MỤC LỤC

CHƯƠNG 1: MỞ ĐẦU

1.1. Giới thiệu ngôn ngữ lập trình Java

1.1.1 Java là gì?

Java là một ngôn ngữ lập trình phổ biến được sử dụng rộng rãi để viết mã cho các ứng dụng web. Java là một ngôn ngữ đa nền tảng, hướng đến đối tượng, lấy mạng làm trung tâm và có thể được sử dụng như một nền tảng. Java là một ngôn ngữ lập trình nhanh, bảo mật, đáng tin cậy dùng để viết mã cho mọi thứ từ ứng dụng di động, phần mềm doanh nghiệp cho đến các ứng dụng dữ liệu lớn và công nghệ phía máy chủ

1.1.2 Ngôn ngữ lập trình Java có công dụng gì?

Java là một ngôn ngữ miễn phí và linh hoạt, nó có thể được dùng để phát triển các phần mềm cục bộ và phân tán. Một số công dụng phổ biến của Java bao gồm:

1. Phát triển trò chơi

Nhiều trò chơi điện tử, trò chơi máy tính và di động nổi tiếng được phát triển bằng Java. Ngay cả những trò chơi hiện đại được tích hợp công nghệ tiên tiến như máy học hay thực tế ảo cũng được phát triển bằng công nghệ Java.

2. Điện toán đám mây

Java thường được gọi là ngôn ngữ WORA (Viết một lần, chạy ở mọi nơi), khiến nó trở thành ngôn ngữ lý tưởng cho các ứng dụng phi tập trung dựa trên đám mây. Các nhà cung cấp đám mây chọn ngôn ngữ Java để chạy các chương trình trên nhiều nền tảng cơ sở khác nhau.

3. Dữ liệu lớn

Java được dùng cho các công cụ xử lý dữ liệu có thể làm việc với những tập dữ liệu phức tạp và số lượng dữ liệu thời gian thực khổng lồ.

4. Trí tuệ nhân tạo

Java là một trung tâm thư viện máy học đồ sộ. Sự ổn định và tốc độ của ngôn ngữ này rất lý tưởng cho việc phát triển ứng dụng trí tuệ nhân tạo như xử lý ngôn ngữ tự nhiên và học sâu.

5. Internet vạn vật

Java đã được sử dụng để lập trình các cảm biến và phần cứng trong thiết bị biên có thể kết nối một cách độc lập với Internet.

1.1.3 Vì sao Java lại trở tránh một lựa chọn phổ biến của các nhà phát triển phần mềm hiện đại

Java phổ biến vì nó được thiết kế để có thể dễ dàng sử dụng.

Một số lý do mà nhà phát triển tiếp tục chọn Java thay vì những ngôn ngữ lập trình khác bao gồm:

1. Tài nguyên học tập chất lượng cao

Java có nguồn tài nguyên học tập đồ sộ, dễ dàng tiếp cận, giúp cho các nhà phát triển, hoặc người mới làm quen có thể dễ dàng bắt đầu và nâng cao kiến thức thông qua nguồn tài liệu phong phú.

2. Các chức năng và thư viện sẵn có

Khi sử dụng Java, nhà phát triển không cần phải viết mọi chức năng mới từ đầu. Thay vào đó, Java cung cấp một hệ sinh thái phong phú gồm các chức năng và thư viện sẵn có để phát triển hàng loạt ứng dụng đa dạng.

3. Sự hỗ trợ tích cực của cộng đồng

Java có rất nhiều người dùng hoạt động và một cộng đồng có thể hỗ trợ nhà phát triển khi họ đối mặt với các thách thức trong việc viết mã. Phần mềm nền tảng Java cũng được duy trì và cập nhật thường xuyên.

4. Công cụ phát triển chất lượng cao

Java cung cấp nhiều công cụ khác nhau để hỗ trợ chỉnh sửa tự động, gỡ lỗi, thử nghiệm, triển khai và quản lý thay đổi. Những công cụ này khiến việc lập trình bằng Java tiết kiệm thời gian và chi phí.

5. Độc lập với nền tảng

Mã Java có thể chạy trên bất kỳ nền tảng cơ sở nào như Windows, Linux, iOS hoặc Android mà không cần viết lại. Đây là điều khiến ngôn ngữ này trở nên đặc biệt mạnh mẽ trong môi trường hiện nay khi chúng ta muốn chạy ứng dụng trên nhiều thiết bị.

6. Bảo mật

Người dùng có thể tải mã Java không tin cậy từ trên mạng xuống và chạy mã này trong môi trường bảo mật để nó không thể gây hại. Mã không tin cậy sẽ không thể lây nhiễm vi-rút cho hệ thống máy chủ và cũng không thể đọc hoặc ghi tệp từ ổ cứng.

1.1.4 Java hoạt động như thế nào

Mọi ngôn ngữ lập trình đều là phương thức để giao tiếp với máy. Phần cứng của máy chỉ phản hồi thông tin giao tiếp điện tử. Các ngôn ngữ lập trình cấp độ cao như Java đóng vai trò là cầu nối giữa ngôn ngữ con người và ngôn ngữ phần cứng. Để sử dụng Java, cần phải hiểu được 2 điều:

1. Ngôn ngữ và API Java

Đây là hoạt động giao tiếp front-end giữa nhà phát triển và nền tảng Java.

2. Máy ảo Java

Đây là hoạt động giao tiếp back-end giữa nền tảng Java và phần cứng cơ sở.

1.1.5 API Java là gì

API là các thành phần phần mềm quan trọng đi kèm với Nền tảng Java. Đây là những chương trình Java viết sẵn có thể "cắm vào là chạy" . Ví dụ: Có thể dùng API Java để nhận ngày giờ, thực hiện các phép toán hoặc điều chỉnh văn bản.

Mọi mã ứng dụng Java do nhà phát triển viết ra thường sẽ kết hợp cả mã mới và cũ từ API và thư viện Java.

1.1.6 Máy ảo Java là gì

Máy ảo Java đóng vai trò là một lớp trừu tượng bổ sung giữa nền tảng Java và phần cứng máy cơ sở. Mã nguồn Java chỉ có thể chạy trên những máy cài đặt JVM. Lịch sử lập trình

Khi những ngôn ngữ lập trình tự nhiên đầu tiên được phát triển, chúng được chia thành 2 loại chính tùy vào cách những ngôn ngữ này giao tiếp với phần cứng cơ sở.

1. Bộ biên dịch: Chương trình hoàn chỉnh được viết bằng cú pháp giống tiếng Anh tự nhiên bằng các bộ biên dịch, sau đó ngôn ngữ sẽ biên dịch (hoặc dịch) toàn bộ mã thành mã máy. Mã được biên dịch lúc này sẽ chạy trên phần cứng.
2. Bộ diễn giải: Đối với các bộ diễn giải, tất cả các câu lệnh mã cấp cao được diễn giải thành mã máy khi phần cứng đang chạy. Phần cứng sẽ chạy ngay những câu lệnh đã được viết trước khi xem xét câu lệnh tiếp theo.

1.1.7 Môi trường thời gian chạy Java

Java là ngôn ngữ đầu tiên kết hợp cả hai phương pháp ở trên bằng một Máy ảo Java (JVM). Bộ biên dịch mã Java được gọi là Máy ảo Java. Trước tiên, mọi tệp Java đều được biên dịch thành bytecode. Bytecode Java chỉ có thể chạy trong JVM. JVM sau đó sẽ diễn giải bytecode để chạy bytecode trên nền tảng phần cứng cơ sở. Như vậy nếu ứng dụng đang chạy trên một máy Windows, JVM sẽ diễn giải ứng dụng đó cho Windows. Nhưng nếu ứng dụng chạy trên một nền tảng mã nguồn mở như Linux, JVM sẽ diễn giải nó cho Linux.

1.1.8 Dịch và thực thi chương trình Java

1. *Viết mã nguồn:* Dùng một chương trình soạn thảo để viết mã nguồn, lưu lại với file tên có đuôi ".java". Tên của file phải đặt giống tên của lớp chính trong chương trình.
2. *Biên dịch ra mã máy ảo:* Dùng trình biên dịch javac để biên dịch mã nguồn ".java" thành mã của máy ảo (java bytecode) có đuôi ".class"

3. *Thông dịch và thực thi:* Việc thông dịch và thực thi dùng lệnh "java".
4. Ví dụ minh họa: Tạo chương trình nguồn

```
/*Chương trình xuất dòng HelloWorld ra Console*/
package ch01;
import java.util.*;

class HelloWorldApp{

public static void main(String[] args){
        //Xuat dong chu "HelloWorld"
        System.out.println("HelloWorld");
    }
}
```

package ch01; ⟶ Dòng đầu tiên khai báo gói chứa chương trình.

import java.util.*; ⟶ Dòng tiếp theo khai báo nạp các lớp sử dụng.

class HelloWorldApp{ ⟶ Khai báo lớp *HelloWordApp* phạm vi toàn cục

public static void main(String[] args){ ⟶ Phương thức main() là điểm bắt đầu thực thi một ứng dụng.

5. Lưu lại với tên HelloWorldApp.java trong thư mục ch01

- Bước 1: Tạo một dự án rỗng (Empty Project): Chọn menu File \ New \ Project. Chọn Empty project → Next. Nhập tên project, chọn Finish.
- Bước 2: Tạo một lớp mới tên HelloWorldApp và đưa vào Project hiện tại: Chọn File \ New \ Class. Nhập tên lớp là HelloWorldApp và chọn Finish.
- Bước 3: Soạn thảo mã nguồn: Cửa sổ WorkSpace - Cửa sổ soạn thảo mã nguồn. Dịch: Bấm F7. Thực thi: Bấm F5

1.2. Một số ứng dụng Java

1. *Java* làm Android

2. Java làm hệ thống giao dịch trong ngành Dịch vụ tài chính
3. Java làm ứng dụng web
4. Java làm phần mềm phát triển: Nhiều phần mềm và công cụ phát triển hữu ích được viết và phát triển bằng Java, ví dụ: Eclipse, InetelliJ Idea và Netbans IDE.
5. Java làm ứng dụng giao dịch
6. Java làm ứng dụng J2ME: Mặc dù sự ra đời của iOS và Android gần như đã giết chết thị trường J2ME, nhưng vẫn có một thị trường của cấp thấp sử dụng J2ME.
7. *Java* làm lập trình nhúng

8. *Java* trong công nghệ Big Data

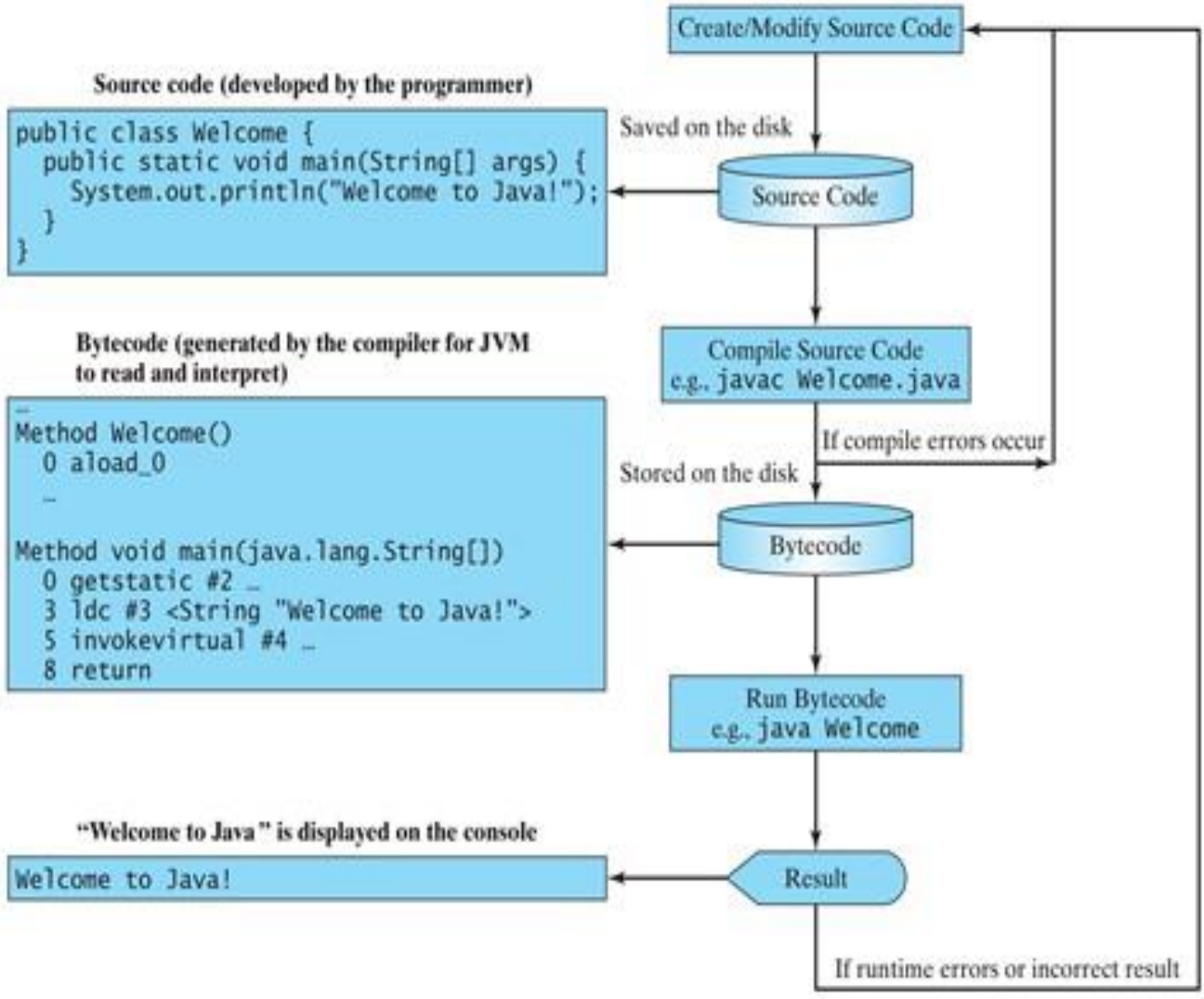
9. Java làm hệ thống hiệu suất cao
10. Java làm ứng dụng khoa học

1.3. Dịch và thực thi một chương trình viết bằng Java

Bên dưới là một chương trình Java được lưu trong file Welcome.java trong folder E:\Demo.

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

Làm thế nào để chạy chương trình Java này? Chúng ta cùng xem quá trình biên dịch và chạy chương trình Java bên dưới.



1. Chúng ta sẽ chạy chương trình bằng các lệnh trong Command Prompt với JDK đã cài đặt. Đầu tiên, thay đổi folder đang thực thi trong Command Prompt đến thư mục chứa file code Java.

