

# Bài giảng JavaCore tiếng Việt

*Giảng viên:* Trần Trung Hiếu

*Tài liệu liên quan xem tại:* <http://fita.hua.edu.vn/tthieu/>

<b>CHƯƠNG 1</b>	<b>TỔNG QUAN NGÔN NGỮ LẬP TRÌNH JAVA</b>	<b>4</b>
1.1	LỊCH SỬ RA ĐỜI CỦA JAVA	4
1.2	MỘT SỐ ĐẶC TÍNH CỦA JAVA	4
1.3	CÔNG NGHỆ JAVA	7
1.3.1	Giới thiệu	7
1.3.2	Các dạng công nghệ Java	7
1.3.3	Các dạng ứng dụng của Java	7
1.3.4	Bộ công cụ phát triển ứng dụng Java	9
1.3.5	Máy ảo Java	10
1.3.5.1	Quy trình biên dịch, thông dịch của Java	10
1.3.5.2	Máy ảo Java	11
1.3.6	Môi trường phát triển tích hợp	11
1.4	HƯỚNG DẪN CÀI ĐẶT	12
<b>CHƯƠNG 2</b>	<b>CÁC CẤU TRÚC LẬP TRÌNH CĂN BẢN TRONG JAVA</b>	<b>14</b>
2.1	VIẾT MỘT CHƯƠNG TRÌNH JAVA ĐƠN GIẢN	14
2.2	CẤU TRÚC MỘT CHƯƠNG TRÌNH JAVA CƠ BẢN	15
2.2.1	Kiến trúc của Java	15
2.2.2	Các bước phát triển một chương trình Java	16
2.2.3	Cấu trúc một chương trình cơ bản	16
2.3	HÀNG, BIẾN, KIỂU DỮ LIỆU, TOÁN TỬ	17
2.3.1	Từ khóa	17
2.3.2	Định danh (identifier)	17
2.3.3	Biến (Variable)	17
2.3.4	Các kiểu dữ liệu (data types)	18
2.3.4.1	Kiểu dữ liệu cơ sở	19
2.3.4.2	Ép kiểu	20
2.3.5	Hằng	21
2.3.6	Toán tử	21
2.4	CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG JAVA	23
2.4.1	Mệnh đề if-else	24
2.4.2	Mệnh đề switch-case	25
2.4.3	Vòng lặp for	26
2.4.4	Vòng lặp while	26
2.4.5	Vòng lặp do-while	26
2.5	NHẬP DỮ LIỆU TỪ BÀN PHÍM	27
2.6	BÀI TẬP CHƯƠNG 2	28
<b>CHƯƠNG 3</b>	<b>LỚP VÀ ĐỐI TƯỢNG</b>	<b>29</b>
3.1	KHÁI NIỆM	29
3.2	KHAI BÁO LỚP	29
3.3	THUỘC TÍNH CỦA LỚP	30
3.4	PHƯƠNG THỨC CỦA LỚP	33
3.5	CHI ĐỊNH TRUY XUẤT LỚP	35
3.6	TẠO ĐỐI TƯỢNG	36
3.6.1	Constructor	36
3.6.2	Biến this	37
3.7	GÓI – PACKAGE	38
3.7.1	Package	38
3.7.2	Class importation	38
3.7.3	Quy cách đặt tên	40
3.8	BÀI TẬP CHƯƠNG 3	40
<b>CHƯƠNG 4</b>	<b>ĐẶC ĐIỂM HƯỚNG ĐỐI TƯỢNG TRONG JAVA</b>	<b>40</b>

4.1	TÍNH ĐÓNG GÓI .....	40
4.2	TÍNH KẾ THỪA.....	43
4.2.1	Tính kế thừa.....	43
4.2.2	Nạp chồng phương thức .....	45
4.2.3	Ghi đè phương thức .....	45
4.2.4	Trường ẩn.....	48
4.2.5	Class Object .....	49
4.3	TÍNH ĐA HÌNH .....	49
4.4	LỚP TRƯU TƯỢNG, LỚP VÔ SINH & GIAO TIẾP .....	53
4.4.1	Lớp trừu tượng – Abstract class .....	53
4.4.2	Lớp vô sinh – Final class .....	54
4.4.3	Giao tiếp - Interface.....	54
4.5	BÀI TẬP CHƯƠNG 4 .....	58
<b>CHƯƠNG 5</b>	<b>MẢNG VÀ XÂU.....</b>	<b>58</b>
5.1	MẢNG .....	58
5.2	XÂU .....	60
5.3	BÀI TẬP CHƯƠNG 5 .....	61
<b>CHƯƠNG 6</b>	<b>CĂN BẢN VỀ LẬP TRÌNH GIAO DIỆN .....</b>	<b>61</b>
6.1	CĂN BẢN VỀ LẬP TRÌNH GIAO DIỆN .....	61
6.1.1	Giới thiệu thiết kế GUI trong Java .....	61
6.1.2	Các thành phần cơ bản – Components.....	62
6.1.2.1	Nhãn – Label .....	63
6.1.2.2	Nút nhấn – Button.....	64
6.1.2.3	Ô văn bản – Text Field.....	65
6.1.2.4	Lựa chọn – Choice.....	67
6.1.3	Đối tượng khung chứa – Container .....	68
6.1.3.1	Frame.....	68
6.1.3.2	Panel.....	70
6.1.4	Bộ quản lý trình bày – Layout Manager .....	71
6.1.4.1	Flow Layout .....	71
6.1.4.2	Border Layout.....	72
6.1.4.3	Grid Layout.....	73
6.1.4.4	GridBag Layout.....	74
6.1.4.5	Null Layout.....	76
6.2	APPLET.....	78
6.2.1	Cơ bản về Applet .....	78
6.2.2	Chuyển từ một ứng dụng đồ họa sang Applet.....	80
6.2.3	Đối tượng đồ họa Graphics.....	81
6.3	BÀI TẬP CHƯƠNG 6 .....	85
<b>CHƯƠNG 7</b>	<b>THAM KHẢO THÊM .....</b>	<b>85</b>
7.1	SWING .....	85
7.2	XỬ LÝ SỰ KIỆN TRONG LẬP TRÌNH GIAO DIỆN .....	85
7.3	XỬ LÝ NGOẠI LỆ.....	85
7.4	LẬP TRÌNH CƠ SỞ DỮ LIỆU .....	85
7.5	BÀI TẬP CHƯƠNG 7 .....	85

# CHƯƠNG 1 TỔNG QUAN NGÔN NGỮ LẬP TRÌNH JAVA

## 1.1 Lịch sử ra đời của Java

Cuối năm 1990, James Gosling và các cộng sự được công ty Sun Microsystems giao nhiệm vụ xây dựng phần mềm lập trình cho các mặt hàng điện tử dân dụng nhằm mục đích cài chương trình vào các bộ xử lý của các thiết bị như VCR, lò nướng, PDA (personal data assistant).

Lúc đầu Gosling và các cộng sự định chọn ngôn ngữ C++ nhưng thấy rằng C++ có những hạn chế. Chương trình viết bằng C++ khi chuyển sang chạy trên một hệ thống máy có bộ vi xử lý khác thì đòi hỏi phải biên dịch lại. Gosling quyết định xây dựng hẳn một ngôn ngữ mới dựa trên nền ngôn ngữ C, C++ và đặt tên là Oak (cây sồi, vì phòng làm việc của Gosling nhìn ra một cây sồi).

Oak đòi hỏi phải độc lập cấu trúc nền (phần cứng, OS) do thiết bị có thể do nhiều nhà sản xuất khác nhau (Platform independent). 1993, Internet và Web bùng nổ, Sun chuyển Oak thành một môi trường lập trình Internet với tên dự án là Java. 1995: Oak đổi tên với tên chính thức là Java. Java là tên một hòn đảo có trồng nhiều cà phê mà nhóm nghiên cứu phát triển đã tham quan và làm việc.

Mục đích của Java để phát triển ứng dụng cho các thiết bị điện tử thông minh, để tạo các trang web có nội dung động (applet). Hiện nay Java được sử dụng để phát triển nhiều loại ứng dụng khác nhau như cơ sở dữ liệu, mạng, Internet, games, viễn thông,...

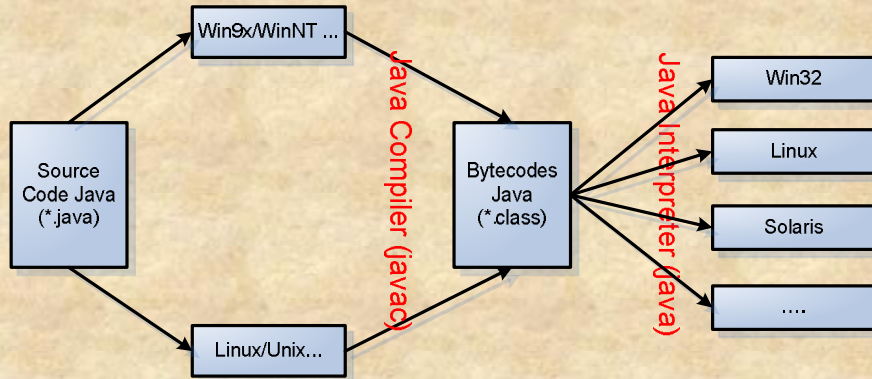
## 1.2 Một số đặc tính của Java

**Đơn giản (simple)**: Java tương tự như C++ nhưng bỏ bớt các đặc tính phức tạp của C++ như quản lý bộ nhớ, pointer, overload toán tử, không dùng include, bỏ struct, union. Java được kế thừa từ C++, và được loại bỏ đi các tính năng khó nhất của C++ nên java dễ sử dụng hơn

**Tính khả chuyển (portable)**: của java do chương trình biên dịch tạo ra mã byte (bytecodes) không phụ thuộc hệ thống máy sử dụng. Bytecodes là tập hợp các câu lệnh tương tự như lệnh mã máy (machine code), nó được tạo ra khi một chương trình Java được biên dịch xong

## MỘT SỐ ĐẶC TÍNH CỦA JAVA

Minh họa tính khả chuyển



9

### Tính hướng đối tượng (OO):

- Hướng đối tượng trong Java tương tự như C++ nhưng Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn, không thể viết một ứng dụng hướng thủ tục trong Java
- Tất cả mọi thứ đề cập đến trong Java đều liên quan đến các đối tượng được định nghĩa trước, thậm chí hàm chính (hàm main) của một chương trình viết bằng Java cũng phải đặt bên trong một lớp.
- Hướng đối tượng trong Java không có tính đa kế thừa (multi inheritance) như trong C++ mà thay vào đó Java đưa ra khái niệm interface để hỗ trợ tính đa kế thừa.

**Phân tán (distributed):** nhằm đến phân bố ứng dụng trên mạng, ứng dụng độc lập platform. Cụ thể là Java có hỗ trợ công nghệ lập trình RMI, CORBA, JavaBean. Các công nghệ này cho phép sử dụng lại các lớp đã tạo ra, triệu gọi các phương thức (method) hoặc các đối tượng từ một máy ở xa.

**Đa tiến trình (multithread):** đặc tính này của Java cho phép tạo nhiều tiến trình, tiến trình có thể chạy song song cùng một thời điểm và có thể tương tác với nhau.

**Tính an toàn (secure):** Kiểm tra an toàn code trước khi thực thi, có nhiều mức kiểm tra bảo mật ở Môi trường thực thi an toàn

- Mức 1: Mức ngôn ngữ, nhờ tính bao gói dữ liệu của OOP, không cho phép truy cập trực tiếp bộ nhớ mà phải thông qua method.

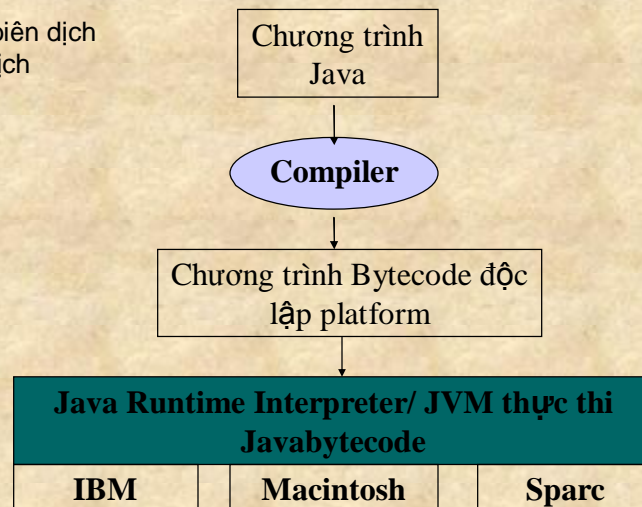
- Mức 2: Mức Compiler, kiểm tra an toàn cho code trước khi biên dịch.
- Mức 3: Mức Interpreter, trước khi bytecode được thực thi, được kiểm tra an toàn.
- Mức 4: Mức Class, các class trước khi nạp được kiểm tra an toàn.

#### **Biên dịch và thông dịch:**

- Java là một ngôn ngữ lập trình có khả năng biên dịch và khả năng thông dịch.
- Chương trình nguồn viết bằng ngôn ngữ lập trình Java có đuôi \*.java đầu tiên được biên dịch thành tập tin có đuôi \*.class và sau đó sẽ được trình thông dịch thông dịch thành mã máy.
- Java class file có thể được dùng ở bất kỳ platform nào (Write Once Run Anywhere).
- Các file tài nguyên → trình biên dịch javac → class file độc lập thiết bị
- Class file → trình thông dịch java → mã máy thực thi, không cần liên kết (link)

### **MỘT SỐ ĐẶC TÍNH CỦA JAVA**

Minh họa biên dịch và thông dịch



14

#### **Giải phóng bộ nhớ (Garbage Collection)**

- Java cung cấp một tiến trình mức hệ thống để theo dõi việc cấp phát bộ nhớ
- Garbage Collection

- Đánh dấu và giải phóng các vùng nhớ không còn được sử dụng.
- Được tiến hành tự động.
- Cơ chế hoạt động phụ thuộc vào các phiên bản máy ảo.

## **1.3 Công nghệ Java**

### **1.3.1 Giới thiệu**

Công nghệ Java phát triển mạnh mẽ nhờ vào Sun Microsystem cung cấp nhiều công cụ, thư viện lập trình phong phú hỗ trợ cho việc phát triển nhiều loại ứng dụng khác nhau. Java bao gồm:

- Ngôn ngữ lập trình
- Môi trường phát triển
- Môi trường thực thi và triển khai

### **1.3.2 Các dạng công nghệ Java**

#### **Desktop applications - J2SE**

- Phiên bản chuẩn – Java 2 Standard Edition. J2SE hỗ trợ viết các ứng dụng đơn, ứng dụng client-server.
- Java Applications: ứng dụng Java thông thường trên desktop
- Java Applets: ứng dụng nhúng hoạt động trong trình duyệt web

#### **Server applications - J2EE**

- Nền tảng Java 2, phiên bản doanh nghiệp - Java 2 Enterprise Edition. Hỗ trợ phát triển các ứng dụng thương mại.
- Chạy trên máy chủ lớn với sức mạnh xử lý và dung lượng bộ nhớ lớn, hỗ trợ gắn liền với servlet, jsp và XML.

#### **Mobile (embedded) applications – J2ME**

- Phiên bản thu nhỏ - Java 2 Micro Edition.
- Hỗ trợ viết các ứng dụng trên các thiết bị di động, không dây, thiết bị nhúng,...

### **1.3.3 Các dạng ứng dụng của Java**

Dùng Java ta có thể viết các dạng ứng dụng sau:

- Ứng dụng độc lập (console application)
- Ứng dụng Applets
- Ứng dụng giao diện (GUI application)
- Ứng dụng Web (Servlet và Jsp)
- Ứng dụng nhúng (embedded application)
- Ứng dụng cơ sở dữ liệu (database application)
- Games.

### **Ứng dụng độc lập (console application):**

- Ứng dụng Console là ứng dụng nhập xuất ở chế độ văn bản tương tự như màn hình Console của hệ điều hành MS-DOS.
- Loại chương trình ứng dụng này thích hợp với những ai bước đầu làm quen với ngôn ngữ lập trình java.
- Các ứng dụng kiểu Console thường được dùng để minh họa các ví dụ cơ bản liên quan đến cú pháp ngôn ngữ, các thuật toán, và các chương trình ứng dụng không cần thiết đến giao diện người dùng đồ họa.

### **Ứng dụng Applets:**

- Java Applet là loại ứng dụng có thể nhúng và chạy trong trang web của một trình duyệt web.
- Từ khi internet mới ra đời, Java Applet cung cấp một khả năng lập trình mạnh mẽ cho các trang web.
- Nhưng gần đây khi các chương trình duyệt web đã phát triển với khả năng lập trình bằng VB Script, Java Script, HTML, DHTML, XML,... cùng với sự cạnh tranh khốc liệt giữa Microsoft và Sun đã làm cho Java Applet lu mờ. Và cho đến bây giờ gần như các lập trình viên đều không còn “mặn mà” với Java Applet nữa.

### **Ứng dụng giao diện (GUI application):**

- Việc phát triển các chương trình ứng dụng có giao diện người dùng đồ họa trực quan giống như những chương trình được viết dùng ngôn ngữ lập trình VC++ hay Visual Basic đã được java giải quyết bằng thư viện AWT và JFC.
- JFC (Swing) là thư viện rất phong phú và hỗ trợ mạnh mẽ hơn nhiều so với AWT. JFC giúp cho người lập trình có thể tạo ra một giao diện trực quan của bất kỳ ứng dụng nào

### **Ứng dụng Web:**

- Java hỗ trợ mạnh mẽ đối với việc phát triển các ứng dụng Web thông qua công nghệ J2EE (Java 2 Enterprise Edition).
- Công nghệ J2EE hoàn toàn có thể tạo ra các ứng dụng Web một cách hiệu quả không thua kém công nghệ .NET mà Microsoft đang quảng cáo.
- Công nghệ viết web hiện có của Java là Servlet và Jsp, ngoài ra còn có sự hỗ trợ của lập trình Socket, Java Bean, RMI và CORBA, EJB.

### **Ứng dụng nhúng:**



- Java Sun đưa ra công nghệ J2ME (The Java 2 Platform, Micro Edition J2ME) hỗ trợ phát triển các chương trình, phần mềm nhúng.
- J2ME cung cấp một môi trường cho những chương trình ứng dụng có thể chạy được trên các thiết bị cá nhân như: điện thoại di động (MIDlet), máy tính bỏ túi PDA hay Palm, cũng như các thiết bị nhúng khác.

#### **Ứng dụng cơ sở dữ liệu:**

- Java cũng hỗ trợ lập trình kết nối và tương tác được với hầu hết các hệ quản trị CSDL nổi tiếng như Oracle, SQL Server, MS-Access, MySQL,...

#### **Games:**

- Lập trình Games bằng Java được phát triển mạnh mẽ. Dùng Java có thể viết được games cho máy desktop và các thiết bị di động.

### **1.3.4 Bộ công cụ phát triển ứng dụng Java**

**JDK (Java Development Kit):** Bộ công cụ phát triển ứng dụng Java bao gồm 4 thành phần: ClassS, Compiler, Debugger, Java Runtime Environment.

