Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c theo sau là v hay z.
6	[^abc]	Khi dấu ^ xuất hiện như là nhân vật đầu tiên trong dấu ngoặc vuông, nó phủ nhận mô hình. Điều này có thể khớp với bất kỳ ký tự nào ngoại trừ a hoặc b hoặc c.
7	[a-d1-7]	phù hợp với một chuỗi giữa a và điểm d và con số từ 1 đến 7.
8	X|Z	Tìm X hoặc Z.
9	XZ	Tìm X và theo sau là Z.
10	$	Kiểm tra kết thúc dòng.

Quy tắc viết biểu thức chính quy

No	Regular Expression	Mô tả
11	\d	Số bất kỳ, viết ngắn gọn cho [0-9]
12	\D	Ký tự không phải là số, viết ngắn gọn cho [^0-9]
13	\s	Ký tự khoảng trắng, viết ngắn gọn cho [\t\n\xob\r\f]
14	\S	Ký tự không phải khoản trắng, viết ngắn gọn cho [^\s]
15	\w	Ký tự chữ, viết ngắn gọn cho [a-zA-Z_0-9]
16	\W	Ký tự không phải chữ, viết ngắn gọn cho [^\w]
17	\S+	Một số ký tự không phải khoảng trắng (Một hoặc nhiều)
18	\b	Ký tự thuộc a-z hoặc A-Z hoặc 0-9 hoặc _, viết ngắn gọn cho [a-zA-Z0-9_].

Quy tắc viết biểu thức chính quy

No	Regular Expression	Mô tả
19	*	Xuất hiện 0 hoặc nhiều lần, viết ngắn gọn cho {0,}
20	+	Xuất hiện 1 hoặc nhiều lần, viết ngắn gọn cho {1,}
21	?	Xuất hiện 0 hoặc 1 lần, ? viết ngắn gọn cho {0,1}.
22	{X}	Xuất hiện X lần, {}
23	{X,Y}	Xuất hiện trong khoảng X tới Y lần.
24	*?	* có nghĩa là xuất hiện 0 hoặc nhiều lần, thêm ? phía sau nghĩa là tìm kiếm khớp nhỏ nhất.

Boundary matchers

- These patterns match the *empty string* if at the specified position:

^	Beginning of a line
$	End of a line
\b	Word boundary
\B	Non-word boundary
\A	Beginning of the input
\G	End of the previous match
\Z	End of the input but for the final terminator, if any
\z	End of the input

Các ký tự đặc biệt trong Java Regex

- **\ . [{ (* + ? ^ $ |**
- Những ký tự liệt kê ở trên là các ký tự đặc biệt. Trong Java Regex muốn nó hiểu các ký tự đó theo cách thông thường cần thêm dấu **** ở phía trước

```
1  // Mẫu regex mô tả một ký tự bất kỳ.
2  String regex = ".";
3
4  // Mẫu regex mô tả  ký tự dấu chấm.
5  String regex = "\\.";
```


Tạo các Matcher

- Sau khi chúng ta có một Pattern được biên dịch, gọi method *matcher(charsequence)* trên nó để tạo một Matcher.
- Thông thường ta truyền các String vào method matcher:

```
Pattern pat=Pattern.compile("cats.*dogs");
Matcher matcher=pat.matcher("cats and dogs");
```

Tìm kiếm: Tìm kiếm chính xác

- **boolean matches()**

  Cố gắng tìm kiếm dãy đầu vào dựa trên khuôn mẫu. Method này chỉ thành công nếu toàn bộ dãy kí tự đầu vào là giống.
- **boolean matches(String regex, CharSequence input)**

  Biên dịch biểu thức chính quy được cung cấp và thử tìm kiếm sự phù hợp với dữ liệu vào.

Tìm kiếm một phần

- **boolean lookingAt()**

  Thử tìm kiếm chuỗi đầu vào, bắt đầu từ đầu dựa trên khuôn mẫu. Giống như method matches(), method này luôn bắt đầu từ đầu của chuỗi đầu vào; và không giông như method đó, nó không đòi hỏi toàn bộ chuỗi đầu vào phải giống.
- **boolean find()**

  Thử tìm chuỗi tiếp theo của chuỗi đầu vào phù hợp với khuôn mẫu. Method này bắt đầu từ đầu chuỗi đầu vào, hoặc tiếp nối ở vị trí tìm kiếm tiếp theo sau một kết quả tìm kiếm thành công.
  find() đặc biệt hữu dụng khi bạn quan tâm đến tất cả các chuỗi con trong chuỗi đầu vào phù hợp với khuôn mẫu được cung cấp

A example

```
String input="a";
String patternStr=".";//Khớp với 1 ký tự bất kỳ
Pattern pattern = Pattern.compile(patternStr);
Matcher macth= pattern.matcher(input);
Boolean macthFound= macth.matches();
System.out.println("_Match " + macthFound);
```

True

A example

```
String input ="abc";
String patternStr=".";
Pattern pattern = Pattern.compile(patternStr);
Matcher matcher = pattern.matcher(input);
Boolean matcherFound=matcher.matches();
System.out.println("--Match : " + matcherFound);
```

False:
Rõ ràng, chuỗi 3 ký tự sao khớp với 1 ký tự bất kỳ?

A example

```
String input ="abc";
String patternStr =".*";
//Khớp với bất kỳ ký tự nào 0 hoặc nhiều lần

Pattern pattern =Pattern.compile(patternStr);
Matcher matcher = pattern.matcher(input);
Boolean matcherFound= matcher.matches();
System.out.println("--Match : " + matcherFound);
```

True

Example

- String input ="abc";
- String patternStr ="^a.*"

// Bắt đầu bởi a

// Sau đó là ký tự bất kỳ, xuất hiện 1 hoặc nhiều lần.

True

Example

- String input ="onnp";
- String patternStr = ".n{1,3}p$";

// Ký tự bất kỳ xuất hiện 1 lần: .

// tiếp theo là n, xuất hiện 1 hoặc tối đa 3 lần.

// Kết thúc bởi p: p$

True

Example

- String input ="2bkbv";
- String patternStr = ".+[abc][zv].*";

// Bắt đầu là ký tự bất kỳ, 1 hoặc nhiểu lần

// Tiếp theo a hoặc b, hoặc c: [abc]

// Tiếp theo z hoặc v: [zv]

// Tiếp theo bất kỳ

True

Patterns Flags

- Trong gói java.util.regex, việc tìm chuỗi theo mặc định là phân biệt chữ hoa-chữ thường và coi mỗi kí tự theo mã ASCII chứ không phải mã Unicode.
- Để thay đổi hành xử mặc định này, bạn có thể cung cấp các cờ cho method compile().
- Tất cả các cờ là các thành viên static int của Pattern.
- Để kết hợp các hành xử, bạn có thể thực hiện OR các cờ với biểu thức "|".

Patterns Flags

Field	Description
static Pattern.CASE_INSENSITIVE	Enables case-insensitive matching of patterns
static Pattern.COMMENTS	Enables the inclusion of whitespace and comments in pattern
static Pattern.DOTALL	Enables the expression containing "." to match any characters
static Pattern.MULTILINE	Enables multiline mode where the expressions ^ or $ match just after or before a line terminator or end of input sequence

Chương trình đầu tiên

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexTest {

    public static void main(String[] args) {
        String input = "Abc\ndef";
        String patternStr = "abc$";

        Pattern pattern = Pattern.compile(patternStr, Pattern.MULTILINE
                | Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(input);
        boolean matcherFound = matcher.find();
        System.out.println(matcherFound);
    }

}
```


A example

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegexTest {
    public static void main(String args[]) {
        String pattern = "[a-z]+";
        String text = "Now is the time";
        Pattern p = Pattern.compile(pattern);
        Matcher m = p.matcher(text);
        while (m.find()) {
            System.out.print(text.substring(m.start(), m.end()) + "*");
        }
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> RegexTest [Java Application] C:\jdk1.7.0_05\bin\javaw.exe (Feb 26, 2013 2:48:07 PM)

ow*is*the*time*

Additional methods

- If m is a matcher, then
  - m.replaceFirst(***replacement***) returns a new String where the first substring matched by the pattern has been replaced by ***replacement***
  - m.replaceAll(***replacement***) returns a new String where every substring matched by the pattern has been replaced by ***replacement***
  - m.find(***startIndex***) looks for the next pattern match, starting at the specified index
  - m.reset() resets this matcher
  - m.reset(***newText***) resets this matcher and gives it new text to examine (which may be a String, StringBuffer, or CharBuffer)

Example of Boundary Matchers

```
public void testBoundaryMatchers()
{
    String[] regex = { "^dog", "dog$"};
    String input = "Love melove my dog";
    System.out.println("Input: " + input);

    for (int i = 0; i < regex.length; i++)
    {
        boolean found = false;
        Pattern pattern = Pattern.compile(regex[i]);
        Matcher matcher = pattern.matcher(input);
        System.out.println("\nPattern: " + regex[i]);

        while (matcher.find())
        {
            int s = matcher.start();
            int e = matcher.end();
            System.out.println("found \"" + input.substring(s, e) + "\"");
            System.out.println("start " + s + " end " + e);
            found = true;
        }

        if (!found)
        {
            System.out.println("No match found!");
        }
    }
}
```

Output

```
Input: Love me love my dog

Pattern: ^dog
No match found!

Pattern: dog$
found "dog"
start 16 end 19
```


Greedy quantifiers

- Assume *X* represents some pattern

X?	optional, *X* occurs zero or one time
X*	*X* occurs zero or more times
X+	*X* occurs one or more times
X{n}	*X* occurs exactly *n* times
X{n,}	*X* occurs *n* or more times
X{n,m}	*X* occurs at least *n* but not more than *m* times
Note that these are all *postfix* operators, that is, they come *after* the operand	

Examples of quantifiers

- Social Security numbers of the form 123-45-6789
  - [0-9]{3}-[0-9]{2}-[0-9]{4}
- The name of an IP address
  - ([0-9*]{1,3}(.)){1,3}[0-9*]{0,3}

Types of quantifiers

- A greedy quantifier [longest match first] (default) will match as much as it can, and back off if it needs to
- A reluctant quantifier [shortest match first] will match as little as possible, then take more if it needs to
  - You make a quantifier reluctant by appending a ?:
    X?? X? X+? X{n}? X{n,}? X{n,m}?*
- A possessive quantifier [longest match and never backtrack] will match as much as it can, and never let go
  - You make a quantifier possessive by appending a +:
    X?+ X+ X++ X{n}+ X{n,}+ X{n,m}+*

Quantifiers examples

succeed

Suppose your text is succeed

- Using the pattern suc*ce{2}d (c* is greedy):
  - The c* will first match cc, but then ce{2}d won't match
  - The c* then "backs off" and matches only a single c, allowing the rest of the pattern (ce{2}d) to succeed
- Using the pattern suc*?ce{2}d (c*? is reluctant):
  - The c*? will first match zero characters (the null string), but then ce{2}d won't match
  - The c*? then extends and matches the first c, allowing the rest of the pattern (ce{2}d) to succeed
- Using the pattern suc*+ce{2}d (c*+ is possessive):
  - The c*+ will match the cc, and will not back off, so ce{2}d never matches and the pattern match fails.

Example of Quantifiers