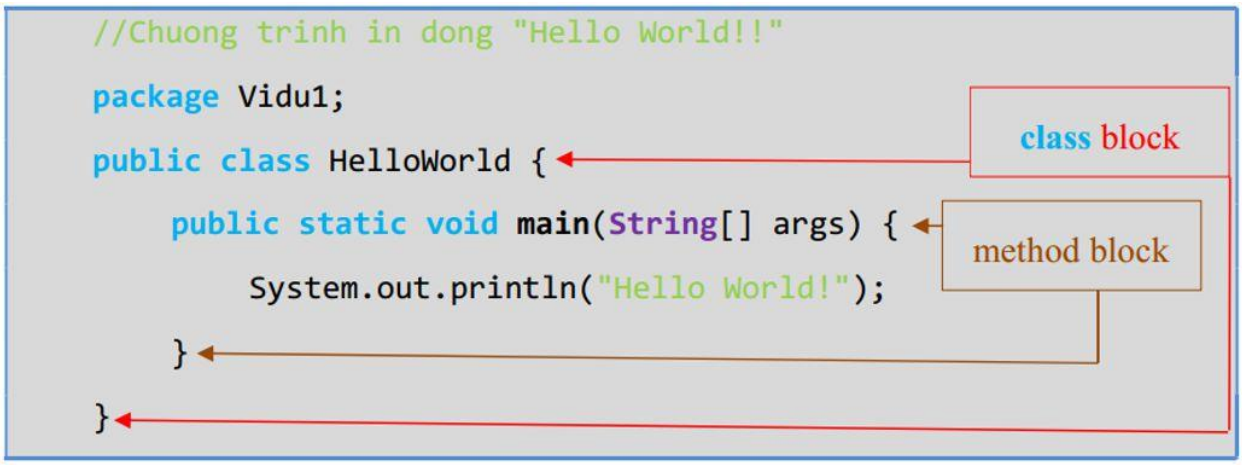
2. Sau đó, sử dụng chương trình javac trong JDK để biên dịch code Java thành bytecode. Một file tên là Welcome.class chứa bytecode sẽ được tự động tạo ra.

3. Cuối cùng, thông dịch bytecode thành mã máy để CPU thực thi với chương trình java trong JDK.

Lưu ý: Thông thường, chúng ta sẽ dùng Java IDE để chạy chương trình Java và không cần phải thực hiện các bước rườm rà trên. Các Java IDE đã hỗ trợ sẵn các thao tác thực thi biên dịch và thông dịch một chương trình Java với chỉ 1 nút nhấn.

1.1. Cấu trúc chung của chương trình Java

Cấu trúc một chương trình đơn giản viết bằng Java



Class block và method block là 2 thành phần chính trong một chương trình Java. Các phương thức (method) trong Java phải luôn được định nghĩa trong lớp (class).

Ngoài ra, một chương trình Java còn có các thành phần khác là chú thích (comments), gói (package), từ khóa (keywords), chỉ định truy cập (access modifiers), câu lệnh (statements), khối (blocks), lớp (classes), phương thức (methods), phương thức main (main method).

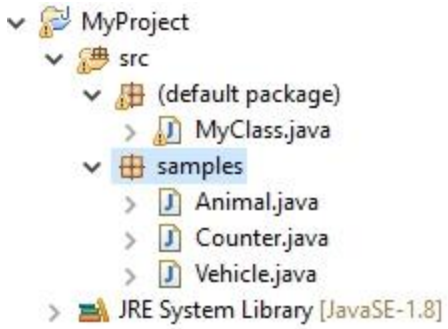
1. Thành phần chú thích (comments)

Trong Java, các chú thích có thể được đặt:

- Sau 2 dấu gạch chéo // trên 1 dòng
- Giữa dấu mở /* và đóng */ trên 1 hoặc nhiều dòng
- Khi trình biên dịch gặp:
- //, nó bỏ qua tất cả các ký tự sau // trên dòng đó
- /*, nó quét tìm đến */ tiếp sau và bỏ qua mọi ký tự giữa /* và */

2. Gói (package)

Một package có thể hiểu như là một tập hợp các lớp, các package con.



Một lớp thuộc package nào thì phải khai báo ở đầu file code định nghĩa lớp đó.

```
package sample;//khai báo package
public class Vehicle {
    private String color;
    // Getter
    public String getColor() {
        return this.color;
    }
    // Setter
    public void setColor(String c) {
        this.color = c;
    }
}
```

Để sử dụng các lớp thuộc package nào đó, phải sử dụng từ khóa import để nạp lớp trong package đó.

```
import samples.Vehicle;
class MyClass {
  public static void main(String[ ] args) {
    Vehicle v1 = new Vehicle();
    v1.setColor("RED");
  }
}
```

Để nạp tất cả các lớp thuộc một package nào đó, có thể dùng ký hiệu đại diện *.

```
import samples.*;
class MyClass {
  public static void main(String[ ] args) {
```

```
    Vehicle v1 = new Vehicle();
    v1.setColor("RED");
  }
}
```

3. Từ khóa (keywords)

Còn gọi là reserved words, là những từ có nghĩa xác định trước đối với trình biên dịch và không thể sử dụng cho các mục đích khác trong chương trình.

Ví dụ: class, public, static, void,…

4. Chỉ định truy cập (access modifiers)

Java sử dụng chỉ định truy cập để xác định phạm vi có thể truy cập của các thuộc tính, phương thức, lớp. Các access modifiers trong Java là public, private, default và protected.

Trong Java còn có các thành phần non-access modifiers như static, abstract, synchronized, native, volatile,… để xác định các thuộc tính, phương thức, lớp có thể được sử dụng như thế nào.

5. Câu lệnh (statements)

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

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

6. Khối (blocks)

Một cặp dấu ngoặc nhọn {} gom các câu lệnh thành một khối lệnh.

Ví dụ: class block, method block,…

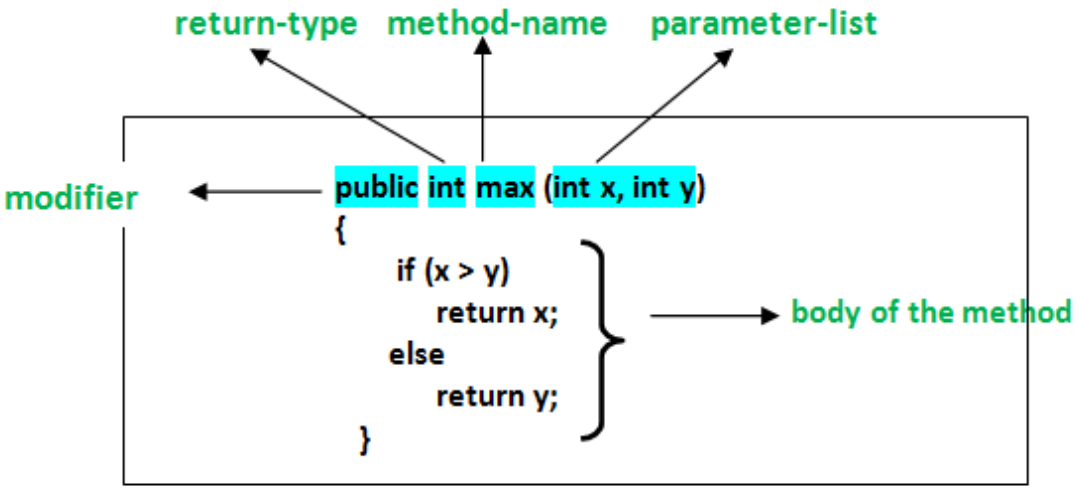
7. Lớp (classes)

Class (lớp) là thành phần thiết yếu trong xây dựng cấu trúc Java. Một class là một khuôn mẫu hay bản thiết kế cho các đối tượng. Một chương trình Java được xác định bằng cách sử dụng một hay nhiều class.

8. Phương thức (methods)

Phương thức (method) là một tập hợp các câu lệnh thực hiện một chức năng cụ thể nào đó trong chương trình.

Hình bên dưới mô tả các thành phần của một phương thức.



Các phương thức (method) trong Java phải luôn được định nghĩa trong lớp (class).

9. Phương thức main (main method)

Main method cung cấp sự kiểm soát luồng chương trình. Trình biên dịch Java thực hiện ứng dụng bằng cách gọi đến main method.

Mọi chương trình Java phải có main method, nó là điểm khởi đầu khi thực hiện chương trình. Dạng thức của main method như sau:

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

1.2. Công cụ lập trình và chương trình dịch

Có 8 công cụ lập trình Java tốt nhất hiện nay

#1. IntelliJ Idea.

#2. Eclipse.

#3. NetBeans.

#4. Oracle JDeveloper.

#5. Android Studio.

#6. MyEclipse.

#7. Jcreator.

#8. DrJava

1.2.1. Các Java IDE thường dùng

IDE (Integrated Development Environment) là một loại phần mềm máy tính có chức năng giúp đỡ các lập trình viên trong việc phát triển phần mềm. Một IDE gồm có các thành phần chính:

- Trình soạn thảo mã nguồn (source code editor): dùng để viết code
- Trình biên dịch (compiler), trình thông dịch (interpreter)
- Công cụ xây dựng tự động: khi sử dụng sẽ biên dịch (hoặc thông dịch) mã nguồn, thực hiện liên kết các thư viện và có thể chạy chương trình một cách tự động
- Trình gỡ lỗi (debugger): hỗ trợ tìm lỗi code

Các Java IDE thường dùng là NetBeans, Eclipse, IntelliJ IDEA,…

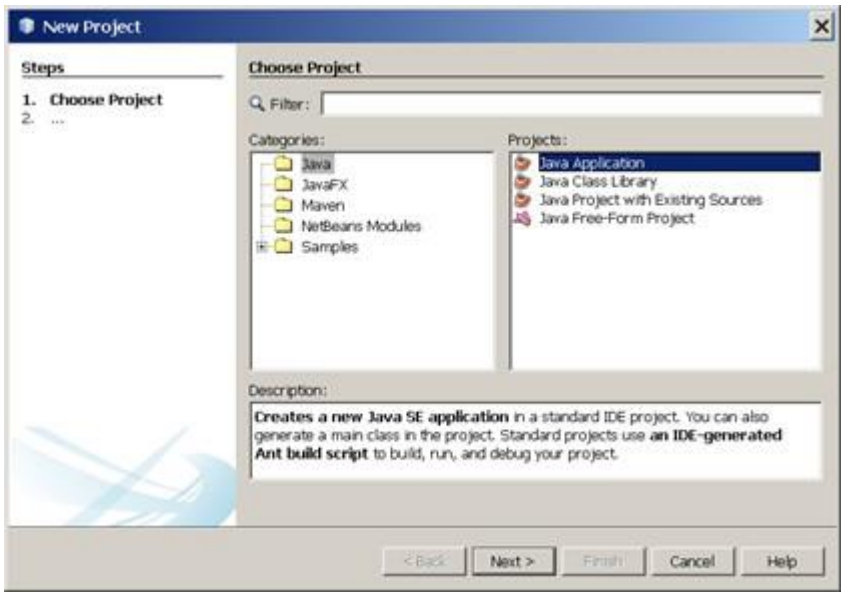
1.2.2. NetBeans IDE

Download NetBeans IDE tại website Apache NetBeans.

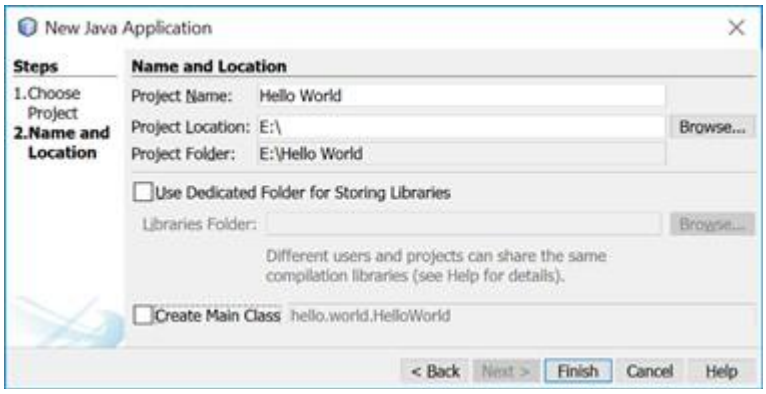
a) Tạo một Java Project

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

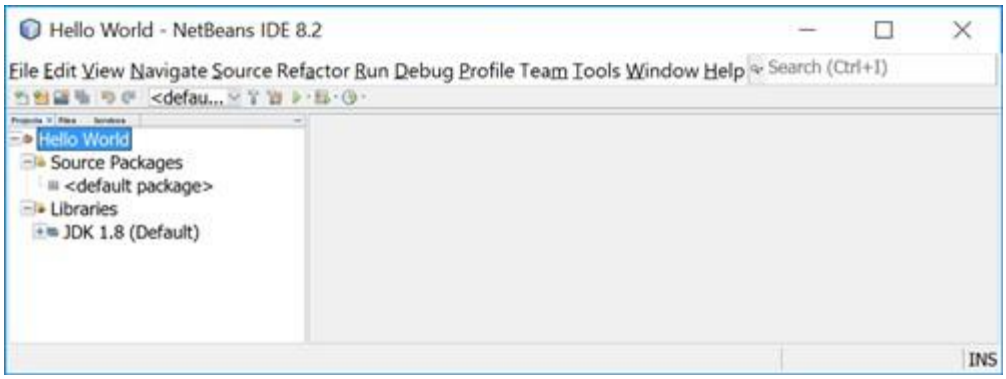
Chọn *File –> New Project*, hộp thoại *New Project xuất hiện.*



Chọn *Java* trong *Categories* và *Java Application* trong *Projects* và click *Next* để hiển thị hộp thoại *New Java Application*.

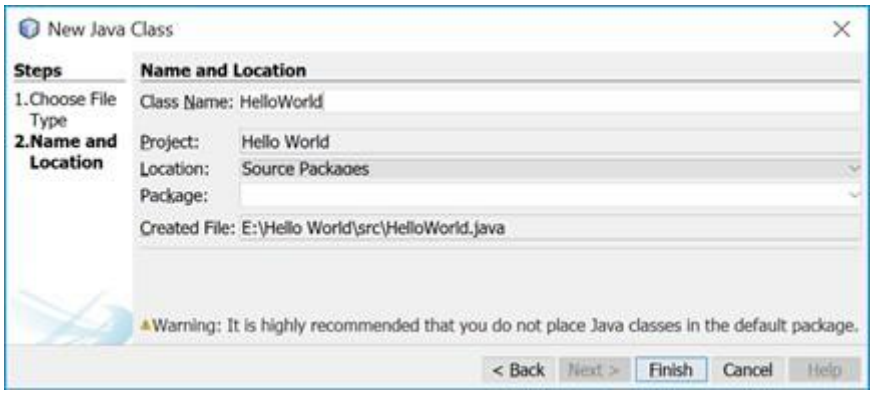


Nhập tên (ví dụ: *Hello World*) trong mục *Project Name* và E:\ trong mục *Project Location*. Click *Finish* để tạo project.



