

**TRƯỜNG CAO ĐẲNG KỸ THUẬT CAO THẮNG**

**KHOA ĐIỆN TỬ - TIN HỌC**



**BÀI GIẢNG**

**NGÔN NGỮ LẬP TRÌNH**

**JAVA**

(Lưu hành nội bộ)

**TP. HỒ CHÍ MINH, 2017**

# MỤC LỤC

<b>MỤC LỤC.....</b>	<b>II</b>
<b>LỜI GIỚI THIỆU .....</b>	<b>IV</b>
<b>CHƯƠNG 1 - TỔNG QUAN VỀ JAVA.....</b>	<b>1</b>
1.1. GIỚI THIỆU JAVA .....	1
1.2. JAVA IDE TOOLS.....	9
1.3. CÁC THÀNH PHẦN CƠ BẢN .....	16
<b>BÀI TẬP CHƯƠNG 1.....</b>	<b>21</b>
<b>CHƯƠNG 2 - KIỂU DỮ LIỆU VÀ TOÁN TỬ.....</b>	<b>22</b>
2.1. BIẾN VÀ KHAI BÁO BIẾN .....	22
2.2. CÁC KIỂU DỮ LIỆU .....	25
<b>BÀI TẬP CHƯƠNG 2.....</b>	<b>34</b>
<b>CHƯƠNG 3 - CẤU TRÚC ĐIỀU KHIỂN .....</b>	<b>35</b>
3.1. CÁC PHÉP TOÁN LOGIC .....	35
3.2. CẤU TRÚC LỰA CHỌN .....	36
3.3. CẤU TRÚC LẶP.....	42
<b>BÀI TẬP CHƯƠNG 3.....</b>	<b>49</b>
<b>CHƯƠNG 4 - PHƯƠNG THỨC (METHODS) .....</b>	<b>52</b>
4.1. GIỚI THIỆU PHƯƠNG THỨC (METHOD) .....	52
4.2. ĐỊNH NGHĨA PHƯƠNG THỨC .....	53
4.3. CÁCH GỌI PHƯƠNG THỨC.....	53
4.4. VOID METHOD.....	55
4.5. TRUYỀN THAM TRI.....	57
4.6. NẠP CHỒNG PHƯƠNG THỨC (OVERLOADING METHODS) .....	58
4.7. PHẠM VI CỦA BIẾN CỤC BỘ .....	59
<b>BÀI TẬP CHƯƠNG 4.....</b>	<b>61</b>

<b>CHƯƠNG 5 - MẢNG (ARRAYS) .....</b>	<b>62</b>
5.1. GIỚI THIỆU .....	62
5.2. MẢNG MỘT CHIỀU .....	62
5.3. MẢNG HAI CHIỀU.....	72
<b>BÀI TẬP CHƯƠNG 5.....</b>	<b>77</b>
<b>CHƯƠNG 6 - HƯỚNG ĐỐI TƯỢNG TRONG JAVA .....</b>	<b>79</b>
6.1. CÁC KHÁI NIỆM CƠ BẢN.....	79
6.2. LỚP VÀ XÂY DỰNG LỚP.....	84
6.3. ĐẶC ĐIỂM HƯỚNG ĐỐI TƯỢNG TRONG JAVA.....	104
<b>BÀI TẬP CHƯƠNG 6.....</b>	<b>139</b>
<b>CHƯƠNG 7 - ĐỒ HỌA VÀ XỬ LÝ SỰ KIỆN.....</b>	<b>141</b>
7.1. GIỚI THIỆU .....	141
7.2. LẬP TRÌNH ĐỒ HỌA (GUI) VỚI AWT .....	141
7.3. SWING .....	188
<b>BÀI TẬP CHƯƠNG 7.....</b>	<b>196</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>199</b>

## LỜI GIỚI THIỆU

Ngôn ngữ lập trình Java được khởi đầu bởi James Gosling và ban đồng nghiệp ở Sun Microsystems năm 1991 và được phát hành lần đầu vào năm 1995. Sau khi ra đời không lâu, ngôn ngữ lập trình này đã được sử dụng rộng rãi và phổ biến. Một số trường cao đẳng, đại học ở Việt Nam dạy môn lập trình Java như một chuyên đề tự chọn cho các sinh viên công nghệ thông tin giai đoạn chuyên ngành.

Sau một thời gian tìm hiểu, làm việc và tham gia giảng dạy chuyên đề lập trình Java cho sinh viên cao đẳng công nghệ thông tin. Nhóm tác giả chúng tôi quyết định biên soạn cuốn bài giảng này nhằm phục vụ công tác giảng dạy cũng như học tập của sinh viên chuyên ngành công nghệ thông tin.

Nội dung bài giảng tập trung vào những kiến thức cơ bản của ngôn ngữ lập trình Java, cũng như trình bày một số vấn đề về phương pháp lập trình hướng đối tượng trong ngôn ngữ lập trình Java. Để có thể đọc hiểu bài giảng này sinh viên cần nắm vững các kiến thức về: nhập môn lập trình, phương pháp lập trình hướng đối tượng.

Trong bài giảng này, nhóm tác giả có sử dụng tài liệu và bài giảng của các tác giả trong và ngoài nước, cũng như tham khảo trên các website.

Đây là lần đầu tiên xuất bản, nên chắc chắn không thể tránh khỏi những sai sót. Nhóm tác giả rất mong nhận được những ý kiến đóng góp của quý thầy cô, các đồng nghiệp và sinh viên để có thể hoàn thiện hơn bài giảng này phục vụ tốt hơn cho việc học tập của sinh viên.

Trân trọng cảm ơn./.

Tp. Hồ Chí Minh, tháng 11/2017

**Nhóm tác giả**

# CHƯƠNG 1 - TỔNG QUAN VỀ JAVA

## 1.1. Giới thiệu Java

### 1.1.1. Lịch sử Java<sup>1</sup>

Cuối năm 1990, James Gosling và các cộng sự được công ty Sun Microsystems giao nhiệm vụ xây dựng phần mềm 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,...

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 một ngôn ngữ mới dựa trên ngôn ngữ C/C++ và đặt tên là Oak (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.

Năm 1993, Internet và Web bùng nổ, Sun chuyển Oak thành một môi trường lập trình Internet với tên dự án là Java.

Năm 1995, Oak đổi tên với tên chính thức là Java, 20/5/1995, Sun World.

HotJava: Trình duyệt Web hỗ trợ Java đầu tiên.

### Các đặc điểm của Java

- Simple (đơn giản): cú pháp của nó dựa trên C++, nhưng bỏ nhiều đặc điểm gây bối rối như con trỏ tường minh, nạp chồng toán tử,...
- Object-oriented (hướng đối tượng): Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn. Trong Java, mọi thứ đều là Đối tượng;
- Distributed (phân tán): nhằm đến phân bố ứng dụng trên mạng, ứng dụng độc lập platform;
- Interpreted (thông dịch): chương trình nguồn Java có đuôi \*.java, đầu tiên được biên dịch thành tập tin có đuôi \*.class sau đó sẽ được trình thông dịch thông dịch thành mã máy;

---

<sup>1</sup> <http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html>

- Robust (mạnh mẽ): Java đã loại bỏ một số loại cấu trúc lập trình dễ bị lỗi có trong các ngôn ngữ khác, có tính năng xử lý ngoại lệ,...
- Secure (bảo mật): Java hỗ trợ bảo mật rất tốt bởi các thuật toán mã hóa như mã hóa một chiều (one way hashing) hoặc mã hóa công cộng (public key)...;
- Architecture-neutral (kiến trúc trung lập): trình biên dịch Java sinh ra định dạng tập tin đối tượng có kiến trúc trung lập, có khả năng thực thi trên nhiều bộ xử lý, với sự hiện diện của hệ thống thực thi Java;
  - Portable (khả chuyen): có thể mang Java Bytecode tới bất cứ nền tảng nào;
  - Performance (hiệu quả cao): nhờ vào trình thu gom rác (Garbage Collection), giải phóng bộ nhớ đôi với các đối tượng không được dùng đến;
- Multithreaded (đa luồng): trong Java một tiến trình có thể thực hiện nhiều luồng đồng thời;
  - Dynamic (linh động): Java được xem là linh động hơn C/C ++ vì nó được thiết kế để thích ứng với nhiều môi trường phát triển.

### 1.1.2. Cài đặt JDK

#### 1.1.2.1. Các phiên bản JDK (Java Development Kit)<sup>2</sup>

- JDK 1.02 (1995);
- JDK 1.1 (1996);
- JDK 1.2 (1998);
- JDK 1.3 (2000);
- JDK 1.4 (2002);
- JDK 1.5 (2004) (JDK 5 hoặc Java 5);
- JDK 1.6 (2006) (JDK 6 hoặc Java 6);
- JDK 1.7 (2011) (JDK 7 hoặc Java 7);
- JDK 1.8 (2014) (JDK 8 hoặc Java 8);

#### Bao gồm:

- JAVAC: Chương trình dịch chuyển mã nguồn sang ByteCode;

---

<sup>2</sup> <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

- JAVA: Bộ thông dịch → Thực thi Java Application;
- APPLETVIEWER: Bộ thông dịch → Thực thi Java Applet;
- 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.

### 1.1.2.2. JDK Editions

#### ✓ *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.

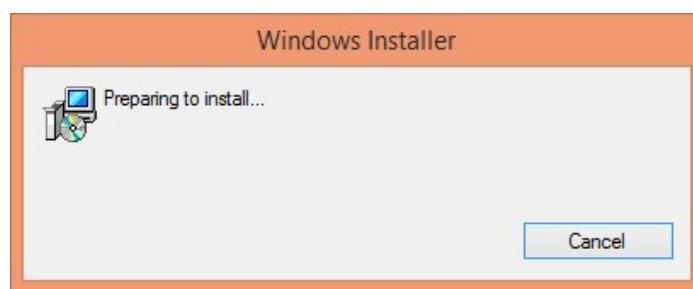
#### ✓ *Mobile (embedded) applications – J2ME*

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

### 1.1.2.3. Cài đặt JDK

Chạy file cài đặt

**Download**<sup>3</sup> được:



Hình 1.1. Chạy file cài đặt download được

---

<sup>3</sup> <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Chọn **Next**:



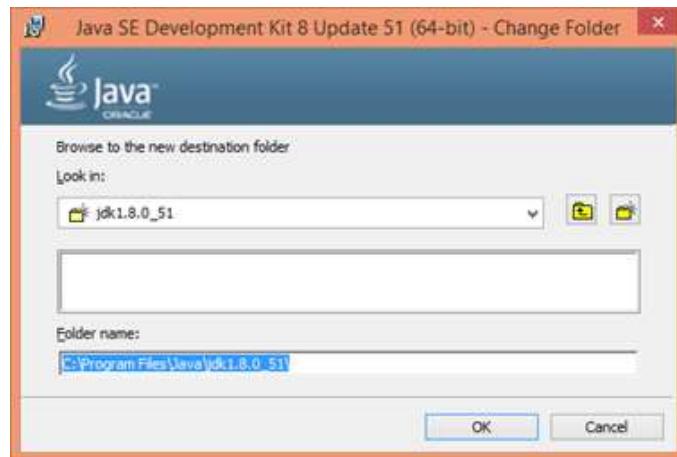
Hình 1.2. Cửa sổ Welcome to the Installation Wizard

Chọn **thư mục** mà **JDK** sẽ được cài đặt:



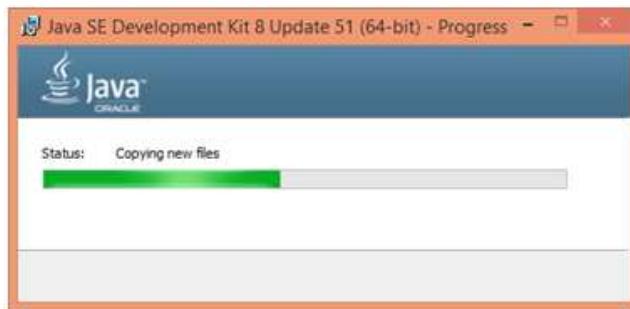
Hình 1.3. Chọn nơi cài đặt JDK

Ở đây chọn mặc định là: “*C:\Program Files\Java\jdk1.8.0\_51\*”



Hình 1.4. Thư mục mà JDK sẽ được cài đặt

Chọn **OK**, tiếp hành cài đặt:



Hình 1.5. Cài đặt JDK

Ngay sau khi cài đặt xong **JDK**, bộ cài đặt sẽ tiếp tục hỏi vị trí **JRE** sẽ được cài đặt. Ở đây chọn mặc định: “*C:\Program Files\Java\jre1.8.0\_51\*”



Hình 1.6. Chọn nơi cài đặt JRE

Chọn **Next**, tiếp tục cài đặt:



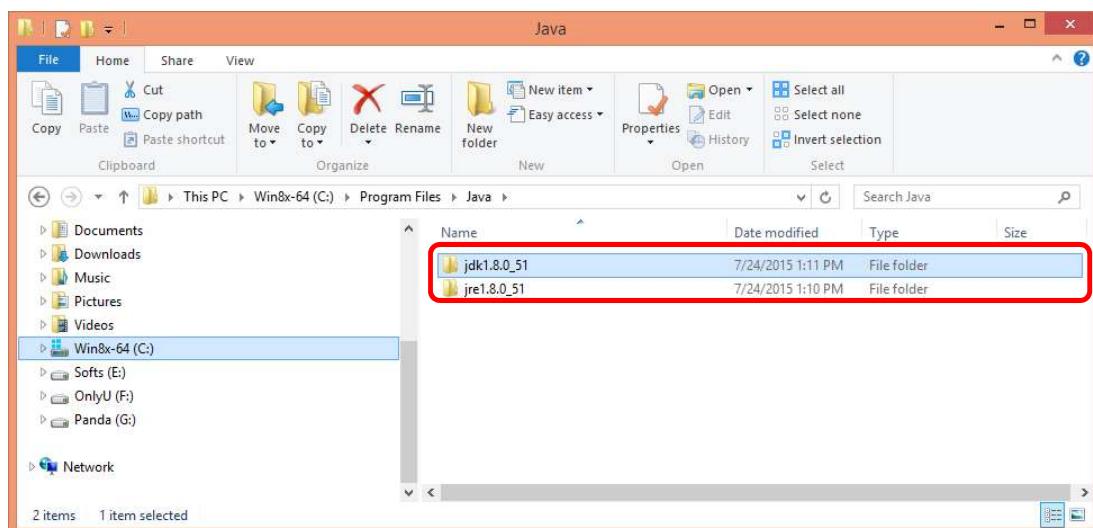
Hình 1.7. Cài đặt JRE

Java đã được cài đặt thành công.



Hình 1.8. Cài đặt thành công

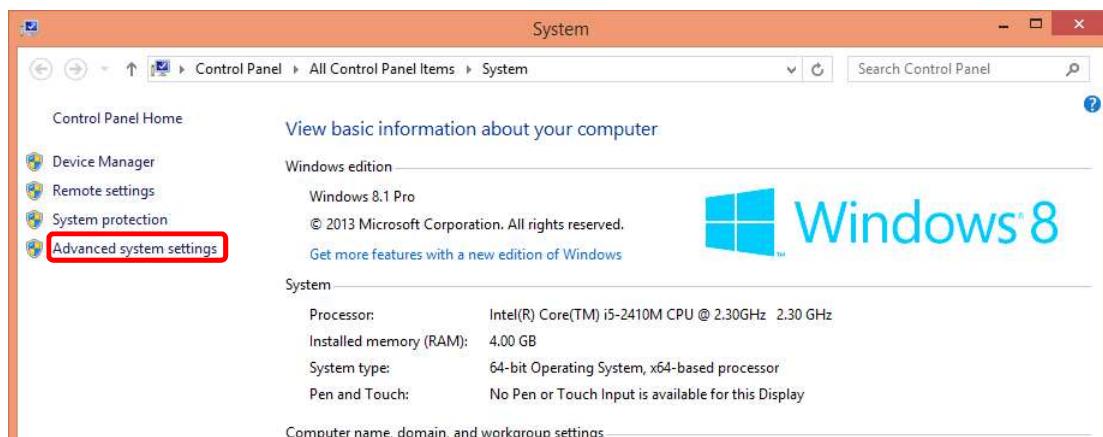
Kết quả có được 2 thư mục:



Hình 1.9. Thư mục sau khi cài đặt thành công

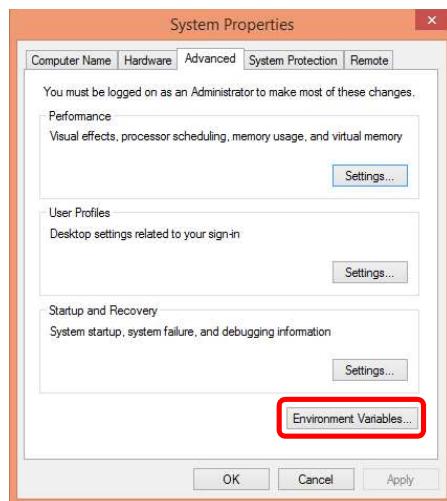
#### 1.1.2.4. Cấu hình biến môi trường cho Java

Chọn **Advanced system settings**:



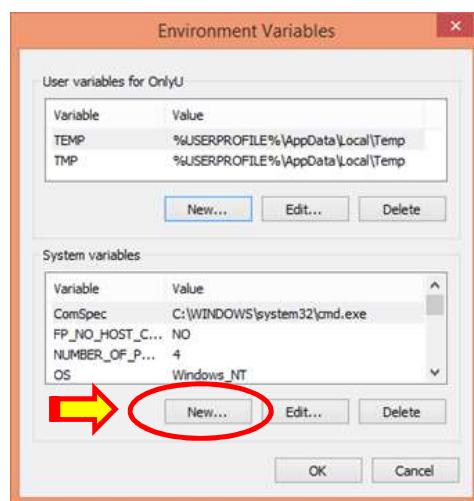
Hình 1.10. System

Chọn **Environment Variables**:



Hình 1.11. System Properties

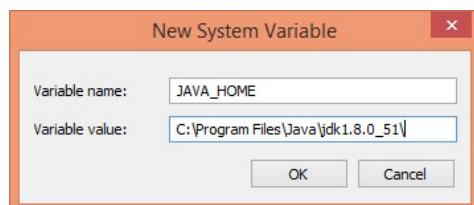
Nhấn **New** để tạo mới một biến môi trường có tên “**JAVA\_HOME**”



Hình 1.12. Environment Variables

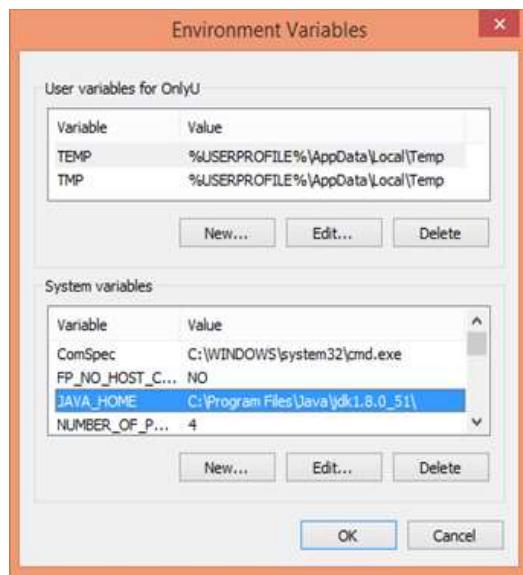
Nhập vào đường dẫn tới thư mục **JDK**.

- Variable name: **JAVA\_HOME**
- Variable value: **C:\Program Files\Java\jdk1.8.0\_51\**



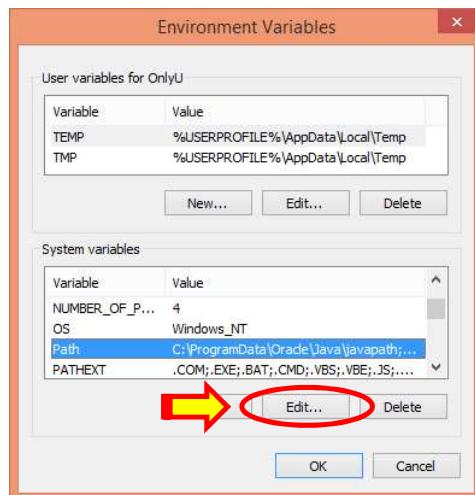
Hình 1.13. New System Variable

Kết quả, tạo được biến môi trường **JAVA\_HOME**:



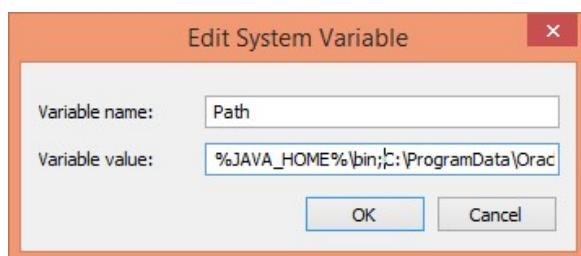
Hình 1.14. Kết quả tạo được biến môi trường **JAVA\_HOME**

Tiếp theo sửa đổi biến môi trường **Path**:



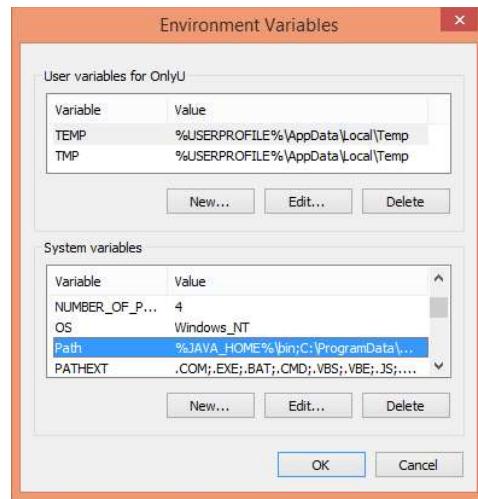
Hình 1.15. Environment Variables

Thêm vào phía trước giá trị của biến môi trường **Path**: **%JAVA\_HOME%\bin;**



Hình 1.16. Sửa đổi biến môi trường Path

Chọn **OK** để kết thúc.



Hình 1.17. Kết quả sau khi cài đặt biến môi trường

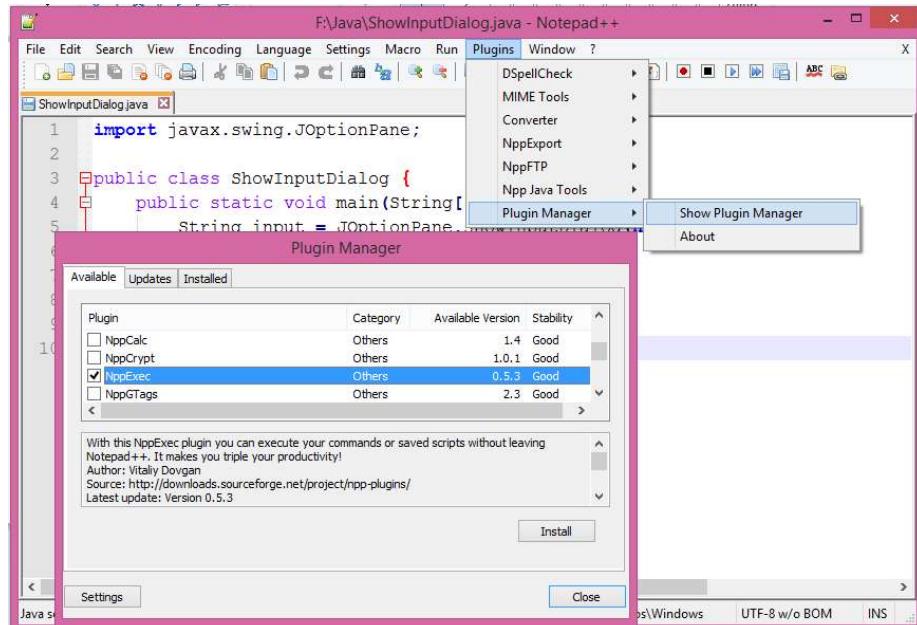
Bạn đã cài đặt và cấu hình Java thành công.

## 1.2. Java IDE Tools

### 1.2.1. NotePad++

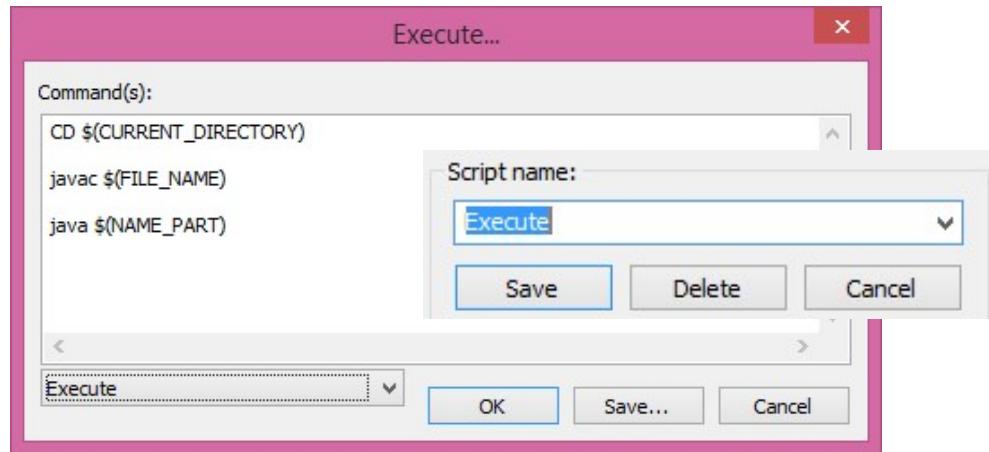
Chạy **notepad++**, vào **Plugins** → **Plugin Manager** → **Show Plugin Manager**.

Trong **Available**, tìm và chọn **NppExec** (như trong hình) và chọn **Install**, sau khi cài đặt xong, xuất hiện thông báo đề nghị khởi động lại **NotePad++**, chọn **Yes**.



Hình 1.18. Add Plugin NppExec

Nhấn **F6**, gõ các lệnh cần thiết rồi nhấn **OK**, để **Compile** và **Run**. Để lưu lại sử dụng về sau, chọn **Save**, và nhập tên rồi nhấn **Save**.



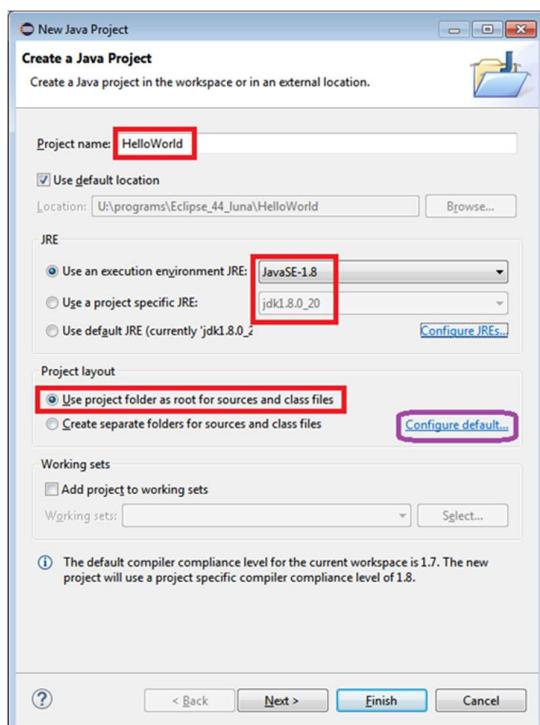
Hình 1.19. Execute

## 1.2.2. Eclipse

### 1.2.2.1. Tạo một Java Project

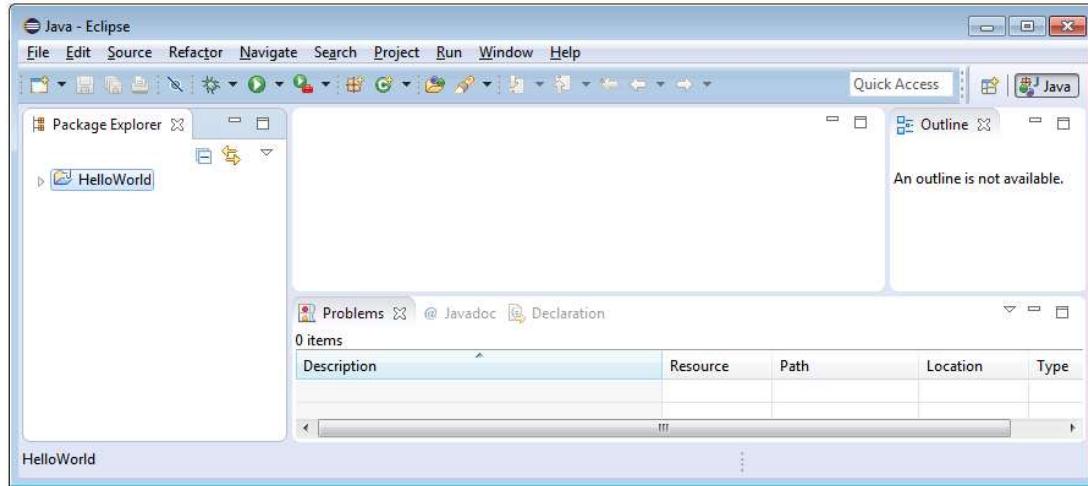
Trước khi tạo một chương trình Java trong **Eclipse**, cần phải tạo một **project**:

- Chọn **File → New → Java Project**, xuất hiện hộp thoại **New Project wizard**, như **Hình 1.20**.



Hình 1.20. Hộp thoại **New Java Project**

2. Nhập tên Project (ví dụ: **HelloWorld**) vào mục **Project name**
3. Check chọn **Use project folder as root for sources and class files**
4. Click chọn **Finish** để tạo, như **Hình 1.21**.

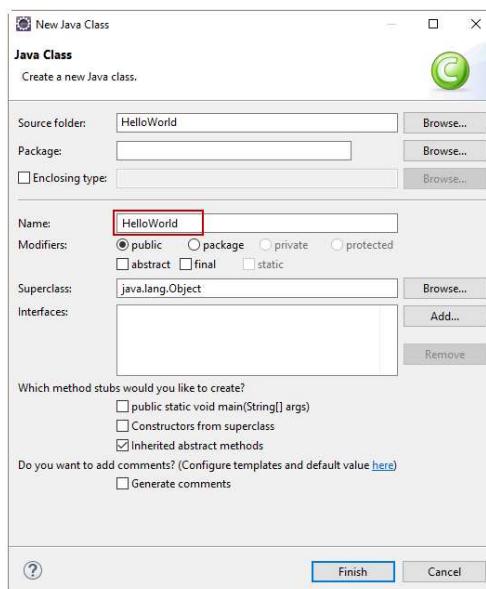


Hình 1.21. **New Java Project** demo đã được tạo

### 1.2.2.2. Tạo một Java Class

Sau khi tạo được *project*, tạo một chương trình Java trong *project* như sau:

1. Chọn **File → New → Class**, xuất hiện hộp thoại **New Java Class**.
2. Nhập tên (ví dụ: **HelloWorld**) trong mục **Name**.
3. Check chọn **public static void main(String[] args)**
4. Click chọn **Finish** để tạo **HelloWorld.java**, như **Hình 1.22**.

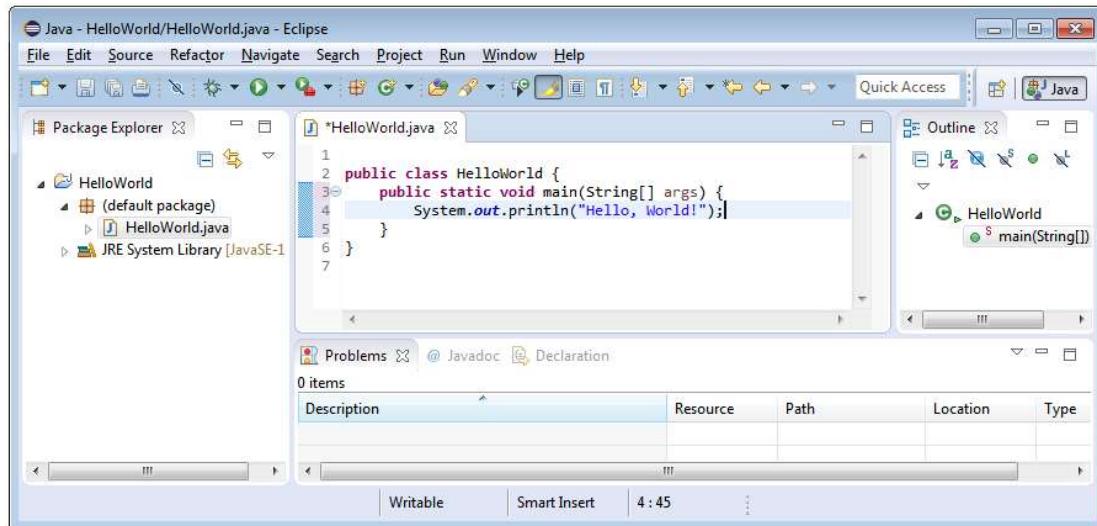


Hình 1.22. Hộp thoại **New Java Class** để tạo **Java class**

### 1.2.2.3. Compiling và Running một Class trong Eclipse

Để chạy chương trình, *right-click* vào **class** trong **project**, xuất hiện một **context menu**. Chọn **Run → Java Application** trong **context menu** để chạy **class**.

Kết quả xuất hiện trong cửa sổ **Console**, như trong **Hình 1.23**.



Hình 1.23. Cửa sổ Edit và chạy chương trình Java trong Eclipse

### 1.2.3. NetBeans

#### 1.2.3.1. Tạo một Java Project

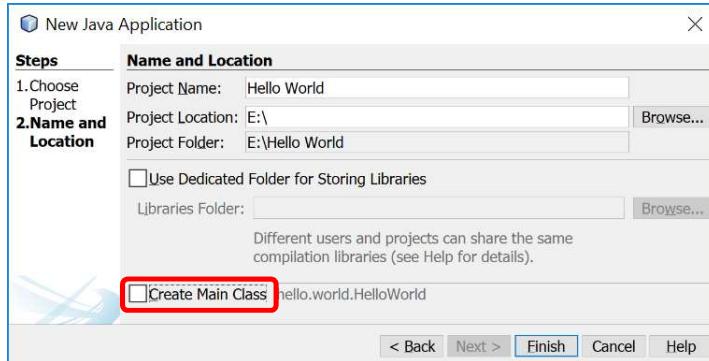
Trước khi tạo một chương trình Java trong **NetBeans**, cần phải tạo một **project**:

- Chọn **File → New Project**, hiện hộp thoại **New Project**, như **Hình 1.25**.



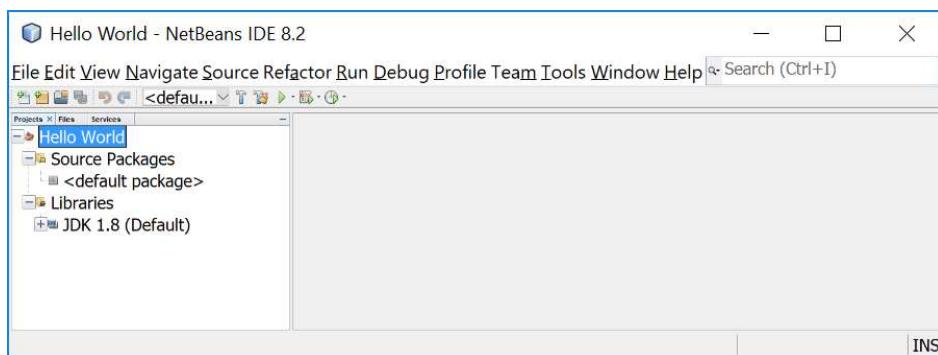
Hình 1.24. Hộp thoại **New Project** để tạo một project mới

2. Chọn **Java** trong **Categories** và **Java Application** trong **Projects** và click **Next** để hiển thị hộp thoại **New Java Application**, như **Hình 1.26**.



Hình 1.25. Hộp thoại **New Java Application**

3. Nhập tên (ví dụ: **Hello World**) trong mục **Project Name** và E:\ trong mục **Project Location**.
4. Click **Finish** để tạo project, như **Hình 1.27**.



Hình 1.26. Một project Java mới với tên demo đã được tạo

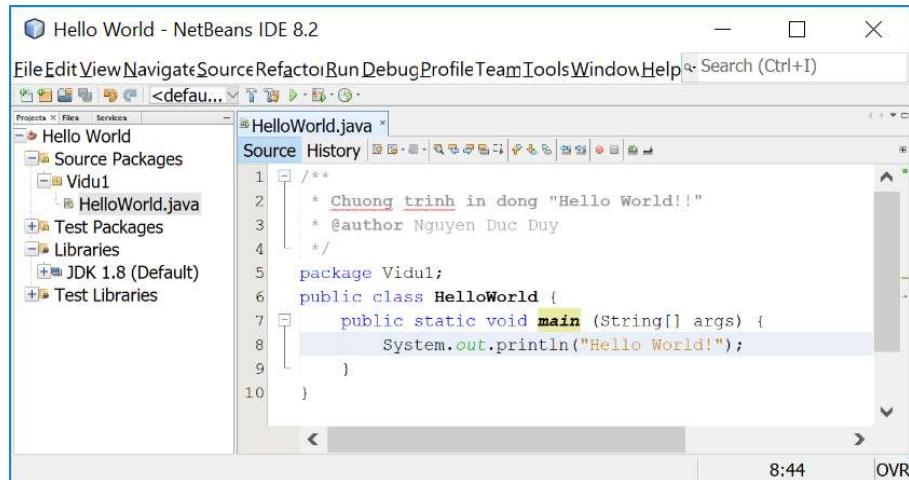
### 1.2.3.2. Tạo một Java Class

1. Right-click vào **Hello World** trong **Project Pane** để hiển thị **context menu**. Chọn **New → Java Class** để hiển thị hộp thoại **New Java Class**, như **Hình 1.28**.



Hình 1.27. Hộp thoại **New Java Class** để tạo **class** mới

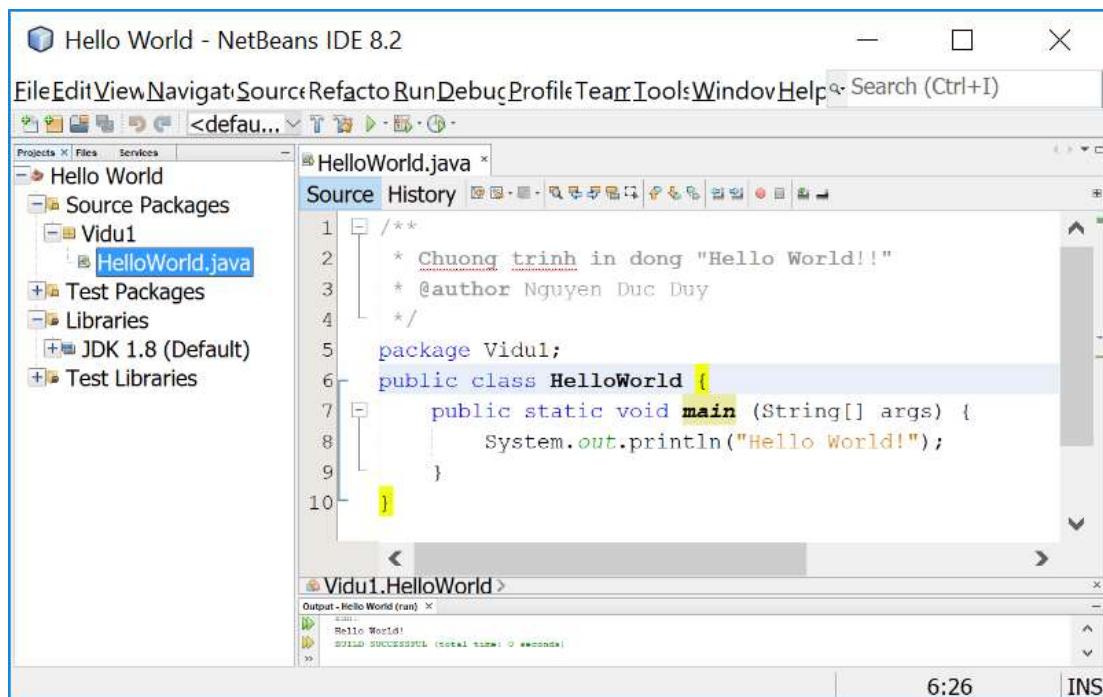
- Nhập **HelloWorld** trong mục **Class Name** và chọn **Source Packages** trong mục **Location**. Để trống mục **Package**, sẽ tạo một lớp trong default **package**.
  - Click **Finish** để tạo **HelloWorld class**. Tập tin **HelloWorld.java** được đặt trong <**package** Vidu1>.
  - Soạn code trong **HelloWorld class**, như **Hình 1.29**.



Hình 1.28. Cửa sổ soạn thảo chương trình và chạy chương trình trong *NetBeans*

### 1.2.3.3. Compiling và Running một Class trong NetBeans

Để chạy `Welcome.java`, right-click `Welcome.java` để hiển thị **context menu** và chọn Run File; Hoặc nhấn Shift + F6.



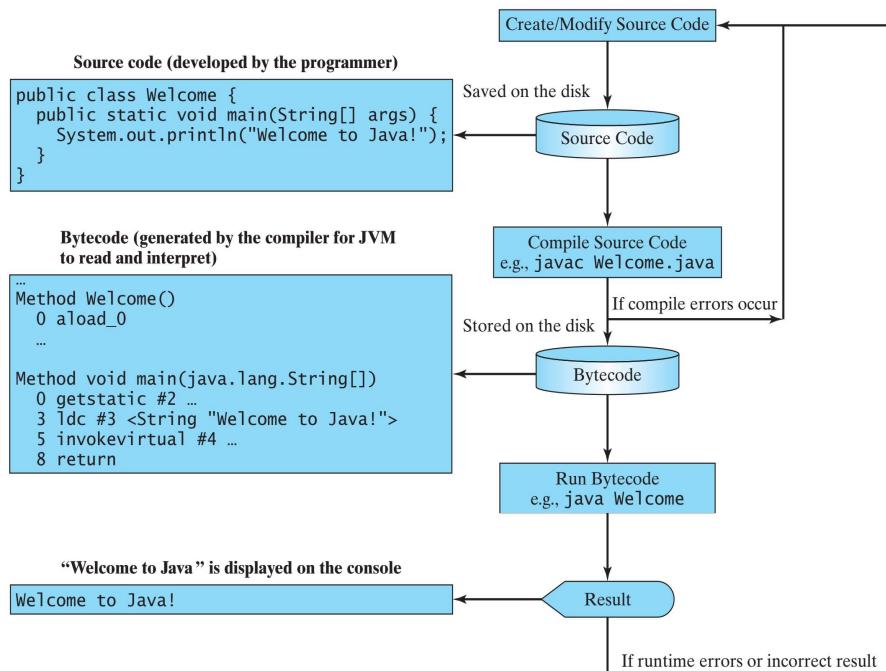
Hình 1.29. Kết quả sẽ hiển thị trong **Output pane**

### 1.2.4. Chương trình Java đầu tiên

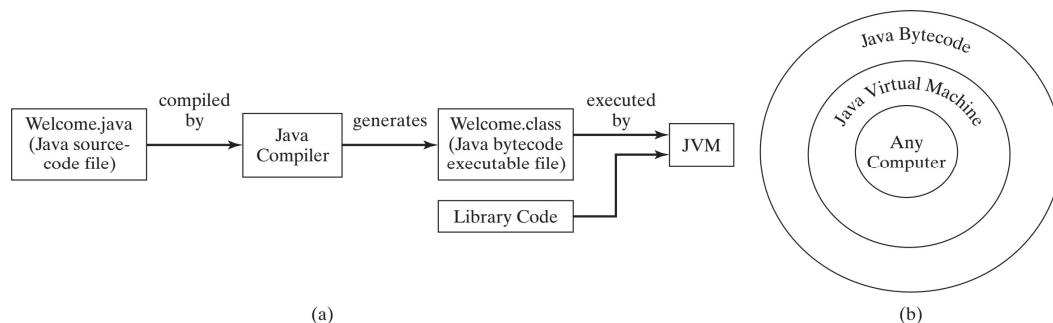
#### 1.2.4.1. Ví dụ 1: *Welcome.java*

```
//Chuong trinh in dong "Welcome to Java!"  
  
package Videl1;  
  
public class Welcome {  
  
    public static void main (String[] args) {  
  
        System.out.println("Welcome to Java!");  
  
    }  
}
```

#### 1.2.4.2. Biên dịch và chạy chương trình:



Hình 1.30. Quá trình phát triển chương trình Java bao gồm việc lặp lại việc tạo/sửa mã nguồn, biên dịch và thực thi chương trình.



Hình 1.31. (a) Mã nguồn Java được dịch sang bytecode. (b) Java bytecode có thể được thực hiện trên bất kỳ máy tính nào với một máy ảo Java.

✓ **Tạo tập tin nguồn Java:**

- Soạn thảo chương trình (Notepad, NotePad++, Wordpad,...)
- Save file tên “**Welcome.java**” vào thư mục (Ví dụ: **E:\Demo**)

✓ **Biên dịch:**

- Trên cửa sổ lệnh (cmd.exe)
- E:\>cd Demo ↵
- E:\Demo>javac Welcome.java ↵

✓ **Chạy:**

- E:\Demo>java Welcome ↵

```
C:\ Command Prompt
C:\Users\dduws>E:
E:\>cd Demo

E:\Demo>dir Welcome.* 
 Volume in drive E is OnlyU
 Volume Serial Number is 103F-6AB5

 Directory of E:\Demo

06-Sep-17  09:04 AM           176 Welcome.java
               1 File(s)      176 bytes
               0 Dir(s)   61,156,515,840 bytes free

E:\Demo>javac Welcome.java

E:\Demo>dir Welcome.* 
 Volume in drive E is OnlyU
 Volume Serial Number is 103F-6AB5

 Directory of E:\Demo

06-Sep-17  09:18 AM           424 Welcome.class
06-Sep-17  09:04 AM           176 Welcome.java
               2 File(s)      600 bytes
               0 Dir(s)   61,156,515,840 bytes free

E:\Demo>java Welcome
Welcome to Java!
```

Hình 1.32. Biên dịch Java dùng Command line

### 1.3. Các thành phần cơ bản

```
//Chương trình in dòng "Hello World!!"

package Videl1;

public class HelloWorld { ← class block

    public static void main(String[] args) { ← method block
        System.out.println("Hello World!");
    }
}
```

### 1.3.1. Chú thích (Comments):

- Trong Java, các chú giải có thể được đặt:
  - Sau 2 dấu gạch chéo `//` trên 1 dòng;
  - Giữa dấu mở `/*` và đóng `*/` trên 1 hoặc nhiều dòng;
- Khi trình biên dịch gấp:
  - `//`, nó bỏ qua tất cả các ký tự sau `//` trên dòng đó;
  - `/*`, nó quét tìm đến `*/` tiếp sau và bỏ qua mọi ký tự giữa `/*` và `*/`.

