MỤC LỤC

[CHƯƠNG 1 5](#_Toc436137891)

[TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH JAVA 5](#_Toc436137892)

[1.1 Giới thiệu về ngôn ngữ Java 5](#_Toc436137893)

[1.1.1 Các ứng dụng của Java 5](#_Toc436137894)

[1.1.2 Biên dịch và thực thi chương trình Java 6](#_Toc436137895)

[1.1.3 Xây dựng chương trình Java đầu tiên 7](#_Toc436137896)

[1.1.4 Công cụ và môi trường lập trình 8](#_Toc436137897)

[1.2 Biến và cách sử dụng biến 8](#_Toc436137898)

[1.2.1 Định nghĩa 8](#_Toc436137899)

[1.2.2 Khai báo biến trong Java 9](#_Toc436137900)

[1.3 Các kiểu dữ liệu cơ sở 10](#_Toc436137901)

[1.3.1 Kiểu số nguyên 10](#_Toc436137902)

[1.3.2 Kiểu dấu chấm động 10](#_Toc436137903)

[1.3.3 Kiểu ký tự 10](#_Toc436137904)

[1.3.4 Kiểu logic (Boolean) 11](#_Toc436137905)

[1.4 Hằng 11](#_Toc436137906)

[1.5 Lệnh và khối lệnh 11](#_Toc436137907)

[1.5.1 Khái niệm 11](#_Toc436137908)

[1.5.2 Phạm vi của biến 11](#_Toc436137909)

[1.6 Toán tử và biểu thức 12](#_Toc436137910)

[1.6.1 Phép gán 12](#_Toc436137911)

[1.6.2 Toán tử toán học 12](#_Toc436137912)

[1.6.3 Toán tử tăng, giảm 13](#_Toc436137913)

[1.6.4 Phép toán quan hệ 14](#_Toc436137914)

[1.6.5 Phép toán logic 16](#_Toc436137915)

[1.6.6 Toán tử ép kiểu 16](#_Toc436137916)

[1.7 Cấu trúc điều khiển 17](#_Toc436137917)

[1.7.1 Cấu trúc rẽ nhánh 17](#_Toc436137918)

[1.7.2 Cấu trúc lặp 21](#_Toc436137919)

[1.8 Lớp bảo vệ dữ liệu cơ sở 25](#_Toc436137920)

[1.9 Kiểu dữ liệu mảng 25](#_Toc436137921)

[1.9.1 Khai báo mảng 26](#_Toc436137922)

[1.9.2 Cấp phát mảng 26](#_Toc436137923)

[1.9.3 Khởi tạo 27](#_Toc436137924)

[1.9.4 Truy cập mảng 27](#_Toc436137925)

[CHƯƠNG 2. 28](#_Toc436137926)

[LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA 28](#_Toc436137927)

[2.1 Tổng quan về hướng đối tượng (Trang 21) 28](#_Toc436137928)

[2.1.1 Sơ lược lịch sử lập trình 28](#_Toc436137929)

[2.1.2 Đối tượng 28](#_Toc436137930)

[2.1.3 Java và hướng đối tượng 28](#_Toc436137931)

[2.1.4 Sự kế thừa và thiết kế OOP 28](#_Toc436137932)

[2.2 Khái niệm về lớp (Class) 28](#_Toc436137933)

[2.2.1 Khai báo, định nghĩa lớp 28](#_Toc436137934)

[2.2.2 Tạo đối tượng của lớp 29](#_Toc436137935)

[2.2.3 Thuộc tính của lớp 29](#_Toc436137936)

[2.2.4 Hàm - phương thức của lớp 30](#_Toc436137937)

[2.2.5 Phương thức nạp chồng 32](#_Toc436137938)

[2.2.6 Constructor (Toán tử khởi tạo) 33](#_Toc436137939)

[2.3 Đặc điểm của lập trình hướng đối tượng 34](#_Toc436137940)

[2.3.1 Đóng gói 34](#_Toc436137941)

[2.3.2 Tính đa hình (Cần xem lại) 35](#_Toc436137942)

[2.3.3 Tính kế thừa 35](#_Toc436137943)

[2.4 Gói (Package) trong Java 36](#_Toc436137944)

[CHƯƠNG 3. 39](#_Toc436137945)

[XÂY DỰNG GIAO DIỆN NGƯỜI DÙNG 39](#_Toc436137946)

[3.1 Tổng quan về xây dựng giao diện người dùng 39](#_Toc436137947)

[3.2 Giới thiệu thư viện AWT, SWING 40](#_Toc436137948)

[3.2.1 Java AWT 40](#_Toc436137949)

[3.2.2 Java Swing 41](#_Toc436137950)

[3.3 Xây dựng giao diện với AWT 43](#_Toc436137951)

[3.3.1 Các gói trong Java AWT 43](#_Toc436137952)

[3.3.2 Các lớp Container 43](#_Toc436137953)

[3.3.3 Các lớp Component 45](#_Toc436137954)

[3.4 Xây dựng giao diện với Swing 48](#_Toc436137955)

[3.4.1 Các gói trong Java Swing 48](#_Toc436137956)

[3.4.2 Swing Container 49](#_Toc436137957)

[3.4.3 Swing Component 49](#_Toc436137958)

[3.5 Xử lý sự kiện trong Java 62](#_Toc436137959)

[3.5.1 Giới thiệu 62](#_Toc436137960)

[3.5.2 Sử dụng sự kiện 64](#_Toc436137961)

[3.5.3 Các sự kiện cơ bản 66](#_Toc436137962)

[CHƯƠNG 4. 67](#_Toc436137963)

[LẬP TRÌNH CƠ SỞ DỮ LIỆU VỚI JDBC 67](#_Toc436137964)

[4.1 Kiến trúc của JDBC 67](#_Toc436137965)

[4.2 Các khái niệm cơ bản 69](#_Toc436137966)

[4.2.1 JDBC Driver 69](#_Toc436137967)

[4.2.2 JDBC URL 71](#_Toc436137968)

[4.3 Kết nối cơ sở dữ liệu với JDBC 71](#_Toc436137969)

[4.3.1 Cấu hình SQL Server 71](#_Toc436137970)

[4.3.2 Kết nối với SQL Server 77](#_Toc436137971)

[4.4 Các thao tác cơ bản trên CSDL 79](#_Toc436137972)

[4.4.1 Tạo kết nối với dữ liệu 79](#_Toc436137973)

[4.4.2 Thao tác truy vấn với dữ liệu 81](#_Toc436137974)

[4.4.3 Cập nhật dữ liệu dữ liệu 81](#_Toc436137975)

[DANH SÁCH BÀI TẬP LỚN JAVA 84](#_Toc436137976)

# CHƯƠNG 1

# TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH JAVA

## Giới thiệu về ngôn ngữ Java

Java là một ngôn ngữ lập trình được Sun Microsystems giới thiệu vào tháng 6 năm 1995. Từ đó, nó đã trở thành một công cụ lập trình của các lập trình viên chuyên nghiệp.

Java được xây dựng trên nền tảng của C và C++. Do vậy nó sử dụng các cú pháp của C và các đặc trưng hướng đối tượng của C++.

Ban đầu Java được thiết kế để làm ngôn ngữ viết chương trình cho các sản phẩm điện tử dân dụng như đầu video, tivi, điện thoại, máy nhắn tin.. . Tuy nhiên với sự mãnh mẽ của Java đã khiến nó nổi tiếng đến mức vượt ra ngoài sự tưởng tượng của các nhà thiết kế ra nó.

Java khởi thuỷ tên là Oak- là cây sồi mọc ở phía sau văn phòng của nhà thiết kế chính ông Jame Gosling, sau này ông thấy rằng đã có ngôn ngữ lập trình tên Oak rồi, do vậy nhóm thiết kế quyết định đổi tên, “Java” là cái tên được chọn, Java là tên của một quán cafe mà nhóm thiết kế java hay đến đó uống.

### Các ứng dụng của Java

* + - 1. *Java và ứng dụng Console*

Các chương trình này chạy từ dấu nhắc lệnh và không sử dụng giao diện đồ  
họa. Các thông tin nhập xuất được thể hiện tại dấu nhắc lệnh.

* + - 1. *Java và ứng dụng Applet*

Applet là chương trình Java được tạo ra để sử dụng trên Internet thông qua  
các trình duyệt hỗ trợ Java như IE hay Netscape. Applet được nhúng bên trong trang Web. Khi trang Web hiển thị trong trình duyệt, Applet sẽ được tải về và thực  
thi tại trình duyệt.

* + - 1. *Java và ứng dụng đồ hoạ AWT, SWING*

Đây là các chương trình Java chạy độc lập cho phép người dùng tương tác  
qua giao diện đồ họa.

* + - 1. *Java và ứng dụng Web*

Java cũng cũng được sử dụng trong thương mại điện tử và ứng dụng web. Rất nhiều Webservices dạng RESTfull được tạo ra sử dụng Spring MVC, Struts 2.0 và những framework tương tự. Ngay cả Servlet đơn giản, trang JSP hay ứng dụng sử dụng Struts khá phổ biến trên các dự án Chính phủ khác nhau. Nhiều chính phủ, các tổ chức y tế, bảo hiểm, giáo dục, quốc phòng … có ứng dụng web đượcxây dựng bằng Java.

* + - 1. *Java và ứng dụng nhúng*

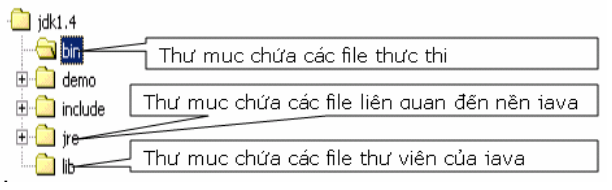
Java rất mạnh trong ứng dụng nhúng. Đôi khi những phần mềm điều kiển màn hình, bàn phím, máy giặt, tủ lạnh, điều hòa… được viết băng Java mà chúng ta không để ý. Bạn chỉ cần khoảng 130KB để có thể sử dụng công nghệ Java trên các thiết bị phần cứng (thẻ thông minh hoặc các cảm biến). Ban đầu, Java được thiết kết cho các ứng dụng nhúng này. Tư tưởng “Write once – Run Anywhere” – viết 1 lần chạy mọi nơi được Java tích hợp cả vào những thiết bị này. Và có vẻ như những thiết bị này đang đền đáp lại cho Java một chỗ đứng vững chắc.

### Biên dịch và thực thi chương trình Java

Để có thể biên dịch và chạy các chương trình Java ta phải cài:

- *JRE (Java Runtime Enviroment)* môi trường thực thi của java, nó bao gồm:  
JVM (Java Virtual Machine) máy ảo Java vì các chương trình Java được thông  
dịch và chạy trên máy ảo java và tập các thư viện cần thiết để chạy các ứng  
dụng Java.

- *Bộ công cụ biên dịch và thông dịch JDK* của Sun Microsystem. Sau khi cài đặt JDK (giả sử thư mục cài đặt là C:\JDK1.4) ta sẽ nhận được một cấu  
trúc thư mục như sau:



Để biên dịch một chương trình java sang mã byte code ta dùng lệnh:

*C:\JDK1.4\BIN\javac* ***filename****.java*

Để thông dịch và chạy chương trình ta sử dụng lệnh:

*C:\JDK1.4\BIN\java* ***filename***

### Xây dựng chương trình Java đầu tiên

Ví dụ đầu tiên: chương trình Hello World (chương trình khi chạy sẽ in ra màn hình lời chào Hello World).

***Các bước:***

Mở một chương trình soạn thảo văn bản hỗ trợ ASCII*, như notepad, wordpad,  
EditPlus*… và gõ vào các dòng sau

public class HelloWorld {

public static void main(String[] args){

System.out.println("Hello World");

}

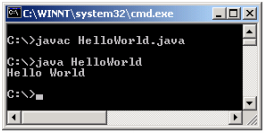
}

Ghi lại với cái tên C:\HelloWorld.java (chú ý tên tệp phải trùng với tên lớp, kể cả chữ hoa chữ thường, phần mở rộng là java)

Mở của sổ ***DOS Prompt***

Chuyển vào thư mục ***C:\***

Gõ lệnh ***javac HelloWorld.java*** để biên dịch chương trình, nếu việc biên dịch thành công (chương trình không có lỗi cú pháp) thì ta sẽ thu được tệp ***HelloWorld.class*** trong cùng thư mục, nếu trong chương trình còn lỗi cú pháp thì trong bứơc này ta sẽ nhận được một thông báo lỗi và lúc này tệp ***HelloWorld.class*** cũng không được tạo ra.



Gõ lệnh ***java HelloWorld*** (chú ý không gõ phần mở rộng) để chạy chương trình HelloWorld.

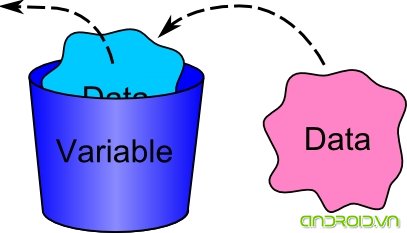
Sau khi thông dịch và chạy ta nhận được kết quả.

### Công cụ và môi trường lập trình

Để làm việc với lập trình Java, chúng ta có rất nhiều IDE khác nhau. Trong khuôn khổ tài liệu này, chúng ta sẽ cùng sử dụng Netbeans.

Để tải và cài đặt Netbeans, ta có thể truy cập qua địa chỉ website dưới đây: https://netbeans.org/downloads/

## Biến và cách sử dụng biến



### Định nghĩa

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

Mỗi biến gắn liền với một kiểu dữ liệu và một định danh duy nhất gọi là tên biến.

Tên biến thông thường là một chuỗi các ký tự (Unicode), ký số.

Tên biến phải bắt đầu bằng một chữ cái hoặc một dấu gạch dưới “\_” hay dấu dollar “$”.

Tên biến không có khoảng trắng ở giữa tên.

Trong Java, biến có thể được khai báo ở bất kỳ nơi đâu trong chương trình.

Tên biến không được trùng với các từ khóa trong Java. (Ví dụ từ khóa: *if, else, while, do, case, switch, int, float,...*)

***Ví dụ:***

Tên biến đúng: ***a , \_a, A, \_b, \_B, $d, hoTen, \_giaTri, sinhVien1, sinhVien2***.

Tên biến sai: ***5a , hoc sinh, 1gia tri, if, try***

**Lưu ý**: Trong Java phân biệt chữ hoa chữ thường, vì vậy chúng ta cần lưu ý khi đặt tên cho các biễn, các đối tương dữ liệu cũng như các xử lý trong chương trình

### Khai báo biến trong Java

Cấu trúc câu lệnh khai báo biến trong java như sau: ***[Kiểu dữ liệu] [tên biến];***

// Khái báo biến có tên là “giaTri”, kiểu dữ liệu là int – kiểu số nguyên.

int giaTri;

//Khai báo biến có tên là “hoTen”, kiểu dữ liệu là String – Là một chuỗi ký tự.

String hoTen;

Ngoài ra còn có thêm từ khóa (***public***, ***private***, ….) trước dòng khai báo biến (vd: private String hoTen), phần này mình sẽ nói khi chúng ta sang phần hướng đối tượng trong Java.

Để gán giá trị cho biến ta chỉ việc dùng cú pháp Tên biến = giá trị, hoặc gán ngay trong quá trình khai báo ví dụ:

int giaTri;

giaTri = 5;

hoặc

int giaTri = 5;

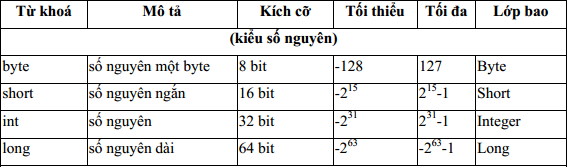
Để in một chuỗi văn bản hoặc giá trị ra màn hình Console ta dùng lệnh dạng như sau:

System.out.print(“Giá trị của biến là: ”+ giaTri1 + giaTri2);

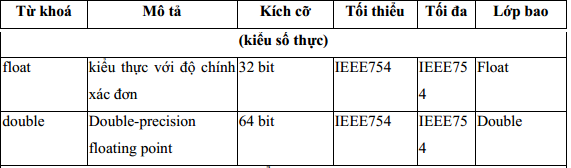
// giaTri1 và giaTri2 là 2 biến đã khai báo và gán giá trị.

## Các kiểu dữ liệu cơ sở

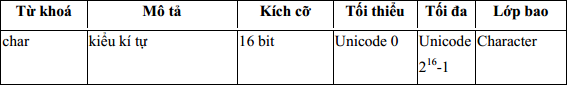
### Kiểu số nguyên



### Kiểu dấu chấm động



### Kiểu ký tự



### Kiểu logic (Boolean)



## Hằng

Hằng là một giá trị thực được sử dụng trong chương trình, được biểu diễn như chính nó chứ không là một giá trị của một biến hay một kết quả của một biểu thức.

Ví dụ: *Các giá trị sau có thể coi là một hằng: 3.1415 thay cho số Pi, số 65 được coi là tuổi về hưu.*

## Lệnh và khối lệnh

### Khái niệm

*Câu lệnh (lệnh):* là một chỉ thị yêu cầu máy phải thực hiện và được kết thúc bằng dấu chấm phẩy “;”.

*Khối lệnh:* Một dãy các khai báo cùng với các câu lệnh nằm trong cặp dấu móc đơn { } được gọi là một khối lệnh.

### Phạm vi của biến

Mỗi biến được khai báo ra có một phạm vi hoạt động, phạm vi của biến là  
nơi mà biến có thể được truy cập, điều này xác định cả tính thấy được và thời gian  
sống của biến.