- |                |      |
|----------------|------|
| - JDK 1.0      | 1996 |
| - JDK 1.1      | 1997 |
| - JDK 1.2      | 1998 |
| - JDK 1.3      | 2000 |
| - Java 1.4     | 2002 |
| - Java 5 (1.5) | 2004 |
| - Java 6       | 2006 |

#### **Bao gồm:**

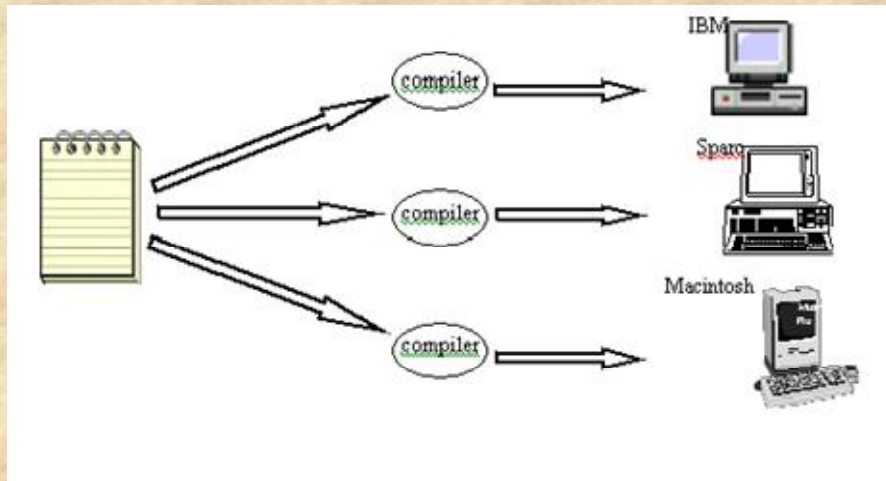
- **javac** Chương trình dịch chuyển mã nguồn sang bytecode
- **java** Bộ thông dịch: Thực thi java application
- **appletviewer** Bộ thông dịch: Thực thi java applet mà không cần sử dụng trình duyệt như Netscape, FireFox hay IE, v.v.
- **javadoc** Bộ tạo tài liệu dạng HTML từ mã nguồn và chú thích
- **jdb** Bộ gỡ lỗi (java debugger)
- **javap** Trình dịch ngược bytecode
- **jar** Dùng để đóng gói lưu trữ các module viết bằng Java (tạo ra file đuôi .jar), là phương pháp tiện lợi để phân phối những chương trình Java.

### 1.3.5 Máy ảo Java

#### 1.3.5.1 Quy trình biên dịch, thông dịch của Java

## JVM – JAVA VIRTUAL MACHINE

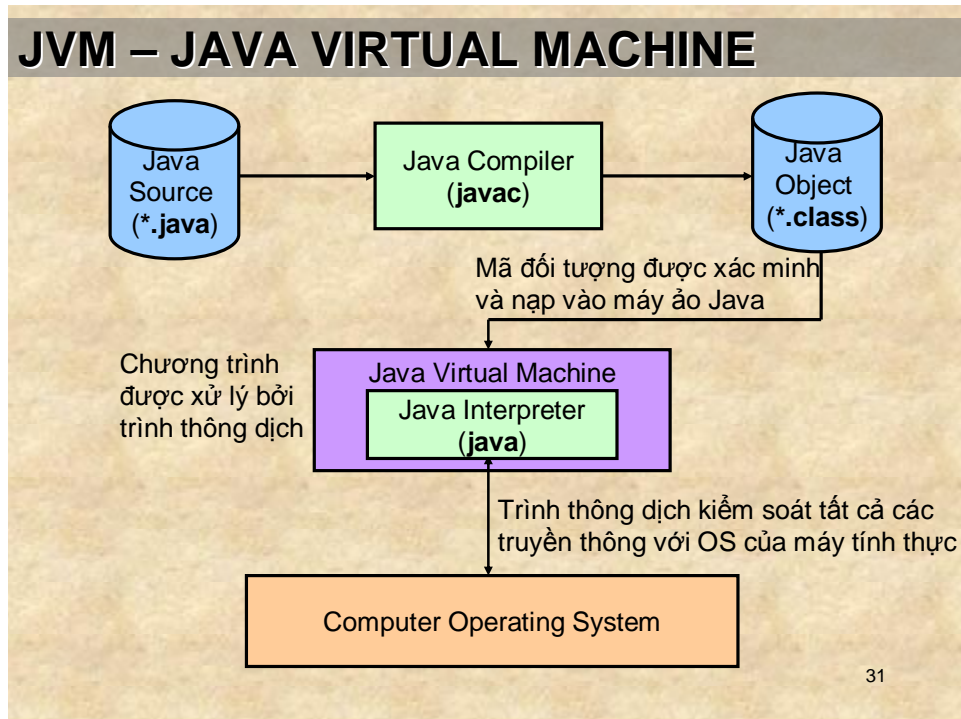
Kiểu dịch của các trình biên dịch ngôn ngữ cũ



29

- Trình biên dịch chuyển mã nguồn thành tập các lệnh không phụ thuộc vào phần cứng cụ thể.
- Trình thông dịch trên mỗi máy chuyển tập lệnh này thành chương trình thực thi
- Máy ảo tạo ra một môi trường để thực thi các lệnh bằng cách:
- Nạp các file .class
- Quản lý bộ nhớ
- Dọn “rác”

### 1.3.5.2 Máy ảo Java



- JVM là một phần mềm dựa trên cơ sở máy tính ảo. JVM cũng được xem như là một hệ điều hành thu nhỏ.
- Máy ảo phụ thuộc vào Platform (phần cứng, OS), nó cung cấp môi trường thực thi cho Java (độc lập với platform).
- Nó thiết lập cho các mã Java đã biên dịch có một cái nhìn trong suốt (trasparence) về các phần cứng bên dưới.

### 1.3.6 Môi trường phát triển tích hợp

IDE (integrated development environment): trong phần mềm máy tính, IDE đề chỉ đến một bộ các công cụ phần mềm để soạn thảo, biên dịch, liên kết, gỡ rối, v...v... Ví dụ như bộ Visual Studio của Microsoft.

IDE giúp phát triển ứng dụng nhanh chóng và hiệu quả hơn. Đơn giản hóa quá trình phát triển phần mềm

Một số IDE dành cho lập trình Java là:

- JCreator
- NetBeans
- Eclipse
- TextPad
- BlueJ
- Java Studio của Sun
- ...

## **1.4 Hướng dẫn cài đặt**

### **Các phần mềm cần chuẩn bị:**

- JDK 1.6 (jdk-6u11-windows-i586-p)
- JDK DOC 1.6 (jdk-6u10-docs) (click vào index.html để tra cứu thư viện Java)  
<http://java.sun.com/javase/downloads/index.jsp>
- BlueJ version 3.0.1 (Có sơ đồ UML sinh tự động)  
<http://www.bluej.org/download/download.html>
- Nếu không dùng một IDE như BlueJ để soạn thảo, dịch và thực thi thì ta có thể dùng trình soạn thảo văn bản như NotePad để soạn thảo

### **Trình tự cài đặt**

- Cài JDK 1.6
- Thiết lập biến môi trường PATH
- BlueJ 3.0.1

**Cài đặt:** Phần này hướng dẫn cài đặt trên hệ điều hành window XP

- Tạo thư mục C:\Java để cài. Nếu không thì để JDK cài ngầm định vào C:\Program Files\Java\jdk1.6.0\_11
- Nhấp đúp vào source JDK 1.6 để cài.
- Thiết lập biến môi trường PATH chỉ đến thư mục BIN của thư mục mà ta mới cài JDK.



## CHƯƠNG 2 CÁC CẤU TRÚC LẬP TRÌNH CĂN BẢN TRONG JAVA

### 2.1 *Viết một chương trình Java đơn giản*

Gõ đoạn mã chương trình sau vào trình soạn thảo notepad:

```
public class HelloWorld{  
  
    public static void main(String[] args){  
        // Display "Hello World!" message  
        System.out.println("Hello World!");  
    }  
}
```

Lưu file: Lưu tên class trùng tên file, file có phần mở rộng là java à HelloWorld.java

à Lưu vào thư mục Java trên ổ D

Mở cmd của WindowsXP lên. Thực hiện 2 bước sau:

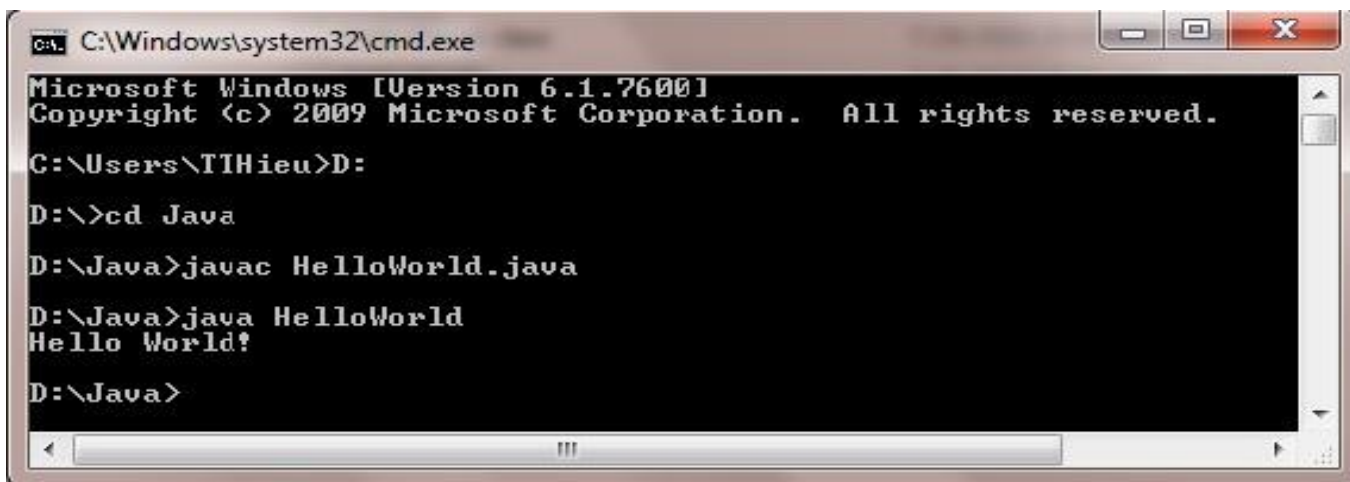
- Dịch file source code ra mã bytecode:

**javac HelloWorld.java**

Nếu dịch thành công một file HelloWorld.class xuất hiện chứa các mã bytecode

- Chạy chương trình bởi máy ảo

**java HelloWorld**



```
C:\Windows\system32\cmd.exe  
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
C:\Users\TTHieu>D:  
D:\>cd Java  
D:\Java>javac HelloWorld.java  
D:\Java>java HelloWorld  
Hello World!  
D:\Java>
```

Các bước chạy chương trình Java sử dụng NotePad và cmd của Window

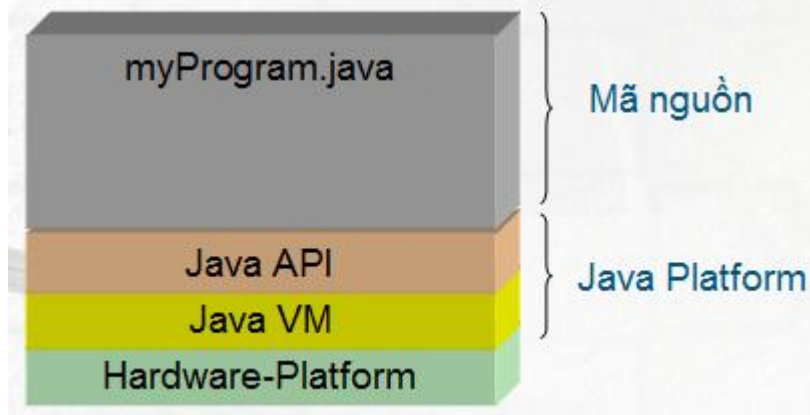
Chú ý: Bạn cần đặt biến môi trường path theo hướng dẫn trong chương 1 để cmd nhận dạng các lệnh javac, java.

## **2.2 Cấu trúc một chương trình java cơ bản**

### **2.2.1 Kiến trúc của Java**

#### **Java Platform**

- Java Virtual Machine (Java VM)
- Java Application Programming Interface (Java API)



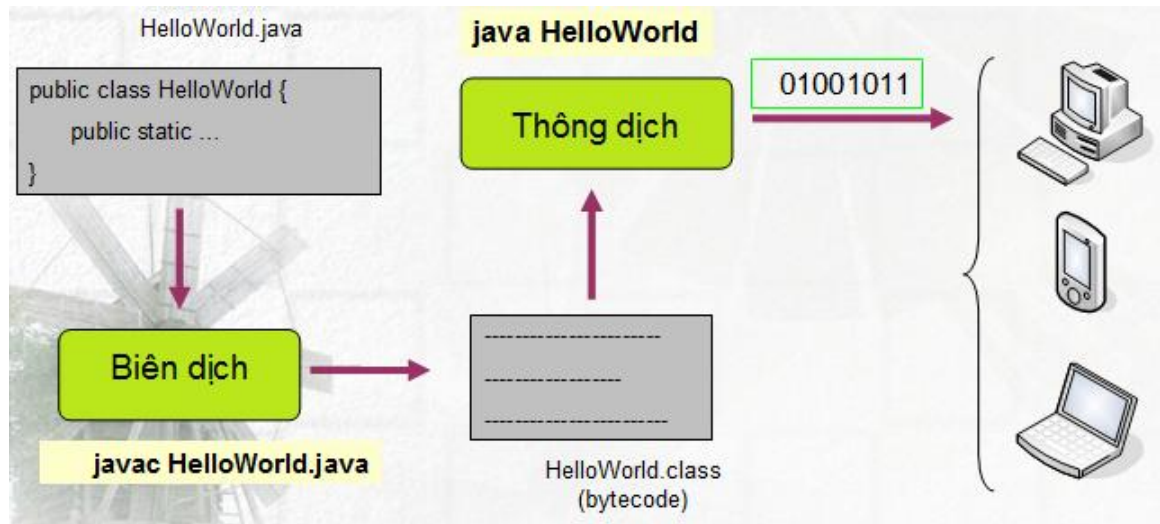
**Thư viện lớp Java:** bộ JDK bao gồm rất nhiều lớp chuẩn đã được xây dựng sẵn. Lập trình viên thường sử dụng các lớp chuẩn để phát triển ứng dụng.

#### **Các gói chuẩn của Java:**

- java.lang
- java.applet
- java.awt
- java.io
- java.util
- java.net
- java.awt.event
- java.rmi
- java.security
- java.sql



### 2.2.2 Các bước phát triển một chương trình Java



### 2.2.3 Cấu trúc một chương trình cơ bản

```
// Ten file: HelloWorld.java
/* Tac gia: Nguyen Van An */
```

```
public class HelloWorld {
    // Phương thức main, điểm bắt đầu của chương trình
    public static void main(String[] args){
        System.out.println("HelloWorld !");
    } // Kết thúc phương thức main

} // Kết thúc class HelloWorld
```

Chương trình trên bao gồm những điểm cần chú ý sau:

- Tên class chứa hàm main phải giống tên file (HelloWorld)
- Từ khóa khai báo lớp class, mỗi chương trình phải có ít nhất một khai báo lớp
- Phương thức main sẽ được gọi đầu tiên, mỗi chương trình thực thi phải có một phương thức main. Phương thức main có 3 bộ từ đặc tả sau:
  - o public: chỉ ra rằng phương thức main là công cộng có thể gọi bởi bất kỳ đối tượng nào
  - o static: chỉ ra rằng phương thức main là một phương thức lớp



- void: chỉ ra rằng phương thức main không trả lại bất kỳ giá trị nào
- Mở đầu kết thúc của class, phương thức (main) bằng các dấu mở, đóng ngoặc nhọn { }
- Các câu lệnh luôn kết thúc bằng dấu chấm phẩy (;)
- Các dấu chú thích của chương trình: //, /\*..\*/ , /\*\*...\*/
- Các từ khóa trong Java là case sensitive, bạn không được tùy tiện thay đổi giữa dạng chữ hoa và thường: public, class, static, void, String, System, ...

## 2.3 *Hằng, biến, kiểu dữ liệu, toán tử*

### 2.3.1 Từ khóa

- Từ khóa cho các kiểu dữ liệu cơ bản : **byte, short, int, long, float, double, char, boolean.**
- Từ khóa cho phát biểu lặp: **do, while, for, break, continue.**
- Từ khóa cho phát biểu rẽ nhánh: **if, else, switch, case, default, break.**
- Từ khóa đặc tả đặc tính một method: **private, public, protected, final, static, abstract, synchronized.**
- Hằng (literal): **true, false, null.**
- Từ khóa liên quan đến method: **return, void.**
- Từ khóa liên quan đến package: **package, import.**
- Từ khóa cho việc quản lý lỗi: **try, catch, finally, throw, throws.**
- Từ khóa liên quan đến đối tượng: **new, extends, implements, class, instanceof, this, super**

### 2.3.2 Định danh (indentifier)

- Định danh là dùng biểu diễn tên của biến, của phương thức, của lớp.
- Trong Java, định danh có thể sử dụng ký tự chữ, ký tự số và ký tự dấu.
- Ký tự đầu tiên phải là ký tự chữ, dấu gạch dưới (\_), hoặc dấu dollar (\$).
- Có sự phân biệt giữa ký tự chữ hoa và chữ thường.  
Ví dụ: Hello, \_prime, var8, tvLang

### 2.3.3 Biến (Variable)

- Biến là vùng nhớ dùng để lưu trữ các giá trị của chương trình.

- Mỗi biến gắn liền với một kiểu dữ liệu và một định danh duy nhất gọi là tên biến.
- Tên biến thông thường là một chuỗi các ký tự (Unicode), ký số.
- Tên biến phải bắt đầu bằng một chữ cái, một dấu gạch dưới hay dấu dollar.
- Tên biến không được trùng với các từ khóa (xem lại các từ khóa trong java).
- Tên biến không có khoảng trắng ở giữa tên.
- Trong java, biến có thể được khai báo ở bất kỳ nơi đâu trong chương trình

Cách khai báo

```
<kiểu_dữ_liệu> <tên_biến>;
<kiểu_dữ_liệu> <tên_biến> = <giá_trị>;
```

Gán giá trị cho biến

```
<tên_biến> = <giá_trị>;
```

Biến công cộng (toàn cục): là biến có thể truy xuất ở khắp nơi trong chương trình, thường được khai báo dùng từ khóa public, hoặc đặt chúng trong một class.