```
public void testQuantifiers()
{
    String[] regex = { "a{1}b{2}", "a{1}?b{2}?", "a{1}+b{2}+"};
    String input = "HabbGaHbJ";

    for (int i = 0; i < regex.length; i++)
    {
        Pattern pattern = Pattern.compile(regex[i]);
        Matcher matcher = pattern.matcher(input);

        System.out.println("\nPattern: " + regex[i] );
        while (matcher.find())
        {
            int s = matcher.start();
            int e = matcher.end();
            System.out.println("text \"" + input.substring(s, e) + "\"");
            System.out.println("start " + s + " end " + e);
        }
    }
}
```

Output

```
Pattern: a{1}b{2}
text "abb"
start 1 end 4

Pattern: a{1}?b{2}?
text "abb"
start 1 end 4

Pattern: a{1}+b{2}+
text "abb"
start 1 end 4
```


Capturing Groups - 1

- In regular expressions, parentheses are used for grouping, but they also capture (keep for later use) anything matched by that part of the pattern
  - Example: ([a-zA-Z]*)([0-9]*) matches any number of letters followed by any number of digits
  - If the match succeeds, \1 holds the matched letters and \2 holds the matched digits
  - In addition, \0 holds everything matched by the entire pattern
- Capturing groups are numbered by counting their *opening parentheses* from left to right:
  - ((A) (B (C)))
    1 2 3 4
    \0 = \1 = ((A)(B(C))), \2 = (A), \3 = (B(C)), \4 = (C)
- Example: ([a-zA-Z])\1 will match a double letter, such as letter

Capturing Groups - 2

- If m is a matcher that has just performed a successful match, then
  - m.group(***n***) returns the String matched by capturing group ***n***
    - This could be an empty string
    - This will be null if the pattern as a whole matched but this particular group didn't match anything
  - m.group() returns the String matched by the entire pattern (same as m.group(0))
    - This could be an empty string
- If m didn't match (or wasn't tried), then these methods will throw an IllegalStateException

Example of Capturing Groups

```
public void testCaptureGroup()
{
    String[] regex = { "[abb]", "(abb)" };
    String input = "HabbGaHbJ";

    for (int i = 0; i < regex.length; i++)
    {
        Pattern pattern = Pattern.compile(regex[i]);
        Matcher matcher = pattern.matcher(input);

        System.out.println("\nPattern: " + regex[i]);
        while (matcher.find())
        {
            int s = matcher.start();
            int e = matcher.end();
            System.out.println("text \"" + input.substring(s, e) + "\"");
            System.out.println("start " + s + " end " + e);
        }
    }
}
```

Output

```
Pattern: [abb]
text "a"
start 1 end 2
text "b"
start 2 end 3
text "b"
start 3 end 4
text "a"
start 5 end 6
text "b"
start 7 end 8

Pattern: (abb)
text "abb"
start 1 end 4
```


Example of Numbering Capture Group

```
public void testCaptureGroupNumber()
{
    String regex = "((a)(bb))" ;
    String input = "HabbGaHbJbb";

    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(input);
    System.out.println("\nPattern: " + regex);

    while (matcher.find())
    {
        for (int g = 1; g <= matcher.groupCount(); g++)
        {
            int s = matcher.start(g);
            int e = matcher.end(g);
            System.out.println("\nGroup " + g + ": \"" + matcher.group(g) + "\"");
            System.out.println("find text \"" + input.substring(s, e) + "\"");
            System.out.println("start " + s + " end " + e);
        }
    }
}
```

Output

```
Pattern: ((a)(bb))

Group 1: "abb"
find text "abb"
start 1 end 4

Group 2: "a"
find text "a"
start 1 end 2

Group 3: "bb"
find text "bb"
start 2 end 4
```


Example use of capturing groups

- Suppose word holds a word in English
- Also suppose we want to move all the consonants at the beginning of word (if any) to the end of the word (so string becomes ingstr)

```
public static void main(String[] args) throws Exception {
    String word="mfsdwsmove all the consonants";
    Pattern p = Pattern.compile("([^aeiou]*)(.*)");
    Matcher m = p.matcher(word);
    if (m.matches()) {
        System.out.println(m.group(2) +"-"+ m.group(1));
    }
}
```

- Note the use of (.*) to indicate "all the rest of the characters"

Double backslashes

- Backslashes have a special meaning in regular expressions; for example, \b means a word boundary
- Backslashes have a special meaning in Java; for example, \b means the backspace character
- Java syntax rules apply first!
  - If you write "\b[a-z]+\b" you get a string with backspace characters in it--this is *not* what you want!
  - Remember, you can quote a backslash with another backslash, so "\\b[a-z]+\\b" gives the correct string
- Note: if you *read in* a String from somewhere, this does not apply--you get whatever characters are actually there

Escaping metacharacters

- A lot of special characters--parentheses, brackets, braces, stars, plus signs, etc.--are used in defining regular expressions; these are called metacharacters
- Suppose you want to search for the character sequence a* (an a followed by a star)
  - "a*"; doesn't work; that means "zero or more as"
  - "a*"; doesn't work; since a star doesn't *need* to be escaped (in Java String constants), Java just ignores the \
  - "a*" *does* work; it's the three-character string a, \, *
- Just to make things even more difficult, it's *illegal* to escape a *non*-metacharacter in a regular expression

Spaces

- There is only one thing to be said about spaces (blanks) in regular expressions, but it's important:
  - ***Spaces are significant!***
- A space stands for a *space*--when you put a space in a pattern, that means to match a space in the text string
- It's a *really bad idea* to put spaces in a regular expression just to make it look better
- Ex:
  - Pattern.compile("a b+").matcher("abb"). matches()

  ⇨ return false.

Additions to the String class

- All of the following are public
  - `public boolean matches(String regex)`
  - `public String replaceFirst(String regex, String replacement)`
  - `public String replaceAll(String regex, String replacement)`
  - `public String[ ] split(String regex)`
  - `public String[ ] split(String regex, int limit)`
-
- If the limit n is greater than zero then the pattern will be applied at most n - 1 times, the array's length will be no greater than n, and the array's last entry will contain all input beyond the last matched delimiter.
- If n is non-positive then the pattern will be applied as many times as possible

Some example s of regular expression

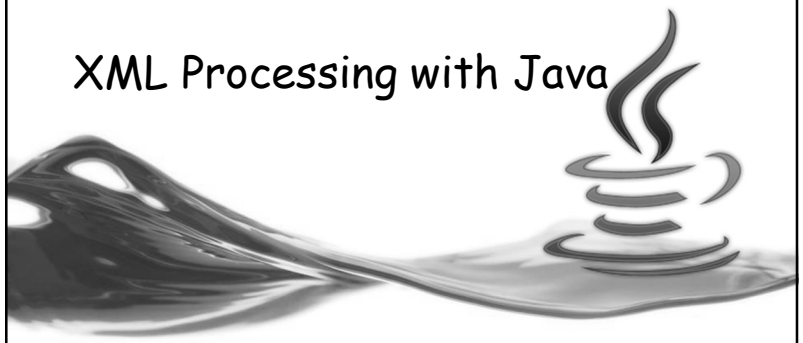
- Write a regular expression for each of the following sets of binary strings
  - all binary strings : (0|1)*
  - all binary strings except empty string : (0|1)(0|1)*
  - ends with 00 : (0|1)*00
  - contains at least three 1s : (0|1)*1(0|1)*1(0|1)*1(0|1)*
  - contains the substring 110 : (0|1)*110(0|1)*
- Regular expression to describe all dates of the form ***dd/mm/yyyy***
  - (0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[012])/((19|20)\\d\\d)

Thinking in regular expressions

- Regular expressions are *not* easy to use at first
  - It's a bunch of punctuation, not words
  - The individual pieces are not hard, but it takes practice to learn to put them together correctly
  - Regular expressions form a miniature programming language
    - It's a different kind of programming language than Java, and requires you to learn new thought patterns
  - In Java you can't just *use* a regular expression; you have to first create Patterns and Matchers
  - Java's syntax for String constants doesn't help, either
- Despite all this, regular expressions bring so much power and convenience to String manipulation that they are well worth the effort of learning

Java Programming Course

XML Processing with Java



Session objectives

Introduction

Parsing XML document

- Using SAX
- Using DOM

XML transformation

XML Introduction

- Evolution of markup languages:
  - GML(***Generalized Markup Language***)
  - SGML (***Standard Generalized Markup Language***)
  - HTML ***(Hyper Text Markup Language)***
- **eXtensible Markup Language** (**XML**) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable
- XML was designed to describe data, with focus on what data is

XML Features

- XML is a markup language much like HTML
- XML was designed to describe data
- XML tags are not predefined. You must define your own tags
- XML uses a Document Type Definition (DTD) or an XML Schema to describe the data
- XML with a DTD or XML Schema is designed to be self-descriptive
- A document consists of one outermost element, called root element that contains all the other elements, plus some optional administrative information at the top, known as XML declaration

Sample XML document

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <student id="2464353">
3      <fullname>Vo Van Hai</fullname>
4      <address>
5          <no>12</no>
6          <street>Nguyen Van Bao</street>
7          <district>Go Vap</district>
8      </address>
9      <email>vovanhaiQN@gmail.com</email>
10 </student>
```


Quick XML review

- Syntax
  - Every XML document has a preamble
    - <?xml version="1.0" ?>
  - An XML document may or may not have a DTD (Document Type Definition) or Schema
    - <!DOCTYPE catalog>
  - Every element has a start and end tag, with optional attributes
    - <catalog version="1.0"> … </catalog>
  - If an element does not contain any data (or elements) nested within, the closing tag can be merged with the start tag like so:
    - <catalog version="1.0"/>

Quick XML review

- Syntax cont.
  - Elements must be properly nested
  - The outermost element is called the root element
  - An XML document that follows the basic syntax rules is called well-formed
  - XML documents do not always require a DTD or Schema, but they must be well-formed

Well-formed and valid XML document

- A "**Well-formed**" XML document has correct XML syntax
  - XML documents must have a root element
  - XML elements must have a closing tag
  - XML tags are case sensitive
  - XML elements must be properly nested
  - XML attribute values must be quoted
- An XML document that is well-formed and conforms to a DTD or Schema is called **valid**

Benefits of XML

- Data independence: separates the content from its presentation
- Easier to parse: frameworks for data exchange
- Reducing server load: using DOM to manipulate the data
- Easier to create: it is text-based
- Web site content: transforms to HTML using XSLT and CSS
- Remote procedure call: allows distributed computing
- E-commerce: sends data from one company to another

XML Namespace

- In XML, element names are defined by the developer
- This often results in a conflict when trying to mix XML documents from different XML applications
- XML Namespaces provide a method to avoid element name conflicts

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

    <table>
        <tr>
            <td>Apples</td>
            <td>Bananas</td>
        </tr>
    </table>