*Biến phạm vi lớp* là biến được khai báo bên trong lớp nhưng bên ngoài các  
phương thức và hàm tạo, tuy nhiên việc khai báo phải xuất hiện trước khi biến được sử dụng

*Biến phạm vi cục bộ* là biến được khai báo bên trong một khối, phạm vi của  
biến tính từ điểm biến được khai báo cho đến cuối khối mà biến được khai báo.

int n = 0;

n = n + 1; //n = 1

{

n = 10;

int m = 10;

n = m + 1; //n = 11

}

m = m + 1; //Sai cú pháp

## Toán tử và biểu thức

### Phép gán

*Cú pháp:* ***Biến = BiểuThức;***

Phép gán được thực hiện bằng toán tử “=”, nó có nghĩa là “hãy tính toán giá trị biểu thức bên phải dấu gán, sau đó đưa giá trị đó vào ô nhớ có tên nằm ở bên trái dấu gán”

Chú ý:

Câu lệnh gán gồm một dấu “=”

Kiểu của biểu thức bên phải dấu gán phải tương thích với kiểu dữ liệu của biến

Trong java ta có thể thực hiện một dẫy gán như sau: i = j = 10;// cả i và j đều có giá trị 10

### Toán tử toán học

Ngôn ngữ java cũng có các phép toán số học như các ngôn ngữ khác:

Phép cộng: “+”

Phép trừ: “-“

Phép nhân: “\*”

Phép chia: “/”

Phép chia lấy phần nguyên: “%”

Ta mô tả tóm tắt các phép toán số học qua bảng tổng kết sau:

|  |  |  |
| --- | --- | --- |
| **Phép toán** | **Sử dụng** | **Mô tả** |
| - | op1 + op2 | Cộng op1 với op2 |
| - | op1 – op2 | Trừ op1 cho op2 |
| \* | op1 \* op2 | Nhân op1 với op2 |
| / | op1 / op2 | chia op1 cho op2 |
| % | op1 % op2 | Tính phần dư của phép chia op1 cho op2 |

### Toán tử tăng, giảm

Giống như ngôn ngữ C/C++, java cũng có phép toán tăng, giảm, ta có thể mô tả tóm tắt qua các bằng sau:

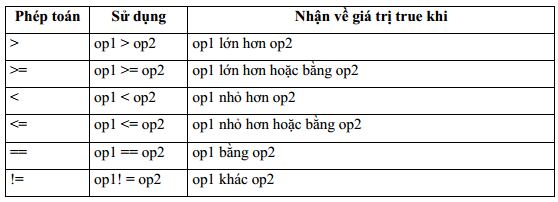
|  |  |  |
| --- | --- | --- |
| Phép toán | Sử dụng | Mô tả |
| ++ | op++ | Tăng op lên 1 đơn vị, giá trị của op được tăng lên trước khi biểu thức chứa nó được tính |
| ++ | ++ op | Tăng op lên 1 đơn vị, giá trị của op được tăng lên sau khi biểu thức chứa nó được tính |
| -- | op-- | Giảm op xuống1 đơn vị, giá trị của op được giảm xuống trước khi biểu thức chứa nó được tính |
| -- | -- op | Giảm op xuống1 đơn vị, giá trị của op được giảm xuống sau khi biểu thức chứa nó được tính |

**Chú ý:** nếu toán tử tăng trước, tăng sau(giảm trước, giảm sau) đứng một mình(không nằm trong biểu thức ) thì chúng hoạt động như nhau, chúng chỉ khác nhau khi chúng nằm trong biểu thức.

### Phép toán quan hệ

Phép toán quan hệ bao giờ cũng cho kết quả boolean, phép toán quan hệ sẽ so sánh 2 giá trị, nó xác định mối quan hệ giữa chúng, ví dụ! = sẽ trả về true nếu 2 toán hạng là khác nhau.

Ta tóm tắt các phép toán qua bảng sau:

******

***Ví dụ:***

//Ví dụ sử dụng các phép toán quan hệ

public class RelationalDemo {

public static void main(String[] args) {

// a few numbers

int i = 37;

int j = 42;

int k = 42;

System.out.println("Variable values...");

System.out.println(" i = " + i);

System.out.println(" j = " + j);

System.out.println(" k = " + k);

//greater than

System.out.println("Greater than...");

System.out.println(" i > j = " + (i > j)); // false

System.out.println(" j > i = " + (j > i));// true

System.out.println(" k > j = " + (k > j));// false, they are equal

//greater than or equal to

System.out.println("Greater than or equal to...");

System.out.println(" i >= j = " + (i >= j));// false

System.out.println(" j >= i = " + (j >= i));// true

System.out.println(" k >= j = " + (k >= j));// true

//less than

System.out.println("Less than...");

System.out.println(" i < j = " + (i < j));// true

System.out.println(" j < i = " + (j < i));// false

System.out.println(" k < j = " + (k < j));// false

//less than or equal to

System.out.println("Less than or equal to...");

System.out.println(" i <= j = " + (i <= j));// true

System.out.println(" j <= i = " + (j <= i));// false

System.out.println(" k <= j = " + (k <= j));// true

//equal to

System.out.println("Equal to...");

System.out.println(" i == j = " + (i == j));// false

System.out.println(" k == j = " + (k == j));// true

//not equal to

System.out.println("Not equal to...");

System.out.println(" i! = j = " + (i! = j));// true

System.out.println(" k! = j = " + (k! = j));// false

}

}

Đây là đầu ra của chương trình:

Variable values...

i = 37

j = 42

k = 42

Greater than...

i > j = false

j > i = true

k > j = false

Greater than or equal to...

i >= j = false

j >= i = true

k >= j = true

Less than...

i < j = true

j < i = false

k < j = false

Less than or equal to...

i <= j = true

j <= i = false

k <= j = true

Equal to...

i == j = false

k == j = true

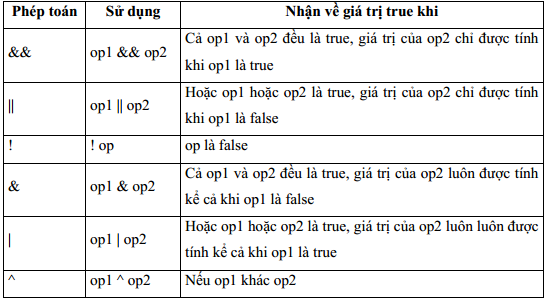
Not equal to...

i! = j = true

k! = j = false

### Phép toán logic

Java hỗ trợ 6 phép toán logic được chỉ ra trong bảng sau:

******

***Nhận xét:***

Phép toán && ( & ) chỉ nhận giá trị true khi và chỉ khi cả hai toán hạng đều là true.

Phép toán || ( | ) chỉ nhận giá trị false khi và chỉ khi cả hai toán hạng là false

Phép toán ^ chỉ nhận giá trị true khi và chỉ khi hai toán hạng khác nhau

### Toán tử ép kiểu

Có 2 kiểu ép kiểu:

* Ép kiểu rộng: *(widening conversion)*: từ kiểu nhỏ sang kiểu lớn (không mất mát thông tin)
* Ép kiểu hẹp: *(narrow conversion)*: từ kiểu lớn sang kiểu nhỏ (Có khả năng mất mát thông tin.

Cú pháp: ***<tên biến> = (kiểu\_dữ\_liệu) <tên\_biến>;***

***Ví dụ:***

**public** **class** EpKieu {  
    **public** **static** **void** main(String[] args) {  
        **float** soThuc;  
        **int** soNguyen;  
   
        soThuc = 10.6f;  
        soNguyen = (int) soThuc;  
        System.**out**.println("Số thực vào là: " + soThuc);  
        System.**out**.println("Số nguyên ép kiểu từ số thực  là: " + soNguyen);  
    }  
}

## Cấu trúc điều khiển

Chương trình là một dẫy các lệnh được bố trí thực hiện theo một trình tự nào đó, nhưng đôi khi ta muốn điều khiển luồng thực hiện của chương trình tuỳ thuộc vào điều kiện nào đó. Ngôn ngữ lập trình java cung cấp một số cấu trúc cho phép ta điều khiển luồng thực kiện của chương trình, chúng ta sẽ cùng tìm hiểu.

### Cấu trúc rẽ nhánh

##### Cấu trúc if- else

* *Dạng khuyết*
* Cấu trúc dạng khuyết

if (Condition)

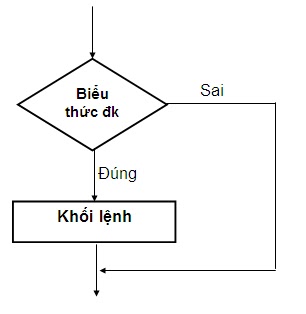
{

// Các lệnh sẽ được thực hiện nếu giá trị

// của Condition là true

}

* Sơ đồ khối cấu trúc if – else khuyết:



* *Dạng đầy đủ*
* Cấu trúc

if (Condition)

{

// Các lệnh sẽ được thực hiện nếu giá trị

// của Condition là true

}

else

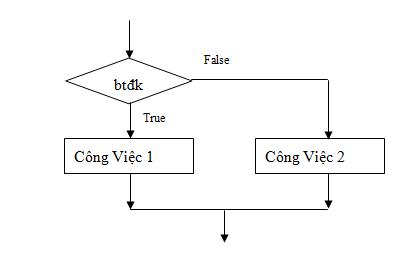
{

// Các lệnh sẽ được thực hiện nếu giá trị

// của Condition là false

}

* Sơ đồ khối cấu trúc if – else dạng đầy đủ



***Ví dụ:***

int a = 10;

int b = 0;

if(a == 10)

{

b = 1;

}

else

{

b = 0;

}

System.out.print(“Giá trị của b là: ” + b);

##### Phép toán ?

* Cú pháp:

**(condition) ? Operation1 : Operation2;**

Nếu điều kiện *condition* có giá trị là true lệnh sẽ trả về giá trị của biểu thức Operation1, ngược lại sẽ trả về giá trị của biểu thức Operation2.

***Ví dụ***: Với ví dụ trên ta có thể viết lại như sau:

int a = 10;

int b = 0;

b = (a==10) ? 1 : 0;

System.out.print(“Giá trị của b là: ” + b);

##### Cấu trúc Switch

* *Dạng khuyết*

**switch**(biểu\_thức)

{

**case** gt\_1: lệnh 1; [ **break**;]

**case** gt\_2: lệnh 2; [ **break**;]

…

**case** gt\_n: lệnh n; [ **break**;]

}

* *Dạng đủ*

switch(biểu\_thức)

{

case gt\_1: lệnh 1; [ break;]

case gt\_2: lệnh 2; [ break;]

…

case gt\_n: lệnh n; [ break;]

default: lệnh n+1;

}

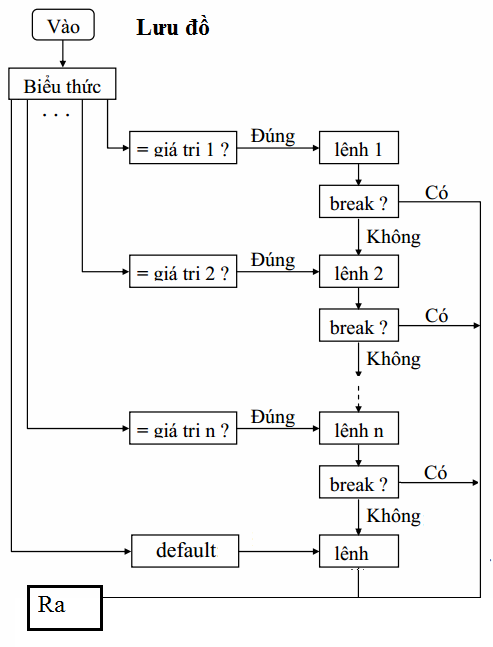
**Chú ý:**

- *biểu\_thức* phải là một biểu thức có kiểu *char, byte, short, int* nhưng không thể là kiểu *long*, nếu *biểu\_thức* có kiểu khác với các kiểu liệt kê ở trên thì java sẽ đưa ra  
một thông báo lỗi.

- Nếu *biểu\_thức* bằng giá trị của *gt\_i* thì các lệnh từ lệnh i cho đến lệnh n nếu không có *default* (lệnh n+1 nếu có *default*) sẽ được thực hiện.

- Câu lệnh *break:* Báo rằng sau khi thực hiện khối lệnh tương ứng, chương trình sẽ thoát ra khỏi cấu trúc *switch* và thực hiện lệnh kế tiếp của khối switch. Nếu trong khối lệnh i không chứa lệnh ngắt break thì chương trình thực hiện tiếp các lệnh của khối lệnh i + 1, i +2,... cho đến khi gặp lệnh ngắt *break* hay đến khi chấm dứt khối lệnh *switch*. Còn các khối lệnh i - 1, i - 2,... không tồn tại (trống) thì chương trình sẽ tự động xem như các trường hợp i - 1 , i - 2 sẽ nhận khối lệnh i để thực hiện.

Sơ đồ khối mô tả sự hoạt động của cấu trúc *switch* trong trường hợp có lệnh *break*:



*Ví dụ:*

int thang = 2;

switch(thang)

{

**case** 1: { System.out.print("Tháng 1"); break;}

**case** 2: { System.out.print("Tháng 2"); break;}

**case** 3: { System.out.print("Tháng 3"); break;}

**case** 4: { System.out.print("Tháng 4"); break;}

**case** 5: { System.out.print("Tháng 5"); break;}

**case** 6: { System.out.print("Tháng 6"); break;}

**case** 7: { System.out.print("Tháng 7"); break;}

**case** 8: { System.out.print("Tháng 8"); break;}

**case** 9: { System.out.print("Tháng 9"); break;}

**case** 10: { System.out.print("Tháng 10"); break;}

**case** 11: { System.out.print("Tháng 11"); break;}

**case** 12: { System.out.print("Tháng 12"); break;}

**default**: { System.out.print("Không phải tháng");}

}

* **Kết quả:** Tháng 2

### Cấu trúc lặp

Vòng lặp là một nguyên lý không thể thiếu trong bất kỳ ngôn ngữ lập trình nào. Không có những vòng lặp, việc giải quyết các vấn đề thực tế sẽ chỉ đạt được một giới hạn nào đó. Các cấu trúc vòng lặp trong Java cũng tương tự như trong ngôn ngữ C/C++.

##### Vòng lặp for

* Cấu trúc

for(khởi\_tạo\_biến\_đếm; đk\_lặp; bước\_nhảy)

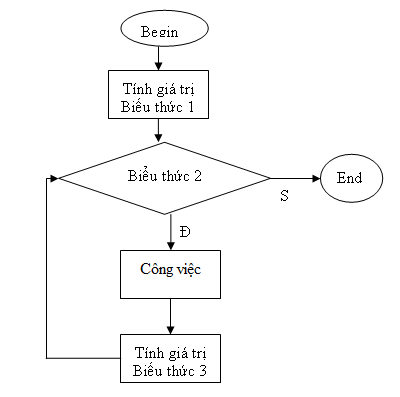
{

<khối\_lệnh>;

}

Bắt đầu với giá trị khởi tạo của biến đếm, khối lệnh được thực hiện. Sau mỗi lần thực hiện xong khối lệnh, biến đếm thay đổi giá trị một lượng bằng bước nhảy. Nếu giá trị của biến đếm thỏa biểu thức điều kiện lặp thì lại thực hiện khối lệnh. Còn không thỏa thì vòng lặp kết thúc.

* Lưu đồ vòng lặp for



*Ví dụ:*

**for**(**int** i = 0; i<= 10; i++)

{

System.out.println(“i = ” + i);

}

##### Vòng lặp while

* Cấu trúc:

while(điều\_kiện\_lặp)

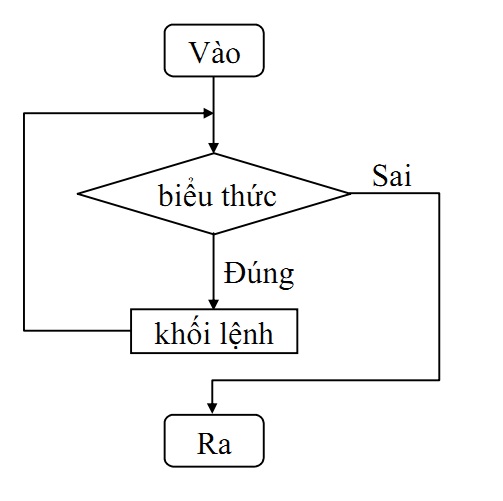
{

khối\_lệnh;

}

Khối lệnh của vòng lặp while được thực hiện khi biểu thức boolean còn có giá trị true. Trong cấu trúc này, Java không hỗ trợ các khởi tạo và bước nhảy. Ta phải chú ý thực hiện chúng trong quá trình thực hiện khối lệnh vì có thể sẽ xảy ra trường hợp lặp vô tận. Nghĩa là biểu thức boolean luôn luôn đúng (không có điều kiện thoát vòng lặp)

* Lưu đồ vòng lặp while



***Ví dụ:***

**int** n = 0;

**while**(n> 10)

{

System.out.**println**(“n = ” + n);

n++;

}

##### Vòng lặp do-while

* Cấu trúc

**do**

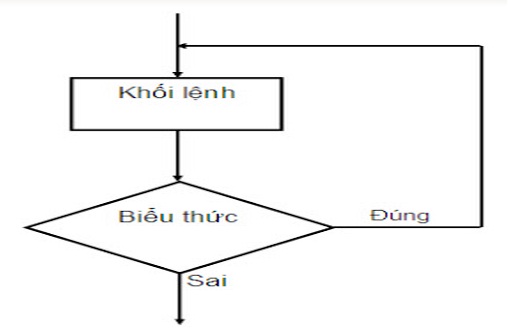
{

Khối\_lệnh;

}

**while** (<biểu thức boolean>);

* Lưu đồ

******

Vòng lặp này cho phép thực hiện khối lệnh rồi mới thực hiện kiểm tra biểu thức boolean. Nếu biểu thức còn được thỏa thì tiếp tục thực hiện khối lệnh. Lệnh này khác với cấu trúc *while* là khối lệnh được thực hiện ít nhất 1 lần.

***Ví dụ:***

// Ví dụ 1: Thực hiện 1 lần

int n = 0;

**do**

{

System.out.printf(“n = ” + n);

n++;

}

**while** (n > 10);

// Ví dụ 2: Thực hiện 10 lần

int n = 0;

**do**

{

System.out.printf(“n = ” + n);

n++;

}

**while** (n < 10);

##### Continue và break

***Break:*** Trong cấu trúc switch chúng ta dùng câu lệnh *break* để thoát thỏi cấu trúc *switch* trong cùng chứa nó. Tương tự như vậy, trong cấu trúc lặp, câu lệnh *break* dùng để thoát khỏi cấu trúc lặp trong cùng chứa nó.

Trong thực tế, đặc biệt là các bài tìm kiếm, sẽ dùng break để thoát vòng lặp khi chúng ta đã tìm thấy kết quả, tránh việc lặp tới tận cùng của vòng lặp, gây lãng phí tài nguyên, vì thế giúp chương trình chạy nhanh hơn!

***Continue:***  Chúng ta thường dùng *continue* để bỏ qua vòng lặp hiện tại khi gặp điều kiện nào đó để tiếp tục sang vòng lặp tiếp theo. Hiểu theo bản chất, *continue* là 1 lệnh nhảy, nó nhảy xuống vị trí kết thúc ("}") của khối lệnh trong vòng lặp chứa nó và bỏ qua tất cả các câu lệnh sau nó – Những câu lệnh thuộc cùng khối lệnh với continue của vòng lặp.