### 1.3.2. Gói (Package):

- Dòng 2 trong chương trình trên (`package Vidu1;`) xác định tên gói `Vidu1` cho **class** `HelloWorld`. Trình biên dịch biên dịch source code trong tệp `HelloWorld.java`, tạo ra tệp `HelloWorld.class`, và lưu `HelloWorld.class` trong thư mục `Vidu1`.

### 1.3.3. Từ khóa (Reserved words):

- **Reserved words hay keywords** là những từ có nghĩa xác định đối với trình biên dịch và không thể sử dụng cho các mục đích khác trong chương trình.  
*Ví dụ:* `class`, `public`, `static`, và `void`. Chúng sẽ được giới thiệu ở phần sau.

### 1.3.4. Từ bối nghĩa (Modifiers):

- Java sử dụng một số từ khóa gọi là **modifiers** để xác định các thuộc tính của dữ liệu, các phương thức, lớp, và chúng có thể được sử dụng như thế nào.
- Các ví dụ từ bối nghĩa là `public`, `static`, `private`, `final`, `abstract`, và `protected`.
- Một dữ liệu, phương thức, hoặc lớp `public` thì có thể truy nhập được bởi chương trình khác. Một dữ liệu hay phương thức `private` thì không thể.

### 1.3.5. Câu lệnh (Statements):

- Một câu lệnh (**statement**) đại diện cho một hoặc một chuỗi các hành động.
- Mọi câu lệnh trong Java kết thúc bởi một dấu chấm phẩy (`;`).

*Ví dụ:* Câu lệnh `System.out.println("Hello World!");`

### 1.3.6. Khối (Blocks):

- Một cặp dấu ngoặc nhọn `{}` gom các câu lệnh thành một khối lệnh.  
*Ví dụ:* `class block`, `method block`,...

### 1.3.7. Lớp (Classes):

- **Class** (lớp) là thiết yếu trong xây dựng cấu trúc Java. Một **class** là một khuôn mẫu hay bản thiết kế cho các đối tượng.
- Để lập trình trong Java, phải hiểu các **class** và có thể viết, sử dụng chúng.
- Những khái niệm của **class** sẽ tiếp tục được khám phá dần. Nay chỉ cần hiểu một chương trình Java được xác định bằng cách sử dụng một hay nhiều **class**.

### 1.3.8. Phương thức (Methods):

- **System.out.println** là gì? Đó là một phương thức (**method**): tập các câu lệnh thực hiện một chuỗi các thao tác để hiển thị một thông tin trên màn hình.
- **Method** có thể được sử dụng mà không cần hiểu đầy đủ chi tiết nó làm việc như thế nào. Nó được sử dụng bằng cách gọi một câu lệnh với tham số chuỗi ký tự (string) được bao bởi cặp dấu nháy kép. Trong ví dụ, tham số là "**Hello World!**".
- Có thể gọi phương thức **println** với các tham số khác nhau để in ra những message khác nhau.

### 1.3.9. Phương thức main (Main method):

- **Main method** cung cấp sự kiểm soát luồng chương trình. Trình biên dịch Java thực hiện ứng dụng bằng cách gọi đến **main method**.
- Mọi chương trình Java phải có **main method**, nó là điểm khởi đầu khi thực hiện chương trình. Dạng thức của **main method**:

```
public static void main(String[] args) {  
    //Statements;  
}
```

### 1.3.10. Programming Style

#### 1.3.10.1. Chú thích

- Đặt một chú thích đầu chương trình để giải thích chương trình làm việc gì, các đặc điểm của chương trình, các cấu trúc dữ liệu mà chương trình hỗ trợ và các kỹ thuật đặc biệt mà chương trình sử dụng.
- Đặt trong chú thích tên và mô tả rõ ràng về tác giả ở đầu chương trình.
- Đặt chú thích thích hợp giải thích các lớp, các đoạn lệnh,...

### 1.3.10.2. Quy ước đặt tên

- Chọn các tên mô tả và có ý nghĩa.

#### Tên biến và phương thức:

- Dùng chữ thường. Nếu tên có chứa nhiều từ, hãy viết liền nhau, sử dụng chữ thường ở từ thứ nhất và viết hoa ký tự đầu tiên ở các từ tiếp theo.

Ví dụ, các biến *radius* và *area*, phương thức *computeArea*.

#### Tên lớp:

- Dùng chữ Hoa. Viết hoa ký tự đầu tiên của mỗi từ trong tên.

Ví dụ: *ComputeArea*.

#### Tên hằng:

- Dùng chữ hoa. Viết hoa tất cả các ký tự.

Ví dụ: *PI*.

### 1.3.10.3. Thụt đầu dòng và khoảng cách dòng

#### Thụt đầu dòng:

- Dùng tab, khoảng trắng. Thụt vào 1 tab hoặc 2 khoảng trắng.
- Cả 2 phía của mỗi toán tử nên có 1 khoảng trắng,

Ví dụ:

```
boolean b = 3 + 4 * 4 > 5 * (4 + 3) - ++i;
```

#### Khoảng cách dòng:

- Sử dụng dòng trống để ngăn cách các đoạn code.

### 1.3.10.4. Block Styles

- Sử dụng *end-of-line style* cho các dấu ngoặc nhọn:

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

Next-line style

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

End-of-line style

### 1.3.11. Programming Errors

1.3.11.1. Syntax Errors: Do trình biên dịch phát hiện.

```
public class ShowSyntaxErrors {  
    public static main(String[] args) {  
        System.out.println('Welcome to Java');  
    }  
}
```

1.3.11.2. Logic Errors: Dẫn đến kết quả chạy chương trình bị sai.

```
public class ShowLogicErrors {  
    public static void main(String[] args) {  
        System.out.println("Celsius 35 is Fahrenheit degree ");  
        System.out.println((9 / 5) * 35 + 32);  
    }  
}
```

1.3.11.3. Runtime Errors: Gây ra chương trình bị bỏ qua.

```
// ShowRuntimeErrors.java: Program contains runtime errors  
public class ShowRuntimeErrors {  
    public static void main(String[] args) {  
        System.out.println(1 / 0);  
    }  
}
```

## BÀI TẬP CHƯƠNG 1

Bài 1-1. Hãy download và cài đặt JDK và IDE lên máy tính.

Bài 1-2. Viết chương trình hiển thị ra màn hình câu “**Welcom to Java!**”, “**Welcome to Computer Science!**”, “**Welcome to Cao Thang?**”.

Bài 1-3. Viết chương trình hiển thị câu “**Welcome to Java!**” 10 lần.

Bài 1-4. Viết chương trình hiển thị ra màn hình:

```
      J      A      V      V      A
      J      A A    V      V      A A
      J      AAAAA  V V    AAAAA
      J J     A       V     A     A
```

Bài 1-5. Viết chương trình hiển thị ra bảng sau:

a	$a^2$	$a^3$
1	1	1
2	4	8
3	9	27
4	16	64

Bài 1-6. Viết chương trình tính:

$$P = \frac{9.5 * 4.5 - 2.5 * 3}{45.5 - 3.5}$$

$$\pi = 4 * \left( \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \right)$$

Bài 1-7. Viết chương trình hiển thị diện tích và chu vi của đường tròn.

```
chuvi = 2 * bankinh * 3.14;
dientich = bankinh * bankinh * 3.14;
```

## CHƯƠNG 2 - KIỂU DỮ LIỆU VÀ TOÁN TỬ

### 2.1. Biến và khai báo biến

#### 2.1.1. Ví dụ: Tính diện tích hình tròn

```
public class ComputeArea {  
    public static void main(String[] args) {  
        double radius; // Khai bao bien ban kinh -> radius  
        double area; // khai bao bien dien tich -> area  
        radius = 20; // Gan gia tri cho bien radius  
  
        area = radius * radius * 3.14159; // Tinh dien tich ->area  
        // Hien thi ket qua ra man hinh  
        System.out.println("The area for the circle of radius " +  
                           radius + " is " + area);  
    }  
}
```

#### 2.1.2. Đọc dữ liệu nhập từ Console

```
import java.util.Scanner; // Scanner is in the java.util package  
public class ComputeAreaWithConsoleInput {  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in); // Tao duong Scanner  
        System.out.print("Enter a number for radius: "); // Nhap bk  
        double radius = input.nextDouble();  
  
        double area = radius * radius * 3.14159; // Compute area  
        // Hien thi ket qua ra man hinh  
        System.out.println("The area for the circle of radius " +  
                           radius + " is " + area);  
    }  
}
```

Bước 1: Tạo một đối tượng Scanner

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

Bước 2: Sử dụng phương thức nextDouble() để nhận giá trị **double**. Ví dụ,

```
System.out.print("Nhập một số double: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

Bảng 2-1 Các phương thức của đối tượng Scanner

Phương thức	Mô tả
nextByte()	Đọc một số nguyên kiểu <b>byte</b>
nextShort()	Đọc một số nguyên kiểu <b>short</b>
nextInt()	Đọc một số nguyên kiểu <b>int</b>
nextLong()	Đọc một số nguyên kiểu <b>long</b>
nextFloat()	Đọc một số kiểu <b>float</b>
nextDouble()	Đọc một số kiểu <b>double</b>
next()	Đọc một <b>string</b> kết thúc trước một ký tự trắng
nextLine()	Đọc một <b>line of text</b> (kết thúc bằng phím Enter)

### 2.1.3. Tên định danh (Identifiers)

- Tên là một chuỗi các ký tự gồm các chữ, số, dấu gạch dưới (\_), và dấu dollar (\$). Tên phải bắt đầu bởi một chữ, dấu gạch dưới (\_), hoặc dấu dollar (\$). Nó không thể bắt đầu bởi một số;
- Tên không thể là một từ khóa;
- Tên không thể là **true**, **false**, hoặc **null**.
- Tên có thể có độ dài bất kỳ.

### 2.1.4. Biến (Variables) - Khai báo biến (Declaring Variables)

Dạng thức:

```
datatype variableName;
```

Ví dụ:

```
//Khai báo x là một biến nguyên (integer)  
int x;  
  
//Khai báo bankinh là một biến số thực (double)  
double bankinh;  
  
//Khai báo a là một biến ký tự (char)  
char a;
```

### 2.1.5. Lệnh gán và biểu thức gán

Dạng thức:

```
variable = expression;
```

Ví dụ:

```
x = 1;           // Gán số nguyên 1 cho biến x  
bankinh = 1.0;   // Gán số thực 1.0 cho biến ban kinh  
a = 'A';         // Gán ký tự 'A' cho biến a
```

### 2.1.6. Khai báo và khởi tạo trong một lệnh

Dạng thức:

```
datatype variableName = expression;
```

Ví dụ:

```
int x = 1;  
double d = 3.6;
```

### 2.1.7. Hằng (Constants)

Dạng thức:

```
final datatype CONSTANTNAME = VALUE;
```

Ví dụ:

```
final double PI = 3.14159;  
final int SIZE = 3;
```

## 2.2. Các kiểu dữ liệu

### 2.2.1. Dữ liệu kiểu số

Bảng 2-2 Kiểu dữ liệu Numeric

Kiểu	Giá trị	Size
<b>byte</b>	$-2^7$ đến $2^7 - 1$ (-128 đến 127)	8-bit
<b>short</b>	$-2^{15}$ đến $2^{15} - 1$ (-32768 đến 32767)	16-bit
<b>int</b>	$-2^{31}$ đến $2^{31} - 1$ (-2147483648 đến 2147483647)	32-bit
<b>long</b>	$-2^{63}$ đến $2^{63} - 1$ (-9223372036854775808 đến 9223372036854775807)	64-bit
<b>float</b>	Âm: $-3.4028235E+38$ đến $-1.4E-45$ Đương: $1.4E-45$ đến $3.4028235E+38$	32-bit IEEE 754
<b>double</b>	Âm: $-1.7976931348623157E+308$ đến $-4.9E-324$ Đương: $4.9E-324$ đến $1.7976931348623157E+308$	64-bit IEEE 754

### 2.2.2. Các toán tử (Operations)

#### 2.2.2.1. Các toán tử số học

Bảng 2-3 Các toán tử số học

Ký hiệu	Phép toán	Ví dụ	Kết quả
+	Cộng (Addition)	$25 + 3$	28
-	Trừ (Subtraction)	$25.0 - 0.3$	24.7
*	Nhân (Multiplication)	$25 * 3$	75
/	Chia (Division)	$5.0 / 2.0$	2.5
%	Chia dư (Remainder)	$25 \% 3$	1

Phép chia lấy phần dư (**Remainder**) rất hữu ích trong lập trình. Ví dụ, một số chẵn % 2 luôn bằng 0 và một số lẻ % 2 luôn bằng 1.

*Ví dụ: DisplayTime.java*

```
import java.util.Scanner;

public class DisplayTime {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter an integer for seconds: ");
        int seconds = input.nextInt();
        int minutes = seconds / 60; //Find minutes in seconds
```

```

        int remainingSeconds = seconds % 60; //Seconds remaining

        System.out.println(seconds + " seconds is " + minutes
        + " minutes and " + remainingSeconds + " seconds");
    }
}

```

**Chú ý:** Các phép tính với số dấu chấm động được lấy xấp xỉ vì chúng được lưu trữ không hoàn toàn chính xác.

**Ví dụ:**

```
System.out.println(1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
```

sẽ hiển thị **0.5000000000000001**, không phải **0.5**.

```
System.out.println(1.0 - 0.9);
```

sẽ hiển thị **0.0999999999999998**, không phải **0.1**.

### 2.2.2.2. Các toán tử gán tắt

Bảng 2-4 Các toán tử gán tắt

Ký hiệu	Phép toán	Ví dụ	Kết quả
<b><code>+=</code></b>	Gán cộng ( <i>Addition assignment</i> )	<code>i += 3</code>	<code>i = i + 3</code>
<b><code>-=</code></b>	Gán trừ ( <i>Subtraction assignment</i> )	<code>i -= 3</code>	<code>i = i - 3</code>
<b><code>*=</code></b>	Gán nhân ( <i>Multiplication assignment</i> )	<code>i *= 3</code>	<code>i = i * 3</code>
<b><code>/=</code></b>	Gán chia ( <i>Division assignment</i> )	<code>i /= 3</code>	<code>i = i / 3</code>
<b><code>%=</code></b>	Gán chia dư ( <i>Remainder assignment</i> )	<code>i %= 3</code>	<code>i = i % 3</code>

### 2.2.2.3. Các toán tử tăng và giảm

Bảng 2-5 Các toán tử tăng và giảm

Ký hiệu	Phép toán	Mô tả	Ví dụ (giả sử $i = 1$ )
<b><code>++biến</code></b>	Tăng trước	Tăng “ <b>biến</b> ” thêm 1 trước, sau đó thực hiện câu lệnh	<code>int j = ++i //j = 2, i = 2</code>
<b><code>biến++</code></b>	Tăng sau	Thực hiện câu lệnh trước, sau đó tăng “ <b>biến</b> ” thêm 1	<code>int j = i++ //j = 1, i = 2</code>
<b><code>--biến</code></b>	Giảm trước	Giảm “ <b>biến</b> ” bớt 1 trước, sau đó thực hiện câu lệnh	<code>int j = --i //j = 0, i = 0</code>
<b><code>biến--</code></b>	Giảm sau	Thực hiện câu lệnh trước, sau đó giảm “ <b>biến</b> ” bớt 1	<code>int j = i— //j = 1, i = 0</code>

Ví dụ:

```
int i = 10;
int newNum = 10 * i++;
System.out.print("i is " + i
+ ", newNum is " + newNum);
```

tương ứng với

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
System.out.print("i is " + i
+ ", newNum is " + newNum);
```

tương ứng với

```
i = i + 1;
int newNum = 10 * i;
```

- Sử dụng các toán tử tăng và giảm giúp các biểu thức ngắn gọn hơn, nhưng cũng làm cho chúng phức tạp và khó đọc hơn.
- Nên tránh sử dụng các toán tử này trong những biểu thức làm thay đổi nhiều biến hoặc sử dụng cùng một biến nhiều lần như sau: [int k = ++i + i](#).

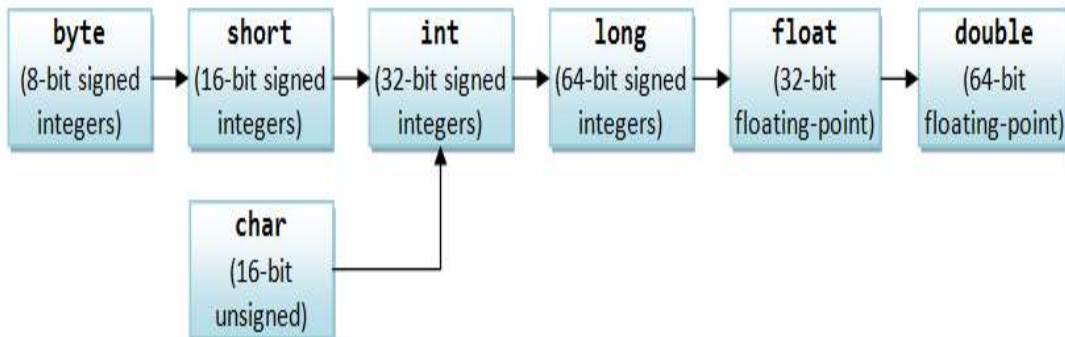
#### 2.2.2.4. Chuyển đổi dữ liệu kiểu số ([ép kiểu](#))

```
byte i = 100;
long k = i * 3 + 2;
double d = i * 3.1 + k/2;
int x = k;           // Sai, int < long
long k = x;          // Đúng, long > int
```

**Luật chuyển:** Khi thực hiện một phép tính nhị phân chứa 2 toán hạng khác kiểu, Java tự động chuyển kiểu toán hạng theo luật sau:

1. Nếu một toán hạng kiểu **double**, các toán hạng khác chuyển thành **double**.
2. Nếu không thì, nếu một toán hạng kiểu **float**, các toán hạng khác được chuyển đổi thành kiểu **float**.
3. Nếu không thì, nếu một toán hạng kiểu **long**, các toán hạng khác được chuyển đổi thành kiểu **long**.
4. Nếu không thì, cả hai toán hạng được chuyển đổi thành kiểu **int**.

### Mức ưu tiên ép kiểu



Hình 2.1 Thứ tự ưu tiên ép kiểu ngầm định

#### Ép kiểu ngầm định và ép kiểu tương ứng

- Ép kiểu ngầm định: **double d = 3;** (mở rộng kiểu)
- Ép kiểu tương ứng: **int i = (int) 3.0;** (thu hẹp kiểu)

### 2.2.3. Kiểu dữ liệu ký tự

```

char letter = 'A';           // ASCII
char numChar = '4';          // ASCII
char letter = '\u0041';        // Unicode
char numChar = '\u0034';        // Unicode
  
```

Với các ký tự đặc biệt: **char tab = '\t';**

Bảng 2-6 Các ký tự đặc biệt

<i>Character</i>	<i>Name</i>	<i>Unicode Code</i>	<i>Decimal</i>
\b	Backspace / Xóa lùi	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed/ Xuống dòng	\u000A	10
\f	Formfeed / Đẩy trang	\u000C	12
\r	Carriage Return / Dấu Enter	\u000D	13
\	Backslash / Dấu \	\u005C	92
\"	Double Quote / Nháy kép	\u0022	34

### Ép kiểu giữa kiểu ký tự và kiểu số

```
int i = 'a';           //Tương tự: int i = (int)'a';
char c = 97;           //Tương tự: char c = (char) 97;
```

#### 2.2.4. Kiểu boolean và các toán tử

##### Kiểu boolean

- Đây là kiểu dữ liệu chỉ nhận một trong 2 giá trị: `true` hoặc `false`.
- Giá trị khởi tạo mặc định của kiểu `boolean` là `false`.

```
boolean a1 = true;
boolean a2 = false;
boolean b1 = (1 > 2);      //b1 = false
boolean b2 = (1 < 2);      //b2 = true
```

Bảng 2-7 Kết quả của phép so sánh là một giá trị logic `Boolean`: `true` hoặc `false`

Ký hiệu	Phép toán	Ví dụ (bankinh = 5)	Kết quả
<	Nhỏ hơn	<code>bankinh &lt; 0</code>	<code>false</code>
<=	Nhỏ hơn hoặc bằng	<code>bankinh &lt;= 0</code>	<code>false</code>
>	Lớn hơn	<code>bankinh &gt; 0</code>	<code>true</code>
>=	Lớn hơn hoặc bằng	<code>bankinh &gt;= 0</code>	<code>true</code>
==	Bằng	<code>bankinh == 0</code>	<code>false</code>
!=	Khác	<code>bankinh != 0</code>	<code>true</code>

#### 2.2.5. Thứ tự ưu tiên các toán hạng

Biểu thức sau được tính như thế nào?

```
3 + 4 * 4 > 5 * (4 + 3) - ++i
```

- Tất nhiên phải ưu tiên trong ngoặc trước, ngoài ngoặc sau;
- Nhiều tầng ngoặc thì thứ tự ưu tiên ngoặc từ trong ra ngoài;

##### Thứ tự ưu tiên

1. `var++`, `var--`

2. `+`, `-` (dấu dương, âm); `++var`, `--var`

3. (type) Casting (ép kiểu)
4. ! (Not)
5. \*, /, % (nhân, chia thường, chia lấy phần dư)
6. +, - (cộng, trừ)
7. <, <=, >, >= (so sánh)
8. ==, !=
9. & (AND không có điều kiện)
10. ^ (Exclusive OR)
11. | (OR không có điều kiện)
12. && (AND có điều kiện)
13. || (OR có điều kiện)
14. =, +=, -=, \*=, /=, %= (toán tử gán)

#### 2.2.6. Sự kết hợp toán tử (Operator Associativity)

✓ Khi tính toán với 2 toán hạng có cùng mức ưu tiên, sự kết hợp toán tử sẽ xác định thứ tự các phép tính. Tất cả các toán tử nhị phân, ngoại trừ toán tử gán, là kết hợp trái (*left-associative*).

- Biểu thức  $a - b + c - d$  là tương đương với  $((a - b) + c) - d$
- Các toán tử gán là kết hợp phải. Do đó biểu thức:

$$a = b += c = 5 \text{ tương đương với } a = (b += (c = 5))$$

- ✓ Luật tính biểu thức:

Luật 1: Tính bất kỳ biểu thức con nào có thể tính được từ trái sang phải.

Luật 2: Các toán hạng được áp dụng theo thứ tự ưu tiên của chúng.

Luật 3: Luật kết hợp áp dụng cho 2 toán hạng cạnh nhau cùng mức ưu tiên.

✓ Các quy tắc ưu tiên và kết hợp xác định thứ tự của các toán tử, nhưng không xác định thứ tự tính toán của các toán hạng nhị phân. Trong Java, các toán hạng được tính từ trái sang phải.

- ✓ Toán hạng bên trái của một toán tử nhị phân được tính trước bất kỳ phần nào của toán hạng bên phải.
  - ✓ Luật này có quyền ưu tiên hơn các luật đã nêu.
    - Khi các toán hạng có hiệu ứng lè (*side effects*), thứ tự tính toán của các toán hạng rất cần quan tâm.

```
int a = 0;  
  
int x = a + (++a); //0 + 1
```

- Nhưng x sẽ bằng 2 trong đoạn lệnh sau, vì ++a tăng nó lên thành 1, rồi cộng với chính nó.

```
int a = 0;  
  
int x = ++a + a; //1 + 1
```

### *Ví dụ:*

**3 + 4 \* 4 > 5 \* (4 + 3) - i**

$3 + 4 * 4 > 5 * (4 + 3) - 1$

↑  
Biểu thức con đầu tiên có thể được tính từ bên trái

$3 + 16 > 5 * (4+3) - 1$

↑  
(2) Cộng

$3 + 16 > 5 * (4+3) - 1$

↑  
(3) Cộng trong ngoặc

$19 > 5 * 7 - 1$

↑  
(4) Nhân

$19 > 35 - 1$

↑  
(5) Trừ

$19 > 34$

↑  
(6) Lớn hơn

false

### 2.2.7. Chuỗi (*String*)

#### **2.2.7.1. Chuỗi và khai báo chuỗi**

- Kiểu **char** chỉ biểu diễn một ký tự. Để biểu diễn một chuỗi ký tự, sử dụng kiểu dữ liệu **String**.

Ví dụ:

```
String message = "Welcome to Java!";
```

- **String** là một lớp được định nghĩa trước trong thư viện Java giống như **System class** và **JOptionPane class**.
  - Kiểu **String** không phải là một kiểu cơ sở mà là một kiểu tham chiếu (*reference type*). Bất kỳ lớp Java nào cũng có thể được sử dụng như một kiểu tham chiếu thay cho một biến.
    - Hiện tại, chỉ cần hiểu cách khai báo một biến **String**, cách gán một chuỗi ký tự cho một biến, và cách ghép các chuỗi.

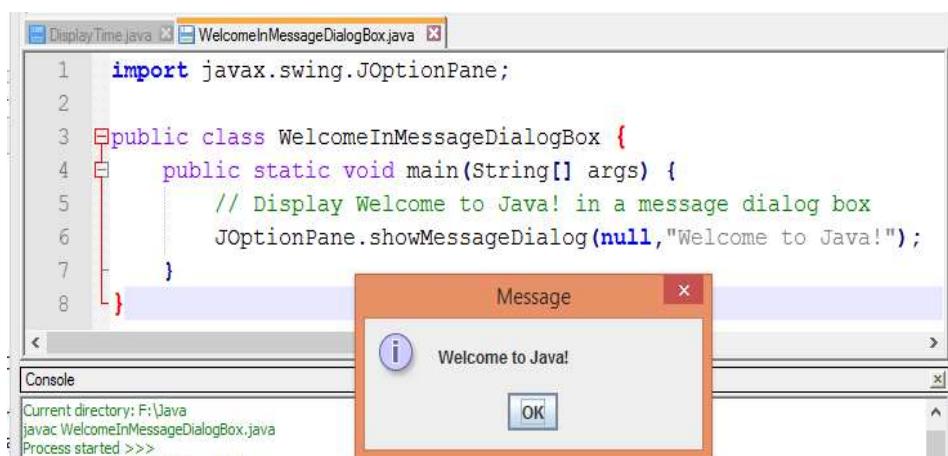
### 2.2.7.2. Ghép chuỗi

```
String message = "Welcome " + "to " + "Java!";
    // message = "Welcome to Java!"
String s = "Chuong" + 2;
    // s trở thành Chuong2
String s1 = "Hello" + ' World!';
    // s1 trở thành Hello World!
```

### 2.2.7.3. Dialog Box

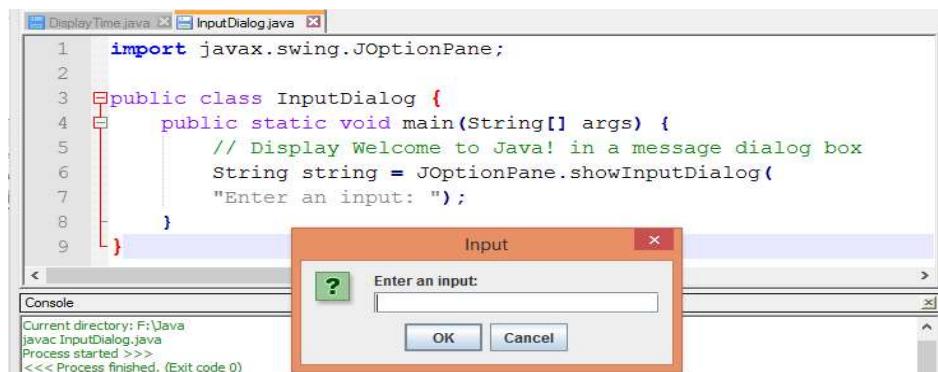
Có thể sử dụng phương thức **showMessageDialog** trong lớp **JOptionPane**. **JOptionPane** là một trong nhiều lớp được định nghĩa trước trong hệ thống Java để có thể tái sử dụng.

#### Phương thức **showMessageDialog**



Hình 2.2 Message Dialog

## Nhập dữ liệu từ Input Dialog Box



Hình 2.3 Input Dialog

### Chuyển đổi chuỗi ký tự thành số nguyên

- Dữ liệu trả về từ **input dialog box** là một chuỗi ký tự. Nếu nhập vào một giá trị số **123**, nó trả về chuỗi **“123”**. Để nhận được dữ liệu là số, phải chuyển đổi.
  - Để chuyển đổi một chuỗi ký tự thành một giá trị **int**, có thể sử dụng phương thức tĩnh **parseInt** trong lớp **Integer** như sau:

```
int intValue = Integer.parseInt(intString);
```

trong đó **intString** là một chuỗi số nguyên như **“123”**.

### Chuyển đổi chuỗi ký tự thành số thực

- Để chuyển đổi một chuỗi ký tự thành một giá trị **double**, bạn có thể sử dụng phương thức tĩnh **parseDouble** trong lớp **Double** như sau:

```
double doubleValue = Double.parseDouble(doubleString);
```

trong đó **doubleString** là một chuỗi số thực như **“123.45”**.

## BÀI TẬP CHƯƠNG 2

Bài 2-1. Viết chương trình cho phép nhập vào giá trị  $^{\circ}\text{C}$  kiểu **double** từ bàn phím. Chuyển  $^{\circ}\text{C}$  vừa nhập sang  $^{\circ}\text{F}$ . Theo công thức sau:

$$\text{fahrenheit} = (9 / 5) * \text{celsius} + 32;$$

Lưu ý: Trong Java,  $9 / 5$  là 1, nhưng  $9.0 / 5$  là 1.8.

Bài 2-2. Viết chương trình cho phép nhập số **double** là **bán kính** của một đường tròn. Tính **chu vi** và **diện tích** của đường tròn và xuất kết quả ra màn hình.

Bài 2-3. Viết chương trình cho phép nhập vào một số **double** là **feet**, chuyển sang **mét** và xuất kết quả ra màn hình. Biết rằng  $1 \text{ foot} = 0.305 \text{ mét}$ .

Bài 2-4. Viết chương trình cho phép nhập vào một số **double** là **found**, chuyển sang **kg** và xuất kết quả ra màn hình. Biết rằng  $1 \text{ found} = 0.454 \text{ kg}$ .

Bài 2-5. Viết chương trình cho phép người dùng nhập vào số **long** là **phút** (ví dụ, 1 tỉ), hiển thị ra màn hình **số năm** và **số ngày** ứng với **số phút** nhập vào. Giả sử một năm có **365** ngày. Ví dụ: **1.000.000.000** phút xấp xỉ **1902 năm** và **214 ngày**.

Bài 2-6. Viết chương trình hiển thị thời gian hiện tại:

Gọi phương thức **System.currentTimeMillis()** để lấy tổng số ms tính từ 0h GMT ngày **01/01/1970**.

Tính giờ, phút, giây hiện tại ở Việt Nam và hiển thị dạng **h:m:s**

Bài 2-7. Viết chương trình cho phép nhập vào 3 điểm  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$ . Tính và xuất diện tích tam giác ứng với 3 đỉnh vừa nhập ra màn hình. Biết rằng:

$$\text{dienTich} = \sqrt{p(p - s_{12})(p - s_{13})(p - s_{23})}$$

$$p = \frac{(s_{12} + s_{13} + s_{23})}{2}$$

$$s_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

## CHƯƠNG 3 - CẤU TRÚC ĐIỀU KHIỂN

### 3.1. Các phép toán logic

Bảng 3-1 Các toán tử Boolean

Ký hiệu	Phép toán	Ví dụ (giả sử i = 1)
!	<b>NOT</b>	$!b$
&&	<b>AND</b>	$(a > 1) \&\& (a < 10)$
	<b>OR</b>	$a    b$
^	<b>EXCLUSIVE OR</b>	$a ^ b$

#### 3.1.1. Phép toán AND

Bảng 3-2 Phép toán AND

AND (&&)	true	false
true	true	false
false	false	false

#### 3.1.2. Phép toán OR

Bảng 3-3 Phép toán OR

OR (  )	true	false
true	true	true
false	true	false

#### 3.1.3. Phép toán NOT

Bảng 3-4 Phép toán NOT

NOT (!)	true	false
Result	false	true

#### 3.1.4. Phép toán XOR

Bảng 3-5 Phép toán XOR

XOR (^)	true	false
true	false	true
false	true	false

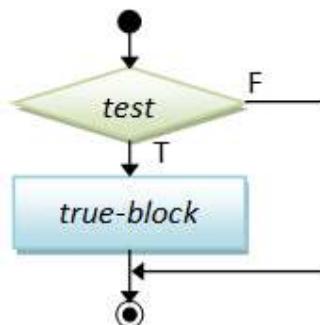
### 3.2. Cấu trúc lựa chọn

#### 3.2.1. Cấu trúc if

Cú pháp:

```
if (test) {  
    true-body;  
}  
next-statement;
```

Lưu đồ:



Hình 3.1 Lưu đồ cấu trúc if

Ví dụ:

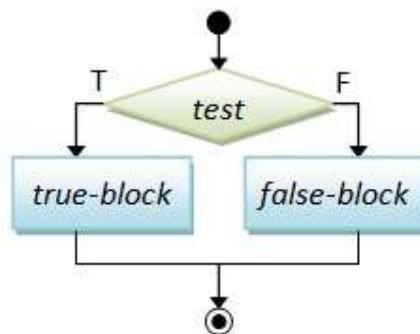
```
if (radius >= 0) {  
    area = radius * radius * PI;  
    System.out.println("The area for the circle of" +  
        " radius " + radius + " is " + area);  
}
```

#### 3.2.2. Cấu trúc if ... else

Cú pháp:

```
if (test) {  
    true-body;  
} else {  
    false-body;  
}  
next-statement;
```

Lưu đồ:



Hình 3.2 Lưu đồ cấu trúc *if... else*

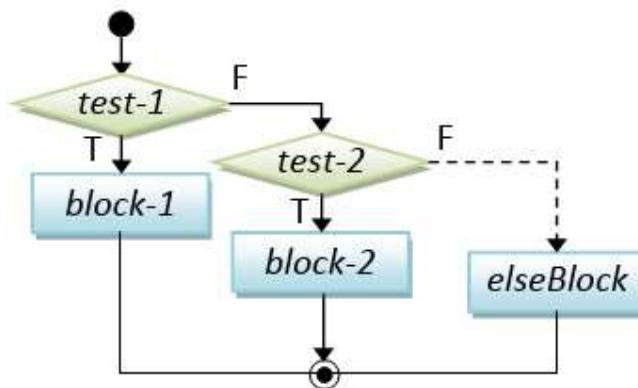
Ví dụ:

```
if (radius >= 0) {  
    area = radius * radius * PI;  
    System.out.println("The area for the circle of radius " +  
                       radius + " is " + area);  
} else {  
    System.out.println("Negative input");  
}
```

### 3.2.3. Nhiều cấu trúc if ... else lồng nhau

```
// nested-if  
if ( booleanExpr-1 ) {  
    block-1 ;  
} else if ( booleanExpr-2 ) {  
    block-2 ;  
} else if ( booleanExpr-3 ) {  
    block-3 ;  
} else if ( booleanExpr-4 ) {  
    .....  
} else {  
    elseBlock ;  
}
```

Lưu đồ:



Hình 3.3 Lưu đồ cấu trúc *if* lồng nhau

Ví dụ:

```

if (mark >= 80) {
    System.out.println("A");
} else if (mark >= 70) {
    System.out.println("B");
} else if (mark >= 60) {
    System.out.println("C");
} else if (mark >= 50) {
    System.out.println("D");
} else {
    System.out.println("F");
}
  
```

**Chú ý:**

- Mệnh đề *else* gắn với mệnh đề *if* gần nhất trong cùng một khối.

```

int i = 1, j = 2, k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
else
    System.out.println("B");
  
```

tương đương

nên dùng  
cách này

```

int i = 1, j = 2, k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
else
    System.out.println("B");
  
```

- Đoạn lệnh trên sẽ không in ra gì cả. Để bắt mệnh đề *else* gắn với mệnh đề *if* đầu tiên, phải thêm một cặp ngoặc {}:

```

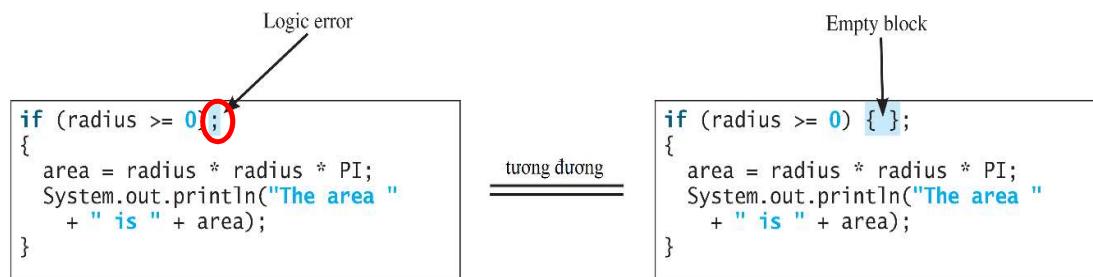
int i = 1, j = 2, k = 3;
if(i > j) {
    if(i > k)
        System.out.println("A");
} else
    System.out.println("B");

```

Đoạn lệnh trên sẽ in ra ký tự **B**.

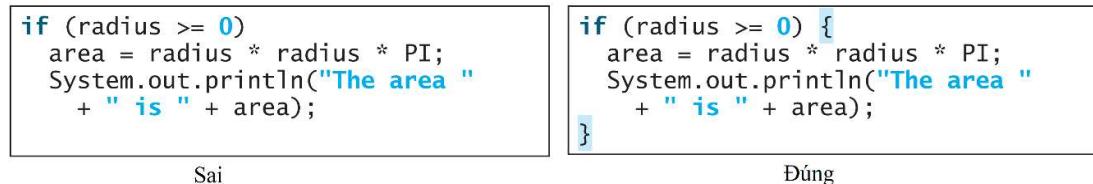
### Lỗi phổ biến:

- Thêm một dấu chấm phẩy ở cuối mệnh đề **if**.



*Lỗi này rất khó tìm, vì nó không phải là lỗi biên dịch hay lỗi chạy chương trình, nó là một lỗi logic.*

- Quên cắp dấu {} cho khối lệnh.



### 3.2.4. Cấu trúc switch

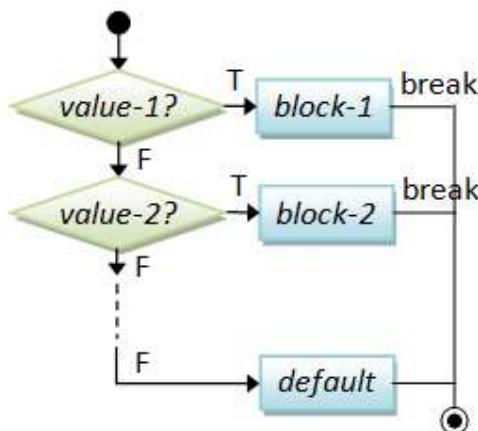
#### Cú pháp:

```

switch ( selector ) {
    case value-1:    block-1;    break;
    case value-2:    block-2;    break;
    case value-3:    block-3;    break;
    .....
    case value-n:   block-n;    break;
    default:         default-block;
}

```

Lưu đồ:



Hình 3.4 Lưu đồ cấu trúc **switch ... case**

Ví dụ:

```

int thang = m, nam = y; //Cho trước giá trị x, y nguyên
switch(thang) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: System.out.println("Thang có 31 ngày!"); break;
    case 4:
    case 6:
    case 9:
    case 11: System.out.println("Thang có 30 ngày!"); break;
    case 2:
        if((nam % 100 != 0 && nam % 4 == 0) || (nam % 400 == 0))
            System.out.println("Thang có 29 ngày!"); break;
        else
            System.out.println("Thang có 28 ngày!"); break;
    default:
        System.out.println("Giá trị tháng x không hợp lệ!");
}
  
```

### Quy tắc lệnh switch:

- Biểu thức **switch** phải sinh ra một giá trị kiểu **char**, **byte**, **short**, hoặc **int**,... và phải luôn được bao trong cặp dấu ngoặc tròn.
  - **gtri1,..., gtriN** phải có cùng kiểu dữ liệu với giá trị của biểu thức **switch**.
  - Từ khóa **break** là tùy chọn, nhưng nên được sử dụng cuối mỗi trường hợp để thoát khỏi phần còn lại của lệnh **switch**. Nếu không có lệnh **break**, lệnh **case** tiếp theo sẽ được thực hiện.
  - Trường hợp **default** là tùy chọn, có thể sử dụng để thực hiện các lệnh khi không có trường hợp nào ở trên là đúng.
    - Thứ tự của các trường hợp (gồm cả trường hợp **default**) là không quan trọng. Tuy nhiên, phong cách lập trình tốt là nên theo một trình tự logic của các trường hợp và đặt trường hợp **default** cuối cùng.

### Lưu ý:

- Đừng quên dùng lệnh **break** khi cần thiết.
- Ví dụ đoạn mã sau luôn hiển thị "**Sai so nam!**" dù giá trị **sonam** là bao nhiêu. Giả sử **sonam** bằng **15**. Lệnh **laisuatnam = 8.5** được thực hiện, tiếp theo là lệnh **laisuatnam = 9.0**, và cuối cùng là lệnh **System.out.println("Sai so nam!")**.

```
switch (sonam) {  
    case 5 : laisuatnam = 6.5;  
    case 10 : laisuatnam = 7.5;  
    case 15 : laisuatnam = 8.5;  
    case 30 : laisuatnam = 9.0;  
    default : System.out.println("Sai so nam!");  
}
```

### 3.2.5. Toán tử điều kiện

#### Cú pháp:

```
(bt_logic) ? bt1 : bt2;
```

Thực hiện **bt1** khi **bt\_logic** đúng, thực hiện **bt2** khi **bt\_logic** sai.

**Ví dụ 1:** `abs = (a > 0) ? a : -a;`

Ví dụ 2: Thay vì sử dụng **if**

```
if (x > 0)
    y = 1;
else
    y = -1;
```

Có thể dùng toán tử ?

```
y = (x > 0) ? 1 : -1;
```

Ví dụ 3:

```
if (so % 2 == 0)
    System.out.println(so + " là số chẵn!");
else
    System.out.println(so + " là số lẻ!");
```

Tương đương với:

```
System.out.println((so % 2 == 0) ?
    so + " là số chẵn!" : so + " là số lẻ!");
```

### 3.3. Cấu trúc lặp

#### 3.3.1. Đặt vấn đề

Giả sử rằng cần in một chuỗi (ví dụ, "**Welcome to Java!**") một trăm lần. Thật sự tẻ nhạt phải viết các phát biểu sau đây một trăm lần:

```
System.out.println ("Welcome to Java!");
```

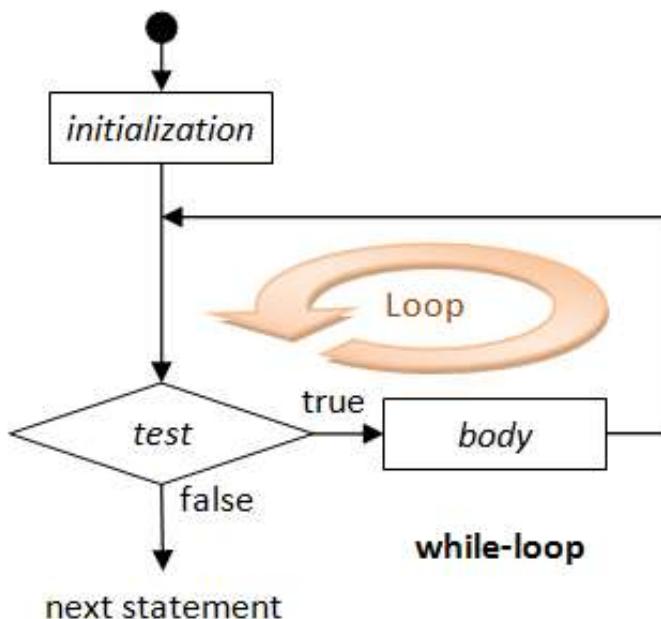
100 times {  
    System.out.println("Welcome to Java!");  
    System.out.println("Welcome to Java!");  
    ...  
    System.out.println("Welcome to Java!");

#### 3.3.2. Lệnh lặp **while**

Cú pháp:

```
initialization-statement;
while (test) {
    loop-body;
}
next-statement;
```

*Lưu đồ:*



Hình 3.5 Lưu đồ cấu trúc lặp **while**

*Ví dụ:*

```

int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java!");
    count++;
}
    
```

*Lưu ý:*

Giá trị dấu chấm động là gần đúng, nếu sử dụng chấm động để kiểm tra đăng thức trong một điều khiển lặp có thể dẫn đến bộ đếm thiếu chính xác và kết quả sai.