    <table>
        <name>Coffee Table</name>
        <width>80</width>
        <length>120</length>
    </table>

</bookstore>
```


XML Namespace example

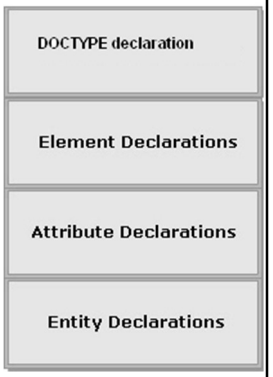
```
<?xml version ="1.0" encoding="UTF-8"?>
<bookstore xmlns:h='http://www.w3.org/TR/htmi4/'
           xmlns:f='http://www.w3schools.com/furniture'>

    <h:table>
        <h:tr>
            <h:td>Apples</h:td>
            <h:td>Bananas</h:td>
        </h:tr>
    </h:table>

    <f:table>
        <f:name>Coffee Table</f:name>
        <f:width>80</f:width>
        <f:length>120</f:length>
    </f:table>

</bookstore>
```


Document Type Definition (DTD)

- The purpose of a DTD is to define the structure of an XML document
- A DTD document defines:
  - The legal building blocks of an XML document
  - The document structure with a list of legal elements and attributes
  - The way elements relate to one another within the document's tree structure
- A DTD can be declared:
  - Inline inside an XML document
  - External reference

Structure of a DTD

1. DOCTYPE declaration
   - Specifies name of that DTD and either its content or location
2. ELEMENT declaration
   - Specifies the name of the element, the content which that element can contain
3. ATTRIBUTE declaration
   - Specifies the element that owns the attributes, the attribute name, its type and its default values (if any)
4. ENTITY declaration
   - Specifies the name of the entity and either its value or location of its values.



Example DTD

```
<!DOCTYPE thuVien [
    <!ELEMENT thuVien (sach+)>
    <!ELEMENT sach (id, ten, tacGia,nhaXuatBan?,gia?)>
    <!ELEMENT id (#PCDATA)>
    <!ELEMENT ten (#PCDATA)>
    <!ELEMENT tacGia (#PCDATA)>
    <!ELEMENT nhaXuatBan (#PCDATA)>
    <!ELEMENT gia (#PCDATA)>
    <!ATTLIST sach theLoai (Khoahọc|Giảitrí|Tinhọc) "Tinhọc">
    <!ATTLIST sach ngonNgu CDATA #IMPLIED>
    <!ENTITY nxbt "Nhà xuất bản Trẻ">
    <!ENTITY nxbgd "Nhà xuất bản Giáo dục">
]>
```


Example of internal DTD declaration

- If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition
- Syntax: **<!DOCTYPE root-element [element-declarations]>**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to       (#PCDATA)>
  <!ELEMENT from     (#PCDATA)>
  <!ELEMENT heading  (#PCDATA)>
  <!ELEMENT body     (#PCDATA)>
]>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend</body>
</note>
```


Example of external DTD declaration

- If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition
- Syntax: **<!DOCTYPE root-element SYSTEM "filename">**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML Schema

- Defines elements that can appear in a document
- Defines attributes that can appear in a document
- Defines which elements are child elements
- Defines the order of child elements
- Defines the number of child elements
- Defines whether an element is empty or can include text
- Defines data types for elements and attributes
- Defines default and fixed values for elements and attributes

DTD limitations and XML Schema

- An objective of XML schema: overcome the drawbacks of DTDs
  - DTDs are written in a non-XML syntax
  - DTDs are not extensible
  - They have no support for namespaces
- The XML Schema language is referred as XML Schema Definition (XSD)

Example of XML Schema

```
<?xml version="1.0"?>
<Blog>
    <Title>Let me know what you think</Title>
    <Author>Yin Yang</Author>
</Blog>
```

```
<?xml version="1.0"?>
<xsd:schema  elementFormDefault="qualified"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Blog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" />
        <xsd:element name="Author" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```


Referencing a Schema in an XML Document

```
<?xml version="1.0"?>
<Blog xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://yinyangit.wordpress.com/test.xsd">
    <Title>Let me know what you think</Title>
    <Author>Yin Yang</Author>
</Blog>
```

- Read more:

  http://www.w3schools.com/xml/default.asp

- …

JAXP

Java API for XML Processing

JAXP – Java API for XML Processing

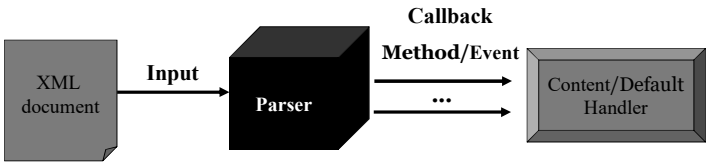
- JAXP is a collection of APIs that you can use in your Java applications to process and translate XML documents
- Consists of three APIs:
  - Simple API for XML (**SAX**): Allows you to use a SAX parser to process the XML documents serially
  - Document Object Model (**DOM**): Allows you to use a DOM parser to process the XML documents in an object-oriented manner
  - XML Stylesheet Language for Transformation (**XSLT**): Allows you to transform XML documents in other formats, such as HyperText Markup Language (HTML)

SAX

SAX

- Simple API for XML
- Started out as a Java API, but now exists for other languages too
- Quantity of memory usage is low
- Only for reading xml-documents
- Uses a event-driven model

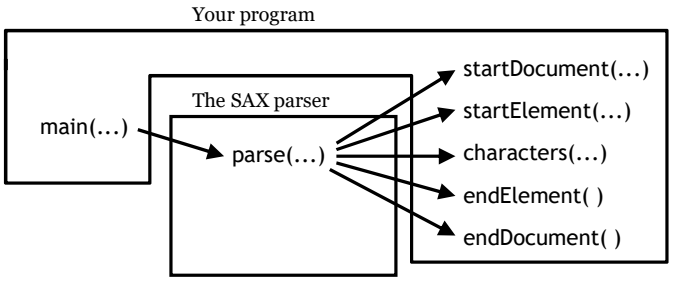
Parsing an XML document



- The three steps followed for parsing an XML document are:
  - Application supplies a SAX content handler to the parser
  - Application tells parser to start parsing a document
  - Parser calls back the ContentHandler/DefaultHandler

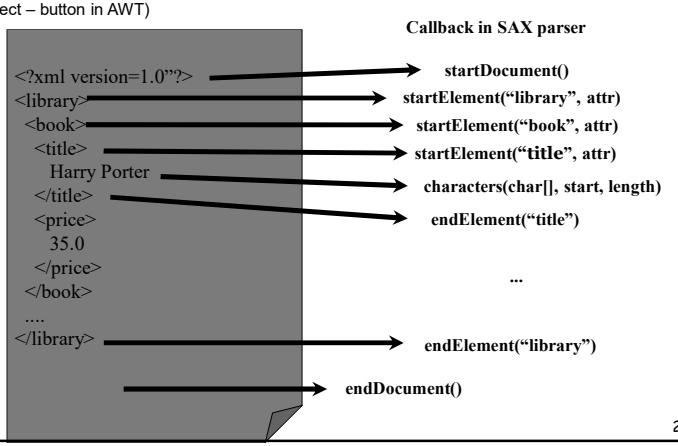
Callbacks

- SAX works through callbacks: you call the parser, it calls methods that you supply



Callback interface

- SAX is used with XMLReader as the Subject and org.xml.sax.ContentHandler/ org.xml.sax.helpers.DefaultHandler as an Observer (is similar to define the event Listener to subject – button in AWT)



DOM

DOM (1)

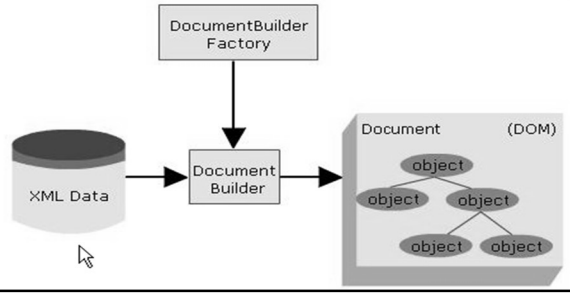
- DOM is an API for HTML and XML documents
- DOM reads the entire XML document into **memory** and stores it as a **tree data structure** (with **each XML element represented as a node)**
- DOM acts as a **language-independent interface** between programs to access and modify data in a document. This provides the retrieved data in XML document a logical structure and style of representation
- Different versions: DOM 1, DOM 2, DOM 3 and DOM 4

DOM (2)

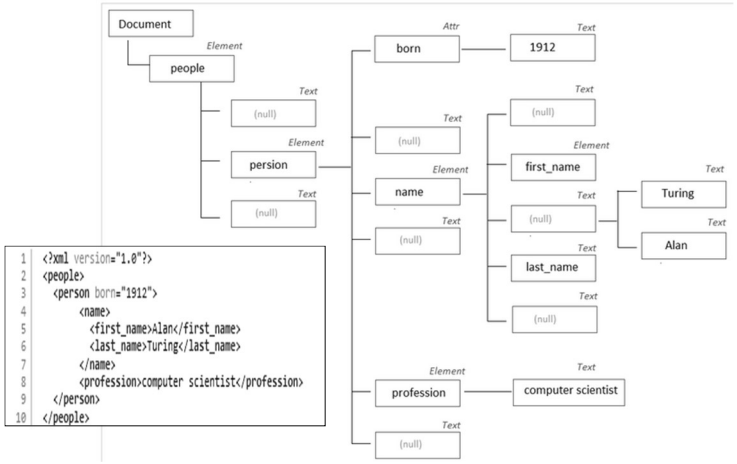
- Benefits
  - provides access to multiple documents: DOM creates the whole tree structure of XML documents while processing
  - manages complex data structures: DOM creates a structures document-tree for an XML document having complex cross references
  - allows modification of documents: it allows both reading and modifying a parsed XML document
  - allows many methods to access a document simultaneously: As the entire document is available in memory after it is parsed, you may use any of the DOM API calls multiple times to access, update, and delete document contents and structure

Working of DOM

- The DOM works with the following steps:
  - The methods of **DocumentBuilder** class in the DOM API create a tree structure of the XML document and represent each element as a node
  - The methods contained in various interfaces in the DOM API provide access to the document and its nodes to add, modify, or delete nodes or elements in the document



Working of DOM



Steps to using DOM: Creating a Parsed Document

1. Import XML-related packages

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;
```

2. Create a DocumentBuilder

```
DocumentBuilderFactory fac
                    = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = fac.newDocumentBuilder();
```

3. Create a Document from a file or stream

```
Document document = builder.parse(new File(file));
```


Steps to using DOM example

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;

public class FirstDOM {
    public static void main(String[]args)throws Exception{
        DocumentBuilderFactory fac=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=fac.newDocumentBuilder();
        Document doc=builder.parse("Books.xml");
        processDocument(doc);
    }
    private static void processDocument(Document doc) {
        //xử lý
    }
}
```


Steps to Using DOM: Extracting data from Parsed Document

4. Extract the root element