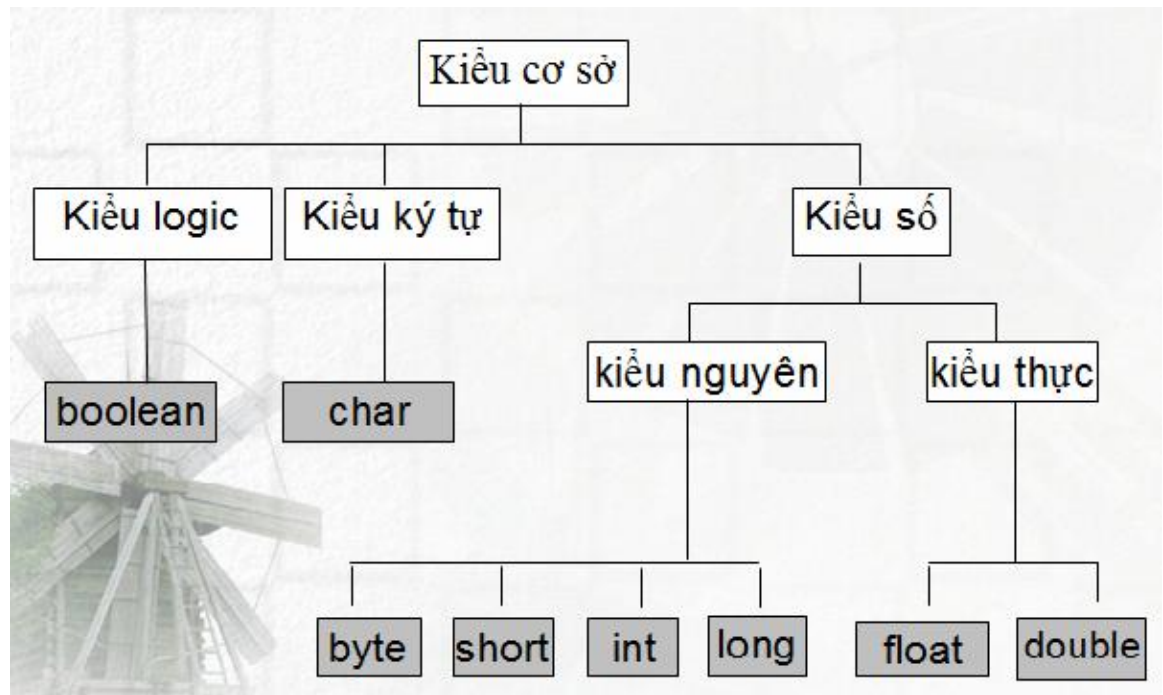
Biến cục bộ: là biến chỉ có thể truy xuất trong khối lệnh nó khai báo

### 2.3.4 Các kiểu dữ liệu (data types)

Java bao gồm các kiểu dữ liệu sau:

- Kiểu dữ liệu cơ sở (Primitive data type)
  - o Kiểu dữ liệu cơ sở của Java bao gồm các nhóm sau: số nguyên, số thực, ký tự, kiểu logic
- Kiểu dữ liệu tham chiếu hay dẫn xuất (reference data type)
  - o Kiểu dữ liệu tham chiếu là các kiểu dữ liệu đối tượng. Ví dụ như: String, Byte, Character, Double, Boolean, Integer, Long, Short, Font,... và các lớp do người dùng định nghĩa

### 2.3.4.1 Kiểu dữ liệu cơ sở



#### Kiểu số nguyên:

Kiểu	Kích thước	Khoảng giá trị
<b>byte</b>	8 bits	-128...127
<b>short</b>	16 bits	-32768...32767
<b>int</b>	32 bits	$-2^{31} \dots 2^{31} - 1$
<b>long</b>	64 bits	$-2^{63} \dots 2^{63} - 1$

Bốn kiểu số nguyên khác nhau là: byte, short, int, long

Kiểu mặc định của các số nguyên là kiểu int

Không có kiểu số nguyên không dấu

Không thể chuyển biến kiểu int và kiểu boolean như trong ngôn ngữ

C/C++

VD: `int x = 0;`  
`long y=100;`  
`int a=1,b,c;`

### Kiểu số thực:

Kiểu	Kích thước	Khoảng giá trị
<b>float</b>	32 bits	-3.4e38...3.4e38
<b>double</b>	64 bits	-1.7e308...1.7e308

### Kiểu boolean:

Nhận giá trị true hoặc false. Giá trị mặc định là false, không thể chuyển thành kiểu nguyên và ngược lại

### Kiểu char: Kiểu ký tự theo chuẩn Unicode

Biểu diễn các ký tự trong bộ mã Unicode

$2^{16} = 65536$  ký tự khác nhau :

từ '\u0000' đến '\uFFFF'

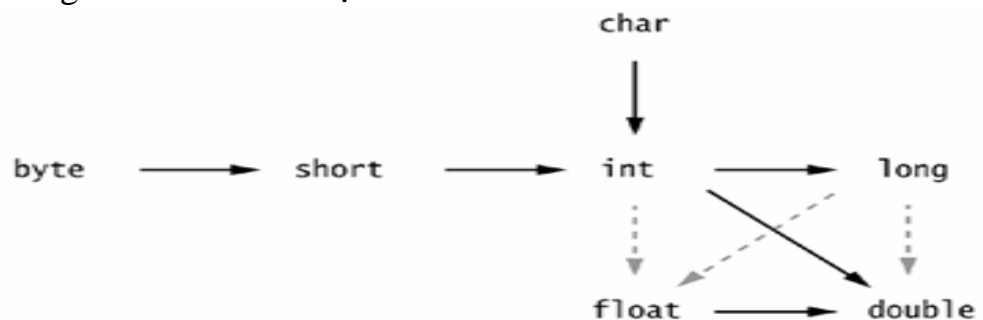
Một số hằng ký tự:

Ký tự	Ý nghĩa
\b	Xóa lùi (BackSpace)
\t	Tab
\n	Xuống hàng
\r	Dấu enter
\"	Nháy kép
\'	Nháy đơn
\\	Số ngược
\f	Đẩy trang
\uxxxx	Ký tự unicode

### 2.3.4.2 Ép kiểu

Khi gặp phải sự không tương thích kiểu dữ liệu chúng ta phải tiến hành chuyển đổi kiểu dữ liệu cho biến hoặc biểu thức

Chuyển đổi giữa các kiểu dữ liệu cơ sở:



Chú ý: Nếu chuyển kiểu theo chiều mũi tên nét đứt, kết quả nhận được có thể không chính xác

**Toán tử ép kiểu:**

<tên biến> = (kiểu\_dữ\_liệu) <tên\_biến>;

float fNum = 2.2;

int iCount = (int) fNum

**Ép kiểu rộng** (widening conversion): từ kiểu nhỏ sang kiểu lớn (không mất mát thông tin)

**Ép kiểu hẹp** (narrow conversion): từ kiểu lớn sang kiểu nhỏ (có khả năng mất mát thông tin)

### 2.3.5 Hằng

Hằng là một giá trị bất biến trong chương trình

Tên hằng được đặt theo qui ước giống như tên biến

Tiếp vĩ ngữ: ***l, L, f, F, d, D***

*int i=1;*

*long i=1L;*

**Hằng ký tự:** là một ký tự đơn nằm giữa 2 dấu nháy đơn.

**Hằng chuỗi:** là tập hợp các ký tự được đặt giữa hai dấu nháy kép “”. Một hằng chuỗi không có ký tự nào là một hằng chuỗi rỗng.

Ví dụ: “Hello World”

Lưu ý: Hằng chuỗi không phải là một kiểu dữ liệu cơ sở nhưng vẫn được khai báo và sử dụng trong các chương trình

### 2.3.6 Toán tử

**Toán tử số học:**

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia nguyên
%	Chia dư
++	Tăng 1
--	Giảm 1

**Toán tử quan hệ và logic:**

Toán tử	Ý nghĩa
==	So sánh bằng
!=	So sánh khác
>	So sánh lớn hơn
<	So sánh nhỏ hơn
>=	So sánh lớn hơn hay bằng
<=	So sánh nhỏ hơn hay bằng
	OR (biểu thức logic)
&&	AND (biểu thức logic)
!	NOT (biểu thức logic)

#### Toán tử gán:

Toán tử	Ví dụ	Giải thích
=	int c=3, d=5, c=4;	
+=	c+=7	c=c+7
-=	d-=4	d=d-4
*=	e*=5	e=e*5
/=	f/=3	f=f/3
%=	g%=9	g=g%9

#### Toán tử điều kiện:

<điều kiện> ? <biểu thức 1> : <biểu thức 2>

*int x = 10;*

*int y = 20;*

*int Z = (x<y) ? 30 : 40;*

#### Độ ưu tiên của các phép toán:

Độ ưu tiên của các phép toán trong ngôn ngữ Java cũng gần giống như ngôn ngữ C/C++. Thứ tự ưu tiên từ trái qua phải và từ trên xuống dưới như bảng sau:

1	.	[]	()	
2	++	--	!	~
3	*	/	%	
4	+	-		
5	<<	>>	>>>	
6	<	>	<=	>=
7	==	!=		
8	&			
9	^			
10	&&			
11				
12	?:			
13	=			

**Một ví dụ về phép toán:**

```
import java.lang.*;
import java.io.*;
class VariableDemo
{
    public static void main(String[] args)
    {
        int x = 10;
        int y = 20;
        int z = x+y;
        System.out.println("x = " + x);
        System.out.println("y = " + y);
        System.out.println("z = x + y =" + z);
        System.out.println("So nho hon la so:" +
                           Math.min(x,y));

        char c = 80;
        System.out.println("ky tu c la: " + c);
    }
}
```

## 2.4 Các cấu trúc điều khiển trong Java

Điều khiển rẽ nhánh:

- Mệnh đề **if-else**
- Mệnh đề **switch-case**

Vòng lặp (Loops):

- Vòng lặp **for**

- Vòng lặp **while**
- Vòng lặp **do-while**

### 2.4.1 Mệnh đề if-else

#### Dạng 1:

if (điều\_kiện) lệnh

```
if(điều_kiện)
    { // khoi lenh
    lệnh1
    lệnh2
    ...
    }
```

#### Dạng 2:

if (điều\_kiện) lệnh1 else lệnh2

#### Dạng 3:

```
if(điều_kiện){
}else if(điều_kiện){
}else if(điều_kiện){
}
...
}else{
}
```

**Ví dụ:** Chương trình kiểm tra xem ngày hiện tại có phải chủ nhật hay không

```
import java.util.Date;
public class TestIf {
    public static void main( String args[ ] ){
        Date today = new Date();
        if( today.getDay() == 0 ){
            System.out.println("Hom nay la chu nhat\n");
        }
        else{
            System.out.println("Hom nay khong la chu nhat\n" );
        }
    }
}
```



### 2.4.2 Mệnh đề switch-case

```
switch (choice)
{
    case 1:
        ...
        break;
    case 2:
        ...
        break;
    case 3:
        ...
        break;
    case 4:
        ...
        break;
    default:
        // bad input
        ...
        break;
}
```

Chú ý:

#### **Ví dụ:**

```
import javax.swing.JOptionPane;
public class TestSwitch
{
    public static void main(String[] args)
    {
        char c;
        String str=JOptionPane.showInputDialog(null,"Nhập vào ký
tu?");
        c = str.charAt(0);
        switch(c)
        {
            case 'a': case 'e': case 'i': case 'o': case 'u':
                System.out.println("Ký tự này là nguyên âm");
                break;
            default:
```

```

        System.out.println("Ky tu nay la phu am");
    }
    System.exit(0); // kết thúc chương trình
}
}

```

### 2.4.3 Vòng lặp for

for(<khởi tạo>; <điều kiện lặp>; <bước nhảy>)  
 <khối lệnh>;

**Ví dụ:** Chương trình tính tổng các số lẻ từ 1 đến 100

```

public class TestFor
{
    public static void main(String[] args)
    {
        int tong = 0;
        for(int i=1; i<=100; i+=2)
            tong+=i;
        System.out.println(tong);
    }
}

```

### 2.4.4 Vòng lặp while

while (<biểu thức boolean>)  
 <khối lệnh>;

**Ví dụ:** Đoạn mã tính tổng các số lẻ từ 1 đến 100

```

int tong = 0, i = 1;
while (i<=100)
{
    tong+=i; i+=2;
}
System.out.println(tong);

```

### 2.4.5 Vòng lặp do-while

```

do
{
    <khối lệnh>;
} while <biểu thức boolean>;

```

**Ví dụ:** Đoạn mã tính tổng các số lẻ từ 1 đến 100

```
int tong = 0, i=1;
do
{
    tong+=i; i+=2;
} while (i<=100);
System.out.println(tong);
```

## 2.5 Nhập dữ liệu từ bàn phím

Có một số cách khác nhau để nhập dữ liệu từ bàn phím, sau đây xin đưa ra một cách sử dụng class Scanner trong gói tiện ích java.util như sau:

- Khởi tạo đối tượng Scanner gắn với luồng chuẩn input 'System.in'

```
Scanner in = new Scanner(System.in);
```

- Sử dụng các method khác nhau của Scanner để đọc dữ liệu input

- o Đọc một dòng dữ liệu input:

```
System.out.print("What is your name? ");
```

```
String name = in.nextLine();
```

- o Đọc dữ liệu kiểu số nguyên:

```
System.out.print("How old are you? ");
```

```
int age = in.nextInt();
```

Các method khác: nextShort(), nextFloat(), nextDouble(), nextBoolean() .v.v.

**Ví dụ:** Chương trình cho phép nhập tên, tuổi. In ra lời chào và dự đoán tuổi năm sau của người đó

```
import java.util.*;
```

```
public class InputTest{
```

```
    public static void main(String[] args){
```

```
        Scanner in = new Scanner(System.in);
```

```
        // get first input
```

```
        System.out.print("What is your name? ");
```

```
        String name = in.nextLine();
```

```
        // get second input
```

```
        System.out.print("How old are you? ");
```

```
        int age = in.nextInt();
```

```

        // display output on console
        System.out.println("Hello, " + name + ". Next year, you'll be " +
(age + 1));
    }
}

```

Việc **nhập ký tự từ bàn phím** có một cách đơn giản hơn sử dụng Scanner, xem ví dụ sau:

```

import java.io.*;
class InputChar
{
    public static void main(String args[])
    {
        char ch = ‘ ’;
        try{
            ch = (char) System.in.read();
        }
        catch(Exception e){
            System.out.println(“Nhập lỗi!”);
        }
        System.out.println(“Ky tu vua nhap:” + ch);
    }
}

```

## 2.6 Bài tập chương 2

**Bài tập 1:** Viết chương trình in ra họ tên, ngày sinh của mình trên 2 dòng, thêm các chú thích về class, tác giả, các dòng code

**Bài tập 2:** Viết chương trình nhập vào điểm bài thi của một sinh viên. Xếp loại căn cứ theo tiêu chí sau:

Điểm  $\geq 90$ : loại A  
 $80 \leq \text{Điểm} < 90$ : loại B  
 $70 \leq \text{Điểm} < 80$ : loại C  
 $60 \leq \text{Điểm} < 70$ : loại D  
Điểm  $< 60$ : loại F

**Bài tập 3:** Viết chương trình nhập vào từ bàn phím 2 số thực. Cho phép lựa chọn tính tổng, hiệu, tích, thương của 2 số đó

## CHƯƠNG 3 LỚP VÀ ĐỐI TƯỢNG

### Mở đầu

- Từ khi ra đời cho đến nay lập trình hướng đối tượng (OOP) đã chứng tỏ được sức mạnh và vai trò của nó trong các đề án tin học
- Lập trình OOP là một phương pháp mạnh mẽ và hiệu quả để xây dựng nên những chương trình ứng dụng trên máy tính
- Ở phần này chúng ta tìm hiểu các vấn đề cơ bản của lập trình hướng đối tượng trong Java thông qua việc tạo các lớp, các đối tượng và các tính chất của chúng

### 3.1 Khái niệm

- Lớp được xem như một khuôn mẫu (template) của đối tượng. Nó bao gồm các thuộc tính (properties) của đối tượng và các phương thức (method) tác động lên các thuộc tính.
  - o Ví dụ: Lớp SinhVien có các thuộc tính MSV, điểm, hạnh kiểm, có các phương thức học tập, thực hành, giải trí .v.v.
- Đối tượng là một thể hiện (class instance) của lớp. Mỗi đối tượng có một lớp định nghĩa các dữ liệu và hành vi của nó.
  - o Ví dụ: Mỗi bạn sinh viên cụ thể là một thể hiện (hay đối tượng) của lớp SinhVien

### 3.2 Khai báo lớp

```
class <ClassName>
{
    <kiểu dữ liệu> <field_1>;
    <kiểu dữ liệu> <field_2>;
    constructor
    method_1
    method_2
}
```

**class**: là từ khóa của java

**ClassName**: là tên chúng ta đặt cho lớp

**field\_1, field\_2**: các thuộc tính (các biến, hay các thành phần dữ liệu của lớp)

**constructor**: là phương thức xây dựng, khởi tạo đối tượng của lớp.

**method\_1, method\_2:** là các phương thức (có thể gọi là hàm) thể hiện các thao tác xử lý, tác động lên các thuộc tính của lớp.

**Ví dụ:**

```
class Pencil {
    public String color = "red";
    public int length;
    public float diameter;
    public static long nextID = 0;
    public void setColor (String newColor) {
        color = newColor;
    }
}
```

### 3.3 Thuộc tính của lớp

```
class <ClassName>
{
    // khai báo những thuộc tính của lớp
    <tiền tố> <kiểu dữ liệu> field1;
    // ...
}
```

**Trong đó**

- Kiểu dữ liệu có thể là kiểu dữ liệu cơ sở hoặc kiểu dữ liệu đối tượng tham chiếu
  - o boolean, char, byte, short, int, long, float, double
- Tiền tố bao gồm:
  - o Chỉ định truy xuất thuộc tính lớp + các bổ nghĩa loại thuộc tính nếu có (Các bổ nghĩa loại thuộc tính có thể là static hoặc final hoặc cả 2)
- Sau tên trường có thể có kèm theo giá trị khởi tạo
- 

**Chỉ định truy xuất thuộc tính lớp**

- private: Có thể truy cập thuộc tính này chỉ bên trong lớp khai báo
- package (~ không có chỉ định truy xuất): Có thể truy cập từ các lớp trong cùng gói (package) và trong bản thân lớp khai báo
- protected: Có thể truy cập từ các lớp trong cùng gói, các lớp thừa kế và bản thân lớp khai báo

- public: Có thể được truy cập từ bất kỳ một lớp nào

**Ví dụ:**

```
public class Pencil {  
    public String color = "red";  
    public int length;  
    public float diameter;  
    private float price;  
    public static long nextID = 0;  
    public void setPrice (float newPrice) {  
        price = newPrice;  
    }  
}
```

```
public class CreatePencil {  
    public static void main (String args[]){  
        Pencil p1 = new Pencil();  
        p1.price = 0.5f;  
    }  
}
```

%> javac Pencil.java

%> javac CreatePencil.java

CreatePencil.java:4: price has private access in Pencil  
p1.price = 0.5f

**Các bổ nghĩa loại thuộc tính**

- **static**

- Chỉ một bản duy nhất của thuộc tính static được tồn tại, chia sẻ giữa các đối tượng của cùng lớp chứa thuộc tính này
- Có thể được truy cập trực tiếp trong bản thân lớp khai báo (truy cập trong hàm main)
- Nếu truy cập từ bên ngoài lớp khai báo, phải kèm theo tên lớp trước tên thuộc tính  
System.out.println(Pencil.nextID);  
hay thông qua một đối tượng phụ thuộc vào lớp khai báo
- Từ bên ngoài lớp, các thuộc tính non-static phải được truy cập thông qua tham chiếu đối tượng

**Ví dụ:**

```

public class CreatePencil {
    public static void main (String args[]){
        Pencil p1 = new Pencil();
        Pencil.nextID++;
        System.out.println(p1.nextID);
        //Result? 1

        Pencil p2 = new Pencil();
        Pencil.nextID++;
        System.out.println(p2.nextID);
        //Result? 2

        System.out.println(p1.nextID);
        //Result? Still 2
    }
}

```

Chú ý: Đây chỉ là một ví dụ minh họa cho thuộc tính static, nó không phải là một thiết kế tốt

- **final**

- Khi đã được khởi tạo, giá trị không thể thay đổi
- Thường được sử dụng cho các hằng số
- Thuộc tính static + final phải được khởi tạo ngay khi khai báo
- Thuộc tính non-static + final phải được khởi tạo ngay khi một đối tượng của lớp được tạo ra

### Khởi tạo giá trị thuộc tính

Không nhất thiết là hằng số, ta có thể khởi tạo giá trị cho mọi biến nếu có quyền

Nếu không khởi tạo, giá trị khởi tạo mặc định sẽ phụ thuộc vào loại thuộc tính

Type	Initial Value
boolean	false
char	'\u0000'
byte, short, int, long	0
float	+0.0f
double	+0.0
object reference	null



### 3.4 Phương thức của lớp

#### Khai báo phương thức lớp

```
<Tiền tố> <kiểu trả về> <Tên phương thức> (<danh sách đối số>)  
{  
    <khối lệnh>;  
}
```

Để xác định quyền truy xuất của các đối tượng khác đối với các phương thức của lớp người ta thường dùng các tiền tố sau:

- Các chỉ định truy xuất của phương thức (cùng ý nghĩa với thuộc tính): **public, protected, private**
- Các bổ nghĩa loại phương thức: **static, final, abstract, synchronized, native, volatile**

<kiểu trả về>: có thể là kiểu void, kiểu cơ sở hay một lớp.