***Ví dụ:***

// Ví dụ về break

**int** count = 0;

**while**(true)

{

count++;

**if**(count >= 10) **break**;

System.out.println(“count = ” + count);

}

// Ví dụ về continue

**for**(**int** i =0; i< 10; i++)

{

**if**(i == 1 || i == 5)

**continue**;

System.out.println(“i = ” + i);

}

## Lớp bảo vệ dữ liệu cơ sở

Java không xem các kiểu cơ bản (primitive) là đối tượng. Java cung cấp các lớp bao để thao tác các kiểu cơ bản như các đối tượng, dữ liệu của kiểu cơ bản được bao trong một đối tượng. Mỗi kiểu dữ liệu cơ bản đều có lớp bao tương ứng trong gói java.lang.

Chú ý lớp bao cài đặt các đối tượng không thay đổi (immutable), nghĩa là sau khi trị có kiểu cơ bản được khởi gán trong đối tượng bao, không thể thay đổi trị này.

Bạn có thể khởi tạo một đối tượng lớp bao, bằng cách gọi constructor tương ứng với trị cơ bản cần bao. Ta gọi là boxing (đóng gói) một trị có kiểu cơ bản

Nếu bạn thay đổi kiểu dữ liệu cơ bản thành đối tượng tương đương (gọi là boxing), bạn cần một lớp bao. Nếu bạn lấy dữ liệu có kiểu cơ bản từ đối tượng (gọi là unboxing), bạn cần gọi các phương thức của lớp bao. Từ Java 5, Java cung cấp autoboxing cho phép bạn gán và lấy dữ liệu kiểu cơ bản mà không cần lớp bao.

## Kiểu dữ liệu mảng

Ngoài kiểu dữ liệu cơ sở như int, byte, char, string,... Java còn có loại dữ liệu dẫn xuất. Có 3 kiểu dữ liệu dẫn xuất:

* Mảng (array)
* Lớp (class)
* Giao diện (interface)

Chúng ta sẽ tìm hiểu lớp và giao diện ở phần sau. Phần này ta sẽ tập trung đến mảng trong Java.

### Khai báo mảng

Đặc trưng của mảng được đặt trong cặp ngoặc vuông “[ ]”. Một mảng được khai báo bằng 2 cách:

Cặp ngoặc vuông được đặt sau tên biến.

Cặp ngoặc vuông được đặt sau kiểu dữ liệu.

*Ví dụ:*

**int**[] intArray;

**int** intArray[];

Cả 2 khai báo trên đều dùng để khia báo một mảng số nguyên (int) tên là intArray. Thông thường ta thường dùng kiểu thứ nhất để khai báo nhiều mảng cùng loại như:

**int**[] intArray1, intArray2, intArray3;

### Cấp phát mảng

Sau khi khai báo, bản thân mảng chưa được xác định. Mảng cần được định vị trong vùng nhớ, nói cách khác cần được cấp vùng nhớ. Không như các ngôn ngữ khác, kích thước mảng không được xác định ngay khi khai báo. Nếu ta khai báo như sau:

**int** intArray[100];

Câu khai báo trên bị lỗi biên dịch. Trong Java mọi hoạt động đều thông qua đối tượng, chúng ta cần dùng từ khóa **new** để định vị cho một mảng trong vùng nhớ.

**int** intArray = **new** **int**[100]; // khai báo và cấp phát

**int** intArray;// khai báo

intArray = new **int**[100]; // cấp phát

Ở ví dụ trên, ta thấy ta có thể khai báo tên mảng và kiểu dữ liệu trước. Sau đó ta mới cấp phát một mảng gồm 100 số kiểu int để sử dụng.

### Khởi tạo

Trong java, ta cũng có thể cấp phát một mảng bằng cách liệt kê các phần tử của mảng.

**int** intArray = {1, 3, 7, 8, 2, 5};

**char** charArray = {‘a’, ‘c’, ‘d’, ‘g’, ’0’};

**string** stringArray = {“abc”, “ABC”, “c”, “fgh”};

### Truy cập mảng

Các phần tử của mảng luôn được đánh số bắt đầu từ số 0. Phần tử đầu tiên là phần tử thứ 0, và phần tử cuối cùng của mảng có n phần tử sẽ có thứ tự là n -1. Ta có thể tham khảo giá trị của một phần tử trong mảng bằng cách gán các số thứ tự của phần tử ta muốn truy cập vào giữa 2 cặp ngoặc vuông.

**int** intArray = {1, 3, 7, 8, 2, 5};

int a = intArray[1]; // a sẽ có giá trị là 3

int b = intArray[3]; // b sẽ có giá trị là 8

Trong nhiều ngôn ngữ khác, chuỗi String được xem là mảng ký tự. Trong Java điều này không còn đúng nữa, Java cho *String* là một lớp, bản thân một biến chuỗi *String* trong Java có thể xử lý nư cắt ghép chuỗi, tìm kiềm trên chuỗi,...

# CHƯƠNG 2.

# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

1. **Đầu mục**

## Tổng quan về hướng đối tượng

Lập trình hướng đối tượng (OOP – Object oriented Programming) là tư duy của mô hình lập trình hiện đại. Hướng đối tượng tạo nên một cách suy nghĩ mới, thức tế hơn và linhhoajt hơn mô hình lập trình thủ tục truyền thống. Hướng đối tượng giúp lập trình viên dễ dàng mở rộng, bổ sung cho các chương trình của mình. Chúng ta hãy theo dõi sự phát triển của việc lập trình qua các giai đoạn của ngành công nghiệp phần mềm.

### Sơ lược lịch sử lập trình

Trong những ngày đầu của ngành tin học, việc lập trình là một công việc vô cùng khó nhọc. Mỗi bước làm việc của máy đều phải được theo dõi vô cùng tỉ mỉ, cẩn thận. Ngôn ngữ dùng để lập trình vào thời gian đó là ngôn ngữ máy (Machine Language), sau đó được phát triển thành hợp ngữ (Assembly Language)

Ta biết rằng việc hướng dẫn cho một người nào đó cột dây giày trong khi họ chưa từng biết dây giày là gì thì quả là một điều khó khăn. Bạn sẽ phải chỉ từng bước: *“Cầm đầu dây bên trái, đặt lên trên đầu dây bên phải, luồn xuống bên dưới, cầm lại đầu dây bên phải,...”,...* Việc lập trình hợp ngữ cũng vậy, chúng ta phải chỉ *“Gán giá trị thanh ghi 1 vào stack, cộng giá trị của thanh ghi thứ 2 với thanh ghi thứ nhất,...”*

Đó là lập trình sơ khởi với chuỗi hành động (step – by – step programming).

Các nhà lập trình đã tạo ra các ngôn ngữ thủ tục như Passcal, C để làm dễ dàng công việc lập trình của mình. Các chuỗi hành động sẽ được tập hợp lại thành từng khối gọi là hàm (function) hay thủ tục (procedure). Các khối sẽ hoạt động như các hộp đen, người sử dụng không cần biết trong khối đó sẽ làm cụ thể những bước nào, mà chỉ cần biết khi truyền những yếu tố cần thiết như nhau ở đầu vào thì sẽ nhận được cùng một kết quả ở đầu ra. Ví dụ: hàm ***printf*** của C hay ***writeln*** của Passcal.

Tuy nhiên, một vấn đề đặt ra cho loại hình ngôn ngữ này là khi viết các hàm như vậy, ltv thực tế đã không sửa chữa gì ở dữ liệu khác với với dữ liệu cơ sở sẵn có. Chúng đã được khép kín. Và điều này gây nên nhiều khó khăn, nhất là khi chúng ta muoons thay đổi một vài giá trị bên trong của hàm (không phải giá trị trả về của hàm). Như thế chúng ta phải tạo thêm những kết nối giữa những hàm sẵn có với những hàm viết thêm cho phù hợp với yêu cầu mới (ở ngôn ngữ C có khái niệm con trỏ hàm – function pointer có khả năng giải quyết được vấn đề này) và ta có một hộp đen mới bao hàm những hộp đen con. Mỗi khi hộp đen con thay đổi, hộp đen cha lại phải kiểm tra lại, biên dịch lại. Nếu không làm vậy có thể sẽ gây ra lỗi ở hộp đen cha.

Với một ứng dụng lớn, vấn đề này gây ra nhiều khó khăn.

Một giải pháp mới là sự phát triển cấu trúc (Structured Development). Sự phát triển cấu trúc không cần phải thay đổi ngôn ngữ đã dùng, nhưng dĩ nhiên là đã cung cấp một tiến trình mới. Với phương pháp này, lập trình viên phải lên được 100% kế hoạch của chương trình trước khi viết bất kỳ dòng lệnh cụ thể nào. Khi một chương trình được lập, một lượng biểu đồ lớn phải được soạn thảo để xác lập được mối tương quan giữa các hàm với nhau và với dữ liệu. Chúng ta cũng phải xác định được lưu đồ của từng dữ liệu cần dùng. Công việc chuẩn bị nặng nhọc đó tỏ ra rất hiệu quả trong nhiều trường hợp nhưng không phải tất cả.

Lập trình hướng đối tượng (OOP) ra đời. Ngôn ngữ cần phải thay đổi. Nhưng quan trọng hơn là OOP đòi hỏi các nhà lập trình thay đổi cách nhìn nhận về lập trình.

Đối tượng là khái niệm cơ sở của OOP. Nó không chỉ là những con số, cũng không chỉ là một cấu trúc thông thường. Nó bao hàm các số liệu như là những thuộc tính riêng (properties) và các hàm như là các phương thức (methods) để nó trao đổi với các đối tượng khác. Thay vì phải chuyển cho nhau những số liệu trong hệ thống mở, nhưng bằng các hàm, trong OOP, các đối tượng dùng thông báo để gửi thông tin, dữ liệu cho nhau và để nhắc đối tượng kia phải thực hiện nhiệm vụ của mình.

OOP không là một kỹ thuật mới. Từ những năm 1970, một nhóm các nhà phát triển phần mềm của Xerox Parc đã định hướng OOP trong các sản phẩm của GUI EtherNet mà đến nay vẫn còn đang được sử dụng. Tuy nhiên OOP không được sử dụng ngay vì lúc đó chưa có nhiều việc để lam, các vấn đề của lập trình chưa được phát triển và nhất là các nhà lập trình cảm thấy phải tốn thêm sức lực trong khi không thấy được sử khác biệt trong phong cách lập trình.

Các nguyên lý làm việc đối với đối tượng là những điều đầu tiên cần phải biết của lập trình viên. Đó là một khối chưa các dữ liệu (gọi là biến, vùng hay thuộc tính) biểu diễn những tính năng của đối tượng và phương thức biểu diễn các ứng xử của đối tượng trước tác động của các đối tượng khác hay của chính nó. Đối tượng có thể được nhân bản, thêm tính năng mà không làm hư các định nghĩa cũ. Khi một chương trình cần mở rộng, chúng ta chỉ cần định nghĩa thêm các thuộc tính, các phương thức cho đối tượng sẵn có hay có thể nhân bản thêm cho phù hợp với những hoạt động mới.

Các đối tượng có thể mở rộng hay dẫn xuất từ một đối tượng khác, do đó thời gian viết lệnh sẽ giảm đi. Quan trọng hơn là việc bẫy lỗi và sửa chữa cũng giảm đi rất nhiều vì ta chỉ cần xem xét lớp mới vừa tạo ra mà thôi.

### Đối tượng

Khái niệm đối tượng không còn gì xa lạ đối với một lập trình viên. Hàng ngày, chúng ta đã làm việc với rất nhiều các đối tượng như: đồng hồ, máy tính, điện thoại, cái bàn, ghế, ... và tất cả các thực thể trong đời sống. Khi làm việc với một đối tượng, chúng ta không tách biệt số lượng và phương thức. Như khi ta mở radio, ta không cần chú ý đến số liệu (như tần số của đài phát sóng), không chú ý đến phương thức (như làm sao có âm thanh, máy chuyển đài như thế nào,...), mà ta chỉ quan tâm tới công tắc điện, chọn đà và ngồi xuống nghe nhạc.

Sử dụng OOP cũng có sự đơn giản tương tự. Ta tạo một cấu trúc chứa dữ liệu và định nghĩa các phương thức để làm việc với dữ liệu. Như thế, đối tượng tạo và kết hợp dữ liệu bằng mã lệnh. Chúng ta có một đối tượng có đủ điều kiện để tồn tại và làm việc.

Trở lại với ví dụ trên, radio là một đối tượng với dữ liệu là hãng sản xuất, nguồn điện sử dụng, các băng tần, chuyển núm âm lượng, cắm tai nghe,... đơn giản đối tượng Radio được định nghĩa trong ngôn ngữ lập trình với ngôn ngữ Java như sau:

**public** **Radio**

{

/\*Định nghĩa dữ liệu của đối tượng \*/

float speed;

double volume;

/\* hành động của đối tượng \*/

public void speedUp()

{

speed = speed + volume;

}

public void speedDown()

{

speed = speed - volume;

}

public void stop()

{

speed = 0;

}

}

Radio khi đó được gọi là một lớp. Chúng ta có nhiều loại đối tượng Radio khác nhau như Sony, Panasonic,... trong lập trình chúng được xem là thuộc về một lớp.

Ví dụ ta có một đối tượng sony thuộc vào lớp Radio

**Radio** sony = **new** **Radio**();

sony.speedUp();

* Lưu ý: Bạn hãy khoan quan tâm đến cú pháp Java, chúng ta sẽ tìm hiểu chúng trong phần tiếp theo.

### Java và hướng đối tượng

Nếu bạn nghĩ C++ là một ngôn ngữ lập trình hướng đối tượng, thì bạn sẽ rất ngạc nhiên sau khi dùng Java viết một vài chương trình, bạn sẽ cảm nhận được nhiều hơn về phần mềm hướng đối tượng này.

Java đề cập đến lớp và đối tượng thật rõ ràng và đơn giản. Chúng không chỉ có nhiều cấu trúc dữ liệu khả dụng cho người lập trình mà còn là cơ sở cho ngôn ngữ lập trình toàn diện.

Trong C++, bạn dễ dàng khai báo một lớp nhưng không bắt buộc. Thay vì vậy bạn có thể khai báo một cấu trúc hoặc một hợp nhất (union). Bạn khai báo một cụm biến phối lỏng lẻo và dùng chúng với các hàm kiểu C. Trong Java, lớp và đối tượng là trung tâm của ngôn ngữ. Bạn không thể khai báo các hàm và thủ tục. Chúng không tồn tại. Bạn cũng không thể dùng cấu trúc, hợp nhất (union) hoặc định nghĩa trước (typedefs). Chúng đã bị loại bỏ khỏi Java. Hoặc là bạn dùng lớp và đối tượng, hoặc là bạn không dùng Java.

Java cung cấp đầy đủ mọi thứ để lập trình hướng đối tượng: cấu trúc phân tầng cho lớp, sự thừa kế, đóng gói và tính đa hình. Trong một phạm vi nào đó thì nó thật sự hữu dụng và hiệu quả. Nếu chưa quen với lập trình hướng đối tượng, bạn có thể tìm thấy những khái niệm này trong các chương sau của tài liệu.

Lý do chính cho việc phát triển phần mềm hướng đối tượng, bên cạnh tính rõ ràng và đơn giản, là những đối tượng bạn phát triển sẽ được dùng lại. Java không chỉ khuyến khích mà còn yêu cầu dùng lại phần mềm. Khi viết chương trình bất kỳ nào bằng Java, cho dẫu có đơn giản, bạn cũng phải xây dựng dựa trên những lớp và phương thức của Java API.

Một khi đã bắt đầu phát triển phần mềm bằng Java, bạn 2 lựa chọn:

* Xây dựng trên những lớp bạn đã phát triển, bằng cách ấy có thể dùng lại chúng.
* Viết lại từ đầu phần mềm của bạn, sao chép và sửa đổi những thành phần hữu dụng của phần mềm đang tồn tại.

Cấu trúc hướng đối tượng của Java giúp bạn phát triển những phần mềm hữu dụng hơn, có tính kết nối hơn và đơn giản hơn.

### Sự kế thừa và thiết kế OOP

Để có thể phát triển ứng dụng theo OOP, cần tuân theo công thức sau:

* *Chia nhỏ mã lệnh thành đơn vị nhỏ nhất:* khi viết một chương trình OOP, điều ần thiết đầu tiên là phải hình dung đủ các thành phần. Sau đó chia nhỏ chúng thành các đơn vị nhỏ nhất. Ví dụ, với yếu tố viên đạn, ta có các đơn vị yếu tố nhỏ hơn là có thể cháy, kích thước viên đạn, tầm sát thương, số lượng,...
* *Xem xét các tính chất chung giữa các đơn vị trên:* bước kế tiếp là xem xét kỹ mối liên hệ giữa các đơn vị thành phần. Ví dụ trong chương trình quản lý thư viện, thủ thư và người mượn đều có tính chung là người: Có tên tuổi, có chức vụ, có địa chỉ,... vậy ta có đưa cả 2 đối tượng này vào cùng một lớp?
* *Xem xét sự khác nhau giữa các đơn vị:* ngược lại trên, ta sẽ phân tích các đơn vị có điều gì khác biệt mà không thể gom lại. Trở lại ví dụ vừa nêu, rõ ràng thủ thư là người quản lý, coi như sở hữu cuốn sách, có nhiệm vụ phải nhận lại cuốn sách đã mượn, còn người mượn thì phải trả lại cuốn sách đã mượn.
* *Tìm tính chất chung nhất của tất cả các đơn vị:* rất hiếm khi ta không thể tìm thấy một tính chất nào chung cho mọi đơn vị. Như thế, có thể có một đơn vị nào đó không dùng chung bất kỳ tính chất nào với những đơn vị khác. Ta hãy xét một chương trình trò chơi chiến đấu: viên đạ, người bảo vệ, máy bay, mìn, nhà cửa rõ ràng là rất khác nhau. Tuy vậy, ta vẫn có thể thấy những điểm chung như máy bay, người, viên đạn có thể di chuyển; còn mìn, nhà cửa, máy bay, người có thể bị nổ. Nhưng tính chất nào bao trùm lên mọi đơn vị? Có một điều rõ ràng mà bạn có thể quên đó là chúng đều có thể được vẽ trên màn hình. Như vậy chúng ta có lớp DrawObject.
* *Gom các tính chất chung và lặp lại:* ta sắp xếp các đơn vị có chung một tính chất lại với nhau và lặp lại nhiều lần như vậy cho đến mức thấp nhất.
* *Sử dụng các đối tượng để thêm vào theo yêu cầu:* Trong nhiều trường hợp, người sử dụng yêu cầu thêm các yếu tố mới, ta chỉ nên tạo những đối tượng mới hơn là định nghĩa những biến mới. Điều này dễ thấy ở những ai đã từng lập trình với các ngôn ngữ khác. Ta thường hay tạo ra một biến mới bất kỳ lúc nào ta cần. Với OOP, khi cần ta hãy tạo một đối tượng mới.