```
Element root = document.getDocumentElement();
```

5. Examine attributes

- getAttribute("attributeName") returns specific attribute
- getAttributes() returns a Map (table) of names/values

6. Examine sub-elements

- getElementsByTagName("subelementName") returns a list of subelements of specified name
- getChildNodes() returns a list of all child nodes

Document interface

- Is the root of the DOM tree, which provides access to the data in the XML document
- Contains factory methods to create the elements, text nodes, comments, and processing instructions

Methods	Descriptions
getDocType	Returns the document type
getDocumentElement	Returns the attribute that allows direct access to the root element of the XML document
createElement	Created an element of the specified type
createTextNode	Creates a Text node with the string specified as the argument
createAttribute	Creates an Attribute in the given name. The instance of the attribute then can be set to an element using the setAttributeNode() method
getElementsByTagName	Returns an instance of the NodeList interface of all the elements with a given tag name in the order in which they are encountered in a pre order traversal of the document tree

Node interface

- Acts as the primary data type for the whole DOM model
- Contains various methods to access and manipulate the nodes in a DOM document

Methods	Descriptions
getNodeName	Returns the name of the node depending on its type
getNodeValue	Returns the value of the node depending on its type
setNodeValue	Assigns a value to the specific node depending on its type
getChildNodes	Returns a NodeList containing Node's children
getAttributes	Returns an instance of the NameNodeMap interface containing the attributes of the node

NodeList & Element interface

- NodeList Interface
  - public Node **item**(int index)
    - returns element at given index
      - item returns Node (parent interface of Element), so you should do a typecast on return value
  - public int **getLength**()
    - returns number of child elements
- Element Interface
  - The Element object is a type of node encountered in a document tree

NodeList example (1)

```
<?xml version="1.0" ?>
<Employees>
  <Employee id="1">
    <name>Pankaj</name>
    <age>29</age>
    <role>Java Developer</role>
    <gender>Male</gender>
  </Employee>
  <Employee id="2">
    <name>Lisa</name>
    <age>35</age>
    <role>CSS Developer</role>
    <gender>Female</gender>
  </Employee>
</Employees>
```

```
static void printName(Document doc) throws Exception {
    NodeList nl = doc.getElementsByTagName("name");
    for (int i = 0; i < nl.getLength(); i++) {
        Node n = nl.item(i);
        System.out.println(n.getTextContent());
    }
}
```

Result:
Pankaj
Lisa

NodeList example (2)

```
static void printName(Document doc) throws Exception {
    NodeList nl = doc.getElementsByTagName("Employee");
    for (int i = 0; i < nl.getLength(); i++) {
        Node n = nl.item(i);
        System.out.println(n.getTextContent());
    }
}
```

```
<?xml version="1.0" ?>
<Employees>
  <Employee id="1">
    <name>Pankaj</name>
    <age>29</age>
    <role>Java Developer</role>
    <gender>Male</gender>
  </Employee>
  <Employee id="2">
    <name>Lisa</name>
    <age>35</age>
    <role>CSS Developer</role>
    <gender>Female</gender>
  </Employee>
</Employees>
```

Result:

```
Pankaj
29
Java Developer
Male

Lisa
35
CSS Developer
Female
```


Attr interface

- Represents an attribute in an Element object
- Typically the allowable values for the attribute are defined in a schema associated with the document
- Attr objects inherit the Node interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree

```
Attr att = doc.createAttribute("name");
att.setValue("value");
```


Getting Attribute example

```
<?xml version="1.0" ?>
<rootOfXml myAttr01="Abc" myAttr02="What is this?">
  This is text 01
  <MyElement1> What is this? </MyElement1>
  This is text 02
</rootOfXml>
```

```
Element root = (Element) doc.getDocumentElement();
System.out.println("Root name: " + root.getNodeName());

NamedNodeMap nodeMap = root.getAttributes();
for (int i = 0; i < nodeMap.getLength(); i++) {
    Attr at = (Attr) nodeMap.item(i);
    System.out.println("Name: " + at.getName() +
                       " Value: " + at.getValue());
}
```

Result:
Root name: rootOfXml
Name: myAttr01 Value: Abc
Name: myAttr02 Value: What is this?

Text interface

- Inherits from CharacterData and represents the textual content (termed character data in XML) of an Element or Attr
- No lexical check is done on the content of a Text node and, depending on its position in the document, some characters must be escaped during serialization using character references:
  - e.g. the characters "<&" if the textual content is part of an element or of an attribute, the character sequence "]]>" when part of an element, the quotation mark character (") or the apostrophe character (') when part of an attribute

Manipulating DOM

Types of Node

- Document
- DocumentFragment
- DocumentType
- ProcessingInstruction
- Text
- Comment
- CDATASection
- Element
- Attr
- EntityReference
- Entity
- Notation

Nodes in DOM tree

- Document
  - The root node of the XML document
  - The Document interface then manipulates the Document node through the methods defined in it
  - This type of node can contain only a single child node. Its child nodes can be a processing instruction element, a document type element, a comment element, elements
- DocumentFragment
  - Holds a portion of a complete document
  - Is created by the methods present in the Document interface
  - Can have processing instruction, comment, text, CDATA section, and entity reference as its child nodes

Nodes in DOM tree

- DocumentType
  - Each document has a DOCTYPE attribute that whose value is either null or a DocumentType object
  - Provides an interface to the entities defined for the document
- ProcessingInstruction
  - This is just a processor specific instruction kept in the XML document
  - The Document interface creates a Processing Instruction node

Nodes in DOM tree

- Element
  - Represents an element in an XML document. Elements may contain attributes, other elements, or text. If an element contains text, the text is represented in a text-node
- Attr
  - Represents an attribute of an Element object. The allowable values for attributes are usually defined in a DTD
  - Because the Attr object is also a Node, it inherits the Node object's properties and methods. However, an attribute does not have a parent node and is not considered to be a child node of an element, and will return null for many of the Node properties

Nodes in DOM tree

- Text
  - Represents the textual content of an element or attribute
- Comment
  - Represents the content of comment nodes in a document
- CDATASection
  - Represents a CDATA section in a document
  - A CDATA section starts with "**<![CDATA[**" and ends with "**]]>**“
  - Contains text that will NOT be parsed by a parser. Tags inside a CDATA section will NOT be treated as markup and entities will not be expanded. The primary purpose is for including material such as XML fragments, without needing to escape all the delimiters
  - CDATA sections cannot be nested

Nodes in DOM tree

- Notation
  - Represents a notation declared in the DTD
- Entity
  - Represents an entity
- EntityReference
  - Represents an entity reference

Creating Nodes

- Creating an Element Node
  - public Element **createElement**(String tagName) throws DOMException
- Creating an Attribute Node
  - public Attr **createAttribute**(String Name) throws DOMException
- Creating a Text Node
  - public Text **createTextNode**(String data)
- Creating a Comment Node
  - public Comment **createComment**(String data)
- Creating a CDATA Section Node
  - public CDATASection **createCDATASection**(String data) throws DOMException

Appending Nodes

- Add a node to the end of a node list
  - public Node **appendChild**(Node newChild) throws DOMException
- Insert a node before a specific node
  - public Node **insertBefore**(Node newChild, Node refChild) throws DOMException
- Set a new attribute and attribute value
  - public void **setAttribute**(String name, String value) throws DOMException
- Insert data into a text node
  - public void **insertData**(int offset, String arg) throws DOMException

Deleting Nodes

- Remove an Element
  - element.getParentNode().**removeChild**(element);
- Remove an Attribute
  - element.**removeAttribute**("attribute name");

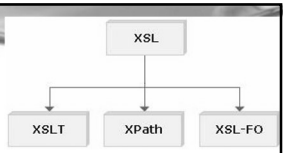
Modifying an Attribute

- removeAttribute
- removeAttributeNode
- setAttributeNode

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = factory.newDocumentBuilder();
Document doc = docBuilder.newDocument();
Element root = doc.createElement("root");
doc.appendChild(root);
//create child element
Element childElement = doc.createElement("Child");
//Add the atribute to the child
childElement.setAttribute("attribute1","The value of Attribute 1");
//Add text value to the child element
childElement.setTextContent("The test value");
root.appendChild(childElement);

TransformerFactory fac=TransformerFactory.newInstance();
Transformer trm=fac.newTransformer();
trm.transform(new DOMSource(doc), new StreamResult(System.out));
childElement.removeAttribute("attribute1");
trm.transform(new DOMSource(doc), new StreamResult(System.out));
```


XSLT

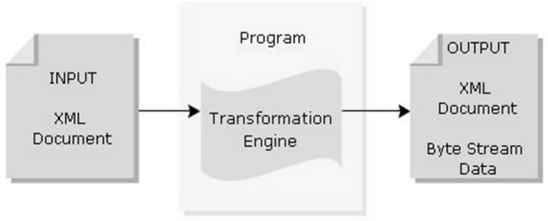
XSL



- XSL (eXtensible Stylesheet Language) is a language for expressing style sheets, consists:
  - XSLT (XSL Transformation): an XML language for transforming XML documents
  - XPath (XML Path Language): a non-XML language used by XSLT, and also available for use in non-XSLT contexts, for addressing the parts of an XML document
  - XSL-FO (XSL Formatting Objects): an XML language for specifying the visual formatting of an XML document

XML Transformations

- XML to XML: The output is an XML document
- XML to Data: The output is a byte stream document



XML Transformations

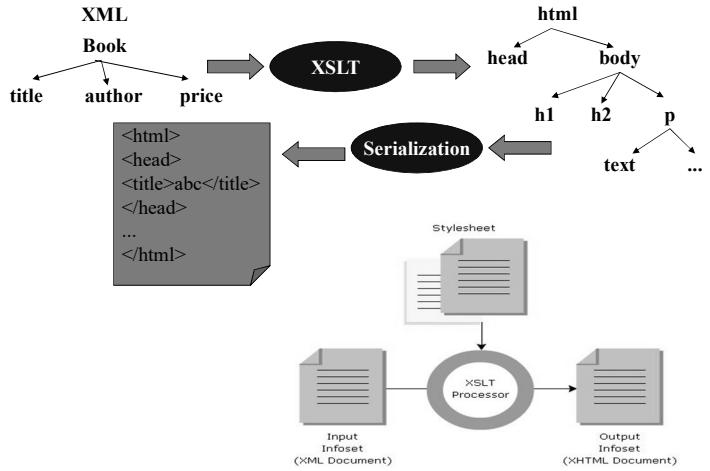
API

Packages	Descriptions
javax.xml.transform	Implements the generic APIs for transformation instructions and executing the transformation, starting from source to destination The interfaces present in this package are ErrorListener, Result, Source, SourceLocator, Templates, URIResolver The classes present in this package are **OutputKeys, Transformer, TransformerFactory**
javax.xml.transform.stream	Implements streams and URI specific transformation APIs. The classes present in this package are **StreamResult**, **StreamSource**
javax.xml.transform.dom	Implements DOM-related transformation APIs. The interface present in this package is DOMLocator
javax.xml.transform.sax	Implements SAX-related transformation APIs. The interfaces present in this package are TemplateHandler, TransformerHandler

XSLT Stylesheet (1)