<Tên phương thức>: đặt theo qui ước giống tên biến.

<danh sách đối số>: có thể rỗng

#### Các bổ nghĩa loại phương thức:

- static: Phương thức dùng chung cho tất cả các thể hiện của lớp, chỉ có thể truy cập đến các thuộc tính hoặc phương thức static trong cùng lớp
- final: phương thức có thuộc tính này không được ghi đè (overridden) trong các lớp thừa kế
- abstract: phương thức không cần cài đặt, sẽ được cài đặt trong các lớp thừa kế
  - o VD: abstract void sampleMethod( );

#### Lời gọi phương thức:

- Sử dụng toán tử (.)
  - o reference.method(arguments)
- Với phương thức static
  - o Bên ngoài class, tham chiếu reference có thể là tên class hoặc tham chiếu đối tượng của class
  - o Bên trong class, không cần tham chiếu reference
- Với phương thức non-static:
  - o “reference” phải là tham chiếu đối tượng

#### Giá trị của các đối số truyền vào trong lời gọi phương thức:

Khi đối số không phải là một tham chiếu đối tượng, nó truyền vào một bản copy giá trị của đối số:

Ví dụ:

```
public void method1 (int a) {  
    a = 6;  
}  
public void method2 ( ) {  
    int b = 3;  
    method1(b);    // now b = ? b = 3  
}
```

Khi đối số là một tham chiếu đối tượng, nó truyền vào một bản copy của tham chiếu tới đối tượng đó

Ví dụ:

```
class PassRef{  
    public static void main(String[] args) {  
        Pencil plainPencil = new Pencil("PLAIN");  
        System.out.println("original color: " + plainPencil.color);  
        paintRed(plainPencil); // truyền vào tham chiếu đối tượng  
        System.out.println("new color: " + plainPencil.color);  
    }  
  
    public static void paintRed(Pencil p) {  
        p.color = "RED"; // đổi màu thành đỏ  
        p = null; // sau đó trở về null  
    }  
}
```

Kết quả chạy:

```
Original color: PLAIN  
New color: RED
```

Quá trình hoạt động:

- Đối tượng Pencil với màu plain được tạo ra, biến plainPencil tham chiếu đến nó
- In ra màu ban đầu của đối tượng Pencil được tạo ra
- Thực hiện gọi hàm paintRed để đổi sang màu đỏ. Hàm này hoạt động như sau:
  - o Copy một bản của tham chiếu đối tượng plainPencil sang biến p làm đối số truyền vào hàm (lúc này có 2 tham chiếu tới đối tượng Pencil)

- Thực hiện đổi màu của đối tượng vừa được tạo ra thành RED thông qua tham chiếu p
- Tham chiếu p được gán giá trị NULL, lúc này không trỏ vào đối tượng Pencil nữa
- In ra màu của đối tượng sau khi đã đổi màu (RED)

### **Nạp chồng phương thức (method overloading)**

Một class có thể có nhiều cách thức có cùng tên nhưng khác nhau về danh sách đối số

```
public class Pencil {
    ...
    public void setPrice (float newPrice) {
        price = newPrice;
    }

    public void setPrice (Pencil p) {
        price = p.getPrice();
    }
}
```

Trình dịch phân biệt sự khác nhau của 2 danh sách đối số bằng cách so sánh số đối số, kiểu của các đối số trong 2 danh sách (có phân biệt thứ tự)

### **3.5 Chỉ định truy xuất lớp**

Một lớp cũng có thể có các chỉ định truy xuất đi trước tên lớp

- **public**
  - Có thể truy xuất từ bất kỳ đâu
  - Không có từ khóa này, lớp chỉ có thể truy cập trong phạm vi cùng gói
- **abstract**
  - Lớp này được thiết kế nhằm tạo ra một lớp có các đặc tính tổng quát, nó định nghĩa các thuộc tính chung cho các lớp con của nó
  - Không thể tạo đối tượng (thể hiện) cho những lớp abstract
- **final**
  - Lớp không thể được thừa kế

### 3.6 Tạo đối tượng

Ví dụ Class:

```
class Student {  
    private long idNum;  
    private String name = "empty";  
    private String address;  
    private static long nextID = 0;  
}
```

Tạo đối tượng mới: `Student std = new Student();`

Trong Java, một đối tượng được tạo ra bằng cách sử dụng phương thức **new**

#### 3.6.1 Constructor

Là một phương thức đặc biệt của lớp. Dùng gọi tự động khi khởi tạo một thể hiện (đối tượng) của lớp, có thể dùng để khởi gán những giá trị mặc định

- Không có giá trị trả về, có thể có hoặc không có tham số
- Phải có cùng tên với lớp và được gọi đến khi dùng từ khóa `new`
- Nếu một lớp không có constructor, Java cung cấp một constructor mặc định, những thuộc tính của lớp sẽ được khởi tạo giá trị mặc định (kiểu số = 0, logic = false, đối tượng = null)

◊ Chú ý: Thông thường để an toàn, để kiểm soát và làm chủ mã nguồn mỗi lớp nên khai báo một constructor

Ví dụ

```
class Student {  
    private long idNum;  
    private String name= "empty";  
    private String address;  
    // biến static chia sẻ giữa các đối tượng của lớp Student  
    private static long nextID = 0;  
  
    Student( ) {  
        // gán giá trị nextID cho idNum, sau đó tăng nextID lên 1  
        idNum = nextID++;  
    }  
  
    Student(String studentName, String addr) {  
        this( );  
        name = studentName;  
    }  
}
```

```

        address = addr;
    }
}

```

Xem xét trước đó chưa có bất kỳ đối tượng Student nào được tạo ra. Xem xét hai trường hợp sau:

**TH1:**

```
Student std = new Student();
```

Khi đó constructor không có tham số Student() được gọi khởi tạo các giá trị cho đối tượng Student: idNum = 0, name = “empty”, address = null, nextID = 1

**TH2:**

```
Student std = new Student(“Hai”, null);
```

```
Student std1 = new Student(“Hau”, “Hung Yen”);
```

Trong trường hợp này ta tạo ra hai đối tượng Student sử dụng constructor có tham số đầu vào. Các giá trị được khởi tạo như sau:

Đối tượng được tham chiếu bởi std: idNum = 0, name = “Hai”, address = null, nextID = 1

Đối tượng được tham chiếu bởi std1: idNum = 1 (giá trị nextID hiện tại), name = “Hau”, address = “Hung Yen”, nextID = 2 (giá trị nextID sau khi tăng)

### 3.6.2 Biến this

- Biến this được sử dụng như một tham chiếu đến đối tượng hiện tại
- Trong một constructor có thể dùng biến this để gọi một constructor khác. Nó phải được đặt trong dòng đầu tiên của constructor sử dụng nó. (Xem ví dụ về constructor)
- Biến this không thể được sử dụng trong một phương thức static (??)

Ví dụ:

```

class Student {
    private long idNum;
    private String name;
    private String address;
    private static long nextID = 0;
    private static LinkedList studentList = new LinkedList();
    ...
    Student(String name, String address) {
        this.name = name;
        this.address = address;
    }
}

```

```

    }
    ...
    private void inQueue() {
        studentList.add(this); // Thêm vào danh sách liên kết
    }
    ...
}

```

### 3.7 Gói – Package

#### 3.7.1 Package

Tập hợp các class có thể được tổ chức thành các nhóm khác nhau gọi là gói (package)

Một số gói trong thư viện chuẩn Java như **java.lang**, **java.util**

Lý do chính của việc sử dụng gói là để đảm bảo tính duy nhất của tên lớp

Các lớp có cùng tên có thể được đặt ở các gói khác nhau

Cách đặt tên gói:

hostname.com -> com.hostname

#### Khai báo package và vị trí đặt file source code

Để thêm một class vào một package làm theo 2 bước:

Khai báo tên gói ở trên cùng của file source code

```

package com.hostname.corejava;
public class Employee {
    ...
}

```

Đặt file source code vào thư mục con theo đường dẫn được chỉ ra ở tên gói. Ở đây file “Employee.java” được lưu trữ trong thư mục con: “../com/hostname/corejava/”

#### 3.7.2 Class importation

Hai cách để truy cập một class có thuộc tính public mà thuộc một package khác

Khai báo đầy đủ tên gói trước tên class

Vd:

```
java.util.Date today = new java.util.Date( );
```

import package bằng cách sử dụng câu lệnh import phía trên cùng của source code, sau câu lệnh khai báo package. Khi đó không cần khai báo tên gói trước tên class trong lời gọi

Để import chỉ một class từ gói java.util

```
import java.util.Date;
```

```
Date today = new Date( );
```

Để import tất cả các class trong gói java.util (không bao gồm các gói con – sub package)

```
import java.util.*;
```

```
Date today = new Date( );
```

Ví dụ:

```
package vn.edu.hua.javacore;
```

```
import javax.swing.*;
```

```
public class SampleClass {
```

```
    MenuEvent c;
```

```
}
```

Dịch chương trình: *javac SampleClass.java*

Kết quả dịch:

*SampleClass.java:4: cannot find symbol*

*Symbol : class MenuEvent*

*Location: class SampleClass*

*MenuEvent c;*

*^*

*1 error*

?? MenuEvent là class trong package javax.swing.event, là sub-package của gói javax.swing. Vì vậy cần khai báo:

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

## **Trường hợp 2 lớp cùng tên được đặt ở hai gói khác nhau**

Ví dụ:

```
import java.util.*;
```

```
import java.sql.*;
```

```
Date today = new Date( );
```

//ERROR:java.util.Date

//or java.sql.Date?

Nếu chỉ cần sử dụng class Date ở một trong 2 gói, import thêm class đó

```
import java.util.*;
```

```
import java.sql.*;
import java.util.Date;
Date today = new Date( );           // java.util.Date
```

Nếu cần sử dụng class Date ở cả 2 gói, sử dụng một khai báo đầy đủ tên gói trước tên class

```
import java.util.*;
import java.sql.*;
java.sql.Date today = new java.sql.Date( );
java.util.Date nextDay = new java.util.Date( );
```

### 3.7.3 Qui cách đặt tên

Qui cách đặt tên (naming convention) là những cách đặt tên được sử dụng rộng rãi như một template chung. Bạn nên tuân thủ để chương trình dễ đọc, dễ quản cho bản thân và các lập trình viên khác khi đọc code.

- Package names: Các chữ cái viết thường
  - o E.g. java.util, java.net, java.io . . .
- Class names: Viết hoa tất cả các chữ cái đầu của mỗi từ
  - o E.g. File, Math, DemoJavaCore . . .
  - o Tránh xung đột với tên gói, các từ khóa chuẩn
- Variable, field & method names: Tương tự tên lớp, viết thường đối với chữ cái đầu tiên
  - o E.g. x, out, abs, firstName, lastName, getFirstName(), setFirstName() . . .
- Constant names: Viết hoa tất cả các chữ cái
  - o E.g. PI . . .

## 3.8 Bài tập chương 3

# CHƯƠNG 4 ĐẶC ĐIỂM HƯỚNG ĐỐI TƯỢNG TRONG JAVA

## 4.1 Tính đóng gói

Xem xét các ví dụ sau đây:

Ví dụ 1:

```
public class Student {
    public long idNum;
    public String name = "empty";
```



```

    public String address;
    public static long nextID = 0;

    Student() {
        idNum = nextID++;
    }

    Student(String studentName, String addr) {
        this();
        name = studentName;
        address = addr;
    }
}

```

**Problem:** Tất cả các thuộc tính là public, có thể được chỉnh sửa bởi tất cả các lớp khác truy cập đến.

Ví dụ 2: Cải tiến ví dụ 1, tất cả các thuộc tính được thay đổi từ public -> private

```

public class Student {
    private long idNum;
    private String name = "empty";
    private String address;
    private static long nextID = 0;

    Student() {
        idNum = nextID++;
    }

    Student(String studentName, String addr) {
        this();
        name = studentName;
        address = addr;
    }
}

```

**Problem:** Làm thế nào để truy cập các thuộc tính?

Ví dụ 3: Cải tiến Vd 2, thêm vào các phương thức để truy cập thuộc tính

```
public class Student {
    private long idNum;
    private String name = "empty";
    private String address;
    private static long nextID = 0;

    Student() {
        idNum = nextID++;
    }

    Student(String studentName, String addr) {
        this();
        name = studentName;
        address = addr;
    }

    public long getID() {return idNum;}
    public String getName() {return name;}
    public String getAddress() {return address;}
}
```

**Note:** Bây giờ các thuộc tính idNum, name & address là chỉ đọc (read-only) đối với các lớp khác

Ví dụ 4: Cải tiến Vd 3, thêm vào các phương thức set để gán giá trị cho các thuộc tính

```
class Student {
    private long idNum;
    private String name = "empty";
    private String address;
    private static long nextID = 0;

    // constructors. . .

    public long getID() {return idNum;}
    public String getName() {return name;}
    public String getAddress() {return address;}

    public void setName(String newName) {name = newName;}
    public void setAddress(String addr) {address = addr;}
}
```

}

**Note:** Bây giờ ta có thể set giá trị cho các thuộc tính name & address. Nhưng idNum thì không

### Tính đóng gói dữ liệu:

Giấu đi một phần chi tiết cài đặt và các dữ liệu cục bộ của nó (bằng cách giới hạn quyền truy cập) và chỉ công bố ra ngoài những gì cần thiết để trao đổi dữ liệu với các đối tượng khác

à Trong các ví dụ trên ta thấy tính đóng gói được thực hiện bằng cách không cho phép truy cập trực tiếp đến các thuộc tính bằng cách dùng từ khóa private để ngăn cấm truy cập trực tiếp và cung cấp các phương thức get, set để quản lý việc truy cập dữ liệu nếu cần thiết.

## 4.2 Tính kế thừa

### 4.2.1 Tính kế thừa

**Tính kế thừa (inheritance):** Ta có thể tạo ra một lớp mới trên cơ sở một lớp cũ đã được tạo ra trước đó. Bằng cách kế thừa, ta có thể sử dụng lại các phương thức và thuộc tính đã được khai báo trong lớp cũ mà không phải khai báo lại, và cũng có thể bổ xung các phương thức, thuộc tính cho lớp mới để giải quyết bài toán với những yêu cầu mới đặt ra.

```
class A extends B
{
    // ...
}
```

A: Subclass, B: Superclass

Subclass ~ lớp con (lớp dẫn xuất) & Superclass – lớp cha

Ví dụ: Lớp con Student kế thừa các thuộc tính, phương thức từ lớp cha Person

Superclass: Person

```
public class Person{
    private String name;

    public Person ( ) {
        name = “no_name_yet”;
    }

    public Person ( String initialName ) {
```

```

        this.name = initialName;
    }

    public String getName ( ) {
        return name;
    }

    public void setName ( String newName ) {
        name = newName;
    }
}

```

#### Subclass: Student extends Person

```

public class Student extends Person {
    private int studentNumber;

    public Student ( ) {
        super( );
        studentNumber = 0;
    }

    public Student (String initialName, int initialStudentNumber) {
        super(initialName);
        studentNumber = initialStudentNumber;
    }

    public int getStudentNumber ( ) {
        return studentNumber;
    }

    public void setStudentNumber (int newStudentNumber ) {
        studentNumber = newStudentNumber;
    }
}

```

Chú ý: Nếu một lớp tạo ra không được khai báo kế thừa từ một lớp khác thì mặc định nó được kế thừa từ lớp Object (lớp gốc trong Java). Trong ví dụ trên lớp Person kế thừa từ lớp Object.