## Khái niệm về lớp (class)

Ta coi class là một khuôn mẫu (template) của một đối tượng, trong đó bao hàm các dữ liệu của đối tượng (gọi là thuộc tính ***property*** của lớp) và các phương thức tác động lên các kiểu dữ liệu đó (gọi tắt là phương thức – ***method*** của lớp)

Lớp có tính kế thừa (inheritance). Một lớp (gọi là lớp con – subclass) có thể thừa kế tất cả các thuộc tính và phương thức của một lớp khác (gọi là siêu lớp – superclass hay có thể gọi là lớp cha).

Lớp còn có tính đa hình (polymorphism) cho phép cài đặt các lớp dẫn xuất rất khác nhau từ cùng một lớp nguồn.

### Khai báo, định nghĩa lớp

Ta dùng từ khóa class trước một định danh và sau đó là một cặp ngoặc nhọn ({ }).

**class** ConNguoi

{

}

Ta đã định nghĩa một lớp tên là ***ConNguoi***. Lưu dòng lệnh trên vào một file tên là ***ConNguoi.java***, ta có thể biên dịch thành một lớp ***ConNguoi*** trong file ***.CLASS***. Và khi thông dịch (chạy chương trình) lớp này sẽ không cho ra kết quả gì vì không có gì trong thân của lớp (trong cặp dấu ngoặc { }).

### Tạo đối tượng của lớp

Class định nghĩa một kiểu dữ liệu mới, do đó việc tạo một đối tượng của class giống như tạo một biến có kiệu dữ liệu là class đó.

*Cú pháp:*

<tên lớp> <tên đối tượng> = **new** <toán tử khởi tạo của lớp>();

***Ví dụ:***

**ConNguoi** nguyen\_van\_a = **new** **ConNguoi**();

Ở phần trên ta đã định nghĩa một lớp có tên là *ConNguoi*. Trong ví dụ trên ta đã khai báo một đối tượng *ConNguoi* có tên *nguyen\_van\_a*.

### Thuộc tính của lớp

Thuộc tính của một lớp bao gồm các biến, các hằng, hay tham số nội tại của lớp đó. Ở đây, vai trò quan trọng nhất của các thuộc tính là các biến vì chúng sẽ có thể bị thay đổi trong suốt quá trình hoạt động của một đối tượng.

Các thuộc tính có thể được xác định kiểu và kiểu của chúng có thể là các kiểu dữ liệu nguyên thủy hay là một lớp đã định nghĩa từ trước.

Ta định nghĩa thuộc tính cho lớp ***ConNguoi***

**class** ConNguoi

{

**String** hoTen;

**int** tuoi;

}

Trong ví dụ trên, ta đã khai báo 2 thuộc tính cho lớp ConNguoi: *hoTen* có kiểu dữ liệu *String*, *tuoi* có kiểu dữ liệu *int*.

### Hàm - phương thức của lớp

Trong lập trình hướng đối tượng, việc ghi tuần tự các lệnh trong một chương trình sẽ làm cho chương trình khó đọc và sửa chữa. Các nhà lập trình đã mô đun hóa tập lệnh thành các khối lệnh con độc lập hay phụ thuộc nhau và cịu tác dụng hay thực hiện một việc cụ thể riêng biệt nào đó. Việc làm này giúp cho người thiết kế vào bảo trì chương trình dễ theo dõi các tiến trình để tiến hành sửa chữa, bổ sung.

Thông thường để thực hiện điều này, nhà thiết kế chương trình sẽ thực hiện việc phân tích từ trên xuống (top – down approach). Họ chia nhỏ chương trình thành từng phần với những chức năng riêng biệt, rồi tiếp tục chia nhỏ thành các phần vừa tính được thành các thành phần nhỏ hơn cho đến khi không thể chia nhỏ hơn.

Ví dụ: Khi bạn phải làm công việc vệ sinh nhà cửa, bạn sẽ tách thành: Vệ sinh phòng ăn, phòng ngủ, phòng khách,... rồi với vệ sinh phòng ăn, bạn sẽ chia nhỏ thành rửa chén bát, lau bàn ghế, lau bếp lò, v..v...

Hàm hay phương thức trong Java (và các ngôn ngữ khác) chính là các khối lệnh đơn vị thực hiện những chức năng riêng biệt.

Trong một vài ngôn ngữ, hàm được dùng có phân biệt với thủ tục (cũng là một khối lệnh thực hiện một chức năng chuyên biệt nào đó). Thủ tục là một khối lệnh khi được thực hiện không trả về bất cứ giá trị nào. Còn hàm thường nhận về một giá trị.

Vì trong Java không có một hàm nào là toàn cục mà luôn phải tương ứng với một đối tượng cụ thực hiệnể, thực hiệnể hiện hành vi ứng xử của đối tượng, nên kể từ đây ta chỉ dùng khái niệm phương thức thay cho khái niệm hàm.

Phương thức được định nghĩa như một hành động hoặc một tác vụ thật sự của đối tượng. Nó còn được định nghĩa như một hành vi mà trên đó các thao tác cần thiết được thực thi.

Phương thức là những ứng xử của lớp lên các vùng, hay lên các tác động bên ngoài. Như đã nêu, một phương thức được xác định theo cấu trúc sau:

[Tính chất bổ sung] <dữ liệu trả về> <tên phương thức> [các ngoại lệ]

{

// Nội dung phương thức

}

Các ngoại lệ sẽ được trình bày ở mục sau. Một phương thức có các tính chất bổ từ như sau:

|  |  |
| --- | --- |
| **Tính chất** | **Ý nghĩa** |
| public | Có thể truy cập từ bên ngoài lớp định nghĩa |
| protected | Chỉ được truy cập từ các lớp định nghĩa và những lớp dẫn xuất của lớp đó. |
| private | Chỉ được truy cập trong bản thân lớp định nghĩa |
| static | Một phương thức của lớp, chung cho mọi mẫu của lớp |
| abstract | Không cài đặt được gì trong lớp |
| final | Không bị định nghĩa chồng ở các lớp dẫn xuất (hằng) |
| native | Một cafid dặt phụ thuộc môi trường trong một ngôn ngữ khác, như C hay hợp ngữ. |
| synchronized | Dùng để chỉ một phương thức tới hạn ngăn các tác động của đối tượng khác lên đối tượng trong khi việc đồng bộ hóa đang thực hiện |

### Phương thức nạp chồng

Nạp chồng của một phương thức là tạo ra nhiều phương thức cùng tên. Chúng chỉ khác nhau ở các tham số. Do đó khi gọi một phương thức có khai báo nạp chồng, chúng ta phải xác định đúng tham số cần gửi theo để phương thức tương ứng được thực hiện. Vậy, việc khai báo nạp chồng luôn đi kèm với việc chọn lựa đối số tương ứng và khác biệt nhau (khác kiểu dữ liệu, khác số lượng tham số).

Ví dụ:

private float cost;

public float CalculateSalePrice()

{

return cost\*1.5f;

}

// phương thức dưới đây là một khai báo nạp chồng hợp lệ

**public** **float** CalculateSalePrice(**double** margin)

{

**return** cost\* (1 + margin);

}

// phương thức dưới đây sẽ xung đột (không hợp lệ) với phương thức trên vì có cùng kiểu dữ liệu và số lượng tham số.

**public float** CalculateSalePrice(**double** multiplier)

{

**return cost\* multiplier;**

}

// phương thức sau là hợp lệ (vì khác về số lượng tham số)

**public float** CalculateSalePrice(**double** multiplier, **double** margin)

{

**return** cost\* (multiplier + margin);

}

### Constructor (Toán tử khởi tạo)

Toán tử khởi tạo được dùng gọi tự động để khởi tạo một thể hiện (instance) của lớp. Tức là gán các giá trị ban đầu và thực hiện các công việc cần thiết trước khi đưa đối tượng đi vào hoạt động. Toán tử khởi tạo phải cùng tên với tên lớp, không có giá trị trả về (kể cả void), được gọi đến khi khởi tạo một đối tượng mới qua từ khóa ***new.***

Như vậy ta thấy toán tử tạo lập cũng có thể nạp chồng giống như phương thức.

***Ví dụ:***

public class Human

{

String name;

int age;

String address;

public Human() {}

public Human(String name, int age, String address)

{

this.name = name;

this.age = age;

this.address = address;

}

}

Một lớp của Java luôn tồn tại một biến ẩn, đặc biệt là biến ***this***. Thành phần này được sử dụng trong khi chạy và tham chiếu đến bản thân của lớp chứa nó. Ta thường thùng biến ***this*** trong toán tử tạo lập như trường hợp trên. Ở đây, các tên biến tham số cùng tên với tên với tên thuộc tính của lớp, ta dùng biến ***this*** để phân biệt đâu là biến tham số, đâu là tên thuộc tính

**this**.name = name ;

/\*

this.name chỉ đến thuộc tính của lớp

name chỉ đến biến tham số

\*/

## Đặc điểm của lập trình hướng đối tượng

### Đóng gói

Một nguyên lý quan trọng của lập trình hướng đối tượng là đóng gói (encapsulation). Đóng gói cho phép đối tượng dấu đi các chi tiết cài đặt của nó và chỉ công bố những gì đối tượng cần công bố để trao đổi với đối tượng khác.

Java dùng các từ khóa ***private, protected, public*** cho việc này.

*public*: Một thuộc tính được gọi là *public* có thể được truy cập từ tất cả các đối tượng khác.

*protected*: Một thuộc tính gọi là *protected* có thể được truy cập trong bản thân lớp đó và các lớp dẫn xuất từ nó. Những lớp khác 2 trường hợp trên không thể truy cập.

*private*: Một thuộc tính gọi là *private* thì chỉ bản thân lớp đó được truy cập. Các lớp khác không thể truy cập bao gồm cả các lớp dẫn xuất.

### Tính đa hình

Đa hình là việc triệu gọi đến các phương thức nạp chồng của đối tượng. Khi một phương thức nạp chồng được gọi, chương trình sẽ dựa vào kiểu các tham số và kiểu trả về để gọi phương thức của đối tượng cho phù hợp.

Ví dụ: Với lớp **Human** ta đã định nghĩa ở mục *2.2.6 Constructor:*

**Human** human1 = **new** **Human**(); // Đối tượng human1 được khởi tạo và không có tham số nào truyền vào qua toán tử tạo lập

**Human** human2 = **new** **Human**(“Nguyễn Văn A”, 22, “Thái Nguyên”);

// Đối tượng human2 được khởi tạo và có tham số được khởi tạo vào ban đầu có tên là Nguyễn Văn A, tuổi 22, địa chỉ Thái Nguyên.

Ví dụ 2:

**public** **class** Animal {

**public** **String** name;

**public** **int** age;

**public** **String** getName()

{

**return** "Animal name: "+ name;

}

}

**public** **class** Cat **extends** Animal{

**public** Cat(**String** name)

{

**this**.name = name;

}

@Override

**public** **String** getName()

{

**return** "Cat name: "+ name;

}

}

**public** **class** AsianCat **extends** Cat{

**public** AsianCat(String name) {

super(name);

}

@Override

**public** **String** getName() {

**return** "Asian cat: "+ name;

}

}

Trong hàm **Main** của chương trình ta khởi tạo như sau, chạy và kiểm tra kết quả

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

Cat cat1 = **new** Cat("Tom");

Cat cat2 = **new** AsianCat("Jerry");

String name1 = cat1.getName();// Tom

String name2 = cat2.getName();// Jerry

}

### Tính kế thừa

Trong Java, rất nhiều khi ta cần tạo một lớp mới (lớp con – subclass) dựa vào một lớp đã biết (lớp cha – superclass). Khi đó tất cả các tính chất của lớp cha (thuộc tính và phương thức) sẽ trở thành tính chất của lớp con. Và có thể lớp con này sẽ lại trở thành lớp cha của một lớp khác.

Ta dùng từ khóa ***extends*** để chỉ rõ lớp dẫn xuất khi định nghĩa lớp.

Mặc định trong Java, mọi lớp đều là dẫn xuất từ lớp Object. Nhưng ta có thể bỏ qua khai báo ***extends*** Object trong khi viết chương trình.

Ví dụ:

public class Human

{

String name;

int age;

String address;

// Hiển thị thông tin của đối tượng

public void ShowInfor()

{

System.out.println(“Họ tên: ” + name);

System.out.println(“Tuổi: ” + age);

System.out.println(“Địa chỉ: ” + address);

}

}

// Khai báo lớp Student kế thừa lớp Human

public class Student extends Human

{

String iClass; // Lớp của sinh viên

// Hiển thị toàn bộ thông tin của sinh viên

public void ShowAllInfo()

{

super.ShowInfor();// Gọi đến phương thức ShowInfor của lớp cha thông qua toán tử super

System.out.println(“Lớp: ”+ address);

}

// Hiển thị tên của sinh viên với tên của sinh viên là thuộc tính của lớp cha Human

public void ShowName()

{

System.out.println(“Họ tên: ”+ name);

}

}

## Gói (Package) trong Java

Ta có thể đóng gói các lớp được tạo thành trong một thư viện dùng chung (package). Việc tổ chức gói thư viện như thế cũng giống như việc tổ chức thư mục trên ổ đĩa. Gói là một nhóm các lớp mà ta có thể đặt cho chúng một tên chung tương tư như thư mục chứa các nhóm file.

Ví dụ: Với gói ***Security*** ta có thể có vô số các lớp khác như ***data, network, web, software, ...***

Để khai báo một lớp là thành phần của package ta phải khai báo lớp và dùng lệnh package, ví dụ như:

***Package Security;***

Khi đó muốn sử dụng lớp này vào chương trình ta phải gắn kèm vào tên lớp là tên gói như:

Thay vì ***import data*;** ta viết ***import Security.data;***

* **Chú ý**
* Dòng lệnh khai báo gói phải là dòng lệnh đầu tiên trong tập tin khai báo lớp
* Các tập tin khai báo lớp trong cùng gói phải được lưu trong cùng một thư mục
* Không nên khai báo toàn bộ một gói vì sẽ làm tốn bộ nhớ trong khi chỉ truy cập đến một vài lớp trong gói.

Thật sự Java tổ chức gói package như một thư mục. Bạn hãy xem quá trình biên dịch và sử dụng gói qua ví dụ sau: Giả sử bạn muốn tạo một thư viện chuyên về thực hiện công việc như tìm kiếm, sắp xếp trên mảng, cây nhị phân bằng những thuật toán phức tạp, thư viện này được đặt tên là ***MyLibrary*** sẽ chứa các lớp đối tượng như ***BubbleSort***, ***QuickSearch***, ***BinarySearch***. Bạn tạo gói ***MyLibrary*** như sau:

**package** MyLibrary;

**public** **class** BubbleSort

{

**public** **void** doSort()

{

System.out.println(“Do bubble sort completed!”);

}

}

Trước hết lưu ý, do Java bắt buộc một file chỉ được chứa một lớp public nên bạn không thể thêm nhiều lớp trong một file. Hãy lưu nội dung file trên với tên là BubbleSort.java. Biên dịch file này từ dòng lệnh bạn sẽ thu được lớp BubbleSort.class. Tiếp theo hãy tạo thêm các lớp QuickSearch.java và BinarySearch.java cho thư viện như sau:

**package** MyLibrary;

**public** **class** QuickSearch

{

**public** **void** doSearch()

{

System.out.println(“Do quick search completed!”);

}

}

**package** MyLibrary;

**public** **class** BinarySearch

{

**public** **void** doSearch()

{

System.out.println(“Do binary search completed!”);

}

}

Như vậy bạn đã được 3 lớp riêng lẻ được khai báo đóng gói trong thư viện MyLibrary. Trong các ngôn ngữ lập trình khác, thường thì thư viện chỉ là 1 file duy nhất. Ở đây ta có 3 lớp là 3 file khác nhau, vậy ta sử dụng chúng như thế nào? Chúng ta hãy thửu viết thử một đoạn chương trình thông thường truy cập lớp BubbleSort xem sao:

public class TestProgram

{

public static void main(String[] args) {

BubbleSort sort = new BubbleSort();

sort.doSort();

}

}

Ngay khi bạn biên dịch chương trình, IDE sẽ báo lỗi không tìm thấy lớp BubbleSort. Do không tìm thấy lớp BubbleSort nên phương thức doSort() cũng không thể thực hiện. Để sử dụng các lớp trong package bạn khai báo sử dụng thư viện bằng từ khóa import như sau:

import MyLibrary.\*;

public class TestProgram

{

public static void main(String[] args) {

BubbleSort sort = new BubbleSort();

sort.doSort();

}

}

Ở đây, ***MyLibrary.\**** hàm ý chỉ tất cả các lớp thuộc gói trong ***package MyLibrary.*** Nếu không muốn dùng từ khóa import, ta cũng có thể truy cập tường minh thông qua tên gói như sau:

public class TestProgram

{

public static void main(String[] args) {

MyLibrary.BubbleSort sort = new

MyLibrary.BubbleSort();

sort.doSort();

}

}

# CHƯƠNG 3.

# XÂY DỰNG GIAO DIỆN NGƯỜI DÙNG

1. **Đầu mục chương 3**

## Tổng quan về xây dựng giao diện người dùng

Giao diện đồ họa người dùng – gọi tắt là GUI (Graphical User Interface) là một thuật ngữ trong ngành công nghiệp máy tính. Đó là một cách giao tiếp với máy tính hay các thiết bị điện tử bằng hình ảnh và chữ viết thay vì chỉ là các dòng lệnh đơn thuần.