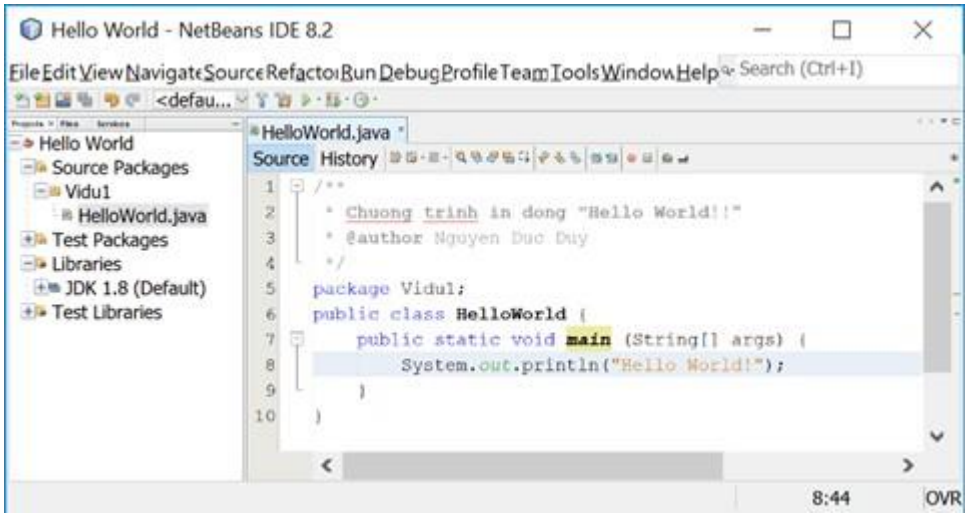
b) Tạo một Java Class

Right click vào *Hello World* trong *Project Pane* để hiển thị *context menu*. Chọn *New* –> *Java Class* để hiển thị hộp thoại *New Java Class*.



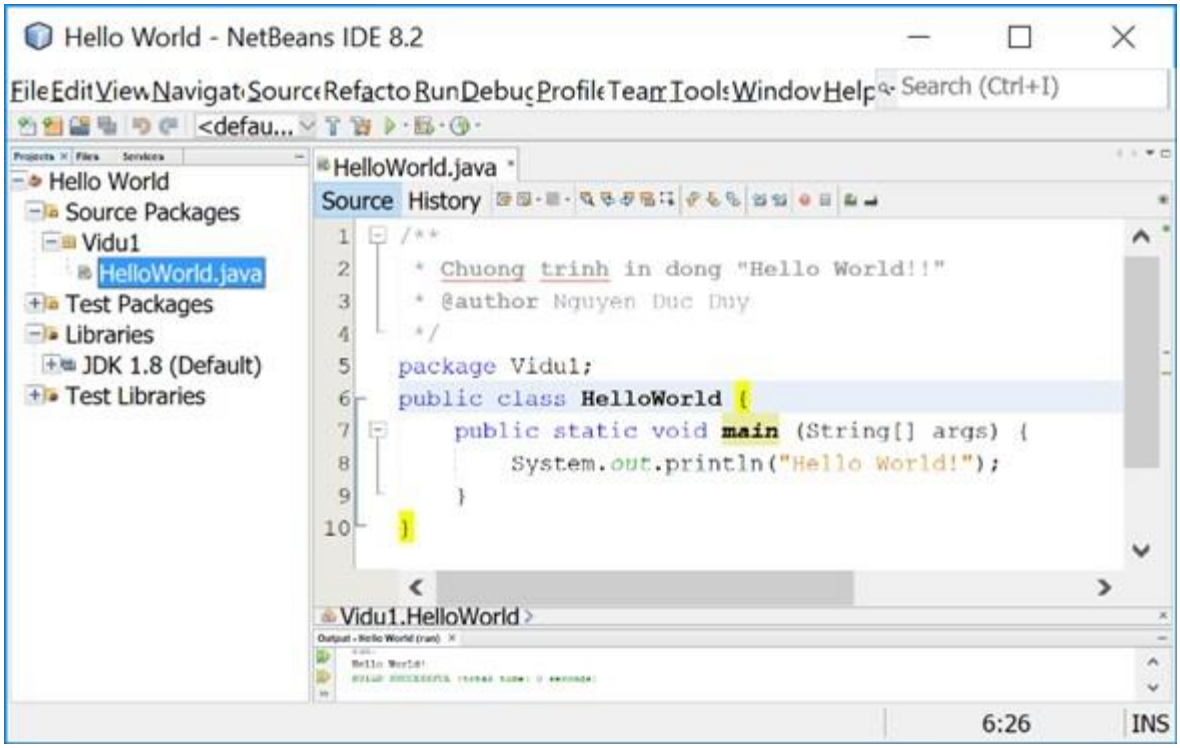
Nhập *HelloWorld* trong mục *Class Name* và chọn *Source Packages* trong mục *Location*. Để trống mục *Package*, sẽ tạo một lớp trong default package. Click *Finish* để tạo *HelloWorld class*.

Bên dưới, tập tin *HelloWorld.java* được đặt trong <package Vidu1>. Soạn thảo code trong *HelloWorld class*



c) Compiling và Running một Class trong NetBeans

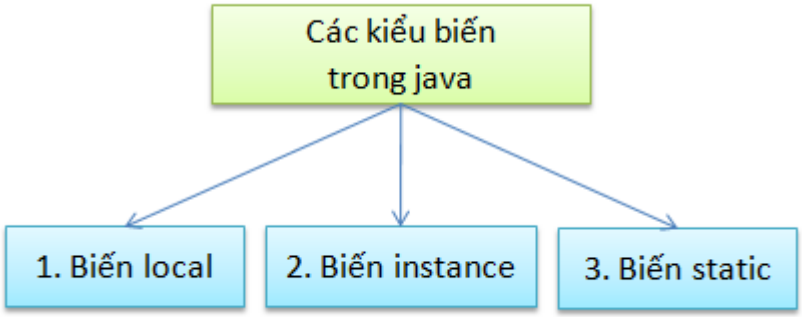
Để chạy *Welcome.java*, right click *Welcome.java* để hiển thị *context menu* và chọn Run File hoặc nhấn Shift + F6.



CHƯƠNG 2: HẰNG, BIẾN, KIỂU DỮ LIỆU, TOÁN TỬ, BIỂU THỨC VÀ CÁC CẤU TRÚC ĐIỀU KHIỂN

2.1. Biến

Biến là tên của vùng nhớ được lưu trong bộ nhớ stack. Có 3 kiểu biến trong java, bao gồm biến cục bộ (hay còn gọi là biến local), biến toàn cục (biến instance) và biến tĩnh(biến static).



a. **Cú pháp khai báo biến:**

DataType varName [= value] [, varName2] [= value2]...;

Trong đó:

- DataType là kiểu dữ liệu của biến
- varName là tên biến.

Quy tắc đặt tên biến trong java:

- Chỉ được bắt đầu bằng một ký tự(chữ), hoặc một dấu gạch dưới(_), hoặc một ký tự dollar($)
- Tên biến không được chứa khoảng trắng
- Bắt đầu từ ký tự thứ hai, có thể dùng ký tự(chữ), dấu gạch dưới(_), hoặc ký tự dollar($)
- Không được trùng với các từ khóa
- Có phân biệt chữ hoa và chữ thường

b. Ví dụ về khai báo biến trong java:

```
package vn.viettuts.bienvadulieu;
 public class Bien {
    public static float PI = 3.14f;   // Đây là biến static
    int n;                            // Đây là biến instance
```

```
    public Bien () {
        char c = 'c';                // Đây là biến local
    }
}
```

c. Biến local trong java

- Biến cục bộ được khai báo trong các phương thức, hàm contructor hoặc trong các block.
- Biến cục bộ được tạo bên trong các phương thức, contructor, block và sẽ bị phá hủy khi kết thúc các phương thức, contructor và block.
- Không được sử dụng "access modifier" khi khai báo biến cục bộ.
- Các biến cục bộ sẽ nằm trên vùng bộ nhớ stack của bộ nhớ.
- Bạn cần khởi tạo giá trị mặc định cho biến cục bộ trước khi có thể sử dụng.

Ví dụ: Khởi tạo biến local:

```
package vn.viettuts.bienvadulieu;
 public class Bien {
        public void sayHello() {
        int n = 10;                    // Đây là biến local
        System.out.println("Gia tri cua n la: " + n);
    }
        public static void main(String[] args) {
        Bien bienLocal = new Bien();
        bienLocal.sayHello();
    }
}
```

Kết quả:

```
Gia tri cua n la: 10
```

Ví dụ: Không khởi tạo biến local

```
package vn.viettuts.bienvadulieu;
 public class Bien {
        public void sayHello() {
        int n;                // Đây là biến local
        System.out.println("Gia tri cua n la: " + n);
```

```
    }
        public static void main(String[] args) {
       Bien bienLocal = new Bien();
       bienLocal.sayHello();
    }
}
```

Kết quả: *Exception in thread "main" java.lang.Error: Unresolved compilation problem:* `The local variable n may not have been initialized`

Khi không khởi tạo biến local, chương trình java sẽ báo lỗi khi biên dịch

d. Biến instance (biến toàn cục) trong java

- Biến instance được khai báo trong một lớp(class), bên ngoài các phương thức, constructor và các block.
- Biến instance được lưu trong bộ nhớ heap.
- Biến instance được tạo khi một đối tượng được tạo bằng việc sử dụng từ khóa "new" và sẽ bị phá hủy khi đối tượng bị phá hủy.
- Biến instance có thể được sử dụng bởi các phương thức, constructor, block, ... Nhưng nó phải được sử dụng thông qua một đối tượng cụ thể.
- Bạn được phép sử dụng "access modifier" khi khai báo biến instance, mặc định là "default".
- Biến instance có giá trị mặc định phụ thuộc vào kiểu dữ liệu của nó. Ví dụ nếu là kiểu int, short, byte thì giá trị mặc định là 0, kiểu double thì là 0.0d, ... Vì vậy, bạn sẽ không cần khởi tạo giá trị cho biến instance trước khi sử dụng.
- Bên trong class mà bạn khai báo biến instance, bạn có thể gọi nó trực tiếp bằng tên khi sử dụng ở khắp nới bên trong class đó.

Ví dụ về biến instance trong java

```
package vn.viettuts.bienvadulieu;
 public class Sinhvien {
   // biến instance "ten" kiểu String, có giá trị mặc định là null
   public String ten;
    // biến instance "tuoi" kiểu Integer, có giá trị mặc định là 0
   private int tuoi;
    // sử dụng biến ten trong một constructor
   public Sinhvien(String ten) {
```

```
        this.ten = ten;
    }
     // sử dụng biến tuoi trong phương thức setTuoi
    public void setTuoi(int tuoi) {
        this.tuoi = tuoi;
    }
     public void showStudent() {
        System.out.println("Ten  : " + ten);
        System.out.println("Tuoi : " + tuoi);
    }
     public static void main(String args[]) {
        Sinhvien sv = new Sinhvien("Nguyen Van A");
        sv.setTuoi(21);
        sv.showStudent();
    }
}
```

Kết quả: Ten : Nguyen Van A

Tuoi : 21

e. Biến static trong java

- Biến static được khai báo trong một class với từ khóa "static", phía bên ngoài các phương thức, constructor và block.
- Sẽ chỉ có duy nhất một bản sao của các biến static được tạo ra, dù bạn tạo bao nhiêu đối tượng từ lớp tương ứng.
- Biến static được lưu trữ trong bộ nhớ static riêng.
- Biến static được tạo khi chương trình bắt đầu chạy và chỉ bị phá hủy khi chương trình dừng.
- Giá trị mặc định của biến static phụ thuộc vào kiểu dữ liệu bạn khai báo tương tự biến instance.
- Biến static được truy cập thông qua tên của class chứa nó, với cú pháp: TenClass.tenBien.
- Trong class, các phương thức sử dụng biến static bằng cách gọi tên của nó khi phương thức đó cũng được khai báo với từ khóa "static".

Ví dụ về biến static trong java

```
package vn.viettuts.bienvadulieu;
 public class Sinhvien {
    // biến static 'ten'
    public static String ten = "Nguyen Van A";

    // biến static 'tuoi'
    public static int tuoi = 21;

    public static void main(String args[]) {
        // Sử dụng biến static bằng cách gọi trực tiếp
        System.out.println("Ten : " + ten);

        // Sử dụng biến static bằng cách gọi thông qua tên class
        System.out.println("Ten : " + Sinhvien.tuoi);
    }
}
```

Kết quả:

Ten : Nguyen Van A

Ten : 21

2.2. Các kiểu dữ liệu cơ sở

Trong Java, kiểu dữ liệu được chia thành hai loại:

- Các kiểu dữ liệu nguyên thủy
- Các kiểu dữ liệu đối tượng

2.2.1. Kiểu dữ liệu nguyên thủy

Java cung cấp các kiểu dữ liệu cơ bản như sau:

Kiểu dữ liệu	Mô tả
byte	Dùng để lưu dữ liệu kiểu số nguyên có kích thước một byte (8 bít). Phạm vi biểu diễn giá trị từ -128 đến 127. Giá trị mặc định là 0.
char	Dùng để lưu dữ liệu kiểu kí tự hoặc số nguyên không âm có kích thước 2 byte (16 bít). Phạm vi biểu diễn giá trị từ 0 đến u\ffff. Giá trị mặc định là 0.

boolean	Dùng để lưu dữ liệu chỉ có hai trạng thái đúng hoặc sai (độ lớn chỉ có 1 bít). Phạm vi biểu diễn giá trị là {“True”, “False”}. Giá trị mặc định là False.
short	Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 2 byte (16 bít). Phạm vi biểu diễn giá trị từ - 32768 đến 32767. Giá trị mặc định là 0.
int	Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 4 byte (32 bít). Phạm vi biểu diễn giá trị từ -2,147,483,648 đến 2,147,483,647. Giá trị mặc định là 0.
long	Dùng để lưu dữ liệu có kiểu số nguyên có kích thước lên đến 8 byte. Giá trị mặc định là 0L.
float	Dùng để lưu dữ liệu có kiểu số thực, kích cỡ 4 byte (32 bít). Giá trị mặc định là 0.0F.
double	Dùng để lưu dữ liệu có kiểu số thực có kích thước lên đến 8 byte. Giá trị mặc định là 0.00D

2.2.2. Kiểu dữ liệu đối tượng

Trong java có 3 kiểu dữ liệu đối tượng:

Kiểu dữ liệu	Mô tả
Array	Một mảng của các dữ liệu cùng kiểu.
class	Dữ liệu kiểu lớp đối tượng do người dùng định nghĩa. Chứa tập các thuộc tính và phương thức..
interface	Dữ liệu kiểu lớp giao tiếp do người dùng định nghĩa. Chứa các phương thức của giao tiếp.

2.3. Hằng

Hằng số là những giá trị không bao giờ thay đổi trong suốt quá trình sử dụng, là một giá trị bất biến trong chương trình.

a) Khai báo hằng số

Ta sử dụng từ khóa static final đặt trước tên hằng số:

[Phạm vi truy cập] static final [kiểu dữ liệu] [tên hằng số] = [giá trị];

Ví dụ:

```
package constant;

public class Constant {
   // Khai báo hằng số
   public static final int HOUR_OF_DAY = 24;
   public static final String CHUOI = "Hello Freetuts!";

   public static void main(String[] args) {
      System.out.println("Một ngày có " + HOUR_OF_DAY + " giờ");
      System.out.println(CHUOI);
   }
 }
```

Kết quả sau khi thực thi chương trình:

```
Một ngày có 24 giờ
Hello Freetuts!
```

2.4. Câu lệnh, khối lệnh trong Java

2.4.1. Câu lệnh

Lệnh hay câu lệnh là tập hợp các từ khóa, tập ký tự trong java và được kết thúc bởi dấu chấm phẩy ;

```
int tuoi = 12; // đây là một dòng lệnh
String ten = "Thaycacac" // đây là dòng lệnh
```

2.4.2. Khối lệnh

Là đoạn chương trình gồm hai lệnh trở lên và được bắt đầu bằng dấu mở ngoặc nhọn { và kết thúc bằng dấu đóng ngoặc nhọc }.