### Lớp dẫn xuất

Một lớp dẫn xuất bao gồm 2 phần:

- Các thuộc tính, cách thức của riêng nó (locally)
- Các thuộc tính, cách thức kế thừa từ lớp cha

? Liên hệ class Student trong ví dụ trên

### Constructor trong lớp dẫn xuất

- Constructor của lớp dẫn xuất có thể triệu gọi constructor của lớp cha bằng cách sử dụng phương thức super
  - o Ví dụ: 2 phương thức super trong class Student gọi đến 2 constructor trong lớp cha Person
- Nếu không có phương thức super được sử dụng thì mặc định phương thức super() được thêm vào trong dòng lệnh đầu tiên của constructor của lớp dẫn xuất sẽ gọi đến constructor không chứa tham số của lớp cha
  - o Ví dụ: Phương thức super() trong constructor đầu tiên của lớp Student có thể được bỏ đi
- Constructor không thể được thừa kế

#### 4.2.2 Nạp chồng phương thức

Nạp chồng phương thức (Overloading method): Cung cấp nhiều phương thức có cùng tên nhưng khác nhau về danh sách tham số

- Xuất hiện trong cùng một class hay trong class dẫn xuất
- Có cùng tên phương thức
- Khác nhau về danh sách tham số (kiểu, số lượng, thứ tự)
- Có thể có kiểu trả về khác nhau

Ví dụ: phương thức print() trong lớp java.io.PrintStream cho phép xử lý print với các tham số đầu vào khác nhau

```
public void print(boolean b)
public void print(char c)
public void print(String s)
```

#### 4.2.3 Ghi đè phương thức

Ghi đè phương thức (Overriding method): Thay thế nội dung thực hiện của một phương thức trong lớp cha bởi nội dung thực hiện riêng của lớp con

- Xuất hiện trong lớp dẫn xuất
- Có cùng tên phương thức
- Có cùng kiểu trả về

- Chỉ định truy xuất của phương thức được ghi đè không được thu hẹp hơn so với phương thức trong lớp cha mà chỉ có thể mở rộng hơn
  - o Nếu phương thức trong lớp cha là public, phương thức trong lớp con chỉ có thể là public
  - o Nếu phương thức trong lớp cha là protected, phương thức trong lớp con có thể là public hay protected
  - o Nếu phương thức trong lớp cha là package, phương thức trong lớp con có thể là package, protected, public
  - o Nếu phương thức trong lớp cha là private, nó không được thừa kế nên không thể ghi đè trong lớp con
- Phương thức được ghi đè chỉ có thể throws ra những exception nằm trong danh sách lớp cha có thể throws ra, có thể là lớp dẫn xuất của các exception đó

### **Khi nào có thể ghi đè phương thức ?**

Một phương thức chỉ có thể được ghi đè nếu nó có thể được truy cập trong trong lớp dẫn xuất hay nói cách khác nó có thể được thừa kế

- Phương thức private trong lớp cha
  - o Không thể được ghi đè
  - o Nếu trong lớp con cũng chứa đựng một phương thức như vậy, phương thức đó hoàn toàn không liên quan tới phương thức trong lớp cha
- Phương thức là package trong lớp cha
  - o Có thể được ghi đè nếu lớp dẫn xuất nằm cùng gói với lớp cha
- Phương thức protected, public
  - o Luôn luôn có thể ghi đè

**Một số ví dụ về overloading và overriding: (Phần giải thích xin dành cho bạn đọc)**

Ví dụ 1:

package P1;

```
public class Base {
    private void pri( ) { System.out.println("Base.pri()"); }
    void pac( ) { System.out.println("Base.pac()"); }
    protected void pro( ) { System.out.println("Base.pro()"); }
    public void pub( ) { System.out.println("Base.pub()"); }

    public final void show( ) {
```

```

        pri(); pac(); pro(); pub();
    }
}

package P2;

import P1.Base;

public class Concrete1 extends Base {
    public void pri( ) { System.out.println("Concrete1.pri()"); }
    public void pac( ) { System.out.println("Concrete1.pac()"); }
    public void pro( ) { System.out.println("Concrete1.pro()"); }
    public void pub( ) { System.out.println("Concrete1.pub()"); }
}

Concrete1 c1 = new Concrete1();
c1.show( );

```

### Output?

```

Base.pri()
Base.pac()
Concrete1.pro()
Concrete1.pub()

```

### Ví dụ 2:

```

package P1;

import P2.Concrete1;

public class Concrete2 extends Concrete1 {
    public void pri( ) { System.out.println("Concrete2.pri()"); }
    public void pac( ) { System.out.println("Concrete2.pac()"); }
    public void pro( ) { System.out.println("Concrete2.pro()"); }
    public void pub( ) { System.out.println("Concrete2.pub()"); }
}

Concrete2 c2 = new Concrete2();
c2.show( );

```

### Output?

```
Base.pri()
Concrete2.pac()
Concrete2.pro()
Concrete2.pub()
```

Ví dụ 3:

```
package P3;
```

```
import P1.Concrete2;
```

```
public class Concrete3 extends Concrete2 {
    public void pri( ) { System.out.println("Concrete3.pri()"); }
    public void pac( ) { System.out.println("Concrete3.pac()"); }
    public void pro( ) { System.out.println("Concrete3.pro()"); }
    public void pub( ) { System.out.println("Concrete3.pub()"); }
}
```

```
Concrete3 c3 = new Concrete3();
c3.show( );
```

**Output?**

```
Base.pri()
Concrete3.pac()
Concrete3.pro()
Concrete3.pub()
```

#### 4.2.4 Trường ẩn

Nếu một thuộc tính xuất hiện cả trong lớp con và lớp cha (có thể khác kiểu). Khi đó từ lớp con nếu muốn truy xuất tới thuộc tính đó trong lớp cha, phải dùng từ khóa super đi kèm

Ví dụ:

```
class Book {
    String author, title, pu; // pu is publisher
}
```

```
//=====
```

```
class ShopBook extends Book {
```

```
    String pu; // pu is purchaser
```



```

public ShopBook(String a, String t, String pub, String purch) {
    author=a;
    title=t;
    super.pu=pub;
    pu=purch;
}

public String toString() {
    return
        title +
        " by " +
        author +
        " (publisher: " + super.pu + ")"
        + "\npurchaser: " + pu;
}

public static void main(String[] args) {
    ShopBook b = new ShopBook("H Jones","Killjoy", "AB Pub Inc",
    "V Smith");
    System.out.println(b);
}
}

```

#### 4.2.5 Class Object

Lớp Object là lớp cha của mọi lớp trong Java, mọi lớp trong Java thừa kế từ lớp này mà không cần khai báo extends

Một số phương thức của lớp Object

- equals: cho biết hai tham chiếu đối tượng có cùng giá trị hay không
  - toString: trả lại một chuỗi đại diện cho đối tượng
- > Khi cần sử dụng trong các class cụ thể, có thể thực hiện overridden method để cho phù hợp với yêu cầu đặt ra
- > Tham khảo 2 phương thức này trong class String

#### 4.3 Tính đa hình

Xem xét ví dụ sau:

```

class SuperShow {

```

```

    public String str = "SuperStr";

    public void show( ) {
        System.out.println("Super.show:" + str);
    }
}

class ExtendShow extends SuperShow {
    public String str = "ExtendedStr";

    public void show( ) {
        System.out.println("Extend.show:" + str);
    }

    public static void main (String[] args) {
        ExtendShow ext = new ExtendShow( );
        SuperShow sup = ext;
        sup.show( );    //1
        ext.show( );    //2  methods   invoked   through   object
reference
        System.out.println("sup.str = " + sup.str); //3
        System.out.println("ext.str = " + ext.str); //4  field access
    }
}

```

### Output?

```

Extend.show: ExtendStr
Extend.show: ExtendStr
sup.str = SuperStr
ext.str = ExtendStr

```

### Tính đa hình (Polymorphism)

- Tính đa hình thể hiện qua việc: cùng một phương thức nhưng có nội dung thực hiện khác nhau trên các đối tượng khác nhau
  - o Ví dụ: Phương thức show() trong ví dụ trên
- Phương thức gọi được xác định thông qua đối tượng được tham chiếu, không thông qua kiểu khai báo của tham chiếu
  - o Ví dụ: biến ext, sup đều tham chiếu đến đối tượng có kiểu SuperShow do đó phương thức show() của class SuperShow được gọi

- Thuộc tính gọi được xác định thông qua kiểu khai báo của tham chiếu
  - o Ví dụ: Các mục 3, 4 trong ví dụ trên

### **Tính tương thích kiểu (type compatibility)**

Ví dụ. Student is subclass of Person

```
public class TypeTest {
    // khai báo mảng static
    static Person[] p = new Person[10];

    static
    { // đoạn mã chạy khi khởi tạo giá trị cho mảng
        for (int i = 0; i < 10; i++) {
            if(i<5)
                p[i] = new Student();
            else
                p[i] = new Person();
        }
    }

    public static void main (String args[]) {
        Person o1 = (Person)p[0];
        Person o2 = p[0];
        Student o3 = p[0]; // 1
        Student o4 = (Student)p[0];
        Student o5 = p[9]; // 2
        Student o6 = (Student)p[9]; // 3
        int x = p[0].getStudentNumber(); // 4
    }
}
```

Thực hiện dịch:

**javac** TypeTest.java

TypeTest.java:17 incompatible types

found : Person

required : Student

```
Student o3 = p[0];
           ^
```

TypeTest.java:19 incompatible types

```

found    : Person
required : Student
        Student o5 = p[9];
                ^

```

```

TypeTest.java:21: cannot resolve symbol
symbol   : method getStudentNumber ()
location: class Person
        int x = p[0].getStudentNumber();
                ^

```

3 errors

Lỗi tại vị trí 1, 2: Khai báo của p[0], p[9] có kiểu Person, là lớp cha của lớp Student, do vậy không cho phép gán giá trị cho một biến có kiểu Student

Lỗi tại vị trí 4: p[0] được khai báo có kiểu Person, tham chiếu tới đối tượng Student, tuy nhiên trong class Person không có method getStudentNumber() do đó không thể gọi phương thức đó.

à Thực hiện thêm phương thức getStudentNumber() vào class Person, in ra giá trị của x, khi đó theo tính đa hình giá trị in ra là 0 (phương thức getStudentNumber() của class Student được gọi)

```

public int getStudentNumber ( ) {
    return 4;
}

```

Thực hiện khóa các dòng lệnh lỗi biên dịch tại các vị trí 1, 2, 4 lại và chạy chương trình:

```

java TypeTest
Exception in thread "main" java.lang.ClassCastException: Person
    at TypeTest.main(TypeTest.java:20)

```

Lỗi tại vị trí 3: Lỗi ép kiểu, p[9] tham chiếu tới đối tượng Person nên không thể ép về kiểu Student

à Đề ý câu lệnh phía trên đây: Student o4 = (Student)p[0]; câu lệnh này hợp lệ vì p[0] tham chiếu đến đối tượng Student

?? Nếu thực hiện ép kiểu từ Student -> Person có hợp lệ không

à Hợp lệ bởi Student là lớp con, nó thừa kế các thuộc tính, phương thức của Person, ngoài ra nó còn chứa các thuộc tính, phương thức của riêng nó.

## Kiểm tra kiểu của một đối tượng

Sử dụng toán tử instanceof

```
Ví dụ:      if ( obj instanceof String)
            {
                String str2 = (String)obj;
            }
```

## 4.4 *Lớp trừu tượng, Lớp vô sinh & Giao tiếp*

### 4.4.1 **Lớp trừu tượng – Abstract class**

Lớp này được thiết kế nhằm tạo ra một lớp có các đặc tính tổng quát, nó định nghĩa các thuộc tính chung cho các **lớp** con của nó

Lớp trừu tượng: được khai báo với từ khóa `abstract`, thường có ít nhất một phương thức trừu tượng, là phương thức chưa được cài đặt, việc cài đặt được thực hiện trong các lớp con thừa kế

```
public abstract class Person {
    protected String name;
    ...
    public abstract String getDescription();
    ...
}
```

```
Class Student extends Person {
    private String code;
    ...
    public String getDescription() {
        return "student name: " + name + "\n student
code: " + code;
    }
    ...
}
```

```
Class Employee extends Person {
    private float salary;
    ...
    public String getDescription() {
        return "an employee with a salary of $ " + salary;
    }
    ...
}
```

- Các phương thức chưa được cài đặt trong lớp trừu tượng phải khai báo abstract
- Nếu class có chứa phương thức abstract, class đó phải được khai báo abstract
- Khi thừa kế từ một lớp trừu tượng có 2 trường hợp xảy ra:
  - o Nếu lớp con được khai báo abstract (cũng là một lớp trừu tượng), nó có thể chỉ cần cài đặt một vài phương thức abstract của lớp cha
  - o Nếu lớp con không phải lớp trừu tượng, nó phải cài đặt tất cả các phương thức abstract của lớp cha
- Không thể tạo các đối tượng của một lớp trừu tượng nhưng có thể khai báo biến thuộc kiểu lớp trừu tượng để tham chiếu đến các đối tượng thuộc lớp con của nó
  - o Vd: `Person p = new Student( );`

#### 4.4.2 Lớp vô sinh – Final class

Lớp mà ta không thể có lớp dẫn xuất từ nó (không có lớp con) gọi là lớp “vô sinh”, hay nói cách khác không thể kế thừa được từ một lớp “vô sinh”. Lớp “vô sinh” dùng để hạn chế, ngăn ngừa các lớp khác dẫn xuất từ nó.

Để khai báo một lớp là lớp “vô sinh”, chúng ta dùng từ khóa final class.

Ví dụ: Lớp String của Java là final class, ta không thể thừa kế và viết phương thức length() của riêng mình

```
public final class String extends Object
implements Serializable, Comparable<String>, CharSequence
```

#### 4.4.3 Giao tiếp - Interface

**Khái niệm:** Interface chỉ ra những gì các class nên làm, mà không chỉ ra phải làm như thế nào. Interface không phải là class nó là tập hợp các yêu cầu cho class muốn thực thi interface

```
Vd: public interface Comparable
{
    int compareTo(Object otherObject);
}
```

Interface này yêu cầu bất kỳ class nào thực thi nó phải cài đặt phương thức compareTo, phương thức này phải có một tham số đầu vào có kiểu Object và trả lại một số integer

**Các từ chỉ định truy xuất (Modifiers)**

- public

- package
- abstract: ngầm định tất cả các interface đều có từ chỉ định này

### Thuộc tính và phương thức trong interface

Interface chỉ chứa các thuộc tính là các hằng, chỉ cần khai báo kiểu dữ liệu và tên hằng, chúng sẽ được hiểu mặc định là public, static, final mà không cần khai báo. Tất cả các hằng phải được khởi tạo giá trị

Các phương thức trong interface được hiểu mặc định là public abstract, chúng không thể là static hay final

Ví dụ:

```
Interface Verbose {
    int SILENT = 0;
    int TERSE = 1;
    int NORMAL = 2;
    int VERBOSE = 3;

    void setVerbosity (int level);
    int getVerbosity();
}
```

### Thực thi một interface - implement interfaces

2 bước:

1. Khai báo thực thi

```
class Employee implements Comparable { . . . }
```

2. Cài đặt tất cả các phương thức trong interface

```
public int compareTo(Object otherObject) {
    Employee other = (Employee) otherObject;
    if (salary < other.salary) return -1;
    if (salary > other.salary) return 1;
    return 0; }
```

Chú ý: Mặc dù phương thức compareTo trong interface không khai báo public nhưng nó được hiểu ngầm định như vậy, do đó trong khi cài đặt ở class Employee bắt buộc phải khai báo public

Nếu một class thực thi interface mà chưa cài đặt tất cả các phương thức trong interface thì nó phải khai báo abstract class

Một class có thể thực thi nhiều interface, phân cách bởi dấu phẩy

```
class Employee implements Comparable, Cloneable { . . . }
```

### Tạo thể hiện cho interface

Không thể tạo thể hiện cho một interface

```
public interface Comparable {  
    ... }
```

Comparable x = new Comparable( ); ❌ Không được phép

Có thể khai báo biến có kiểu interface

```
Comparable x;  
nhưng chúng phải tham chiếu tới một class thực thi interface  
class Employee implements Comparable {  
    ...  
}  
x = new Employee( );
```

### Tính thừa kế trong interface

Interface hỗ trợ đa thừa kế, một interface có thể thừa kế từ nhiều interface khác

Interface cha: Superinterfaces

Interface con: Subinterfaces

Ví dụ:

```
public interface SerializableRunnable extends java.io.Serializable,  
Runnable { ... }
```

### Khi nào sử dụng interface

Xem xét ví dụ sau:

```
interface Act {  
    void act();  
}
```

```
class Actor1 implements Act {  
    public void act() {  
        System.out.println("To be, or not to be");  
    }  
}
```

```
class Actor2 implements Act {  
    public void act() {  
        System.out.println("Wherefore art thou Romeo?");  
    }  
}
```



```

public class TryOut {
    public static void main(String args[]) {
        Actor1 hamlet = new Actor1();
        Actor2 juliet = new Actor2();
        tryout(hamlet);
        tryout(juliet);
    }

    private static void tryout(Act actor) {
        actor.act();
    }
}

```

- Để cung cấp một giao diện lập trình tới người sử dụng, che dấu đi sự thực thi phía sau nó
  - o Đây là tính chất đóng gói (encapsulation) của OOP
  - o Sự thực thi phía sau có thể thay đổi mà không ảnh hưởng tới người sử dụng
- Để có các class độc lập với nhau cùng cài đặt một phương thức
  - o Giữa các class không có quan hệ thừa kế
  - o Các class độc lập có thể cùng thực thi một interface và cài đặt các phương thức theo cách của riêng chúng
- Để giải quyết vấn đề đa thừa kế trong OOP
  - o Một class có thể thực thi cùng lúc nhiều interface trong khi nó chỉ có thể thừa kế từ một class

### **Update một interface hay abstract class ?**

Có một interface hay abstract class đã được phát triển trước đó. Yêu cầu update đặt ra:

- Thêm phương thức cho interface
- Thêm phương thức abstract cho abstract class

→ Phải update lại code trong tất cả các class thực thi interface hoặc thừa kế abstract class

### **→ Giải pháp ?**

Sử dụng thêm một interface hay một abstract class khác thừa kế từ interface, abstract class cũ, thực hiện thêm phương thức mới và sử dụng interface hay abstract mới tạo ra này cho các lớp thực thi hay thừa kế

### **So sánh interface và abstract class**

Một class chỉ có thể thừa kế từ một abstract class nhưng nó có thể thực thi nhiều interface

Một abstract class có thể chứa các phương thức đã được cài đặt (để giảm sự lặp lại code trong các lớp con thừa kế), có thể chứa các phần (thuộc tính, phương thức) khai báo static, protected. Một interface chỉ cho phép chứa các hằng số, các phương thức public mà chưa được cài đặt

Abstract class thường được dùng cho những lớp thuộc top-level trong cây thừa kế để định nghĩa những tính chất chung nhất mà các lớp con thừa kế phải thực hiện, nó ít thay đổi, có tính giới hạn truy xuất cao hơn (điển hình là sử dụng cho thiết kế framework). Interface thường dùng cho các thiết kế thường phải update các thay đổi mang tính giao tiếp với bên ngoài

## 4.5 Bài tập chương 4

# CHƯƠNG 5 MẢNG VÀ XÂU

## 5.1 Mảng

**Khái niệm:** Mảng là tập hợp nhiều phần tử có cùng tên, cùng kiểu dữ liệu và mỗi phần tử trong mảng được truy xuất thông qua chỉ số của nó trong mảng

### Khai báo mảng

```
<kiểu dữ liệu> <tên mảng>[];  
<kiểu dữ liệu>[] <tên mảng>;
```

### Ví dụ:

```
int arrInt[];  
int[] arrInt;  
Person[] p;  
int[] arrInt1, arrInt2, arrInt3;
```

Khai báo không hợp lệ:

```
int[5] iarray;  
int iarray[5];
```

### Cấp phát

```
iarray = new int[100];  
p = new Person[10];
```

### Khởi tạo

Chúng ta có thể khởi tạo giá trị ban đầu cho các phần tử của mảng khi nó được khai báo

```
int[] iarray = { 1, 2, 3, 5, 6};  
char[] carray = { 'a', 'b', 'c' };  
String arrString[] = { "ABC", "EFG", "GHI" };
```

*Chú ý: Luôn khởi tạo hoặc cấp phát mảng trước khi sử dụng*

### **Truy cập mảng**

Chỉ số mảng trong Java bắt đầu từ 0. Vì vậy phần tử đầu tiên có chỉ số là 0, và phần tử thứ n có chỉ số là n-1. Các phần tử của mảng được truy xuất thông qua chỉ số của nó được đặt giữa cặp dấu ngoặc vuông [].

```
iarray[3] = 0;  
carray[1] = 'z';
```

### **Lấy số phần tử mảng**    iarray.length

Ví dụ 1: Tìm phần tử nhỏ nhất, lớn nhất trong một mảng

```
class MinMax2
```

```
{    public static void main(String args[])  
    {    int nums[] = { 99, -10, 100123, 18, -978, 5623, 463, -9,  
287, 49 };  
  
        int min, max;  
        min = max = nums[0];  
        for(int i=1; i < 10; i++)  
        {  
            if(nums[i] < min) min = nums[i];  
            if(nums[i] > max) max = nums[i];  
        }  
        System.out.println("Min and max: " + min + " " + max);  
    }  
}
```

Ví dụ 2: Khởi tạo và in ma trận

```
class TwoD_Arr
```

```
{    public static void main(String args[])  
    {    int t, i;  
        int table[][] = new int[3][4];  
        for(t=0; t < 3; ++t)  
        {    for(i=0; i < 4; ++i)  
            {    table[t][i] = (t*4)+i+1;  
                System.out.print(table[t][i] + " ");  
            }  
            System.out.println();  
        }  
    }
```

```
}  
}
```

## 5.2 Xâu

Trong những ngôn ngữ lập trình khác (C chẳng hạn), một xâu được xem như một mảng các ký tự.

Trong java thì khác, java cung cấp một lớp String để làm việc với đối tượng dữ liệu xâu cùng các thao tác trên đối tượng dữ liệu này.