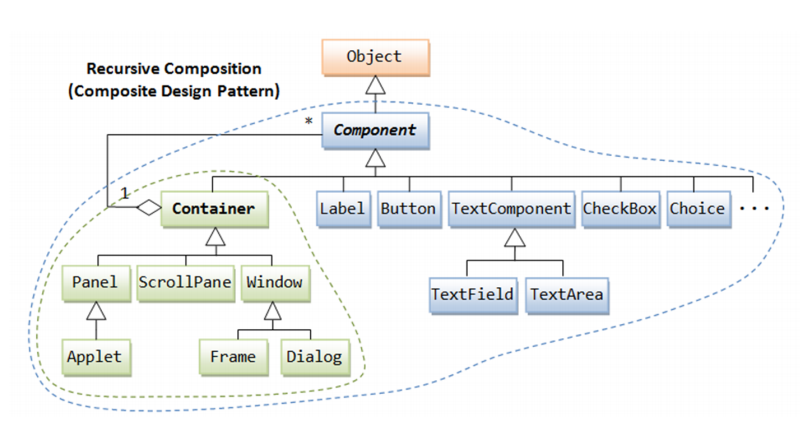
GUI cung cấp chức năng nhập liệu theo cách thân thiện với người dùng. GUI đa dạng từ ứng dụng đến ứng dụng và có thể chứa nhiều điều khiển như hộp văn bản, nhã, hộp danh sách hay các điều khiển khác. Các ngôn ngữ lập trình khác nhau cung cấp nhiều cách khác nhau để tạo GUI. Các ngôn ngữ như VB hay VC++ có thể cung cấp chức năng kéo và thả trong khi đó phần mềm giống như C++ yêu cầu người lập trình phải viết toàn bộ mã để xây dựng GUI.

Java cung cấp các gói lập trình giao diện chính sau:

* ***java.awt***: cung cấp các lớp cơ bản để lập trình giao diện
* ***javax.swing***: cung cấp các lớp mới để xây dựng giao diện chương trình dễ dàng, mềm dẻo hơn
* ***java.swt***: được phát triển bởi IBM

## Giới thiệu thư viện AWT, SWING

### Java AWT



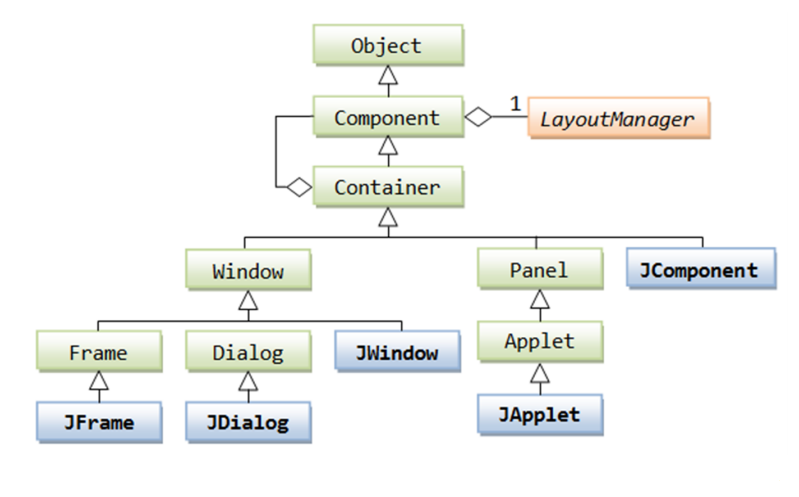
*Các thành phần cơ bản trong Java AWT*

AWT là viết tắt của Abtract Window Toolkit, cho phép xây dựng các ứng dụng dựa trên ứng dụng giao diện đồ họa người dùng. Và cho phép nhập liệu của người dùng thông qua bàn phím và chuột.

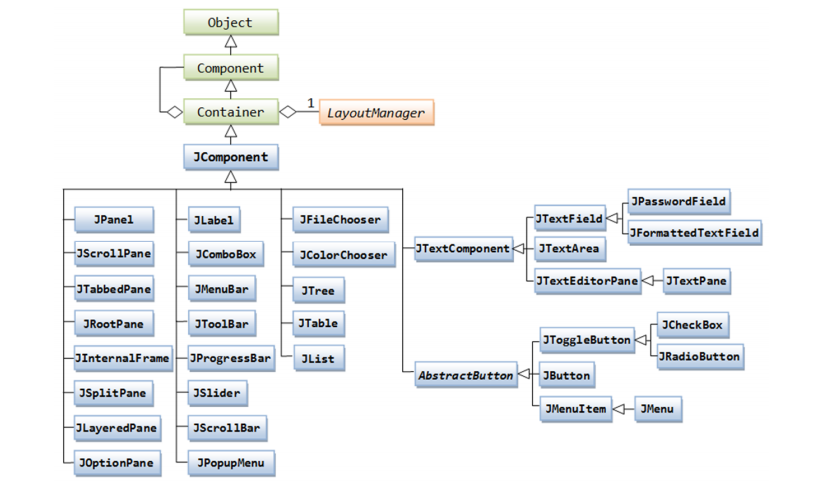
AWT cung cấp các thành phần khác nhau để tạo một giao diện hiệu quả đối với người dùng.

* *Component*: một thành phần có thể hiển thị trên màn hình đồ họa
* *Container*: thành phần chứa, bao chứa các thành phần khác. Một đối tượng Container có thể chứa các đối tượng Container khác.
* *Label*: Nhãn – cho phép người dùng hiển thị một chuỗi lên màn hình và không cho phép sửa đổi nội dung trực tiếp trên giao diện.
* *Button*: nút bấm.
* *Checkbox*: ô lựa chọn
* *TextComponent*: nhập xuất dữ liệu dạng text

### Java Swing



*Các thành phần cơ bản của Swing*

**

*Các thành phần cơ bản của Swing (tiếp)*

Swing là gói thư viện dựa trên mô hình MVC do Java cung cấp, dùng để thiết kế giao diện người dùng.

Phần lớn, các thành phần (component) của  Swing được dẫn xuất từ lớp cha là Jcomponent được thừa kế (extends) từ lớp Container của AWT. Chính vì lý do này, Swing được xem là một mô tả tốt nhất ở lớp trên của AWT hơn là thay thế cho nó.

Swing hỗ trợ công nghệ gọi là “Pluggable – Look  – And – Feel ” (PLAF), nơi mà các thành phần có thể trình bày trên mọi hệ điều hành. Nói cách khác,  trên hệ diều hành linux, một nút (button) có thể nhìn giống như trên hệ điều hành Window.

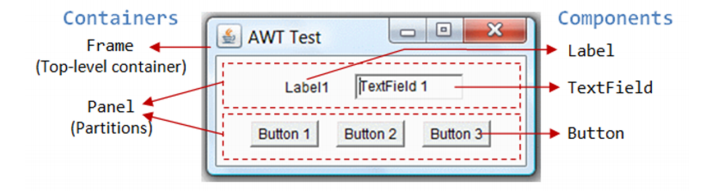
## Xây dựng giao diện với AWT

### Các gói trong Java AWT

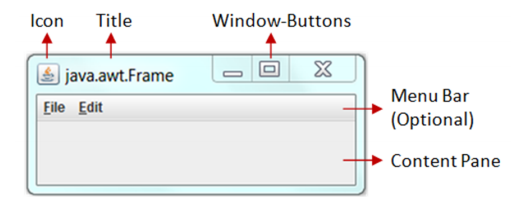
Java AWT có 12 gói cung cấp các lớp để xây dựng giao diện đồ họa (GUI). Trong đó, có 2 gói được sử dụng thường xuyên

* ***java.awt*** gồm các lớp GUI cơ bản:
  + Các lớp Component (như Button, TextComponent, Label): Component là đối tượng cơ bản tạo nên giao diện, cho phép người dùng tương tác với chương trình thông qua nó.
  + Các lớp Container – lớp chứa (Frame, Panel, Dialog, ScollPanel). Container là đối tượng chứa các component. Bản thân container cũng là một component và một container có thể chứa các container khác.
  + Các lớp Layout Manager (FlowLayout, BorderLayout, GridLayout)
  + Các lớp đồ họa tùy biến (Graphics, Color, Font).
* ***java.awt.event*** gồm các lớp xử lý sự kiện trên giao diện
  + Các lớp sự kiện (ActionEvent, MouseEvent, KeyEvent và WindowEvent)
  + Các giao diện nghe sự kiện(MouseListener, KeyListener...)
  + Các lớp Adapter (MouseAdapter, KeyAdapter)

### Các lớp Container



1. *Top – level container: Frame, Dialog và Applet*

**

* *Frame*: cửa sổ chính của giao diện chương trình.
  + Xây dựng cửa sổ chương trình bằng cách kế thừa lớp Frame hoặc kết tập 1 đối tượng Frame.
  + Kế thừa để sử dụng ngay các thành viên của Frame
* *Dialog*: cửa sổ pop-up được sử dụng để tạo ra các tương tác nằm ngoài cửa sổ chính.
* *Applet*: sử dụng trên xây dựng chương trình chạy trên trình duyệt Web

Ví dụ:

// Một chương trình với giao diện đồ họa kế thừa lớp Frame

**import** java.awt.Frame

**public** **class** MyGUIProgram extends Frame {

//Phương thức khởi tạo. Các đối tượng component cũng được tạo ra tại đây

**public** MyGUIProgram() {......}

// Các phương thức khác

// Phương thức main

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

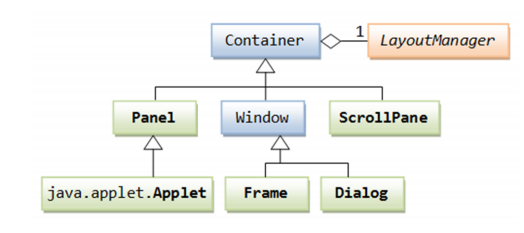
// Gọi hàm khởi tạo

**new** MyGUIProgram();

}

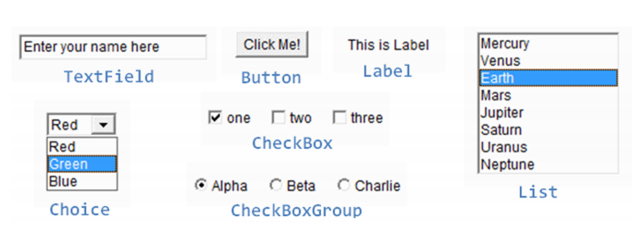
}

1. *Secondary Containers: Panel và ScrollPane*

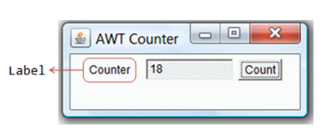


* *Panel*: khung chữ nhật nằm trong một top-level container, được sử dụng để tạo layout cho chương trình-
* *ScrollPane*: tạo ra hiệu ứng cuộn chuột (ngang/dọc) cho một component.

### Các lớp Component



1. *java.awt.Label*

**

* Chức năng: Label dùng để hiển thị một nội dung nào đó dưới dạng văn bản.
* Toán tử khởi tạo:

**public** **Label**(**String** strLabel, **int** alignment);

**public** **Label**(**String** strLabel);

**public** **Label**();

* Một số phương thức:

// Lấy nội dung văn bản của chính đối tượng label đó.

**public** **String** getText();

// Đặt nội dung văn bản cho đối tượng label đó

**public** **void** setText(String strLabel);

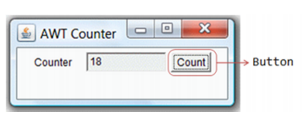
// Lấy vị trí căn lề hiện tại của label

**public** **int** getAlignment();

// đặt vị trí căn lề hiện tại của lable

**public** **void** setAlignment(int alignment);

1. *java.awt.Button*



* Tạo ra một hành động nào đó của chương trình qua sự kiện nhấp chuột
* Toán tử khởi tạo:

**public** Button(**String** buttonLabel);

**public** Button();

* Một số phương thức:

**public** **String** getLabel();

**public** **void** setLabel(**String** buttonLabel);

**public** **void** setEnable(**boolean** enablevt);

***Ví dụ:***

**Button** btnColor = new **Button**("Red");

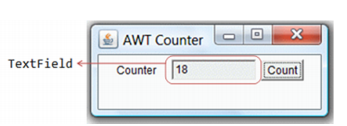
add(btnColor);

btnColor.setLabel("green");

btnColor.getLabel();

add(new **Button**("Blue")); // anynomous Button

1. *java.awt.TextField*



* Sử dụng để nhập/xuất một dòng văn bản.
* Khi đang ở trong TextField, nhấp phím Enter có thể kích hoạt một hành động nào đó của chương trình.
* Phương thức khởi tạo:

**public** **TextField**(**String** strInitText, **int** columns);

**public** **TextField**(**String** strInitText);

**public** **TextField**(**int** columns);

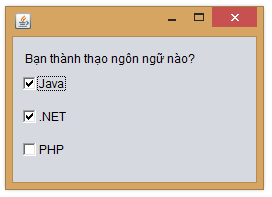
* Một số phương thức

**String** getText(); // Lấy nội dung văn bản của đối tượng

setText(**String** strText);// đặt nội dung cho đối tượng

setEditable(**boolean** editablevt); // Đặt trạng thái có cho phép người dùng thay đổi nội dung trên giao diện hay không.

1. *Checkbox*

**

* Khi ta muốn liệt kê ra danh sách các mục và cho phép người dùng chọn nhiều phần tử trong danh sách đó thì ta dùng Checkbox.
* Toán tử khởi tạo:

**public** Checkbox();

**public** Checkbox(**String** label);

**public** Checkbox(**String** label, **boolean** state);

**public** Checkbox(**String** label, **boolean** state, **CheckboxGroup** group);

* Phương thức cơ bản:

setLabel(**String** label);

setState(**boolean** state);

## Xây dựng giao diện với Swing

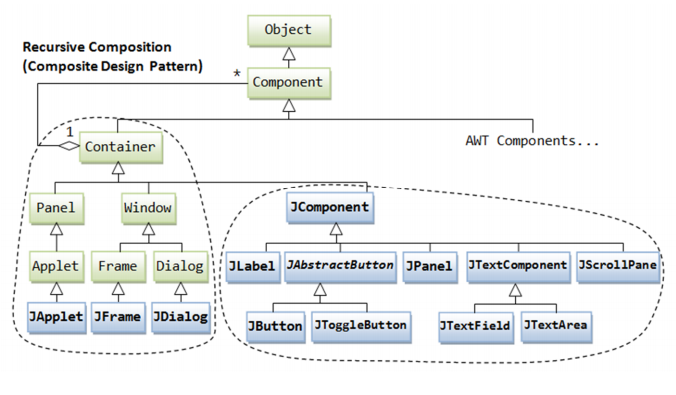
### Các gói trong Java Swing

Swing cung cấp 18 gói có thể sử dụng xây dựng giao diện đồ họa

Ta thường sử dụng thư viện ***javax.swing.\**** để chương trình trở nên 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 HĐH
  + Hỗ trợ các thao tác sử dụng bàn phím thay chuột
  + Sử dụng tài nguyên hiệu quả hơn

### Swing Container



* Top-level container bao gồm:
  + JFrame: sử dụng cho các cửa sổ chính của chương trình
  + JApplet: sử dụng trên trình duyệt
  + JDialog: cửa sổ thông báo
* Secondary container: Jpanel

### Swing Component

1. *JLabel*

*Chức năng:* Chức năng của JLabel giống như Label của thư viện AWT. Tuy nhiên với JLabel có 1 số điểm khác.

* Toán tử khởi tạo bao gồm:

**public** Jpanel();

// Khởi tạo với một Icon hình ảnh

**public** JPanel(**Icon** image);

// Khởi tạo với nội dung text

**public** JPanel(**String** text);

// Khởi tạo với một icon hình ảnh và căn chỉnh nội dung theo chiều ngang là TRÁI/PHẢI/GIỮA

**public** JPanel(**Icon** image, **int** horizontalAlignment);

**public** JPanel(**String** text, **int** horizontalAlignment);

**public** JPanel(**String** text, **Icon** image, **int** horizontalAlignment);

1. *JButton*

*Chức năng:* Dùng để tạo ra một sự kiện nào đó khi người dùng click chuột.

Đối với JButton cũng có điểm khác so với Button của thư viện AWT.

* Toán tử khởi tạo bao gồm:

// Toán tử khởi tạo mặc định

**public** JButton();

// Toán tử khởi tạo với Action cho nút bấm

**public** JButton(**Action** a);

// Toán tử khởi tạo có Icon cho nút bấm

**public** JButton(**Icon** icon);

// Toán tử khởi tạo với nội dung text cho nút bấm

**public** JButton(**String** text);

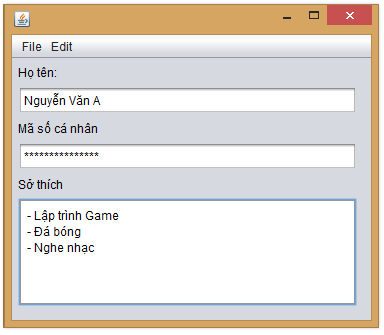
// Toán tử khởi tạo với cả 2 tham số: Nội dung text và Icon

**public** JButton(**String** text, **Icon** icon);

1. *JtextField/ JPasswordField – JText Area*

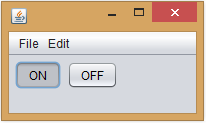
*Chức năng*: Tạo một vùng cho phép người dùng nhập dữ liệu vào từ bàn phím

* *JTextField:* Vùng nhập liệu cơ bản nằm trên một dòng
* *JPasswordField:* Vùng nhập liệu giúp ẩn thông tin, phù hợp với những thông tin như mật khẩu, mã số cá nhân, ...
* *JtextArea:* Cho phép nhập thông tin lớn nằm trên nhiều dòng.



1. *JToggle Button*

*Chức năng:* Đây là một dạng nút công tắc có 2 trạng thái tắt/mở. Khi nhấp vào nút sẽ chuyển từ trạng thái này sang trạng thái kia và ngược lại.

**

* *Toán tử khởi tạo:*

// Khởi tạo với một Action

**public** JtoggleButton(**Action** a);

// khởi tạo với một Icon

**public** JtoggleButton(**Icon** icon);

// Khởi tạo với nội dung text của Button

**public** JtoggleButton(**String** text);

// Khởi tạo với một Icon và trạng thái nút được bật hay tắt

**public** JtoggleButton(**Icon** a, **boolean** selected);

**public** JtoggleButton(**String** text, **Icon** icon);

**public** JtoggleButton(**String** text, **boolean** selected);

**public** JtoggleButton(**String** text, **Icon** icon, **boolean** selected);

* *Phương thức:*

// Đặt trạng thái bật/tắt cho nút

**void** setSelected(**boolean** selected);

// Lấy trạng thái nút đang được bật hay tắt

**boolean** isSelected ();

// Đặt nội dung text cho nút

**void** setText();

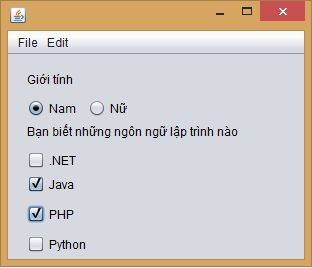
// Lấy nội dung text của nút

**String** getText();

1. *JRadioButton/JCheckBox – JButtonGroup*

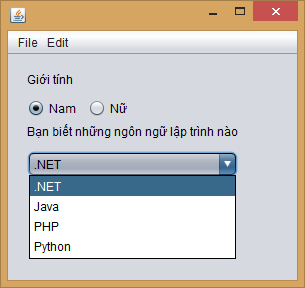
*Chức năng:* Đôi khi ta muốn liệt kê ra nhiều phần tử cho người dùng lựa chọn, lúc này ta sẽ cần tới CheckBox hoặc RadioButton.