Ví dụ sau nên sử dụng giá trị **nguyên** cho biến **data**. Nếu **data** có **kiểu thực** thì **data != 0** có thể là **true** dù **data** bằng **0**.

```

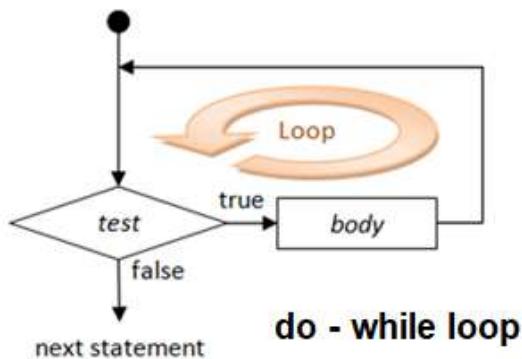
double data = Math.pow(Math.sqrt(2),2) - 2;
System.out.println((data == 0) ?
    "Gia tri data bang 0!" : "Gia tri data khac khong!");
    
```

### 3.3.3. Lệnh lặp do ... while

Cú pháp:

```
initialization;  
do {  
    loop-body;  
} while (test);  
next-statement;
```

Lưu đồ:



Hình 3.6 Lưu đồ cấu trúc lặp do ... while

Ví dụ:

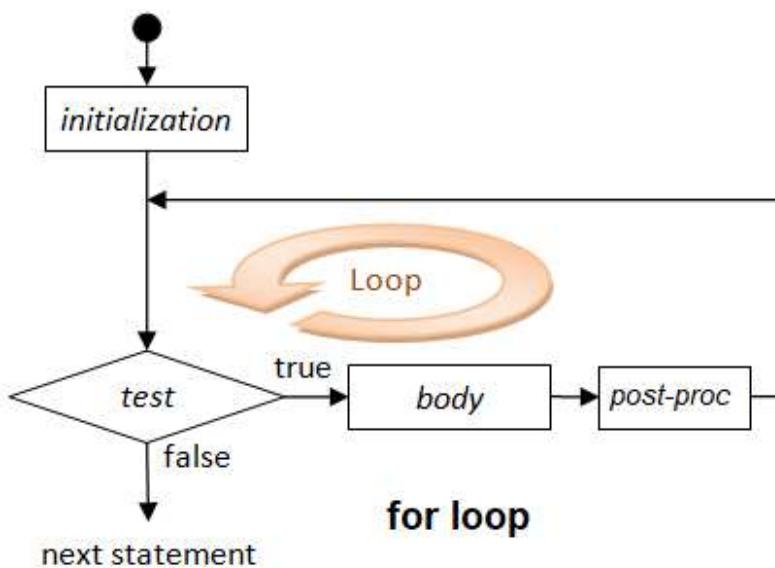
```
int count = 0;  
do {  
    System.out.println("Welcome to Java!");  
    count++;  
} while (count < 100);
```

### 3.3.4. Lệnh lặp for

Cú pháp:

```
for (initialization; test; post-processing) {  
    loop-body;  
}  
next-statement;
```

*Lưu đồ:*



Hình 3.7 Lưu đồ cấu trúc lặp *for*

*Ví dụ:*

```

int i;
for (i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
    
```

*Lưu ý:*

- Các trường hợp sau đây là đúng:

```

for (int i = 1; i < 1000; System.out.println(i++));
    
```

```

for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    //Do somthing
}
    
```

- Điều kiện tiếp tục lặp luôn là *true*:

```

for (; ); {
    //Do somthing
}
    
```

```

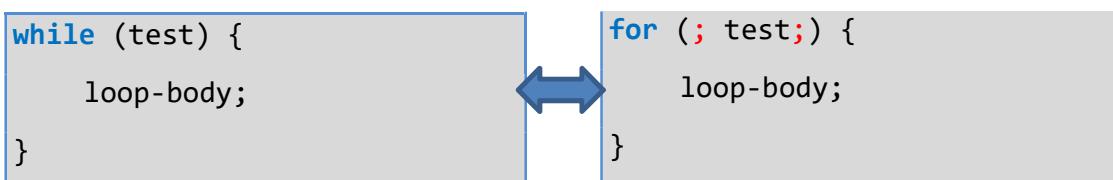
for (; true); {
    //Do somthing
}
    
```

```

while (true) {
    //Do somthing
}
    
```

### 3.3.4.1. Cách sử dụng lệnh lặp

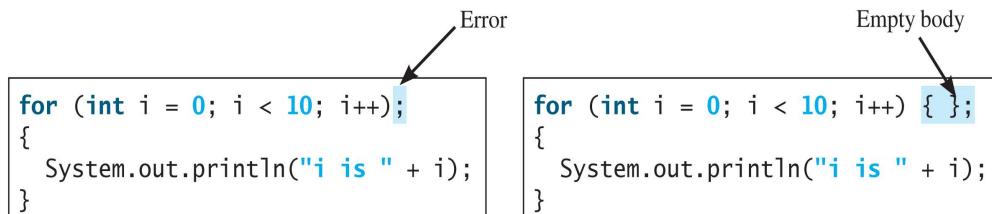
- Ba lệnh lặp **while**, **do...while**, và **for** là tương đương nhau trong nhiều trường hợp; nghĩa là có thể viết một vòng lặp bằng một dạng bất kỳ trong 3 dạng trên.
- Lệnh lặp **for** có thể sử dụng khi biết trước số lần lặp, ví dụ khi bạn muốn in ra một thông báo 100 lần.
- Lệnh lặp **while** có thể sử dụng khi không biết trước số lần lặp, ví dụ như trong trường hợp đọc vào các số đến khi gặp số 0.
- Lệnh lặp **do...while** có thể sử dụng thay lệnh **while** khi thân vòng lặp phải được thực hiện trước khi kiểm tra điều kiện tiếp tục lặp.
- Sự tương đương của các lệnh lặp:



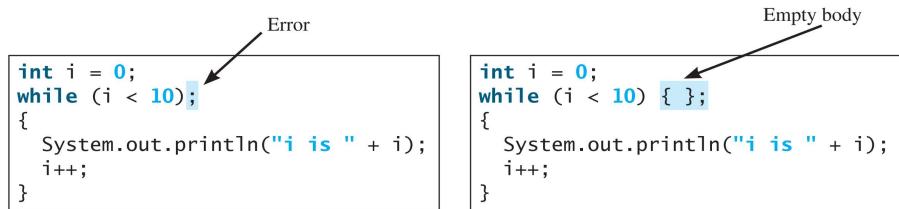
### 3.3.4.2. Một số lỗi thường gặp

- Dùng dấu chấm phẩy cuối lệnh lặp:

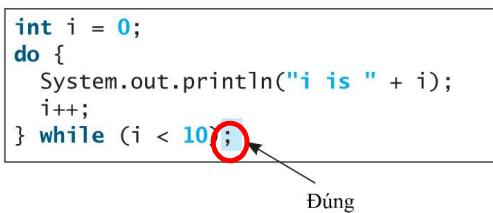
Câu lệnh **for**:



Câu lệnh **while**:



Riêng câu lệnh **do ... while**:



### 3.3.4.3. Từ khóa break

- Chúng ta đã sử dụng từ khóa **break** trong câu lệnh **switch**. Chúng ta cũng có thể sử dụng từ khóa **break** trong lệnh lặp để kết thúc ngay vòng lặp.

*Ví dụ:*

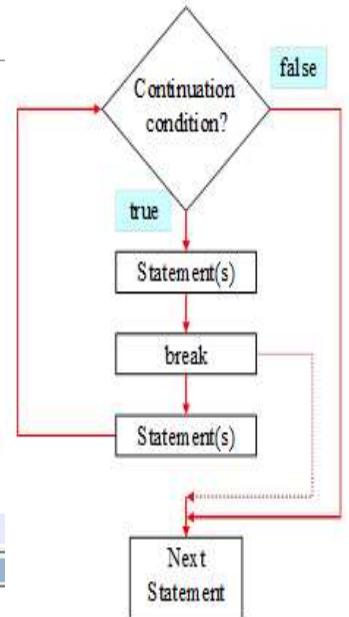
```

1 public class TestBreak {
2     public static void main(String[] args) {
3         int sum = 0;
4         int number = 0;
5
6         while(number < 20) {
7             number++;
8             sum += number;
9             if (sum >= 100)
10                break;
11
12         }
13         System.out.println("The number is "+ number)
14         System.out.println("The sum is "+ sum);
15     }
16 }
```

Console

```

Java TestBreak
Process started >>>
The number is 14
The sum is 105
```



### 3.3.4.4. Từ khóa continue

- Chúng ta cũng có thể dùng từ khóa **continue** trong vòng lặp. Khi gặp từ khóa **continue**, nó sẽ kết thúc vòng lặp hiện tại. Nói cách khác, **continue** thoát khỏi **một vòng lặp**, còn **break** thì kết thúc **một lệnh lặp**.

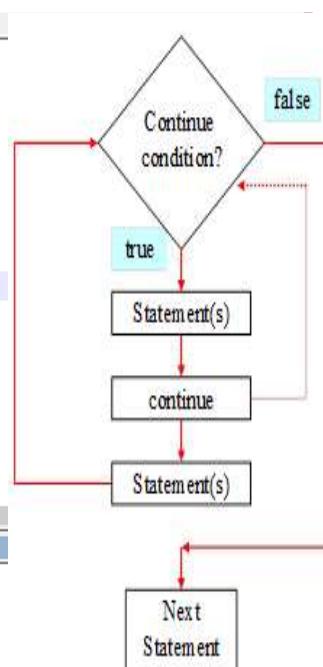
```

1 public class TestContinue {
2     public static void main(String[] args) {
3         int sum = 0;
4         int number = 0;
5         //Tinh tong cac so tu 1 -> 10 la 55
6         while(number < 10) {
7             number++;
8             if(number == 5 || number == 6)
9                 //Bo qua 5 va 6, nen tong con lai la 44
10                continue;
11             sum += number;
12         }
13
14         System.out.println("The sum is "+ sum);
15     }
16 }
```

Console

```

<<< Process finished. (Exit code 0)
java TestContinue
Process started >>>
The sum is 44
<<< Process finished. (Exit code 0)
===== READY =====
```



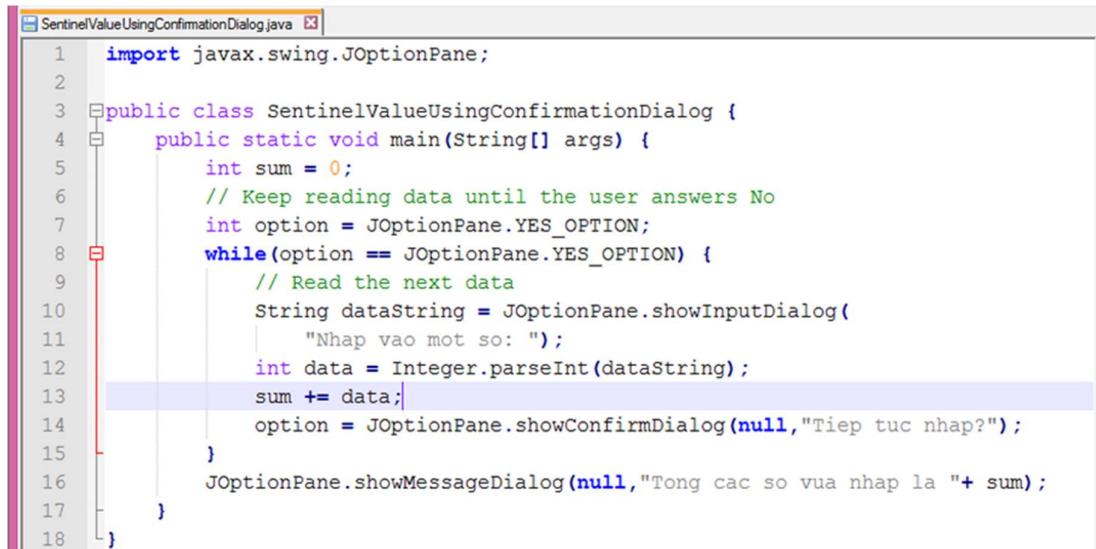
### 3.3.4.5. Kiểm soát vòng lặp với Confirmation Dialog

```

int option = JOptionPane.YES_OPTION;
while(option == JOptionPane.YES_OPTION) {
    System.out.println("continue loop");
    option = JOptionPane.showConfirmDialog(null, "Continue?");
}

```

Ví dụ: *SentinelValueUsingConfirmationDialog.java*

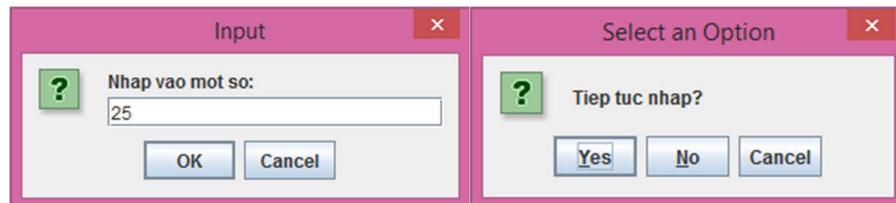


```

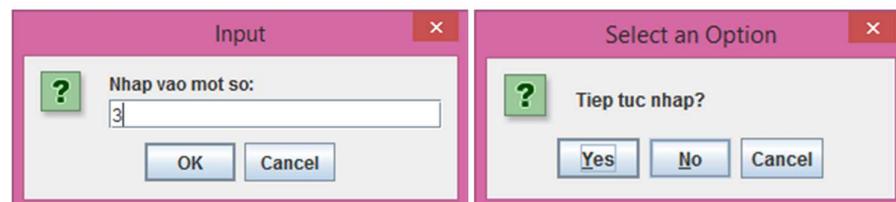
1 import javax.swing.JOptionPane;
2
3 public class SentinelValueUsingConfirmationDialog {
4     public static void main(String[] args) {
5         int sum = 0;
6         // Keep reading data until the user answers No
7         int option = JOptionPane.YES_OPTION;
8         while(option == JOptionPane.YES_OPTION) {
9             // Read the next data
10            String dataString = JOptionPane.showInputDialog(
11                "Nhập vào một số: ");
12            int data = Integer.parseInt(dataString);
13            sum += data;
14            option = JOptionPane.showConfirmDialog(null, "Tiếp tục nhập? ");
15        }
16        JOptionPane.showMessageDialog(null, "Tổng các số vừa nhập là " + sum);
17    }
18 }

```

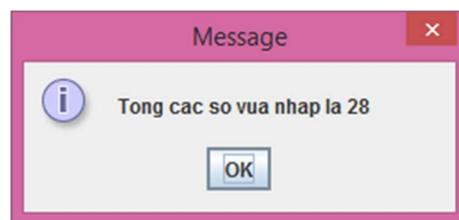
- Chạy chương trình, nhập vào số **25**, nhấn **OK**. Tiếp tục nhập? Chọn **Yes**.



- Nhập vào số **3**, nhấn **OK**. Tiếp tục nhập? Chọn **No**.



- Kết quả:



## BÀI TẬP CHƯƠNG 3

Bài 3-1. Viết chương trình nhập 2 số  $a, b$ . Tìm **ước số chung lớn nhất** của 2 số đó?

Bài 3-2. Nhập 3 cạnh  $a, b, c$ . Kiểm tra xem có lập thành tam giác không, nếu có thì là **tam giác gì** và tính **chu vi, diện tích** tam giác đó?

Bài 3-3. Tìm lượng tiền bán hàng:

Bạn vừa mới bắt đầu công việc bán hàng trong một cửa hàng. Thu nhập của bạn bao gồm một lương cơ bản **\$5,000/năm** và tiền hoa hồng được tính như sau:

Sales Amount	Commission Rate
\$0.01–\$5,000	8 %
\$5,000.01–\$10,000	10 %
>= \$10,000.01	10 %

Mục đích của bạn là kiểm được **\$30,000/năm**. Viết chương trình tìm lượng tiền bán hàng nhỏ nhất bạn phải tạo ra để giúp bạn đạt được mục đích.

Bài 3-4. Viết chương trình nhập vào tháng và năm (dương lịch). Xuất ra màn hình tháng vừa nhập có bao nhiêu ngày? Biết rằng:

*Tháng 1/3/5/7/8/10/12 có 31 ngày. Tháng 4/6/9/11 có 30 ngày. Tháng 2 có 28 ngày nếu năm nhuận, 29 ngày nếu không nhuận.*

*Điều kiện năm nhuận: ((năm%100 != 0 && năm %4 == 0) || năm%400 == 0)*

Bài 3-5. Sử dụng các lệnh lặp lồng nhau để in ra màn hình như các hình sau:

a./

1  
1 2 1  
1 2 4 2 1  
1 2 4 8 4 2 1  
1 2 4 8 16 8 4 2 1  
1 2 4 8 16 32 16 8 4 2 1  
1 2 4 8 16 32 64 32 16 8 4 2 1  
1 2 4 8 16 32 64 128 64 32 16 8 4 2 1

### Bài tập Chương 3 – Cấu trúc điều khiển

b./

Pattern A	Pattern B	Pattern C	Pattern D
1	1 2 3 4 5 6		1 2 3 4 5 6
1 2	1 2 3 4 5	2 1	1 2 3 4 5
1 2 3	1 2 3 4	3 2 1	1 2 3 4
1 2 3 4	1 2 3	4 3 2 1	1 2 3
1 2 3 4 5	1 2	5 4 3 2 1	1 2
1 2 3 4 5 6	1	6 5 4 3 2 1	1

Bài 3-6. Viết chương trình cho phép nhập vào một số nguyên (giả sử là 7). Sử dụng các lệnh lặp lồng nhau để in ra màn hình:

			1			
			2	1	2	
			3	2	1	2
			4	3	2	1
			5	4	3	2
			6	5	4	3
7	6	5	4	3	2	1

Bài 3-7. Viết chương trình cho phép nhập vào một số nguyên  $n$ . Tìm và hiển thị ra màn hình tất cả các **số nguyên tố** nhỏ hơn  $n$ .

Bài 3-8. Viết chương trình tính xấp xỉ các giá trị sau:

a./ Với  $\frac{(-1)^{i+1}}{2i-1} \geq 0.000001$

$$\pi = 4 \left( 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots + \frac{(-1)^{i+1}}{2i-1} \right)$$

b./

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots + \frac{1}{i!}$$

Với  $i = 10.000, 20.000, \dots, 100.000$ .

Bài 3-9. Viết chương trình cho phép người dùng nhập vào **năm** và **thứ** của ngày đầu của năm. Ví dụ, 2013 và 2 (là Thứ 3), nghĩa là **ngày 01 tháng 01 năm 2013** là **Thứ 3**. Hãy hiển thị ra màn hình Thứ của ngày đầu tiên của mỗi tháng.

January 1, 2013 is Tuesday

...

December 1, 2013 is Sunday

Bài 3-10. Viết chương trình nhập vào 2 số nguyên **chiSoCu**, **chiSoMoi** ứng với chỉ số điện cũ và chỉ số điện mới.

Tính **soKwTieuThu** = **chiSoMoi** - **chiSoCu**, và xuất ra màn hình tiền điện phải trả, biết rằng:

Số kw tiêu thụ	Đơn giá (đồng)
Dưới 50 kw	1.750
Từ 50 kw đến dưới 100 kw	2.225
Từ 100 kw đến dưới 200 kw	2.750
Từ 200 kw trở lên	3.250

Bài 3-11. Viết chương trình nhập vào 3 điểm A( $x_1, y_1$ ), B( $x_2, y_2$ ), C( $x_3, y_3$ ). Kiểm tra xem 3 điểm A, B, C có tạo thành tam giác hay không? Nếu có, thì là tam giác gì? Tính và xuất diện tích tam giác ứng với 3 đỉnh vừa nhập ra màn hình.

**Hướng dẫn:**

- Ba điểm tạo thành tam giác khi 3 điểm đó không thẳng hàng.
- Đường thẳng qua 2 điểm A,B:  $y = m*x + b$ , với  $m = \frac{y_B - y_A}{x_B - x_A}$ ;  $b = y_A - m*x_A$ .
- Điểm B không thuộc về AB  $\Rightarrow A, B, C$  không thẳng hàng.

$$dienTich = \sqrt{p(p - s_{12})(p - s_{13})(p - s_{23})}; p = \frac{(s_{12} + s_{13} + s_{23})}{2}$$

$$\text{với } s_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

# CHƯƠNG 4 - PHƯƠNG THỨC (METHODS)

## 4.1. Giới thiệu phương thức (method)

Giả sử, cần tính tổng các số từ 1 đến 10, từ 25 đến 35 và từ 37 đến 49. Chúng ta có thể viết code như sau:

```

int sum = 0;
for(int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Tong tu 1 den 10 la " + sum);

sum = 0;
for(int i = 25; i <= 35; i++)
    sum += i;
System.out.println("Tong tu 25 den 35 la " + sum);

sum = 0;
for(int i = 37; i <= 49; i++)
    sum += i;
System.out.println("Tong tu 37 den 49 la " + sum);

```

Chúng ta thấy rằng việc tính tổng từ 1-10, 25-35, và 37-49 là giống nhau. Sẽ tốt đẹp hơn, nếu chúng ta có thể viết code một lần và tái sử dụng nó?

Chúng ta có thể làm như vậy bằng cách định nghĩa một phương thức và gọi lại.

The screenshot shows a Java code editor and a terminal window. The code editor contains the following Java code:

```

1 public class methodSum{
2     public static int sum (int a, int b){
3         int result = 0;
4         for(int i = a; i <= b; i++)
5             result += i;
6
7         return result;
8     }
9
10    public static void main(String[] args) {
11        System.out.println("Tong tu 1 den 10 la " + sum(1,10));
12        System.out.println("Tong tu 25 den 35 la " + sum(25,35));
13        System.out.println("Tong tu 37 den 49 la " + sum(37,49));
14    }
15 }

```

The terminal window below shows the execution results:

```

Process started >>>
Tong tu 1 den 10 la 55
Tong tu 25 den 35 la 330
Tong tu 37 den 49 la 559
<<< Process finished. (Exit code 0)
=====READY =====

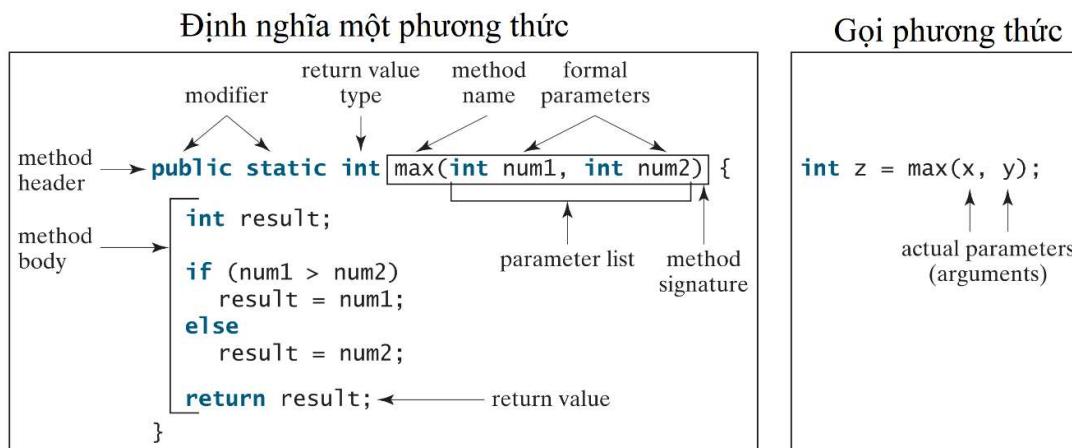
```

## 4.2. Định nghĩa phương thức

Một phương thức là một tập các câu lệnh được nhóm lại với nhau nhằm thực hiện một công việc nào đó.

*Cú pháp:*

```
modifier returnType methodName(list of parameters) {
    // Method body;
}
```



Hình 4.1 Một phương thức bao gồm **method header** và **method body**

- **parameter profile** gồm kiểu, thứ tự, và số tham số của một phương thức.
- **method signature (header)** gồm tên phương thức và **parameter profiles**.
- Các tham số khai báo trong **method header** được gọi là tham số hình thức (**formal parameters**).
  - Khi phương thức được gọi, các tham số hình thức được thay thế bởi các biến hoặc dữ liệu, được gọi là các tham số thực sự (**actual parameters**).
  - Một phương thức có thể trả về một giá trị. Kiểu của giá trị đó là kiểu dữ liệu của phương thức trả về. Nếu phương thức không trả về một giá trị, kiểu của phương thức trả về dùng từ khóa **void**.

*Ví dụ*, kiểu giá trị trả về trong phương thức **main** là **void**.

## 4.3. Cách gọi phương thức

- Khi định nghĩa phương thức, chúng ta xác định phương thức này để làm gì? Để thực hiện phương thức, chúng ta phải gọi nó.

- Có hai cách để gọi một phương thức, tùy thuộc vào việc phương thức có trả về một giá trị hay không.
  - Nếu một phương thức trả về một giá trị, gọi đến phương thức này thường được coi là một giá trị.

*Ví dụ:*

```
int max = max(3, 4);  
System.out.println(max(3, 4));
```

- Nếu một phương thức có kiểu trả về là **void**, việc gọi phương thức phải là một **statement**.

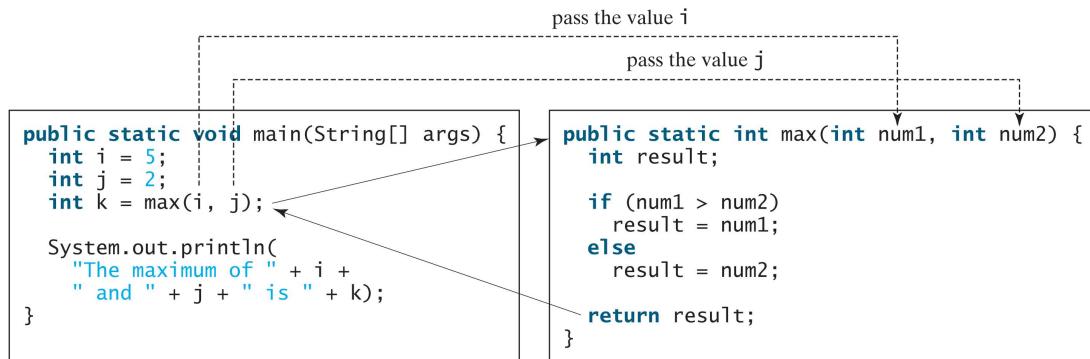
*Ví dụ*, phương thức **println** trả về **void**.

```
System.out.println("Welcome to Java!");
```

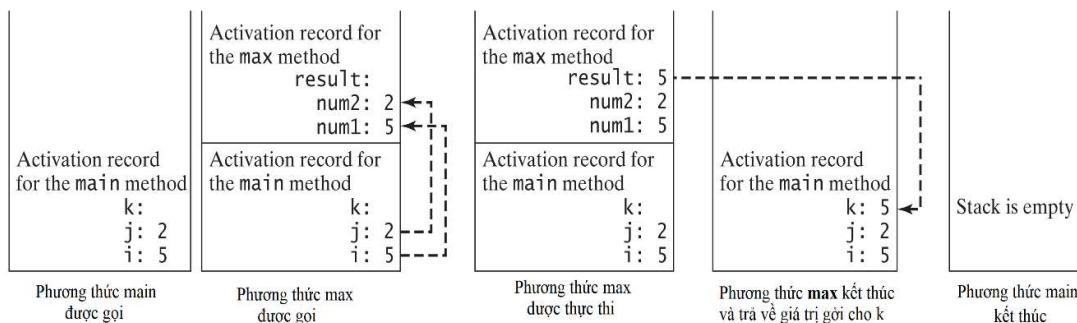
*Ví dụ: TestMax.java*

```
public class TestMax {  
    /* Main method */  
    public static void main(String[] args) {  
        int i = 5;  
        int j = 2;  
        int k = max(i, j);  
        System.out.println("The maximum of " + i +  
                           " and " + j + " is " + k);  
    }  
    /* Return the max of two numbers */  
    public static int max(int num1, int num2) {  
        int result;  
        if (num1 > num2)  
            result = num1;  
        else  
            result = num2;  
        return result;  
    }  
}
```

- Khi phương thức **max** được gọi, chuyển điều khiển cho nó. Sau khi phương thức max thực hiện xong, nó sẽ trả về điều khiển lại cho nơi gọi.



- Các biến được định nghĩa trong **main method** của chương trình trên là ***i, j, và k***. Các biến được định nghĩa trong phương thức **max** là ***num1, num2, và result***. Các biến ***num1*** và ***num2*** được định nghĩa **method signature** và các tham số của phương thức **max**. Giá trị của chúng được chuyển qua phương thức gọi.



#### 4.4. void method

- Một **void method** không **return** về một giá trị.

*Ví dụ: TestVoidMethod.java*

```

public class TestVoidMethod {
    public static void main(String[] args) {
        System.out.print("The grade is ");
        printGrade(78.5);
        System.out.print("The grade is ");
        printGrade(59.5);
    }
}
    
```

```
public static void printGrade(double score) {  
    if (score >= 90.0) {  
        System.out.println('A');  
    }  
    else if (score >= 80.0) {  
        System.out.println('B');  
    }  
    else if (score >= 70.0) {  
        System.out.println('C');  
    }  
    else if (score >= 60.0) {  
        System.out.println('D');  
    }  
    else {  
        System.out.println('F');  
    }  
}
```

Lưu ý:

- Câu lệnh trả về giá trị bắt buộc phải có đối với phương thức **non-void**.
- Phương thức sau đúng về logic, nhưng có lỗi biên dịch vì trình biên dịch Java nghĩ rằng phương thức này không trả về bất kỳ giá trị nào.

```
public static int xMethod (int n) {  
    if (n > 0)      return 1;  
    else if (n == 0) return 0;  
    else if (n < 0)  return -1;  
}
```

- Để sửa lỗi này, xóa **if(n<0)** trong đoạn mã trên.

```
public static int xMethod (int n) {  
    if (n > 0)      return 1;  
    else if (n == 0) return 0;  
    return -1;  
}
```

#### 4.5. Truyền tham trị

- Các đối số được truyền bằng giá trị cho các tham số khi gọi phương thức.

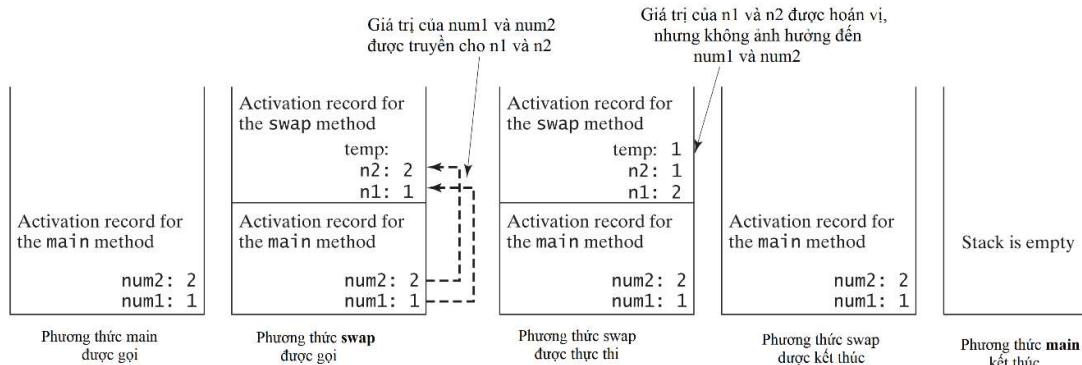
```
public static void nPrintln(String message, int n) {  
    for (int i = 0; i < n; i++)  
        System.out.println(message);  
}
```

Ví dụ: TestPassByValue.java

```
public class TestPassByValue {  
    /* Main method */  
    public static void main(String[] args) {  
        // Declare and initialize variables  
        int num1 = 1;  
        int num2 = 2;  
        System.out.println("Before invoking the swap method,  
                           num1 is " + num1 + " and num2 is " + num2);  
        // Invoke the swap method to attempt to swap two variables  
        swap(num1, num2);  
        System.out.println("Before invoking the swap method,  
                           num1 is " + num1 + " and num2 is " + num2);  
    }  
    /* Swap two variables */  
    public static void swap(int n1, int n2) {  
        System.out.println("\tInside the swap method");  
        System.out.println("\t\tBefore swapping, n1 is " + n1 +  
                           " and n2 is " + n2);  
        // Swap n1 with n2  
        int temp = n1;  
        n1 = n2;  
        n2 = temp;  
        System.out.println("\t\tAfter swapping, n1 is " + n1 +  
                           " and n2 is " + n2);  
    }  
}
```

Kết quả chạy chương trình:

```
Before invoking the swap method, num1 is 1 and num2 is 2
Inside the swap method
Before swapping, n1 is 1 and n2 is 2
After swapping, n1 is 2 and n2 is 1
After invoking the swap method, num1 is 1 and num2 is 2
```



#### 4.6. Nạp chồng phương thức (Overloading Methods)

- **Nạp chồng phương thức (Overloading Methods)** cho phép định nghĩa các phương thức với tên giống nhau.
- Phương thức **max** đã được sử dụng trước đó chỉ làm việc với kiểu **int**. Nhưng nếu chúng ta cần phải tìm **max** của hai số **double**? Giải pháp là tạo ra một phương thức khác có cùng tên nhưng với các tham số có kiểu **double**.

**Ví dụ:** TestMethodOverloading.java

```
public class TestMethodOverloading {
    /* Main method */
    public static void main(String[] args) {
        // Invoke the max method with int parameters
        System.out.println("The maximum of 3 and 4 is "
            + max(3, 4));
        // Invoke the max method with the double parameters
        System.out.println("The maximum of 3.0 and 5.4 is "
            + max(3.0, 5.4));
        // Invoke the max method with three double parameters
        System.out.println("The maximum of 3.0, 5.4,
            and 10.14 is " + max(3.0, 5.4, 10.14));
    }
}
```

```

/* Return the max of two int values */
public static int max(int num1, int num2) {
    if (num1 > num2)
        return num1;
    return num2;
}

/* Find the max of two double values */
public static double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    return num2;
}

/* Return the max of three double values */
public static double max(double num1, double num2, double num3) {
    return max(max(num1, num2), num3);
}
}

```

Kết quả chạy chương trình:

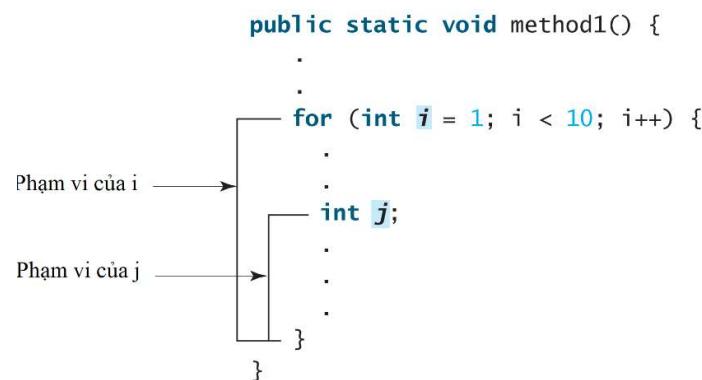
```

The maximum of 3 and 4 is 4
The maximum of 3.0 and 5.4 is 5.4
The maximum of 3.0, 5.4, and 10.14 is 10.14

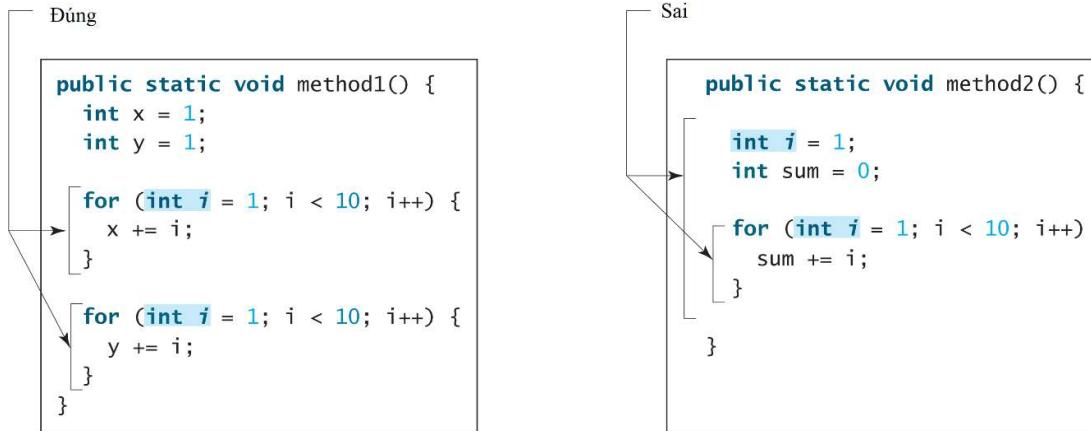
```

#### 4.7. Phạm vi của biến cục bộ

- Biến cục bộ: biến được khai báo trong một phương thức.
- Phạm vi: phần chương trình mà biến có thể được tham chiếu.
- Phạm vi của một biến cục bộ bắt đầu từ khi khai báo đến cuối **block** chứa biến đó. Một biến cục bộ phải được khai báo trước khi sử dụng.



- Chúng ta có thể khai báo một biến cục bộ trùng tên nhiều lần trong các khối riêng rẽ không lồng nhau trong một phương thức, nhưng không thể khai báo một biến cục bộ 2 lần trong các khối lồng nhau.



## BÀI TẬP CHƯƠNG 4

Bài 4-1. Viết chương trình nhập một số nguyên n. Dùng đệ quy, tính n!. Biết:

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n-1);$$

Bài 4-2. Viết chương trình cho phép người dùng nhập vào **năm** và **thứ** của ngày đầu của năm. Ví dụ, **2013** và **2** (là Thứ 3), nghĩa là ngày **01** tháng **01** năm **2013** là **Thứ 3**. Hãy hiển thị lịch của mỗi tháng trong năm như sau:

January 2013							December 2013						
Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5		1	2	3	4	5	6
6	7	8	9	10	11	12	8	9	10	11	12	13	14
13	14	15	16	17	18	19	15	16	17	18	19	20	21
20	21	22	23	24	25	26	22	23	24	25	26	27	28
27	28	29	30	31			29	30	31				

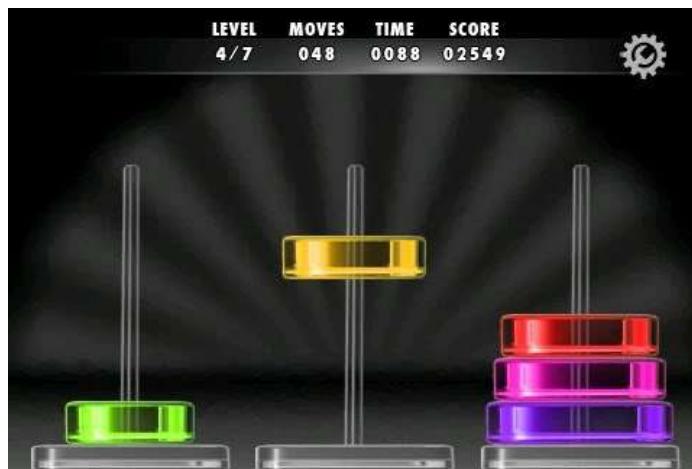
Bài 4-3 .Viết chương trình nhập một số nguyên n. Tính dãy **Fibonacci(n)**. Biết:

$$\text{fib}(0) = 0;$$

$$\text{fib}(1) = 1;$$

$$\text{fib}(n) = \text{fib}(n-2) + \text{fib}(n-1); n \geq 2;$$

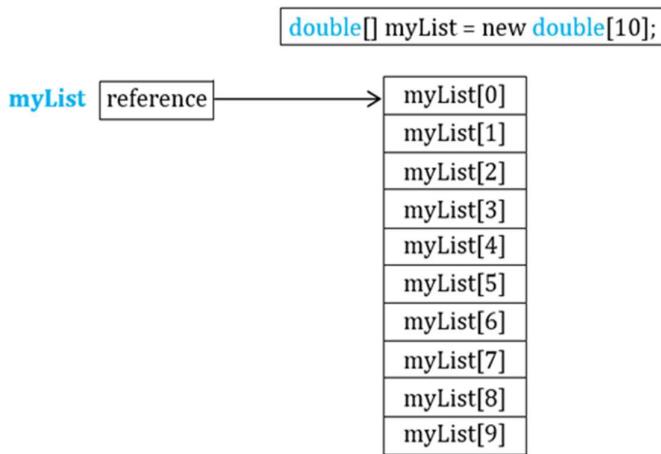
Bài 4-4. Viết chương trình giải bài toán **Tháp Hà Nội**.



# CHƯƠNG 5 - MẢNG (ARRAYS)

## 5.1. Giới thiệu

Mảng là một cấu trúc dữ liệu biểu diễn một tập các dữ liệu cùng kiểu.



Hình 5.1 Một mảng 10 phần tử kiểu **double**

## 5.2. Mảng một chiều

### 5.2.1. Khai báo biến mảng

- Khi một mảng được tạo ra, kích thước của nó là cố định. Để truy cập các phần tử trong một mảng chúng ta sử dụng một chỉ mục.
- Để sử dụng một mảng trong một chương trình, chúng ta phải khai báo một biến tham chiếu đến mảng và chỉ định kiểu dữ liệu cho các phần tử của mảng.

*Cú pháp khai báo một biến mảng:*

`elementType[] arrayRefVar;`

hoặc:

`elementType arrayRefVar[];`

**elementType** có thể là một kiểu dữ liệu bất kỳ, tất cả các phần tử trong mảng có cùng kiểu dữ liệu.

*Ví dụ*, khai báo một biến **myList** tham chiếu đến một mảng các phần tử **double**:

`double[] myList;`  
`double myList[];`

### 5.2.2. Tạo mảng

Cú pháp:

```
arrayRefVar = new elementType[arraySize];
```

1. Tạo một mảng bằng `new elementType[arraySize]`.
2. Gán mảng vừa tạo vào biến tham chiếu `arrayRefVar`.

Ví dụ:

```
myList = new double[10];
```

- `myList[0]` tham chiếu phần tử đầu tiên của mảng.
- `myList[9]` tham chiếu phần tử cuối cùng của mảng.

### 5.2.3. Khai báo và tạo mảng trong một bước

Cú pháp:

```
elementType[] arrayRefVar = new elementType[arraySize];
```

hoặc:

```
elementType arrayRefVar[] = new elementType[arraySize];
```

Ví dụ:

```
double[] myList = new double[10];  
double myList[] = new double[10];
```

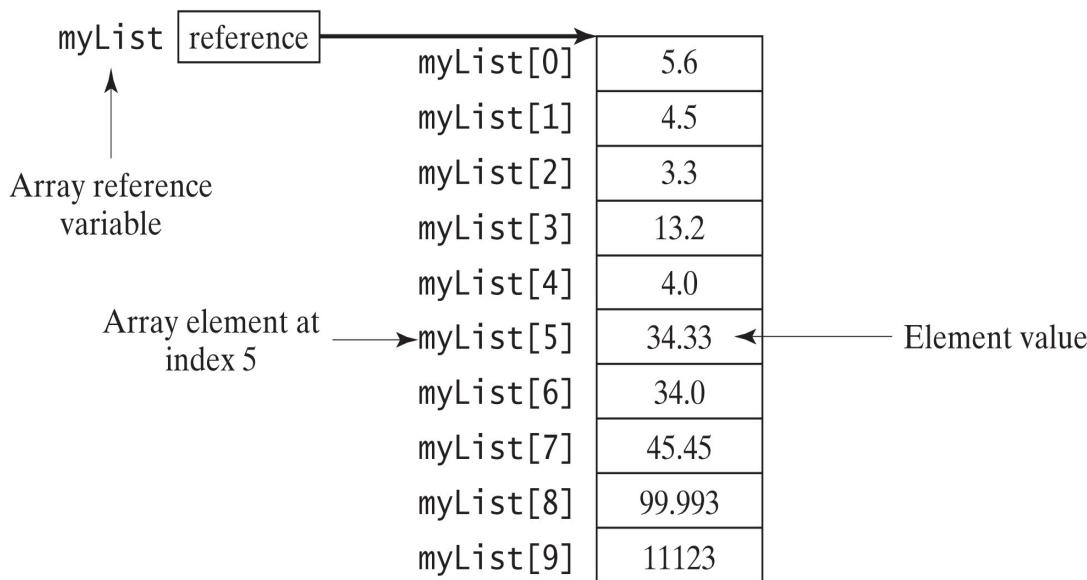
### 5.2.4. Gán giá trị cho các phần tử của mảng

Cú pháp:

```
arrayRefVar[index] = value;
```

Ví dụ:

```
myList[0] = 5.6;  
myList[1] = 4.5;  
myList[2] = 3.3;  
.....  
myList[8] = 99.993;  
myList[9] = 11123;
```

Hình 5.2 Mảng *myList* có 10 phần tử *double* với chỉ số từ 0 đến 9

### 5.2.5. Độ dài mảng và giá trị mặc định

- Khi một mảng được cấp phát, số lượng phần tử mảng là cố định và không thể thay đổi. Để lấy độ dài mảng chúng ta dùng *arrayRefVar.length*.

*Ví dụ*, *myList.length* là 10.

- Khi một mảng được tạo, giá trị các phần tử mảng được gán một giá trị mặc định: 0 nếu *kiểu số*, \u0000 nếu *char*, và false nếu *boolean*.

### 5.2.6. Truy cập phần tử mảng

- Các phần tử của mảng được truy cập thông qua chỉ số (*index*). Bắt đầu là 0, nghĩa là từ phần tử thứ 0 đến *arrayRefVar.length-1*.

*Ví dụ*, *myList* có 10 phần tử kiểu *double* có chỉ số từ 0 đến 9.

*Cú pháp:*

```
arrayRefVar[index];
```

### 5.2.7. Khởi tạo mảng

- Java cho phép kết hợp việc khai báo và khởi tạo mảng.

*Cú pháp:*

```
elementType[] arrayRefVar = {value0, value1, ..., valuek};
```

Ví dụ:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

- Khai báo, tạo, và khởi tạo một mảng *myList* gồm 4 phần tử:

```
double[] myList = new double[4];  
myList[0] = 1.9;  
myList[1] = 2.9;  
myList[2] = 3.4;  
myList[3] = 3.5;
```

- Khi tạo giá trị các phần tử mảng, chúng ta có thể sử dụng một vòng lặp.

```
double[] myList = new double[10];
```

1. Khởi tạo mảng với giá trị nhập từ bàn phím:

```
java.util.Scanner input = new java.util.Scanner(System.in);  
System.out.print("Enter " + myList.length + " values: ");  
for (int i = 0; i < myList.length; i++) {  
    myList[i] = input.nextDouble();  
}
```

2. Khởi tạo mảng với giá trị phát sinh ngẫu nhiên:

```
for (int i = 0; i < myList.length; i++) {  
    myList[i] = Math.random() * 100;  
}
```

Hàm *random()*; phát sinh ngẫu nhiên giá trị từ 0.0 → 1.0.