### Có nhiều cách để tạo đối tượng String:

```
String str1 = new String("ABC");  
String str2 = "ABC";  
String str3 = new String(str2);
```

### Một số thao tác cơ bản liên quan đến xâu:

- *int length()*
  - o Trả về độ dài xâu
- *str3 = str1 + str2*
  - o Cộng xâu
- *String substring(int beginIndex, int lastIndex)*
  - o Trả về xâu con từ xâu mẹ
- *boolean str1.equal(str2)*
  - o So sánh 2 xâu
- *String str1.trim()*
  - o Loại bỏ khoảng trống đầu cuối
- *char charAt(int index)*
  - o Trả về ký tự ở vị trí index
- *String toLowerCase()*
  - o Trả về một xâu mới với tất cả các ký tự viết thường
- *String toUpperCase()*
  - o Trả về một xâu mới với tất cả các ký tự viết hoa
- *int indexOf(String str)*
  - o Trả về vị trí của str trong xâu cần tìm bắt đầu từ 0, nếu không tồn tại, trả lại giá trị -1

### So sánh xâu

Để kiểm tra 2 xâu có giống nhau hay không sử dụng phương thức `equal: str1.equal(str2)`

Ví dụ: `str1 = "Hello"; str2="Hello";` Khi đó `str1.equal(str2) = true`

Nếu so sánh không phân biệt chữ hoa, thường, dùng phương thức `equalsIgnoreCase`

Chú ý: Không sử dụng toán tử `"=="` để kiểm tra sự giống, hay khác nhau của 2 xâu

Để so sánh 2 chuỗi, quan tâm đến kết quả lớn hơn, nhỏ hơn hay giống nhau sử dụng phương thức: *int compareTo(String other)*

result = str1.compareTo(str2);

Nếu result = 0 hai chuỗi là giống nhau

Nếu result > 0 kết luận str1 > str2

Nếu result < 0 kết luận str1 < str2

**Ví dụ minh họa lấy chuỗi con từ một chuỗi:**

```
class SubStr
{
    public static void main(String args[])
    {
        String orgstr = "Mot Hai Ba Bon";
        // Lấy chuỗi con dùng hàm
        // public String substring(int beginIndex, int
        // endIndex)
        String substr = orgstr.substring(4, 7);
        System.out.println("Chuỗi gốc: " + orgstr);
        System.out.println("Chuỗi con: " + substr);
    }
}
```

### 5.3 Bài tập chương 5

## CHƯƠNG 6 CĂN BẢN VỀ LẬP TRÌNH GIAO DIỆN

### 6.1 Căn bản về lập trình giao diện

#### 6.1.1 Giới thiệu thiết kế GUI trong Java

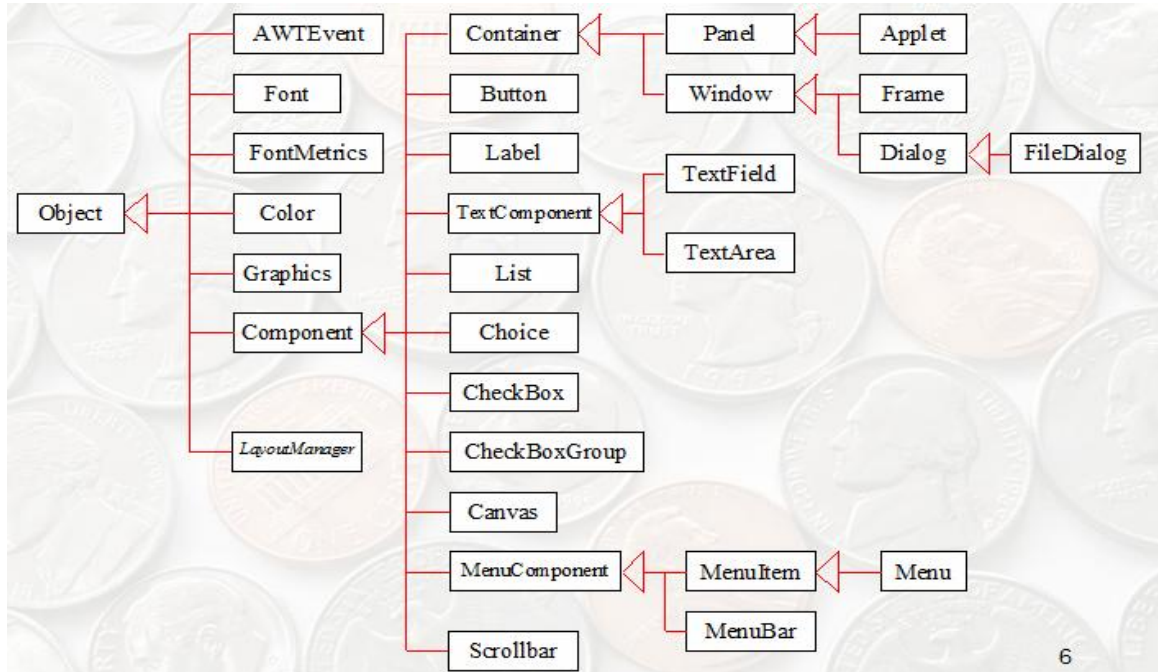
Thư viện hỗ trợ: tập hợp các lớp java cung cấp hỗ trợ thiết kế, xây dựng GUI (Graphic User Interface – Giao diện đồ họa người sử dụng) bao gồm:

- awt (java.awt.\*)
- swing (javax.swing.\*)

Trong nội dung của phần này, chúng ta đi tìm hiểu về AWT. Để lập trình giao diện và xử lý sự kiện trong AWT, cần sử dụng đến các class trong 2 gói:

- java.awt.\*;
- java.awt.event.\*;

Các thành phần của AWT và mối quan hệ giữa chúng thể hiện qua cây thừa kế:



Nguyên tắc xây dựng GUI:

- Lựa chọn một khung chứa: Frame, Window, Dialog, Applet,...
- Tạo các control: (buttons, text areas, list, choice, checkbox,...)
- Đưa các control vào khung chứa
- Sắp xếp các control trong khung chứa (Layout).
- Thêm các xử lý sự kiện (Listeners)

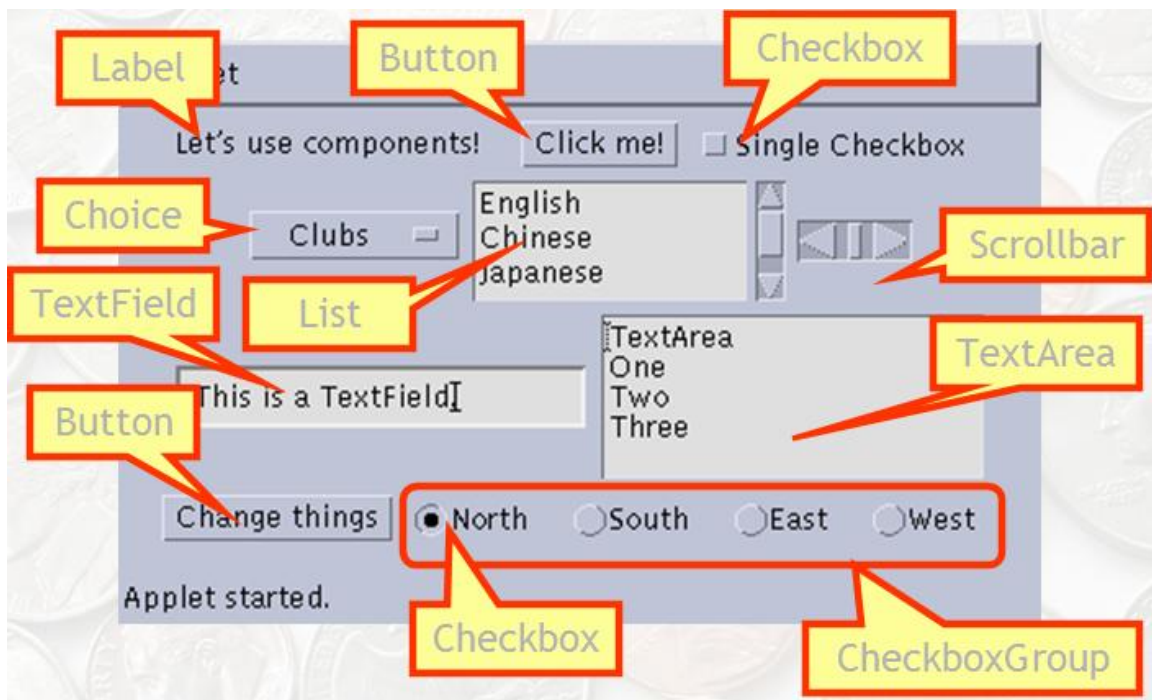
### 6.1.2 Các thành phần cơ bản – Components

Tất cả các thành phần cấu tạo nên giao diện GUI được gọi là component.

Ví dụ:

Frame, Window, Dialog, Applet,...

TextFields, Labels, CheckBoxes, TextArea, Button, Choice, List, Scrollbars,...



**Giới thiệu một số thành phần cơ bản (các thành phần chưa giới thiệu ở đây, có thể xem ở phần tham khảo thêm):**

#### **6.1.2.1 Nhãn – Label**

Nhãn được dùng để trình bày một chuỗi văn bản ra màn hình

Một số phương thức của Label:

- public Label(); // tạo nhãn
- public Label(String s); // tạo nhãn với nội dung s
- public Label(String s, int align); // tạo và canh lề
- void setText(String s); // đặt nội dung nhãn
- void setAlignment(int align); // canh lề nhãn
- ...

Ví dụ:

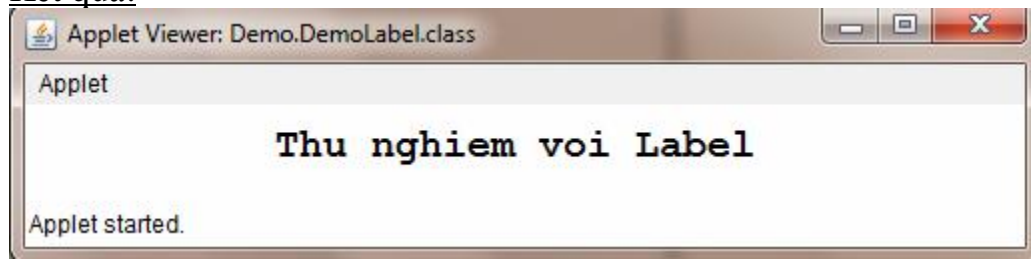
```
import java.applet.Applet;
import java.awt.*;
public class DemoLabel extends Applet{
    private Label label;
    public void init(){
        Font font = new Font("Courier", Font.BOLD, 20);
        label = new Label("Thu nghiem voi Label");
        label.setFont(font);
        add(label);
    }
}
```

```

    }
}

```

Kết quả:



### 6.1.2.2 Nút nhấn – Button

Một số phương thức của Button

- Button(); // tạo nút nhấn
- Button(String s); // tạo nút nhấn có tên s
- void setLabel(String s); // đổi tên nút
- String getLabel(); // lấy tên nút nhấn

Để lắng nghe sự kiện nhấn nút ta cần cài đặt giao tiếp *ActionListener*.

Ví dụ:

```

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class DemoButton extends Applet implements ActionListener
{
    private Button blueButton;
    private Button whiteButton;
    private Button helloButton;
    public void init()
    {
        blueButton = new Button("Blue");
        whiteButton = new Button("White");
        helloButton = new Button("Hello");
        blueButton.addActionListener(this);
        whiteButton.addActionListener(this);
        helloButton.addActionListener(this);
        add(blueButton);
        add(whiteButton);
        add(helloButton);
    }
    // Cài đặt phương thức của giao tiếp ActionListener
    public void actionPerformed(ActionEvent event)

```



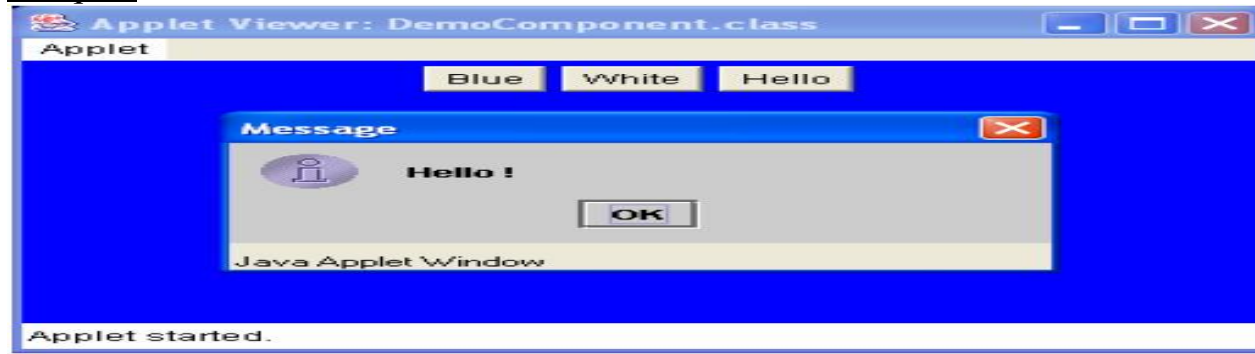
```

        {
            if(event.getSource() == helloButton)

javax.swing.JOptionPane.showMessageDialog(this, "Hello !");
        else{
            if (event.getSource() == blueButton){
                this.setBackground(Color.BLUE);
            }else if (event.getSource() == whiteButton){
                this.setBackground(Color.WHITE);
            }
            repaint();
        }
    }
}

```

Kết quả:



### 6.1.2.3 Ô văn bản – Text Field

Ô văn bản cho phép nhận dữ liệu từ bàn phím trên một dòng

Một số phương thức

- TextField(...); // các cấu tử
- void setEditable(boolean b); // đặt/tắt chế độ nhập
- void setEchoChar(char c); // đặt kí tự hiển thị

Để lắng nghe sự kiện cần cài đặt các giao tiếp:

- ActionListener: Xử lý sự kiện sau khi đã hoàn thành nhập dữ liệu vào text field
  - o Cài đặt phương thức actionPerformed
- TextListener: Xử lý sự kiện khi có sự thay đổi giá trị text
  - o Cài đặt phương thức textValueChanged

Ví dụ:

```

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

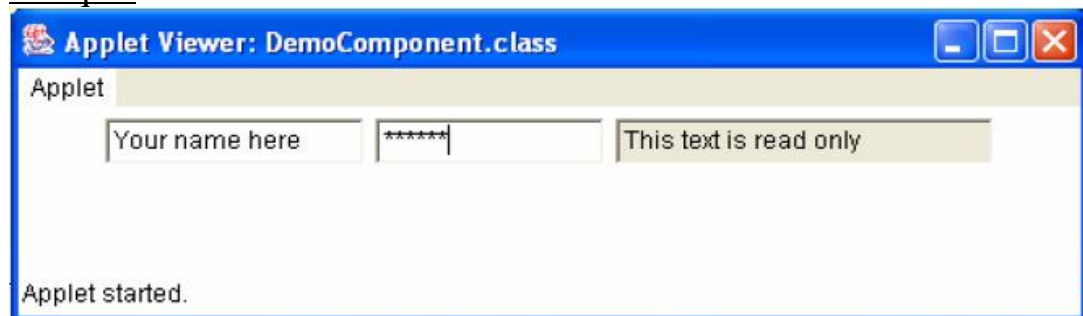
```

```
public class DemoTextField extends Applet implements  
ActionListener
```

```
{  
    private TextField txtEdit;  
    private TextField txtReadOnly;  
    private TextField txtPass;  
    private final String PASSWORD = "Java";  
    public void init()  
    {  
        txtEdit = new TextField("Your name here");  
        txtPass = new TextField(12);  
        // Hien thi password nhap vao duoi dang dau *  
        txtPass.setEchoChar('*');  
        // Lang nghe su kien sau khi nhap xong password va go  
        txtPass.addActionListener(this);  
        txtReadOnly = new TextField("This text is read only");  
        txtReadOnly.setEditable(false);  
        add(txtEdit);  
        add(txtPass);  
        add(txtReadOnly);  
    }  
    public void actionPerformed(ActionEvent event)  
    {  
        if(txtPass.getText().equals(PASSWORD))  
            txtReadOnly.setText("Password is valid");  
        else  
            txtReadOnly.setText("Invalid password !");  
    }  
}
```

Enter

Kết quả:



#### 6.1.2.4 Lựa chọn – Choice

Choice cung cấp khả năng lựa chọn một trong số các hạng mục sẵn có.

Một số phương thức

- Choice(); // cấu tử
- void addItem(String s); // thêm item
- String getItem(int index); // lấy item có chỉ số index
- String getSeclectedItem(); // trả về item được chọn
- int getSelectedIndex(); // trả về index của item được chọn

Lớp nghe cài đặt giao tiếp ItemListener. Cài đặt phương thức itemStateChanged(...)