Bên trong một khối lệnh có thể chứa một hay nhiều lệnh hoặc chứa các khối lệnh khác.

```
{ // khối 1
  { // khối 2
    lệnh 2.1;
    lệnh 2.2;
    ...
  } // kết thúc khối lệnh 2
  lệnh 1.1;
```

```
  lệnh 1.2;
  ...
} // kết thúc khối lệnh 1
{ // bắt đầu khối lệnh 3
  // Các lệnh thuộc khối lệnh 3
  // ...
} // kết thúc thối lệnh 3
```

2.5. Các toán tử và biểu thức

2.5.1. Toán tử

Toán tử trong java là một ký hiệu được sử dụng để thực hiện một phép tính/chức năng nào đó. Java cung cấp các dạng toán tử sau:

- Toán tử số học
- Toán tử bit
- Toán tử quan hệ
- Toán tử logic
- Toán tử điều kiện
- Toán tử gán

2.5.2. Biểu thức

Biểu thức bao gồm các các biến, toán tử và lời gọi phương thức và tính toán cho ra một kết quả nào đó.

2.5.3. Các cấu trúc điều khiển trong Java

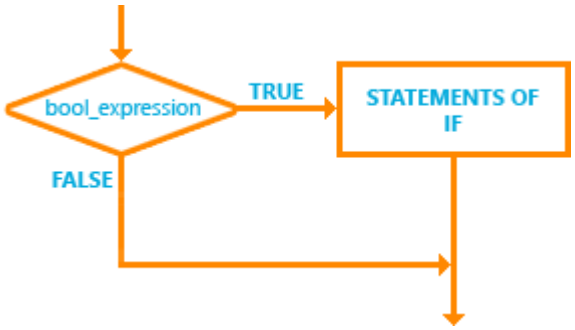
2.5.3.1. Cấu trúc rẽ nhánh if, if-else

a) Cấu trúc rẽ nhánh cơ bản if

- **Cú pháp**

```
if(conditional) {
statement1;
statement2;
}
```

- **Mô tả thực thi**



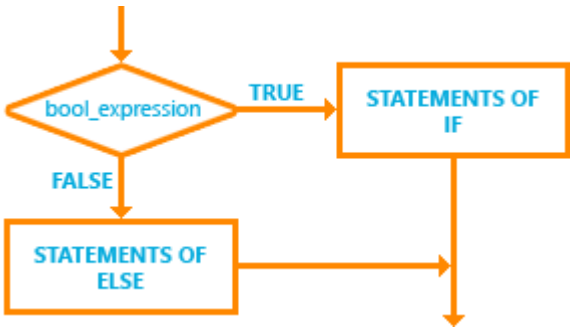
- Nếu conditional trả về true, thực thi khối statement1 và statement2.
- Nếu conditional trả về false thì bỏ qua khối này.

b) Cấu trúc rẽ nhánh mở rộng if-else

- **Cú pháp**

```
if(conditional) {
   statement1;
   statement2;
}
else {
   statement3;
   statement4;
}
```

- **Mô tả thực thi**



Nếu conditional trả về true, thực thi khối statement1 và statement2.
Nếu conditional trả về false, thực thi khối statement3 và statement4.

- **Ví dụ**

```
int x = 3;
```

```
if(x > 2) {
   System.out.println("X > 2");
}
else {
   System.out.println("X <= 2");
}
```

d) Cấu trúc if-else lồng nhau

- **Cú pháp**

```
if(conditional) {
   statement1;
}
else {
   if(conditional2) {
      statement2;
   }
   else {
      statement3;
   }
}
```

- Nếu conditional trả về true, thực thi statement1.
- Nếu conditional trả về false, xét tiếp khối else như sau:
  - Nếu conditional2 trả về true,thực thi statement2.
  - Nếu conditional2 trả về false, thực thi statement3.

2.6.3.2. Vòng lặp for

- **Cú pháp:**

```
for (<khởi đầu>; <điều kiện lặp>; <bước nhảy>) {
   //code block
}
```

- **Nguyên lý hoạt động của vòng lặp for**

✓ <khởi đầu> khởi tạo giá trị của biến và chỉ thực thi một lần duy nhất.

✓ Sau đó, nếu <điều kiện lặp> đúng (true) thì thực thi các dòng lệnh trong vòng lặp for. Nếu <điều kiện lặp> sai (false) thì kết thúc vòng lặp.

- ✓ <bước nhảy> sẽ thay đổi giá trị của biến lúc <khởi đầu>. Giá trị của biến này sẽ được kiểm tra lại (kiểm tra <điều kiện lặp>) sau mỗi lần lặp.

Ví dụ bên dưới sẽ in ra "Java is fun" 5 lần.

- **Ví dụ:**

```
class Main {
   public static void main(String[] args) {
      int n = 5;
      // for loop
      for (int i = 1; i <= n; ++i) {
        System.out.println("Java is fun");
      }
   }
}
```

- **Kết quả:**

```
Java is fun
Java is fun
Java is fun
Java is fun
Java is fun
```

Chú ý: Giá trị khởi tạo của vòng lặp for <khởi đầu> là bất kỳ.

- **Ví dụ:**

```
class Main {
   public static void main(String[] args) {
      int n = 5;
      // for loop
      for (int i = 0; i < n; ++i) {
        System.out.println("Java is fun");
      }
   }
}
```

2.6.3.3. Vòng lặp for each

Để dễ dàng duyệt qua các mảng (array) và tập hợp (collection) ta sử dụng vòng lặp for-each.

- **Cú pháp:**

```
for(dataType item : array) {
   ...
}
```

- **Trong đó:**

✓ array là một mảng hay một tập hợp.

✓ item là mỗi phần tử trong mảng hay tập hợp

✓ dataType là kiểu dữ liệu của các phần tử trong mảng hay tập hợp

- **Ví dụ:**

```
class Main {
   public static void main(String[] args) {
      // create an array
      int[] numbers = {3, 9, 5, -5};
      // for each loop
      for (int number: numbers) {
        System.out.println(number);
      }
   }
}
```

Kết quả:

```
3
9
5
-5
```

Chương trình trên sử dụng for-each để duyệt mảng. Ở lần lặp thứ 1 thì item là 3, lần lặp thứ 2 thì item là 9, lần lặp thứ 3 thì item là 5, lần lặp thứ 4 thì item là -5.

Lưu ý: Vòng lặp for và for-each cho kết quả duyệt mảng như nhau, nhưng for-each ngắn gọn và dễ hiểu hơn.

2.6.3.4. Vòng lặp while

- **Cú pháp:**

```
while (<điều kiện lặp>)
{
    //code block
}
```

<điều kiện lặp> thường là biểu thức với các toán tử quan hệ, kết quả trả về là true (0) hoặc false (0).

- **Nguyên lý hoạt động của vòng lặp while như sau:**

Kiểm tra <điều kiện lặp>, nếu <điều kiện lặp> đúng (true) thì dòng lệnh trong while sẽ được thực thi.

Quá trình kiểm tra <điều kiện lặp> và thực thi dòng lệnh trong while sẽ chấm dứt cho đến khi <điều kiện lặp> sai (false). Tức là, nếu bất cứ khi nào <điều kiện lặp> sai (false) thì vòng lặp while sẽ chấm dứt.

Ví dụ:

```
class Main {
   public static void main(String[] args) {
      // declare variables
      int i = 1, n = 5;
      // while loop from 1 to 5
      while(i <= n) {
        System.out.println(i);
        i++;
      }
   }
}
```

Kết quả:

```
1
2
3
4
5
```

2.6.3.5. Vòng lặp do while

- **Cú pháp:**

```
do {
```

//code block

}while (<điều kiện lặp>);

<điều kiện lặp> thường là biểu thức với các toán tử quan hệ, kết quả trả về là true (0) hoặc false (0).

- **Nguyên lý hoạt động**

Thực thi dòng lệnh trong do while trước. Sau đó, kiểm tra <điều kiện lặp>, nếu <điều kiện lặp> đúng (true) thì thực thi dòng lệnh trong do while lần nữa.

Quá trình kiểm tra <điều kiện lặp> và thực thi dòng lệnh trong do while sẽ chấm dứt cho đến khi <điều kiện lặp> sai (false). Tức là, nếu bất cứ khi nào <điều kiện lặp> sai (false) thì vòng lặp do while sẽ chấm dứt.

Ví dụ:

```
class Main {
   public static void main(String[] args) {
      int i = 1, n = 5;
      // do...while loop from 1 to 5
      do {
        System.out.println(i);
        i++;
      } while(i <= n);
   }
}
```

Kết quả

```
1
2
3
4
5
```

2.5.4. Dữ liệu kiểu mảng

2.5.4.1. Mảng một chiều

Mảng là một tập hợp các phần tử có kiểu tương tự nhau mà có vị trí ô nhớ liền kề. Mảng trong Java là một đối tượng chứa các phần tử có kiểu dữ liệu giống nhau.

a) Khai báo

[kiểu dữ liệu] [] [tên mảng] ;

- **Ví dụ**

```
public class Test {
   public static void main(String[] args){
      int []a;
   }
}
```

- **Có thể khởi tạo mảng bằng cách dùng từ khóa new bằng cú pháp:**

[kiểu_dữ_liệu] [] [tên_mảng] = new [kiểu_dữ_liệu] [kích_thước_mảng] ;

Bằng cú pháp này cho phép vừa khai báo mảng vừa khởi tạo mảng.

Nếu đã khai báo mảng thì có thể khởi tạo mảng bằng cú pháp sau.

[tên_mảng] = new [kiểu_dữ_liệu] [kích_thước_mảng];

Ví dụ:

```
public class Test {
   public static void main(String[] args){
      int []a;//khai báo mảng
      a = new int [5];//khởi tạo mảng
   }
}
```

b) Truy cập đến phần tử của mảng

tên_mảng[vị_trí] ;

Lưu ý: mảng bắt đầu bằng từ phần tử 0 đến kích_thước_mảng -1.

Ví dụ:

```
public class Test {
   public static void main(String[] args){
      int []a;
      a = new int [5];
      for(int i=0;i< a.length;i++){
         a[i] = i;
      }
   }
}
```

Trong ví dụ trên i chạy từ 1 đến 4 và lần lượt gán giá trị của chính nó cho phần tử thứ i của mảng. Thuộc tính length để xác định kích thước mảng.

In giá trị của mảng a ra màn hình

```
public class Test {
   public static void main(String[] args){
      int []a;
      a = new int [5];
      for(int i=0; i< a.length;i++){
         a[i] = i;
      }
      for(int i=0;i < 5;i++){
         System.out.println("Gia tri cua a"+i+" la: "+a[i]);
      }
   }
}
```

```
Gia tri cua a0 la: 0
Gia tri cua a1 la: 1
Gia tri cua a2 la: 2
Gia tri cua a3 la: 3
Gia tri cua a4 la: 4
```

c) Nhập xuất mảng một chiều

Sử dụng vòng lặp for để nhập các phần tử của mảng từ bàn phím. Sau đó, duyệt mảng để xuất các phần tử của mảng ra màn hình.

Ví dụ:

```
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
      double[] myList = new double[5];
      Scanner input = new Scanner(System.in);
      System.out.println("Nhap " + myList.length + " cua mang:");
     for (int i = 0; i < myList.length; i++) {
         myList[i] = input.nextDouble();
      }
```

```
        System.out.print("Cac phan tu trong mang: ");
        for (int i = 0; i < myList.length; i++) {
            System.out.print(myList[i] + " ");
        }
    }
}
```

d) Một số thao tác trên mảng một chiều

3.3.1. Tính tổng các phần tử trong mảng

Sử dụng vòng lặp for duyệt qua từng phần tử trong mảng rồi cộng dồn các phần tử đó.

```
class Main {
    public static void main(String[] args) {
        double[] myList = {2.3, 5.0, 7.1, 5.5, 9.2};
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
                total += myList[i];
        }
        System.out.print("Tong cac phan tu trong mang = " + total);
    }
}
```

3.3.2. Tìm phần tử nhỏ nhất trong mảng

Gán phần tử đầu tiên là phần tử min. So sánh phần tử min với từng phần tử trong mảng. Nếu một phần tử trong mảng nhỏ hơn min thì gán min bằng phần tử đó.

```
class Main {
    public static void main(String[] args) {
        double[] myList = {2.3, 5.0, 7.1, 5.5, 9.2};
        double min = myList[0];
        int indexMin = 0;
        for (int i = 0; i < myList.length; i++) {
                if (myList[i] < min){
                  min = myList[i];
                  indexMin = i;
                }
```

```
        }
        System.out.println("Phan tu min trong mang = " + min);
        System.out.println("Chi so phan tu min trong mang = " + indexMin);
    }
}
```

e) **Duyệt mảng với vòng lặp for-each**

Ví dụ:

```
class Main {
    public static void main(String[] args) {
        double[] myList = {2.3, 5.0, 7.1, 5.5, 9.2};
        double sum = 0;
        double average = 0;
        //tính tổng các phần tử trong mảng
        for (double number: myList) {
            sum += number;
        }
        //lấy kích thước của mảng
        int arrayLength = myList.length;
        //tính trung bình cộng các phần tử trong mảng
        average =  sum/arrayLength;
        System.out.println("Tong = " + sum);
        System.out.println("Trung binh = " + average);
    }
}
```

CHƯƠNG 3: HƯỚNG ĐỐI TƯỢNG TRONG JAVA

3.1. Lớp

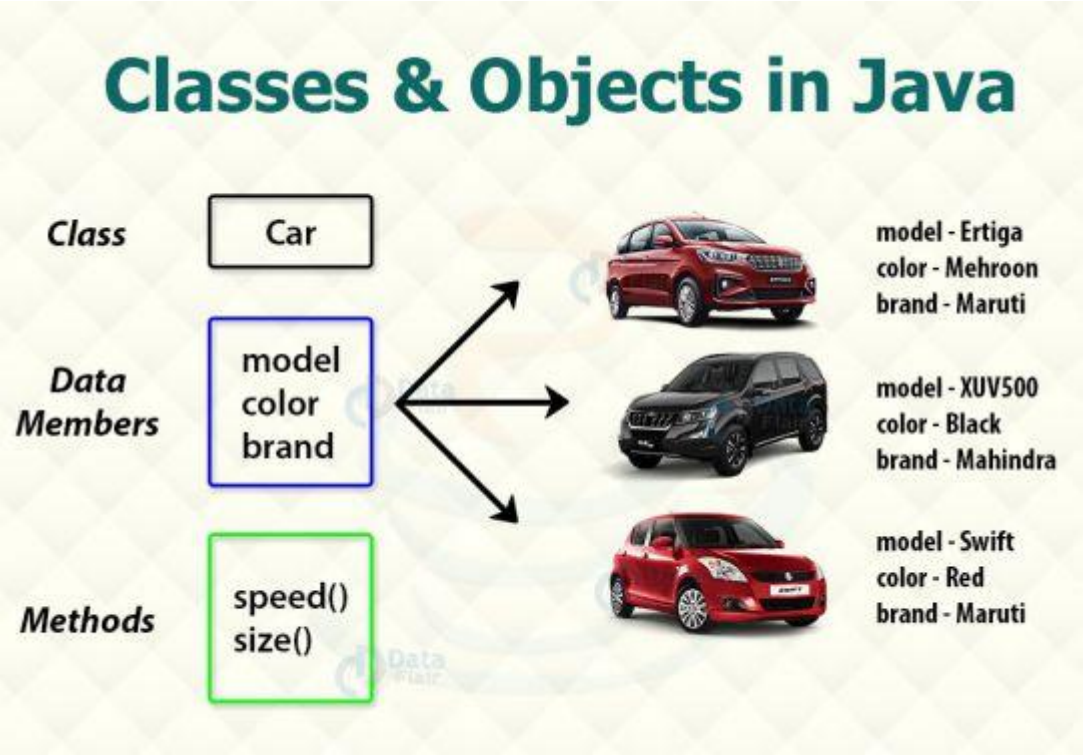
Lớp (class) là một thiết kế (blueprint), mẫu (prototype) cho các đối tượng cùng kiểu. Lớp có thể coi là khuôn mẫu để tạo các đối tượng. Ví dụ: Lớp: Người, Xe, Động vật,…

Lớp chính là kết quả của quá trình trừu tượng hóa dữ liệu:

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

Lớp đóng gói các thuộc tính (attribute) và phương thức (method) chung của các đối tượng.