### 5.2.8. Một số thao tác trên mảng

#### 5.2.8.1. Xuất mảng ra màn hình

Để xuất một mảng ra màn hình, chúng ta phải xuất từng phần tử.

```
for (int i = 0; i < myList.length; i++) {  
    System.out.print(myList[i] + " ");  
}
```

### 5.2.8.2. Tính tổng các phần tử của mảng

```
double total = 0;
for (int i = 0; i < myList.length; i++) {
    total += myList[i];
}
```

### 5.2.8.3. Tìm phần tử lớn nhất trong mảng

```
double max = myList[0];
for (int i = 0; i < myList.length; i++) {
    if (myList[i] > max)
        max = myList[i];
}
```

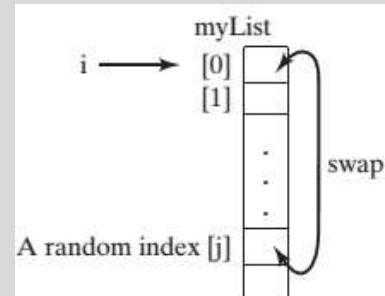
### 5.2.8.4. Tìm chỉ số nhỏ nhất của phần tử lớn nhất trong mảng

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 0; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
        indexOfMax = i;
    }
}
```

### 5.2.8.5. Trộn ngẫu nhiên các phần tử mảng

- Trong nhiều chương trình, có thể cần xáo trộn ngẫu nhiên các phần tử mảng.

```
for (int i = myList.length - 1; i > 0; i--) {
    // Generate a random index j randomly with 0 <= j <= i
    int j = (int)(Math.random() * (i + 1));
    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}
```



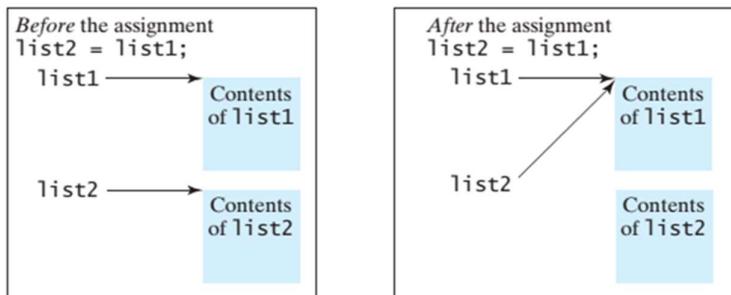
### 5.2.8.6. Dịch các phần tử mảng sang phải hoặc sang trái

- Đôi khi chúng ta cần dịch các phần tử sang trái hoặc sang phải.

```
double temp = myList[0];// Retain the first element
// Shift elements left
for(int i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}
// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```

### 5.2.8.7. Sao chép mảng

- Để sao chép nội dung của một mảng sang mảng khác, chúng ta phải sao chép từng phần tử của mảng vào mảng khác.
  - Trong Java, chúng ta không thể dùng câu lệnh `list2 = list1`; để sao chép mảng. Vì sau đó vùng nhớ `list2` không còn được tham chiếu tới, và sẽ thành rác.



Hình 5.3 `list1 = list2`

- Vì vậy, nếu muốn sao chép mảng, chúng ta phải làm như sau:

```
int[] sourceArray = {2,3,1,5,10};
int[] targetArray = new int[sourceArray.length];
for(int i = 0; i < sourceArray.length; i++) {
    targetArray[i] = sourceArray[i];
}
```

- Một phương pháp khác là sử dụng các phương pháp `arraycopy` trong lớp `java.lang.System` để sao chép mảng thay vì sử dụng một vòng lặp.

Cú pháp cho `arraycopy` là:

```
arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
```

Các thông số `src_pos` và `tar_pos` chỉ ra vị trí bắt đầu trong `sourceArray` và `targetArray`, tương ứng. Số lượng các phần tử copy từ `sourceArray` đến `targetArray` được chỉ định bởi `length`.

Ví dụ:

```
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

### 5.2.9. Vòng lặp `for-each`

Java hỗ trợ vòng lặp `for - each`, cho phép chúng ta duyệt liên tục qua các phần mảng mà không cần sử dụng một biến chỉ số.

Cú pháp:

```
for (elementType element: arrayRefVar) {  
    // Process the element  
}
```

Ví dụ, đoạn mã sau in tất cả các phần tử trong mảng `myList`:

```
for (double e: myList) {  
    System.out.println(e);  
}
```

### 5.2.10. Truyền mảng cho phương thức

– Java sử dụng **truyền tham trị** để truyền các tham số cho phương thức. Có nhiều sự khác nhau quan trọng khi truyền tham trị của biến có kiểu **dữ liệu cơ sở** và **biến mảng**.

– Với tham số **kiểu dữ liệu cơ sở**, giá trị thực được truyền. Thay đổi giá trị của tham số cục bộ trong phương thức không làm thay đổi giá trị của biến bên ngoài phương thức.

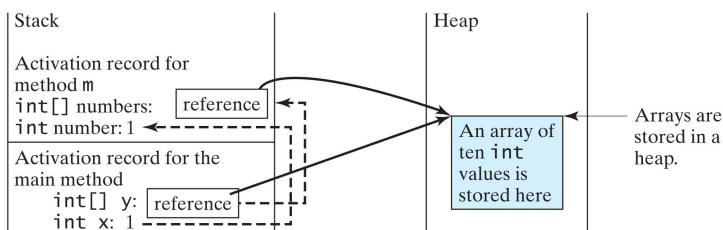
– Với tham số **kiểu mảng**, giá trị của tham số chứa một tham chiếu tới mảng; tham chiếu này được truyền cho phương thức. Bất kỳ sự thay đổi nào xuất hiện trong thân phương thức sẽ làm thay đổi mảng gốc được truyền.

**Ví dụ 1:**

```
public class Test {
    public static void main(String[] args) {
        int x = 1; //x represents an int value
        int[] y = new int[10]; //y represents an array of int values
        m(x, y); // Invoke m with arguments x and y
        System.out.println("x is " + x);
        System.out.println("y[0] is " + y[0]);
    }
    public static void m(int number, int[] numbers) {
        number = 1001; // Assign a new value to number
        numbers[0] = 5555; // Assign a new value to numbers[0]
    }
}
```

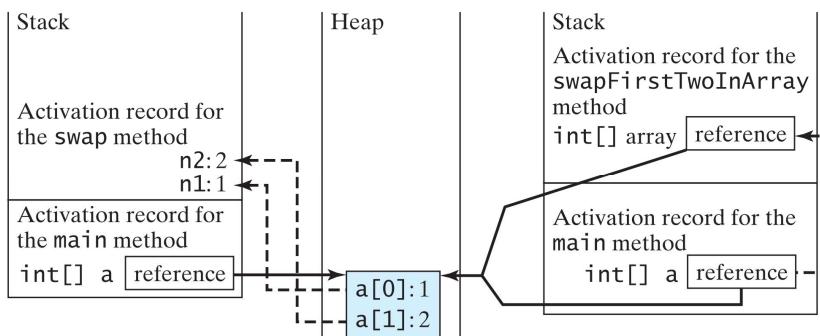
Kết quả chạy chương trình:

```
x is 1
y[0] is 5555
```



Hình 5.4 Giải trị x kiểu **int** truyền cho **number** và giá trị y kiểu tham chiếu truyền cho **numbers**

**Ví dụ 2:**



Hình 5.5 Khi truyền mảng cho phương thức , thì tham chiếu của mảng truyền cho phương thức

```
public class TestPassArray {  
    public static void main(String[] args) {  
        int[] a = {1, 2};  
        // Swap elements using the swap method  
        System.out.println("Before invoking swap");  
        System.out.println("array is {" + a[0] + ", " + a[1] + "});  
        swap(a[0], a[1]);  
        System.out.println("After invoking swap");  
        System.out.println("array is {" + a[0] + ", " + a[1] + "});  
        // Swap elements using the swapFirstTwoInArray method  
        System.out.println("Before invoking swapFirstTwoInArray");  
        System.out.println("array is {" + a[0] + ", " + a[1] + "});  
        swapFirstTwoInArray(a);  
        System.out.println("After invoking swapFirstTwoInArray");  
        System.out.println("array is {" + a[0] + ", " + a[1] + "});  
    }  
    /* Swap two variables */  
    public static void swap(int n1, int n2) {  
        int temp = n1;  
        n1 = n2;  
        n2 = temp;  
    }  
    /* Swap the first two elements in the array */  
    public static void swapFirstTwoInArray(int[] array) {  
        int temp = array[0];  
        array[0] = array[1];  
        array[1] = temp;  
    }  
}
```

Kết quả chạy chương trình:

```
Before invoking swap  
array is {1, 2}  
After invoking swap  
array is {1, 2}  
Before invoking swapFirstTwoInArray  
array is {1, 2}  
After invoking swapFirstTwoInArray  
array is {2, 1}
```

### 5.2.11. Trả về một mảng từ một phương thức

Kết quả trả về của một phương thức có thể là một mảng.

```
public static int[] reverse(int[] list) {  
    int[] result = new int[list.length];  
    for (int i=0, j=result.length-1; i<list.length; i++,j--) {  
        result[j] = list[i];  
    }  
    return result;  
}
```

Gọi phương thức `reverse(list1);` gán kết quả vào `list2`.

```
int[] list1 = {1, 2, 3, 4, 5, 6};  
int[] list2 = reverse(list1);
```

### 5.2.12. Arrays Class

Lớp `java.util.Arrays` cung cấp các phương thức tinh cho việc sắp xếp mảng và tìm kiếm trong mảng, so sánh mảng,....

*Ví dụ:*

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers); // Sort the whole array  
java.util.Arrays.parallelSort(numbers); // Sort the whole array  
.....
```

### 5.3. Mảng hai chiều

#### 5.3.1. Giới thiệu

Dữ liệu trong một bảng hoặc một ma trận có thể được biểu diễn bằng một mảng hai chiều.

Bảng 5-1 Bảng Distance

	<b>Chicago</b>	<b>Boston</b>	<b>New York</b>	<b>Atlanta</b>	<b>Miami</b>	<b>Dallas</b>	<b>Houston</b>
<b>Chicago</b>	0	983	787	714	1375	967	1087
<b>Boston</b>	983	0	214	1102	1763	1723	1842
<b>New York</b>	787	214	0	888	1549	1548	1627
<b>Atlanta</b>	714	1102	888	0	661	781	810
<b>Miami</b>	1375	1763	1549	661	0	1426	1187
<b>Dallas</b>	967	1723	1548	781	1426	0	239
<b>Houston</b>	1087	1842	1627	810	1187	239	0

```
double[][] distances = {
    {0, 983, 787, 714, 1375, 967, 1087},
    {983, 0, 214, 1102, 1763, 1723, 1842},
    {787, 214, 0, 888, 1549, 1548, 1627},
    {714, 1102, 888, 0, 661, 781, 810},
    {1375, 1763, 1549, 661, 0, 1426, 1187},
    {967, 1723, 1548, 781, 1426, 0, 239},
    {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

Hình 5.6 Mảng hai chiều lưu bảng Distance

Một phần tử trong một mảng hai chiều được truy cập thông qua chỉ số dòng và chỉ số cột.

#### 5.3.2. Khai báo và tạo biến mảng nhiều chiều

Cú pháp để khai báo một mảng hai chiều:

```
elementType[][][] arrayRefVar;
```

hoặc:

```
elementType arrayRefVar[][]; // Được, nhưng không khuyến khích
```

Ví dụ: khai báo một biến mảng ma trận hai chiều của giá trị kiểu **int**:

```
int[][] matrix;
```

hoặc:

```
int matrix[][]; // Không khuyến khích
```

### Cú pháp để tạo mảng hai chiều:

```
matrix = new int[5][5];
```

Chỉ số của dòng và cột của mảng 2 chiều kiểu **int** và bắt đầu từ **0**.

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix = new int[5][5];
```

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	7	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix[2][1] = 7;
```

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

```
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

Hai câu lệnh sau là tương đương:

```
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

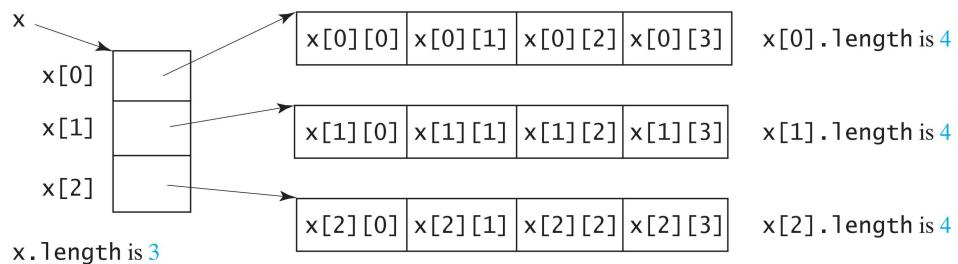
tương đương

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

### 5.3.3. Độ dài của mảng nhiều chiều

Mảng hai chiều là một mảng mà mỗi phần tử là một mảng một chiều.

```
x.length;
x[0].length;
x[1].length;
x[2].length;
```

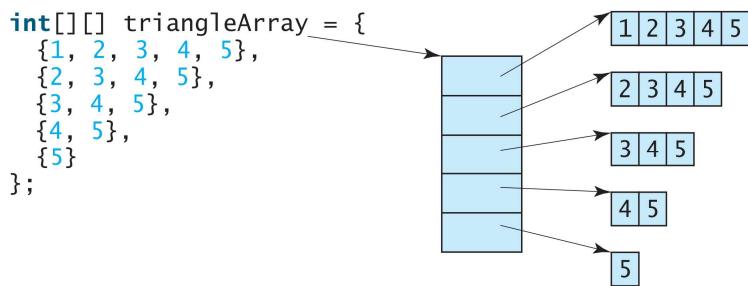


Hình 5.7 Độ dài của mảng 2 chiều

### 5.3.4. Mảng gồ ghề - Ragged Arrays

Mỗi dòng của một mảng 2 chiều là một mảng đơn. Vì vậy, mỗi dòng có thể có độ dài khác nhau. Một mảng như vậy được gọi là mảng gồ ghề.

Ví dụ:



Nếu biết kích thước của từng dòng:

```

int[][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];

```

Sau đó, chúng ta có thể gán giá trị cho mảng:

```

triangleArray[0][3] = 50;
triangleArray[4][0] = 45;

```

### 5.3.5. Các thao tác xử lý mảng 2 chiều

Giả sử một mảng **matrix** được tạo ra như sau:

```

int[][] matrix = new int[10][10];

```

#### 5.3.5.1. Khởi tạo mảng, với giá trị do người dùng nhập từ bàn phím

```

java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length +
    " rows and " + matrix[0].length + " columns: ");
for(int row = 0; row < matrix.length ; row++) {
    for(int column = 0; column < matrix[row].length; column++) {
        matrix[row][column] = input.nextInt();
    }
}

```

### 5.3.5.2. Khởi tạo mảng, với giá trị ngẫu nhiên từ 0 đến 99

```
for(int row = 0; row < matrix.length ; row++) {  
    for(int column = 0; column < matrix[row].length ; column++) {  
        matrix[row][column]=(int)(Math.random()*100);  
    }  
}
```

### 5.3.5.3. Xuất mảng 2 chiều ra màn hình

```
for(int row = 0; row < matrix.length ; row++) {  
    for(int column = 0; column < matrix[row].length ; column++) {  
        System.out.print(matrix[row][column] + " ");  
    }  
    System.out.println();  
}
```

### 5.3.5.4. Tính tổng tất cả các phần tử

```
int total = 0;  
  
for(int row = 0; row < matrix.length; row++) {  
    for(int column = 0; column < matrix[row].length; column++) {  
        total += matrix[row][column];  
    }  
}
```

### 5.3.5.5. Trộn ngẫu nhiên

```
for(int i = 0; i < matrix.length; i++) {  
    for(int j = 0; j < matrix[i].length; j++) {  
        int i1 = (int)(Math.random() * matrix.length);  
        int j1=(int)(Math.random()*matrix[i].length);  
        // Swap matrix[i][j] with matrix[i1][j1]  
        int temp = matrix[i][j];  
        matrix[i][j] = matrix[i1][j1];  
        matrix[i1][j1] = temp;  
    }  
}
```

### 5.3.5.6. Tính tổng các phần tử theo cột

```
for(int column = 0; column < matrix[0].length; column++){
    int total = 0;
    for(int row = 0; row < matrix.length; row++)
        total += matrix[row][column];
    System.out.println("Sum for column " + column + " is " + total);
}
```

## BÀI TẬP CHƯƠNG 5

Bài 5-1: Viết chương trình cho phép nhập 6 số từ bàn phím, tìm số lớn nhất và đếm số lần xuất hiện của giá trị đó. Giả sử nhập vào 3, 5, 2, 5, 5, và 5. Số lớn nhất là 5 và số lần xuất hiện là 4.

Bài 5-2. Viết chương trình phát sinh ngẫu nhiên 100 ký tự chữ thường và gán cho mảng mảng ký tự.

- Đếm lần số xuất hiện của mỗi ký tự trong mảng.
- Tìm trung bình (mean) và độ lệch chuẩn (deviation) của các lần đếm.

$$mean = \frac{\sum_{i=1}^n x_i}{n}, deviation = \sqrt{\frac{\sum_{i=1}^n (x_i - mean)^2}{n-1}}$$

Bài 5-3. Viết chương trình cộng và nhân 2 ma trận: sử dụng mảng 2 chiều để tạo 2 ma trận A, B rồi tính ma trận tổng A+B và ma trận tích A\*B.

**Cộng:**

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & a_{14}+b_{14} & a_{15}+b_{15} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & a_{24}+b_{24} & a_{25}+b_{25} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} & a_{34}+b_{34} & a_{35}+b_{35} \\ a_{41}+b_{41} & a_{42}+b_{42} & a_{43}+b_{43} & a_{44}+b_{44} & a_{45}+b_{45} \\ a_{51}+b_{51} & a_{52}+b_{52} & a_{53}+b_{53} & a_{54}+b_{54} & a_{55}+b_{55} \end{pmatrix}$$

**Nhân:**

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \end{pmatrix}$$

$$\text{Với } c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j} + a_{i4} \times b_{4j} + a_{i5} \times b_{5j}$$

Bài 5-4. Viết chương trình tạo một mảng số nguyên ngẫu nhiên có n phần tử:

- Xuất giá trị các phần tử của mảng.
- Tìm phần tử có giá trị lớn nhất, nhỏ nhất.

- Tính tổng giá trị của các phần tử là số nguyên tố.
- Đếm số phần tử có tổng các chữ số lớn hơn 10.
- Sắp xếp mảng tăng dần/giảm dần.
- Sắp xếp số chẵn giảm dần, số lẻ tăng dần.
- Tìm phần tử có giá trị x.

Bài 5-5. Cho ma trận số nguyên cấp nxm. Cài đặt các hàm sau:

- Nhập ma trận.
- In ma trận.
- Tìm phần tử nhỏ nhất.
- Tìm phần tử lẻ lớn nhất.
- Tìm dòng có tổng lớn nhất.
- Tính tổng các số không phải là số nguyên tố.

Bài 5-6. Cho ma trận vuông số nguyên cấp n. Cài đặt các hàm sau:

- Nhập ma trận.
- In ma trận.
- Tổng các phần tử thuộc tam giác trên.
- Tổng các phần tử thuộc tam giác dưới.
- Kiểm tra xem ma trận có đối xứng qua đường chéo chính?

Bài 5-7. Tìm kiếm tuyến tính: viết chương trình tạo ngẫu nhiên 10 phần tử mảng kiểu **int** rồi hiển thị. Nhận **key** từ bàn phím rồi thực hiện tìm kiếm tuyến tính.

Bài 5-8. Tìm kiếm nhị phân: viết chương trình tạo mảng 10 phần tử kiểu **int** rồi hiển thị. Nhận **key** từ bàn phím rồi thực hiện tìm kiếm nhị phân.

Bài 5-9. Sắp xếp mảng: viết chương trình sử dụng các phương pháp sắp xếp (Sort) để sắp xếp dãy số thực.

# CHƯƠNG 6 - HƯỚNG ĐỐI TƯỢNG TRONG JAVA

## 6.1. Các khái niệm cơ bản

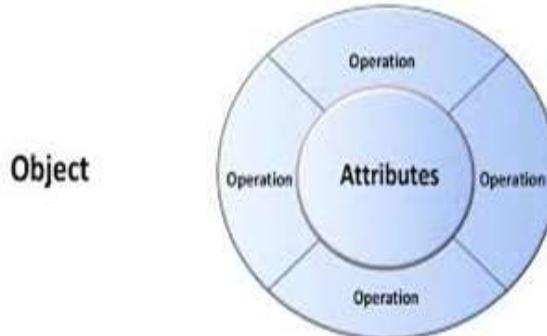
### 6.1.1. Đối tượng

- Trong thế giới thực: sinh viên, xe ô tô, thẻ ngân hàng,...
- Mỗi đối tượng đều có:
  - Các thông tin, trạng thái: Sinh viên có: họ tên, ngày sinh, trường, lớp,...
  - Các hoạt động: Sinh viên có các hoạt động: ăn, ngủ, đi học,...



Hình 6.1 Đối tượng Thẻ ngân hàng

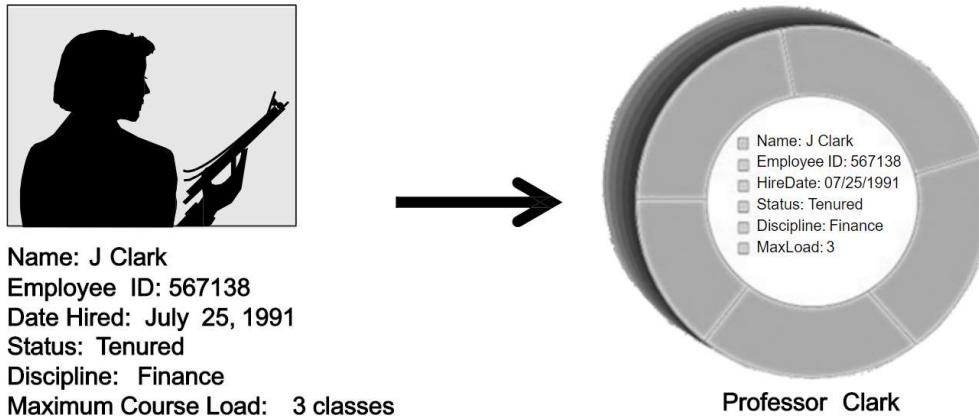
- Đối tượng là duy nhất: không có hai đối tượng giống nhau dù cùng chia sẻ các tính chất, trạng thái.
- Mô hình hóa vào trong lập trình: đóng gói thành trạng thái (*state*) và hành vi (*behavior*).
  - Trạng thái được biểu diễn bởi các thuộc tính (*attributes*) và các mối quan hệ (*relationships*).
  - Hành vi được biểu diễn bởi các thao tác (*operations*) hay phương thức (*methods*).



Hình 6.2 Mô hình hóa đối tượng trong lập trình

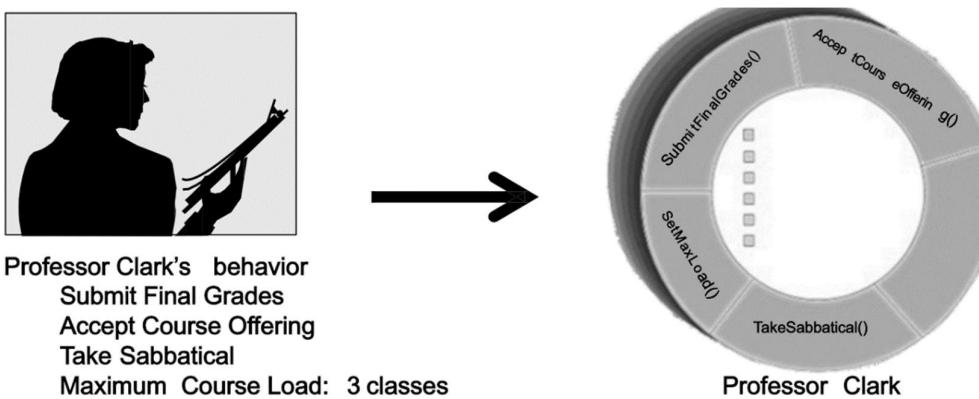
### 6.1.2. Trạng thái và hành vi

- Trạng thái của đối tượng là 1 trong các điều kiện mà tại đó đối tượng tồn tại.
- Trạng thái của một đối tượng có thể thay đổi theo thời gian.



Hình 6.3 Trạng thái của đối tượng

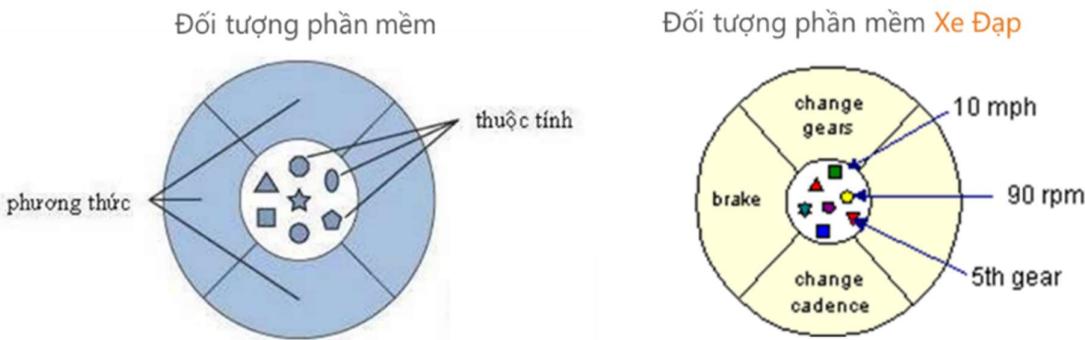
- Hành vi quyết định đối tượng đó hành động và đáp trả như thế nào đối với bên ngoài.
- Hành vi nhìn thấy được của một đối tượng được mô hình thành một tập các thông điệp nó có thể đáp trả (các thao tác mà đối tượng đó thực hiện).



Hình 6.4 Hành vi của đối tượng

### 6.1.3. Đối tượng phần mềm

- Đối tượng (*object*) là một thực thể phần mềm bao bọc các thuộc tính và các phương thức liên quan.
  - Thuộc tính được xác định bởi giá trị cụ thể gọi là **thuộc tính thể hiện**.
  - Một đối tượng cụ thể được gọi là **một thể hiện**.



Hình 6.5 Đối tượng phần mềm

#### 6.1.4. Lớp

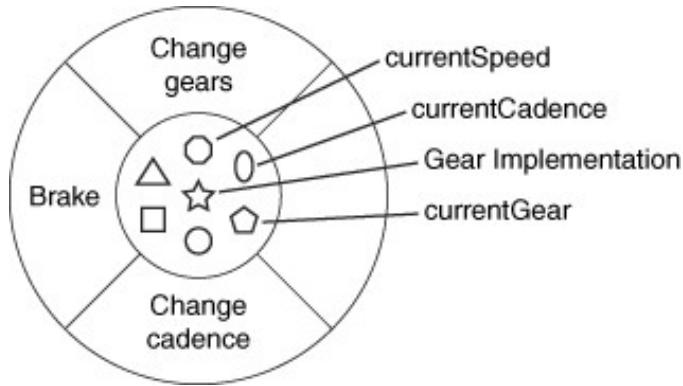
- Lớp là một thiết kế (*blueprint*), mẫu (*prototype*) cho các đối tượng cùng kiểu.

*Ví dụ:*

- Lớp *XeDap* là thiết kế chung cho nhiều đối tượng *xe đạp* được tạo ra.
- Lớp định nghĩa các thuộc tính và các phương thức chung cho tất cả các đối tượng của cùng một loại nào đó.
- Một đối tượng là một thể hiện cụ thể của một lớp.

*Ví dụ:*

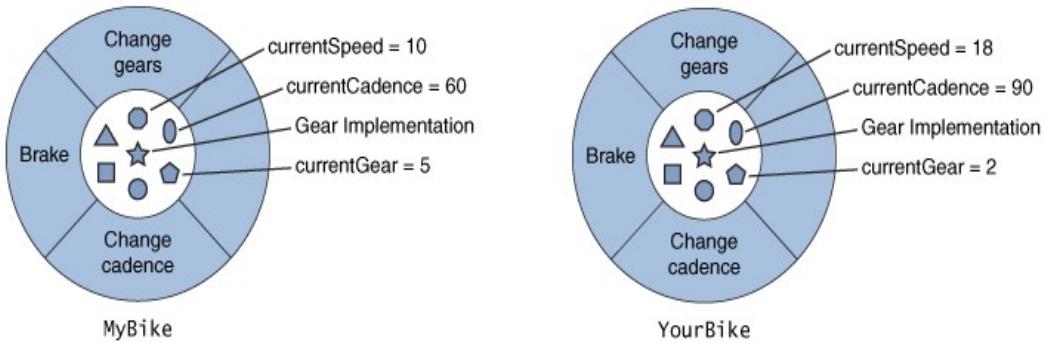
- Mỗi đối tượng *xe đạp* là một thể hiện của lớp *XeDap*
- Mỗi thể hiện có thể có những thuộc tính thể hiện khác nhau.



Hình 6.6 Khai báo lớp *XeDap*

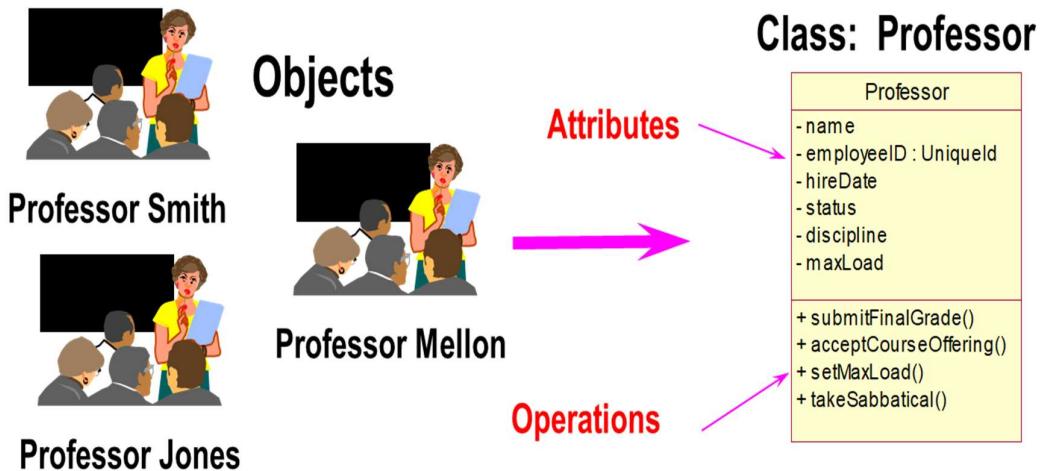
*Ví dụ:*

- Một *xe đạp* có thể đang ở bánh răng thứ 5 trong khi một xe khác có thể là đang ở bánh răng thứ 2.



Hình 6.7 Các đối tượng của lớp *XeDap*

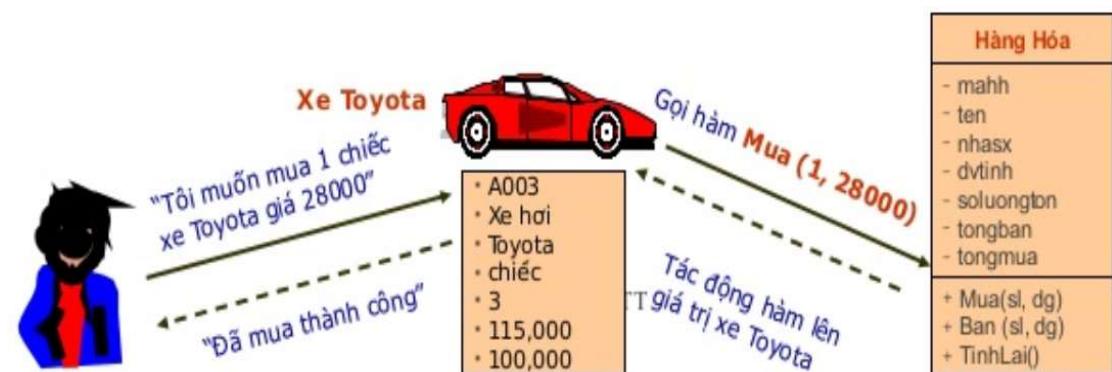
#### 6.1.5. Lớp và đối tượng



Hình 6.8 Lớp và đối tượng *Professor*

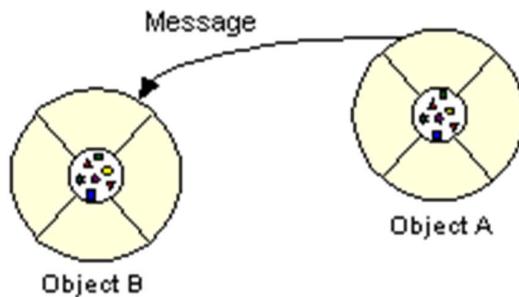
#### 6.1.6. Tương tác giữa các đối tượng

- Các đối tượng trong thế giới thực có thể tương tác được với nhau.



Hình 6.9 Tương tác trong thế giới thực

- Trong lập trình các đối tượng giao tiếp với nhau bằng cách gửi thông điệp.



Hình 6.10 Gửi thông điệp (tương tự với gọi hàm)

- Gọi hàm (call function):

- Chỉ ra chính xác đoạn mã nào sẽ được thực hiện.
- Chỉ có duy nhất một sự thực thi của một hàm với một tên nào đó.
- Không có hai hàm trùng tên.

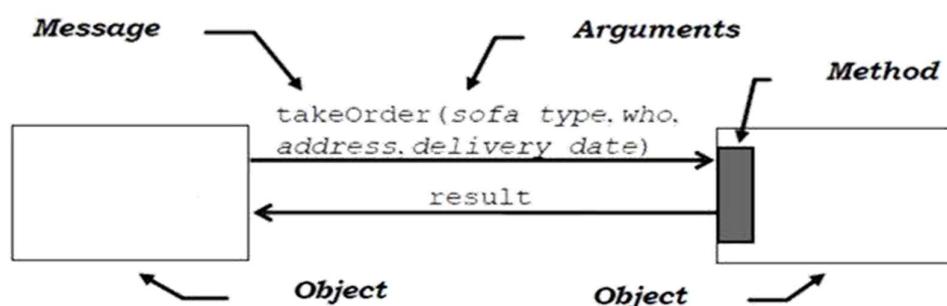
- Gửi thông điệp (send message):

- Yêu cầu một dịch vụ từ một đối tượng và đối tượng sẽ quyết định cần phải làm gì.
  - Các đối tượng khác nhau sẽ có các cách thực thi các thông điệp theo cách khác nhau.

– Thông điệp: được gửi từ đối tượng này đến đối tượng kia, không bao gồm đoạn mã thực sự sẽ được thực thi.

- Phương thức:

- Thủ tục/hàm trong ngôn ngữ lập trình cấu trúc.
- Là sự thực thi dịch vụ được yêu cầu bởi thông điệp.
- Là đoạn mã sẽ được thực thi để đáp ứng thông điệp được gửi đến cho đối tượng.



Hình 6.11 Thông điệp và Phương thức

## 6.2. Lớp và xây dựng lớp

### 6.2.1. Lớp và thành phần của lớp

#### 6.2.1.1. Lớp (*class*)

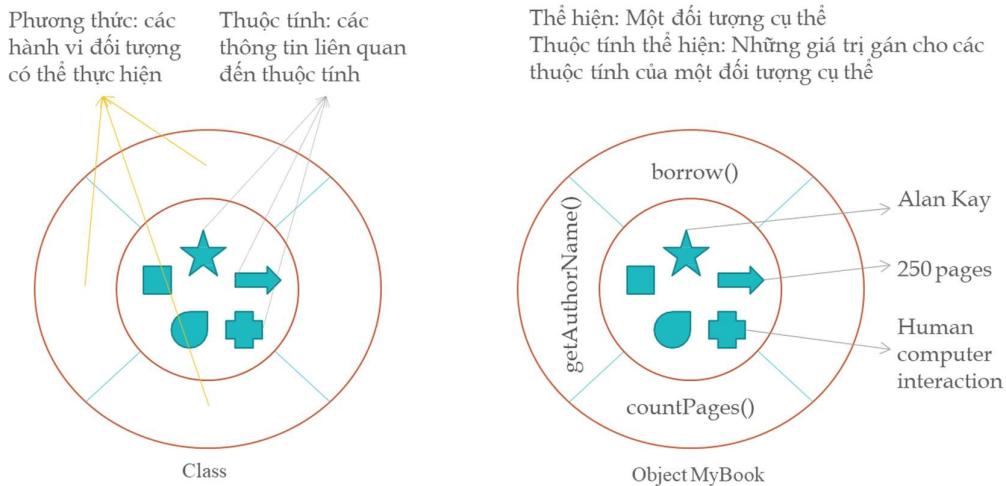
- Lớp (*class*) là cách phân loại (*classify*) các đối tượng dựa trên đặc điểm chung của các đối tượng đó.
- Lớp có thể coi là khuôn mẫu để tạo các đối tượng.

*Ví dụ:* Người, Xe, Động vật,...

- Lớp chính là kết quả của quá trình trừu tượng hóa dữ liệu:
  - Lớp định nghĩa 1 kiểu dữ liệu mới, trừu tượng hóa 1 tập các đối tượng.
  - Một đối tượng gọi là một thể hiện của lớp.

#### 6.2.1.2. Các thành phần của lớp

- Lớp đóng gói các *phương thức* và *thuộc tính* chung của các đối tượng



Hình 6.12 Các thành phần của một lớp

#### *Thuộc tính:*

- Một thuộc tính của một lớp là một trạng thái chung được đặt tên của tất cả các thể hiện của lớp đó có thể có.

*Ví dụ:* Lớp Ô tô có các thuộc tính: Màu sắc, Vận tốc, Hãng sản xuất,...

- Các thuộc tính của lớp cũng là các giá trị trừu tượng. Mỗi đối tượng có bản sao các thuộc tính của riêng nó.

*Ví dụ:* một chiếc Ô tô đang đi có thể có màu đen, vận tốc 60 km/h.

**Phương thức:**

- Xác định các hoạt động chung mà tất cả các thể hiện của lớp thực hiện được.
- Xác định cách một đối tượng đáp ứng lại một thông điệp.
- Thông thường các phương thức sẽ hoạt động trên các thuộc tính và thường làm thay đổi các trạng thái của lớp.
- Bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó.

**Ví dụ:** Lớp **Ô tô** có các phương thức: Tăng tốc độ, Giảm tốc độ,...

- Phạm vi nhìn thấy được xác định khả năng nhìn thấy được của một thành phần của chương trình với các thành phần khác của chương trình.
  - Đối với lớp, phạm vi nhìn thấy được có thể được áp dụng cho các thành phần của lớp: **private**: chỉ truy cập được bên trong lớp đó; **public**: có thể truy cập được tại mọi nơi.

### 6.2.2. Xây dựng lớp

#### 6.2.2.1. Gói

- Gói (**package**) giống như thư mục giúp:
  - Tổ chức, xác định vị trí lớp dễ dàng và sử dụng các lớp 1 cách phù hợp.
  - Tránh cho việc đặt tên lớp bị xung đột (trùng tên): Các **package** khác nhau có thể chứa các lớp có cùng tên.
    - Bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn so với mối quan hệ giữa các lớp.
    - Một **package** cũng có thể chứa các **package** khác.

#### 6.2.2.2. Gói trong Java

- Có thể tự tạo ra các gói để tổ chức các lớp:

**Cú pháp:**

```
package <tên gói>;
```

- Tên gói phải được viết trên cùng của file mã nguồn.
- Chỉ được phép có một câu khai báo gói trong mỗi file mã nguồn, và khai báo này sẽ được áp dụng cho tất cả các dữ liệu trong file đó.

- Một gói có thể được đặt trong một gói khác: Phân cách bằng dấu .

*Ví dụ:*

```
package vn.edu.caothang.cntt;
```

*Quy ước đặt tên gói:*

- Tên gói được viết toàn bộ bằng chữ thường để tránh xung đột với tên lớp hay giao diện.
- Công ty có tên miền Internet: Sử dụng tên miền đảo để đặt tên gói.

*Ví dụ:* tên miền **example.com** sẽ đặt tên gói là **com.example.mypackage**.

### 6.2.2.3. Các package trong Java

java.applet	javax.rmi
java.awt	javax.security
java.beans	javax.sound
java.io	javax.sql
java.lang	javax.swing
java.math	javax.transaction
java.net	javax.xml
java.nio	org.ietf.jgss
java.rmi	org.omg.CORBA
java.security	org.omg.CosNaming
java.sql	org.omg.Dynamic
java.text	org.omg.IOP
java.util	org.omg.Messaging
javax.accessibility	org.omg.PortableInterceptor
javax.crypto	org.omg.PortableServer
javax.imageio	org.omg.SendingContext
javax.naming	org.omg.stub.java.rmi
javax.net	org.w3c.dom
javax.print	org.xml

– **java.lang:**

- Cung cấp các lớp cơ bản cho thiết kế ngôn ngữ lập trình **Java**.
- Bao gồm **Wrapper classes, String, StringBuffer, Object,...**
- **Import** ngầm định vào tất cả các lớp.

– **java.util:**

- Bao gồm tập hợp framework, mô hình sự kiện, date time,...

– **java.io:**

- Cung cấp khả năng vào/ra hệ thống với các luồng dữ liệu và file.

– **java.math:**

- Cung cấp các lớp thực thi các phép toán với số.

– **java.sql:**

- Cung cấp các API cho phép truy nhập và xử lý dữ liệu được lưu trữ trong một nguồn dữ liệu (thường sử dụng cơ sở dữ liệu quan hệ).

– **javax.swing:**

- Cung cấp các lớp và giao diện cho phép tạo ra các ứng dụng đồ họa.

– ...

#### 6.2.2.4. Khai báo lớp

**Cú pháp:** sử dụng từ khóa **class**

```
class <Tên Lớp> {  
    // Nội dung lớp  
}
```

**Ví dụ:**

```
class Dog {  
    // Nội dung lớp  
}
```

**Cú pháp khai báo lớp sử dụng chỉ định truy cập:**

```
accessmodifier class <Tên Lớp> {  
    // Nội dung lớp  
}
```

- Chỉ định truy cập:
  - **public**: lớp có thể được truy cập từ bất cứ đâu, kể cả bên ngoài **package** chứa lớp đó.
  - **private**: lớp chỉ có thể được truy cập trong phạm vi lớp đó.
  - **mặc định**: lớp có thể được truy cập từ bên trong **package** chứa lớp đó.

Bảng 6-1 Chỉ định truy cập

	<b>public</b>	<b>mặc định</b>	<b>private</b>
Cùng lớp	✓	✓	✓
Cùng gói	✓	✓	✗
Khác gói	✓	✗	✗

#### 6.2.2.5. Thuộc tính

- Là các thông tin, trạng thái mà đối tượng của lớp đó có thể mang.
- Các thuộc tính phải được khai báo bên trong lớp.
- Mỗi đối tượng có bản sao các thuộc tính của riêng nó.
  - Giá trị của một thuộc tính thuộc các đối tượng khác nhau là khác nhau.
  - Bản chất của các thuộc tính là các thành phần dữ liệu của đối tượng.
    - Khai báo: tương tự như biến.

**Cú pháp khai báo thuộc tính:**

```
accessmodifier kiểu tênThuộcTính;
```

- Thuộc tính có thể được khởi tạo khi khai báo.
  - Các giá trị mặc định sẽ được sử dụng nếu không được khởi tạo.

```
package com.megabank.models;

public class BankAccount {
    private String owner;
    private double balance = 0.0;
}
```

**access modifier**

**type**

**name**

### 6.2.2.6. Phương thức

- Xác định cách một đối tượng đáp ứng lại thông điệp.
- Phương thức xác định các hoạt động của lớp.
- bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó.

Cú pháp:

```
accessmodifier kiểuTrảVề tênPhươngThức (ds tham số) {  
    // Nội dung phương thức  
}
```

access modifier    return type    method name        parameter list

```
public void debit( double amount ) {  
    //Method body  
}
```

### 6.2.3. Tạo và sử dụng đối tượng

#### 6.2.3.1. Khai báo và khởi tạo dữ liệu

- Trong Java, mọi dữ liệu cần phải được khai báo và khởi tạo trước khi sử dụng

Cú pháp khai báo:

```
<kiểu> tên biến;  
<TênLớp> tên đối tượng;
```

Ví dụ khai báo:

```
int i;  
BankAccount acc; // Đối tượng acc là một BankAccount
```

- Khởi tạo:

- Đối với kiểu dữ liệu nguyên thủy: Dùng toán tử “=”.
- Đối với kiểu dữ liệu tham chiếu: Khởi tạo bằng toán tử **new**

Ví dụ khởi tạo:

```
i = 3;  
acc = new BankAccount();
```

- Nếu không được khởi tạo: đối tượng mang giá trị **null**.
- Khi đối tượng được khởi tạo, các thành phần dữ liệu (thuộc tính) của đối tượng được khởi tạo với giá trị mặc định của kiểu dữ liệu tương ứng.
  - number data type  $\leftarrow 0$ ;
  - reference type  $\leftarrow \text{null}$ ;
  - boolean  $\leftarrow \text{false}$ ;
- Kết hợp khai báo và khởi tạo: Có thể kết hợp khai báo và khởi tạo dữ liệu

**Cú pháp:**

```
<Tên Lớp> tên đối tượng = new <Tên Lớp>();
```

**Ví dụ:**

```
BankAccount acc = new BankAccount();
int i = 3;
```

#### 6.2.3.2. Truy cập đến phương thức và thuộc tính của đối tượng

- Sử dụng toán tử “.”.
- Không cần thiết nếu truy cập từ trong cùng một lớp.