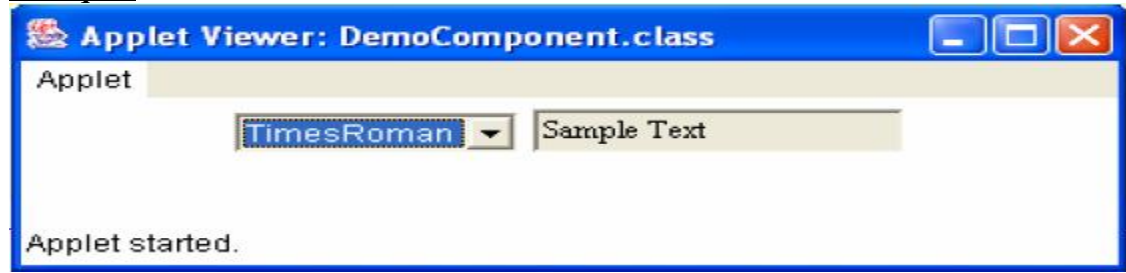
Ví dụ: Chương trình cho phép lựa chọn font chữ cho ô text bên cạnh

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class DemoChoice extends Applet implements ItemListener
{
    private Choice choice;
    private TextField txtText;
    private Font font;
    public void init()
    {
        choice = new Choice();
        choice.addItem("TimesRoman");
        choice.addItem("Courier");
        choice.addItem("Helvetica");
        choice.addItemListener(this);
        txtText = new TextField("Sample Text", 16);
        txtText.setEditable(false);
        font = new Font(choice.getItem(0), Font.PLAIN, 12);
        txtText.setFont(font);
        add(choice);
        add(txtText);
    }
    public void itemStateChanged(ItemEvent event)
    {
        font = new Font(choice.getSelectedItem(), Font.PLAIN,
12);

        txtText.setFont(font); // Thay doi font chu cho o text
    }
}
```

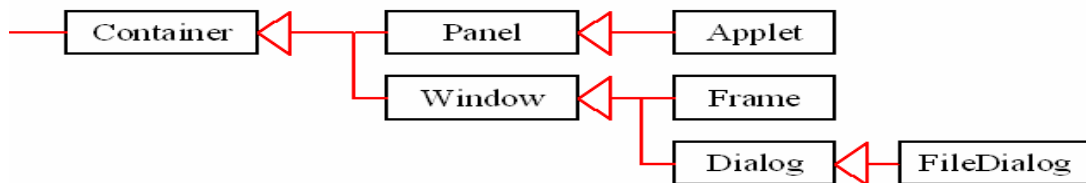
}

Kết quả:



### 6.1.3 Đối tượng khung chứa – Container

Là các thành phần mà có thể chứa các thành phần khác, có thể vẽ và tô màu. Gồm có: Frame, Applet, Panel, ScrollPane, Dialog, FileDialog.



Trong phần giới thiệu về các component cơ bản, ta đã được làm quen với khung chứa Applet, Applet sẽ được giới thiệu chi tiết ở một phần riêng. Ở đây xin giới thiệu về khung chứa Frame, Panel các khung chứa thường hay được sử dụng.

#### 6.1.3.1 Frame

- Frame được dùng để xây dựng các ứng dụng GUI chạy độc lập.
- Frame là một cửa sổ có thanh tiêu đề và các đường biên. Bộ cục mặc định của Frame là BorderLayout.
- Frame kế thừa từ Window, nó có thể nghe các sự kiện xảy ra trên cửa sổ khi cài đặt giao tiếp *WindowListener*.
- Các ứng dụng độc lập thường tạo ra cửa sổ kế thừa từ lớp Frame.

#### Một số phương thức:

- add(Component comp)  
Thừa kế từ class Container, dùng để add thêm một component vào khung chứa như label, button,...
- add(Component comp, int index)  
Thêm vào khung chứa ở một vị trí chỉ định
- setSize (width, height);  
Thiết lập kích thước cho frame

- setVisible(boolean):  
Cài đặt việc hiển thị (true) hay không hiển thị cho frame (false)

Ví dụ:

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame{
    public static void main(String[] args){
        Frame frame = new Frame("Example on Frame");
        Label label = new Label("This is a label in Frame",
Label.CENTER);
        frame.add(label, BorderLayout.CENTER);
        frame.setSize(500,500);
        frame.setVisible(true);

        frame.addWindowListener(new
MyWindowListener());
    }
}

// Lop nghe doc lap (external listener)
Class MyWindowListener extends WindowAdapter{
    public void windowClosing(WindowEvent event){
        System.exit(0);
    }
}
```

Kết quả:



Chú ý: Nếu không muốn viết lớp nghe độc lập, có thể viết trực tiếp code trong class DemoFrame như sau:

Thay thế đoạn code:

```
frame.addWindowListener(new MyWindowListener());
```

bởi:

```

frame.addWindowListener(new WindowAdapter()
    {
        // Cài đặt trực tiếp
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
    });

```

Chú ý:

- Các ứng dụng độc lập dùng Frame phải có hàm main và được chạy trực tiếp bằng lệnh java.
- Cần có lệnh setSize, setVisible(true) để có thể hiển thị Frame.
- Cần cài đặt lắng nghe sự kiện đóng để có thể đóng được cửa sổ frame

### 6.1.3.2 Panel

Panel được sử dụng để nhóm một số các thành phần lại với nhau. Panel không thể được nhìn thấy trực tiếp. Do đó, chúng ta cần thêm panel vào một khung chứa có thể hiển thị được như Frame, Applet. Một giao diện có thể có nhiều panel sắp xếp theo một layout nhất định, mỗi panel lại có các component sắp xếp theo một layout riêng

Panel có bố cục mặc định là FlowLayout

Ví dụ:

```

import java.awt.*;
import java.applet.Applet;

public class PanelDemo extends Applet{
    public void init(){
        Choice choice = new Choice();
        choice.add("Red");
        choice.add("Green");
        choice.add("Blue");

        Button ok = new Button("Ok");
        Button cancel = new Button("Cancel");
        // Tạo panel chứa 2 nút bấm Ok, Cancel
        Panel panel = new Panel();
        panel.add(ok);
        panel.add(cancel);

        // Set layout cho khung chứa Applet
    }
}

```

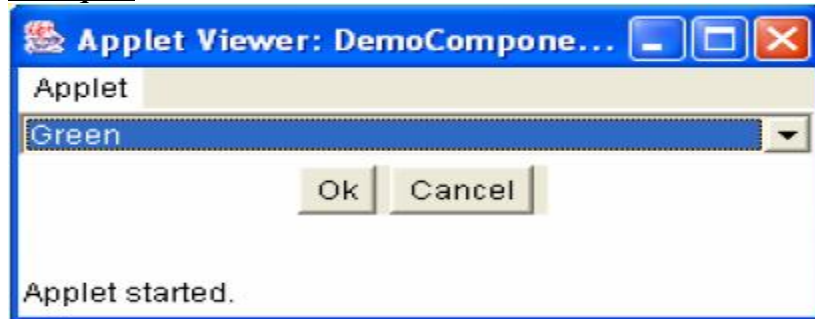
```

this.setLayout(new BorderLayout());

// Thêm các thành phần vào khung chứa Applet
this.add(choice, BorderLayout.NORTH);
this.add(panel, BorderLayout.CENTER);
}
}

```

Kết quả:



#### 6.1.4 Bộ quản lý trình bày – Layout Manager

Có năm bộ quản lý trình bày:

- Flow Layout
- Border Layout
- Grid Layout
- Gridbag Layout
- Null Layout

##### 6.1.4.1 Flow Layout

Đối với một container trình bày theo kiểu FlowLayout thì:

- Các component gắn vào được sắp xếp theo thứ tự từ trái sang phải và từ trên xuống dưới.
- Các component có kích thước như mong muốn.
- Nếu chiều rộng của Container không đủ chỗ cho các component thì chúng tự động tạo ra một dòng mới.
- FlowLayout thường được dùng để sắp xếp các button trong 1 panel.
- Chúng ta có thể điều chỉnh khoảng cách giữa các component

Ví dụ:

```

import java.awt.*;
import java.lang.Integer;
class FlowLayoutDemo
{

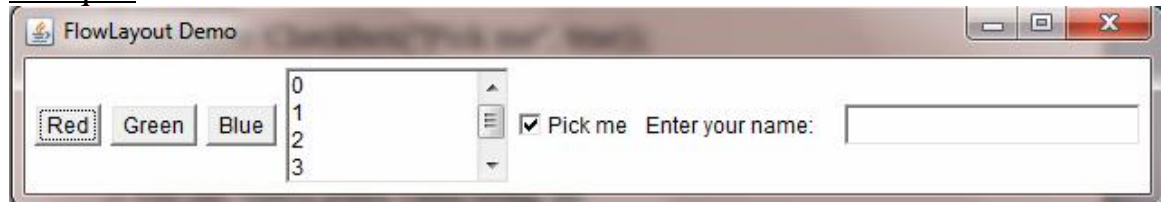
```

```

public static void main(String args[])
{
    Frame fr = new Frame("FlowLayout Demo");
    fr.setLayout(new FlowLayout());
    fr.add(new Button("Red"));
    fr.add(new Button("Green"));
    fr.add(new Button("Blue"));
    List li = new List();
    for (int i=0; i<5; i++)
    {
        li.add(Integer.toString(i));
    }
    fr.add(li);
    fr.add(new Checkbox("Pick me", true));
    fr.add(new Label("Enter your name:"));
    fr.add(new TextField(20));
    // Phương thức pack để điều chỉnh kích thước của số vừa
    // với các thành phần chứa trong nó
    fr.pack();
    fr.setVisible(true);
}
}

```

Kết quả:



#### 6.1.4.2 Border Layout

Đối với một container trình bày theo kiểu BorderLayout thì:

- Bộ trình bày khung chứa được chia làm 5 vùng: NORTH, SOUTH, WEST, EAST và CENTER. Bộ trình bày loại này cho phép sắp xếp và thay đổi kích thước của những components chứa trong nó sao cho vừa với 5 vùng
- Không cần phải gắn component vào cho tất cả các vùng.
- Các component ở vùng NORTH và SOUTH có chiều cao tùy ý nhưng có chiều rộng đúng bằng chiều rộng vùng chứa.
- Các component ở vùng EAST và WEST có chiều rộng tùy ý nhưng có chiều cao đúng bằng chiều cao vùng chứa.



- Các component ở vùng CENTER có chiều cao và chiều rộng phụ thuộc vào các vùng xung quanh.

Ví dụ:

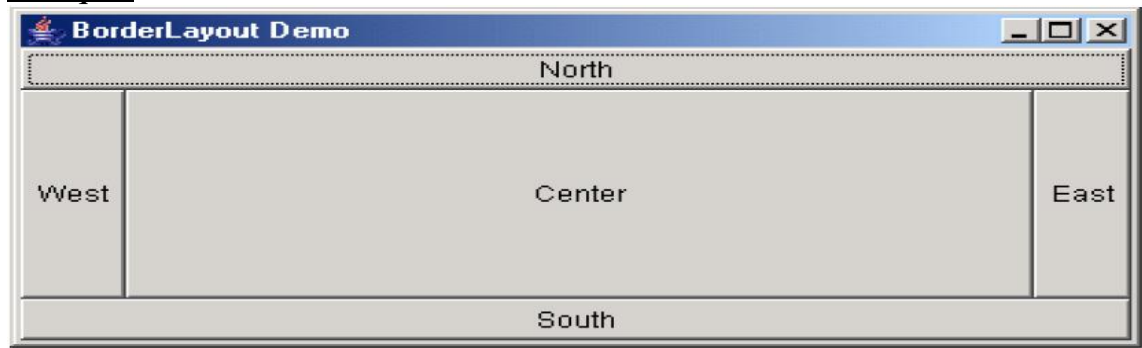
```
import java.awt.*;

class BorderLayoutDemo extends Frame{
    private Button north, south, east, west, center;

    public BorderLayoutDemo(String sTitle){
        super(sTitle);
        north = new Button("North");
        south = new Button("South");
        east = new Button("East");
        west = new Button("West");
        center = new Button("Center");
        this.add(north, BorderLayout.NORTH);
        this.add(south, BorderLayout.SOUTH);
        this.add(east, BorderLayout.EAST);
        this.add(west, BorderLayout.WEST);
        this.add(center, BorderLayout.CENTER);
    }

    public static void main(String args[]){
        Frame fr = new BorderLayoutDemo ("Border Layout Demo");
        fr.pack();
        fr.setVisible(true);
    }
}
```

Kết quả:



### 6.1.4.3 Grid Layout

Đối với một container trình bày theo kiểu GridLayout thì:

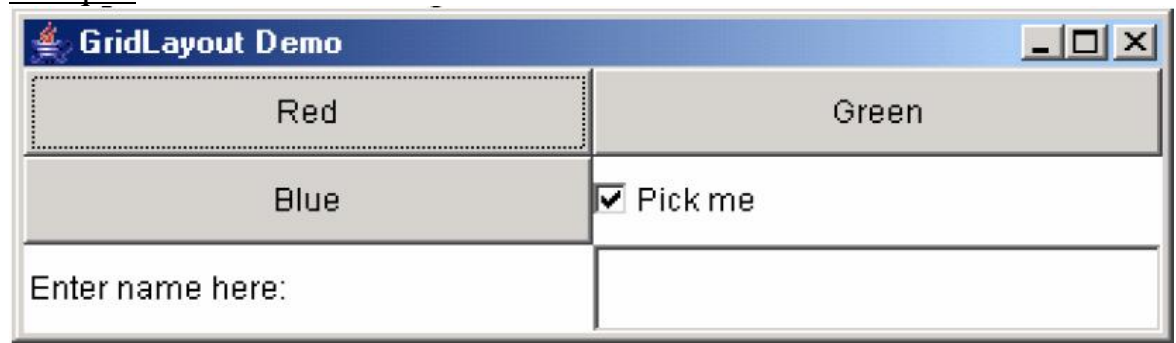
- Bộ trình bày tạo một khung lưới vô hình với các ô bằng nhau.

- Các đối tượng sẽ đặt vừa kích thước với từng ô đó. Thứ tự sắp xếp từ trái qua phải và từ trên xuống dưới

Ví dụ:

```
import java.awt.*;
public class GridLayoutDemo
{
    public static void main(String arg[])
    {
        Frame f = new Frame("GridLayout Demo");
        f.setLayout(new GridLayout(3,2)); // 3 dòng, 2 cột
        f.add(new Button("Red"));
        f.add(new Button("Green"));
        f.add(new Button("Blue"));
        f.add(new Checkbox("Pick me", true));
        f.add(new Label("Enter name here:"));
        f.add(new TextField());
        f.pack();
        f.setVisible(true);
    }
}
```

Kết quả:



#### 6.1.4.4 GridBag Layout

Đối với một container trình bày theo kiểu GridBagLayout thì:

Các components khi được đưa vào khung chứa sẽ được trình bày trên 1 khung lưới vô hình tương tự như GridLayout. Tuy nhiên khác với GridLayout kích thước các đối tượng không nhất thiết phải vừa với 1 ô trên khung lưới mà có thể là 2, 3 ô hay nhiều hơn tùy theo các ràng buộc mà ta chỉ định thông qua đối tượng GridBagConstraints.

Lớp **GridBagConstraints** dẫn xuất từ lớp Object. Lớp GridBagConstraints dùng để chỉ định ràng buộc cho những components trình bày trong khung chứa container theo kiểu GridBagLayout.

**gridx, gridy:** vị trí ô của khung lưới vô hình mà ta sẽ đưa đối tượng con vào o gridwidth, gridheight: kích thước hay vùng trình bày cho đối tượng con.

**Insets:** là một biến đối tượng thuộc lớp Insets dùng để qui định khoảng cách biên phân cách theo 4 chiều (trên, dưới, trái, phải).

**weightx, weighty:** chỉ định khoảng cách lớn ra tương đối của các đối tượng con với nhau

Ví dụ:

```
import java.awt.*;
```

```
import javax.swing.*;
```

```
class GridBagLayoutExample {
```

```
    public static void main(String[] args) {
```

```
        JFrame frame = new JFrame();
```

```
        // Bước phải getContentPane khi sử dụng JFrame
```

```
        Container cp = frame.getContentPane();
```

```
        cp.setLayout(new GridBagLayout());
```

```
        String[] fieldNames = {
```

```
            "Last name", "Forenames", "Country", "Email Address",
```

```
            "Land Line", "Mobile/Cell Phone"
```

```
        };
```

```
        int[] fieldWidths = {20,20,30,30,15,15};
```

```
        GridBagConstraints gbc = new GridBagConstraints();
```

```
        gbc.gridwidth = GridBagConstraints.REMAINDER;
```

```
        gbc.anchor = GridBagConstraints.CENTER;
```

```
        gbc.insets = new Insets(20,0,15,0);
```

```
        cp.add(new JLabel("Personal Information Form"),gbc);
```

```
        gbc.anchor = GridBagConstraints.WEST;
```

```
        gbc.insets = new Insets(5,10,5,5);
```

```
        for(int i=0;i<fieldNames.length;++i) {
```

```
            gbc.gridwidth = GridBagConstraints.RELATIVE;
```

```
            cp.add(new JLabel(fieldNames[i]),gbc);
```

```
            gbc.gridwidth = GridBagConstraints.REMAINDER;
```

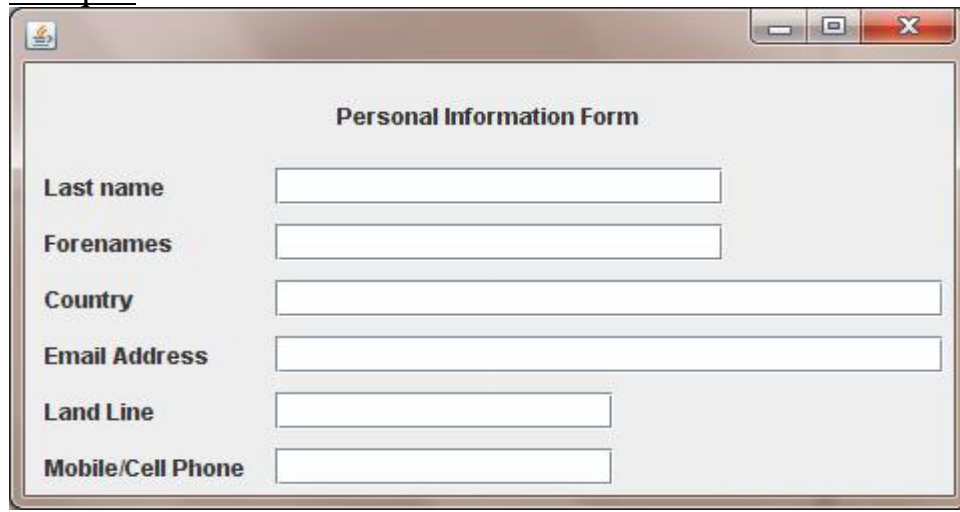
```

        cp.add(new JTextField(fieldWidths[i]),gbc);
    }

    frame.pack();
    // JFrame hỗ trợ thêm phương thức
    // lang nghe sự kiện close mac định
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}
}

```

Kết quả:



The screenshot shows a Java Swing window titled "Personal Information Form". The window has a standard Mac OS X-style title bar with minimize, maximize, and close buttons. Inside the window, there are several text input fields arranged vertically, each with a label to its left: "Last name", "Forenames", "Country", "Email Address", "Land Line", and "Mobile/Cell Phone". The fields are empty and have a light gray border.

#### 6.1.4.5 Null Layout

Một khung chứa được trình bày theo kiểu Null Layout có nghĩa là người lập trình phải tự làm tất cả từ việc qui định kích thước của khung chứa, cũng như kích thước và vị trí của từng đối tượng component trong khung chứa.

Để thiết lập cách trình bày là Null Layout cho một container ta chỉ việc gọi phương thức `setLayout(null)` với tham số là **null**

Một số phương thức của lớp trừu tượng `Component` dùng để định vị và qui định kích thước của component khi đưa chúng vào khung chứa trình bày theo kiểu kiểu tự do:

- `public void setLocation(Point p)`
- `public void setSize(Dimension p)`
- `public void setBounds(Rectangle r)`

**Ví dụ:**

- `MyButton.setSize(new Dimension(20, 10));`
- `MyButton.setLocation(new Point(10, 10));`

- MyButton.setBounds(10, 10, 20, 10);

Ví dụ:

```
import java.awt.*;
class NullLayoutDemo
{
    public static void main(String args[])
    {
        Frame fr = new Frame("NullLayout Demo");
        fr.setLayout(null);
        Button buttOk = new Button("OK");
        buttOk.setBounds(100, 150, 50, 30);
        Button buttCancel = new Button("Cancel");
        buttCancel.setBounds(200, 150, 50, 30);
        Checkbox checkBut = new Checkbox("Check box",
true);

        checkBut.setBounds(100, 50, 100, 20);
        List li = new List();
        for (int i=0; i<5; i++)
        {
            li.add(Integer.toString(i));
        }
        li.setBounds(200, 50, 50, 50);
        fr.add(buttOk);
        fr.add(buttCancel);
        fr.add(checkBut);
        fr.add(li);
        fr.setBounds(10, 10, 400, 200);
        fr.setVisible(true);
    }
}
```

Kết quả:



## 6.2 Applet

### 6.2.1 Cơ bản về Applet

**Khái niệm:** Applet là chương trình Java được tải xuống trong trình duyệt, sau đó nó sẽ chạy bên trong cửa sổ của trình duyệt

#### **Xây dựng một Applet**

Lớp Applet: Trong lập trình giao diện căn bản chúng ta đã biết Java có lớp **java.applet.Applet** kế thừa từ lớp `java.awt.Component` cho phép tạo ra các applet trong Web. Mọi lớp applet do người dùng tạo ra đều phải kế thừa từ lớp Applet.