- Is a language used for transforming XML documents into XHTML documents for Web pages
- Uses XPath language to navigate between XML documents and XHTML documents
- Defines the source documents, which will be matched with a predefined set of templates in transformation process. If XSLT gets the same document, it transforms the matching part of the source document into the output document

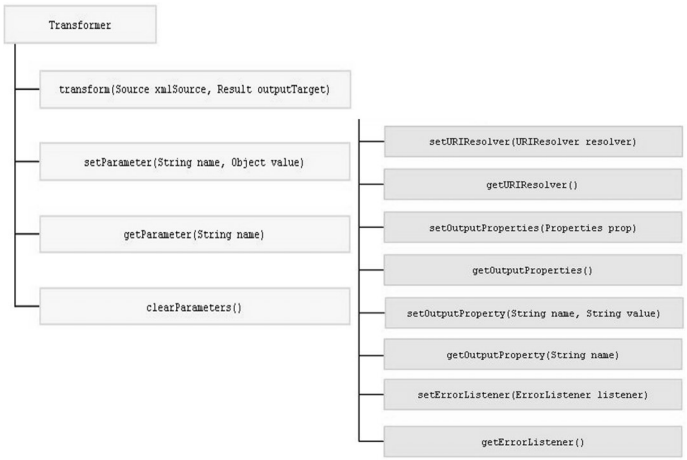
XSLT Stylesheet (2)



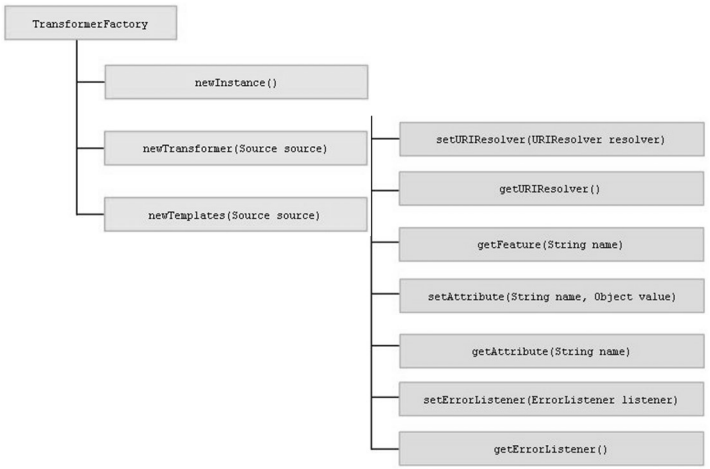
XSLT Example

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <class>
3      <maLop>ACCP0808</maLop>
4      <maMon>AFSF</maMon>
5      <sinhVien>
6          <maSV>AP0001</maSV>
7          <ho>Trần Văn</ho>
8          <ten>An</ten>
9          <diem>80</diem>
10     </sinhVien>
11     <sinhVien>
12         <maSV>AP0002</maSV>
13         <ho>Đặng Thái</ho>
14         <ten>Thịnh</ten>
15         <diem>55</diem>
16     </sinhVien>
17     <sinhVien>
18         <maSV>AP0003</maSV>
19         <ho>Tăng An</ho>
20         <ten>Hộ</ten>
21         <diem>34</diem>
22     </sinhVien>
23 </class>
```

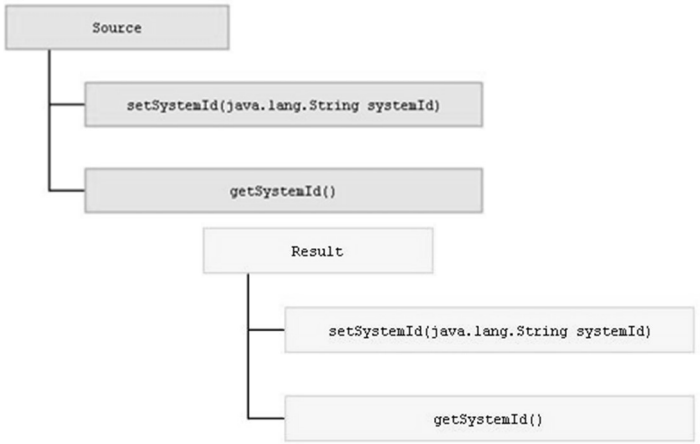
Transformer



TransformerFactory



Source and Result



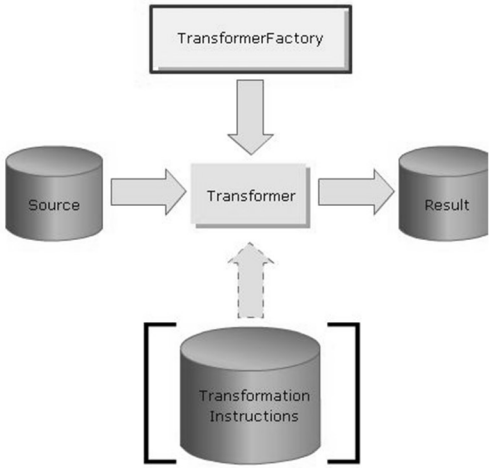
65

Template

- The object, which implements the Template interface, is the runtime result of transformation instructions
- Can be called from multiple threads for a particular instance

66

Transforming XML Document



67

Example

```
 8 public class SimpleTransform_XSLT
 9 {
10    public static void main(String[] args)
11            throws Exception
12    {
13        String xmlFile ="files/CustomerOrders.xml";
14        String xslFile = "files/OrderProcessing.xslt";
15        String outFile = "files/output.html";
16        StreamSource xslSource = new StreamSource(xslFile);
17        TransformerFactory factory =
18            TransformerFactory.newInstance();
19        Transformer transformer =
20            factory.newTransformer(xslSource);
21
22        transformer.transform( new StreamSource(xmlFile),
23                new StreamResult(outFile));
24    }
25 }
```

You can display the output in console by using `System.out` instead of using `outfile`.

Using `transformer.setOutputProperty(OutputKeys.INDENT, "yes")` to display result with indentation

68

Transform with parameter

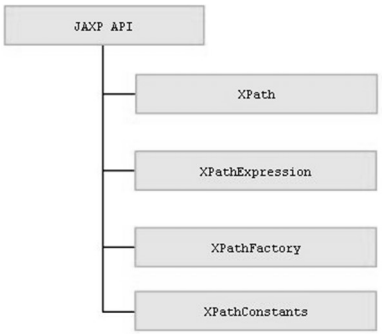
```
public class TransformEX {
     public static void main(String[] args) throws Exception {
          Transformer trans =TransformerFactory.newInstance().newTransformer(
                    new StreamSource("cd.xsl"));
          trans.setParameter("para", "9");
          trans.transform(new StreamSource("cdcatalog.xml"),
                    new StreamResult(System.out));
     }
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:param name="para"/>

    <xsl:template>
    <xsl:template match="/">
        <table border="1" width="100%">
            <xsl:for-each select="catalog/cd">
                <xsl:sort data-type="number" order="descending" select="price"/>
                <xsl:if test="price &gt; $para">
                    <tr>
                </xsl:if>
            </xsl:for-each>
        </table>
    </xsl:template>
</xsl:stylesheet>
```


JAXP API For XPath Processing

JAXP API For XPath Processing (1)

- Define in the **javax.xml.xpath** package
- Has 2 interfaces and 2 classes:



JAXP API For XPath Processing (2)

- The **XPath** interface gives syntax for traversing through the nodes in an XML document
- The **XPathExpression** interface deals with location path and predicates
- The **XPathFactory** class is used for creating XPath objects
- The **XPathConstants** class defines the data types such as Boolean, NodeSet, number, and string for working with nodes in and XML document

Terminology

```
<library>
  <book>
    <chapter>
    </chapter>

    <chapter>
      <section>
        <paragraph/>
        <paragraph/>
      </section>
    </chapter>
  </book>
</library>
```

- library is the parent of book; book is the parent of the two chapters
- The two chapters are the children of book, and the section is the child of the second chapter
- The two chapters of the book are siblings (they have the same parent)
- library, book, and the second chapter are the ancestors of the section
- The two chapters, the section, and the two paragraphs are the descendents of the book

XPath

- /library = the root element (if named library)
- /library/book/chapter/section = *every* section element in a chapter in every book in the library
- section = *every* section element that is a child of the current element
- . = the current element
- .. = parent of the current element
- /library/book/chapter/* = all the elements in /library/book/chapter

Slashes

- A path that begins with a / represents an absolute path, starting from the top of the document
  - Example: /email/message/header/from
  - Note that even an absolute path can select more than one element
  - A slash by itself means "the whole document"
- A path that does not begin with a / represents a path starting from the current element
  - Example: header/from
- A path that begins with // can start from anywhere in the document
  - Example: //header/from selects every element from that is a child of an element header
  - This can be expensive, since it involves searching the entire document

Brackets and last()

- A number in brackets selects a particular matching child (counting starts from 1, except in Internet Explorer)
  - Example: /library/book[1] selects the first book of the library
  - Example: //chapter/section[2] selects the second section of every chapter in the XML document
  - Example: //book/chapter[1]/section[2]
  - Only matching elements are counted; for example, if a book has both sections and exercises, the latter are ignored when counting sections
- The function last() in brackets selects the last matching child
  - Example: /library/book/chapter[last()]
- You can even do simple arithmetic
  - Example: /library/book/chapter[last()-1]

Stars

- A star, or asterisk, is a “wild card”—it means “all the elements at this level”
  - Example: /library/book/chapter/* selects every child of every chapter of every book in the library
  - Example: //book/* selects every child of every book (chapters, tableOfContents, index, etc.)
  - Example: /*/*/*/paragraph selects every paragraph that has exactly three ancestors
  - Example: //* selects every element in the entire document

Attributes (1)

- You can select attributes by themselves, or elements that have certain attributes
  - Remember: an attribute consists of a name-value pair, for example in <chapter num="5">, the attribute is named num
  - To choose the attribute itself, prefix the name with @
  - Example: @num will choose every attribute named num
  - Example: //@* will choose every attribute, everywhere in the document
- To choose elements that have a given attribute, put the attribute name in square brackets
  - Example: //chapter[@num] will select every chapter element (anywhere in the document) that has an attribute named num

Attributes (2)

- //chapter[@num] selects every chapter element with an attribute num
- //chapter[not(@num)] selects every chapter element that does *not* have a num attribute
- //chapter[@*] selects every chapter element that has *any* attribute
- //chapter[not(@*)] selects every chapter element with *no* attributes

Values of attributes

- //chapter[@num='3'] selects every chapter element with an attribute num with value 3
- //chapter[not(@num)] selects every chapter element that does *not* have a num attribute
- //chapter[@*] selects every chapter element that has *any* attribute
- //chapter[not(@*)] selects every chapter element with *no* attributes
- The normalize-space() function can be used to remove leading and trailing spaces from a value before comparison
  - Example: //chapter[normalize-space(@num)="3"]

XPath Example

```
public static void main(String[] args)
throws ParserConfigurationException, SAXException,
IOException, XPathExpressionException {