Ví dụ:



3.1.1. Các thành phần của lớp

- Thuộc tính:

- Một thuộc tính của một lớp là một trạng thái chung được đặt tên của lớp đó. Ví dụ: Lớp Ôtô có các thuộc tính: màu sắc, vận tốc, hãng sản xuất,…
- Mỗi đối tượng của lớp có các giá trị của thuộc tính khác nhau. Ví dụ: một chiếc Ôtô bạn đang sử dụng có thể có màu đen, vận tốc 60 km/h.

- Phương thức:

- Xác định các hoạt động chung mà tất cả các đối tượng của lớp thực hiện được.
- Ví dụ: Lớp Ôtô có các phương thức: tăng tốc độ, giảm tốc độ,…

3.1.2. Khai báo

Lớp sẽ nằm trong một package (gói) nào đó. Sử dụng từ khóa class để khai báo và định nghĩa lớp.

3.3.3. Cú pháp:

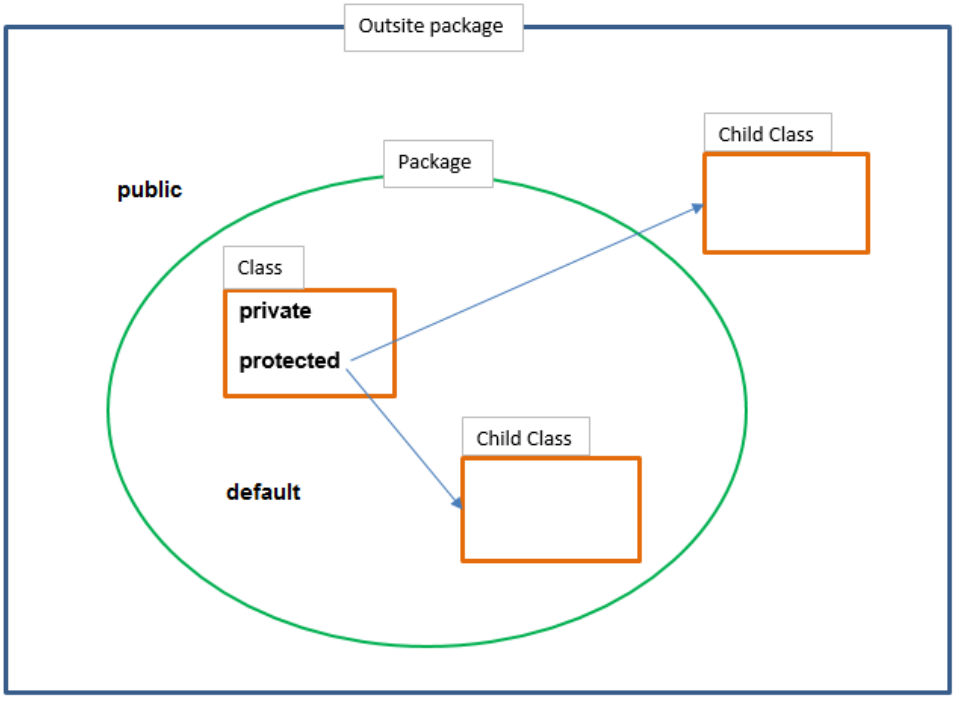
```
accessmodifier class <Tên Lớp> {
    //các thuộc tính
    //các phương thức
}
```

accessmodifier gọi là chỉ định truy cập, accessmodifier của lớp (class) có thể là:

- public: lớp có thể được truy cập từ bất cứ đâu, kể cả bên ngoài package chứa lớp đó.
- mặc định (default): lớp có thể được truy cập từ bên trong package chứa lớp đó. Khi không ghi rõ accessmodifier cho lớp thì đó là mặc định.

Access Modifier	Trong lớp	Trong package	Ngoài package bởi lớp con	Ngoài package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Minh họa



Ví dụ:

Khai báo và định nghĩa lớp với accessmodifier mặc định.

```
package gochocit.Demo;
```

```
class Dog {
    //thuộc tính
    private String ten;
    private String maulong;
    //phương thức
    public void bark(){
        System.out.println("BARRRRK!");
    }
}
```

Khai báo và định nghĩa lớp với accessmodifier public.

```
package gochocit.Demo;
public class Dog {
    //thuộc tính
    private String ten;
    private String maulong;
    //phương thức
    public void bark(){
        System.out.println("BARRRRK!");
    }
}
```

3.3.4. Thuộc tính của lớp

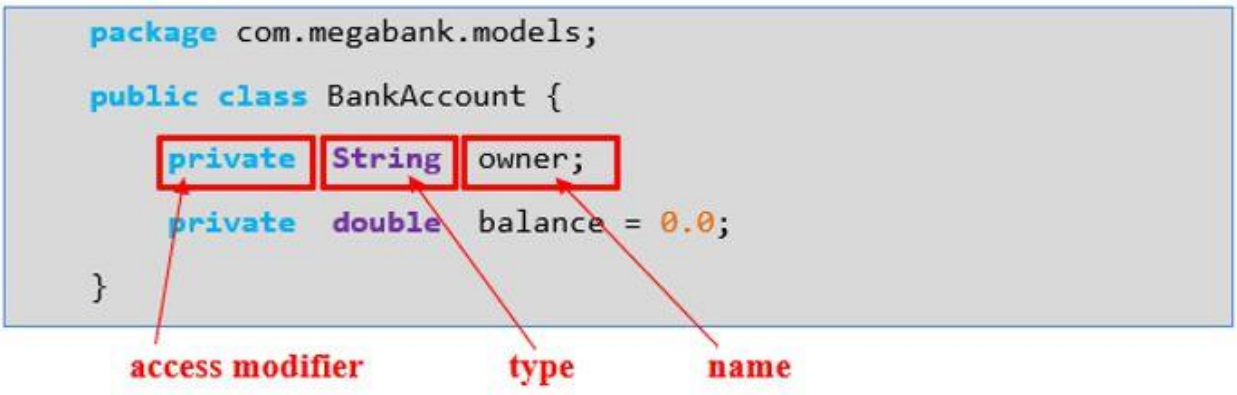
Các thuộc tính phải được khai báo bên trong lớp. Mỗi đối tượng có bản sao các thuộc tính của riêng nó. Giá trị của một thuộc tính thuộc các đối tượng khác nhau là khác nhau.

Bản chất của các thuộc tính là các thành phần dữ liệu của đối tượng. Khai báo tương tự như biến.

3.3.5. Cú pháp

accessmodifier kiểudữliệu tênThuộcTính;

Thuộc tính có thể được khởi tạo khi khai báo. Nếu không được khởi tạo thì các giá trị mặc định sẽ được sử dụng.



3.2. Đối tượng

Sau khi định nghĩa lớp, ta có thể khai báo các biến thuộc kiểu lớp để tạo các đối tượng.

3.2.1. **Cú pháp:**

<Tên_lớp> danh_sách_đối_tượng;

- **Ví dụ:**

Circle c1, c2;

Có thể khai báo các đối tượng khi định nghĩa lớp theo cú pháp sau:

class <Tên_lớp>{

...

}<danh_sách_đối_tượng>;

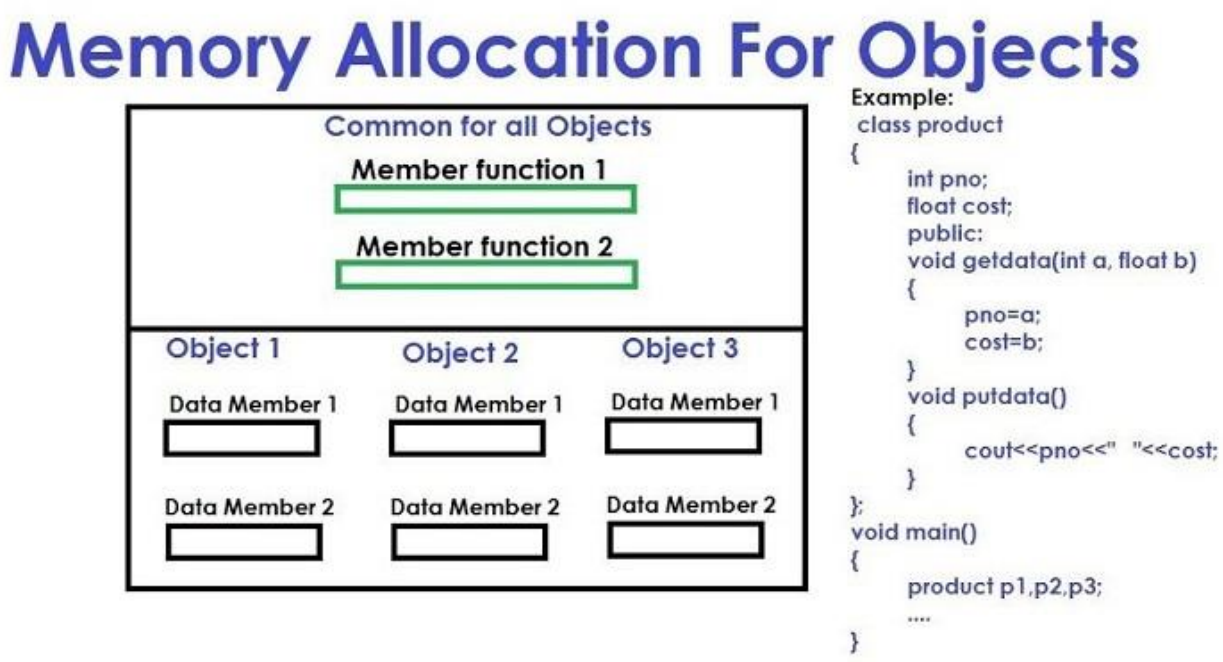
- **Vi dụ:**

class Circle{

...

} c1, c2;

Mỗi đối tượng sau khi khai báo sẽ được cấp phát một vùng nhớ riêng để lưu trữ các thuộc tính của nó. Không có vùng nhớ riêng để chứa các hàm thành phần của đối tượng. Các hàm thành phần được sử dụng chung cho tất cả các đối tượng cùng lớp.



3.2.2. Truy cập các thành phần của đối tượng

Sau khi khai báo đối tượng (object), ta sử dụng toán tử chấm "." để truy nhập đến các thành phần của đối tượng.

- **Cú pháp:**

Tên_đối_tượng.Tên_thuộc_tính;

Tên_đối_tượng.Tên_hàm(giá_trị_các_khai_báo_tham_số);

Lưu ý:

Các thành phần có khai báo là private chỉ có thể được truy cập bởi những hàm thành phần của cùng một lớp, đối tượng của lớp không thể truy cập trực tiếp được.

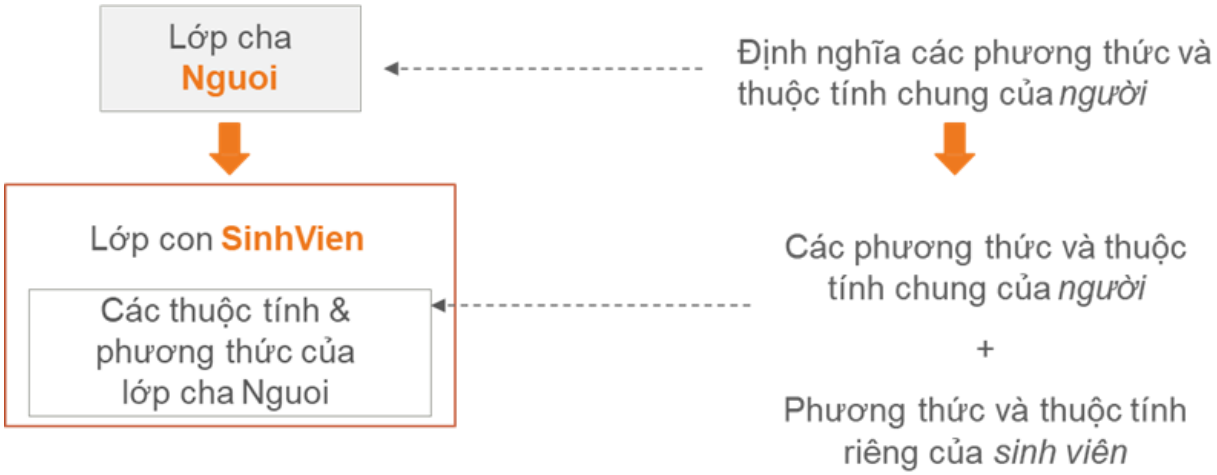
3.3. Kế thừa

Bản chất kế thừa (inheritance) là phát triển lớp mới dựa trên các lớp đã có. Xây dựng các lớp mới có sẵn các đặc tính của lớp cũ, đồng thời lớp mới có thể mở rộng các đặc tính của nó.

- **Ví dụ:**

- Lớp Người có các thuộc tính như tên, tuổi, chiều cao, cân nặng,…; các phương thức như ăn, ngủ, chơi,…

- Lớp Sinh Viên thừa kế từ lớp Người, thừa kế được các thuộc tính tên, tuổi, chiều cao, cân nặng,…; các phương thức ăn, ngủ, chơi,…Bổ sung thêm các thuộc tính như mã số sinh viên, số tín chỉ tích lũy,…, các phương thức như tham dự lớp học, thi…



Lớp kế thừa gọi là lớp con (child, subclass), lớp dẫn xuất (derived class). Lớp được kế thừa gọi là lớp cha (parent, superclass), lớp cơ sở (base class).