Ví dụ:

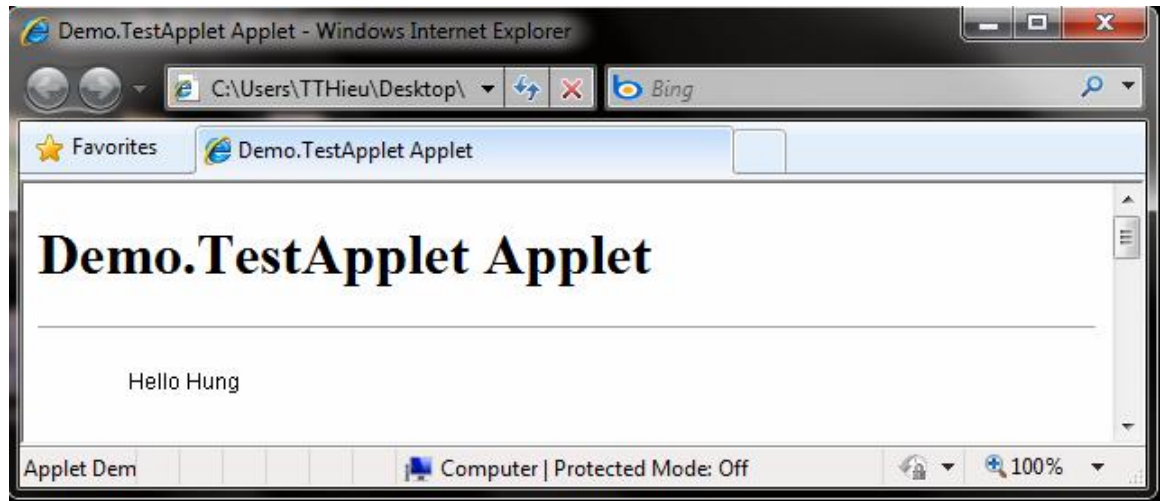
```
import java.applet.Applet;
import java.awt.Graphics;
public class TestApplet extends Applet
{
    String name;
    public void init()
    {
        name = "Hung";
    }

    public void paint(Graphics g)
    {
        g.drawString("Hello " + name, 50, 25);
    }
}
```

Thực thi applet:

- Cách 1: Tạo file TestApplet.html có nội dung như sau:
  - o `<APPLET CODE="TestApplet.class" WIDTH=500 HEIGHT=500 </APPLET>`
  - o Mở file này bằng trình duyệt WEB (internet explorer, firefox)
- Cách 2: Dùng công cụ appletviewer.
  - o Gõ lệnh: `appletviewer TestApplet.html`

Kết quả:



### Hoạt động của một Applet

Một Applet định nghĩa cấu trúc của nó từ 4 sự kiện xảy ra trong suốt quá trình thực thi. Đối với mỗi sự kiện, một phương thức được gọi một cách tự động. Các phương thức này được minh họa trong bảng:

Phương thức	Chức năng
init()	Được gọi trong quá trình khởi tạo applet. Trong quá trình khởi tạo, nó sẽ tạo đối tượng để cung cấp cho applet. Phương thức này được dùng để tải các hình ảnh đồ họa, khởi tạo các biến và tạo các đối tượng.
start()	Được gọi khi một applet bắt đầu thực thi. Một khi quá trình khởi tạo hoàn tất, thì applet được khởi động. Phương thức này được dùng để khởi động lại applet sau khi nó đã ngừng trước đó
stop()	Được gọi khi ngừng thực thi một applet. Một applet bị ngừng trước khi nó bị huỷ.
destroy()	Được dùng để huỷ một applet. Khi một applet bị huỷ, thì bộ nhớ, thời gian thực thi của vi xử lý, không gian đĩa được trả về cho hệ thống.

Ngoài những phương thức cơ bản này, còn có những phương thức 'paint()' và 'repaint()'. Phương thức paint() dùng để hiển thị một đường thẳng (line), text, hoặc một hình ảnh trên màn hình. Đối số của phương thức này là đối tượng của lớp Graphics. Phương thức 'repaint()' được dùng khi cửa sổ cần cập nhật lại. Phương thức này chỉ cần một thông số. Tham số này là đối tượng của lớp Graphics.

Các phương thức của applet `init()`, `start()`, `stop()`, `destroy()`, và `paint()` được thừa kế từ một applet. Nhưng mặc định những phương thức này không thực thi một thao tác nào cả, có thể thực hiện ghi đè để thực hiện các hoạt động cụ thể.

Phương thức `init()` và `paint()` thường được dùng để thực hiện một số hàm để khởi tạo và vẽ applet. Phương thức '`g.drawString()`' chỉ ra vị trí mà đoạn văn bản được vẽ ở đâu trên màn hình.

#### **Vòng đời của một Applet**

- Nạp một applet: applet được khởi tạo và thực thi
- Chuyển hoặc trở về trang Web: Các phương thức `stop` và `start` sẽ được gọi
- Nạp lại applet: như quá trình nạp applet
- Thoát khỏi trình duyệt: phương thức `stop` và `destroy` sẽ được gọi

#### **6.2.2 Chuyển từ một ứng dụng đồ họa sang Applet**

Sau đây là các bước chuyển từ một ứng dụng giao diện đồ họa Java sang Applet để có thể nhúng trong trình duyệt:

- Loại bỏ phương thức `main` trong ứng dụng đồ họa
- Đặt đoạn mã khởi tạo của sổ ứng dụng trong phương thức `init()`, sử dụng phương thức `add` để đặt cửa sổ ứng dụng vào khung chứa Applet
- Loại bỏ các phương thức `setSize`, `setDefaultCloseOperation`, `setTitle`, `setVisible` nếu có. Các phương thức này là không cần bởi Applet có cách thực hiện riêng

#### Ví dụ:

Bạn đã viết được một ứng dụng đồ họa Java có chức năng tính toán của một chiếc máy tính, muốn đưa lên trình duyệt Web để có thể truy cập qua Internet.

Class của ứng dụng là: `CalculatorPanel`

Việc chuyển đổi như sau:

#### Tạo Applet:

```
import java.awt.*;  
import javax.swing.*;  
import java.applet.Applet;
```

```
public class CalculatorApplet extends Applet{  
    public void init(){  
        // Khởi tạo ứng dụng calculator  
        CalculatorPanel panel = new CalculatorPanel();
```



```

        // Thêm vào khung chứa Applet
        add(panel);
    }
}

```

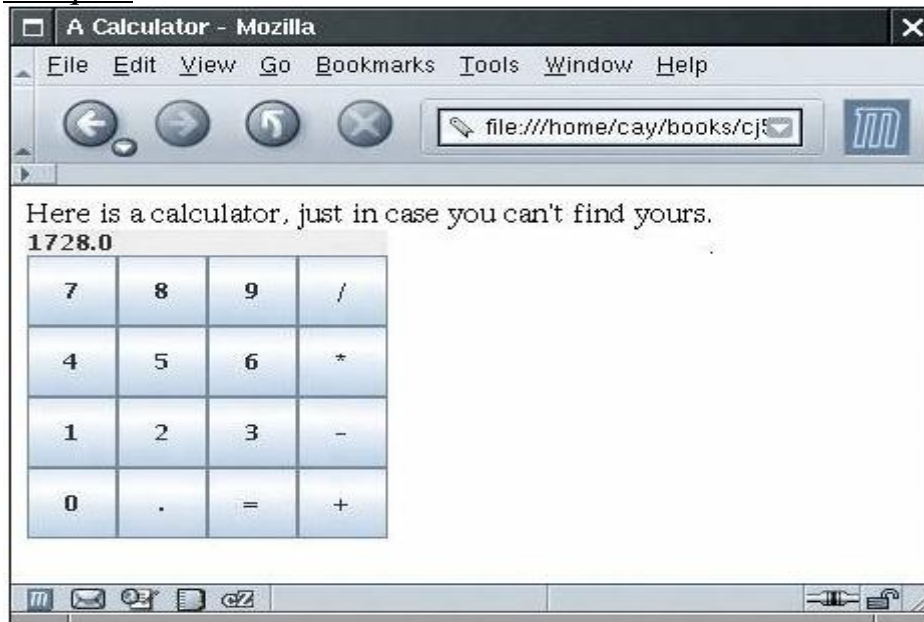
Tạo trang HTML để thực thi:

```

<html>
<head><title>A Calculator</title></head>
<body>
<p>Here is a calculator, just in case you can't find yours.</p>
<applet code="CalculatorApplet.class" width="180" height="180">
</applet>
</body>
</html>

```

Kết quả:



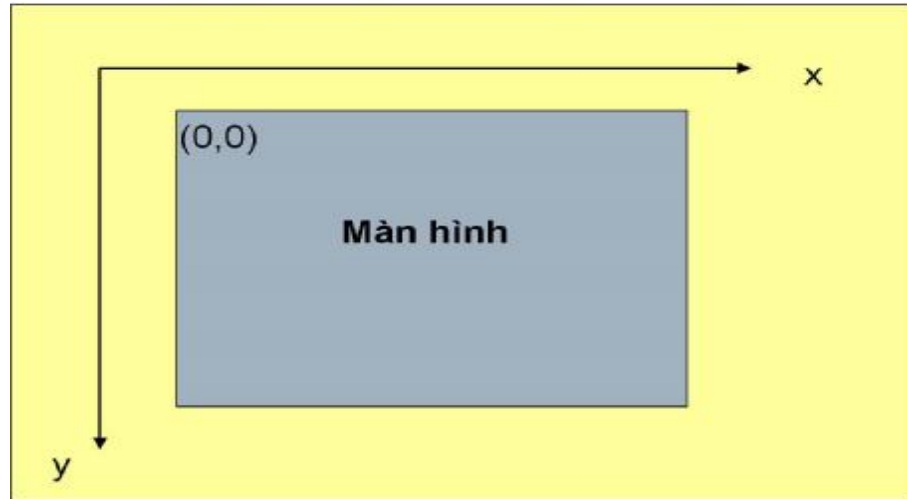
### 6.2.3 Đối tượng đồ họa Graphics

java.awt.Graphics là lớp cung cấp các phương thức vẽ đồ họa cơ bản:

- Đường thẳng (Line)
- Đường oval (Oval)
- Hình chữ nhật (Rectangle)
- Đa giác (Polygon)

- Văn bản(Text)
- Hình ảnh (Image)
- ...

Hệ tọa độ:



Vẽ đường thẳng

```
public void drawLine(int x1, int y1, int x2, int y2);
```

Vẽ hình chữ nhật

```
public void drawRect(int x, int y, int width, int height);
```

Tô một hình chữ nhật

```
public void fillRect(int x, int y, int width, int height);
```

Xoá một vùng chữ nhật

```
public void clearRect(int x, int y, int width, int height);
```

Vẽ đa giác

```
public void drawPolygon(int[] x, int[] y, int numPoint);
```

```
public void drawPolygon(Polygon p);
```

Ví dụ:

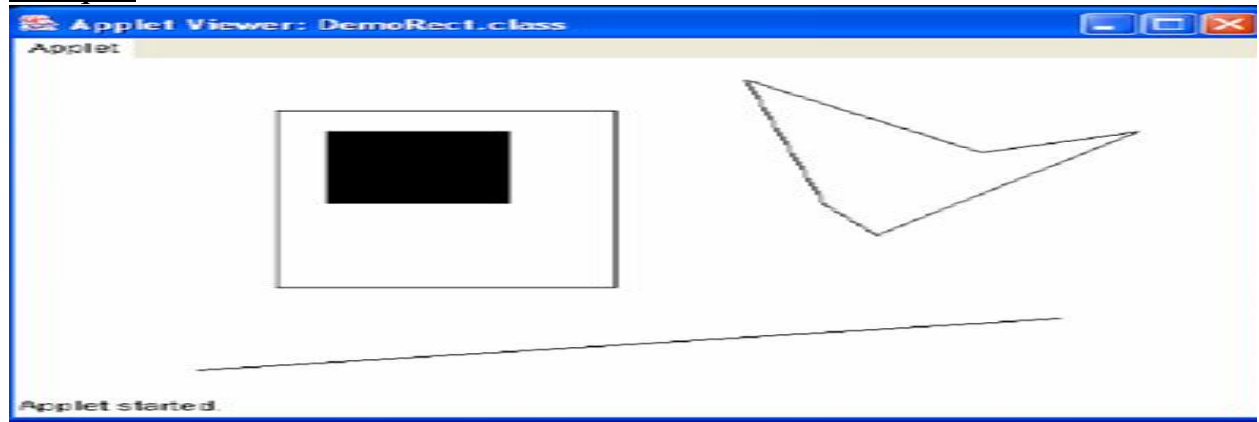
```
import java.applet.Applet;
import java.awt.Graphics;
public class DemoRect extends Applet
{
    public void init(){
        System.out.println("Demonstration of some simple figures");
    }
    public void paint(Graphics g){
        g.drawLine(70, 300, 400, 250);
        g.drawRect(100, 50, 130, 170);
    }
}
```

```

        g.fillRect(120, 70, 70, 70);
        int[] x = { 280, 310, 330, 430, 370 };
        int[] y = { 280, 140, 170, 70, 90 };
        g.drawPolygon(x, y, x.length);
    }
}

```

Kết quả:



Vẽ đường tròn/elip

```
public void drawOval(int x, int y, int width, int height);
```

Tô đường tròn/elip

```
public void fillOval(int x, int y, int width, int height);
```

Vẽ cung tròn

```
public void drawArc(int x, int y, int width, int height, int
                    startAngle, int arcAngle);
```

Vẽ xâu kí tự

```
public void drawString(String str, int x, int y);
```

Vẽ ảnh

```
public void drawImage(Image img, int x, int y,...);
```

Ví dụ: Vẽ ảnh

```

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
public class DemoImage extends Applet
{
    public void init(){
        System.out.println("Demonstration of imaging");
    }
    public void paint(Graphics g){

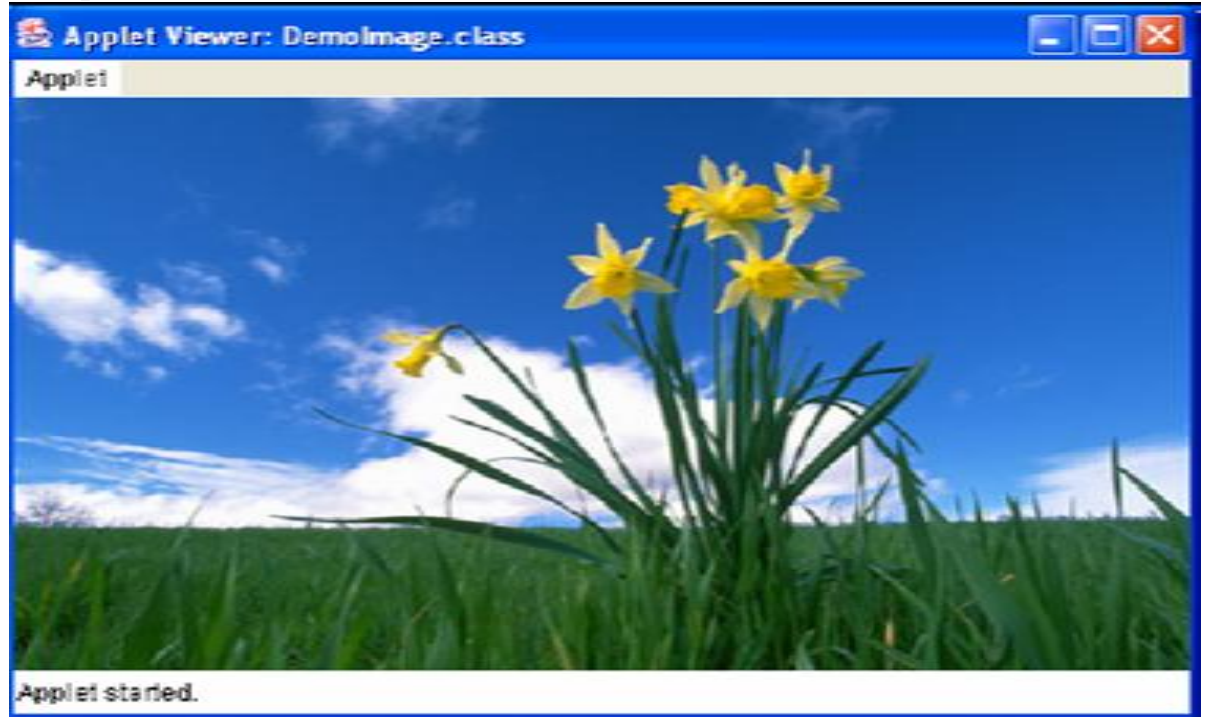
```

```

        Image image = getToolkit().getImage("summer.jpg");
        g.drawImage(image, 0, 0, this);
    }
}

```

Kết quả:



### **Các lớp tiện ích khác:**

**Lớp *Point*:** biểu diễn điểm trên màn hình

**Lớp *Dimension*:** biểu diễn kích thước về chiều rộng và chiều cao của một đối tượng

**Lớp *Rectangle*:** biểu diễn hình chữ nhật

**Lớp *Polygon*:** biểu diễn đa giác

**Lớp *Color*:** biểu diễn màu sắc

**Lớp *Font*:** biểu diễn font chữ

### Ví dụ:

```

import java.applet.Applet;
import java.awt.*;
public class DemoColor extends Applet
{
    public void paint(Graphics g)
    {
        Dimension size = getSize();
    }
}

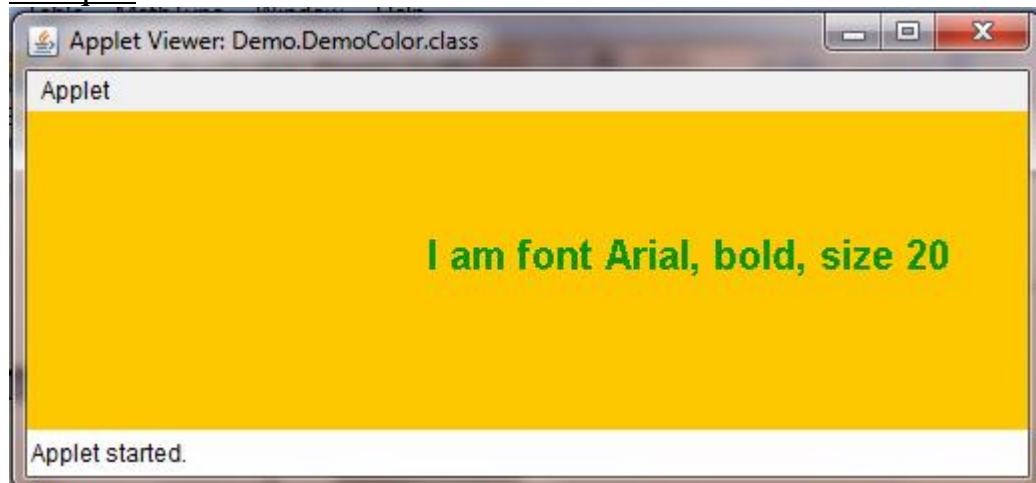
```

```

        g.setColor(Color.orange);
        g.fillRect(0, 0, size.width, size.height);
        Color color = new Color(10, 150, 20);
        g.setColor(color);
        Font font = new Font("Arial", Font.BOLD, 20);
        g.setFont(font);
        g.drawString("I am font Arial, bold, size 20",
                    size.width/2 -50, size.height/2);
    }
}

```

Kết quả:



### **6.3 Bài tập chương 6**

## **CHƯƠNG 7 THAM KHẢO THÊM**

### **7.1 Swing**

### **7.2 Xử lý sự kiện trong lập trình giao diện**

### **7.3 Xử lý ngoại lệ**

### **7.4 Lập trình cơ sở dữ liệu**

### **7.5 Bài tập chương 7**