```
BankAccount account = new BankAccount() ;
account.setOwner("Alan Kay") ;
account.credit(5000000.00) ;
```

BankAccount method

```
public void credit(double amount) {
    setBalance(getBalance() + amount) ;
}
```

#### 6.2.3.3. Tự tham chiếu

- Sử dụng từ khóa **this**.
- Cho phép truy cập vào đối tượng hiện tại của lớp.
- Quan trọng khi phương thức thành phần thao tác trên 2 hay nhiều đối tượng.
- Xóa đi sự nhập nhằng giữa biến cục bộ, tham số với thành phần dữ liệu lớp.
- Không dùng bên trong các khối lệnh **static**.

Ví dụ:

```
public class Account {  
    // instance variable  
    String owner; // Account name  
    long balance; // Balance  
    //...  
    // value setting method  
    void setAccountInfo(String owner, long balance) {  
        this.owner = owner;  
        this.balance = balance;  
    }  
    //...  
}
```

#### 6.2.3.4. Tham chiếu đến lớp khác gói

- Đối với lớp trong cùng một gói: chỉ cần tên lớp.

Ví dụ: **BankAccount**

- Đối với lớp khác gói: phải cung cấp đầy đủ tên lớp và tên gói.

Ví dụ trong Java:

```
vn.edu.caothang.cntt.BankAccount
```

- Sử dụng lệnh **import** để khai báo các **package** hoặc các lớp để khi sử dụng không cần nêu tên đầy đủ.

```
import javax.swing.JOptionPane;  
String result;  
  
result = JOptionPane.showInputDialog("Hay nhap ten ban:");  
JOptionPane.showMessageDialog(null, "Xin chao " + result + "!");
```

hoặc **import javax.swing.\*;**  
để **import** tất cả các lớp  
trong gói

Hoặc:

```
String result;  
  
result = javax.swing.JOptionPane.showInputDialog("Hay nhap ten:");  
javax.swing.JOptionPane.showMessageDialog(null, "Chao " + result);
```

#### 6.2.4. Thành viên hằng và tĩnh

##### 6.2.4.1. Thành viên tĩnh

- Trong lập trình cấu trúc, các biến địa phương khai báo cục bộ trong hàm:
  - Trong trường hợp các biến địa phương không khai báo là biến **static** thì mỗi lần gọi hàm chương trình dịch lại đăng ký tạo ra biến mới.
  - Khi chúng ta khai báo các biến địa phương là các biến **static** thì chương trình dịch sẽ chỉ khởi tạo duy nhất một lần (ở lần gọi đầu tiên) biến địa phương này và thông qua con trỏ **stack** ở những lần gọi sau chỉ tham chiếu tới biến đã tạo ra này để sử dụng lại chúng mà không tạo ra biến mới.
  - Tạo một lần/tham chiếu nhiều lần/lưu giá trị của lần tham chiếu trước.

Biến địa phương	Biến địa phương không static
<pre>void f() {     static int x = 0;     x++; }</pre>	<pre>void f() {     int x = 0;     x++; }</pre>
Lần gọi 1: f()	0
Lần gọi 2: f()	1

##### 6.2.4.2. Thành viên tĩnh trong OOP

- Trong lập trình hướng đối tượng:
  - Các thành viên bình thường là thành viên thuộc về đối tượng.
  - Các thành viên tĩnh (**static**) là các thành viên thuộc về lớp.

Cú pháp khai báo thành viên **static**:

<chỉ định truy cập> **static** <kiểu> tên biến;

##### 6.2.4.3. Thuộc tính static

- Là thuộc tính mang thông tin chung của một lớp.
- Thay đổi giá trị của một thành viên **static** trong một đối tượng của lớp sẽ thay đổi giá trị của thành viên này của tất cả các đối tượng khác của lớp đó.

Ví dụ:

```
class TestStatic {  
    public static int iStatic;  
    public int iNonStatic;  
}  
  
public class Test {  
    public static void main(String[] args) {  
        TestStatic obj1 = new TestStatic();  
        obj1.iStatic = 10;  
        obj1.iNonStatic = 11;  
        System.out.println(obj1.iStatic + " , " + obj1.iNonStatic);  
        TestStatic obj2 = new TestStatic();  
        System.out.println(obj2.iStatic + " , " + obj2.iNonStatic);  
        obj2.iStatic = 12;  
        System.out.println(obj1.iStatic + " , " + obj1.iNonStatic);  
    }  
}
```

Kết quả chạy chương trình:

```
10 , 11  
10 , 0  
12 , 11
```

#### 6.2.4.4. Khối static trong Java

- Được sử dụng để khởi tạo thành viên dữ liệu **static**.
- Nó được thực thi trước phương thức **main** tại lúc tải lớp.

Ví dụ về khối static trong Java

```
public class KhoiStatic {  
    static { System.out.println("Khoi static!"); }  
    public static void main(String args[]) {  
        System.out.println("Ham main!");  
    }  
}
```

Kết quả:

Khai static!

Hàm main!

#### 6.2.4.5. Phương thức static

- Các phương thức không tương tác với các thành phần của lớp.
- Các phương thức tiện ích, không cần thiết phải khởi tạo đối tượng để sử dụng.
- Các phương thức **static** chỉ có thể truy cập vào các thuộc tính **static** và chỉ có thể gọi các phương thức **static** trong cùng lớp.

Ví dụ:

```
class MyUtils {  
    public static double mean(int[] p) {  
        int sum = 0;  
        for (int i = 0; i < p.length; i++) {  
            sum += p[i];  
        }  
        return ((double)sum) / p.length;  
    }  
}  
// Gọi phương thức tĩnh bên trong lớp  
double avgAtt = mean(attendance);  
// Gọi phương thức tĩnh bên ngoài lớp  
double avgAtt = MyUtils.mean(attendance);
```

- Phương thức **mean** có thể không phải khai báo **static** tuy nhiên muốn gọi nó phải thông qua một đối tượng.

#### 6.2.4.6. Thành viên lớp và thành viên đối tượng

Thành viên đối tượng	Thành viên lớp ( <b>static</b> )
<ul style="list-style-type: none"><li>• Thuộc tính/phương thức chỉ được truy cập thông qua đối tượng.</li></ul>	<ul style="list-style-type: none"><li>• Thuộc tính/phương thức có thể được truy cập thông qua lớp.</li></ul>
<ul style="list-style-type: none"><li>• Mỗi đối tượng có một bản sao riêng của một thuộc tính đối tượng.</li></ul>	<ul style="list-style-type: none"><li>• Các đối tượng có chung một bản sao của một thuộc tính lớp.</li></ul>
<ul style="list-style-type: none"><li>• Giá trị của một thuộc tính đối tượng của các đối tượng khác nhau là khác nhau.</li></ul>	<ul style="list-style-type: none"><li>• Giá trị của một thuộc tính lớp của các đối tượng khác nhau là giống nhau.</li></ul>

**Ví dụ:** Lớp **JOptionPane** trong **javax.swing**

Field Summary	
Fields	
Modifier and Type	Field and Description
static int	<b>CANCEL_OPTION</b> Return value from class method if CANCEL is chosen.
static int	<b>CLOSED_OPTION</b> Return value from class method if user closes window without selecting anything, more than likely this should be treated as either a CANCEL_OPTION or NO_OPTION.
static int	<b>DEFAULT_OPTION</b> Type meaning Look and Feel should not supply any options -- only use the options from the JOptionPane.
static int	<b>ERROR_MESSAGE</b> Used for error messages.

Hình 6.13 Thuộc tính **static** của **JOptionPane**

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description		
static void	<b>setRootFrame(Frame newRootFrame)</b> Sets the frame to use for class methods in which a frame is not provided.		
static int	<b>showConfirmDialog(Component parentComponent, Object message)</b> Brings up a dialog with the options Yes, No and Cancel; with the title, <b>Select an Option</b> .		
static int	<b>showConfirmDialog(Component parentComponent, Object message, String title, int optionType)</b> Brings up a dialog where the number of choices is determined by the optionType parameter.		
static int	<b>showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType)</b> Brings up a dialog where the number of choices is determined by the optionType parameter, where the messageType parameter determines the icon to display.		
static int	<b>showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)</b> Brings up a dialog with a specified icon, where the number of choices is determined by the optionType parameter.		

Hình 6.14 Phương thức **static** của **JOptionPane**

```
import javax.swing.JOptionPane;
public class JYesNoOption {
    public static void main(String[] args) {
        JOptionPane.showConfirmDialog(null,
        "Ban co muon thoat khong?", "Thoat", JOptionPane.YES_NO_OPTION);
    }
}
```

Kết quả chạy:



Hình 6.15 **ConfirmDialog** của **JOptionPane**

#### 6.2.4.7. Thành viên hằng

- Một thuộc tính/phương thức không thể thay đổi giá trị/nội dung trong quá trình sử dụng.

**Cú pháp khai báo:** Sử dụng từ khóa **final**

```
<chỉ định truy cập> final <kiểu> tên hằng = giá trị;
```

*Ví dụ:*

```
final double PI = 3.141592653589793;  
public final int VAL_THREE = 39;  
private final int[] A = { 1, 2, 3, 4, 5, 6 };
```

#### 6.2.4.8. Biến final trống

- Biến **final** không khởi tạo tại thời điểm khai báo được gọi là biến **final** trống.

*Ví dụ:*

```
public class Student {  
    int id;  
    String name;  
    final String NUMBER; // Biến final trống  
    public Student() {  
        NUMBER = "0306170012";  
    }  
}
```

#### 6.2.4.9. Phương thức final trong Java

- Trong Java, không thể ghi đè phương thức **final**.

*Ví dụ:*

```
class Bike {  
    final void run() {  
        System.out.println("Running!");  
    }  
}
```

```
public class Honda extends Bike {  
    void run() { System.out.println("Tốc độ 80Km/h!"); }  
    public static void main(String args[]) {  
        Honda xeLead = new Honda();  
        xeLead.run();  
    }  
}
```

Kết quả:

```
Honda.java:7: error: run() in Honda cannot override run() in Bike
```

#### 6.2.4.10. Lớp **final** trong Java

- Trong Java, không thể kế thừa lớp **final**.

*Ví dụ:*

```
final class Bike {}  
public class Yamaha extends Bike {  
    void run() {  
        System.out.println("Tốc độ 80Km/h!");  
    }  
    public static void main(String args[]) {  
        Yamaha jupiter = new Yamaha();  
        jupiter.run();  
    }  
}
```

Kết quả:

```
Yamaha.java:2: error: cannot inherit from final Bike
```

- Phương thức **final** được kế thừa nhưng không thể ghi đè.

#### 6.2.4.11. Thành viên hằng và tĩnh

- Thông thường các hằng số liên quan đến lớp, được khai báo là **static final**.

*Ví dụ:*

```
public class MyClass {  
    public static final double PI;  
    static {  
        PI = 3.14159;  
    }  
    public static void main(String args[]) {  
        double banKinh = 20.0;  
        System.out.println("Dien tich hinh tron : ");  
        System.out.println(banKinh*banKinh*MyClass.PI);  
    }  
}
```

## 6.2.5. Một số kỹ thuật xây dựng lớp

### 6.2.5.1. Phương thức khởi tạo (Constructor)

#### *Khởi tạo đối tượng:*

- Để khởi tạo đối tượng: sử dụng từ khóa **new**.

#### *Ví dụ:*

```
BankAccount acc = new BankAccount();
```

**Phương thức khởi  
tạo (ngầm định)**

- Phương thức khởi tạo ngầm định (implicit) sẽ tự động gán các giá trị mặc định cho các thành phần dữ liệu.
  - Được thực hiện trước khi lập trình viên có thể tác động lên đối tượng.
  - Nếu không muốn sử dụng phương thức khởi tạo ngầm định → có thể viết phương thức khởi tạo tường minh (explicit) cho lớp.

#### *Phương thức khởi tạo:*

- Là phương thức được gọi để gán các giá trị cho các thành phần dữ liệu khi đối tượng được khởi tạo:
  - Tên của phương thức khởi tạo trùng với tên lớp;
  - Không có kiểu dữ liệu trả về (kể cả **void**);
  - Có thể có hoặc không có tham số.
- Phương thức khởi tạo không được coi là thành viên của lớp.

- Java sẽ không sử dụng phương thức khởi tạo ngầm định khi đã viết phương thức khởi tạo cho lớp.

*Ví dụ:* phương thức khởi tạo không có tham số (còn gọi là constructor mặc định).

```
class BankAccount {  
    private String owner;  
    private long balance;  
    public BankAccount() {  
        this.owner = "NONAME";  
        this.balance = 0;  
    }  
}
```

*Ví dụ:* phương thức khởi tạo có tham số.

```
class BankAccount {  
    private String owner;  
    private long balance;  
    public BankAccount(String owner, double balance) {  
        this.owner = owner;  
        this.balance = balance;  
    }  
}
```

- Mục đích: giúp khởi tạo đối tượng dễ dàng hơn.

```
BankAccount account = new BankAccount("Alan Kay", 5000000);
```

*Lưu ý:* khi người dùng đã cài đặt phương thức khởi tạo, phương thức khởi tạo ngầm định do Java cung cấp không còn nữa.

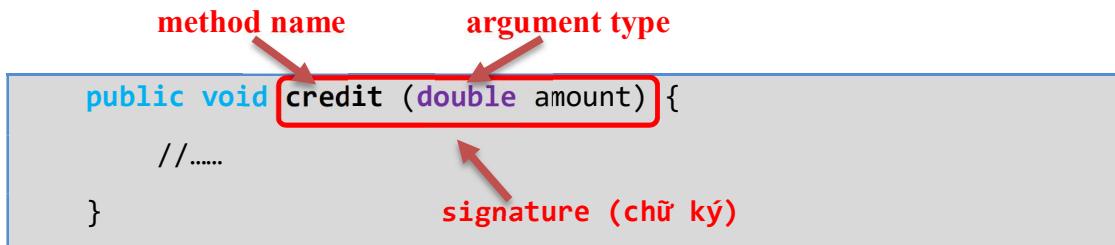
```
public class BankAccount {  
    private String owner;  
    private double balance;  
    public BankAccount(String name) {  
        owner = name;  
    }  
}
```

```
public class Test {  
    public static void main(String args[]) {  
        BankAccount account1 = new BankAccount(); //Error  
        BankAccount account2 = new BankAccount("Alan Kay");  
    }  
}
```

### 6.2.5.2. Nạp chồng (Overload)

*Chữ ký phương thức:*

- Chữ ký của phương thức bao gồm:
  - Tên phương thức (**method name**).
  - Số lượng các đối số và kiểu của chúng.



*Nạp chồng phương thức:*

- Nạp chồng hay chồng phương thức (**method overloading**): các phương thức trong cùng một lớp có thể trùng tên nhưng khác chữ ký.
  - Số lượng tham số khác nhau.
  - Nếu cùng số lượng tham số thì kiểu dữ liệu các tham số phải khác nhau.
- Mục đích:
  - Tên trùng nhau để mô tả bản chất công việc.
  - Thuận tiện cho lập trình vì không cần nhớ quá nhiều tên phương thức mà chỉ cần nhớ một tên và lựa chọn các tham số cho phù hợp.

*Ví dụ:*

```
class MyDate {  
    int year, month, day;  
    public boolean setMonth(int m) { ... }  
    public boolean setMonth(String s) { ... }  
}
```

```
public class Test{  
    public static void main(String args[]){  
        MyDate d = new MyDate();  
        d.setMonth(9);  
        d.setMonth("September");  
    }  
}
```

- Trong Java:

- Phương thức *println()* trong System.out.*println()* có 10 khai báo với các tham số khác nhau: *boolean, char[], char, double, float, int, long, Object, String*, và một không có tham số.
- Không cần sử dụng các tên khác nhau (chẳng hạn *printString* hoặc *printDouble*) cho mỗi kiểu dữ liệu muốn hiển thị.

**Chú ý:**

- Các phương thức chỉ được xem xét là chồng khi chúng thuộc cùng một lớp.
- Chỉ nên sử dụng kỹ thuật này với các phương thức có cùng mục đích, chức năng; tránh lạm dụng.
- Khi dịch, trình dịch căn cứ vào số lượng hoặc kiểu dữ liệu của tham số để quyết định gọi phương thức nào phù hợp.  
→ Nếu không chọn được hoặc chọn được nhiều phương thức thì sẽ báo lỗi.

**Chồng phương thức khởi tạo:**

- Trong cùng một lớp ta có thể xây dựng nhiều phương thức khởi tạo với danh sách tham số khác nhau.
  - Chồng phương thức khởi tạo (*constructor overloading*).
- Nếu muốn gọi đến phương thức khởi tạo khác của lớp: sử dụng toán tử *this*.

**This** (danh sách tham số);

### 6.2.5.3. Kết tập (Aggregation)

**Mối quan hệ kết tập:**

- Lớp toàn thể chứa các đối tượng của lớp thành phần.

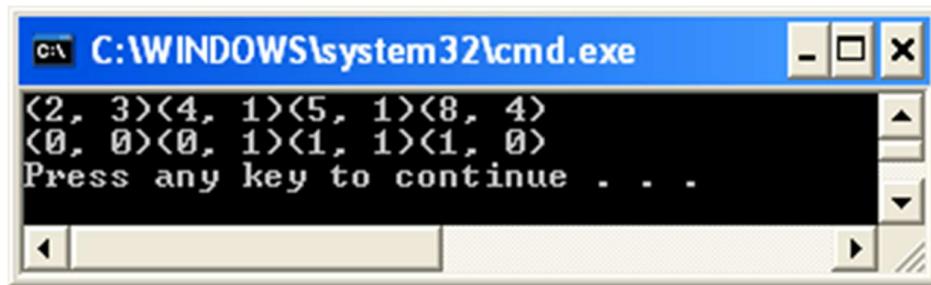
- Đối tượng lớp thành phần: là một phần (*is-a-part of*) của lớp toàn thể.
- Quan hệ chứa/có (“*has-a*”) hoặc là một phần (*“is-a-part-of”*).

Ví dụ:

- Tú giác gồm 4 Điểm
- Ô tô gồm 4 Bánh xe

```
class Diem {  
    private int x, y;  
  
    public Diem() {}  
  
    public Diem(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public void setX(int x) { this.x = x; }  
    public int getX() { return x; }  
    public void hienThiDiem(){  
        System.out.print("(" + x + ", " + y + ")");  
    }  
}  
  
class TuGiac {  
    private Diem d1, d2, d3, d4;  
  
    public TuGiac(Diem p1, Diem p2, Diem p3, Diem p4) {  
        d1 = p1;      d2 = p2;      d3 = p3;      d4 = p4;  
    }  
    public TuGiac() {  
        d1 = new Diem();      d2 = new Diem(0,1);  
        d3 = new Diem (1,1); d4 = new Diem (1,0);  
    }  
    public void printTuGiac() {  
        d1.printDiem();      d2.printDiem();  
        d3.printDiem();      d4.printDiem();  
        System.out.println();  
    }  
}
```

```
public class Test {  
    public static void main(String args[])  
    {  
        Diem d1 = new Diem(2,3);      Diem d2 = new Diem(4,1);  
        Diem d3 = new Diem(5,1);      Diem d4 = new Diem(8,4);  
        TuGiac tg1 = new TuGiac(d1, d2, d3, d4);  
        TuGiac tg2 = new TuGiac();  
        tg1.printTuGiac();  
        tg2.printTuGiac();  
    }  
}
```



#### Bản chất của kết tập:

- Kết tập (aggregate)

- Các thành phần của lớp mới là các đối tượng của các lớp có sẵn.
- Tái sử dụng các thành phần dữ liệu và các hành vi của lớp thành phần thông qua đối tượng của lớp thành phần.

- Lớp mới: lớp toàn thể (Aggregate/Whole)

- Lớp cũ: lớp thành phần (Part).

#### Thứ tự khởi tạo:

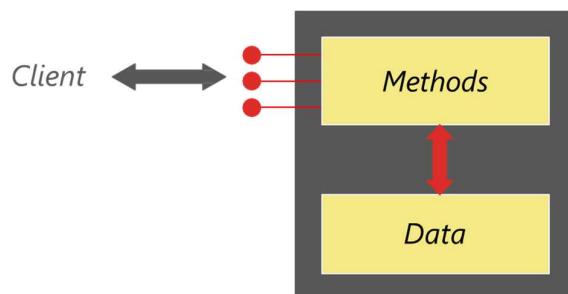
- Khi một đối tượng được tạo mới, các thuộc tính của đối tượng đó đều phải được khởi tạo và gán những giá trị tương ứng.
- Các đối tượng thành phần được khởi tạo trước.
  - Các phương thức khởi tạo các đối tượng của lớp thành phần được thực hiện trước.

### 6.3. Đặc điểm hướng đối tượng trong Java

#### 6.3.1. Đóng gói (Encapsulation)

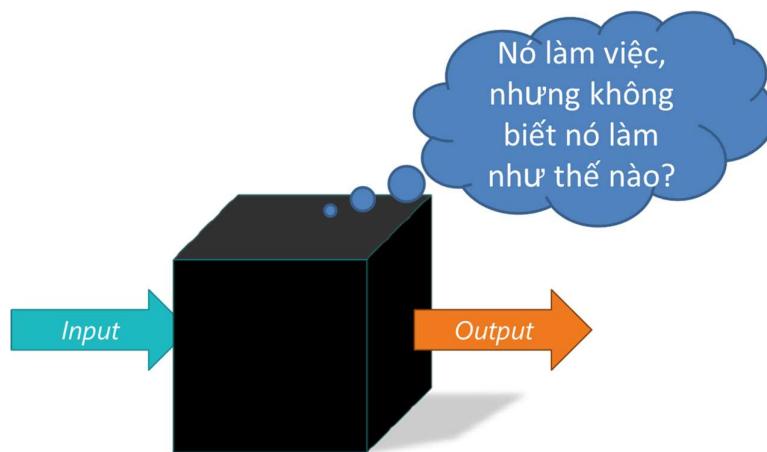
- Một đối tượng có hai khung nhìn:

- Bên trong: chi tiết về các thuộc tính và các phương thức của lớp tương ứng với đối tượng.
- Bên ngoài: các dịch vụ mà một đối tượng có thể cung cấp và cách đối tượng đó tương tác với phần còn lại của hệ thống.



##### 6.3.1.1. Đóng gói

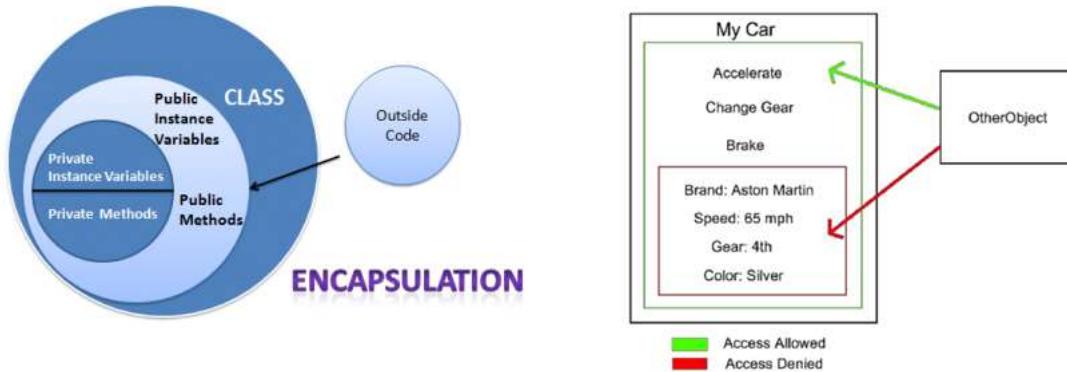
- Dữ liệu và phương thức được đóng gói trong một lớp.
- Dữ liệu được che giấu ở bên trong lớp và chỉ được truy cập và thay đổi ở các phương thức bên ngoài.
  - Một đối tượng là một thực thể được đóng gói, cung cấp tập các dịch vụ nhất định.
  - Một đối tượng được đóng gói có thể được xem như một hộp đen – các công việc bên trong là ẩn so với client.



Hình 6.16 Đóng gói đối tượng như một hộp đen

### 6.3.1.2. Che giấu dữ liệu:

- Dữ liệu được che giấu ở bên trong lớp và chỉ được truy cập và thay đổi ở các phương thức bên ngoài.
- Tránh thay đổi trái phép hoặc làm sai lệch dữ liệu.



Hình 6.17 Tính che dấu dữ liệu

### 6.3.1.3. Cơ chế che giấu dữ liệu:

- Các thành viên dữ liệu: chỉ có thể truy cập từ các phương thức bên trong lớp; chỉ định truy cập là **private** để bảo vệ dữ liệu.
  - Các đối tượng khác muốn truy nhập vào dữ liệu riêng tư này phải thông qua các phương thức **public**.
  - Các thành phần dữ liệu là **private** → để truy cập và chỉnh sửa các giá trị của dữ liệu, lớp cần phải cung cấp các dịch vụ:

**Accessor (getter):** Trả về giá trị hiện tại của một thuộc tính (dữ liệu).

**Mutator (setter):** Thay đổi giá trị của một thuộc tính.

Thường là **getX** và **setX**, trong đó X là tên thuộc tính.

### 6.3.1.4. Phương thức Get

- Các phương thức truy vấn (query method, accessor) là các phương thức dùng để hỏi về giá trị của các thành viên dữ liệu của một đối tượng
- Có nhiều loại câu hỏi truy vấn có thể:
  - truy vấn đơn giản (“**giá trị của x là bao nhiêu?**”);
  - truy vấn điều kiện (“**thành viên x có lớn hơn 100 không?**”);
  - truy vấn dẫn xuất (“**tổng giá trị của các thành viên x và y bao nhiêu?**”).

- Đặc điểm quan trọng của phương thức truy vấn là không nên thay đổi trạng thái hiện tại của đối tượng: không thay đổi giá trị của thành viên dữ liệu nào.

#### 6.3.1.5. Phương thức Set

- Các phương thức thiết lập (mutator, setter) là các phương thức dùng để thay đổi giá trị các thành viên dữ liệu.
- Ưu điểm của việc sử dụng các phương thức **setter** là có thể sử dụng các phương thức **setter** để đảm bảo tính hợp lệ của các thành phần dữ liệu: Kiểm tra giá trị đầu vào trước khi gán vào các thuộc tính.

```

public class Time {
    private int hour;
    private int minute;
    private int second;
}

public Time () { setTime(0, 0, 0); }
public void setHour (int h) { hour = ((h >= 0 && h < 24) ? h : 0; }
public void setMinute (int m) {
    minute = ( ( m >= 0 && m < 60 ) ? m : 0;
}
public void setSecond (int s) {
    second = ( ( s >= 0 && s < 60 ) ? s : 0;
}
public void setTime (int h, int m, int s) {
    setHour(h);
    setMinute(m);
    setSecond(s);
}

public int getHour () { return hour; }
public int getMinute () { return minute; }
public int getSecond () { return second; }

```

**private:** không cho phép truy cập từ bên ngoài lớp. Nhưng chúng ta cần biết và cập nhật giá trị của chúng!

**set method:** public cho phép cập nhật giá trị dữ liệu của thành viên private

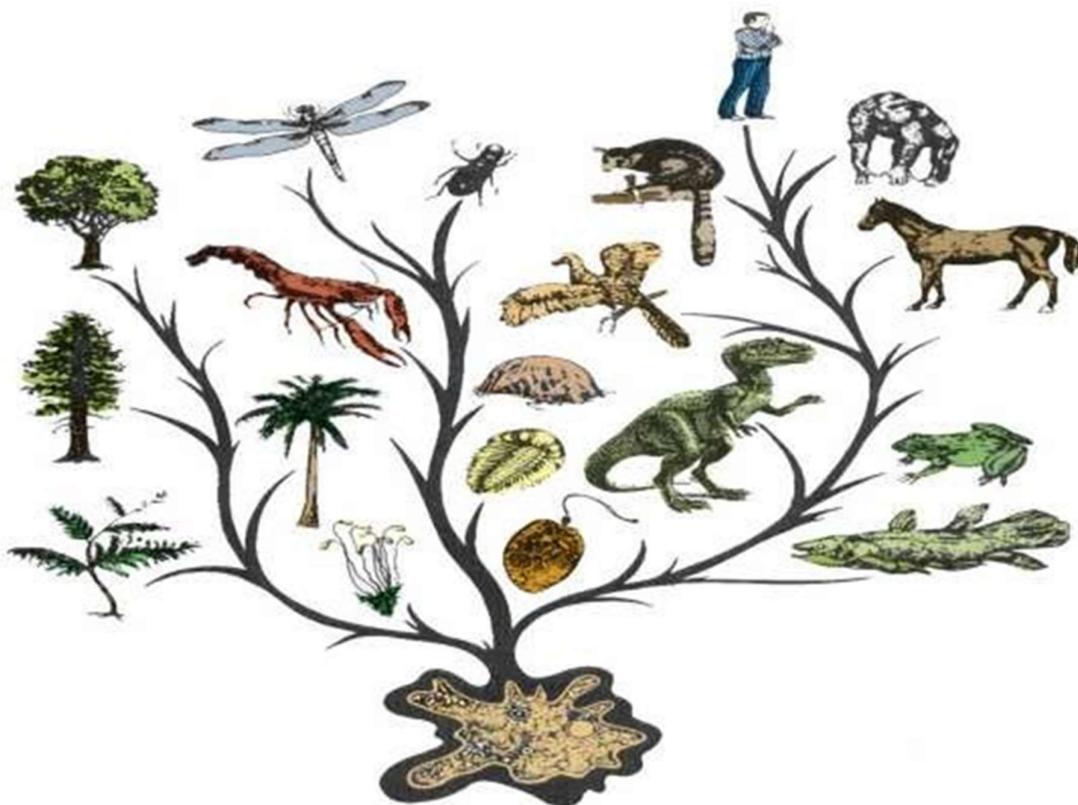
**get method:** public cho phép đọc giá trị dữ liệu của thành viên private

### 6.3.2. Tính kế thừa (Inheritance)

#### 6.3.2.1. Khái niệm kế thừa

##### Kế thừa là gì?

- Xây dựng các lớp mới có sẵn các đặc tính của lớp cũ, đồng thời chia sẻ hay mở rộng các đặc tính sẵn có

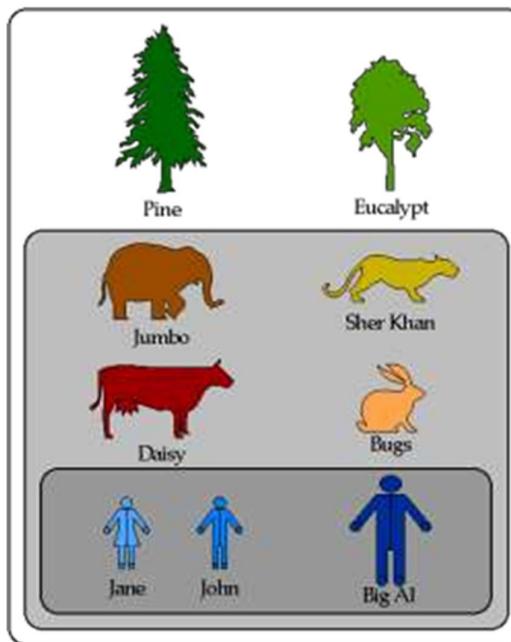


Hình 6.18 Phân lớp các đối tượng

- Phát triển lớp mới dựa trên các lớp đã có.

##### Ví dụ:

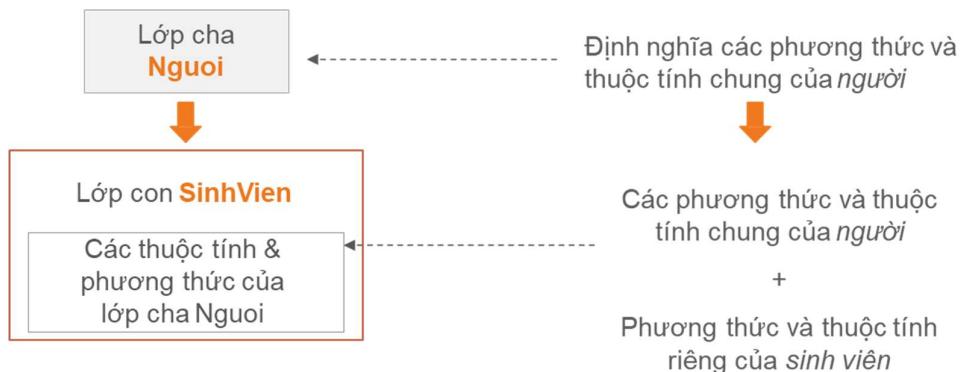
- Lớp **Người** có các thuộc tính như tên, tuổi, chiều cao, cân nặng,...; các phương thức như ăn, ngủ, chơi,...
- Lớp **Sinh Viên** thừa kế từ lớp **Người**, thừa kế được các thuộc tính tên, tuổi, chiều cao, cân nặng,...; các phương thức ăn, ngủ, chơi,...
  - Bổ sung thêm các thuộc tính như: mã số sinh viên, số tín chỉ tích lũy,...; các phương thức như: tham dự lớp học, thi,...
- Chính là nguyên lý phân cấp trong trừu tượng hóa.



Hình 6.19 Sơ đồ phân cấp

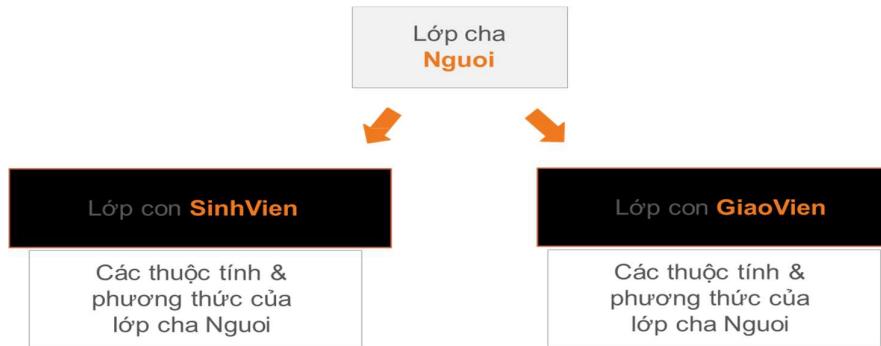
- Khái niệm:
  - Lớp cũ: Lớp cha (parent, superclass), lớp cơ sở (base class).
  - Lớp mới: Lớp con (child, subclass), lớp dẫn xuất (derived class).

**Ví dụ:** SinhVien thừa kế (dẫn xuất) từ lớp Nguoi.



**Mối quan hệ kế thừa:**

- Lớp con và lớp cha có tính tương đồng:
  - Lớp **SinhVien** kế thừa từ lớp **Nguoi**.
  - Một sinh viên là một người.
- Cả **GiaoVien** và **SinhVien** đều có quan hệ là (*is-a*) với lớp **Nguoi**
- Cả giáo viên và sinh viên đều có một số hành vi thông thường của con người.



– Lớp con:

- Là một loại (*is-a-kind-of*) của lớp cha.
- Kế thừa các thành phần dữ liệu và các hành vi của lớp cha.
- Chi tiết hóa cho phù hợp với mục đích sử dụng mới:

*Extension*: Thêm các thuộc tính/hành vi mới.

*Redefinition* (Overriding): chỉnh sửa lại các hành vi kế thừa từ lớp cha.

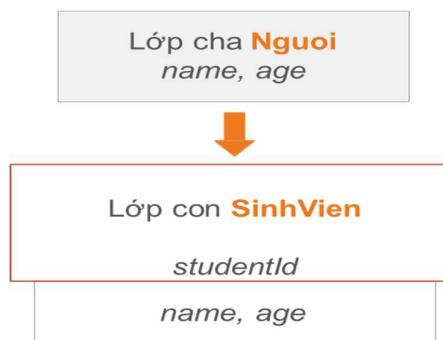
*Cú pháp (Java)*:

```
<Lớp con> extends <Lớp cha>
```

*Ví dụ:*

```
class Nguoi {  
    String name;  
  
    int age;  
}  
class SinhVien extends Nguoi {  
  
    int studentId;  
}
```

– Lớp con mở rộng các đặc tính của lớp cha.



### Bản chất kế thừa:

- Là một kỹ thuật tái sử dụng mã nguồn.
  - Tái sử dụng mã nguồn thông qua lớp.
- Ví dụ: Lớp **SinhViên** tái sử dụng được các thuộc tính như tên, tuổi... và các phương thức của lớp **Người**.

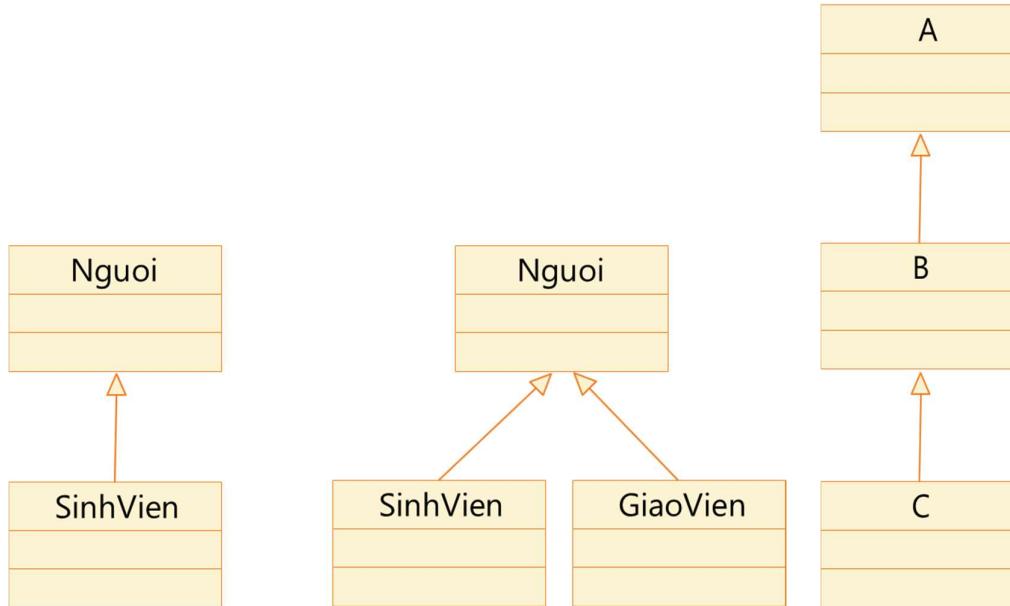
### Kế thừa và kết tập:

Kế thừa	Kết tập
<ul style="list-style-type: none"> <li>• Tái sử dụng thông qua lớp</li> </ul>	<ul style="list-style-type: none"> <li>• Tái sử dụng thông qua đối tượng</li> </ul>
<ul style="list-style-type: none"> <li>• Quan hệ: là một loại (<i>is a-kind-of</i>)</li> </ul>	<ul style="list-style-type: none"> <li>• Quan hệ: là một phần (<i>is a-part-of</i>)</li> </ul>

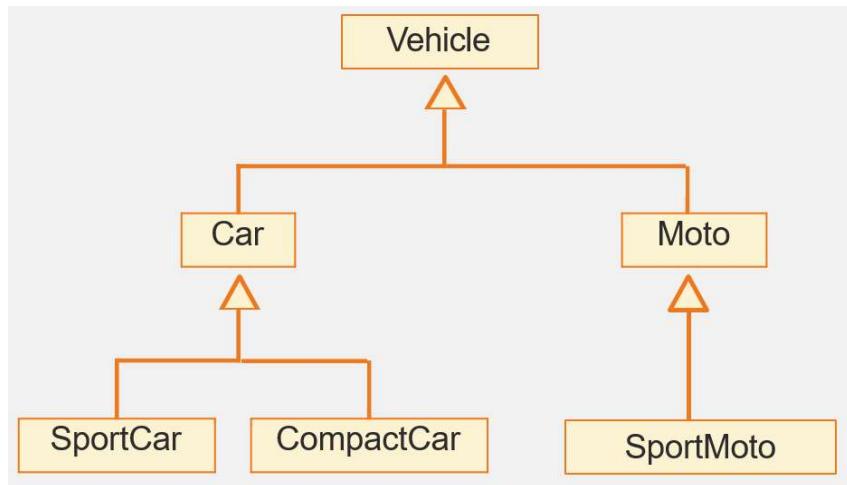
#### 6.3.2.2. Biểu diễn quan hệ kế thừa trong biểu đồ lớp

##### Cây phân cấp kế thừa:

- Dùng mũi tên với tam giác rỗng ở đầu.



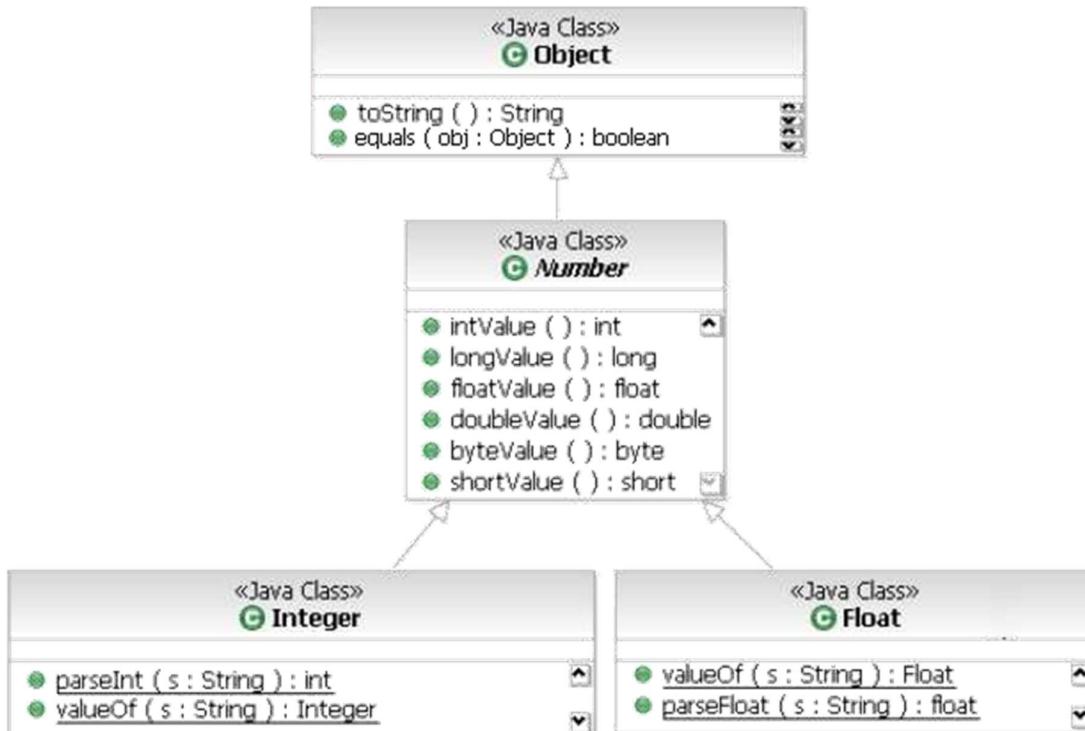
- Cấu trúc phân cấp hình cây, biểu diễn mối quan hệ kế thừa giữa các lớp.
  - Dẫn xuất trực tiếp: B dẫn xuất trực tiếp từ A.
  - Dẫn xuất gián tiếp: C dẫn xuất gián tiếp từ A.
- Các lớp con có cùng lớp cha gọi là các lớp anh/chị em (siblings).



### Lớp Object:

- Lớp **Object** là lớp gốc trên cùng của tất cả các cây phân cấp kế thừa
  - Nếu một lớp không được định nghĩa là lớp con của một lớp khác thì mặc định nó là lớp con trực tiếp của lớp **Object**.
- Được định nghĩa trong **package** chuẩn **java.lang**.
- Chứa một số phương thức hữu ích kế thừa lại cho tất cả các lớp.

Ví dụ: **toString()**, **equals()**...



Hình 6.20 Ví dụ cây phân lớp Lớp **Object**

### Nguyên lý kế thừa:

- Lớp con có thể thừa kế được gì từ lớp cha?
  - Kế thừa được các thành viên được khai báo là **public** và **protected** của lớp cha.
  - Không kế thừa được các thành viên **private**.

- Thành viên **protected** trong lớp cha được truy cập trong:
  - Các thành viên lớp cha;
  - Các thành viên lớp con;
  - Các thành viên các lớp cùng thuộc một **package** với lớp cha.

Bảng 6-2 Nguyên lý kế thừa trong Java

	<b>public</b>	<b>protected</b>	<b>Mặc định</b>	<b>private</b>
Cùng lớp	✓	✓	✓	✓
Lớp bất kỳ cùng gói	✓	✓	✓	✗
Lớp con khác gói	✓	✓	✗	✗
Lớp bất kỳ khác gói	✓	✗	✗	✗

- Các phương thức không được phép kế thừa:
  - Các phương thức khởi tạo và hủy:
    - Làm nhiệm vụ khởi tạo và gỡ bỏ các đối tượng.
    - Chúng chỉ biết cách làm việc với từng lớp cụ thể.
  - Toán tử gán “==”, làm nhiệm vụ giống như phương thức khởi tạo.

### Ví dụ 1:

```
public class TuGiac {
    protected Diem d1, d2, d3, d4;
    public void setD1(Diem _d1) { d1=_d1; }
    public Diem getD1(){ return d1; }
    public void printTuGiac(){...}
}
```

```

public class HinhVuong extends TuGiac {
    public HinhVuong() {
        d1 = new Diem(0,0);   d2 = new Diem(0,1);
        d3 = new Diem(1,0);   d4 = new Diem(1,1);
    }
    public class Test {
        public static void main(String args[]){
            HinhVuong hv = new HinhVuong();
            hv.printTuGiac();←
        }
    }
}

```

Sử dụng các thành phần *protected* của lớp cha trong lớp con.

Gọi phương thức *public* của lớp cha trong đối tượng lớp con

Ví dụ 2:

```

import java.util.Date;
class Person {
    private String name;
    private Date bithday;
}

class Employee extends Person {
    private double salary;
    public boolean setSalary(double sal) {
        salary = sal;
        return true;
    }
    public String getDetail() {
        String s = name + ", " + salary;
        return s;
    }
}

public class Test{
    public static void main(String args[]){
        Employee e = new Employee();
        e.getDetail();
    }
}

```

```

classDiagram
    class Person {
        -name: String
        -birthday: Date
        +setName()
        +setBirthday()
    }
    class Employee {
        -salary: double
        +setSalary()
        +getDetails()
    }
    Person <|-- Employee

```

Person

- name: String
- birthday: Date
- + setName()
- + setBirthday()

Employee

- salary: double
- + setSalary()
- + getDetails()

ERROR: name has private access in Person

*Ví dụ 3:* Cùng gói.

```
import java.util.Date; //Lop con cung goi
package abc;
public class Person {
    Date birthday;
    String name;
}
package abc.Person;
public class Employee extends Person {
    double salary;
    public String getDetail() {
        String s = name + "," + salary;
        return s;
    }
}
```

*Ví dụ 3:* Khác gói.

