1. Mở đầu

* Mỗi dòng code trong Java phải ở trong một class
* Tên class có phân biệt chữ hoa chữ thường
* Khi đặt tên file jave phải đúng tên của class (ví dụ class Main => Main.java)
* Bất kì chương trình java nào cũng cần có method main() để chạy chương trình

1. **Comment**

* /\* và \*/

1. **Variable**

* String: có thể gán giá trị trực tiếp, bắt đầu đếm từ 0
* int
* byte (-128 -> 127)
* long (-: khi gán giá trị thì ở cuối giá trị phải có chữ L
* float: kết thúc giá trị bằng chữ f
* double: kết thúc giá trị bằng chữ d
* char: có thể gán mã ASCII
* boolean: *true* or *false*

=> Để khởi tạo một biến cần ghi theo cú pháp: type variable tên biến = giá trị

Ex: String name = “John”;

* Từ khóa final dùng để ngăn việc thay đổi giá trị của một biến về sau

Ex: final int age=15;

* Có thể gán giá trị cho nhiều biến cùng lúc:

Ex: x=y=z=15;

* tên biến có thể chứa dấu $ và \_
* Kiểu dữ liệu không nguyên thủy: dùng để chỉ các object được định nghĩa bởi người dùng

1. String:

* Concat: dùng phép + hoặc method *.concat()*
* Nếu cộng một số và một string thì sẽ có tác dụng như concat
* Cho các kí tự đặc biệt sau dấu backslash \

Ex: String txt = "We are the so-called \"Vikings\" from the north.";

\n New Line

\r Carriage Return

\t Tab

\b Backspace

\f Form Feed

**3. Math class**

* .sqrt : return float
* .max
* .min
* .abs
* .random [0.0 -> 1.0)

**4. Conditions:**

* giống hệt c/c++;
* kể cả toán tử ba ngôi

**5. Switch case**

* Giống c/c++
* Break, default : tương tự

6. Loops

* giống c/c++: for and while
* for each: giống c++

Ex:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars){}

7. Arrays:

* Giống c/c++

II. Methods

1. Method (aka functions)

public class Main {

static void myMethod() {

// code to be executed

}

}

static: đây là method của class main và thuộc về class đó chứ không phải object của class

1. Gọi một method:

public class Main {

static void myMethod() {

System.out.println("I just got executed!");

}

public static void main(String[] args) {

myMethod();

}

}

* gọi tên method đó + ()

1. Method overload:

* Có thể đặt tên các method trùng nhau với tham số khác nhau

static int plusMethod(int x, int y) {

return x + y;

}

static double plusMethod(double x, double y) {

return x + y;

}

* Nên overload method hơn là viết một method với tên khác nhưng thực hiện công việc gần tương tụ

1. Scope:

* Có thể sử dụng block để gói code. Các biến được khai báo trong block sẽ không thể sử dụng bên ngoài block
* Block kí hiệu bằng { }

III. OOP

1. OOP

* Giữ cho code DRY (don’t repeat yourself)

1. Class/Objects:

* Từ khóa class
* Để tạo một object cho một class: sử dụng từ khóa new, gọi tên của class

public class Main {

Main myObj = new Main();

* Tạo nhiều class: có thể tạo một object của class này trong class khác và access dữ liệu => tạo một class để giữ các biến giá trị và method, class còn lại chứa method main() để execute code
* attribute: có thể access các attribute của một class qua các object của nó
* Sử dụng từ khóa final để khóa giá trị của attribute

1. Class methods

* static vs public:
* static: truy cập không thông qua một object
* public: truy cập thông qua một object
* Nên tạo một object của class A và truy cập nó ở một class B khác

1. Constructor:

* dùng để khởi tạo giá trị ban đầu
* sẽ được execute khi một object của class đó ra đời

Ex:

public class Main {

int x; // Create a class attribute

// Create a class constructor for the Main class

public Main() {

x = 5; // Set the initial value for the class attribute x

}

public static void main(String[] args) {

Main myObj = new Main(); // Create an object of class Main (This will call the constructor)

System.out.println(myObj.x); // Print the value of x

}

}

* Constructor có thể nhận tham số

public Main(int y) {

x = y; // Set the initial value for the class attribute x

}

public static void main(String[] args) {

Main myObj = new Main(5); // Create an object of class Main (This will call the constructor)

System.out.println(myObj.x); // Print the value of x

}

}

1. Modifiers

* Access modifiers: kiểm soát phạm vi/mức độ access
* Non-access modifiers: kiểm soát các tính năng khác
* với class:

public: class có thể được truy cập bởi class khác

default: class này không thể được truy cập bởi các class khác cùng module

* với method và attribute:

static: method và attribute thuộc về một class

abstract: chỉ có thể sử dụng trong một class abstract. Method này không có body mà body sẽ được định nghĩa trong một subclass

transient: bỏ qua method và attribute khi serialize một object ( convert thông tin và dữ liệu của object đó thành các byte và có thể được convert lại để sử dụng ở một nơi khác)

synchronized: method và attribute chỉ có thể được sử dụng bởi một luồng một lúc (one thread at a time)

volatile: giá trị của attribute không được lưu local với luồng mà luôn được truy cập từ main memory

1. Tính đóng gói

* khai báo một method/attribute là private
* cần cung cấp hai method là get và set để lấy và update các biến private (cú pháp là get/set + tên class) (this)

-> Tăng tính bảo mật

-> Có thể thay đổi một bộ phận code mà không ảnh hưởng đến các bộ phận còn lại

1. Package

* Một ví dụ là các folder
* Có hai loại package
* Built-in package: được cung cấp sẵn từ Java API
* User-defined package: tự tạo
* Có thể import cả một package hoặc chỉ một class của nó

Ex:

import java.util.Scanner;

-> import một class

import java.util.\*;

-> import toàn bộ package đó (có dấu \*)

1. Tính thừa kế
2. Subclass (lớp con): được kế thừa từ một lớp khác
3. Superclass (lớp cha ): được lớp con thừa kế

* sử dụng từ khóa *extends*

Ex: class Car extends Vehicle

* dùng từ khóa final để không cho lớp con nào có thể kế thừa được lớp đang chọn

9. Tính đa hình

* Tùy kế thừa các method từ cùng một superclass, nhưng các subclass có thể thực hiện khác nhau và cho ra kết quả khác nhau

VD:

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

10. Lớp trong (inner class)

* mục đích: nhóm các class có tính chất giống nhau
* Để có thể truy cập được lớp trong, ta cần tạo ra một object của lớp ngoài rồi truy cập qua một object của lớp trong

VD:

class OuterClass {

int x = 10;

class InnerClass {

int y = 5;

}

}

public class Main {

public static void main(String[] args) {

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass();

System.out.println(myInner.y + myOuter.x);

}

}

* Inner class có thể là private hoặc protected -> object bên ngoài sẽ không thể access được
* Hoặc inner class có thể là static -> không cần object để access. Static Inner class không thể access outerclass

11. Abstract class

* Một abstract class bị giới hạn truy cập và không thể dùng để tạo các object khác , nó phải được kế thừa bởi một lớp khác trước đã

12. Interface

* Một abstract class hoàn toàn: tất cả các method trong interface đều không có body và phải được lớp khác kế thừa trước
* dùng từ khóa *implements* thay vì *extends*
* Và lớp con phải implement tất cả method của interface nếu bỏ sót sẽ lỗi