3.3.1. Mối quan hệ kế thừa

Lớp con là một loại (is-a-kind-of) của lớp cha, kế thừa các thành phần dữ liệu và các hành vi của lớp cha. Có thể khai báo thêm thuộc tính, phương thức cho phù hợp với mục đích sử dụng mới.

Ví dụ:

Lớp SinhVien kế thừa từ lớp Nguoi, lớp GiaoVien kế thừa từ lớp Nguoi thì một đối tượng SinhVien là một Nguoi, một đối tượng GiaoVien là một Nguoi. Nhưng một đối tượng Nguoi chưa chắc là SinhVien bởi vì có thể đối tượng Nguoi đó là GiaoVien. Tương tự, môt đối tượng Nguoi chưa chắc là GiaoVien bởi vì có thể đối tượng Nguoi đó là SinhVien.

Bản chất kế thừa là một kỹ thuật tái sử dụng mã nguồn thông qua lớp.

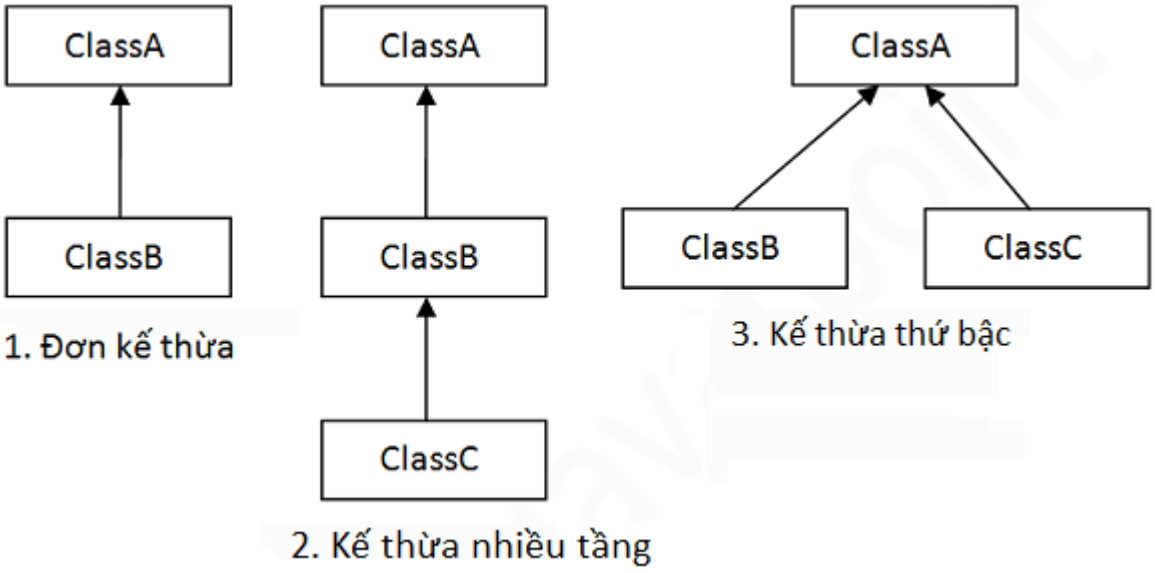
Ví dụ:

Lớp SinhVien tái sử dụng được các thuộc tính như tên, tuổi… và các phương thức của lớp Nguoi.

3.3.2. Các loại kế thừa trong Java

- Đơn kế thừa (single inheritance): Một lớp con kế thừa từ một lớp cha.
- Kế thừa nhiều cấp (multilevel inheritance): Một lớp con kế thừa từ một lớp cha. Lớp cha đó lại kế thừa từ một lớp khác.

- Kế thừa phân cấp (hierarchical inheritance): Nhiều lớp con kế thừa từ một lớp cha. Các lớp con có cùng lớp cha gọi là các lớp anh chị em (siblings).



Java không hỗ trợ đa kế thừa (multiple Inheritance)

3.3.3. Khai báo

- **Cú pháp:**

```
<Lớp con> extends <Lớp cha>{
    //thân lớp con
}
```

- **Ví dụ:**

```
class Nguoi {
   String name;
   //hàm đi chơi
   void diChoi(){
      System.out.println(name + " di choi thoi!");
   }
}
class SinhVien extends Nguoi {
```

```
    int studentId;
    //hàm đi học
    void diHoc(){
        System.out.println(name + " mssv: " + studentId + " di hoc thoi!");
    }
}
public class Main {
    public static void main(String args[]){
        SinhVien sv1 =  new SinhVien();
        sv1.name = "An";
        sv1.studentId = 123456789;
        sv1.diChoi();
        sv1.diHoc();
    }
}
```

Kết quả:

An di choi thoi!

An mssv: 123456789 di hoc thoi!

Trong ví dụ trên, lớp SinhVien kế thừa từ lớp Nguoi. Lớp SinhVien kế thừa thuộc tính String name và hàm diChoi() từ lớp Nguoi.

Lớp con kế thừa các thành viên được khai báo là public, protected và mặc định (default) của lớp cha. Không kế thừa được các thành viên private.

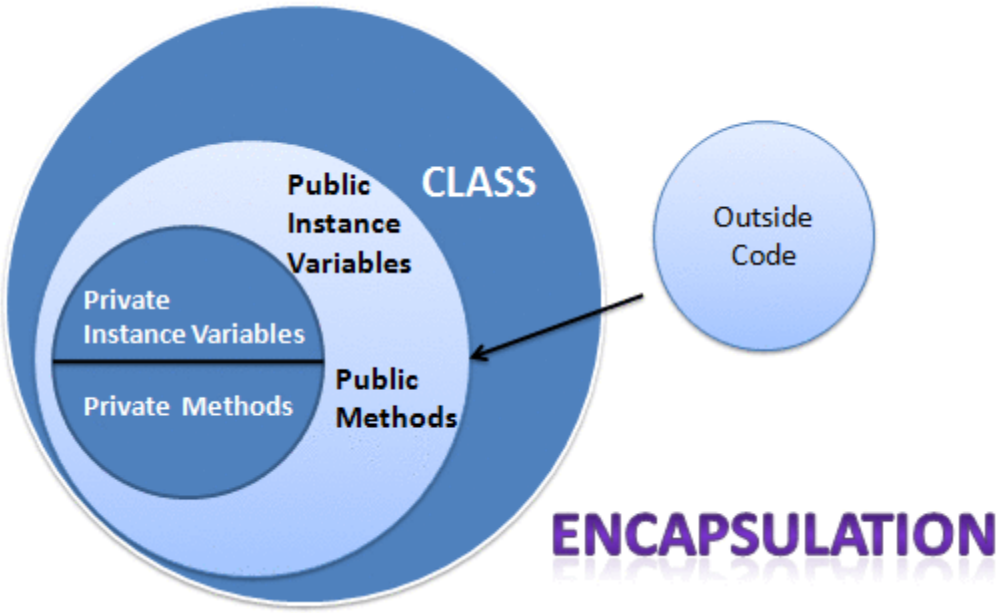
3.4. Đóng gói

Việc ràng buộc giữa code và data với nhau tạo thành một khối duy nhất được gọi là đóng gói.

Ví dụ: viên thuốc con nhộng được đóng gói với nhiều loại thuốc bên trong.

Một class trong java là một ví dụ về đóng gói. Java bean là một lớp được đóng gói hoàn toàn vì tất cả các dữ liệu thành viên là private.

Tính đóng gói trong java là kỹ thuật ẩn giấu thông tin không liên quan và hiện thị ra thông liên quan. Mục đích chính của đóng gói trong java là giảm thiểu mức độ phức tạp phát triển phần mềm.



Đóng gói cũng được sử dụng để bảo vệ trạng thái bên trong của một đối tượng. Bởi việc ẩn giấu các biến biểu diễn trạng thái của đối tượng. Việc chỉnh sửa đối tượng được thực hiện, xác nhận thông qua các phương thức. Hơn nữa, việc ẩn giấu các biến thì các lớp sẽ không chia sẻ thông tin với nhau được. Điều này làm giảm số lượng khớp nối có thể có trong một ứng dụng.

- **Lợi ích của đóng gói trong java**

Có thể tạo lớp read-only hoặc write-only bằng việc cài đặt phương thức setter hoặc getter.

Kiểm soát đối với dữ liệu. Giả sử muốn đặt giá trị của id chỉ lớn hơn 100 ta có thể viết logic bên trong lớp setter.

Ví dụ về đóng gói trong java

ví dụ đóng gói trong java với một lớp chỉ có một trường và các phướng thức setter và getter của nó.

File: Student.java

```
public class Student {
   private String name;

   public String getName() {
      return name;
   }
```

```
    public void setName(String name) {
        this.name = name;
    }
}
File: Test.java
class Test {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Hai");
        System.out.println(s.getName());
    }
}
```

Kết quả:

```
Hai
```

3.5. Đa hình

Tính đa hình (polymorphism) hiểu đơn giản là các đối tượng, các phương thức giống nhau có thể có các hành vi khác nhau tùy vào từng tình huống khác nhau.

Ví dụ:

Main.java

```
class Polygon {
  //phương thức render của lớp Polygon
  public void render() {
    System.out.println("Rendering Polygon...");
  }
}
class Square extends Polygon {
  //ghi đè phương thức render
  @Override
  public void render() {
    System.out.println("Rendering Square...");
  }
```

```
}
class Circle extends Polygon {
  //ghi đè phương thức render
  @Override
  public void render() {
    System.out.println("Rendering Circle...");
  }
}
class Main {
  public static void main(String[] args) {
    // create an object of Square
    Square s1 = new Square();
    s1.render();
    // create an object of Circle
    Circle c1 = new Circle();
    c1.render();
  }
}
```

Kết quả

```
Rendering Square...
Rendering Circle...
```

Ở ví dụ trên, lớp Polygon có 2 lớp con là Square and Circle. Chúng đều có hàm render() nhưng hàm này thực thi khác nhau trong từng lớp. Đó là biểu hiện của tính đa hình (polymorphism).

- **Tại sao sử dụng tính đa hình (polymorphism)**

Tính đa hình (polymorphism) trong Java cho phép chúng ta tạo ra những mã code nhất quán. Lấy ví dụ ở phần 1, chúng ta có thể tạo ra các hàm render với các tên khác nhau như renderSquare() và renderCircle() cho từng lớp Square và Circle.

Chương trình cũng sẽ hoạt động tốt thôi nhưng việc tạo ra các tên khác nhau cho một hàm có chức năng giống nhau thì sẽ làm code không nhất quán, rườm rà, cồng kềnh.

Tính đa hình cho phép chúng ta tạo ra một hàm tên là render() duy nhất nhưng sẽ thực thi khác nhau với từng lớp khác nhau.

Trong Java, chúng ta có thể đạt được tính đa hình (polymorphism) với nạp chồng phương thức (method overloading), ghi đè phương thức (method overriding).

Lưu ý: Trong Java không hỗ trợ lập trình viên tự định nghĩa nạp chồng toán tử (operator overloading) như trong C++.

3.6. Giao diện

Một "lớp trừu tượng" được sử dụng để nhóm các phương thức với các phần thân trống

Trong Java, một giao diện là một kiểu tham chiếu tương tự như một lớp nhưng hoàn toàn trừu tượng, nghĩa là nó không chứa mã triển khai. Nó định nghĩa một tập hợp các phương thức mà một lớp triển khai giao diện phải triển khai.

Một interface là một lớp trừu tượng hoàn toàn (completely abstract class).

- **Các phương thức trong interface**

Các phương thức trong một interface đều là các phương thức trừu tượng (những phương thức mà không có thân hàm).

Lưu ý: Trong lớp abstract thì một phương thức trừu tượng thì cần phải được khai báo với từ khóa abstract. Còn trong interface thì phương thức không có thân phương thức là phương thức trừu tượng.

- **Các biến trong interface**

Các biến trong interface mặc định được khai báo là public, static, final.

Nếu bất kỳ biến nào được khai báo không có public, static và final thì trình biên dịch cũng sẽ tự động xác định biến đó được khai báo với public, static, final.

Biến trong interface chỉ cho phép access modifier là public.

Mọi biến trong interface phải được khởi tạo trong interface.

Những lớp con kế thừa interface không thể thay đổi giá trị của biến trong interface nhưng có thể sử dụng nó để tính toán.

- **Khai báo một interface**

Sử dụng từ khóa interface để tạo một interface trong Java.

```
interface Language {
   public static final double PI = 3.14;
   public void getType();
   public void getVersion();
```

}

Trong ví dụ trên, Language là một interface, biến double PI là một biến hằng và khai báo với public, static, final, các phương thức trừu tượng là `getType()` và `getVersion()`.

CHƯƠNG 4: THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

4.1. Mở đầu

Giao diện người dùng (Graphical User Interface – GUI)

Một giao diện người dùng (gọi tắt là GUI) là một cửa sổ (window) chứa các điều khiển (controls) như button, text box, combo box, v.v. Người dùng tương tác với giao diện bằng cách dùng chuột, con trỏ hay bàn phím.

Java cho phép chúng ta tạo giao diện người dùng bằng cách sử dụng một trong hai gói là AWT hay Swing.

AWT (Asbtract Window Toolkit) là tập con của Swing nên Swing được dùng chủ yếu nhưng trong chương trình Java phải *import* cả hai gói Swing và AWT:

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

Các điều khiển trong giao diện đồ họa dùng gói Swing luôn bắt đầu bằng chữ "J"

Ví dụ *JLabel*, *JButton*,v.v. Muốn tạo giao diện người dùng, một lớp phải kế thừa lớp JFrame như sau:

```
import java.awt.*;
import javax.swing.*;
 public class MainClass extends JFrame {
 public static void main(String[] args)  {
     // TODO Auto-generated method stub
   }
 }
```

4.2. Giới thiệu Java Swing

Là một bộ công cụ tiện ích GUI cho Java. Nó là một phần của các lớp Java Foundation Classes (JFC) của Oracle - một API để cung cấp giao diện người dùng đồ họa cho các chương trình Java. Được xây dựng dựa trên Abstract Windowing Toolkit (AWT) Application Interface (API) và được viết bằng Java. Không giống như AWT, Java Swing cung cấp các thành phần không phụ thuộc vào nền tảng và nhẹ hơn. Java Swing hoặc Swing được phát triển dựa trên các API trước đó được gọi là "Bộ công cụ trừu tượng Windows (AWT)". Swing cung cấp các thành phần GUI phong phú và phức tạp hơn AWT.

Gói javax.swing cung cấp các lớp cho Java Swing API như JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser… Gói này chứa tập hợp các

lớp interface mở rộng và cải tiến các thành phần của gói java.awt cho phép tạo giao diện đẹp hơn.

Các thành phần của gói javax.swing đều bắt đầu bằng ký tự J như: JButton, JFrame, JRadioButton, JCheckbox, JMenu, JColorChooser...