```
import java.util.Date; //Lop con khac goi
package abc;
public class Person {
    protected Date birthday;
    protected String name;
}
package abc.Person;
public class Employee extends Person {
    double salary;
    public String getDetail() {
        String s = name + "," + salary;
        return s;
    }
}
```

### 6.3.2.3. Khởi tạo và hủy bỏ đối tượng trong kế thừa

– Khởi tạo đối tượng:

- Lớp cha được khởi tạo trước lớp con.
- Các phương thức khởi tạo của lớp con luôn gọi phương thức khởi tạo của lớp cha ở câu lệnh đầu tiên.

○ Tự động gọi (ngầm định - implicit): khi lớp cha *có* phương thức khởi tạo mặc định (hoặc ngầm định).

○ Gọi trực tiếp (tường minh - explicit): sử dụng từ khóa **super**.

– Hủy bỏ đối tượng: Ngược lại so với khởi tạo đối tượng.

**Gọi phương thức của lớp cha:**

– Tái sử dụng các đoạn mã của lớp cha trong lớp con.

– Gọi phương thức khởi tạo:

**super(danh sách tham số);**

- Bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định.
- Phải được khai báo **tại dòng lệnh đầu tiên** trong phương thức khởi tạo của lớp con.

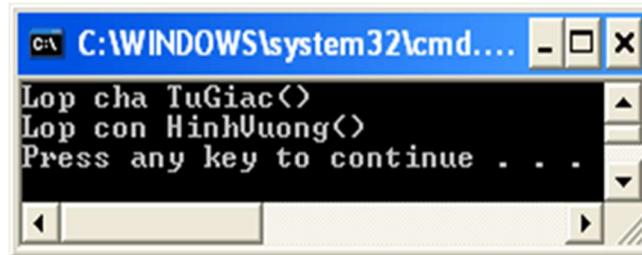
– Gọi các phương thức của lớp cha.

**super.tênPt(danh sách tham số);**

**Ví dụ 1:**

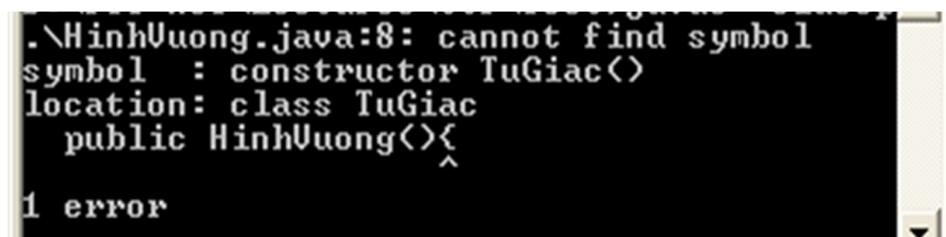
```
public class TuGiac {  
    protected Diem d1, d2;      protected Diem d3, d4;  
    public TuGiac() {  
        System.out.println("Lop cha TuGiac()");  
    }  
}  
public class HinhVuong extends TuGiac {  
    public HinhVuong() {  
        // Tu dong goi TuGiac()  
        System.out.println("Lop con HinhVuong()");  
    }  
}
```

```
public class Test {  
    public static void main(String arg[]) {  
        HinhVuong hv = new HinhVuong();  
    }  
}
```



Ví dụ 2:

```
public class TuGiac {  
    protected Diem d1, d2;  
    protected Diem d3, d4;  
    public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {  
        System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
public class HinhVuong extends TuGiac {  
    public HinhVuong() {  
        System.out.println("Lop con HinhVuong()");  
    }  
}  
public class Test {  
    public static void main(String arg[]) {  
        HinhVuong hv = new HinhVuong();  
    }  
}
```

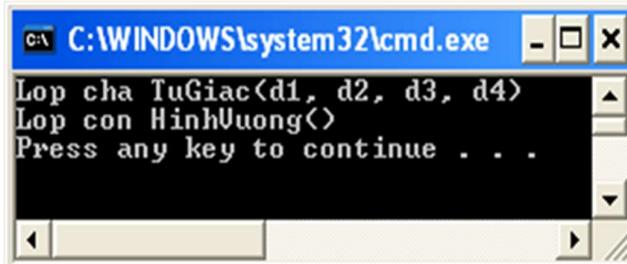


The screenshot shows a terminal window with the following error message:

```
.\\HinhVuong.java:8: cannot find symbol
symbol  : constructor TuGiac()
location: class TuGiac
    public HinhVuong(){}
               ^
1 error
```

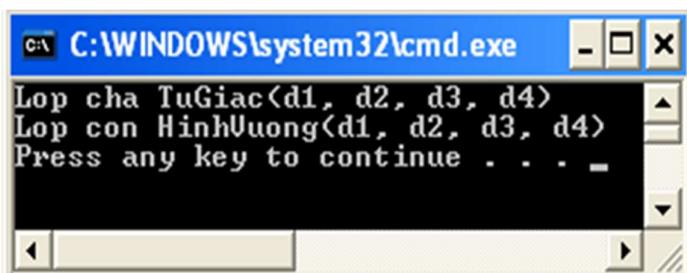
Ví dụ 3:

```
public class TuGiac {
    protected Diem d1, d2, d3, d4;
    public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {
        System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");
        this.d1 = d1; this.d2 = d2;
        this.d3 = d3; this.d4 = d4;
    }
}
public class HinhVuong extends TuGiac {
    public HinhVuong() {
        super(new Diem(0,0), new Diem(0,1),
              new Diem(1,1),new Diem(1,0));
        System.out.println("Lop con HinhVuong()");
    }
}
public class Test {
    public static void main(String arg[]) {
        HinhVuong hv = new HinhVuong();
    }
}
```



**Ví dụ 4:**

```
public class TuGiac {  
    protected Diem d1, d2, d3, d4;  
    public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {  
        System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong(Diem d1, Diem d2, Diem d3, Diem d4) {  
        super(d1, d2, d3, d4);  
        System.out.println("Lop con HinhVuong(d1, d2, d3, d4)");  
    }  
}  
  
public class Test {  
    public static void main(String arg[]) {  
        HinhVuong hv = new HinhVuong(  
            new Diem(0,0), new Diem(0,1),new Diem(1,1),new Diem(1,0)  
        );  
    }  
}
```



### 6.3.3. Một số kỹ thuật kế thừa

#### 6.3.3.1. Định nghĩa lại / ghi đè (Redefine/Overiding)

- Lớp con có thể định nghĩa phương thức trùng tên với phương thức trong lớp cha.

Ví dụ:

```
class Shape {  
    protected String name;  
    Shape(String n) { name = n; }  
    public String getName() { return name; }  
    public float calculateArea() { return 0.0f; }  
}  
  
class Circle extends Shape {  
    private int radius;  
    Circle(String n, int r){  
        super(n);  
        radius = r;  
    }  
  
    public float calculateArea() {  
        float area = (float) (3.14 * radius * radius);  
        return area;  
    }  
}  
  
class Square extends Shape {  
    private int side;  
    Square(String n, int s) {  
        super(n);  
        side = s;  
    }  
  
    public float calculateArea() {  
        float area = (float) side * side;  
        return area;  
    }  
}
```

Thêm lớp **Triangle**

```
class Triangle extends Shape {  
    private int base, height;  
    Triangle(String n, int b, int h) {  
        super(n);  
        base = b;  
        height = h;  
    }  
    public float calculateArea() {  
        float area = 0.5f * base * height;  
        return area;  
    }  
}
```

**this** và **super**

```
package abc;  
public class Person {  
    protected String name;  
    protected int age;  
    public String getDetail() {  
        String s = name + "," + age;  
        return s;  
    }  
}
```

```
import abc.Person;  
public class Employee extends Person {  
    double salary;  
    public String getDetail() {  
        String s = super.getDetail() + "," + this.salary;  
        return s;  
    }  
}
```

### 6.3.3.2. Lớp trừu tượng (Abstract class)

- Không thể thê hiện hóa (instantiate – tạo đối tượng của lớp) trực tiếp.

```
abstract class Shape {  
    protected String name;  
    Shape(String n) { name = n; }  
    public String getName() { return name; }  
    public abstract float calculateArea();  
}  
class Circle extends Shape {  
    private int radius;  
    Circle(String n, int r){  
        super(n);  
        radius = r;  
    }  
    public float calculateArea() {  
        float area = (float) (3.14 * radius * radius);  
        return area;  
    }  
}
```

```
import java.awt.Graphics;  
abstract class Action {  
    protected int x, y;  
    public void moveTo(Graphics g, int x1, int y1) {  
        erase(g);  
        x = x1; y = y1;  
        draw(g);  
    }  
    abstract public void erase(Graphics g);  
    abstract public void draw(Graphics g);  
}
```

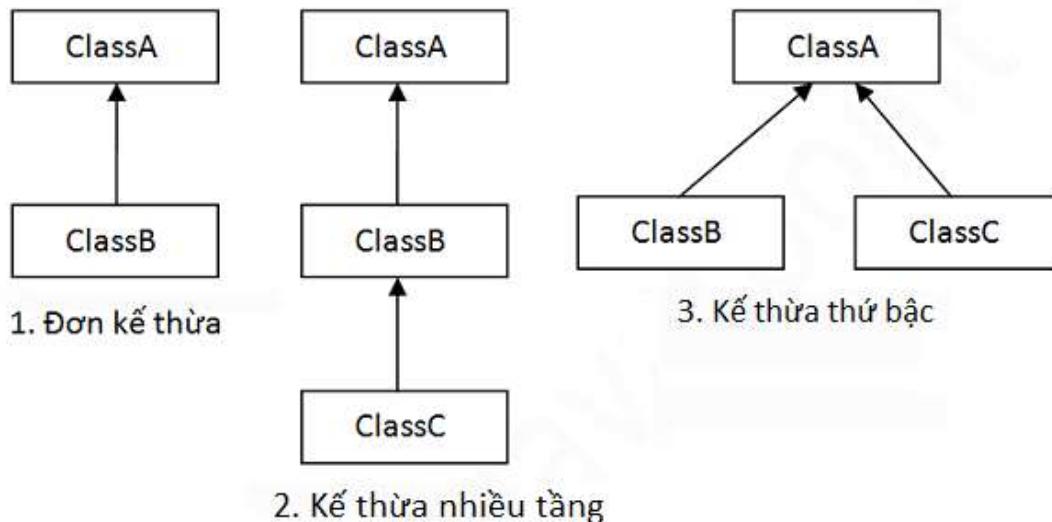
```

class Circle extends Action {
    int radius;
    public Circle(int x, int y, int r) {
        super(x, y);
        radius = r;
    }
    public void draw(Graphics g) {
        System.out.println("Draw circle at (" +x+ "," +y+ ")");
        g.drawOval(x-radius, y-radius, 2*radius, 2*radius);
    }
    public void erase(Graphics g) {
        System.out.println("Erase circle at (" +x+ "," +y+ ")");
    }
}

```

### 6.3.3.3. Các kiểu kế thừa trong java

- Có 3 kiểu kế thừa trong java đó là đơn kế thừa, kế thừa nhiều cấp, kế thừa thứ bậc.
- Khi một **class** được kế thừa từ nhiều **class** được gọi là đa kế thừa. Trong java, đa kế thừa chỉ được hỗ trợ thông qua **interface**.



**Ví dụ về đơn kế thừa: [TestInheritance1.java](#)**

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking...");  
    }  
}  
  
public class TestInheritance1 {  
    public static void main(String args[]) {  
        Dog d = new Dog();  
        d.bark();  
        d.eat();  
    }  
}
```

Kết quả:

```
barking...  
eating...
```

**Ví dụ về kế thừa nhiều cấp: [TestInheritance2.java](#)**

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking...");  
    }  
}
```

```
class BabyDog extends Dog {  
    void weep() {  
        System.out.println("weeping...");  
    }  
}  
public class TestInheritance2 {  
    public static void main(String args[]) {  
        BabyDog d = new BabyDog();  
        d.weep();  
        d.bark();  
        d.eat();  
    }  
}
```

Kết quả:

```
weeping...  
barking...  
eating...
```

Ví dụ về kế thừa thứ bậc: *TestInheritance3.java*

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking...");  
    }  
}  
class Cat extends Animal {  
    void meow() {  
        System.out.println("meowing...");  
    }  
}
```

```
public class TestInheritance3 {  
    public static void main(String args[]) {  
        Cat c = new Cat();  
        c.meow();  
        c.eat();  
        // c.bark(); // compile error  
    }  
}
```

Kết quả:

```
meowing...  
eating...
```

**Tại sao đa kế thừa không được hỗ trợ trong java?**

- Để giảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ, đa kế thừa không được hỗ trợ trong java.

**Ví dụ:** Có 3 lớp **A**, **B**, **C**, trong đó lớp **C** kế thừa từ các lớp **A** và **B**. Nếu các lớp **A** và **B** có phương thức giống nhau và gọi nó từ đối tượng của lớp con **C**, như vậy khó có thể xác định được việc gọi phương thức của lớp **A** hay **B**.

```
class A {  
    void msg() { System.out.println("Hello!"); }  
}  
class B {  
    void msg() { System.out.println("Welcome!"); }  
}  
public class C extends A,B {  
    public static void main(String args[]) {  
        C obj = new C();  
        obj.msg();  
    }  
}
```

Kết quả:

```
Compile Time Error
```

#### 6.3.3.4. Giao diện (Interface)

- Một **interface** trong Java là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng (**abstract**). **Interface** là một kỹ thuật để thu được tính trừu tượng hoàn toàn và hỗ trợ đa kế thừa trong Java. **Interface** trong Java cũng biểu diễn mối quan hệ **is-a**. Nó không thể được khởi tạo giống như lớp trừu tượng.
- Nói cách khác, các trường của **interface** là **public**, **static** và **final** theo mặc định và các phương thức là **public** và **abstract**.
- Một **interface** trong Java là một tập hợp các phương thức trừu tượng (**abstract**). Một **class** triển khai một **interface**, do đó kế thừa các phương thức **abstract** của **interface**.
  - Một **interface** không phải là một lớp. Viết một **interface** giống như viết một lớp, nhưng chúng có 2 định nghĩa khác nhau. Một lớp mô tả các thuộc tính và hành vi của một đối tượng. Một **interface** chứa các hành vi mà một **class** triển khai.
    - Trừ khi một lớp triển khai **interface** là lớp trừu tượng **abstract**, còn lại tất cả các phương thức của **interface** cần được định nghĩa trong **class**.
    - Một **interface** tương tự với một **class** bởi những điểm sau đây:
      - Một **interface** được viết trong một file với định dạng **\*.java**, với tên của **Interface** giống tên của file.
      - **Bytecode** của **interface** được lưu trong file có định dạng **\*.class**.
      - Khai báo **interface** trong một **package**, những file **bytecode** tương ứng cũng có cấu trúc thư mục có cùng tên **package**.
    - Một **interface** khác với một **class** ở một số điểm sau đây:
      - Không thể khởi tạo một **interface**.
      - Một **interface** không chứa bất cứ **Constructor** nào.
      - Tất cả các phương thức của **interface** đều là **abstract**.
      - Một **interface** không thể chứa một trường nào, trừ các trường vừa **static** và **final**.
        - Một **interface** không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
        - Một **interface** có thể kế thừa từ nhiều **interface** khác.

*Ví dụ đơn giản về interface trong Java*

- Trong ví dụ này, **Printable interface** chỉ có một phương thức, trình triển khai của nó được cung cấp bởi lớp **A**.

```
interface Printable {
    void print();
}

class A implements Printable {
    public void print() {
        System.out.println("Hello world!");
    }

    public static void main(String args[]){
        A obj = new A();
        obj.print();
    }
}
```

- Khi ghi đè các phương thức được định nghĩa trong **interface**, có các quy tắc:
  - Các **checked exception** không nên được khai báo trong phương thức **implements**, thay vào đó nó nên được khai báo trong phương thức **interface** hoặc các lớp phụ được khai báo bởi phương thức **interface**.
  - **Signature** (chữ ký) của phương thức **Interface** và kiểu trả về nên được duy trì khi ghi đè phương thức (**overriding method**).
  - Một lớp triển khai chính nó có thể là **abstract** và vì thế các phương thức **interface** không cần được triển khai.
- Khi triển khai **interface**, có vài quy tắc sau:
  - Một lớp chỉ có thể kế thừa một lớp khác, nhưng có thể triển khai một hoặc nhiều **interface** tại một thời điểm.
  - Một **interface** có thể kế thừa từ một **interface** khác, tương tự cách một lớp có thể kế thừa lớp khác.

### **Đa kế thừa trong Java bởi Interface**

- Nếu một lớp triển khai đa kế thừa, hoặc một **interface** kế thừa từ nhiều **interface** thì đó là đa kế thừa.

```
interface Printable { void print(); }

interface Showable { void show(); }

class A implements Printable, Showable {
    public void print() { System.out.println("Hello!"); }
    public void show() { System.out.println("Welcome!"); }
    public static void main(String args[]){
        A obj = new A();
        obj.print();
        obj.show();
    }
}
```

– Tại sao đa kế thừa không được hỗ trợ thông qua lớp trong Java nhưng là có thể bởi *interface*?

- Vì tránh tính lưỡng nghĩa khi trình triển khai được cung cấp bởi lớp *implementation*.

Ví dụ:

```
interface Printable { void print(); }

interface Showable { void print(); }

class TestTneterface implements Printable, Showable{
    public void print() {
        System.out.println("Hello");
    }
    public static void main(String args[]) {
        TestTneterface obj = new TestTneterface();
        obj.print();
    }
}
```

Trong ví dụ trên, *Printable* và *Showable interface* có cùng các phương thức nhưng trình triển khai của nó được cung cấp bởi lớp *TestTneterface*, vì thế không có tính lưỡng nghĩa ở đây.

### Kế thừa interface trong Java

- Một lớp triển khai **interface**, nhưng một **interface** kế thừa từ **interface** khác.

```
interface Printable { void print(); }

interface Showable extends Printable { void show(); }

class Testinterface implements Showable {

    public void print() { System.out.println("Hello!"); }

    public void show() { System.out.println("Welcome!"); }

    public static void main(String args[]){
        Testinterface obj = new Testinterface();
        obj.print();
        obj.show();
    }
}
```

### Sự khác nhau giữa Abstract class và Interface

- Abstract class** và **interface** đều được sử dụng để có được sự trừu tượng mà ở đó chúng ta có thể khai báo các phương thức trừu tượng. Nhưng có một vài sự khác nhau giữa **abstract class** và **interface**:

Abstract class	Interface
<b>Abstract class</b> có phương thức <b>abstract</b> (không có thân phương thức) và phương thức <b>non-abstract</b> (có thân phương thức).	<b>Interface</b> chỉ có phương thức <b>abstract</b> . Từ java 8, nó có thêm các phương thức <b>default</b> và <b>static</b> .
<b>Abstract class</b> không hỗ trợ <b>đa kế thừa</b> .	<b>Interface</b> có hỗ trợ <b>đa kế thừa</b> .
<b>Abstract class</b> có các biến <b>final</b> , <b>non-final</b> , <b>static</b> và <b>non-static</b> .	<b>Interface</b> chỉ có các biến <b>static</b> và <b>final</b> .
<b>Abstract class</b> có thể cung cấp nội dung cài đặt cho phương thức của <b>interface</b> .	<b>Interface</b> không thể cung cấp nội dung cài đặt cho phương thức của <b>abstract class</b> .
Tùy khóa <b>abstract</b> được sử dụng để khai báo <b>abstract class</b> .	Tùy khóa <b>interface</b> được sử dụng để khai báo <b>interface</b> .
<b>Ví dụ:</b>	<b>Ví dụ:</b>

<pre>public abstract class Shape {     public abstract void draw(); }</pre>	<pre>public interface Drawable {     void draw(); }</pre>
---	---

- Đơn giản để hiểu, đó là *Abstract class* có được trùu tượng 1 phần (0 tới 100%), còn *Interface* có được trùu tượng toàn phần (100%).

*Ví dụ:* kết hợp giữa *Abstract class* và *Interface*.

```
//tạo interface có 4 phương thức  
  
interface A {  
    void a();  
    abstract void b();  
    public void c();  
    public abstract void d();  
}  
  
//tạo abstract class cài đặt cho một phương thức của interface A  
abstract class B implements A {  
    public void c() {  
        System.out.println("Day la c");  
    }  
}  
  
//tạo subclass của abstract class B, cài đặt các phương thức còn lại  
class M extends B {  
    public void a() {  
        System.out.println("Day la a");  
    }  
    public void b() {  
        System.out.println("Day la b");  
    }  
    public void d() {  
        System.out.println("Day la d");  
    }  
}
```

```
//tạo lớp Test để gọi các phương thức của interface A
public class Test {
    public static void main(String args[]) {
        A a = new M();
        a.a();
        a.b();
        a.c();
        a.d();
    }
}
```

Kết quả:

```
Day la a
Day la b
Day la c
Day la d
```

#### 6.3.4. Tính đa hình (polymorphism)

##### 6.3.4.1. Chuyển đổi kiểu dữ liệu đối tượng (Up-casting và down-casting)

*Chuyển đổi kiểu dữ liệu nguyên thủy*

- Java tự động chuyển đổi kiểu khi:
  - Kiểu dữ liệu tương thích.
  - Chuyển đổi từ kiểu hẹp hơn sang kiểu rộng hơn.

```
int i;
double d = i;
```

- Phải ép kiểu khi:
  - Kiểu dữ liệu tương thích.
  - Chuyển đổi từ kiểu rộng hơn sang kiểu hẹp hơn.

```
int i;
byte b = i;
byte b = (byte)i;
```

### Chuyển đổi kiểu dữ liệu tham chiếu

- Kiểu dữ liệu tham chiếu có thể được chuyển đổi kiểu khi:

- Kiểu dữ liệu tham chiếu (lớp) tương thích:

*Nằm trên cùng một cây phân cấp thừa.*

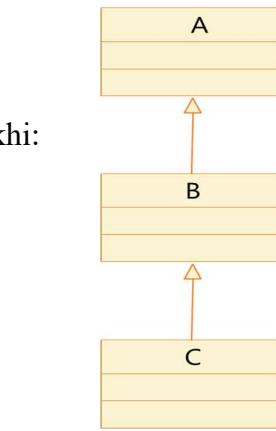
- Hai cách chuyển đổi: Up-casting và Down-casting

#### Up-casting

- Đi lên trên cây phân cấp thừa (moving up the inheritance hierarchy)
- Up casting là khả năng nhìn nhận đối tượng thuộc lớp dẫn xuất như là một đối tượng thuộc lớp cơ sở.
- Tự động chuyển đổi kiểu.

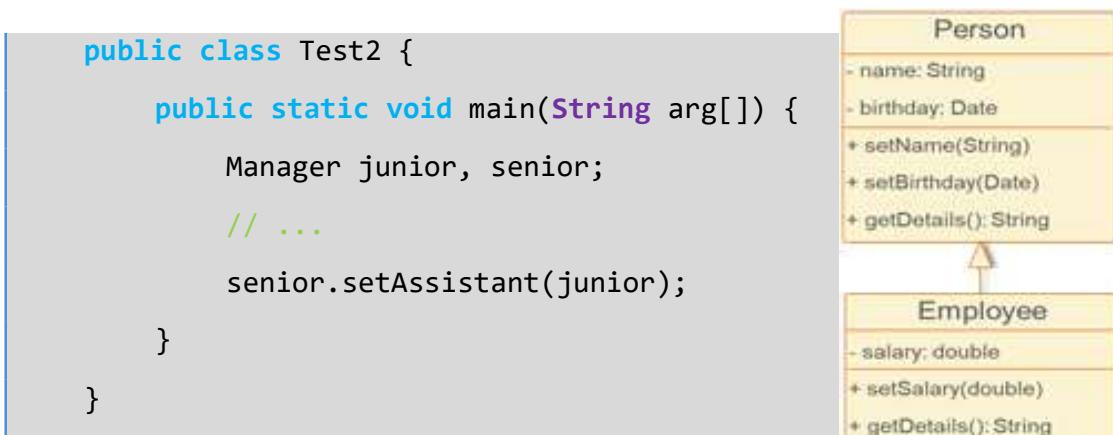
#### Ví dụ 1:

```
public class Test1 {
    public static void main(String arg[]) {
        Employee e = new Employee();
        Person p;
        p = e;
        p.setName("Alan Kay");
        p.setSalary(70000);
        // compile error
    }
}
```



#### Ví dụ 2:

```
class Manager extends Employee {
    Employee assistant;
    ...
    public void setAssistant(Employee e) {
        assistant = e;
    }
    ...
}
```



*Ví dụ 3:*

```

public class Test3 {
    String static teamInfo(Person p1, Person p2) {
        return "Leader: " + p1.getName() +
               ", member: " + p2.getName();
    }

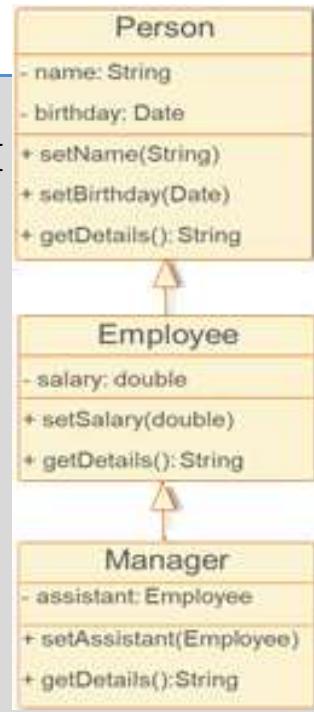
    public static void main(String arg[]) {
        Employee e1, e2;
        Manager m1, m2;
        //...
        System.out.println(teamInfo(e1, e2));
        System.out.println(teamInfo(m1, m2));
        System.out.println(teamInfo(m1, e2));
    }
}
  
```

### ***Down-casting***

- Di xuống cây phân cấp thừa kế (move back down the inheritance hierarchy).
- ***Down casting*** là khả năng nhìn nhận một đối tượng thuộc lớp cơ sở như một đối tượng thuộc lớp dẫn xuất.
- Không tự động chuyển đổi kiểu:  
→ Phải ép kiểu.

Ví dụ:

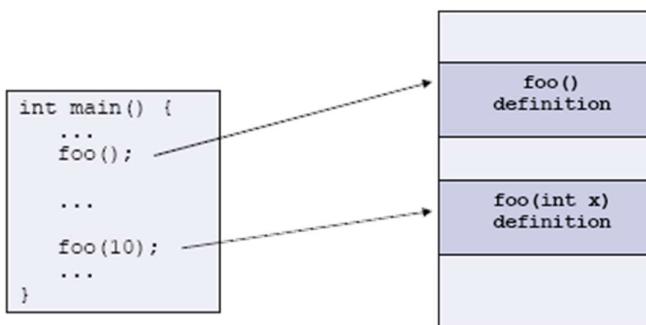
```
public class Test {
    public static void main(String arg[]) {
        Employee e = new Employee();
        Person p = e; // up casting
        Employee ee = (Employee) p;
        // down casting
        Manager m = (Manager) ee;
        // run-time error
        Person p2 = new Manager();
        Employee e2 = (Employee) p2;
    }
}
```



#### 6.3.4.2. Liên kết tĩnh và liên kết động (Static binding và Dynamic binding)

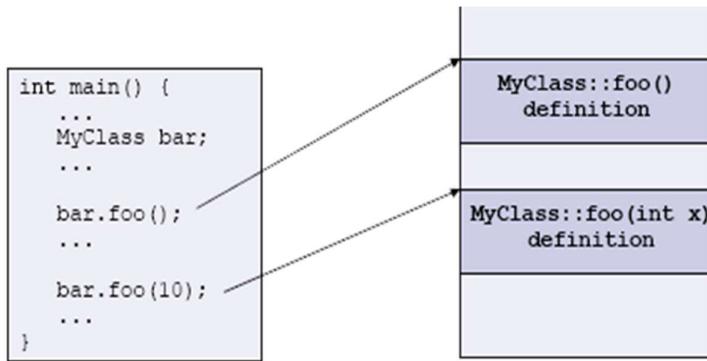
##### Liên kết lời gọi hàm

- Liên kết lời gọi hàm (*function call binding*) là quy trình xác định khối mã hàm cần chạy khi một lời gọi hàm được thực hiện.
  - C: đơn giản vì mỗi hàm có duy nhất một tên
  - C++: chồng phương thức, phân tích chữ ký kiểm tra danh sách tham số.



##### Liên kết lời gọi phương thức trong ngôn ngữ hướng đối tượng (OOP)

- Đối với các lớp độc lập (không thuộc cây thừa kế nào), quy trình này gần như không khác với *function call binding*:
  - So sánh tên phương thức, danh sách tham số để tìm định nghĩa tương ứng.
  - Một trong số các tham số là tham số ẩn: con trỏ **this**.



### Liên kết tĩnh

- Liên kết tại thời điểm biên dịch:
  - Early Binding/Compile-time Binding.
  - Lời gọi phương thức được quyết định khi biên dịch, do đó chỉ có một phiên bản của phương thức được thực hiện.
    - Nếu có lỗi thì sẽ có lỗi biên dịch.
    - Ưu điểm về tốc độ.
- **C/C++ function call binding**, và **C++ method binding** cơ bản đều là ví dụ của liên kết tĩnh (**static function call binding**).

### Liên kết động

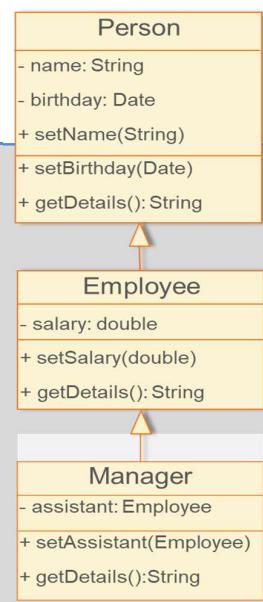
- Lời gọi phương thức được quyết định khi thực hiện (run-time):
  - Late binding/Run-time binding.
  - Phiên bản của phương thức phù hợp với đối tượng được gọi.
  - Java mặc định sử dụng liên kết động.

### Ví dụ:

```

public class Test {
    public static void main(String arg[]){
        Person p = new Person();
        // ...
        Employee e = new Employee();
        // ...
        Manager m = new Manager();
        // ...
    }
}

```



```

        Person pArr[] = {p, e, m};

        for (int i=0; i< pArr.length; i++) {
            System.out.println(pArr[i].getDetail());
        }
    }
}

```

### 6.3.4.3. Đa hình (Polymorphism)

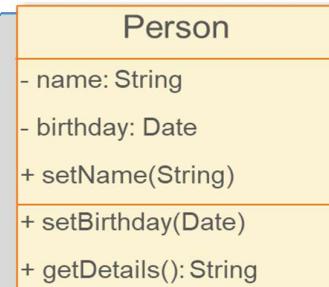
- Polymorphism: Nhiều hình thức thực hiện, nhiều kiểu tồn tại.
- Đa hình trong lập trình:
  - Đa hình phương thức:
    - phương thức trùng tên, phân biệt bởi **chữ ký**.
  - Đa hình đối tượng:
    - Nhìn nhận đối tượng theo nhiều kiểu khác nhau.
    - Các đối tượng khác nhau cùng đáp ứng chung danh sách các thông điệp có giải nghĩa thông điệp theo cách thức khác nhau.

*Ví dụ 1:*

```

Person p1 = new Person();
Person p2 = new Employee();
Person p3 = new Manager();
//...
System.out.println(p1.getDetail());
System.out.println(p2.getDetail());
System.out.println(p3.getDetail());

```

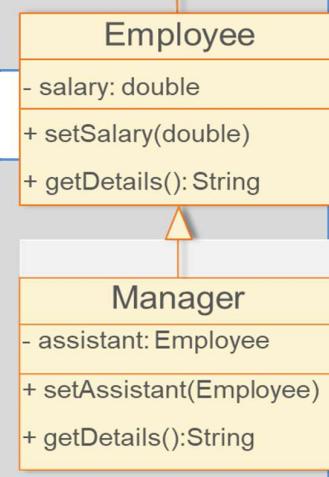


*Ví dụ 2:*

```

class EmployeeList {
    Employee list[];
    //...
    public void add(Employee e) {...}
}

```



```
public void print() {  
    for (int i=0; i<list.length; i++) {  
        System.out.println(list[i].getDetail());  
    }  
}  
...  
EmployeeList list = new EmployeeList();  
Employee e1;  
Manager m1;  
...  
list.add(e1);  
list.add(m1);  
list.print();
```

#### 6.3.4.4. Toán tử instanceof

- Toán tử *instanceof* trong java được sử dụng để kiểm tra một đối tượng có phải là thẻ hiển của một kiểu dữ liệu cụ thể không (lớp, lớp con, interface).
- *instanceof* trong java được gọi là toán tử so sánh kiểu vì nó so sánh thẻ hiện với kiểu dữ liệu. Nó trả về giá trị *boolean* là *true* hoặc *false*. Nếu bạn dùng toán tử *instanceof* với bất kỳ biến nào mà có giá trị *null*, giá trị trả về sẽ là *false*.

Ví dụ:

```
public class Simple {  
    public static void main(String args[]) {  
        Simple s = new Simple();  
        System.out.println(s instanceof Simple); // true  
    }  
}
```

Kết quả:

```
true
```

- Một đối tượng có kiểu của lớp con thì cũng có kiểu của lớp cha.

**Ví dụ**, nếu **Dog** kế thừa từ **Animal** thì đối tượng của **Dog** có thể tham chiếu đến cả hai lớp **Dog** và **Animal**.

```
class Animal {}  
public class Dog extends Animal { // Dog inherits Animal  
    public static void main(String args[]) {  
        Dog dog = new Dog();  
        System.out.println(dog instanceof Animal); // true  
    }  
}
```

Kết quả:

```
true
```

- Nếu sử dụng toán tử **instanceof** với biến có kiểu bất kỳ có giá trị **null** thì giá trị trả về luôn là **false**.

**Ví dụ**:

```
public class Dog {  
    public static void main(String args[]) {  
        Dog d = null;  
        System.out.println(d instanceof Dog); // false  
    }  
}
```

Kết quả:

```
false
```

## BÀI TẬP CHƯƠNG 6

Bài 6-1. Hãy thiết kế lớp **Rectangle**, biểu diễn hình chữ nhật bao gồm:

- Hai data field **width** và **height** có kiểu là **double** xác định chiều rộng và chiều cao của hình chữ nhật, gán giá trị mặc định là 1.0 cho cả hai.
- Một phương thức khởi tạo mặc định hình chữ nhật (constructor).
- Một phương thức khởi tạo hình chữ nhật với 2 tham số **width** và **height**.
- Một phương thức **getArea()** trả về diện tích hình chữ nhật.
- Một phương thức **getPerimeter()** trả về chu vi hình chữ nhật.

Viết chương trình tạo 2 đối tượng Rectangle, một với **width = 15** và **height = 40**, một với **width = 12.5** và **height = 25.2**. Hiển thị chiều rộng (width), chiều dài (height), diện tích (area), và chu vi (perimeter) của mỗi hình chữ nhật.

Bài 6-2. Hãy thiết kế lớp **Account**, bao gồm:

- Một **private int** data field **id** cho tài khoản (default 0).
- Một **private double** data field **balance** cho số dư tài toán (default 0).
- Một **private double** data field **annualInterestRate** xác định lãi suất hàng năm (default 0). Giả sử tất cả các tài khoản có lãi suất như nhau.
- Một **private Date** data field **dateCreated** xác định ngày tạo tài khoản (lấy ngày hệ thống).
- Một **constructor** để tạo tài khoản mặc định (default account).
- Một **constructor** để tạo tài khoản với **id** và **balance**.
- **Accessor** và **mutator** methods cho **id**, **balance**, và **annualInterestRate**.
- **Accessor** method cho **dateCreated**.
- Phương thức **getMonthlyInterestRate()** trả về lãi suất hàng tháng.
- Phương thức **getMonthlyInterest()** trả về lãi hàng tháng.
- Phương thức **withdraw** để rút một khoản tiền nhất định từ tài khoản.
- Phương thức **deposit** để gửi một khoản tiền nhất định vào tài khoản.

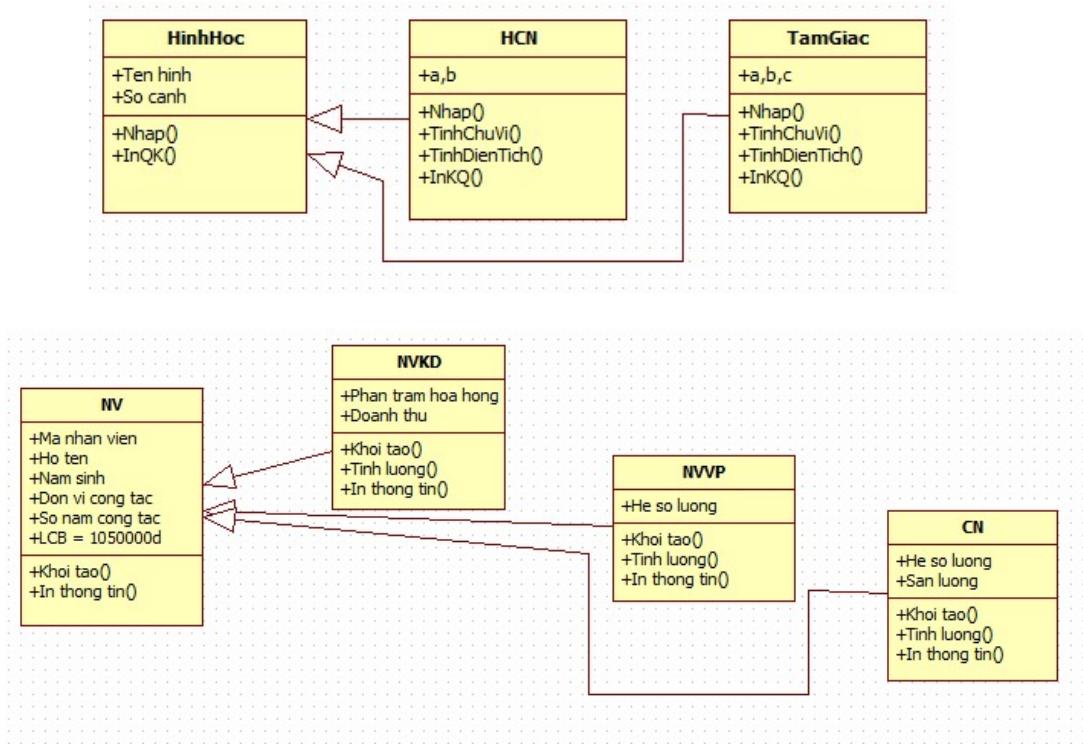
**Lưu ý:**

- Phương thức **getMonthlyInterest()** trả về lãi hàng tháng, không phải lãi suất.

- Lãi hàng tháng là ***balance \* monthlyInterestRate***, lãi suất hàng tháng ***monthlyInterestRate*** là ***annualInterestRate / 12***. Lãi suất năm ***annualInterestRate*** thì ở dạng phần trăm (%), (e.g., 6.5%, nhớ đem chia cho 100).

Viết chương trình tạo một đối tượng ***Account*** với id là 111, số dư (balance) là 20000 USD, và ***annualInterestRate*** là 6.5%. Dùng phương thức ***withdraw*** để rút số tiền 2500 USD, và phương thức ***deposit*** để gởi số tiền là 4200 USD. In số dư, tiền lãi hàng tháng, và ngày tạo tài khoản.

Bài 6-3. Xây dựng các lớp theo các mô hình:



**Yêu cầu:**

1. Khởi tạo các đối tượng của lớp con.
2. Nhập dữ liệu cho các đối tượng vừa khởi tạo.
3. Thực hiện Tính toán và in kết quả ra màn hình.

# CHƯƠNG 7 - ĐỒ HỌA VÀ XỬ LÝ SỰ KIỆN

## 7.1. Giới thiệu

Java APIs cho lập trình đồ họa: AWT (Abstract Windowing Toolkit) và Swing:

- AWT API đã được giới thiệu trong JDK 1.0. Hầu hết các thành phần AWT đã trở nên lỗi thời và cần được thay thế bởi các thành phần Swing mới hơn.
- Swing API là một bộ thư viện đồ họa toàn diện với nhiều thành phần nâng cao so AWT, đã được giới thiệu như một phần của Java Foundation Classes (JFC) sau khi phát hành JDK 1.1. JFC bao gồm Swing, Java2D, Accessibility,... JFC đã được tích hợp vào lõi Java từ JDK 1.2.

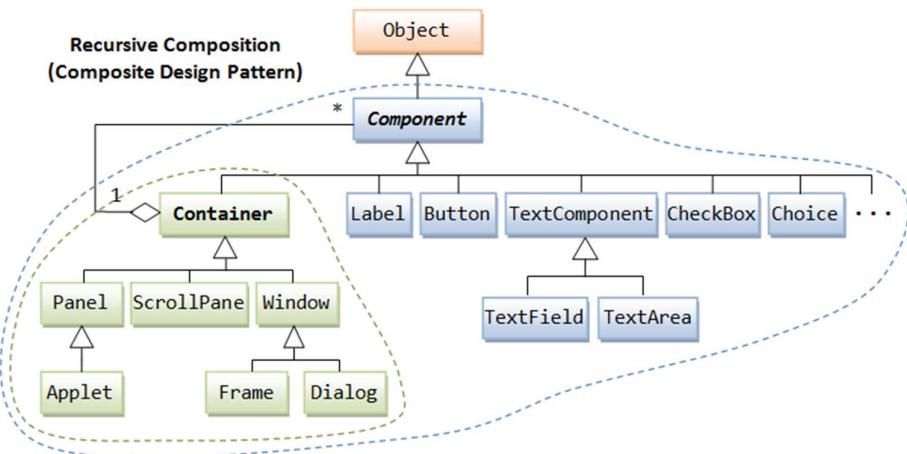
Ngoài các API đồ họa AWT/Swing được cung cấp trong JDK, cũng có nhiều Graphics APIs làm việc với Java, như SWT (Standard Widget Toolkit) của Eclipse, GWT (Google Web Toolkit) dùng cho Android, 3D Graphics API như JOGL (Java bindings for OpenGL) và Java3D.

Một số trang web tham khảo:

- <http://docs.oracle.com/javase/8/docs/api/index.html>
- <http://docs.oracle.com/javase/tutorial/uiswing>
- <http://docs.oracle.com/javase/tutorial/2d/index.html>

## 7.2. Lập trình đồ họa (GUI) với AWT

### 7.2.1. AWT Packages



Hình 7.1. Cây phân cấp AWT

AWT bao gồm 12 gói cung cấp 370 lớp để xây dựng giao diện đồ họa (GUI).

Trong đó chỉ có 2 gói “java.awt” và “java.awt.event” thường được sử dụng.

Gói java.awt gồm các lớp GUI cơ bản:

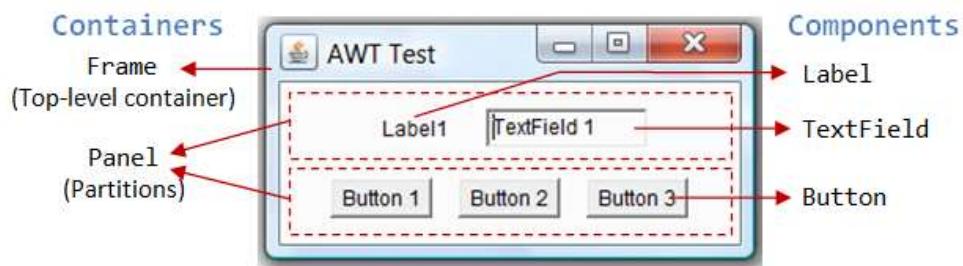
- Các lớp GUI Component, như Button, TextField, và Label.
- Các lớp GUI Container, như Frame và Panel.
- Các lớp Layout Managers, như FlowLayout, BorderLayout và GridLayout.
- Các lớp Custom graphics, như Graphics, Color và Font.

Gói java.awt.event hỗ trợ xử lý sự kiện:

- Các lớp Event, như ActionEvent, MouseEvent, KeyEvent và WindowEvent.
- Các giao diện (Interfaces) Event Listener, như ActionListener, MouseListener, KeyListener và WindowListener.
- Các lớp Event Listener Adapter, như MouseAdapter, KeyAdapter, và WindowAdapter.

AWT cung cấp một nền tảng độc lập với thiết bị để phát triển các chương trình đồ họa chạy trên tất cả các nền tảng, bao gồm Windows, Mac OS và Unixes.

#### 7.2.2. Containers và Components



Hình 7.2. Containers và Components

Có 2 loại thành phần GUI:

- Component: là các thành phần GUI cơ bản, như Button, Label, TextField,...
- Container: là các vùng chứa, như Frame và Panel, được sử dụng để chứa các components theo một bố cục (như FlowLayout hay GridLayout). Một container cũng có thể chứa một container khác (sub-containers).

*Trong hình 7.2. có 3 containers (một Frame và hai Panels). Frame là container mức cao nhất của chương trình AWT. Frame có một thanh tiêu đề (chứa icon, tiêu đề*

và 3 nút Minimize/Maximize/Close), thanh trình đơn tùy chọn và khu vực hiển thị nội dung. Một Panel là một vùng hình chữ nhật được sử dụng để gom nhóm các thành phần GUI liên quan trong một bối cảnh nhất định. Trong trên, Frame chứa hai Panels. Có 5 component (Label, TextField, và 3 Buttons).

Trong một chương trình GUI, một component phải nằm trên một container, cần xác định một container để chứa các component. Mỗi container chứa một phương thức gọi là **add(Component c)**. Một container (như c) có thể gọi **c.add (aComponent)** để thêm một aComponent lên chính nó.