* *JCheckBox:* Cho phép chọn nhiều lựa chọn trong danh sách
* *JRadioButton:* Chỉ cho phép lựa chọn duy nhất 1 phần tử trong danh sách.
* *JButtonGroup:* Khi trong cùng 1 Frame, ta có nhiều danh sách cho phép người dùng lựa chọn. Để phân biệt được phần tử này thuộc vào danh sách nào ta cần tới JButtonGroup. JButtonGroup là một component ẩn không hiển thị lên giao diện người dùng.

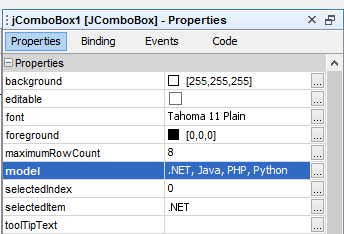


1. *JComboBox*

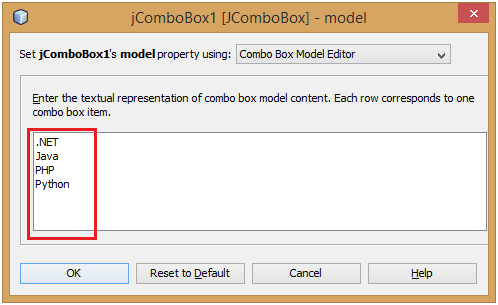
*Chức năng:* JComboBox là component cho phép người dùng lựa chọn phần tử từ một danh sách định sẵn.



* *Cách thêm phần tử bằng giao diện trên Netbean:* Trên bảng thuộc tính của component JComboBox, ta chọn thuộc tính ***Model*** của component:

**

Bảng model sẽ xuất hiện, chúng ta điền các phần tử vào bảng với mỗi phần tử được nằm trên 1 dòng:



* *Toán tử khởi tạo:*

// Toán tử khởi tạo mặc định

public JComboBox();

// KHởi tạo với các phần tử lấy từ ComboBoxModel

public JComboBox(**ComboBoxModel** aModel);

// Khởi tạo với các phần tử là mảng các Object

public JComboBox(**Object**[] items);

// Khởi tạo với các phần tử của component lấy từ Vector

public JComboBox(**Vector** items);

* *Các phương thức*

// Thiết lập các phần tử cho JComboBox

**void** setModel(**ComboBoxModel**<**String**> model);

// Thiết đặt phần tử được chọn dựa vào chỉ số của phần tử

**void** setSelectedIndex(**int** index);

// Thiết đặt phần tử được chọn

**void** setSelectedItem(**Object** obj);

/\* Thiết đặt trạng thái có cho phép sửa đổi nội dung của component hay không \*/

**void** setEditable(**boolean** aFlag);

// Lấy chỉ số phần tử được chọn

**int** getSelectedIndex();

// Lấy phần tử được chọn

**Object** getSelectedItem();

/\* Lấy trạng thái component có cho phép sửa đổi nội dung hay không \*/

**boolean** isEditable();

// Lấy phần tử tại vị trí index

**String** getItemAt(int index);

// Lấy số lượng phần tử của JComboBox

**int** getItemCount();

// Lấy model của ComboBox

**ComboBoxModel** getModel();

**Ví dụ:**

**DefaultComboBoxModel** model;

// Cách 1: Tạo danh sách phần tử với DefaultComboBoxModel

**public** **void** InitModel() {

String[] items = {"C#2", ".NET", "Java", "PHP"};

model = new **DefaultComboBoxModel**(items);

jComboBox1.setModel(model);

}

// Cách 2: Thao tác với DefaultComboBoxModel

**public** **void** InitModel2()

{

model = new **DefaultComboBoxModel**();

model.addElement("C#");

model.addElement(".NET");

model.addElement("Java");

model.addElement("PHP");

model.addElement("Python");

jComboBox1.setModel(model);

}

// Cách 3: Thao tác trực tiếp với JComboBox

**public** **void** InitModel3()

{

// Thêm 1 phần tử vào cuối danh sách của ComboBox

jComboBox1.addItem("1");

jComboBox1.addItem("2");

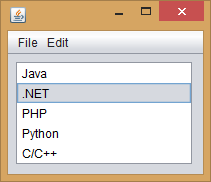
// Chèn 1 phần tử vào vị trí thứ 0 của comboBox

jComboBox1.insertItemAt("0", 0);

}

1. *JList*

*Chức năng*: JList là component cho phép hiển thị thông tin theo dạng danh sách cho phép người dùng tương tác lựa chọn.



* *Toán tử khởi tạo*

**public** JList(**ListModel** dataModel)

**public** JList(**Object**[] listData)

**public** JList(**Vector** listData)

**public** JList()

* *Các phương thức của JList*

// Thiết lập các phần tử cho JList

**void** setModel(**ListModel**<**String**> model);

// Thiết đặt phần tử được chọn dựa vào chỉ số của phần tử

**void** setSelectedIndex(**int** index);

// Thiết đặt phần tử được chọn

**void** setSelectedItem(**Object** obj);

// Thiết lập danh sách các phần tử được chọn

**void** setSelectedIndices(int[] indices);

// Thiết lập danh sách phần tử của JList từ mảng chuỗi

**void** setListData(**String**[] items);

// Thiết lập độ dãn cách dòng của các phần tử

**void** setFixedCellHeight(**int** height);

// Thiết lập chế độ chọn phần tử cho JList: Multiple/Single

**void** setSelectionMode(**ListSelectionModel** mode)

// Lấy chỉ số phần tử được chọn

**int** getSelectedIndex();

// Lấy danh sách các chỉ số các phần tử được chọn

**int**[] getSelectedIndeces();

// Lấy giá trị phần tử được chọn

**String** getSelectedValue();

// Lấy phần tử được chọn

**Object** getSelectedItem();

// Trả về dữ liệu được hiển thị trên JList

**ListModel** getModel();

**Ví dụ:**

**public** **void** InitModel()

{

model = **new** **DefaultListModel**();

model.addElement(".NET");

model.addElement("Java");

model.addElement("PHP");

model.addElement("Delphi");

// Thêm phần tử Python vào vị trí thứ 2 của List

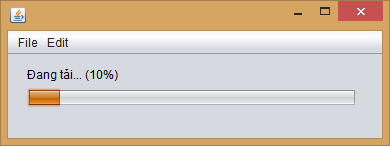
model.set(2, "Python");

jList1.setModel(model);

}

1. *JprogressBar*

*Chức năng:* Component được dùng cho trường hợp chúng ta cần thể hiện tiến độ hoàn thành của một công việc.

**

* *Toán tử khởi tạo*

// Khởi tạo với hướng quay của JprogressBar

**public** JProgressBar(**int** orient);

// Khởi tạo với giá trị nhỏ nhất/ lớn nhất của ProgressBar

**public** JProgressBar(**int** min, **int** max);

**public** JProgressBar**(int** orient**, int** min**, int** max**);**

* *Phương thức cơ bản*

// Đặt trạng thái hiển thị giá trị % hoàn thành tiến độ

**void** setStringPainted(**boolean** b);

// Đặt giá trị tiến độ hoàn thành của JProgressBar

**void** setValue**(int** value**);**

// Đặt giá trị lớn nhất của JProgressBar

**void** setMaximum (**int** max);

// Đặt giá trị nhỏ nhất của JProgressBar

**void** setMinimum(**int** min);

// Lấy giá trị tiến độ hoàn thành của JProgressBar

**int** getValue();

//Lấygiá trị lớn nhất của JProgressBar

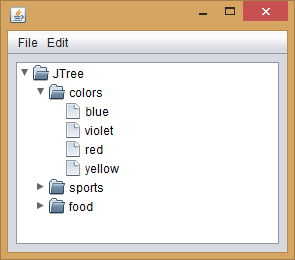
**int** getMaximum();

// Lấy giá trị nhỏ nhất của JProgressBar

**int** getMinimum();

1. *JTree*

*Chức năng:* JTree dùng để thể hiện dữ liệu theo sơ đồ hình cây



* *Toán tử khởi tạo*

**public** JTree();

**public** JTree(**Hashtable**<?, ?> value);

**public** JTree(**Object**[] value);

**public** JTree(**TreeModel** model);

**public** JTree(**TreeNode** root);

**public** JTree(**Vector**<?> value);

**public** JTree(**TreeNode** root, boolean asksAllowsChildren);

* *Các phương thức cơ bản*

**TreeModel** getModel();

**int** getSelectionCount();

**TreePath** getSelectionPath();

**void** setModel(**TreeModel** model);

**void** setEditable(**boolean** flag);

**void** setSelectionPath(**TreePath** path);

*Ví dụ:*

**public** **void** InitModel() {

// Khởi tạo 1 model mới

DefaultTreeModel model;

// Khởi tạo node gốc

DefaultMutableTreeNode root = CreateNode("ROOT" ,null);

// Khởi tạo các thành phần con của cây

DefaultMutableTreeNode node1 = CreateNode("Node 1", root);

CreateNode("Node 2", root);

CreateNode("Node 3", root);

CreateNode("Node 4", root);

CreateNode("Node 5", root);

CreateNode("Sub Node 1", node1);

CreateNode("Sub Node 2", node1);

CreateNode("Sub Node 3", node1);

CreateNode("Sub Node 4", node1);

// Khởi tạo 1 model với node là node root

model = new DefaultTreeModel(root);

// Thiết đặt model cho cây

jTree1.setModel(model);

}

**public** **DefaultMutableTreeNode** CreateNode(**String** name, **DefaultMutableTreeNode** parent)

{

**DefaultMutableTreeNode** node = **new** DefaultMutableTreeNode(name);

if (parent != null)

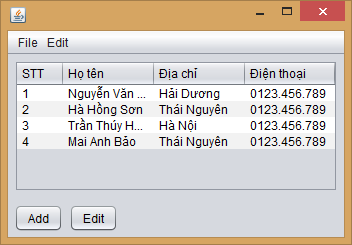
parent.add(node);

**return** node;

}

1. *JTable*

*Chức năng:* JTable được dùng để thể hiện dữ liệu theo dạng bảng.

**

* *Toán tử khởi tạo*

**public** JTable();

**public** JTable(**TableModel** tm);

**public** JTable(**Object**[][] rowData, **Object**[] colsName);

**public** JTable(**TableModel** dm, **TableColumnModel** cm);

**public** JTable(**Vector** rowData, **Vector** colsName);

**public** JTable(**int** numRow, **int** numCol);

**public** JTable(**TableModel** dm, **TableColumnModel** cm, **ListSelectionModel** sm)

* *Phương thức cơ bản*

**TableModel** getModel();

**int** getColumnCount();

**TableColumnModel** getColumnModel();

**String** getColumnName(**int** index)**;**

**void** addColumn(**TableColumn** col);

**void** setModel(**TableModel** model);

// Đặt giá trị cho 1 ô của JTable

**public** **void** setValueAt(**Object** aValue, **int** row, **int** column)

**Ví dụ:**

public void InitTable()

{

// Khởi tạo model cho Table

**DefaultTableModel** model = **new** **DefaultTableModel**();

// Tạo cột

model.addColumn("STT");

model.addColumn("Name");

model.addColumn("Address");

model.addColumn("Phone number");

// Thêm dữ liệu vào bảng

model.addRow(CreateRow(1, “Nguyễn Văn A”,”Thái Nguyên”, “0942.123456”));

// Đặt model vào JTable

jTable1.setModel(model);

}

**public** **Object**[] CreateRow(**int** stt, **String** name, **int** address, **String** phoneNumber)

{

**return** {stt,name,address ,phoneNumber };

}

**BẢNG MỘT SỐ PHƯƠNG THỨC HAY DÙNG**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **void** setLocation(**int** x, **int** y)  **void** setLocation(**Point** p)  **Point** getLocation(); | Đặt lại/Lấy vị trí cho conponent đó so với vị trí container chứa nó. |
| **void** setEnabled(**boolean** b)  **boolean** isEnabled(); | Đặt/Lấy trạng thái component có cho phép người dùng tương tác với nó trên giao diện hay không. |
| **void** setVisible(**boolean** b)  **boolean** isVisible(); | Đặt/lấy trạng component có hiển thị ra giao diện người dùng hay không. |
| **void** setForeground(**Color** c)  **Color** getForeground(); | Đặt/lấy màu của component. Đó thường là màu nội dung text của component. |
| **void** setBackground(**Color** c)  **Color** getBackground(); | Đặt/lấy màu nền component |
| **void** setSize(**int** w, **int** h)  **Point** getSize(); | Đặt/lấy kích thước của component. |
| **void** setText(**String** text)  **String** getText() | Đặt/lấy nội dung text của component đối với 1 số component như JLable, JButton, JRadioButton, Jcheckbox,... |
| **void** setToolTipText(**String** text)  **String** getToolTipText(); | Đặt văn bản trợ giúp cho component khi di chuyển chuột đến component đó. |

### Layout Manager trong thiết kế giao diện

Layout Manager giúp bố trí sắp xếp các thành phần bên trong Container theo một phương thức nhất định. Nói cách khác, chúng ta đặt các thành phần tại một vị trí cụ thể bên trong Container. Tác vụ bố trí này được thực hiện tự động bởi LayoutManager. Nếu chúng ta không sử dụng LayoutManager thì các thành phần cũng sẽ được tự động bố trí bởi LayoutManager mặc định. Tuy nhiên, để xử lý một số lượng thành phần lớn với kích cỡ, hình dạng khác nhau và muốn chúng được bố trí theo cách bạn muốn thì việc sử dụng các LayoutManager là cần thiết.

##### Null Layout

* *Đặc điểm:* Là loại Layout cơ bản nhất, không tự điều chỉnh kích thước và vị trí các phần tử và cho phép các phần tử xếp chồng lên nhau. Với layout này thường được dùng cho bố trí các phần tử không cần theo một quy tắc chung.
* *Kích thước và vị trí phần tử:* Phải được xác định chính xác bằng setBounds hoặc setSize và setLocation.
* *Ưu điểm/Nhược điểm:* 
  + Có thể tự do sắp xếp và thiết lập kích thước cho các phần tử.
  + Khó sắp xếp lại nếu Container thay đổi.
  + Nếu phần tử không được set size và location thì phần tử không hiện thị

*Ví dụ:*

private void initComponents2()

{

setLayout(null);

// Khởi tạo 1 JButton

javax.swing.JButton btn1 = new

javax.swing.JButton("Button 1");

javax.swing.JButton btn2 = new

javax.swing.JButton("Button 2");

// Thiết lập cho button 1 kích thước 100x30

btn1.**setSize**(100,30);

// Thiết lập vị trí cho button 1 ở tọa độ 20, 20

btn1.**setLocation**(new Point(20,20));

// Dùng phương thức setBounds để thay thế

// cho cả 2 phương thức setSize và setLocation

// Tham số: kích thước 100x30 và vị trí 140:20

btn2.**setBounds**(140, 20, 100, 30);

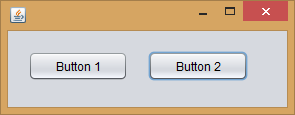
// Gắn Button vào JFrame

this.add(btn1);

this.add(btn2);

}

Kết quả sau khi chạy chương trình:

**

##### Border Layout

*Đặc điểm:*

* Các phần tử trong layout này bị tự động kéo dãn theo kích thước từng phần của layout.
* Khi bạn sử dụng IDE kéo thả có thể thấy panel thành 5 khu vực *trên* (first)-*dưới* (last) – *trái* (west) - *phải* (east) - *giữa* (center) nhưng code được sinh ra là *trên* (PAGE\_START) - *dưới* (PAGE\_END) – *trái* (LINE\_START) - *phải* (LINE\_END) - *giữa* (Center)
* Khi kích thước cửa sổ thay đổi thì vị trí và kích thước của tất cả các phần tử đều thay đổi, riêng witdh của Line\_start và Line\_end (setPreferredSize()) thì không đổi

BorderLayout thường được dùng cho bố trí root (bố trí cho các Container gốc)

*Ví dụ:*

private void initComponents3(){

javax.swing.JButton jButton1 = new

javax.swing.JButton();

javax.swing.JButton jButton2 = new

javax.swing.JButton();

javax.swing.JButton jButton3 = new

javax.swing.JButton();

javax.swing.JButton jButton4 = new

javax.swing.JButton();

javax.swing.JButton jButton5 = new

javax.swing.JButton();

setDefaultCloseOperation(

javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jButton1.setText("jButton1");

getContentPane().add(jButton1,

java.awt.BorderLayout.PAGE\_START);

jButton2.setText("jButton2");

getContentPane().add(jButton2,

java.awt.BorderLayout.LINE\_END);

jButton3.setText("jButton3");

getContentPane().add(jButton3,

java.awt.BorderLayout.LINE\_START);

jButton4.setText("jButton4");

getContentPane().add(jButton4,

java.awt.BorderLayout.PAGE\_END);

jButton5.setText("jButton5");

getContentPane().add(jButton5,

java.awt.BorderLayout.CENTER);

}

Kết quả sau khi chạy chương trình:



##### FollowLayout

*Đặc điểm:* FollowLayout cho phép hiển thị các phần tử theo dạng từ trái qua phải, từ trên xuống dưới. Khi chiều dài của phần tử không đủ sắp xếp trên hàng thì phần tử đó sẽ được tự động đẩy xuống hàng tiếp theo.

Follow Layout thường được dùng để hiển thị cho các chức năng điều khiển, giả sử như thiết kế thanh MenuBar chẳng hạn.

Ví dụ:

private void initFollowLayout() {

this.setSize(300, 150);

javax.swing.JButton jButton1 =

new javax.swing.JButton();

javax.swing.JButton jButton2 =

new javax.swing.JButton();

javax.swing.JButton jButton3 =

new javax.swing.JButton();

javax.swing.JButton jButton4 =

new javax.swing.JButton();

javax.swing.JButton jButton5 =

new javax.swing.JButton();

setDefaultCloseOperation(

javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

getContentPane().setLayout(new java.awt.FlowLayout());

// Sắp xếp theo hướng từ trái sang phải

// getContentPane().setLayout(

//new java.awt.FlowLayout(

//java.awt.FlowLayout.LEFT));

jButton1.setText("jButton1");

this.add(jButton1);

jButton2.setText("jButton2");

this.add(jButton2);

jButton3.setText("jButton3");

this.add(jButton3);

jButton4.setText("jButton4");

this.add(jButton4);

jButton5.setText("jButton5");

this.add(jButton5);

}

Kết quả sau khi chạy chương trình:



##### BoxLayout

BoxLayout cho phép chúng ta sắp xếp các control trên một hàng ngang hoặc trên một cột thẳng đứng.