    DocumentBuilderFactory domFactory =
        DocumentBuilderFactory.newInstance();
    domFactory.setNamespaceAware(true);
    DocumentBuilder builder = domFactory.newDocumentBuilder();
    Document doc = builder.parse("files/persons.xml");

    XPath xpath = XPathFactory.newInstance().newXPath();
    // XPath Query for showing all nodes value
    XPathExpression expr = xpath.compile("//person[@id='2']/*/text()");

    Object result = expr.evaluate(doc, XPathConstants.NODESET);
    NodeList nodes = (NodeList) result;
    for (int i = 0; i < nodes.getLength(); i++) {
        System.out.println(nodes.item(i).getNodeValue());
    }
}
```


Difference between SAX and DOM

SAX	DOM
SAX reads the XML document and sends an event for each element that it encounters	DOM reads the entire XML document into memory and stores it as a tree data structure
Sequential access	Random access
Fast and requires very little memory	Slow and requires huge amounts of memory, so it cannot be used for large XML documents
Cannot update documents	Changing the XML document in memory

Advanced Java Programming Course

Java Database Connectivity

By Võ Văn Hải
Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Session objectives

- JDBC basic
- Working with JDBC
- Advanced JDBC programming

JDBC basic

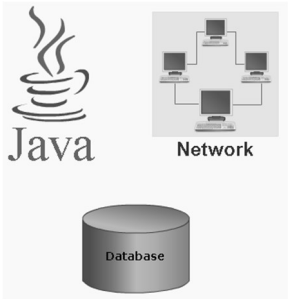
Part 1

Definitions of JDBC

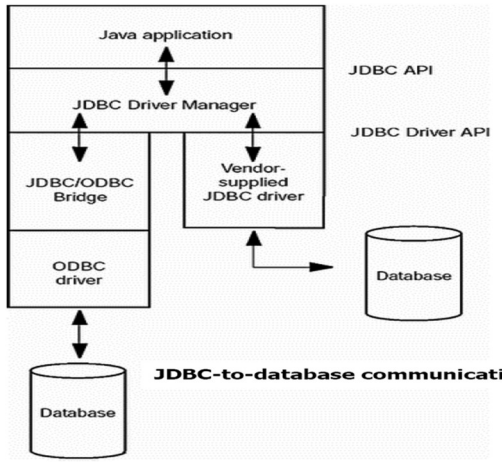
- JDBC APIs, which provides a set of classes and interfaces written in Java to access and manipulate different kinds of database.
- Defines the way how DB and application communicate with each other.
- JDBC APIs has implementation separately by set of classes for a specific DB engine known as JDBC driver.
  - Define how a connection is opened
  - How requests are communicated to a DB
  - How SQL queries are executed, the query results retrieved.

Advantages of JDBC

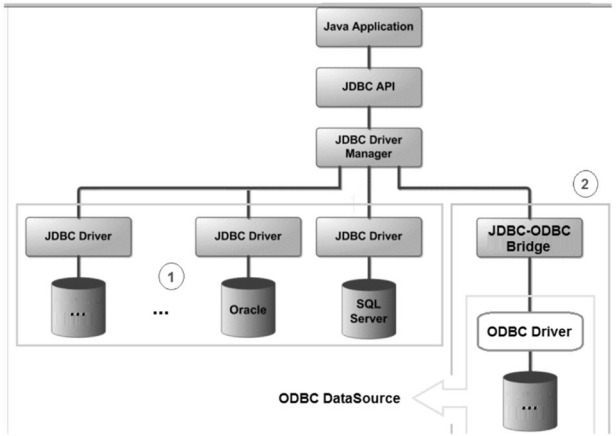
- Continued usage of existing data
  - Even if the data is stored on different DBMS
- Vendor independent
  - Thanks to the combination of Java API and the JDBC API
- Platform independent
- Ease of use



How Application and Backend Database communicate?



JDBC architecture

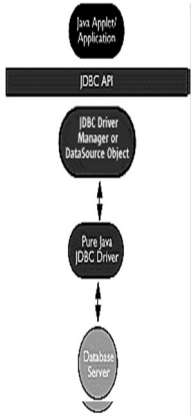


JDBC Drivers

- **JDBC driver** is base of JDBC API
- Responsible for ensuring that an application has a consistent and uniform access to any DB
- Converts the client request to a DB understandable, native format and then presents it to the DB
- Handled DB response, and gets converted to the Java format and presented to the client.

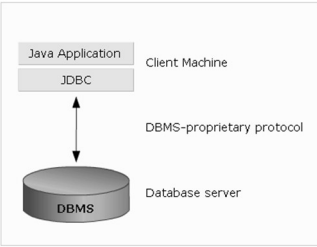
Types of JDBC driver

Driver Type	Driver Name	Description
Type I	ODBC-JDBC Bridge	Translates JDBC calls into ODBC calls
Type II	Native API-Java/Partly Java	Translates JDBC calls into database-specific calls or native calls
Type III	JDBC Network-All Java	Maps JDBC calls to the underlying "network" protocol, which in turn calls native methods on the server
Type IV	Native Protocol-All Java	Directly calls RDBMS from the client machine



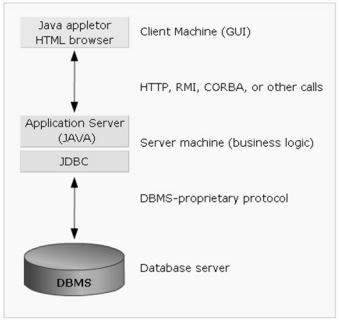
Two-Tier Data Processing Model

- The client communicates directly to the DB server without the help of any middle-ware technologies or another server.
- JDBC API translate and sent the client request to the DB.
- The result are delivered back to the client again though the JDBC API



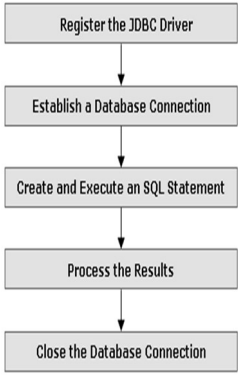
Three-Tier Data Processing Model

- The "middle-tier" is employed to send the client request to the DB server.
- The DB server processes the request
- Then DB server sent back the results to the middle tier which again sends it to the client.

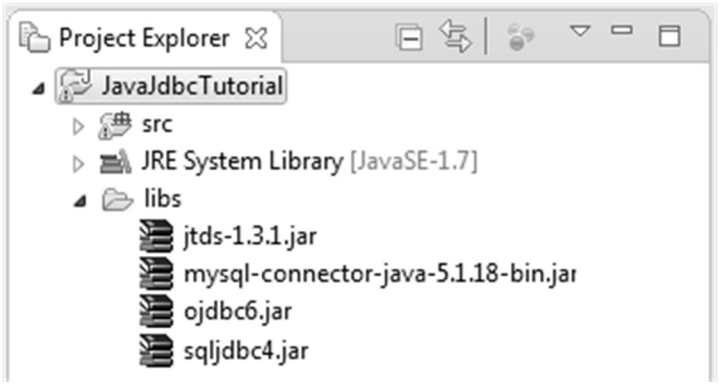


JDBC Application Development Process

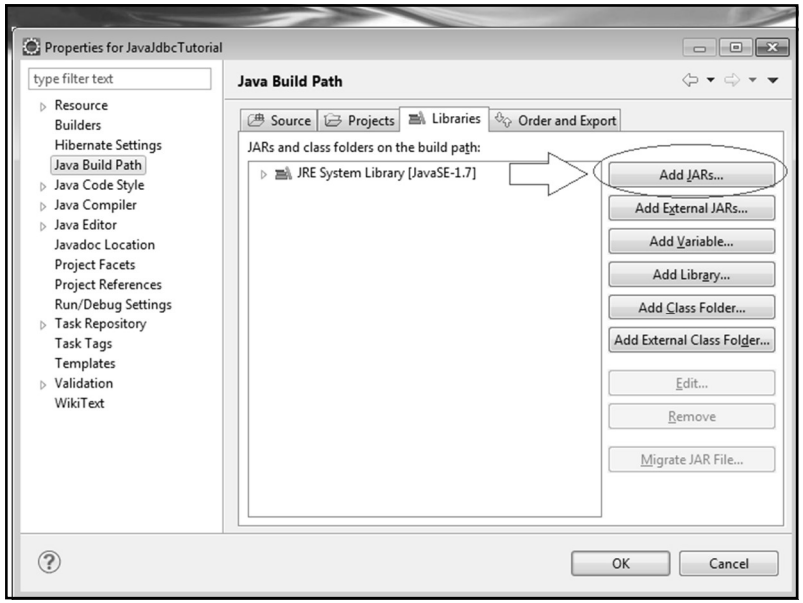
0. Register and Loading the *JDBC driver* with **DriverManager**
1. Establish a DB **connection**
2. Create a SQL statement
3. Execute a **SQL statement**
4. Process the **SQL Query results**
5. Close the DB connection



Create project with JDBC



13



#0: Loading and register the JDBC driver with DriverManager

- DriverManager class
  - Manages the set of JDBC drivers available for an application.
- Class.forName() method
  - To create an instance of the specified driver class
  - Register it with the DriverManager class.
- Syntax: Class.forName (<driver>);
- Popular drivers:

```
String odbc_driver="sun.jdbc.odbc.JdbcOdbcDriver";
String msSQL_driver="com.microsoft.sqlserver.jdbc.SQLServerDriver";
String mySQL_driver="com.mysql.jdbc.Driver";
String oracle_driver="oracle.jdbc.driver.OracleDriver";
```

15

#1: Establishing Database Connection

- Use getConnection() method of DriverManager class
- Connection URL:

  Syntax protocol : <subprotocol> : <subname>

  1. protocol: protocol name for accessing the DB
  2. subprotocol: recognizes the underlying DB source
  3. subname: identifies the DB name
- Syntax:

```
Connection con=DriverManager.getConnection(url,"username","password");
```

- Popular URL

```
String jdbc_url="jdbc:odbc:your_DSN";
String msSQL_url="jdbc:sqlserver://serverName:1433;databaseName=databaseName";
String mySQL_url="dbc:mysql://serverName:3306/databaseName";
String oracle_url="jdbc:oracle:oci:@serverName/schemaName";
```

16

Establishing Database Connection

Example: Connect to Ms Access

```
import java.sql.Connection;
import java.sql.DriverManager;

public class ConnectMsAccessDBSample {
    public static void main(String[] args) throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=null;
        try {
            String filepath="data/myDB.accdb";
            String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb, *.accdb)};" +
                    "DBQ="+filepath;
            con=DriverManager.getConnection(url,"admin","");
            //do your works
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            con.close();
        }
    }
}
```


Get a Connection to database

Example: Connect to MySQL

```
Connection con = null;
try{
    //Step 1: Get a connection to database
    String url = "jdbc:mysql://localhost:3306/employeedb";
    String user = "root";
    String password = "12345678";
    con = DriverManager.getConnection(url, user, password);
    System.out.println("Success");
}catch(Exception ex){
    ex.printStackTrace();
}finally{
    try{
        if(con != null)
            con.close();
    }catch(Exception ex){
        ex.printStackTrace();
    }
}
```


#2: Creating a SQL Statement

- "Statement" object represent a "command" send to the database for query execution
- Three categories of Statement

1. Statement:
   - Execute SQL queries with no parameters to be parsed.
2. PreparedStatement:
   - Execute a precomplied SQL statement with or without IN parameters
3. CallableStatement:
   - Execute a call to store procedure.

- Samples:

```
Statement stmt=con.createStatement();
PreparedStatement pstmt= con.prepareStatement(sql);
CallableStatement cstmt=con.prepareCall(sql);
```

example: Create a Statement object

- The Statement object is based on connection.
  - It will be used later to execute SQL query.