*Ví dụ:*

```
Panel pnl = new Panel(); // Panel là một container  
Button btn = new Button("Yes"); // Button là một component  
pnl.add(btn); // Panel container thêm Button lên component
```

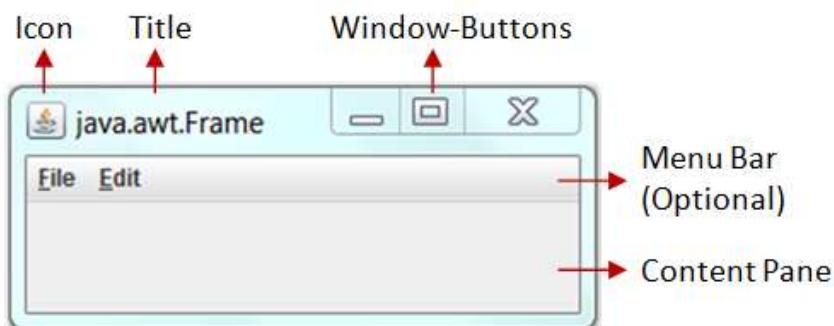
### 7.2.3. Các lớp AWT Container

#### 7.2.3.1. Top-Level Containers: Frame, Dialog and Applet

Mỗi chương trình GUI có một top-level container. Trong AWT top-level containers thường sử dụng là Frame, Dialog và Applet:

Frame cung cấp một cửa sổ chính cho ứng dụng GUI, có một thanh tiêu đề -title bar (gồm một icon, một tiêu đề, và các nút minimize, maximize/restore-down và close), một menu tự chọn (ẩn/hiện) và một vùng hiển thị nội dung. Để viết một chương trình GUI, chúng ta thường bắt đầu với một lớp con mở rộng từ lớp **java.awt.Frame** để kế thừa cửa sổ chính (main window).

*Ví dụ:*



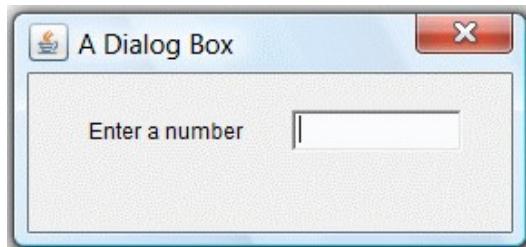
Hình 7.3. Một frame kế thừa từ lớp Frame

Chương trình GUI khai báo lớp FrameDemo là lớp con của Frame. Lớp con này kế thừa tất cả các thuộc tính của Frame như icon, title, buttons, content-pane.

### Khai báo class Frame: [java.awt.Frame](#)

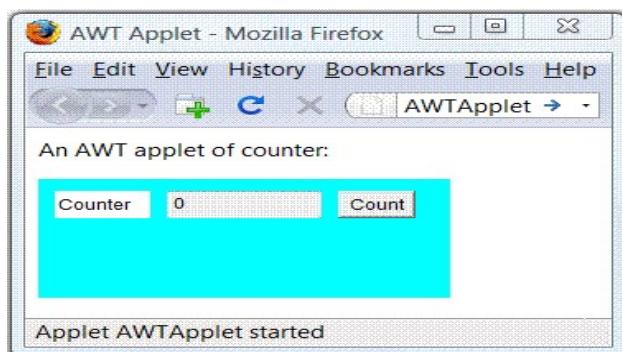
```
import java.awt.*;
class FrameDemo extends Frame {
    FrameDemo() {
        setTitle("Demo Container: Frame");
        setSize(500,300); //frame rộng 500 và cao 300
    }
    public static void main(String args[]){
        FrameDemo frameDemo = new FrameDemo();
        frameDemo.setVisible(true);
    }
}
```

Dialog: cửa sổ pop-up được sử dụng để tạo ra các tương tác nằm ngoài cửa sổ chính.



Hình 7.4. Dialog box

Applet: xây dựng chương trình chạy trên trình duyệt Web.

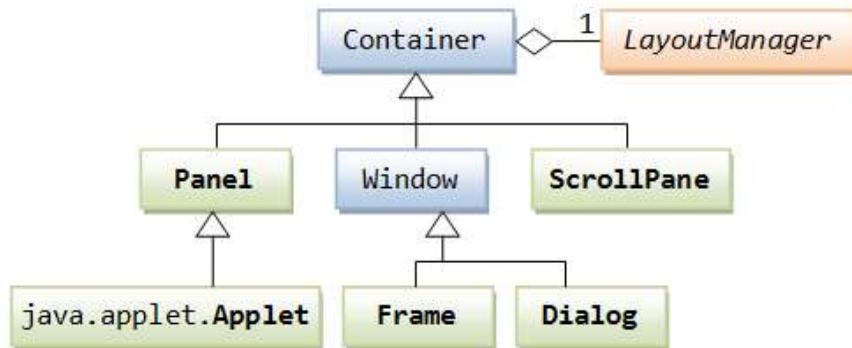


Hình 7.5. Applet

### 7.2.3.2. Secondary Containers: Panel và ScrollPane

Panel: khung chữ nhật nằm trong một top-level container, được sử dụng để tạo layout cho chương trình.

ScrollPane: tạo ra hiệu ứng cuộn chuột (ngang/dọc) cho một component.



Hình 7.6. Cây phân cấp của AWT container

### 7.2.4. Các lớp Component

AWT cung cấp nhiều GUI components trong gói `java.awt`. Thông dụng như: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, và Choice.



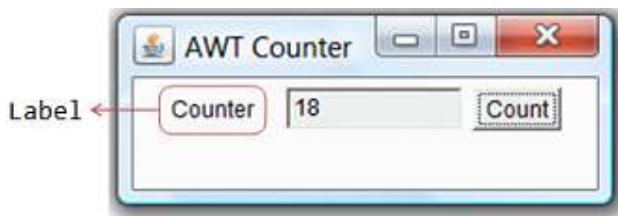
Hình 7.7. Các components thông dụng

Thêm đối tượng component:

1. Khai báo đối tượng component cần thêm.
2. Khởi tạo đối tượng với các phương thức khởi tạo phù hợp.
3. Xác định container chứa component cần thêm.
4. Dùng phương thức `add(aComponent)` để thêm component vào container.

#### 7.2.4.1. AWT GUI Component: java.awt.Label

Hiển thị một nội dung nào đó dưới dạng văn bản.



Hình 7.8. Label

Các phương thức khởi tạo:

```
// Khoi tao Label voi chuoi text, alignment cua text
public Label(String strLabel, int alignment);
public Label(String strLabel) // Khoi tao Label voi chuoi text
public Label(); // Khoi tao mot Label rong
```

Một số phương thức:

```
public String getText();
public void setText(String strLabel);
public int getAlignment();
public void setAlignment(int alignment);
```

Ví dụ:

```
Label lblInput; // Khai bao Label lblInput
lblInput = new Label("Enter ID: ");
add(lblInput);
lblInput.setText("Enter password: ");
lblInput.getText();
```

Nhân ẩn danh (Anonymous Instance): không thẻ tương tác.

```
add(new Label("Enter Name: ", Label.RIGHT));
Label xxx = new Label("Enter Name: ", Label.RIGHT));
// xxx assigned by compiler
add(xxx);
```

### 7.2.4.2. AWT GUI Component: java.awt.Button

Tạo ra một hành động nào đó của chương trình qua sự kiện nhấp chuột.



Hình 7.9. Button

*Các phương thức khởi tạo:*

```
public Button(String btnLabel); // Khoi tao Button voi nhan  
public Button(); // Khoi tao Button voi nhan rong
```

*Một số phương thức:*

```
public String getLabel(); // Get the label of this Button instance  
// Set the label of this Button instance  
public void setLabel(String btnLabel);  
// Enable or disable this Button. Disabled Button cannot be clicked  
public void setEnable(boolean enable);
```

*Sự kiện:*

Nhấp vào button sẽ kích hoạt cái gọi là ActionEvent và thực hiện một hành động được lập trình trước.

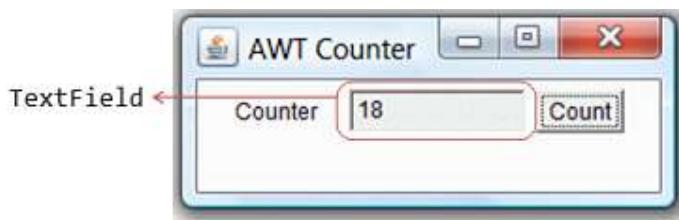
*Ví dụ:*

```
// Declare and allocate a Button instance called btnColor  
  
Button btnColor = new Button("Red");  
add(btnColor); // "this" Container adds the Button  
btnColor.setLabel("Green"); // Change the button's label  
btnColor.getLabel(); // Read the button's label
```

*Button ẩn danh (Anonymous Instance):*

```
// Create an anonymous Button. It CANNOT be referenced later  
add(Button("Blue"));
```

### 7.2.4.3. AWT GUI Component: java.awt.TextField



Hình 7.10. TextField

Sử dụng để nhập/xuất một dòng văn bản.

**Các phương thức khởi tạo:**

```
public TextField(String initialText, int columns);
// Khoi tao TextField voi chuoi text cho truoc va so cot columns.

public TextField(String initialText);
// Khoi tao TextField voi mot chuoi cho truoc.

public TextField(int columns);
// Khoi tao TextField voi so cot columns.
```

**Một số phương thức:**

```
public String getLabel(); // Get the label of this Button instance
public void setLabel(String btnLabel);
// Set the label of this Button instance
public void setEnable(boolean enable);
// Enable or disable this Button. Disabled Button cannot be clicked.
```

**Sự kiện:**

Khi đang ở trong TextField, nháy phím Enter có thể kích hoạt một hành động nào đó của chương trình.

**Ví dụ:**

```
TextField tfInput = new TextField(30);
// Declare and allocate an TextField instance called tfInput
add(tfInput); // "this" Container adds the TextField
TextField tfResult = new TextField();
```

```
// Declare and allocate an TextField instance called tfResult  
tfResult.setEditable(false); // Set to read-only  
add(tfResult); // "this" Container adds the TextField  
// Doc mot so int tu TextField "tfInput", va gan vao "tfResult".  
// getText() tra ve String, can chuyen sng int  
int number = Integer.parseInt(tfInput.getText());  
number *= number;  
// setText() yeu cau la String, can chuyen so kieu int sang String.  
tfResult.setText(number + "");
```

#### 7.2.4.4. Một chương trình đơn giản:

*Ví dụ 1: AWTCounter*

```
import java.awt.*; //Using AWT container and component classes  
import java.awt.event.*; //AWT event classes and listener interfaces  
//Lop AWTCounter ke thua tu top-level container java.awt.Frame  
public class AWTCounter extends Frame implements ActionListener {  
    private Label lblCount;          // Declare a Label component  
    private TextField tfCount;       // Declare a TextField component  
    private Button btnCount;         // Declare a Button component  
    private int count = 0;           // Counter's value  
  
    // Constructor to setup GUI components and event handlers  
    public AWTCounter () {  
        setLayout(new FlowLayout());  
        // sets its layout to FlowLayout, left-to-right,va top-to-bottom.  
        lblCount = new Label("Counter");  
        add(lblCount); // adds Label component  
        tfCount = new TextField("0", 10); // TextField component  
        tfCount.setEditable(false); // set to read-only  
        add(tfCount); // adds TextField component  
        btnCount = new Button("Count");  
        add(btnCount); // adds Button component
```

```
btnCount.addActionListener(this);

/* "btnCount" is the source object that fires an ActionEvent when
clicked. The source add "this" instance as an ActionEvent listener,
which provides an ActionEvent handler called actionPerformed(). */
// Clicking "btnCount" invokes actionPerformed().

setTitle("AWT Counter"); // "super" Frame sets its title
setSize(250, 100); // Frame sets its initial window size
// For inspecting the Container/Components objects
// System.out.println(this);
// System.out.println(lblCount);
// System.out.println(tfCount);
// System.out.println(btnCount);
setVisible(true); // "super" Frame shows
// System.out.println(this);
// System.out.println(lblCount);
// System.out.println(tfCount);
// System.out.println(btnCount);

}

public static void main(String[] args) {
// Invoke the constructor to setup the GUI, by allocating an instance
    AWTCounter app = new AWTCounter();
// or simply "new AWTCounter();" for an anonymous instance
}

// ActionEvent handler - Called back upon button-click.

@Override
public void actionPerformed(ActionEvent evt) {
    ++count; // Increase the counter value
    // Display the counter value on the TextField tfCount
    tfCount.setText(count + ""); // Convert int to String
}
}
```

**Kiểm tra Container/Components → `toString()`:**

Thêm đoạn code vào trước và sau `setVisible(true);`

```
System.out.println(this);
System.out.println(lblCount);
System.out.println(tfCount);
System.out.println(btnCount);

setVisible(true); // "super" Frame shows

System.out.println(this);
System.out.println(lblCount);
System.out.println(tfCount);
System.out.println(btnCount);
```

Xem kết quả như thế nào?

**Ví dụ 2: `AWTAccumulator`**

```
import java.awt.*; // Using AWT container and component classes
import java.awt.event.*; // AWT event classes and listener interfaces

public class AWTAccumulator extends Frame implements ActionListener {
    private Label lblInput;           // Declare input Label
    private Label lblOutput;          // Declare output Label
    private TextField tfInput;         // Declare input TextField
    private TextField tfOutput;        // Declare output TextField
    private int sum = 0;               // Accumulated sum, init to 0

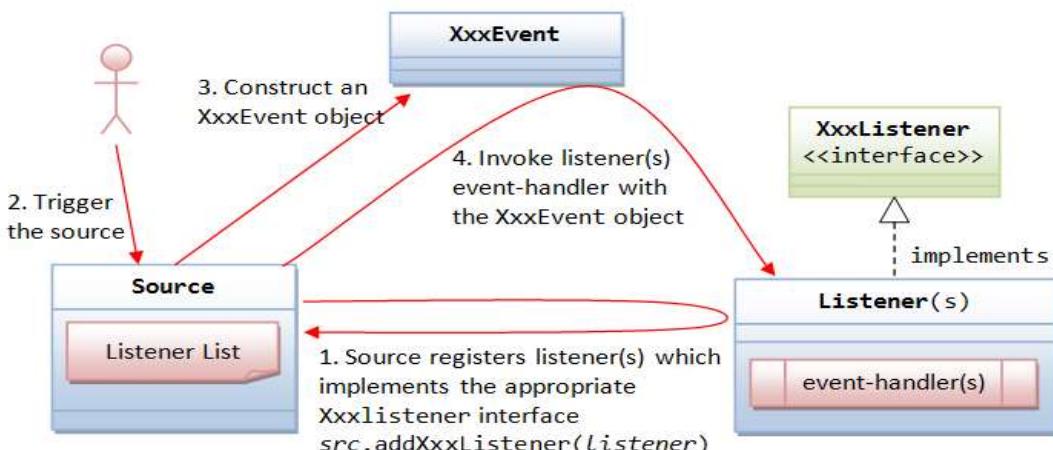
    // Constructor to setup the GUI components and event handlers
    public AWTAccumulator() {
        setLayout(new FlowLayout()); // sets layout to FlowLayout
        lblInput = new Label("Enter an Integer:"); // Construct Label
        add(lblInput); // adds Label component
        tfInput = new TextField(10); // Construct TextField
```

```
        add(tfInput);      // "super" Frame adds TextField
        tfInput.addActionListener(this);
    // "tfInput" is the source object that fires an ActionEvent upon entered.
    /* The source add "this" instance as an ActionEvent listener, which
    provides an ActionEvent handler called actionPerformed(). */
    // Hitting "enter" on tfInput invokes actionPerformed().
    lblOutput = new Label("The Accumulated Sum is: ");
    add(lblOutput); // "super" Frame adds Label
    tfOutput = new TextField(10); // allocate TextField
    tfOutput.setEditable(false); // read-only
    add(tfOutput); // "super" Frame adds TextField
    setTitle("AWT Accumulator"); // "super" Frame sets title
    setSize(350, 120); // initial window size
    setVisible(true); // "super" Frame shows
}
// The entry main() method
public static void main(String[] args) {
// Gọi constructor để cài đặt GUI, bằng một anonymous instance
    new AWTAccumulator();
}
// ActionEvent handler - Được gọi khi nhấn phím "enter" trên TextField
@Override
public void actionPerformed(ActionEvent evt) {
// Get the String entered into the TextField tfInput, convert to int
    int numberIn = Integer.parseInt(tfInput.getText());
    sum += numberIn; // Accumulate numbers entered into sum
    tfInput.setText(""); // Clear input TextField
    tfOutput.setText(sum + ""); // Xuất tổng ra TextField
                                // chuyển sang String
}
}
```

### 7.2.5. Lắng nghe và xử lý sự kiện trên AWT (AWT Event-Handling)

#### 7.2.5.1. Giới thiệu

- Người dùng tương tác với chương trình qua giao diện.
- Chương trình phải nghe được các sự kiện trên giao diện (nhập dữ liệu, nhấn phím Enter, nhấp chuột, đóng cửa sổ chương trình...) để thực hiện hành động tương ứng → lập trình hướng sự kiện.
- Tham gia sự kiện luôn có 3 đối tượng: nguồn (source) sinh sự kiện, bộ nghe sự kiện (listener), và sự kiện (event).
- Nguồn (source): là nơi phát sinh sự kiện (button, textfield,...).
- Mỗi nguồn sẽ đăng ký các bộ nghe sự kiện khác nhau.
- Khi có sự kiện nào đó xảy ra từ nguồn, phương thức xử lý sự kiện (event handler) trên bộ nghe sự kiện sẽ được gọi để xử lý.



Hình 7.11. Nghe và xử lý sự kiện

#### 7.2.5.2. Trình tự các bước thực hiện

1. Đối tượng nguồn (source object) đăng ký các bộ lắng nghe sự kiện:

Đối tượng nguồn khởi tạo sự kiện khi kích hoạt.

**Ví dụ:** nhấp vào một Button kích hoạt ActionEvent, nhấp chuột kích hoạt MouseEvent, nhấn phím kích hoạt KeyEvent,...

Cần khai báo một interface được gọi là XxxListener (ví dụ, MouseListener) chứa các phương thức xử lý.

**Lưu ý:** một interface chỉ chứa các phương thức trừu tượng không thực thi được.

**Ví dụ:**

```
/** MouseListener interface, which declares the signature of the
handlers for the various operational modes. */
public interface MouseListener {
    public void mousePressed(MouseEvent evt);
    // Called back upon mouse-button pressed
    public void mouseReleased(MouseEvent evt);
    // Called back upon mouse-button released
    public void mouseClicked(MouseEvent evt);
    // Called back upon mouse-button clicked (pressed and released)
    public void mouseEntered(MouseEvent evt);
    // Called back when mouse pointer entered the component
    public void mouseExited(MouseEvent evt);
    // Called back when mouse pointer exited the component
}
```

Tất cả listeners trong XxxEvent phải thực thi XxxListener interface. Các abstract methods được khai báo trong XxxListener interface. Bằng cách này, các listener(s) có thể phản ứng lại những sự kiện này một cách hợp lý.

**Ví dụ:**

```
/** An example of MouseListener, which provides implementation to
the handler methods*/
class MyMouseListener implements MouseListener {
    @Override
    public void mousePressed(MouseEvent e) {
        System.out.println("Mouse-button pressed!");
    }
    @Override
    public void mouseReleased(MouseEvent e) {
        System.out.println("Mouse-button released!");
    }
}
```

```
@Override  
public void mouseClicked(MouseEvent e) {  
    System.out.println("Mouse-button clicked (pressed and released)!");  
}  
@Override  
public void mouseEntered(MouseEvent e) {  
    System.out.println("Mouse-pointer entered the source component!");  
}  
@Override  
public void mouseExited(MouseEvent e) {  
    System.out.println("Mouse exited-pointer the source component!");  
}  
}
```

Định nghĩa hai methods: addXxxListener() và removeXxxListener() để thêm và xóa một listener. Chữ ký của phương thức là:

```
public void addXxxListener(XxxListener l);  
public void removeXxxListener(XxxListener l);
```

Đối tượng nguồn sau đó đăng ký đối tượng Listener thông qua phương thức addXxxListener ():

```
aSource.addXxxListener(alistener); //aSource->aListener cho XxxEvent
```

2. Kích hoạt sự kiện.
3. Khởi tạo một sự kiện XxxEvent.
4. Gọi phương thức xử lý sự kiện XxxEven.

#### 7.2.5.3. Ví dụ : AWTCounter: ActionEvent và ActionListener Interface

Click vào Button (nhấn "Enter" trên TextField) kích hoạt một ActionEvent tới tất cả các trình lắng nghe ActionEvent của nó. Một ActionEvent listener phải thực thi ActionListener interface, khai báo một abstract method **actionPerformed()** như sau:

```
public interface ActionListener {  
    public void actionPerformed(ActionEvent evt);  
    // Goi khi click button (Button), nhan phím ( TextField)  
}
```

### Các bước thực hiện:

1. Xác định **btnCount** (Button) làm đối tượng nguồn.
2. Click Button kích hoạt ActionEvent đến tất cả ActionEvent listener(s).
3. Các trình lắng nghe sự kiện yêu cầu thực thi ActionListener interface, cài đè phương thức **actionPerformed()**

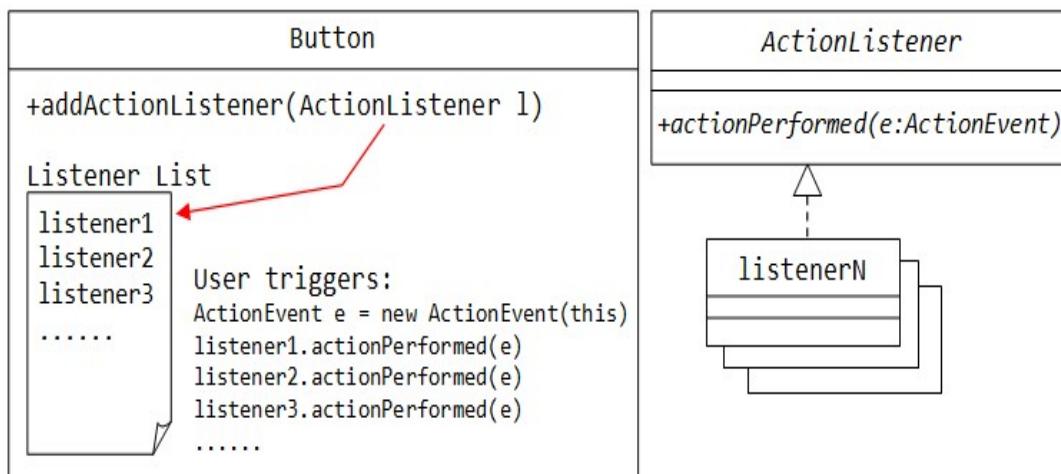
```
public class AWTCounter extends Frame implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent evt) {
        // Increment the count value and display on the TextField
        ++count;
        tfCount.setText(count + "");
    }
}
```

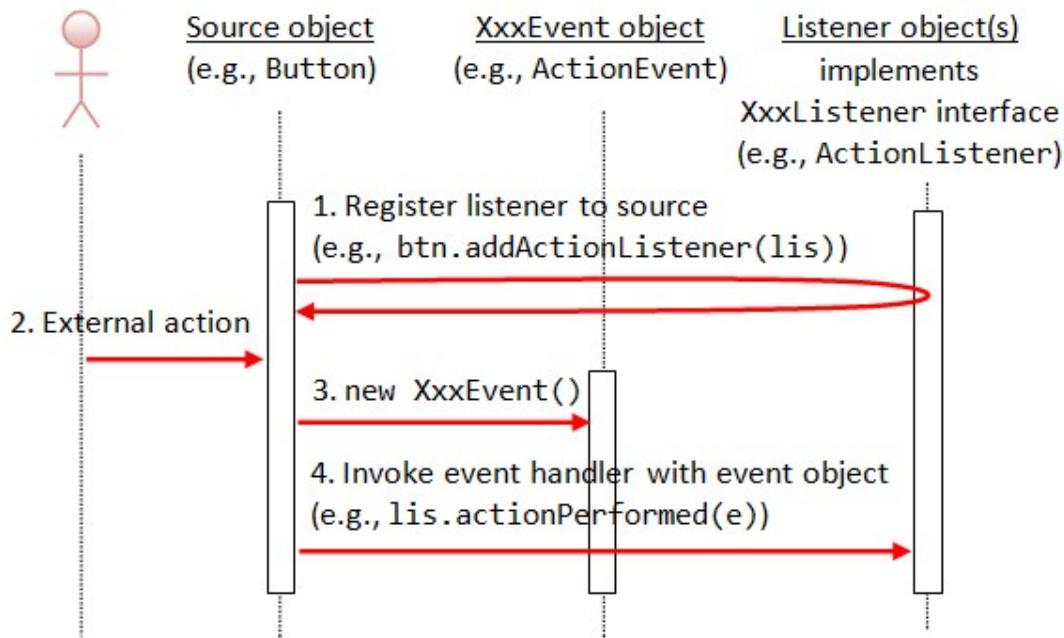
Đối tượng nguồn đăng ký listener qua addListener(). Trong ví dụ này:

```
btnCount.addActionListener(this);
```

Sau khi kích nút *btnCount* tạo một đối tượng ActionEvent và gọi phương thức actionPerformed (ActionEvent) của tất cả các trình lắng nghe đã đăng ký với đối tượng ActionEvent được tạo ra:

```
ActionEvent evt = new ActionEvent( ..... );
listener.actionPerformed(evt); // for all its listener(s)
```





Hình 7.12. Diagram các bước thực hiện

#### 7.2.5.4. Ví dụ: AWTAccumulator: ActionEvent và ActionListener Interface

Trong ví dụ này:

1. Xác định ***tfInput*** (TextField) làm đối tượng nguồn.
2. Nhấn "Enter" trên TextField kích hoạt một ActionEvent đến tất cả trình lắng nghe sự kiện của ActionEvent.
3. Chọn trình lắng nghe sự kiện ActionEvent listener.
4. Đối tượng nguồn ***tfInput*** (TextField) đăng ký lắng nghe sự kiện ***tfInput.addActionListener(this)***.
5. Trình lắng nghe sự kiện ActionEvent yêu cầu thực thi ActionListener interface, và ghi đè (override) phương thức ***actionPerformed()***.

#### 7.2.5.5. Ví dụ : WindowEvent và WindowListener Interface



```
public interface WindowListener{  
    public void windowClosing(WindowEvent evt);  
    public void windowOpened(WindowEvent evt);  
    public void windowClosed(WindowEvent evt);  
    public void windowActivated(WindowEvent evt);  
    public void windowDeactivated(WindowEvent evt);  
    public void windowIconified(WindowEvent evt);  
    public void windowDeiconified(WindowEvent evt);  
}
```

**Ví dụ: AWTCounter**

```
import java.awt.*; // AWT containers va components  
import java.awt.event.*; // AWT events classes va listener interfaces  
// CT AWT GUI ke thua top-level container java.awt.Frame  
public class WindowEventDemo extends Frame  
    implements ActionListener, WindowListener {  
    // This class acts as listener for ActionEvent and WindowEvent  
    // Mot class chi ke thua tu 1 lop, nhung co the nhieu interfaces.  
    private TextField tfCount; // Declare a TextField component  
    private Button btnCount; // Declare a Button component  
    private int count = 0; // Counter's value  
    // Constructor to setup the GUI components and event handlers  
    public WindowEventDemo() {  
        setLayout(new FlowLayout()); // sets to FlowLayout  
        add(new Label("Counter")); // adds an anonymous Label  
        tfCount = new TextField("0", 10); // Construct the TextField  
        tfCount.setEditable(false); // read-only  
        add(tfCount); // "super" Frame adds TextField  
        btnCount = new Button("Count"); // Construct the Button  
        add(btnCount); // "super" Frame adds Button  
        btnCount.addActionListener(this);  
    }  
    // Implementing the WindowListener interface  
    public void windowClosing(WindowEvent evt) {  
        System.out.println("Window is closing...");  
    }  
    public void windowIconified(WindowEvent evt) {  
        System.out.println("Window is iconified...");  
    }  
    public void windowDeiconified(WindowEvent evt) {  
        System.out.println("Window is deiconified...");  
    }  
    public void windowOpened(WindowEvent evt) {  
        System.out.println("Window is opened...");  
    }  
    public void windowActivated(WindowEvent evt) {  
        System.out.println("Window is activated...");  
    }  
    public void windowClosed(WindowEvent evt) {  
        System.out.println("Window is closed...");  
    }  
    public void windowDeactivated(WindowEvent evt) {  
        System.out.println("Window is deactivated...");  
    }  
}
```

```
// btnCount (source object) fires ActionEvent upon clicking
// btnCount adds "this" object as an ActionEvent listener
    addWindowListener(this);

// "super" Frame (source object) fires WindowEvent.
// "super" Frame adds "this" object as a WindowEvent listener.
    setTitle("WindowEvent Demo"); // "super" Frame sets title
    setSize(500, 200); // "super" Frame sets initial size
    setVisible(true); // "super" Frame shows

}
// The entry main() method

public static void main(String[] args) {
    new WindowEventDemo(); // Let the construct do the job
}

/* ActionEvent handler */

@Override
public void actionPerformed(ActionEvent evt) {
    ++count;
    tfCount.setText(count + "");
}

/* WindowEvent handlers */

// Called back upon clicking close-window button

@Override
public void windowClosing(WindowEvent evt) {
    System.exit(0); // Terminate the program
}

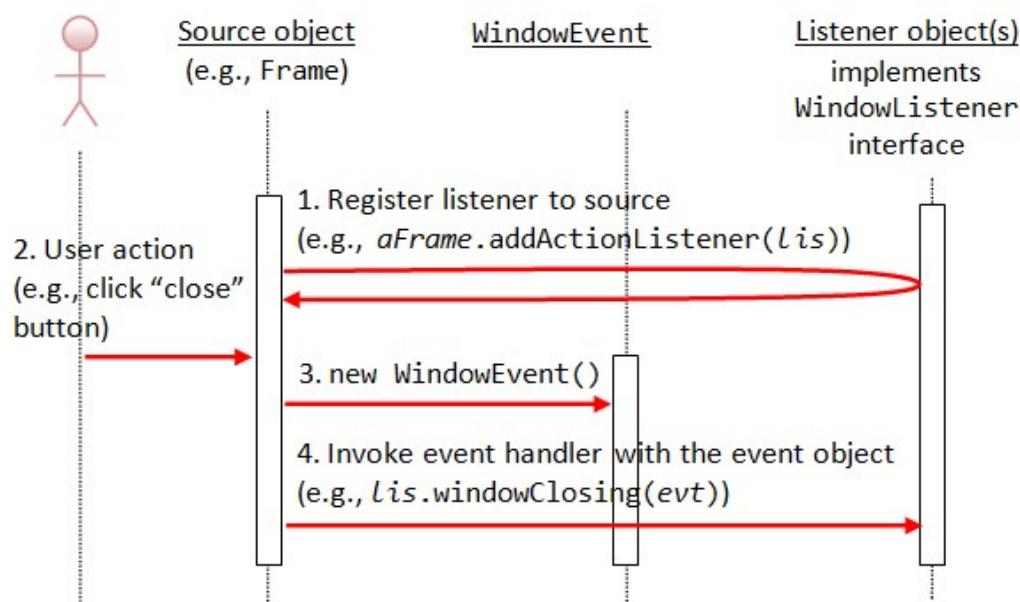
// Not Used, but need to provide an empty body to compile.

@Override public void windowOpened(WindowEvent evt) { }
@Override public void windowClosed(WindowEvent evt) { }
@Override public void windowIconified(WindowEvent evt) { }
@Override public void windowDeiconified(WindowEvent evt) { }
@Override public void windowActivated(WindowEvent evt) { }
@Override public void windowDeactivated(WindowEvent evt) { }

}
```

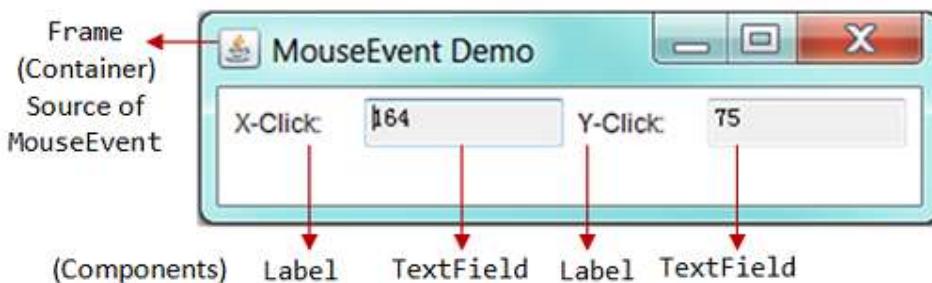
Ví dụ này, sửa đổi ví dụ AWTCounter trước đó để xử lý WindowEvent.

1. Xác định **Frame** làm đối tượng nguồn.
2. Frame kích hoạt WindowEvent đến tất cả trình lắng nghe sự kiện của WindowEvent.
3. Chọn trình lắng nghe sự kiện WindowEvent listener.
4. Đăng ký lắng nghe sự kiện WindowEvent thông qua phương thức `addWindowListener(this)`.
5. Trình lắng nghe sự kiện WindowEvent yêu cầu thực thi WindowListener, gồm: `windowOpened()`, `windowClosed()`, `windowClosing()`, `windowActivated()`, `windowDeactivated()`, `windowIconified()` và `windowDeiconified()`.
6. Override phương thức `windowClosing()` xử lý đóng cửa sổ chương trình sử dụng `System.exit(0)`. Cài đặt rỗng 6 phương thức còn lại.



Hình 7.13. Diagram WindowEvent

#### 7.2.5.6. Ví dụ: MouseEvent và MouseListener Interface



MouseEvent được kích hoạt khi nhấn, thả, click mouse-button (trái / phải) của đối tượng nguồn.

Trình lắng nghe sự kiện MouseEvent phải được thực thi MouseListener interface, với các abstract methods:

```
public void mouseClicked(MouseEvent evt)
//Called-back: mouse-button has been clicked on the source.

public void mousePressed(MouseEvent evt)

public void mouseReleased(MouseEvent evt)
//Called-back: mouse-button has been pressed/released on the source.
//Click -> goi mousePressed(), mouseReleased() and mouseClicked().

public void mouseEntered(MouseEvent evt)

public void mouseExited(MouseEvent evt)
// Called-back when the mouse-pointer has entered/exited the source.
```

Ví dụ:

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

public class MouseEventDemo extends Frame implements MouseListener {
    private TextField tfMouseX; // to display mouse-click-x
    private TextField tfMouseY; // to display mouse-click-y
    // Constructor - Setup the UI components and event handlers
    public MouseEventDemo() {
        setLayout(new FlowLayout()); //sets layout to FlowLayout
        // Label (anonymous)
        add(new Label("X-Click: ")); // adds Label component
        // TextField
        tfMouseX = new TextField(10); // 10 columns
        tfMouseX.setEditable(false); // read-only
        add(tfMouseX); // "super" frame adds TextField component
        // Label (anonymous)
        add(new Label("Y-Click: ")); //adds Label component
    }
}
```

```
// TextField

    tfMouseY = new TextField(10);

    tfMouseY.setEditable(false); // read-only

    add(tfMouseY); // "super" frame adds TextField component

    addMouseListener(this);

// "super" frame (source) fires the MouseEvent.

// "super" frame adds "this" object as a MouseEvent listener.

    setTitle("MouseEvent Demo"); // "super" Frame sets title

    setSize(500, 200); // "super" Frame sets initial size

    setVisible(true); // "super" Frame shows

}

public static void main(String[] args) {

    new MouseEventDemo(); // Let the constructor do the job
}

/* MouseEvent handlers */

// Called back upon mouse clicked

@Override

public void mouseClicked(MouseEvent evt) {

    tfMouseX.setText(evt.getX() + "");

    tfMouseY.setText(evt.getY() + "");

}

// Not used - need to provide an empty body to compile.

@Override

public void mousePressed(MouseEvent evt) { }

@Override

public void mouseReleased(MouseEvent evt) { }

@Override

public void mouseEntered(MouseEvent evt) { }

@Override

public void mouseExited(MouseEvent evt) { }

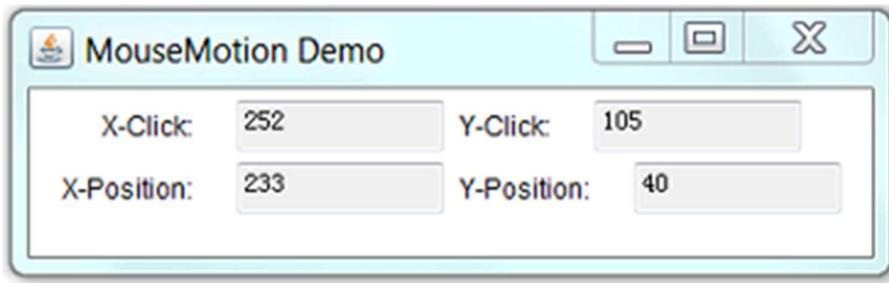
}
```

Trong ví dụ này, bố trí 4 components (2 Labels và 2 non-editable TextFields) trên top-level container Frame với Layout Manager là FlowLayout

**MouseEvent:**

1. Xác định **Frame** làm đối tượng nguồn.
2. Frame kích hoạt MouseEvent đến tất cả trình lắng nghe sự kiện của MouseEvent.
3. Chọn bộ lắng nghe sự kiện MouseEvent listener.
4. Đăng ký lắng nghe sự kiện MouseEvent thông qua phương thức **addMouseListener(this)**.
5. Trình lắng nghe sự kiện WindowEvent yêu cầu thực thi MouseListener, gồm 5 abstract methods: **mouseClicked()**, **mousePressed()**, **mouseReleased()**, **mouseEntered()**, và **mouseExit()**.
6. Override phương thức **mouseClicked()** để hiển thị tọa độ (x, y) khi click chuột vào 2 vùng **TextFields**. Bỏ qua các phương thức khác, nhưng cần cài đặt rõ ràng biên dịch.

#### 7.2.6. Ví dụ 5: MouseEvent và MouseMotionListener Interface



MouseEvent cũng được kích hoạt khi di chuyển và rê con trỏ chuột tại đối tượng nguồn. Nhưng cần sử dụng MouseMotionListener để điều khiển di chuyển chuột và rê chuột. MouseMotionListener interface có hai phương thức trừu tượng sau đây:

```
public void mouseDragged(MouseEvent e)
/*Called-back when a mouse-button is pressed on the source
component and then dragged*/
public void mouseMoved(MouseEvent e)
/*Called-back when the mouse-pointer has been moved onto the source
component but no buttons have been pushed*/
```

Ví dụ:

```
import java.awt.*;
import java.awt.event.*;
// AWT GUI program ke thua top-level container java.awt.Frame
public class MouseMotionDemo extends Frame
    implements MouseListener, MouseMotionListener {
// This class acts as MouseListener and MouseMotionListener
// To display the (x, y) of the mouse-clicked
    private TextField tfMouseClickX;
    private TextField tfMouseClickY;
// To display the (x, y) of the current mouse-pointer position
    private TextField tf.mousePositionX;
    private TextField tf.mousePositionY;
// Constructor to setup the GUI components and event handlers
    public MouseMotionDemo() {
        setLayout(new FlowLayout()); //sets to FlowLayout
        add(new Label("X-Click: "));
        tfMouseClickX = new TextField(10);
        tfMouseClickX.setEditable(false);
        add(tfMouseClickX);
        add(new Label("Y-Click: "));
        tfMouseClickY = new TextField(10);
        tfMouseClickY.setEditable(false);
        add(tfMouseClickY);
        add(new Label("X-Position: "));
        tf.mousePositionX = new TextField(10);
        tf.mousePositionX.setEditable(false);
        add(tf.mousePositionX);
        add(new Label("Y-Position: "));
        tf.mousePositionY = new TextField(10);
        tf.mousePositionY.setEditable(false);
```

```
        add(tfMousePositionY);
        addMouseListener(this);
        addMouseMotionListener(this);

    // "super" frame (source) fires MouseEvent.
    // adds "this" object as MouseListener and MouseMotionListener.
    setTitle("MouseMotion Demo"); // "super" Frame sets title
    setSize(400, 300); // "super" Frame sets initial size
    setVisible(true); // "super" Frame shows

}

// The entry main() method
public static void main(String[] args) {
    new MouseMotionDemo(); // Let the constructor do the job
}

/** MouseListener handlers */
// Called back when a mouse-button has been clicked
@Override
public void mouseClicked(MouseEvent evt) {
    tfMouseClickX.setText(evt.getX() + "");
    tfMouseClickY.setText(evt.getY() + "");
}

// Not Used, but need to provide an empty body for compilation
@Override
public void mousePressed(MouseEvent evt) { }

@Override
public void mouseReleased(MouseEvent evt) { }

@Override
public void mouseEntered(MouseEvent evt) { }

@Override
public void mouseExited(MouseEvent evt) { }

/** MouseMotionEvent handlers */
// Called back when the mouse-pointer has been moved
```

```

@Override
public void mouseMoved(MouseEvent evt) {
    tfMousePositionX.setText(evt.getX() + "");
    tfMousePositionY.setText(evt.getY() + "");
}

// Not Used, but need to provide an empty body for compilation

@Override
public void mouseDragged(MouseEvent evt) { }

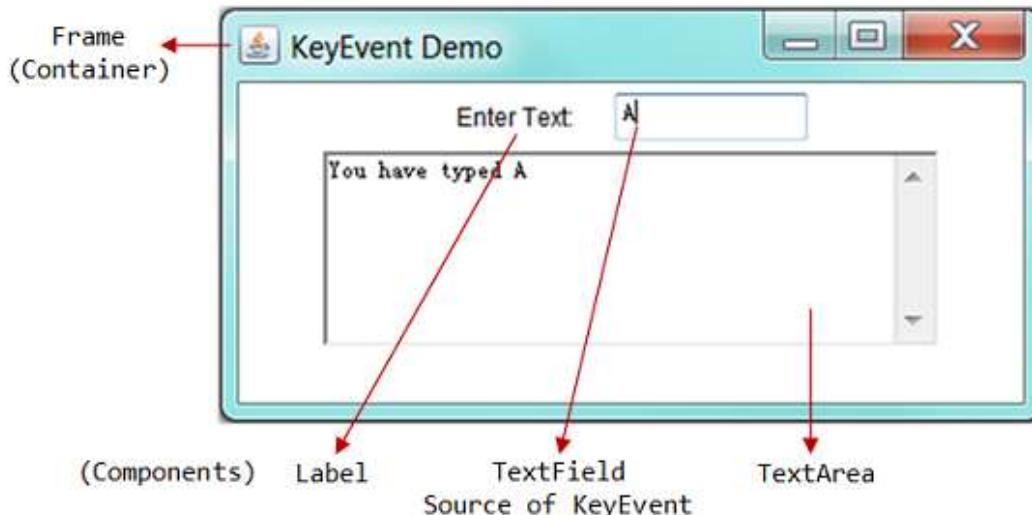
}

```

Trong ví dụ này, minh họa cả MouseListener và MouseMotionListener:

1. Xác định **Frame** làm đối tượng nguồn.
2. Frame kích hoạt MouseEvent đến tất cả trình lắng nghe sự kiện của MouseListener và MouseMotionListener.
3. Chọn bộ lắng nghe sự kiện MouseListener và MouseMotionListener.
4. Đăng ký lắng nghe sự kiện thông qua phương thức **addMouseListener(this)** và **addMouseMotionListener(this)**.
5. Trình lắng nghe sự kiện MouseMotionListener yêu cầu thực thi 2 abstract methods: **mouseMoved()** và **mouseDragged()** trong MouseMotionListener interface.
6. Override phương thức **mouseMoved()** để hiển thị vị trí (x, y) của chuột. Bỏ qua phương thức **MouseDragged()**, nhưng cần cài đặt rỗng để biên dịch.

#### 7.2.6.1. Ví dụ: KeyEvent và KeyListener Interface



Một KeyEvent được kích hoạt khi nhấn, thả và nhập trên đối tượng nguồn bàn phím. KeyEvent listener phải thực thi KeyListener interface, với abstract methods:

```
public void keyTyped(KeyEvent e)
// Called-back when a key has been typed (pressed and released).
public void keyPressed(KeyEvent e)
public void keyReleased(KeyEvent e)
// Called-back when a key has been pressed or released.
```

Ví dụ:

```
import java.awt.*;
import java.awt.event.*;
// AWT GUI program ke thua top-level container java.awt.Frame
public class KeyEventDemo extends Frame implements KeyListener {
// This class acts as KeyEvent Listener
    private TextField tfInput; //Single-line TextField
    private TextArea taDisplay; // Multi-line TextArea
// Constructor to setup the GUI components and event handlers
    public KeyEventDemo() {
        setLayout(new FlowLayout()); // sets to FlowLayout
        add(new Label("Enter Text: "));
        tfInput = new TextField(10);
        add(tfInput);
        taDisplay = new TextArea(5, 40); // 5 rows, 40 columns
        add(taDisplay);
        tfInput.addKeyListener(this);
// tfInput TextField (source) fires KeyEvent.
// tfInput adds "this" object as a KeyEvent listener.
        setTitle("KeyEvent Demo"); // "super" Frame sets title
        setSize(500, 300); // "super" Frame sets initial size
        setVisible(true); // "super" Frame shows
    }
}
```

```
// The entry main() method
public static void main(String[] args) {
    new KeyEventDemo(); // Let the constructor do the job
}

/** KeyEvent handlers */
// Called back when a key has been typed (pressed and released)
@Override
public void keyTyped(KeyEvent evt) {
    taDisplay.append("You have typed "+evt.getKeyChar()+"\n");
}

// Not Used, but need to provide an empty body for compilation
@Override
public void keyPressed(KeyEvent evt) { }
@Override
public void keyReleased(KeyEvent evt) { }
}
```

Trong ví dụ này,

1. Xác định **tfInput** (TextField) làm đối tượng nguồn.
2. KeyEvent kích hoạt khi press/release/type một phím đến tất cả trình lắng nghe sự kiện của KeyEvent.
3. Chọn bộ lắng nghe sự kiện KeyEvent listener.
4. Đăng ký lắng nghe sự kiện KeyEvent listener đến đối tượng nguồn TextField thông qua phương thức **tfInput.addKeyListener(this)**.
5. Trình lắng nghe sự kiện KeyEvent listener yêu cầu thực thi KeyListener interface, thông qua 3 abstract methods: **keyTyped()**, **keyPressed()**, **keyReleased()**.
6. Override phương thức **keyTyped()** để hiển thị giá trị được nhập ra TextArea (x, y) của chuột. Bỏ qua phương thức **keyPressed()** and **keyReleased()**, nhưng cần cài đặt rỗng để biên dịch.

### 7.2.7. Lớp Nested (Inner)