Khi ta sắp xếp theo hàng ngang thì khác với FlowLayout, các control sẽ không tự động xuống dòng nếu như không đủ chỗ cho chúng.

*Ví dụ:*

private void initComponents4(){

this.setSize(500, 400);

javax.swing.JButton jButton1 = new

javax.swing.JButton();

javax.swing.JButton jButton2 = new

javax.swing.JButton();

javax.swing.JButton jButton3 = new

javax.swing.JButton();

setDefaultCloseOperation(

javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

this.setLayout(new

javax.swing.BoxLayout(getContentPane(),

javax.swing.BoxLayout.Y\_AXIS));

jButton1.setText("jButton1");

// Nếu thiết lập maxSize cho phần tử, phần tử

// sẽ tự co dãn cho phù hợp với Container

// khi Size của Container thay đổi

// Button1.setMaximumSize(

new Dimension(getWidth(), 50));

this.add(jButton1);

jButton2.setText("jButton2");

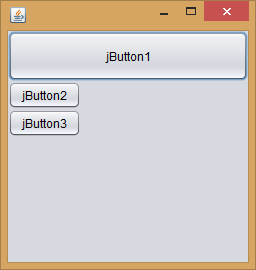
this.add(jButton2);

jButton3.setText("jButton3");

this.add(jButton3);

}

Kết quả sau khi chạy chương trình:



##### GridLayout

*Đặc điểm:* GridLayout cho phép hiển thị các phần tử theo dạng lưới:

* Container sẽ được chia thành các hàng hoặc cột.
* Các phần tử trong mỗi phần có kích thước được co dãn theo kích thước của từng phần của layout.
* Vị trí các phần tử tùy thuộc vào việc layout bị co dãn

Grid layout thường dùng để chia pane thành nhiều phần bằng nhau

Ví dụ: Thiết lập Layout có 2 hàng và cột.

private void initGridLayout() {

this.setSize(400, 300);

javax.swing.JButton jButton1 =

new javax.swing.JButton("Button 1");

javax.swing.JButton jButton2 =

new javax.swing.JButton("Button 2");

javax.swing.JButton jButton3 =

new javax.swing.JButton("Button 3");

javax.swing.JButton jButton4 =

new javax.swing.JButton("Button 4");

javax.swing.JButton jButton5 =

new javax.swing.JButton("Button 5");

javax.swing.JButton jButton6 =

new javax.swing.JButton("Button 6");

setDefaultCloseOperation(

javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

// Thiết lập Layout có 2 hàng, 3 cột

getContentPane().setLayout(

new java.awt.GridLayout(2, 3));

this.add(jButton1);

this.add(jButton2);

this.add(jButton3);

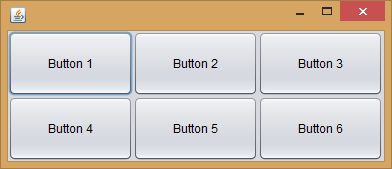
this.add(jButton4);

this.add(jButton5);

this.add(jButton6);

}

Kết quả sau khi chạy chương trình:



## Xử lý sự kiện trong Java

### Giới thiệu

Các hệ thống GUI xử lý các tương tác người dùng với sự trợ giúp của mô hình hướng sự kiện (event-driven). Tương tác của người dùng có thể là di chuyển chuột, nhấn phím, nhả phím v.v…Tất cả các thao tác này thiết lập một sự kiện của một loại nào đó.

Việc xử lý những sự kiện này phụ thuộc vào ứng dụng. Abstract Windowing Toolkit (AWT) xử lý một vài sự kiện. Môi trường mà các ứng dụng này được thi hành như trình duyệt cũng có thể xử lý các sự kiện khác. Người lập trình cũng cần phải xử lý những sự kiện nhất định và cần phải viết hàm xử lý các sự kiện đó.

Ứng dụng cần đăng ký một hàm xử lý sự kiện với một đối tượng. Hàm xử lý sự kiện này sẽ được gọi bất cứ khi nào sự kiện tương ứng phát sinh. JDK1.2 làm việc theo mô hình xử lý sự kiện này.

Trong quy trình này, ứng dụng cho phép bạn đăng ký các phương thức (handler), hay gọi là listener với các đối tượng. Những handler này tự động được gọi khi một sự kiện thích hợp phát sinh.

Một Event Listener lắng nghe một sự kiện nào đó mà một đối tượng đã thiết lập. Mỗi event listener cung cấp các phương thức xử lý những sự kiện này. Lớp thi hành listener cần phải định nghĩa những phương thức này. Để sử dụng mô hình này, bạn làm theo các bước sau:

* Cài đặt giao diện listener thích hợp. Cấu trúc như sau:

**public** **class** MyApp **extends** Frame **implements** ActionListener

* Xác định tất cả các thành phần tạo ra sự kiện. Các thành phần có thể là các button, label, menu item, hay window.
* Xác định tất cả các sự kiện được xử lý. Các sự kiện có thể là một ‘ActionEvent’ nếu một button được click hay một ‘mouseEvent’ nếu như chuột được kéo đi.
* Thi hành các phương thức của listener và viết hàm xử lý sự kiện tương ứng với các phương thức.

**CÁC LỚP SỰ KIỆN TRONG JAVA**

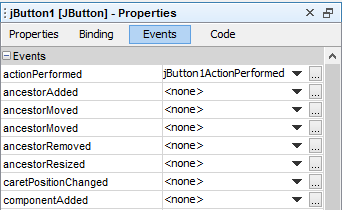
|  |  |
| --- | --- |
| **Lớp sự kiện** | **Mô tả** |
| ActionEvent | Phát sinh khi một button được nhấn, một item trong danh sách chọn lựa được nhấn đúp (double-click) hay một menu được chọn. |
| AdjustmentEvent | Phát sinh khi một thanh scrollbar được sử dụng. |
| ComponentEvent | Phát sinh khi một thành phần được thay đổi kích thước, được di chuyển, bị ẩn hay làm cho hoạt động được. |
| FocusEvent | Phát sinh khi một thành phần mất hay nhận focus từ bàn phím. |
| ItemEvent | Phát sinh khi một mục menu được chọn hay bỏ chọn; hay khi một checkbox hay một item trong danh sách được click. |
| WindowEvent | Phát sinh khi một cửa sổ được kích hoạt, được đóng, được mở hay thoát. |
| TextEvent | Phát sinh khi giá trị trong thành phần textfield hay textarea bị thay đổi. |
| MouseEvent | Phát sinh khi chuột di chuyển, được click, được kéo hay thả ra. |
| KeyEvent | Phát sinh khi bàn phím ấn, nhả. |

### Sử dụng sự kiện

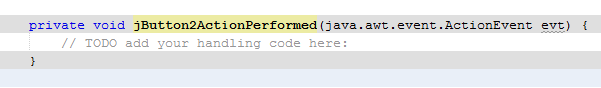
*Cách 1. Bắt sự kiện trên giao diện Netbean.*

Tại cửa sổ ***Properties*** của component, chọn sang tab ***Events***. Tại đây chúng ta sẽ có danh sách tất cả các sự kiện tương ứng với component đó.

Để bắt sự kiện cho component, ta chỉ việc nhấp double- click vào tên sự kiện. Ở đây ta chọn sự kiện ***actionPerformed***



Sau đó, IDE sẽ tự động tạo một phương thức cho sự kiện tương ứng. Chúng ta sẽ viết code để thực hiện xử lý sự kiện trong thân phương thức này.



*Cách 2. Bắt sự kiện trong Code*

Tại hàm khởi tạo ta thêm sự kiện cho nút như sau:

**public** **void** AddEvent()

{

jButton1.addActionListener(**new** java.awt.event.**ActionListener**() {

**public** **void** actionPerformed(java.awt.event.**ActionEvent** evt) {

jButton1ActionPerformed(evt);

}

});

}

**private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt)

{

// TODO: Xử lý khi xảy ra sự kiện

}

Tại toán tử khởi tạo của Frame, ta gọi phương thức AddEvent

**public** NewJFrame()

{

initComponents();

AddEvent();

}

### Các sự kiện cơ bản

|  |  |  |
| --- | --- | --- |
| **Events** | **Component** | **Description** |
| JButton | ActionPerformed | Xảy ra khi có sự kiện Click chuột. |
| JLabel | PropertyChange | Xảy ra khi nội dung Text của Lable được thay đổi. |
| JToggleButton | ActionPerformed | Xảy ra khi có sự kiện Click chuột. |
| StateChanged | Xảy ra khi trạng thái ON/OFF của nút bị thay đổi. |
| JCheckBox/  JRadioButton | StateChanged | Xảy ra khi trạng thái check của CheckBox/RadioButton được thay đổi. |
| JComboBox | ActionPerformed | Xảy ra khi có sự lựa chọn phần tử được thực hiện. |
| JList | ValueChanged | Xảy ra khi có sự lựa chọn phần tử thay đổi |
| TextField/  PasswordField | ActionPerformed | Sự kiện xảy ra khi con trỏ chuột đang ở vùng nhập liệu của TextField mà nhấn phím Enter. |

# CHƯƠNG 4.

# LẬP TRÌNH CƠ SỞ DỮ LIỆU VỚI JDBC

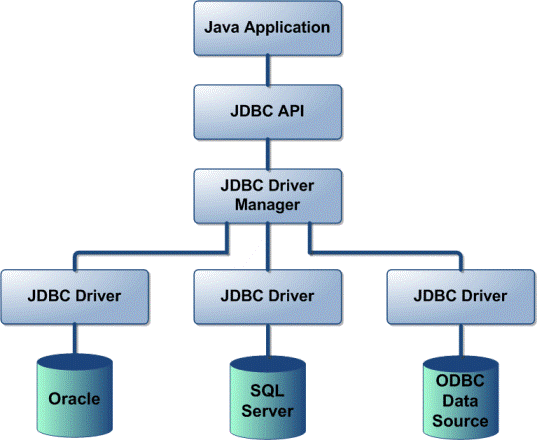
1. **Lập trình cơ sở dữ liệu**

JDBC là viết tắt của Java Database Conectivit. JDBC là chuẩn kết nối cơ sở dữ liệu với các ứng dụng viết bằng ngôn ngữ java. Nói cách khác, JDBC là trung gian giúp chương trình và cơ sở dữ liệu có thể làm việc với nhau.

Khác với các chuẩn kết nối bằng thư viện ADO hay ADO.NET của .NET, JDBC không được xây dựng sẵn các Driver kết nối tới các RDBMS (hệ quản trị csdl). JDBC ứng với từng hệ quản trị csdl mà sẽ sử dụng các driver khác nhau, do đó, tính mềm dẻo của csdl JDBC rất cao, có thể tương thích với hầu hết các RDBMS.

JDBC và Java nói chung đều có đặc điểm là không có nhiều các lớp công nghệ (class) được xây dựng sẵn, mà bù lại, Java định ra nhiều Interface, từ đó các hãng, các tổ chức open source sẽ Implements các Interface này … do đó, dù nhiều công nghệ được phát triển, nhưng Java vẫn có tính thống nhất chung…

## Kiến trúc của JDBC

****

Trong Java có 2 lớp chủ yếu chịu trách nhiệm về thiết lập kết nối đến một cơ sở dữ liệu.

Lớp đầu tiên là *DriverManager*. Đó là một trong rất ít các lớp thực sự do JDBC API cung cấp. *DriverManager* chịu trách nhiệm quản lý một nhóm (pool) các driver đã đăng kí, mà thực chất là là trừu tượng hóa các chi tiết về việc sử dụng một driver, cho nên lập trình viên không cần phải làm việc trực tiếp với driver đó.

Lớp thứ 2 là một lớp *JDBC Driver* thực sự. Nó được cung cấp bởi các nhà sản xuất phần mềm độc lập. Lớp *JDBC Driver* chịu trách nhiệm thiết lập đường kết nối cơ sở dữ liệu và xử lý tất cả các giao tiếp với cơ sở dữ liệu đó. Các *JDBC Driver* chia thành 4 kiểu khác nhau. Chúng ta sẽ chia nó ra làm 2 phần:

JDBC API (các gói java.sql và javax.sql )

Các kiểu JDBC Driver JDBC API, JDBC API có sẵn trong các gói java.sql và javax.sql.

Sau đây là các lớp JDBC, các giao diện và các lỗi ngoại lệ quan trọng trong gói ***java.sql***:

*DriverManager* - Nạp các JDBC driver vào trong bộ nhớ. Có thể sử dụng nó để mở các kết nối tới một nguồn dữ liệu.

*Connection* - Biểu thị một kết nối đến một nguồn dữ liệu. Được dùng để tạo ra các đối tượng Statement, PreparedStatement và CallableStatement. Statement - Biểu diễn một lệnh SQL tĩnh. Có thể sử dụng nó để thu về đối tượng ResultSet.

*PreparedStatement* - Một giải pháp thay thế hoạt động tốt hơn đối tượng Statement, thực thi một câu lệnh SQL đã được biên dịch trước.

*CallableStatement* – biểu diễn một thủ tục được lưu trữ. Có thể được sử dụng để thực thi các thủ tục được lưu trữ trong một RDBMS có hỗ trợ chúng.

*ResultSet* - biểu diễn một tập kết quả trong cơ sở dữ liệu tạo ra bởi việc sử dụng một câu lệnh SQL là SELECT.

*SQLException* - một lớp xử lý lỗi ngoại lệ chứa các lỗi truy cập cơ sở dữ liệu.

Gói thứ hai, ***javax.sql*** là một bộ phận của *J2SE 1.4* và *J2EE 1.3*. Nó bổ sung các tính năng sau đây vào JDBC để hỗ trợ thêm cho các tính năng đã có trong gói java.sql:

*DataSource* - Trừu tượng hóa một nguồn dữ liệu. Đối tượng này có thể sử dụng thế cho 8. *DriverManager* để tạo ra một cách có hiệu quả các kết nối cơ sở dữ liệu (có khả năng sử dụng việc chứa/phân chia các đường kết nối ngầm). Tạo sẵn cơ chế phân chia đường kết nối (built-in connection pooling).

*XADataSource, XAConnection* – Cho phép/hỗ trợ các giao dịch phân phối. RowSet – Nó mở rộng giao diện ResultSet để tăng thêm sự hỗ trợ đối với các tập kết nối bị ngắt.

## Các khái niệm cơ bản

### JDBC Driver

Như ta đã biết trong kiến trúc của JDBC, lớp *JDBC Driver* chịu trách nhiệm thiết lập đường kết nối cơ sở dữ liệu và xử lý tất cả các giao tiếp với cơ sở dữ liệu.

Có 4 loại JDBC driver

* JDBC Driver loại 1 (JDBC-ODBC bridge)

Cung cấp các truy cập JDBC thông qua các truy cập ODBC. Driver này chuyển các lời gọi JDBC thành lời gọi các hàm trong ODBC.

* *Đặc điểm*
* Dễ kết nối, kết nối thẳng đến CSDL.
* ODBC driver phụ thuộc vào hệ điều hành và đòi hỏi client phải được cài driver.
* Tốc độ truy xuất chậm.
* Trên Access, ODBC driver bị lỗi khi chuyển mã ký tự.
* JDBC Driver loại 2 (Native-API partly Java driver)

Chuyển các lệnh gọi JDBC thành các lệnh gọi API trên máy client của CSDL (như Oracle, SyBase, DB2, …)

* *Đặc điểm*
* Đòi hỏi trên máy client cần phải nạp trước một số lệnh nhị phân của các trình chạy trên máy client của CSDL.
* JDBC driver loại này phụ thuộc nền tảng sử dụng.
* Không phải CSDL nào cũng có driver dạng này.
* Driver này tương tích hầu hết tất cả các ứng dụng Java ngoại trừ Applet
* JDBC Driver loại 3 (Network-Protocol driver)

Đây là dạng driver giao thức mạng. Driver này chuyển các lệnh gọi JDBC thành các giao thức mạng độc lập với hệ quản trị CSDL (DBMS Independent Net Protocol). Chính phần mềm trên máy chủ trung gian này có thể kết nối các máy khánh có nên Java với nhiều loại CSDL khác nhau.

* Ưu điểm: JDBC này rất linh hoạt vì có thể kết nối đến nhiều loại hệ quản trị CSDL khác nhau mà không cần cài đặt driver dưới máy client.
* Khuyết điểm:
* Phụ thuộc nhiều vào nhà cung cấp phần mềm trung gian.
* Kết nối thông qua một lớp nữa sẽ có thể dẫn đến tình trạng thắt cổ chai. Nhưng thông thường điều này được khắc phục ở các nhà cung cấp phần mềm trung gian tốt.
* JDBC Driver loại 4 (Native-Protocol driver)

Đây là dạng driver với giao thức gốc hoàn toàn Java. Được viết hoàn toàn bằng Java. Biến các lệnh gọi JDBC thành các giao thức mạng gọi trực tiếp từ hệ quản trị CSDL.

Trình này cho phép gọi trực tiếp từ máy khách đến máy chủ chứa DBMS và là giải pháp thực tế cho việc truy cập Intranet.

* Ưu điểm:
* JDBC này không cần chuyển sang lời gọi nào như ODBC.
* Không cần lớp trung gian để liên hệ với các CSDL.
* Được viết bằng Java nên hoàn toàn không phụ thuộc nào nền tảng sử dụng.
* Cải thiện hiệu suất truy cập.
* Khuyết điểm: Cần driver riêng cho từng loại CSDL

### JDBC URL

JDBC URL là chuỗi có cấu trúc bao gồm tên hệ quản trị cơ sở dữ liệu, địa chỉ IP, port và tên cơ sở dữ liệu cần kết nối. Thông qua chuỗi kết nối này, DriverManager có thể kết nối tới hệ quản trị cơ sở dữ liệu tương ứng với nó.