Toàn bộ thư viện Swing có tổng cộng 18 package sau:

- javax.accessibility
- javax.swing
- javax.swing.border
- javax.swing.colorchooser
- javax.swing.event
- javax.swing.filechooser
- javax.swing.plaf
- javax.swing.plaf.basic
- javax.swing.plaf.metal
- javax.swing.plaf.multi
- javax.swing.plaf.synth
- javax.swing.table
- javax.swing.text
- javax.swing.text.html
- javax.swing.text.html.parser
- javax.swing.text.rtf
- javax.swing.tree
- javax.swing.undo

4.3. Jframe

JFrame là một lớp trong thư viện Java Swing được sử dụng để tạo và quản lý cửa sổ giao diện người dùng trong ứng dụng Java. Nó là một thành phần chính của mô hình MVC (Model-View-Controller) trong Java Swing.

JFrame cung cấp một cửa sổ đồ họa đơn giản và linh hoạt, cho phép bạn tạo giao diện người dùng đa dạng và tương tác với người dùng. Bằng cách sử dụng JFrame, bạn có thể tạo các ứng dụng desktop đa nền tảng, từ ứng dụng đơn giản cho đến các ứng dụng phức tạp.

Một số tính năng quan trọng của JFrame bao gồm:

1. Quản lý cửa sổ: JFrame cho phép bạn quản lý cửa sổ giao diện người dùng, bao gồm kích thước, vị trí, tiêu đề và các thuộc tính khác của cửa sổ.
2. Bố cục: Bằng cách sử dụng JFrame, bạn có thể xây dựng bố cục giao diện người dùng bằng cách thêm các thành phần như các nút, hộp văn bản, bảng và các thành phần khác vào cửa sổ.
3. Xử lý sự kiện: JFrame cho phép bạn xử lý sự kiện người dùng như nhấp chuột, nhập liệu từ bàn phím và các sự kiện khác. Bằng cách sử dụng các phương thức và lớp lắng nghe sự kiện, bạn có thể phản ứng và xử lý các tương tác của người dùng.
4. Tùy chỉnh giao diện: JFrame cho phép bạn tùy chỉnh giao diện bằng cách thay đổi các thuộc tính như màu nền, hình nền, phông chữ và các yếu tố khác của các thành phần trong cửa sổ.

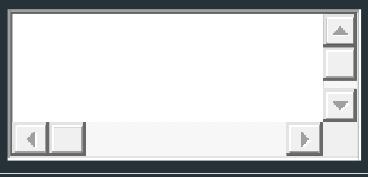
Với những tính năng và linh hoạt của nó, JFrame là một công cụ mạnh mẽ để xây dựng giao diện người dùng đa dạng trong ứng dụng Java Swing.

- **Cách sử dụng JFrame trong ứng dụng**

Để sử dụng JFrame trong ứng dụng của bạn, bạn có thể tuân theo các bước sau:

1. Import các package cần thiết:

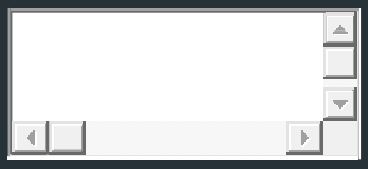
Java



```
import javax.swing.JFrame;
import javax.swing.JLabel;
```

2. Tạo một lớp chứa phương thức main():

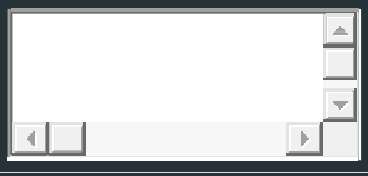
Java



```
public class MyApp {
    public static void main(String[] args) {
        // Code của bạn sẽ được viết trong phần này
    }
}
```

3. Tạo một đối tượng JFrame:

```
Java
```

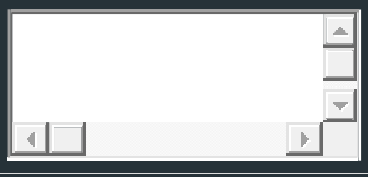


```
JFrame frame = new JFrame("Ứng dụng của tôi");
```

Trong đó, "Ứng dụng của tôi" là tiêu đề của cửa sổ JFrame.

4. Thêm các thành phần vào JFrame (ví dụ: JLabel):

```
Java
```

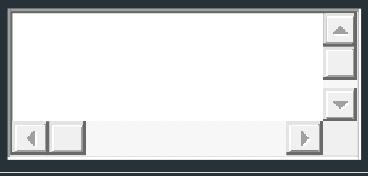


```
JLabel label = new JLabel("Chào mừng đến với ứng dụng của tôi");
frame.getContentPane().add(label);
```

Có thể thêm các thành phần khác như JButton, JTextField, JTable, v.v. vào JFrame theo nhu cầu

5. Thiết lập thuộc tính của JFrame:

```
Java
```

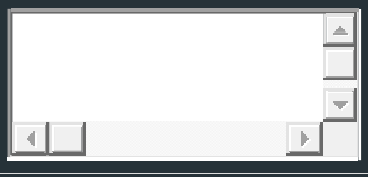


```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Đóng ứng dụng khi đóng cửa sổ
frame.setSize(400, 300); // Đặt kích thước cửa sổ
frame.setVisible(true); // Hiển thị cửa sổ
```

Tùy chỉnh các thuộc tính khác như vị trí, màu nền, v.v.

6. Chạy ứng dụng:

```
Java
```



```
MyApp.main(null);
```

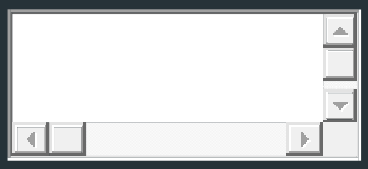
- **Xử lý sự kiện trên JFrame**

Để xử lý sự kiện trên JFrame, có thể sử dụng các lớp lắng nghe sự kiện (event listener) và gắn kết chúng với các thành phần trong JFrame.

Ví dụ về cách xử lý sự kiện nhấn nút trên JFrame:

1. Import các package cần thiết:

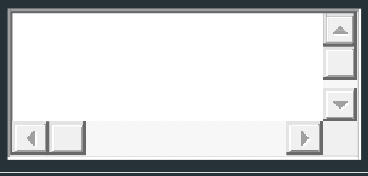
```
Java
```



```
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

2. Tạo một lớp chứa phương thức main():

```
Java
```



```
public class MyApp {
    public static void main(String[] args) {
        // Tạo JFrame
        JFrame frame = new JFrame("Ứng dụng của tôi");
        // Tạo JButton
        JButton button = new JButton("Click me");
        // Gắn lắng nghe sự kiện với JButton
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Xử lý sự kiện khi nút được nhấn
                System.out.println("Nút đã được nhấn");
            }
        });
        // Thêm JButton vào JFrame
```

```
        frame.getContentPane().add(button);
        // Thiết lập thuộc tính của JFrame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

Trong ví dụ trên, chúng ta đã tạo một JButton và gắn lắng nghe sự kiện ActionListener cho nó. Khi nút được nhấn, phương thức actionPerformed() sẽ được gọi và chúng ta có thể xử lý các hành động mong muốn trong đó.

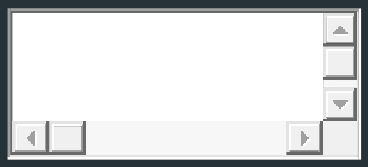
- **Tùy chỉnh giao diện JFrame**

Để tùy chỉnh giao diện JFrame trong ứng dụng Java Swing, có thể sử dụng các phương thức và thuộc tính của lớp JFrame.

Ví dụ về cách tùy chỉnh giao diện JFrame:

1. Thiết lập kích thước cửa sổ:

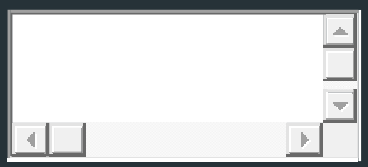
```
Java
```



```
frame.setSize(500, 400); // Đặt kích thước cửa sổ
frame.setResizable(false); // Không cho phép thay đổi kích thước cửa sổ
```

2. Đặt vị trí cửa sổ:

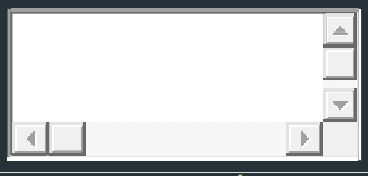
```
Elm
```



```
frame.setLocationRelativeTo(null); // Hiển thị cửa sổ ở giữa màn hình
frame.setLocation(200, 100); // Đặt vị trí cửa sổ tại tọa độ (x, y)
```

3. Đặt tiêu đề của cửa sổ:

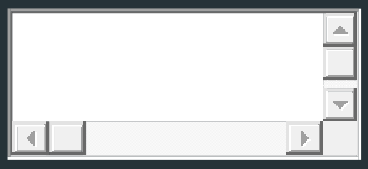
```
Java
```



```
frame.setTitle("Ứng dụng của tôi"); // Đặt tiêu đề cửa sổ
```

4. Thiết lập màu nền của cửa sổ:

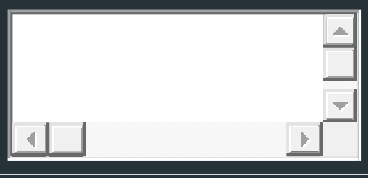
```
Java
```



```
frame.getContentPane().setBackground(Color.WHITE); // Đặt màu nền của cửa sổ
```

5. Đóng ứng dụng khi đóng cửa sổ:

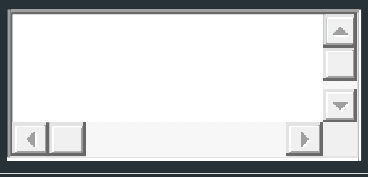
```
Java
```



```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

6. Thêm hình ảnh biểu tượng cho cửa sổ:

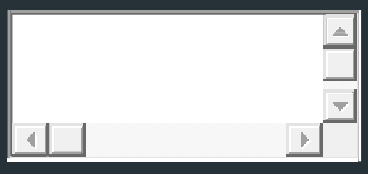
```
Java
```



```
ImageIcon icon = new ImageIcon("path/to/icon.png");
frame.setIconImage(icon.getImage());
```

7. Thay đổi kiểu giao diện mặc định:

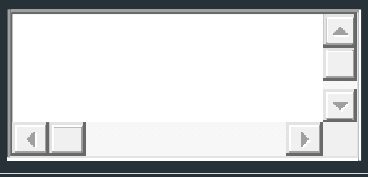
```
Java
```



```
try {
UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
} catch (Exception ex) {
  ex.printStackTrace();
}
```

8. Hiển thị cửa sổ:

```
Java
```



```
frame.setVisible(true);
```

Java 78. Tìm hiểu về cửa sổ chương trình JFrame | Lập trình Java - YouTube

4.4. Jdialog

JDialog là một thành phần trong Java Swing cho phép tạo và quản lý các cửa sổ dialog (hộp thoại) trong ứng dụng Java. JDialog được sử dụng để hiển thị thông tin chi tiết, yêu cầu người dùng nhập liệu hoặc xác nhận các hành động trong giao diện người dùng.

JDialog cung cấp các tính năng linh hoạt như tiêu đề, nút đóng, kích thước, vị trí, và quản lý sự kiện. Có thể tạo JDialog đơn giản chỉ với một tiêu đề và nội dung, hoặc tùy chỉnh giao diện với các thành phần như nút, trường nhập liệu, danh sách, và hình ảnh.

Với JDialog, ta có thể xác định cách hiển thị dialog trên JFrame hoặc một JDialog cha, và xử lý các sự kiện như người dùng nhấn nút, nhập liệu, hoặc đóng dialog.
JDialog có nhiều ứng dụng trong giao diện người dùng, bao gồm hiển thị thông báo cho người dùng, xác nhận và thông báo kết quả, cho phép người dùng chọn và nhập dữ liệu, và hiển thị các cấu hình và tùy chọn.

Sử dụng JDialog trong Java Swing, ta có thể tạo các cửa sổ dialog linh hoạt và tương tác với người dùng một cách dễ dàng và hiệu quả.
public class JDialog extends Dialog implements WindowConstants, Accessible, RootPaneContainer

Các Constructor thường được sử dụng:

Constructor	Description
JDialog()	Nó được sử dụng để tạo một hộp thoại không có mô hình không có tiêu đề và không có chủ sở hữu Khung được chỉ định.
JDialog(Frame owner)	Nó được sử dụng để tạo một hộp thoại không có mô hình với Frame được chỉ định làm chủ sở hữu của nó và một tiêu đề trống.
JDialog(Frame owner, String title, boolean modal)	Nó được sử dụng để tạo một hộp thoại với tiêu đề được chỉ định, Khung chủ sở hữu và phương thức.

Bài 30 - Java Swing - JDialog - YouTube

4.5. Jtextfield

Lớp JTextField trong Java Swing là một thành phần cho phép sửa đổi một dòng text đơn.

- **Cú pháp khai báo của lớp javax.swing.JTextField:**

Khai báo:

public class JTextField extends JTextComponent
implements SwingConstants

Lớp này kế thừa các phương thức từ các lớp sau:

- javax.swing.text.JTextComponent
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Lớp JTextField có trường static String notifyActio. Trường này là tên của action để gửi thông báo rằng các nội dung của trường này đã được chấp nhận.

Các constructor của lớp JTextField trong Java Swing

JTextField(): Xây dựng một TextField mới.

STT	Phương thức & Mô tả
1	void setActionCommand(String command) Thiết lập chuỗi lệnh được sử dụng cho action event
2	void setColumns(int columns) Thiết lập số cột trong TextField này, và sau đó làm mất hiệu lựa layout đó
3	void setDocument(Document doc) Liên kết editor với một tài liệu text
4	void setFont(Font f) Thiết lập font hiện tại
5	void setHorizontalAlignment(int alignment) Thiết lập căn chỉnh ngang cho text
6	void setScrollOffset(int scrollOffset) Thiết lập scroll offset, giá trị pixel
7	protected void actionPropertyChanged(Action action, String propertyName) Cập nhật trạng thái của textfield trong phản hồi các thay đổi của thuộc tính trong action liên kết với
8	void addActionListener(ActionListener l) Thêm action listener đã cho để nhận các action event từ textfield này
9	protected void configurePropertiesFromAction(Action a) Thiết lập các thuộc tính của textfield này để kết nối chúng trong Action đã cho
10	protected PropertyChangeListener createActionPropertyChangeListener(Action a) Tạo và trả về PropertyChangeListener mà chịu trách nhiệm nghe các thay đổi từ Action đã cho và cập nhật các thuộc tính thích hợp
11	protected Document createDefaultModel() Tạo trình triển khai mặc định của model để được sử dụng tại sự xây dựng nếu không được cung cấp tường minh
12	Action[] getActions() Gọi danh sách lệnh cho trình soạn thảo Editor
13	void postActionEvent()

	Xử lý action event xảy ra trên textfield này bằng cách gửi chúng tới bất cứ đối tượng ActionListener đã được đăng ký nào
14	void removeActionListener(ActionListener l) Xóa action listener đã cho để nó không bao giờ nhận action event từ textfield này nữa
15	void scrollRectToVisible(Rectangle r) Cuốn trường này sang trái hoặc phải
16	void setAction(Action a) Thiết lập Action cho ActionEvent source

JTextField(Document doc, String text, int columns): Xây dựng một JTextField mới mà sử dụng mô hình lưu trữ text đã cho và số cột đã cho.

JTextField(int columns): Xây dựng một TextField mới và trống với số cột đã cho.

JTextField(String text): Xây dựng một TextField mới được khởi tạo với text đã cho.

JTextField(String text, int columns): Xây dựng một TextField mới được khởi tạo với text và các cột đã cho.

- **Các phương thức được sử dụng phổ biến của lớp JTextField trong Java Swing**

Java 84. Cách sử dụng JTextField để hiển thị và nhập dữ liệu văn bản - YouTube

4.6. Jlabel

Lớp JLabel trong Java Swing có thể hiển thị hoặc text, hoặc hình ảnh hoặc cả hai. Các nội dung của Label được gán bởi thiết lập căn chỉnh ngang và dọc trong khu vực hiển thị của nó. Theo mặc định, các label được căn chỉnh theo chiều dọc trong khu vực hiển thị. Theo mặc định, text-only label là căn chỉnh theo cạnh, image-only label là căn chỉnh theo chiều ngang.

- **Cú pháp của lớp javax.swing.JLabel**

public class JLabel extends JComponent
implements SwingConstants, Accessible

Lớp javax.swing.JLabel có một trường là protected Component labelFor.

- **Constructor của lớp JLabel trong Java Swing**

1. JLabel(): Tạo một instance của JLabel, không có hình ảnh, và với một chuỗi trống cho title.

2. JLabel(Icon image): Tạo một instance của JLabel với hình ảnh đã cho.

3. JLabel(Icon image, int horizontalAlignment): Tạo một instance của JLabel với hình ảnh và căn chỉnh ngang đã cho.

4. JLabel(String text): Tạo một instance của JLabel với text đã cho.

5. JLabel(String text, Icon icon, int horizontalAlignment): Tạo một instance của JLabel với text, hình ảnh, và căn chỉnh ngang đã cho.

7. JLabel(String text, int horizontalAlignment): Tạo một instance của JLabel với text và căn chỉnh ngang đã cho.

- **Một số phương thức thường được sử dụng của lớp JLabel:**

STT	Phương thức & Mô tả
1	void setDisabledIcon(Icon disabledIcon) Thiết lập icon để được hiển thị nếu JLabel này là "disabled" (JLabel.setEnabled(false))
2	void setDisplayedMnemonic(char aChar) Xác định displayedMnemonic như là một giá trị char
3	void setDisplayedMnemonic(int key) Xác định keycode mà chỉ dẫn một mnemonic key
4	void setDisplayedMnemonicIndex(int index) Cung cấp một hint cho L&F, từ đó ký tự trong text nên được trang trí để biểu diễn mnemonic
5	void setHorizontalAlignment(int alignment) Thiết lập sự căn chỉnh nội dung của label theo trục X
6	void setHorizontalTextPosition(int textPosition) Thiết lập vị trí theo chiều ngang của phần text của label, cân xứng với hình ảnh của nó
7	void setIcon(Icon icon) Định nghĩa icon mà thành phần này sẽ hiển thị

8	void setIconTextGap(int iconTextGap) Nếu cả hai thuộc tính icon và text được thiết lập, thuộc tính này xác định khoảng trống giữa chúng
9	void setLabelFor(Component c) Thiết lập thành phần đang gán nhãn cho
10	void setText(String text) Định nghĩa trường text dòng đơn mà thành phần này sẽ hiển thị
11	void setUI(LabelUI ui) Thiết lập đối tượng L&F mà biểu diễn thành phần này
12	void setVerticalAlignment(int alignment) Thiết lập sự căn chỉnh nội dung của label theo trục Y
13	void setVerticalTextPosition(int textPosition) Thiết lập vị trí theo chiều dọc của phần text của label, cân xứng với hình ảnh của nó
14	void updateUI() Phục hồi thuộc tính UI về một giá trị từ L&F hiện tại

Java Swing 006: Cách sử dụng label-nhãn - YouTube

4.7. Jcheckbox

Hướng dẫn sử dụng check-box trong java Swing - YouTube

Lớp JCheckBox trong Java Swing là một trình triển khai của một checkbox, là một item mà có thể được lựa chọn (selected) hoặc không được lựa chọn (unselected), và hiển thị trạng thái của nó tới người dùng.

- **Khai báo cho lớp javax.swing.JCheckBox:**

public class JCheckBox extends JToggleButton
implements Accessible

- **Lớp này kế thừa các phương thức từ các lớp sau:**

- javax.swing.AbstractButton
- javax.swing.JToggleButton

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Lớp javax.swing.JCheckBox có trường static String BORDER_PAINTED_FLAT_CHANGED_PROPERTY. Trường này nhận diện một thay đổi tới thuộc tính flat.

- **Các constructor của lớp JCheckBox**

1. JCheckBox(): Tạo một unselected checkbox ban đầu không có text và icon.

2. JCheckBox(Action a): Tạo một checkbox, với các thuộc tính được lấy từ Action đã cho.

3. JCheckBox(Icon icon): Tạo một unselected checkbox với một icon.

4. JCheckBox(Icon icon, boolean selected): Tạo một checkbox với một icon và xác định rằng ban đầu nó là selected hoặc không .

5. JCheckBox(String text): Tạo một unselected checkbox ban đầu với text.

6. JCheckBox(String text, boolean selected): Tạo một checkbox với text và xác định rằng ban đầu nó là selected hoặc không.

7. JCheckBox(String text, Icon icon): Tạo một unselected checkbox ban đầu với text và icon đã cho.

8. JCheckBox(String text, Icon icon, boolean selected): Tạo một checkbox với text và icon, và xác định rằng ban đầu nó là selected hoặc không.

- **Các phương thức của lớp JCheckBox trong Java Swing**

AccessibleContext getAccessibleContext(): Lấy AccessibleContext được liên kết với JCheckBox này.

String getUIClassID(): Trả về một chuỗi xác định tên của lớp L&F mà truyền thành phần này.

boolean isBorderPaintedFlat(): Lấy giá trị của thuộc tính borderPaintedFlat.

protected String paramString(): Trả về một biểu diễn chuỗi của JCheckBox này.

void setBorderPaintedFlat(boolean b): Thiết lập thuộc tính borderPaintedFlat, mà cung cấp một hint tới L&F tới bề mặt của đường viền của checkbox.

void updateUI(): Phục hồi thuộc tính UI về một giá trị từ L&F hiện tại.

4.8. Jtextarea

Lớp JTextArea trong Java Swing được sử dụng để tạo một khu vực dành cho text. Nó là một khu vực gồm nhiều dòng và chỉ hiển thị thuần text.

Cú pháp khai báo của lớp javax.swing.JTextArea:

public class JTextArea extends JTextComponent

- **Lớp này kế thừa các phương thức từ các lớp sau:**

- javax.swing.text.JTextComponent
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

- **Các constructor của lớp JTextArea trong Java Swing**

JTextArea(): Tạo một TextArea mới.

JTextArea(String s): Tạo một TextArea mới với text đã cho.

JTextArea(int row, int column): Tạo một TextArea trống, mới với số hàng và cột đã cho.

JTextArea(String s, int row, int column): Tạo một TextArea mới với text, số hàng và cột đã cho.

- **Các phương thức được sử dụng phổ biến của lớp JTextArea trong Java Swing**

1. public void setRows(int rows): Được sử dụng để thiết lập số hàng đã cho.
2. public void setColumns(int cols): Được sử dụng để thiết lập số cột đã cho.
3. public void setFont(Font f): Được sử dụng để thiết lập font đã cho.
4. public void insert(String s, int position): Được sử dụng để chèn text vào vị trí đã cho.
5. public void append(String s): Được sử dụng để phụ thêm text đã cho vào cuối tài liệu.

Java 85. Cách sử dụng JTextArea để hiển thị và nhập dữ liệu văn bản nhiều dòng - YouTube

4.9. JradioButton

Lớp JRadioButton trong Java Swing là một trình triển khai của một radio button, một item mà có thể được lựa chọn hoặc không, và hiển thị trạng thái của nó tới người dùng. Lớp này nên được thêm vào trong ButtonGroup để chỉ lựa chọn một radio button.

- **Cú pháp khai báo cho lớp javax.swing.JRadioButton:**

public class JRadioButton extends JToggleButton
implements Accessible

- **Lớp này kế thừa các phương thức từ các lớp sau:**

- javax.swing.AbstractButton
- javax.swing.JToggleButton
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

- **Các constructor được sử dụng phổ biến của lớp JRadioButton:**

JRadioButton(): Tạo một unselected radiobutton không có text.
JRadioButton(String s): Tạo một unselected radiobutton với text đã cho.
JRadioButton(String s, boolean selected): Tạo một radiobutton với text đã cho và trạng thái là selected.

- **Các phương thức được sử dụng phổ biến của lớp JRadioButton**

AccessibleContext getAccessibleContext(): Lấy AccessibleContext được liên kết với JRadioButton này.
String getUIClassID(): Trả về tên của lớp L&F mà truyền thành phần này.
protected String paramString(): Trả về biểu diễn chuỗi của JRadioButton này.
void updateUI(): Phục hồi thuộc tính UI về một giá trị từ L&F hiện tại.
Java 95. Cách sử dụng JRadioButton và JCheckbox để lựa chọn trong Java Swing - YouTube

4.10. Thiết kế GUI cho chương trình

- **Hướng dẫn tạo Form trong Netbean**

Bước 1: Tạo Project(Ctrl + Shift + N)
Bước 2: Chọn loại Project cần tạo
Nếu làm winform hoặc Console thì chọn Java(Lớp Java 1)
Nếu làm webform JSP thì chọn Java Web

Chọn như hình và nhấn Next để chuyển tiếp

Bước 3: Đặt tên cho Project

Nhấn Finish để kết thúc

Bước 4: Thiết lập làm Project chính

Bước 5: Tạo Class tương ứng

Đặt tên class

Lưu file có tên trùng với tên của class

Có phần mở rộng .java

https://timoday.edu.vn/phan-1-forms-va-cac-control-thong-dung/

4.11. Xử lý sự kiện

- **Xử lý sự kiện cho nhiều điều khiển**

Cách thức thực thi phương thức *addPerformed* trong lớp MainClass như trên có nhược điểm là chỉ thực thi cùng một đoạn mã cho các sự kiện của nhiều điều khiển. Giả sử rằng chúng ta có hai button và khi click chuột mỗi button sẽ xuất hiện những thông điệp khác nhau, thì với cách thực thi phương thức *addPerformed* như trên sẽ không hiệu quả. Có hai giải pháp:

- Sử dụng lớp nặc danh (anonymous class)
- Tạo các lớp thực thi *addPerformed* cho các yêu cầu khác nhau

Sử dụng lớp nặc danh

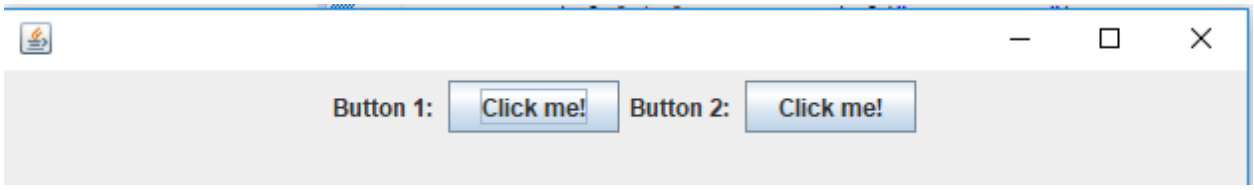
Chúng ta có thể thực thi phương thức *addPerformed* của giao diện ActionListener bằng cách dùng lớp nặc danh. Lúc này, các lớp nặc danh sẽ đóng vai trò tham số cho phương thức *addActionListener* của các điều khiển. Khi sử dụng lớp nặc danh, chúng ta không cần khai báo *implements* ActionListener tại lớp MainClass.

Thêm hai JLabel và hai JButton đến giao diện như sau:

```
public class MainClass extends JFrame {
    public static void main(String[] args)  {
      // TODO Auto-generated method stub
      MainClass mainWindow = new MainClass();
    }
    public MainClass() {
```

```
            // tạo một đối tượng JPanel
            JPanel panel = new JPanel(new FlowLayout());
             // Thêm một JLabel
             JLabel label1 = new JLabel("Button 1: ");
             panel.add(label1);
             //Thêm button 1
             JButton btn1 = new JButton("Click me!");
             panel.add(btn1);
             // Thêm một JLabel
             JLabel label2 = new JLabel("Button 2: ");
             panel.add(label2);
             //Thêm button 2
             JButton btn2 = new JButton("Click me!");
             panel.add(btn2);
             // Thiết lập nội dung hiện tại cho panel
             this.setContentPane(panel);
             // thiết lập kích thước cửa sổ
             this.setSize(640, 480);
            // thiết lập thao tác đóng cửa sổ
             this.setDefaultCloseOperation(EXIT_ON_CLOSE);
            // cho phép cửa sổ hiển thị
             this.setVisible(true);
          }
        }
```

Kết quả



Lắng nghe và xử lý sự kiện *Click* cho Button 1:

```
// Thực thi ActionListener dùng lớp nặc danh
 btn1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
```

```
        // Show a message box.
         JOptionPane.showMessageDialog(null,"I am button 1");
        }
    });
```

Lắng nghe và xử lý sự kiện *Click* cho Button 2:

```
    // Thực thi ActionListener dùng lớp nặc danh
     btn2.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {
        // Show a message box.
        JOptionPane.showMessageDialog(null,"I am button 2");
       }
     });
```

- **Tạo các lớp thực thi**

Vì Java cho phép các lớp lồng nhau nên chúng ta có thể định nghĩa các lớp thực thi phương thức *actionPerformed* cho từng yêu cầu sự kiện khác nhau của các điều khiển. Lưu ý rằng, với mỗi lớp thực thi phải khai báo thực thi giao diện ActionListener. Định nghĩa hai lớp Button1Handler và Button2Handler trong MainClass:

```
    public class MainClass extends JFrame{
        public static void main(String[] args)  {
          // TODO Auto-generated method stub
          MainClass mainWindow = new MainClass();
        }
        public MainClass() {
         ...
        }
        private class Button1Handler implements ActionListener{
          @Override
          public void actionPerformed(ActionEvent e) {
            // TODO Auto-generated method stub
            JOptionPane.showMessageDialog(null,"I am button 1");
          }
         }
```

```
    private class Button2Handler implements ActionListener{
      @Override
      public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        JOptionPane.showMessageDialog(null,"I am button 2");
       }
     }
  }
```

Để tiện sử dụng, các biến như *label1, label2, btn1*,…được khai báo như là biến thành viên của lớp MainClass:

```
public class MainClass extends JFrame{
   private JPanel panel;
   private JLabel label1;
   private JLabel label2;
   private JButton btn1;
   private JButton btn2;
   public static void main(String[] args)  {
     // TODO Auto-generated method stub
     MainClass mainWindow = new MainClass();
   }
   ...
 }
```

Sử dụng các lớp thực thi cho từng button:

```
btn1.addActionListener(new Button1Handler());
 btn2.addActionListener(new Button2Handler());
```