```
public class MyOuterClass {    // outer class defined here
    .....
    private class MyNestedClass1 { ..... }
    // an nested class defined inside the outer class
    public static class MyNestedClass2 { ..... }
    // an "static" nested class defined inside the outer class
    .....
}
```

#### 7.2.7.1. Ví dụ: Lớp Named Inner cho Event Listener

```
import java.awt.*;
import java.awt.event.*;
// AWT GUI program ke thua top-level container java.awt.Frame
public class AWTCounterNamedInnerClass extends Frame {
    /** This class is NOT a ActionListener, hence, it does not implement
    ActionListener interface*/
    /** The event-handler actionPerformed() needs to access these
    "private" variables*/
    private TextField tfCount;
    private Button btnCount;
    private int count = 0;
    // Constructor to setup the GUI components and event handlers
    public AWTCounterNamedInnerClass () {
        setLayout(new FlowLayout()); // sets to FlowLayout
        add(new Label("Counter")); //An anonymous instance of Label
        tfCount = new TextField("0", 10);
        tfCount.setEditable(false); // read-only
        add(tfCount); // "super" Frame adds tfCount
        btnCount = new Button("Count");
        add(btnCount); // "super" Frame adds btnCount
    }
}
```

```
/** Construct an anonymous instance of BtnCountListener (a named inner class)*/
// btnCount adds this instance as a ActionListener.

    btnCount.addActionListener(new BtnCountListener());

    setTitle("AWT Counter");

    setSize(400, 200);

    setVisible(true);

}

public static void main(String[] args) { // The entry main method

    new AWTCounterNamedInnerClass(); // Let the constructor do the job

}

/**BtnCountListener is a "named inner class" used as ActionListener.
This inner class can access private variables of the outer class.*/
private class BtnCountListener implements ActionListener {

    @Override

    public void actionPerformed(ActionEvent evt) {

        ++count;

        tfCount.setText(count + "");

    }
}
}
```

### 7.2.7.2. Ví dụ: Lớp Anonymous Inner cho Event Listener

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

// AWT GUI program ke thua top-level container java.awt.Frame

public class AWTCounterAnonymousInnerClass extends Frame {

    /* This class is NOT a ActionListener, hence, it does not implement
    ActionListener interface*/

    // The event-handler actionPerformed() needs to access these private variables

    private TextField tfCount;

    private Button btnCount;

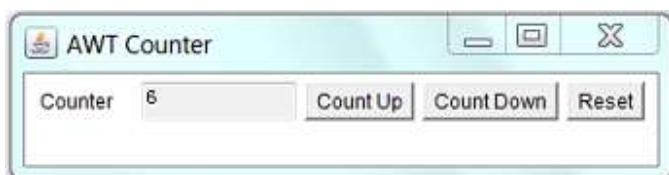
    private int count = 0;
```

```
// Constructor to setup the GUI components and event handlers

public AWTCounterAnonymousInnerClass () {
    setLayout(new FlowLayout()); // "super" Frame sets to FlowLayout
    add(new Label("Counter")); // An anonymous instance of Label
    tfCount = new TextField("0", 10);
    tfCount.setEditable(false); // read-only
    add(tfCount); // "super" Frame adds tfCount
    btnCount = new Button("Count");
    add(btnCount); // "super" Frame adds btnCount
    // Construct an anonymous instance of an anonymous class.
    // btnCount adds this instance as a ActionListener.
    btnCount.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent evt) {
            ++count;
            tfCount.setText(count + "");
        }
    });
    setTitle("AWT Counter");
    setSize(300, 200);
    setVisible(true);
}

public static void main(String[] args) { // The entry main method
    new AWTCounterAnonymousInnerClass();
    // Let the constructor do the job
}
}
```

#### 7.2.7.3. Ví dụ: Lớp Anonymous Inner cho từng Source



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

// An AWT GUI program inherits the top-level container java.awt.Frame

public class AWTCounter3Buttons extends Frame {
    private TextField tfCount;
    private Button btnCountUp, btnCountDown, btnReset;
    private int count = 0;

    // Constructor to setup the GUI components and event handlers
    public AWTCounter3Buttons () {
        setLayout(new FlowLayout());
        add(new Label("Counter")); // an anonymous instance of Label
        tfCount = new TextField("0", 10);
        tfCount.setEditable(false); // read-only
        add(tfCount); // "super" Frame adds tfCount
        btnCountUp = new Button("Count Up");
        add(btnCountUp);

        // Construct an anonymous instance of an anonymous inner class.
        // The source Button adds the anonymous instance as ActionEvent listener
        btnCountUp.addActionListener(new ActionListener() {
            @Override public void actionPerformed(ActionEvent evt) {
                ++count;
                tfCount.setText(count + "");
            }
        });
        btnCountDown = new Button("Count Down");
        add(btnCountDown);
        btnCountDown.addActionListener(new ActionListener() {
            @Override public void actionPerformed(ActionEvent evt) {
                count--;
                tfCount.setText(count + "");
            }
        });
    }
}
```

```
btnReset = new Button("Reset");
add(btnReset);
btnReset.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent evt) {
        count = 0;
        tfCount.setText("0");
    }
});
setTitle("AWT Counter");
setSize(400, 200);
setVisible(true);
}

public static void main(String[] args) { // The entry main method
    new AWTCounter3Buttons(); // Let the constructor do the job
}
}
```

#### 7.2.7.4. Ví dụ: Sử dụng cùng Listener Instance cho tất cả Buttons

*Sử dụng `getActionCommand()` của `ActionEvent`*

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

// An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTCounter3Buttons1Listener extends Frame {
    private TextField tfCount;
    private Button btnCountUp, btnCountDown, btnReset;
    private int count = 0;
    // Constructor to setup the GUI components and event handlers
    public AWTCounter3Buttons1Listener () {
        setLayout(new FlowLayout());
        add(new Label("Counter"));
    }
}
```

```
tfCount = new TextField("0", 10);
tfCount.setEditable(false);
add(tfCount);

// Construct Buttons
btnCountUp = new Button("Count Up");
add(btnCountUp);

btnCountDown = new Button("Count Down");
add(btnCountDown);

btnReset = new Button("Reset");
add(btnReset);

// Allocate an instance of the "named" inner class BtnListener.
BtnListener listener = new BtnListener();

// Use the same listener instance for all the 3 Buttons.
btnCountUp.addActionListener(listener);
btnCountDown.addActionListener(listener);
btnReset.addActionListener(listener);
setTitle("AWT Counter");
setSize(400, 200);
setVisible(true);
}

public static void main(String[] args) { //The entry main method
    new AWTCounter3Buttons1Listener(); //Let the constructor do the job
}

/** BtnListener is a named inner class used as ActionEvent listener
for all the Buttons*/
private class BtnListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent evt) {
        // Need to determine which button fired the event.
        // the getActionCommand() returns the Button's label
        String btnLabel = evt.getActionCommand();
```

```
        if (btnLabel.equals("Count Up")) {
            ++count;
        } else if (btnLabel.equals("Count Down")) {
            --count;
        } else {
            count = 0;
        }
        tfCount.setText(count + "");
    }
}
```

Sử dụng getSource() của EventObject

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

public class AWTCounter3ButtonsgetSource extends Frame {
    private TextField tfCount;
    private Button btnCountUp, btnCountDown, btnReset;
    private int count = 0;

    // Constructor to setup the GUI components and event handlers
    public AWTCounter3ButtonsgetSource () {
        setLayout(new FlowLayout());
        add(new Label("Counter"));
        tfCount = new TextField("0", 10);
        tfCount.setEditable(false);
        add(tfCount);

        // Construct Buttons
        btnCountUp = new Button("Count Up");
        add(btnCountUp);
        btnCountDown = new Button("Count Down");
        add(btnCountDown);
        btnReset = new Button("Reset");
        add(btnReset);
    }
}
```

```
// Allocate an instance of inner class BtnListener.  
BtnListener listener = new BtnListener();  
  
// Use the same listener instance to all the 3 Buttons.  
btnCountUp.addActionListener(listener);  
btnCountDown.addActionListener(listener);  
btnReset.addActionListener(listener);  
  
setTitle("AWT Counter");  
setSize(400, 200);  
setVisible(true);  
}  
public static void main(String[] args) { //The entry main method  
    new AWTCounter3ButtonsGetSource(); //Let the constructor do the job  
}  
/** BtnListener is a named inner class used as ActionEvent listener  
for all the Buttons*/  
  
private class BtnListener implements ActionListener {  
  
    @Override  
    public void actionPerformed(ActionEvent evt) {  
  
        // Need to determine which button has fired the event.  
        Button source = (Button)evt.getSource();  
  
        // Get a reference of the source that has fired the event.  
        // getSource() returns a java.lang.Object. Downcast back to Button.  
        if (source == btnCountUp) {  
            ++count;  
        } else if (source == btnCountDown) {  
            --count;  
        } else {  
            count = 0;  
        }  
        tfCount.setText(count + "");  
    }  
}
```

### 7.2.8. Adapter Classes của Event Listener

Trong trường hợp chỉ cần định nghĩa một số phương thức bắt sự kiện trong số các phương thức mà giao diện Listener yêu cầu triển khai, có thể sử dụng lớp Adapter.

Lớp Adapter định nghĩa sẵn các phương thức mà giao diện Interface yêu cầu triển khai với nội dung rỗng.

Khi sử dụng lớp Adapter, chỉ cần định nghĩa đè lên các phương thức cần dùng.

#### 7.2.8.1. Sử dụng WindowAdapter thay cho WindowListener

*Sử dụng WindowListener Interface:*

```
import java.awt.*;
import java.awt.event.*;
// An AWT GUI program inherits the top-level container java.awt.Frame
public class WindowEventDemoWithInnerClass extends Frame {
    private TextField tfCount;
    private Button btnCount;
    private int count = 0;
    // Constructor to setup the GUI components and event handlers
    public WindowEventDemoWithInnerClass () {
        setLayout(new FlowLayout());
        add(new Label("Counter"));
        tfCount = new TextField("0", 10);
        tfCount.setEditable(false);
        add(tfCount);
        btnCount = new Button("Count");
        add(btnCount);
        btnCount.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent evt) {
                ++count;
                tfCount.setText(count + "");
            }
        });
    }
}
```

```
// Allocate an anonymous instance of an anonymous inner class
// that implements WindowListener.

// "super" Frame adds this instance as WindowEvent listener.

    addWindowListener(new WindowListener() {

        @Override
        public void windowClosing(WindowEvent evt) {
            System.exit(0); // terminate the program
        }

        // Need to provide an empty body for compilation

        @Override
        public void windowOpened(WindowEvent evt) { }

        @Override
        public void windowClosed(WindowEvent evt) { }

        @Override
        public void windowIconified(WindowEvent evt) { }

        @Override
        public void windowDeiconified(WindowEvent evt){}
        @Override
        public void windowActivated(WindowEvent evt) { }

        @Override
        public void windowDeactivated(WindowEvent evt){}
    });

    setTitle("WindowEvent Demo");
    setSize(400, 200);
    setVisible(true);
}

public static void main(String[] args) { //The entry main method
    new WindowEventDemoWithInnerClass();
    //Let the constructor do the job
}
}
```

### Sử dụng WindowAdapter Superclass

```
import java.awt.*;
import java.awt.event.*;
// An AWT GUI program inherits the top-level container java.awt.Frame
public class WindowEventDemoAdapter extends Frame {
    private TextField tfCount;
    private Button btnCount;
    private int count = 0;
// Constructor to setup the GUI components and event handlers
    public WindowEventDemoAdapter () {
        setLayout(new FlowLayout());
        add(new Label("Counter"));
        tfCount = new TextField("0", 10);
        tfCount.setEditable(false);
        add(tfCount);
        btnCount = new Button("Count");
        add(btnCount);
        btnCount.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent evt) {
                ++count;
                tfCount.setText(count + "");
            }
        });
// Allocate an anonymous instance of an anonymous inner class
// that extends WindowAdapter.
// "super" Frame adds the instance as WindowEvent listener.
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent evt) {
                System.exit(0); // Terminate the program
            }
        });
    }
}
```

```
        setTitle("WindowEvent Demo");
        setSize(400, 200);
        setVisible(true);
    }

/** The entry main method */
public static void main(String[] args) {
    new WindowEventDemoAdapter(); //Let the constructor do the job
}
}
```

### 7.2.9. Layout Managers và Panel

- Layout: các thức sắp xếp các phần tử (component) trên cửa sổ. Trên một cửa sổ Frame chỉ được chọn một layout.
- Khi cần sử dụng nhiều layout khác nhau trên cửa sổ, cần sử dụng đói tượng lớp Panel. Panel là một lớp thừa thứ cấp (secondary).
- Các layout trong AWT: FlowLayout, GridLayout, BorderLayout, Box,...

*Phương thức setLayout() của Container:*

Một container có một phương thức setLayout() để set layout manager:

```
// java.awt.Container
public void setLayout(LayoutManager mgr)
```

*Ví dụ:*

```
// Allocate a Panel (container)
Panel pnl = new Panel();
// Allocate a new Layout object. The Panel container sets to this layout.
pnl.setLayout(new FlowLayout());
// The Panel container adds components in the proper order.
pnl.add(new JLabel("One"));
pnl.add(new JLabel("Two"));
pnl.add(new JLabel("Three"));
.....
```

*Container's getLayout() method:*

```
Panel pnl = new Panel();
System.out.println(pnl.getLayout());
// java.awt.FlowLayout[hgap=5,vgap=5,align=center]
```

*Panel's Initial Layout:*

```
public void Panel(LayoutManager layout)
// Construct a Panel in the given layout
// By default, Panel (and JPanel) has FlowLayout
// For example, create a Panel in BorderLayout
Panel pnl = new Panel(new BorderLayout());
```

#### 7.2.9.1. FlowLayout: java.awtFlowLayout



- Các phần tử được sắp xếp từ trái qua phải theo thứ tự trong mã nguồn.
- Khi hết chiều ngang trên một hàng, các phần tử tiếp theo tự động xuống theo chiều ngang.

*Constructors:*

```
public FlowLayout();
public FlowLayout(int alignment);
public FlowLayout(int alignment, int hgap, int vgap);
// alignment: FlowLayout.LEFT (or LEADING),
// FlowLayout.RIGHT (or TRAILING), or FlowLayout.CENTER
// hgap, vgap: horizontal/vertical gap between the components
// By default: hgap = 5, vgap = 5, alignment = FlowLayout.CENTER
```

Ví dụ:

```
import java.awt.*;
import java.awt.event.*;
// An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTFlowLayoutDemo extends Frame {
    private Button btn1, btn2, btn3, btn4, btn5, btn6;
    // Constructor to setup GUI components and event handlers
    public AWTFlowLayoutDemo () {
        setLayout(new FlowLayout());
        // "super" Frame sets layout to FlowLayout, which arranges the components
        // from left-to-right, and flow from top-to-bottom.
        btn1 = new Button("Button 1");
        add(btn1);
        btn2 = new Button("This is Button 2");
        add(btn2);
        btn3 = new Button("3");
        add(btn3);
        btn4 = new Button("Another Button 4");
        add(btn4);
        btn5 = new Button("Button 5");
        add(btn5);
        btn6 = new Button("One More Button 6");
        add(btn6);
        setTitle("FlowLayout Demo"); // "super" Frame sets title
        setSize(300, 150); // "super" Frame sets initial size
        setVisible(true); // "super" Frame shows
    }
    public static void main(String[] args) { //The entry main() method
        new AWTFlowLayoutDemo(); // Let the constructor do the job
    }
}
```

### 7.2.9.2. GridLayout: java.awt.GridLayout

- Hiển thị theo dạng lưới, từ trái sang, từ trên xuống.



*Constructors:*

```
public GridLayout(int rows, int columns);
public GridLayout(int rows, int columns, int hgap, int vgap);
// By default: rows = 1, cols = 0, hgap = 0, vgap = 0
```

*Ví dụ:*

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

// An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTGridLayoutDemo extends Frame {
    private Button btn1, btn2, btn3, btn4, btn5, btn6;
    // Constructor to setup GUI components and event handlers
    public AWTGridLayoutDemo () {
        setLayout(new GridLayout(3, 2, 3, 3));
        // sets layout to 3x2 GridLayout, horizontal and vertical gaps of 3 pixels
        // The components are added from left-to-right, top-to-bottom
        btn1 = new Button("Button 1");
        add(btn1);
        btn2 = new Button("This is Button 2");
        add(btn2);
        btn3 = new Button("3");
        add(btn3);
        btn4 = new Button("Another Button 4");
    }
}
```

```
        add(btn4);

        btn5 = new Button("Button 5");
        add(btn5);

        btn6 = new Button("One More Button 6");
        add(btn6);

        setTitle("GridLayout Demo"); // "super" Frame sets title
        setSize(300, 150); // "super" Frame sets initial size
        setVisible(true); // "super" Frame shows

    }

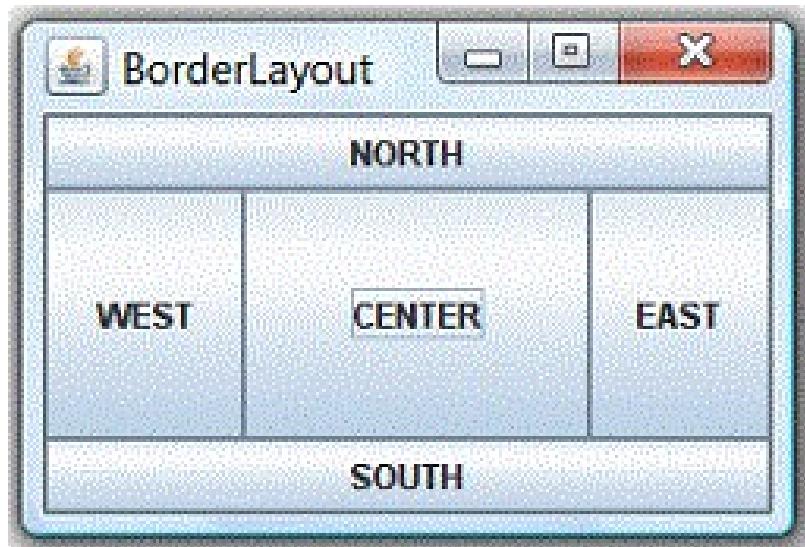
// The entry main() method

public static void main(String[] args) {
    new AWTGridLayoutDemo(); // Let the constructor do the job
}

}
```

### 7.2.9.3. BorderLayout: java.awt.BorderLayout

- Container chia thành 5 vùng: EAST, WEST, SOUTH, NORTH và CENTER.



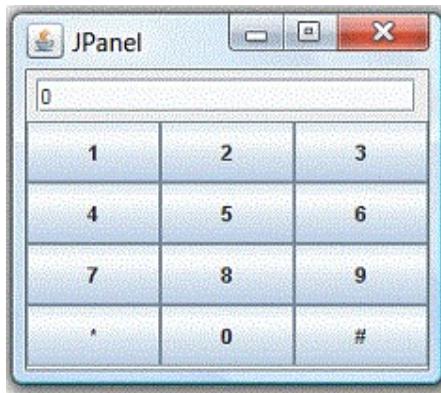
*Constructors:*

```
public BorderLayout();
public BorderLayout(int hgap, int vgap);
// By default hgap = 0, vgap = 0
```

Ví dụ:

```
import java.awt.*;
import java.awt.event.*;
// An AWT GUI program inherits the top-level container java.awt.Frame
public class AWTBorderLayoutDemo extends Frame {
    private Button btnNorth, btnSouth, btnCenter, btnEast, btnWest;
// Constructor to setup GUI components and event handlers
    public AWTBorderLayoutDemo () {
        setLayout(new BorderLayout(3, 3));
        // "super" Frame sets layout to BorderLayout,
        // horizontal and vertical gaps of 3 pixels
        // The components are added to the specified zone
        btnNorth = new Button("NORTH");
        add(btnNorth, BorderLayout.NORTH);
        btnSouth = new Button("SOUTH");
        add(btnSouth, BorderLayout.SOUTH);
        btnCenter = new Button("CENTER");
        add(btnCenter, BorderLayout.CENTER);
        btnEast = new Button("EAST");
        add(btnEast, BorderLayout.EAST);
        btnWest = new Button("WEST");
        add(btnWest, BorderLayout.WEST);
        setTitle("BorderLayout Demo"); // "super" Frame sets title
        setSize(300, 150); // "super" Frame sets initial size
        setVisible(true); // "super" Frame shows
    }
    public static void main(String[] args) { // The entry main() method
        new AWTBorderLayoutDemo(); //Let the constructor do the job
    }
}
```

#### 7.2.9.4. Sử dụng Panels như Sub-Container để quản lý các Components



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

// An AWT GUI program inherits the top-level container java.awt.Frame

public class AWTPanelDemo extends Frame {
    private Button[] btnNumbers; // Array of 10 numeric Buttons
    private Button btnHash, btnStar;
    private TextField tfDisplay;

    // Constructor to setup GUI components and event handlers
    public AWTPanelDemo () {
        // Set up display panel
        Panel panelDisplay = new Panel(new FlowLayout());
        tfDisplay = new TextField("0", 20);
        panelDisplay.add(tfDisplay);

        // Set up button panel
        Panel panelButtons = new Panel(new GridLayout(4, 3));
        btnNumbers = new Button[10];
        // Construct an array of 10 numeric Buttons
        btnNumbers[1] = new Button("1"); // Construct Button "1"
        panelButtons.add(btnNumbers[1]); //The Panel adds this Button
        btnNumbers[2] = new Button("2");
        panelButtons.add(btnNumbers[2]);
        btnNumbers[3] = new Button("3");
```

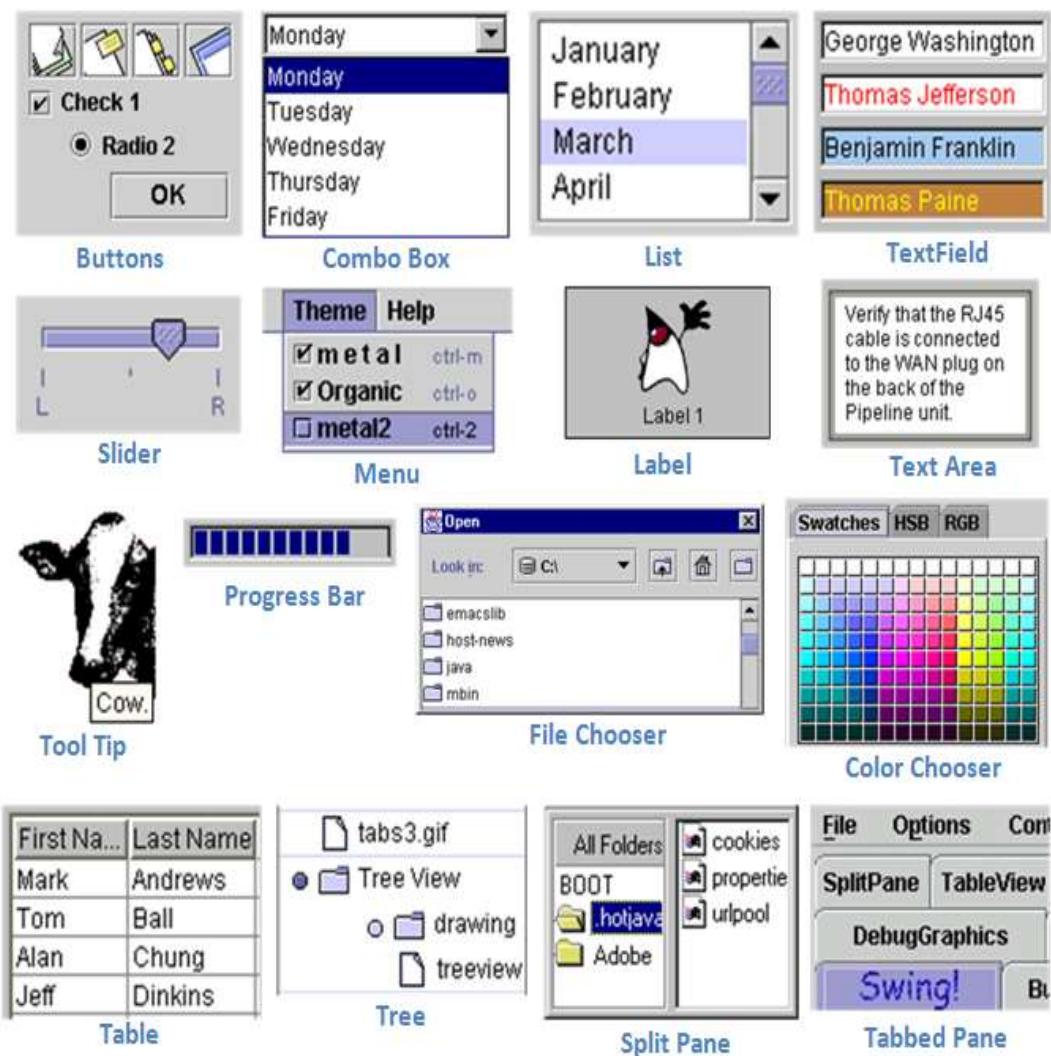
```
panelButtons.add(btnNumbers[3]);  
btnNumbers[4] = new Button("4");  
panelButtons.add(btnNumbers[4]);  
btnNumbers[5] = new Button("5");  
panelButtons.add(btnNumbers[5]);  
btnNumbers[6] = new Button("6");  
panelButtons.add(btnNumbers[6]);  
btnNumbers[7] = new Button("7");  
panelButtons.add(btnNumbers[7]);  
btnNumbers[8] = new Button("8");  
panelButtons.add(btnNumbers[8]);  
btnNumbers[9] = new Button("9");  
panelButtons.add(btnNumbers[9]);  
// You should use a loop for the above statements!!!  
btnStar = new Button("*");  
panelButtons.add(btnStar);  
btnNumbers[0] = new Button("0");  
panelButtons.add(btnNumbers[0]);  
btnHash = new Button("#");  
panelButtons.add(btnHash);  
setLayout(new BorderLayout()); // sets to BorderLayout  
add(panelDisplay, BorderLayout.NORTH);  
add(panelButtons, BorderLayout.CENTER);  
setTitle("BorderLayout Demo"); // "super" Frame sets title  
setSize(300, 300); // "super" Frame sets initial size  
setVisible(true); // "super" Frame shows  
}  
  
public static void main(String[] args) { // The entry main() method  
    new AWTPanelDemo(); // Let the constructor do the job  
}  
}
```

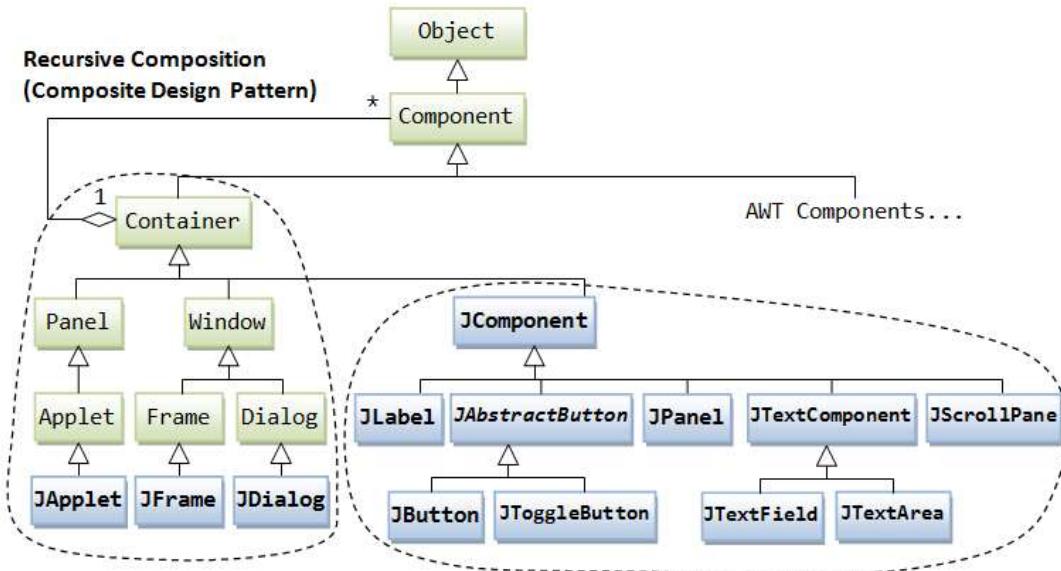
### 7.3. Swing

#### 7.3.1. Giới thiệu

- Swing cung cấp 18 gói có thể sử dụng xây dựng giao diện đồ họa.
- Thường sử dụng lệnh `import javax.swing.*` để chương trình ngắn gọn.
- Ưu điểm của Swing so với AWT:
  - Cung cấp thêm các đối tượng mới để xây dựng giao diện đồ họa.
  - look-and-feel: tùy biến để các thành phần giao diện của Swing nhìn giống như các thành phần giao diện của hệ điều hành.
  - Hỗ trợ các thao tác sử dụng bàn phím thay chuột.
  - Sử dụng tài nguyên hiệu quả hơn.

#### 7.3.2. Một số thành phần của Swing





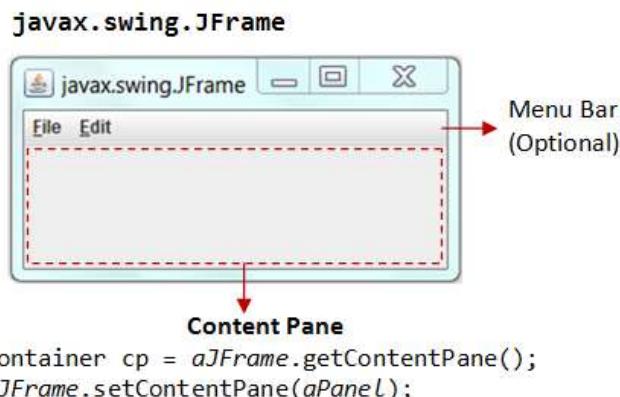
Hình 7.14. Các phần tử của Swing

#### **Top-Level và Secondary Containers trong Swing:**

- Giống như AWT, Swing cũng yêu cầu một top-level container. Có 3 top-level containers trong Swing:
  - JFrame: sử dụng cho những ứng dụng main window (icon, title, minimize/maximize/close, tùy chọn menu-bar, và content-pane).
  - JDialog: secondary pop-up window (title, close, và content-pane).
  - JApplet: sử dụng cho ứng dụng applet (content-pane).

#### **Content-Pane của Top-Level Container trong Swing:**

Tuy nhiên không giống AWT, JComponents sẽ không được thêm vào top-level container (JFrame, JApplet) trực tiếp. JComponents phải được thêm vào content-pane của top-level container. Content-pane thực sự là một java.awt.Container.



1. **get content-pane** thông qua **getContentPane()** từ top-level container, và thêm components vào nó.

*Ví dụ:*

```
public class SwingDemo extends JFrame {  
    // Constructor  
    public SwingDemo() {  
        // Get the content-pane of this JFrame, which is a java.awt.Container  
        // All operations, such as setLayout() and add() operate on the content-pane  
        Container cp = getContentPane();  
        cp.setLayout(new FlowLayout());  
        cp.add(new JLabel("Hello, world!"));  
        cp.add(new JButton("Button"));  
        ....  
    }  
    ....  
}
```

2. **set content-pane** vào JPanel thông qua **setContentPane()** của JFrame.

```
public class SwingDemo extends JFrame {  
    // Constructor  
    public SwingDemo() {  
        // The "main" JPanel holds all the GUI components  
        JPanel mainPanel = new JPanel(new FlowLayout());  
        mainPanel.add(new JLabel("Hello, world!"));  
        mainPanel.add(new JButton("Button"));  
        // Set the content-pane of this JFrame to the main JPanel  
        setContentPane(mainPanel);  
        ....  
    }  
    ....  
}
```

**Lưu ý:** Nếu một thành phần được thêm trực tiếp vào một khung JFrame, nó sẽ được thêm vào khung nội dung của JFrame thay thế. Nghĩa là,

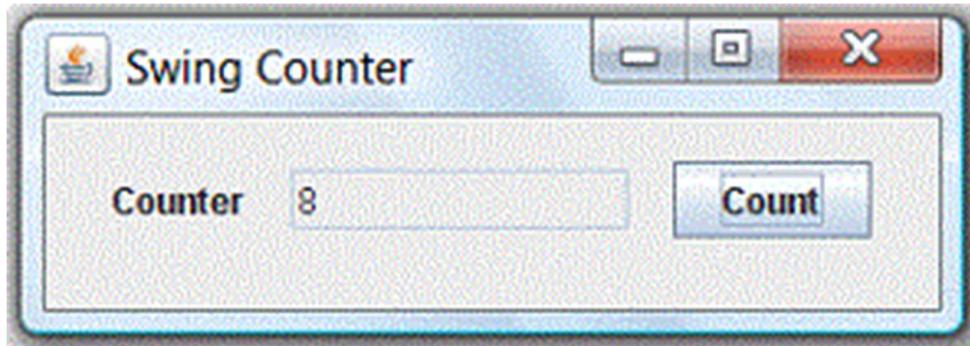
```
// Suppose that "this" is a JFrame  
add(new JLabel("add to JFrame directly"));  
// is executed as  
getContentPane().add(new JLabel("add to JFrame directly"));
```

### 7.3.3. Swing Program Template

```
import java.awt.*; // Using AWT layouts  
import java.awt.event.*; //Using AWT event classes, listener interfaces  
import javax.swing.*; // Using Swing components and containers  
//A Swing GUI application inherits from top-level container javax.swing.JFrame  
public class ..... extends JFrame {  
    // Private instance variables  
    // .....  
    // Constructor to setup the GUI components and event handlers  
    public .....() {  
        // Retrieve the top-level content-pane from JFrame  
        Container cp = getContentPane();  
        // Content-pane sets layout  
        cp.setLayout(new ....Layout());  
        // Allocate the GUI components // .....  
        // Content-pane adds components  
        cp.add(....);  
        // Source object adds listener // .....  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        // Exit the program when the close-window button clicked  
        setTitle("....."); // "super" JFrame sets title  
        setSize(300, 150); // "super" JFrame sets initial size  
        setVisible(true); // "super" JFrame shows  
    }  
}
```

```
// The entry main() method
public static void main(String[] args) {
    // Run GUI codes in Event-Dispatching thread for thread-safety
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new .....(); // Let the constructor do the job
        }
    });
}
```

#### 7.3.4. Ví dụ: SwingCounter



```
import java.awt.*; // Using AWT layouts
import java.awt.event.*; // Using AWT event classes and listener interfaces
import javax.swing.*; // Using Swing components and containers
// A Swing GUI application inherits from top-level container javax.swing.JFrame
public class SwingCounter extends JFrame { // JFrame instead of Frame
    private JTextField tfCount;
    // Use Swing's JTextField instead of AWT's TextField
    private JButton btnCount;
    // Using Swing's JButton instead of AWT's Button
    private int count = 0;
    // Constructor to setup the GUI components and event handlers
    public SwingCounter() {
```

```
// Retrieve the content-pane of the top-level container JFrame
// All operations done on the content-pane

Container cp = getContentPane();
cp.setLayout(new FlowLayout());
// The content-pane sets its layout
cp.add(new JLabel("Counter"));
tfCount = new JTextField("0");
tfCount.setEditable(false);
cp.add(tfCount);
btnCount = new JButton("Count");
cp.add(btnCount);

// Allocate an anonymous instance of an anonymous inner class that
// implements ActionListener as ActionEvent listener
btnCount.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent evt) {
        ++count;
        tfCount.setText(count + "");
    }
});

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// Exit program if close-window button clicked
setTitle("Swing Counter"); // "super" JFrame sets title
setSize(400, 200); // "super" JFrame sets initial size
setVisible(true); // "super" JFrame shows

}

// The entry main() method
public static void main(String[] args) {
/** Run the GUI construction in the Event-Dispatching thread for
thread-safety*/
```

```
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {
        new SwingCounter(); // Let the constructor do the job
    }
});
```

### 7.3.5. Ví dụ: SwingAccumulator

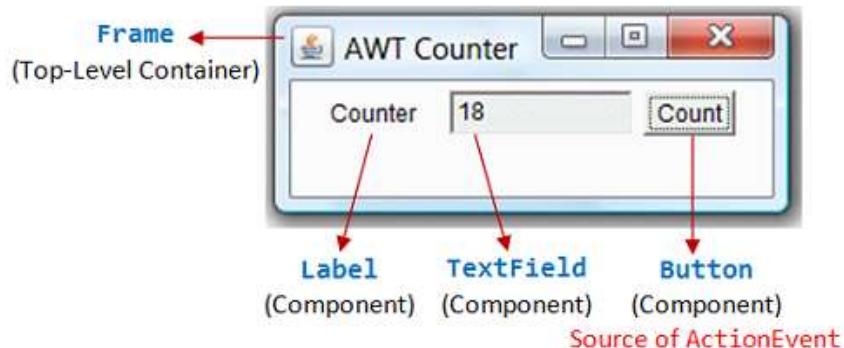
```
import java.awt.*; // Using layouts
import java.awt.event.*; // Using AWT event classes and listener interfaces
import javax.swing.*; // Using Swing components and containers
// A Swing GUI application inherits the top-level container javax.swing.JFrame
public class SwingAccumulator extends JFrame {
    private JTextField tfInput, tfOutput;
    private int sum = 0; // accumulated sum, init to 0
    // Constructor to setup the GUI components and event handlers
    public SwingAccumulator() {
        // Retrieve the content-pane of the top-level container JFrame
        // All operations done on the content-pane
        Container cp = getContentPane();
        cp.setLayout(new GridLayout(2, 2, 5, 5));
        // The content-pane sets its layout
        cp.add(new JLabel("Enter an Integer: "));
        tfInput = new JTextField("10");
        cp.add(tfInput);
        cp.add(new JLabel("The Accumulated Sum is: "));
        tfOutput = new JTextField("0");
        tfOutput.setEditable(false); // read-only
        cp.add(tfOutput);
    }
}
```

```
// Allocate an anonymous instance of an anonymous inner class that
// implements ActionListener as ActionEvent listener

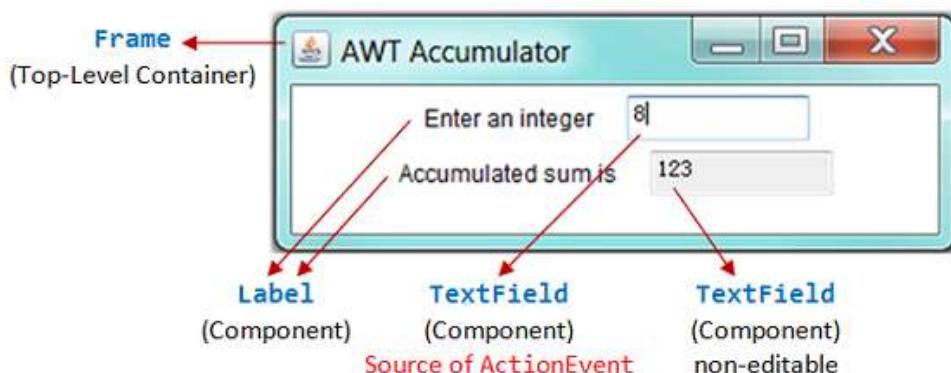
    tfInput.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent evt) {
            // Get the String entered into the input TextField, convert to int
            int numberIn = Integer.parseInt(tfInput.getText());
            sum += numberIn;
            //accumulate numbers entered into sum
            tfInput.setText("");
            tfOutput.setText(sum + "");
            // display sum on the output TextField
        }
    });
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // Exit program if close-window button clicked
    setTitle("Swing Accumulator"); // "super" Frame sets title
    setSize(400, 150); // "super" Frame sets initial size
    setVisible(true); // "super" Frame shows
}
public static void main(String[] args) { // The entry main() method
/** Run the GUI construction in the Event-Dispatching thread for
thread-safety*/
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new SwingAccumulator();
            // Let the constructor do the job
        }
    });
}
```

## BÀI TẬP CHƯƠNG 7

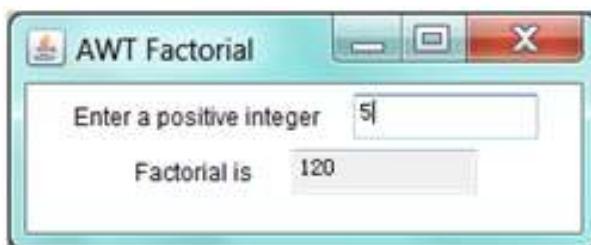
Bài 7-1. Viết chương trình sử dụng *AWT* như sau:



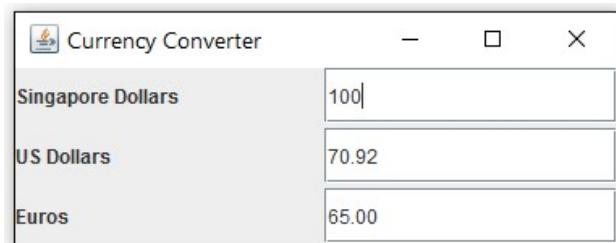
Bài 7-2. Viết chương trình sử dụng *AWT* như sau:



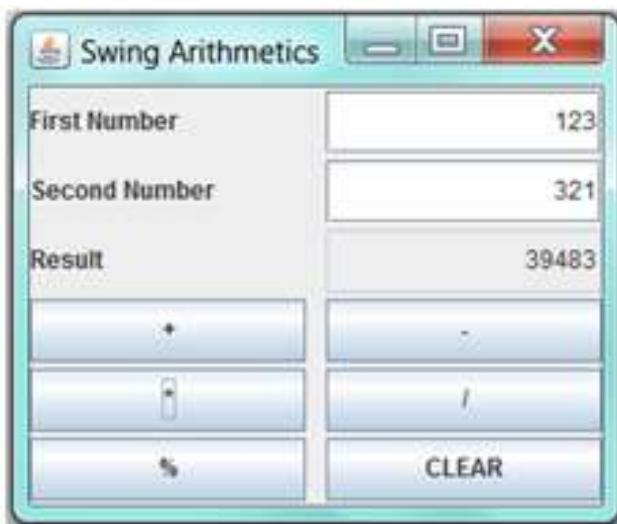
Bài 7-3. Viết chương trình sử dụng *AWT* tính giá trị  $n!$  như sau:



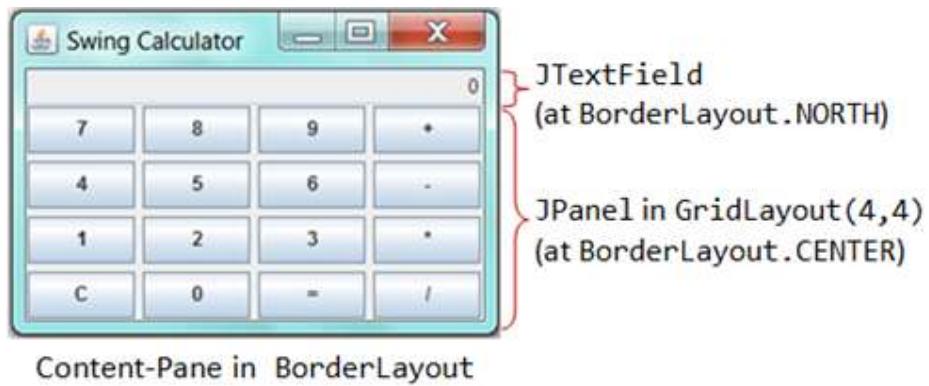
Bài 7-4. Viết chương trình sử dụng *Swing* như sau:



Bài 7-5. Viết chương trình sử dụng **Swing** như sau:



Bài 7-6. Viết chương trình sử dụng **Swing** như sau:



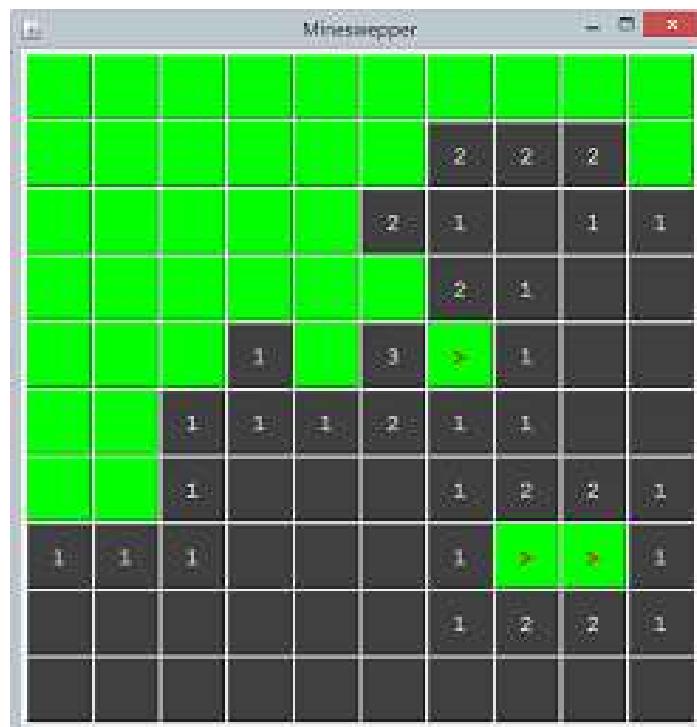
Bài 7-7. Viết chương trình Giải phương trình bậc 2 như sau:



Bài 7-8. Viết chương trình *Sudoku* sử dụng *Eclipse/NetBeans GUI Builder* như sau:



Bài 7-9. Viết chương trình *Minesweeper* sử dụng *Eclipse/NetBeans* như sau:



## TÀI LIỆU THAM KHẢO

- [1]. C.Thomas Wu, *An Introduction to Object-Oriented Programming with Java<sup>TM</sup>*, 5<sup>th</sup> Edition, McGraw-Hill. 2010.
- [2]. Y. Daniel Liang, *Introduction to Java Programming*, 10<sup>th</sup> Edition, Prentice Hall, 2015.
- [3]. Stuart Reges, Marty Stepp, *Building Java Programs: A Back to Basics Approach*, 4<sup>th</sup> Edition, Pearson, 2016.
- [4]. <http://www.oracle.com/technetwork/java/javase/documentation/index.html>
- [5]. The online Java tutorial <http://docs.oracle.com/javase/tutorial/>
- [6]. <https://www.ntu.edu.sg/home/ehchua/programming/index.html>