Định dạng một số chuỗi kết nối:

* SQLServer:

jdbc:sqlserver://<host>:<port>;databaseName=<database name>

* MySQL:

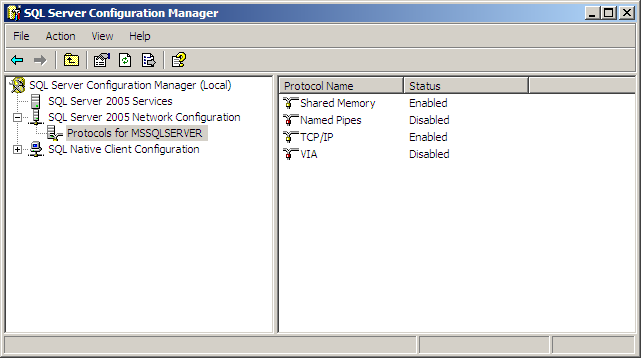
jdbc:mysql://<host>:<port>/<database\_name>

## Kết nối cơ sở dữ liệu với JDBC

### Cấu hình SQL Server

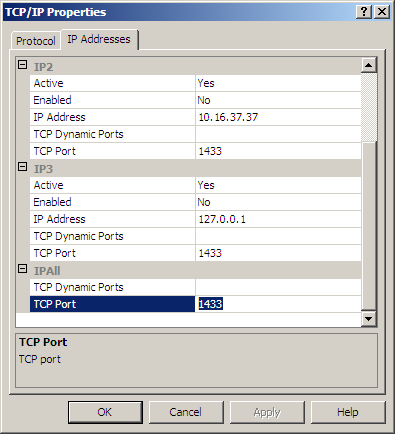
Để kết nối với SQL cần mở cổng cho SQL, cách cài đặt:

Mở ***Configuration Tools > SQL Server Configuration Manager.***



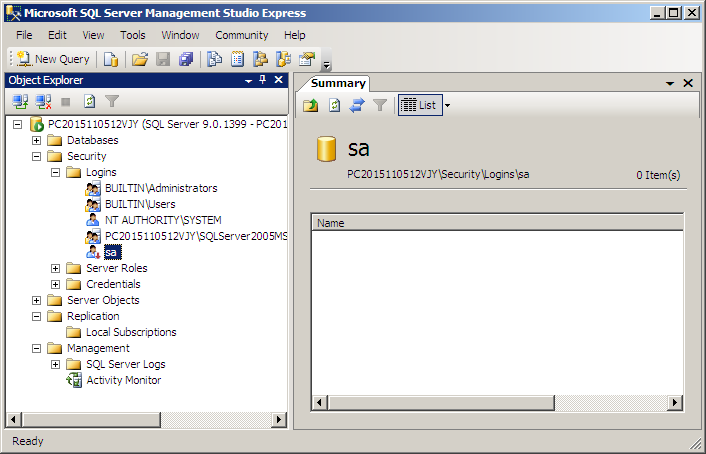
Mở ***SQL Server Network Configuration > Protocols for MSSQLSERVER.*** Chọn ***TCP/IP*** và thực hiện đặt trạng thái Enabled.

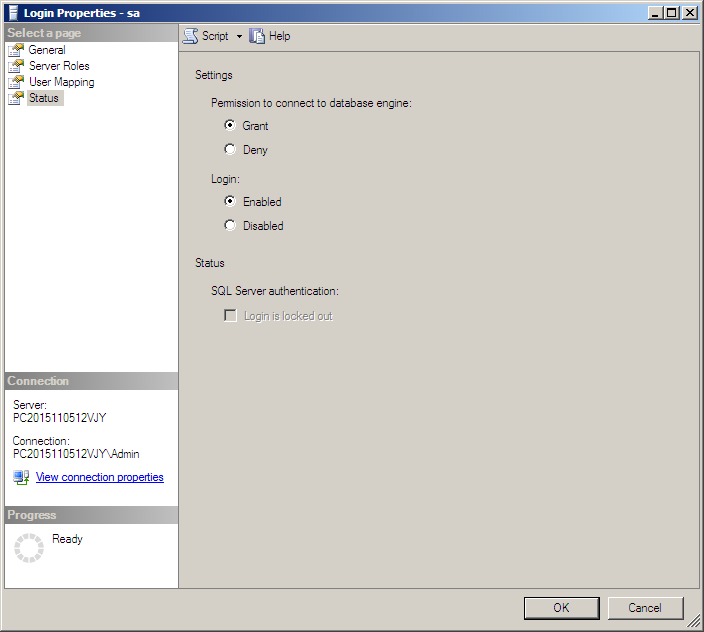
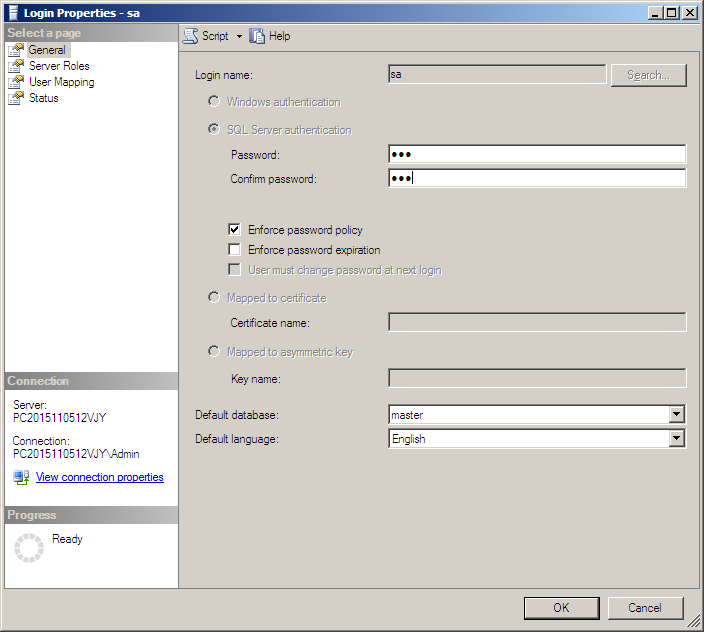
Click double vào ***TCP/IP*** và đặt cổng là 1433, sau đó hãy khởi động lại SQL Server.



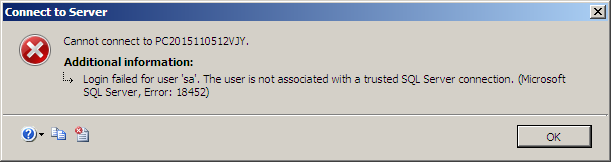
Như vậy chúng ta đã mở cổng cho SQL Server để kết nối vào cơ sở dữ liệu.

Bước tiếp theo chúng ta cần thiết lập 1 tài khoản để sử dụng cho việc kết nối vào SQL Server. Trong trường hợp này, chúng ta sẽ sử dụng tài khoản ***sa*** (tài khoản root) để kết nối. Thực hiện đổi mật khẩu cho tài khoản ***sa*** và đặt trạng thái Enable cho phép sử dụng tài khoản **sa** để đăng nhập vào cơ sở dữ liệu như sau:



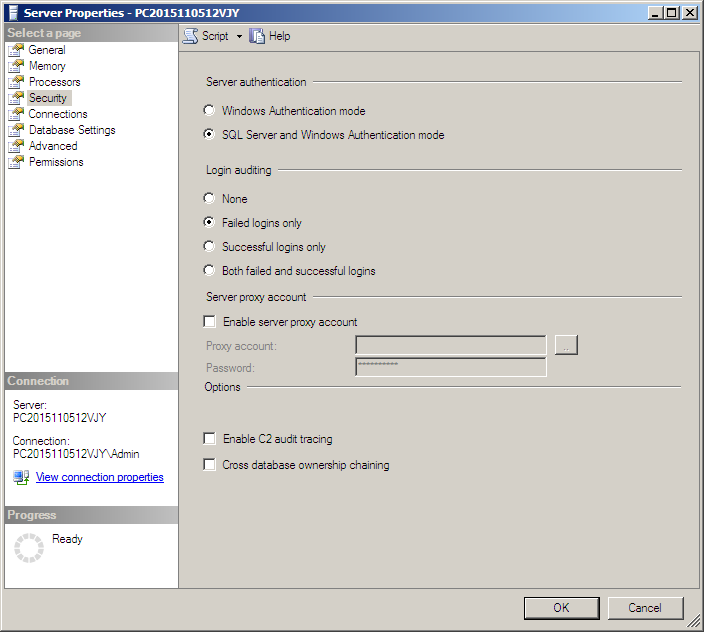
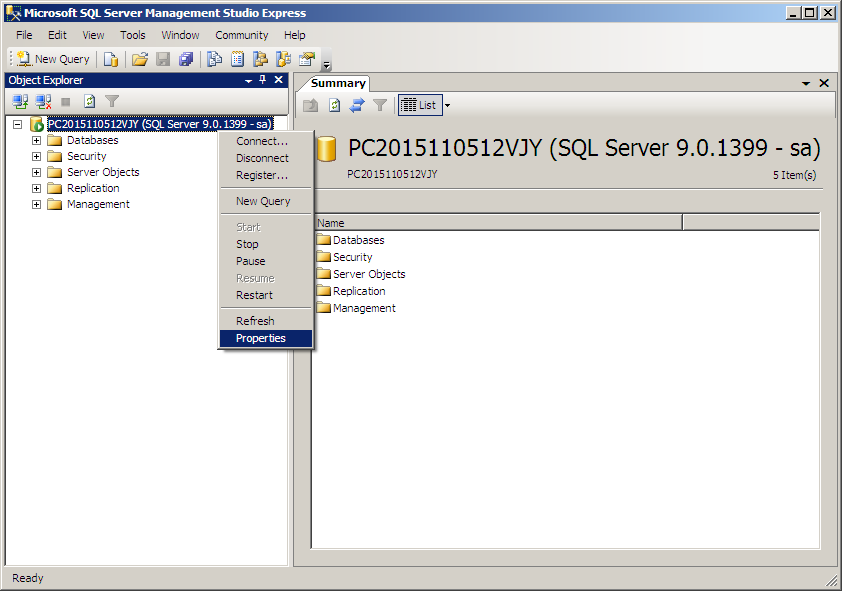


Thực hiện ngắt kết nối với SQL Server và đăng nhập bằng tài khoản ***sa***. Trong trường hợp không đăng nhập được với lỗi như hình dưới:



Ta thực hiện cấu hình lại như sau:

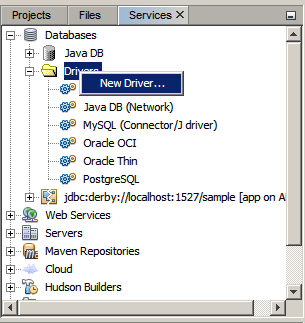
Click phải chuột vào ***Server Name> Properties,*** chuyển sang ***Tab Security,*** chọn chế độ ***SQL Server and Windows Authentication mode.***



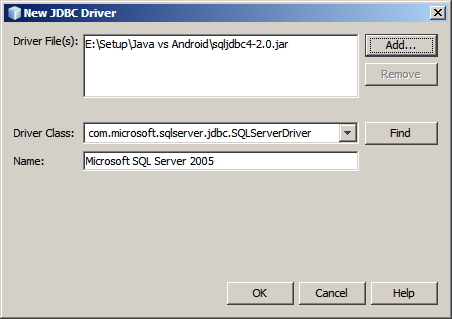
Thực hiện khởi động lại SQL Server và đăng nhập bằng tài khoản **sa.**

### Kết nối với SQL Server

Trên ***Netbean***, tại cửa sổ ***Services (Windows > Services),***  chúng ta tiến hành thêm Driver của SQLServer

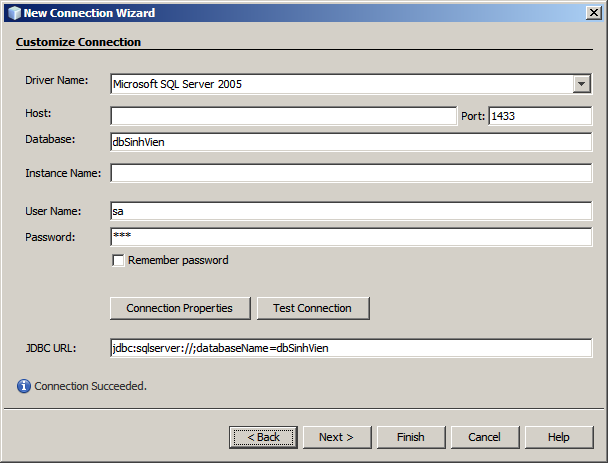
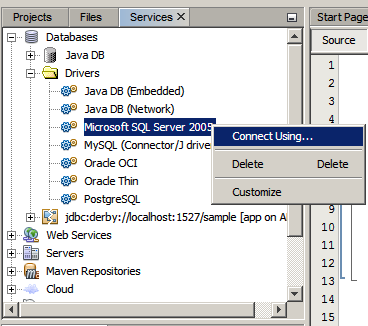
**

*Thêm Driver của SQL Server*



*Add driver từ đường dẫn*

Sau khi thêm driver xong, tại cửa sổ ***Service*** chúng ta sẽ thấy có Driver ***Microsoft SQL Server 2005***. Click phải chuột chọn ***Connect Using...*** Tiến hành cấu hình kết nối với các thông tin như hình:



Trong đó

***Port:*** Là cổng kết nối chúng ta đã cài đặt trên SQL Server ở phần 4.3.1

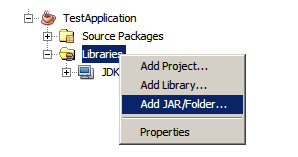
***Database:*** Tên cơ sở dữ liệu chúng ta cần kết nối tới

***Username, password:*** Tên tài khoản/Mật khẩu chúng ta đã thiết lập

Nhấn ***Test Connection*** để kiểm tra kết nối tới SQL Server thành công hay chưa.

## Các thao tác cơ bản trên CSDL

Trước hết để thao tác được với cơ sở dữ liệu, ta cần thêm thư viện vào project. Chọn ***Libraries> Add JAR/Folder...,*** lần tới file ***sqljdbc.jar*** (file Driver ta đã thao tác ở bước kết nối) và thêm vào project.



### Tạo kết nối với dữ liệu

* *Tạo kết nối tới CSDL/Đóng kết nối:* Để tạo kết nối tới cơ sở dữ liệu, ta thông qua đối tượng *Connection* do Java cung cấp nằm trong gói ***Java.sql***

// Tạo biến tên cơ sở dữ liệu

**public** **static** **String** dbName = "";

// Phương thức kết nối tới cơ sở dữ liệu

**public** **Connection** getConnectDB(**String** dbName) {

**Connection** conn = null;

**try**

{

**String** URL = "jdbc:sqlserver://127.0.0.1:1433;databaseName=" + dbName;

String userName = "sa";

String pwd = "123";

/\*

Trong đó:

- 127.0.0.1:1433 – Địa chỉ ip localhost và port 1433 ta đã thiết lập.

- userName và pwd là mật khẩu và password của tài khoản đăng nhập vào SQLServer

\*/

**Class**.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

conn = **DriverManager**.getConnection(URL, userName, pwd);

System.out.println("Kết nối thành công!");

}

**catch** (**ClassNotFoundException** | **SQLException** ex)

{

System.out.println("Kết nối thất bại!");

}

**return** conn;

}

// Phương thức đóng kết nối

**public** **void** CloseConnection(**Connection** conn)

{

if (conn != null)

**try**

{

conn.close();

}

**catch** (Exception ex) {}

}

### Thao tác truy vấn với dữ liệu

* *Hiển thị dữ liệu:* Để truy vấn dữ liệu trong bảng cơ sở dữ liệu, ta sử dụng đối tượng ***ResultSet***. Đối tượng ResultSet sẽ chứa toàn bộ dữ liệu ta truy vấn được, thông qua đó ta có thể hiển thị dữ liệu lên giao diện người dùng từ đối tượng ResultSet.

**public** **ResultSet** ShowData(**String** sql)

{

**Connection** conn = getConnectDB(dbName);

**Statement** state = null;

**ResultSet** rs = null;

try

{

state = conn.createStatement();

rs = state.executeQuery(sql);

}

catch (Exception ex) {}

**return** rs;

}

### Cập nhật dữ liệu dữ liệu

* *Cập nhật dữ liệu:* Tạo phương thức cho phép thực hiện cập nhật dữ liệu bao gồm thêm, sửa, xóa dữ liệu

**public** **boolean** UpdateData(**String** sql)

{

**Connection** conn = getConnectDB(dbName);

**int** row = 0;

**try**

{

Statement state = (Statement) conn.createStatement();

row = state.executeUpdate(sql);

if (row > 0) **return** **true**;

}

**catch** (SQLException ex)

{

System.err.println("Error: " + ex.getMessage());

}

**finally** { CloseConnection(conn); }

**return** **false**;

}

**Ví dụ:** Cơ sở dữ liệu có bảng tblSinhVien với các thông tin như sau:

|  |  |  |
| --- | --- | --- |
| **Tên trường** | **Kiểu dữ liệu** | **Mô tả** |
| ID | int | Mã sinh viên |
| Name | String | Tên sinh viên |
| Address | String | Địa chỉ |
| Age | int | Tuổi |

1. Hãy viết chương trình hiển thị dữ liệu ra màn hình
2. Viêt chương trình thêm/sửa/xóa dữ liệu từ bảng tblSinhVien

**Đáp án:**

Với các phương thức ta đã xây dựng ở phần trên, ta đưa vào trong class ***DBHelper***

File ***SinhVien.java*** (class ***SinhVien***) ta viết:

// Khởi tạo đối tượng DBHelper

**public** **DBHelper** dbHelper = new DBHelper();

// Hàm khởi tạo SinhVien

**public** SinhVien() {

dbHelper.dbName = "dbSinhVien";

}

// Hiển thị dữ liệu

**public** **void** ShowData()

{

**String** sql = "Select \* from tblSinhVien";

**try**

{

**ResultSet** rs = dbHelper.ShowData(sql);

**while** (rs.next()) {

String data = "ID: " + rs.getString(0);

data += " - Name: " + rs.getString(1);

data += " - Address: " + rs.getString(2);

data += " - Age: " + rs.getString(3);

System.out.println(data);

}

}

**catch** (SQLException ex) {}

}

// Thêm dữ liệu

**public** **void** InsertData()

{

**String** sql = "INSERT INTO tblSinhVien values('1', 'Nguyen van A', '22', 'Thai Nguyen')";

**if**(dbHelper.UpdateData(sql))

System.out.println("Thêm thành công!");

**else**

System.out.println("Thêm thất bại!");

}

/\* Với các hàm sửa, xóa ta làm tương tự hàm thêm dữ liệu, chỉ cần thay câu lệnh truy vấn sql \*/

File ***Main.java***

// Khởi tạo đối tượng SinhVien

**static** **SinhVien** sv = new **SinhVien**();

**public** **static** **void** main(**String** args[])

{

sv.ShowData();

sv.InsertData();

}