```
//Step 1: Get a connection to database
String url = "jdbc:mysql://localhost:3306/employeedb";
String user = "root";
String password = "12345678";
con = DriverManager.getConnection(url, user, password);
//Step 2: Create a statement object
Statement myStmt = con.createStatement();
```

#3: Execute a SQL statement

- Execute methods of Statement object:
  - executeUpdate: exec a non return value query (delete,insert,update…queries).
  - executeQuery: execute and return a set of records.
  - executeBatch: execute a batch of commands.
  - execute: multipurpose command to execute any types of SQL command.

Example:

```
try
//Step 2: Create a statement object
Statement myStmt = con.createStatement();
//Step 3: Execute SQL query
ResultSet myRs = myStmt.executeQuery("Select * From Employees");
```


#4 - Process the SQL Query results

- Represent as "ResultSet" object
- A table of data representing a database result set
- Generated by executing a SQL SELECT statement
- In the beginning, the cursor is positioned before the first row.
- The next() method moves the cursor to the next row
  - Returns false when there are no more rows in the ResultSet object
- A default ResultSet object is not updatable and has a cursor that moves forward only

BOF
Employee_Name
Annal
Peter
....
Jack
EOF

Example

```
Select Emp_Id, Emp_No, Emp_Name from Employee
```

next()
next()
next()

	1 EMP_ID	2 EMP_NO	3 EMP_NAME
1	7839	E7839	KING
2	7566	E7566	JONES
3	7902	E7902	FORD
4	7369	E7369	SMITH
5	7698	E7698	BLAKE
6	7499	E7499	ALLEN
7	7521	E7521	WARD
8	7654	E7654	MARTIN
9	7782	E7782	CLARK
10	7788	E7788	SCOTT
11	7844	E7844	TURNER
12	7876	E7876	ADAMS
13	7900	E7900	ADAMS
14	7934	E7934	MILLER

```
Connection connection = ....;

Statement statement =
                connection.createStatement();


ResultSet rs = statement.executeQuery(sql);


while (rs.next()) {
   int empId = rs.getInt(1);
   String empNo = rs.getString(2);
   String empName = rs.getString("Emp_Name");
}
```


ResultSet object

- ResultSet object :
  - Get a set of records after execute a select sql command.

    `ResultSet rs = statement.executeQuery(sql);`
  - Using getXXX() methods to get specify values from ResultSet

    `String id=rs. getString("StudentID");`
  - Parameters in getXXX() methods can be a database field name or the index of DB field. Database column numbers start at 1.
  - Each *getXXX()* method will make reasonable type conversions when the type of the method doesn't match the type of the column.
  - SQL data types and Java data types are not exactly the same

SQL data type vs. Java data type

Or.	SQL data type	Java data type
1	INTEGER or INT	int
2	SMALLINT	short
3	NUMERIC(*m*,*n*), DECIMAL(*m*,*n*)or DEC(*m*,*n*)	java.sql.Numeric
4	FLOAT(n)	double
5	REAL	float
6	DOUBLE	double
7	CHARACTER(*n*)or CHAR(*n*)	String
8	VARCHAR(*n*)	String
9	BOOLEAN	boolean
10	DATE	java.sql.Date
11	TIME	java.sql.Time
12	TIMESTAMP	java.sql.Timestamp
13	BLOB	java.sql.Blob
14	CLOB	java.sql.Clob
15	ARRAY	java.sql.Array

SQL data types and their corresponding Java language types

ResultSet example

```
//Step 3: Execute SQL query
ResultSet myRs = myStmt.executeQuery(
        "Select last_name,first_name,email " +
        "From Employees");
//Step 4: Process Results set
while(myRs.next()){
    System.out.println(myRs.getString(1));
    System.out.println(myRs.getString("first_name"));
    System.out.println(myRs.getString(3));
}
```


Batch Updates

- Batch Update allows to submit multiple update commands together as a single unit, or batch, to the underlying DBMS.
- The commands in a batch can be actions such as INSERT, UPDATE, and DELETE .However, you cannot add SELECT commands to a batch.
- To execute a batch, you first create a Statement/PreparedStatement object in the usual way:
  - `Statement statement = conn.createStatement();`
- Now, instead of calling executeUpdate, you call the addBatch method:
  - `String sqlCommand = ". . ."`
  - `statement.addBatch(sqlCommand);`
- Finally, you submit the entire batch:
  - `int[] counts = stat.executeBatch();`

#5: Closing the Database Connection

- Use close() method of Connection, Statement, ResultSet object.
- Database connections should be closed within a finally block.

```
Connection con=null;
try {
    con=DriverManager.getConnection(url,"username","password");
    //do your work
} catch (Exception e) {
    //processing your exceptions
}finally{
    //close all connections, statements,...
    con.close();
}
```

JDBC data access statements

```
// Statement creation
Statement statement = connection.createStatement();
// Query string
String query = <SQLQuery>; // CRUD
// for retrieve data
ResultSet result = statement.executeQuery(query);
while (result.next()){
   result.getString(<ColName>);
   result.getInt(<ColName>);
   result.getFloat(<ColName>);
}
// for data modification: insert, update, delete
int nbUpdated = statement.executeUpdate(query);
```


JDBC ResultSet update preparation

```
// for use with ResultSet only
// No "previous" method using, no update
connection.createStatement(
   ResultSet.TYPE_FORWARD_ONLY,
   ResultSet.CONCUR_READ_ONLY);

// with "previous" method using, update
connection.createStatement(
   ResultSet.TYPE_SCROLL_SENSITIVE,
   ResultSet.CONCUR_UPDATABLE);
```


Working with JDBC

Part 2

Execute queries

- Execute select sql statement :

```
String sql="select * from sampleTable";
ResultSet rs=statement.executeQuery(sql);
//processing with resultset object
```

- Execute insert,update, delete sql statement

```
String sql="insert into aTable values('a','b')";
int recNum=statement.executeUpdate(sql);
if(recNum>0)
    System.out.println("successfull");
```

Execute sql statement with parameters

- Using parameters in your sql statement likes sample follow :
  - `String pSql="select * from tblClass where classID=?"`
    where ? replace for an parameter
- To passing parameter in this case, we must use PrepareStatement objecct instead of using Statement object.
  - `PreparedStatement pstmt=con.prepareStatement(pSql);`
- Before executing the prepared statement, you must bind the host variables to actual values with a setXXX() method.
  - Ex: setString(1, 'CDTH4C');
  - The position 1 denotes the first "?" and so on.

Scrollable and Updatable Result Sets

Scrollable Result Set

- Scrollable
  - Refers to capability to move forward, backward or go directly to a specific row.

ResultSet type	Description
TYPE_FORWARD_ONLY	For a ResultSet object whose cursor may move only forward.
TYPE_SCROLL_INSENSITIVE	For a ResultSet object that is scrollable but generally not sensitive to changes made by others.
TYPE_SCROLL_SENSITIVE	For a ResultSet object that is scrollable and generally sensitive to changes made by others.

Scrollable and Updatable Result Sets

Updatable Result Set

- Updatable
  - Allow change the data in the existing row, to insert a new row, or delete an existing row then copy the changes to the database.
  - The concurrency type of a result set determines whether it is updatable or not.

Concurrency name	Description
CONCUR_READ_ONLY	For a ResultSet object that may NOT be updated.
CONCUR_UPDATABLE	For a ResultSet object that may be updated.

resultSetType	Ý nghĩa
TYPE_FORWARD_ONLY	- ResultSet chỉ cho phép duyệt từ trên xuống dưới, từ trái sang phải. Đây là kiểu mặc định của các ResultSet.
TYPE_SCROLL_INSENSITIVE	- ResultSet cho phép cuộn tiến lùi, sang trái, sang phải, nhưng không nhạy với các sự thay đổi dữ liệu dưới DB. Nghĩa là trong quá trình duyệt qua một bản ghi và lúc nào đó duyệt lại bản ghi đó, nó không lấy các dữ liệu mới nhất của bản ghi mà có thể bị ai đó thay đổi.
TYPE_SCROLL_SENSITIVE	- ResultSet cho phép cuộn tiến lùi, sang trái, sang phải, và nhạy cảm với sự thay đổi dữ liệu.

resultSetConcurrency	Ý nghĩa
CONCUR_READ_ONLY	- Khi duyệt dữ liệu với các ResultSet kiểu này bạn chỉ có thể đọc dữ liệu.
CONCUR_UPDATABLE	- Khi duyệt dữ liệu với các ResultSet kiểu này bạn chỉ có thể thay đổi dữ liệu tại nơi con trỏ đứng, ví dụ update giá trị cột nào đó.

Scrollable and Updatable Result Sets

- To obtain scrolling result sets from your queries, you must obtain a different Statement object with the method:

```
Statement stat = conn.createStatement(rs_type,
    concurrency);
```

- For a prepared statement, use the call

```
PreparedStatement stat = conn.prepareStatement(
    command, rs_type, concurrency);
```

- Example:

```
Statement stmt = conn.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
ResultSet rs=stmt.executeQuery(sql);
```


Scrollable and Updatable Result Sets

Scrollable Result Set

- After create the scrollable ResultSet, we can scrolling is very simple:
  - rs.next(); *//move next record.*
  - rs.previous(); *//move previous record.*
  - rs.first(); *//move to first record.*
  - rs.last(); *//move to last record.*
  - rs.relative(*n*); *//move the cursor over n record(s).*
  - rs.absolute(*n*); *//set the cursor to a row $n^{th.}$*
  - int *n* = rs.getRow();*//gets the number of the current row*
    //Rows are numbered starting with 1.
  - rs.beforeFirst();*//before the first position.*
  - rs.afterLast();*// after the last position.*

Scrollable and Updatable Result Sets

Updatable Result Set

- Update the row under the cursor :
  - Using updateRow() method of updatable resulset
  - Example:
    - rs.updateString("className", "your enter class Name");
    - rs.updateRow();
- Insert new row to ResultSet:
  - rs.moveToInsertRow(); //create new insert row
  - Call rs.updateXXX("fieldName",value) methods.
  - Call rs.insertRow() ;//actual insert row
  - Call rs.moveToCurrentRow();//move to previous row.
- Delete the row under the cursor:
  - rs.deleteRow();

Student object

```
public class Student {
    private String studentID;
    private String lastName;
    private String firstName;
    private String email;
    private String classID;
    private float avgScore;
```

Select sql statement

```
public List<Student> getAllStudents() {
    ArrayList<Student> list = new ArrayList<Student>();
    Connection con = Database.getInstance().getConnection();
    PreparedStatement stmt = null;
    ResultSet rs = null;
    try {
        stmt = con.prepareStatement("select * from student");
        rs = stmt.executeQuery();
        while(rs.next()){
            Student st = revertRowToStudent(rs);
            list.add(st);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }finally {
        close(rs, stmt);
    }
    return list;
}
```


insert sql statement

```
public void addStudent(Student student) {
    Connection con = Database.getInstance().getConnection();
    PreparedStatement stmt = null;
    try{
        stmt = con.prepareStatement("insert into student values(?, ?, ?, ?, ?, ?)");
        stmt.setString(1, student.getStudentID());
        stmt.setString(2, student.getLastName());
        stmt.setString(3, student.getFirstName());
        stmt.setString(4, student.getEmail());
        stmt.setString(5, student.getClassID());
        stmt.setFloat(6, student.getAvgScore());
        stmt.executeUpdate();
    }catch(SQLException ex){
        ex.printStackTrace();
    }finally {
        close(stmt);
    }
}
```


Update sql statement

```
public void updateStudent(Student student) {
    Connection con = Database.getInstance().getConnection();
    PreparedStatement stmt = null;
    String sql = "update student "
            + "set lastname = ?, "
            + "firstname = ?, "
            + "email = ?, "
            + "classid = ?, "
            + "avgscore = ? "
            + "where studentid = ?";
    try {
        stmt = con.prepareStatement(sql);
        stmt.setString(1, student.getLastName());
        stmt.setString(2, student.getFirstName());
        stmt.setString(3, student.getEmail());
        stmt.setString(4, student.getClassID());
        stmt.setFloat(5, student.getAvgScore());
        stmt.setString(6, student.getStudentID());
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }finally {
        close(stmt);
    }
}
```


delete sql statement

```
public void deleteStudent(String studentID) {
    Connection con = Database.getInstance().getConnection();
    PreparedStatement stmt = null;
    String sql = "delete from student where studentid = ?";
    try {
        stmt = con.prepareStatement(sql);
        stmt.setString(1, studentID);
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }finally {
        close(stmt);
    }
}
```

Exam sql statement with parameters

```
public Student getStudent(String studentID) {
    Connection con = Database.getInstance().getConnection();
    PreparedStatement stmt = null;
    ResultSet rs = null;
    Student st = null;
    String sql = "select * from student where studentid = ?";
    try {
        stmt = con.prepareStatement(sql);
        stmt.setString(1, studentID);
        rs = stmt.executeQuery();
        if(rs.next())
            st = revertRowToStudent(rs);
    } catch (SQLException e) {
        e.printStackTrace();
    }finally {
        close(rs, stmt);
    }
    return st;
}
```


Advanced JDBC programming

Part 3

CallableStatement

Introduction

- A CallableStatement object provides a way to call stored procedures in a standard way for all RDBMSs.
- This call is written in an escape syntax that may take one of two forms:
  - One form with a result parameter.
  - And the other without one.
- Both forms may have a variable number of parameters used for input (IN parameters), output (OUT parameters), or both (INOUT parameters).

CallableStatement

Syntax for calling stored procedure

- Syntax for calling a stored procedure in JDBC is:
  - `{call procedure_name[(?, ?, ...)]}`
- Stored procedure that returns a result parameter is:
  - `{? = call procedure_name[(?, ?, ...)]}`
- Stored procedure without parameter:
  - `{call <procedure_name>}`
- Creating a CallableStatement Object :

```
CallableStatement cstmt = con.prepareCall("{call
            procedureName(?, ?)}");
```

  - Note: The "?" placeholders are IN, OUT, or INOUT parameters depends on the stored procedure procedureName.

CallableStatement

IN parameters

- Passing in any IN parameter values to a CallableStatement object is done using the setXXX methods inherited from PreparedStatement.
- The type of the value being passed in determines which setXXX method to use.
- For example:
  - CallableStatement cs=con.prepareCall ("{call myProc(?)}");
  - cs.setString(1,"cntt");

49

CallableStatement

OUT parameters

- If the stored procedure returns OUT parameters, the JDBC type of each OUT parameter must be registered before the CallableStatement object can be executed .
- Registering the JDBC type is done with the method registerOutParameter.
- Then after the statement has been executed, CallableStatement's getXXX() methods can be used to retrieve OUT parameter values.

```
CallableStatement cstmt = con.prepareCall("{call getTestData(?)}");
cstmt.registerOutParameter(1,java.sql.Types.TINYINT);
ResultSet rs = cstmt.executeQuery();
// . . . retrieve result set values with rs.getXXX methods
byte x = cstmt.getByte(1);
//do yout works
```

50

CallableStatement

INOUT parameters

- A parameter that supplies input as well as accepts output (an INOUT parameter) requires a call to the appropriate setXXX method (inherited from PreparedStatement) in addition to a call to the method registerOutParameter.
- The setXXX method sets a parameter's value as an input parameter, and the method registerOutParameter registers its JDBC type as an output parameter.

```
CallableStatement cstmt = con.prepareCall( "{call reviseTotal(?)}");
cstmt.setInt(1, (byte) 25); //IN
cstmt.registerOutParameter(1,java.sql.Types.INTEGER);
cstmt.execute();
byte x = cstmt.getByte(1);//OUT
```

51

Transaction

Introduction

- The major reason for grouping commands into transactions is database integrity.
- If you group updates to a transaction, then the transaction either succeeds in its entirety and it can be committed, or it fails somewhere in the middle. In that case, you can carry out a rollback and the database automatically undoes the effect of all updates that occurred since the last committed transaction.
- By default, a database connection is in *autocommit* mode, and each SQL command is committed to the database as soon as it is executed. Once a command is committed, you cannot roll it back.
- To check the current autocommit mode setting, call the getAutoCommit() method of the Connection class.

52

Transaction

Implementing transaction

1. You turn off autocommit mode with the command:

```
conn.setAutoCommit(false);
```

2. Now you create a statement object in the normal way:

```
Statement stat = conn.createStatement();
```

3. Call executeUpdate any number of times.
4. When all commands have been executed, call the commit method:

```
conn.commit();
```

5. However, if an error occurred, call :

```
conn.rollback();
```


Transaction

Implementing transaction with batch update

- For proper error handling in batch mode, you want to treat the batch execution as a single transaction. If a batch fails in the middle, you want to roll back to the state before the beginning of the batch.

```
boolean autoCommit = con.getAutoCommit();
try{
     con.setAutoCommit(false);
     Statement stat = con.createStatement()
     //. . .
     // keep calling stat.addBatch(. . .);
     //. . .
     stat.executeBatch();
     con.commit();
}catch(SQLException sqlEx){
     con.rollback();
}
finally{
     con.setAutoCommit(autoCommit);
}
```


Metadata

- JDBC can give you additional information about the structure of a database and its tables. For example, you can get a list of the tables in a particular database or the column names and types of a table
- In SQL, data that describes the database or one of its parts is called metadata (to distinguish it from the actual data that is stored in the database). You can get two kinds of metadata: about a database and about a result set.

Metadata

Getting Information about a Result Set

- When you send a SELECT statement using JDBC, you get back a ResultSet object containing the data that satisfied your criteria. You can get information about this ResultSet object by creating a ResultSetMetaData object and invoking ResultSetMetaData methods on it.
- Ex:

```
ResultSet rs = stmt.executeQuery("select * from tblClass");
ResultSetMetaData rsmd = rs.getMetaData() ;

int numCols = rsmd.getColumnCount();
for (int i = 1; i<=numCols; i++){
    System.out.println(
            "Column Name:"+rsmd.getColumnName(i)+
            " Type:"+rsmd.getColumnTypeName(i)+
            " Nullable:"+rsmd.isNullable(i)
            );
}
```

Metadata

Getting Information about a Database or Database System

- Once you have an open connection with a DBMS, you can create a DatabaseMetaData object that contains information about that database system. Using the Connection object con, the following line of code creates the DatabaseMetaData object dbmd:
  - `DatabaseMetaData dbmd = con.getMetaData();`
- So you can use it's method to get all tables, store procedure, view,… from the database.
- Example

```
DatabaseMetaData dbmd=con.getMetaData();
String[]tblType={"TABLE","VIEW"};
ResultSet rs=dbmd.getTables(null,null,null,tblType);
```

57

Row Sets

- The RowSet interface extends the ResultSet interface, but row sets don't have to be tied to a database connection.
- The javax.sql.rowset package provides the following interfaces that extend the RowSet interface:
  - CachedRowSet
  - WebRowSet
  - FilteredRowSet and JoinRowSet
  - JdbcRowSet

58

Row Sets

CachedRowSet (1)

- A CachedRowSet allows disconnected operation.
- A cached row set contains all data from a result set.
- You can close the connection and still use the row set.
- Each user command simply opens the database connection, issues a query, puts the result in a row set, and then closes the database connection.
- You can modify the data in a cached row set.
- The modifications are not immediately reflected in the database.
- You need to make an explicit request to accept the accumulated changes.
- The CachedRowSet reconnects to the database and issues SQL commands to write the accumulated changes.

59

Row Sets

CachedRowSet (2)

- Cached row sets are not appropriate for large query results.
- You can populate a CachedRowSet from a result set:

```
ResultSet rs=stmt.executeQuery(sql);
CachedRowSet rowset=new com.sun.rowset.CachedRowSetImpl();
rowset.populate(rs);
con.close();//now ok to close database connection
```

- If you modified the row set contents, you must write it back to the database by calling
  - `rowset.acceptChanges(con);`
- A row set that contains the result of a complex query will not be able to write back changes to the database.
- You should be safe if your row set contains data from a single table.

60

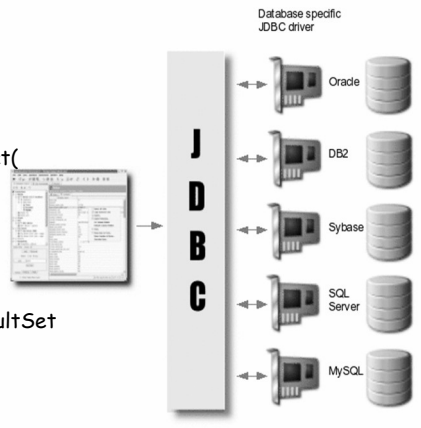
Row Sets

Other Row sets

- A WebRowSet is a cached row set that can be saved to an XML file. The XML file can be moved to another tier of a web application, where it is opened by another WebRowSet object.
- The FilteredRowSet and JoinRowSet interfaces support lightweight operations on row sets that are equivalent to SQL SELECT and JOIN operations. These operations are carried out on the data stored in row sets, without having to make a database connection.
- A JdbcRowSet is a thin wrapper around a ResultSet. It adds useful getters and setters from the RowSet interface, turning a result set into a "bean."

Summary

- Connect to database
- JDBC driver
- Manipulating with JDBC object(Connection, Statement, CallableStatement, ResultSet...)
- Scrollable and Updatable ResultSet
- Transaction
- Metadata
- Row sets



FAQ

That's all for this session!

Thank you all for your attention and patient !