Source:
Full List về OOP và Constructor
https://www.w3schools.com/java/java_oop.asp
Java Variable in Java:
https://viblo.asia/p/primitives-object-references-MkNLrOoqVgA

Java Course Note 2

Các phần trong bài:

- Java Variable trong Java
- Constructor trong class
- 4 tính chất OOP trong Java:
    - Encapsulation
    - Inheritance
    - Polymorphism
    - Abstraction

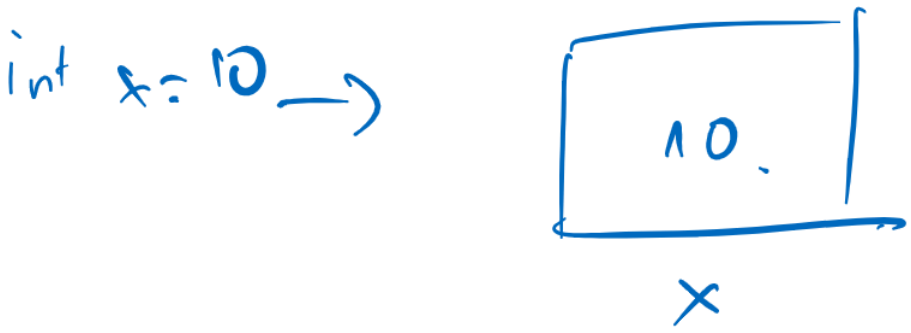
Java Variable:

Trong Java, có 2 loại biến:

- Primitive type (biến nguyên thủy)
- Reference type (biến tham chiếu)

a. Primitive type (ô memory lưu giá trị):

Được dùng để lưu các biến là primitive type (gồm các dạng có sẵn của Java: int, char, float, double)

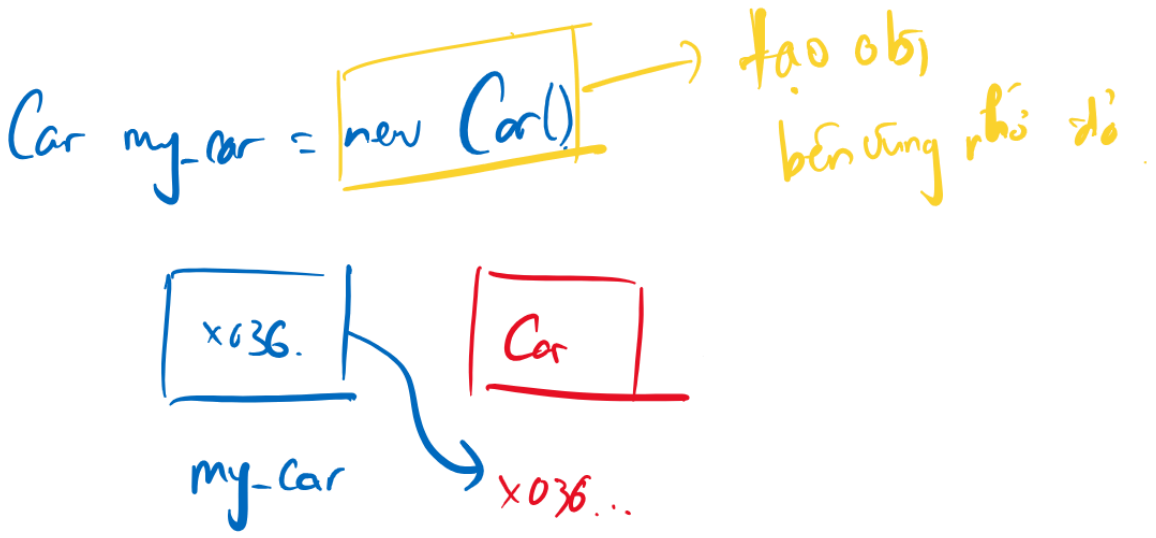


Khi khai báo biến primitive (int) thì ô memory x có mang giá trị của biến x luôn. Vì vậy khi ta in ra giá trị của biến x, nó sẽ trả về 10.

b. Reference type (ô memory lưu địa chỉ)

Được dùng để lưu các biến là object được tạo ra từ class.

Thay vì ô memory my_car lưu trực tiếp giá trị object Car, thì nó chỉ lưu giá trị là địa chỉ ô memory chứa giá trị của object Car. Từ đây, khi khai báo một object, ta phải dùng từ khóa **new** + **constructor method** -> tạo ra một object và lưu trong ô memory màu đỏ, và dùng assign để lấy địa chỉ của ô màu đỏ làm value cho my_car




```java
public class Constructor {
    Run | Debug
    public static void main(String[] args) {
        Car_2 car = new Car_2(brand:"Toyota", number_plate:"ABC123");
        System.out.println(car);
    }
}
```

```
TERMINAL  PORTS  PROBLEMS 13  OUTPUT  DEBUG CONSOLE  Run: Constructor

Try the new cross-platform PowerShell https://aka.ms/pscore6

Loading personal and system profiles took 2204ms.
(base) PS E:\TA\Java\TA>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'E:\TA\Java\TA\bin' 'Constructor'
Car_2{brand='Toyota', number_plate='ABC123'}
(base) PS E:\TA\Java\TA>  e:; cd 'e:\TA\Java\TA'; & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'E:\TA\Java\TA\bin' 'Constructor'
Car_2@36baf30c
```

Constructor trong class

Hàm khởi tạo của class, là hàm sẽ được gọi trong khi tạo object của class đó. Khi kết thúc hàm khởi tạo, thì object của class sẽ được tạo ra.

```java
class Car_2 {
    private String brand;
    private String number_plate;

    // Constructor for Car_2
    public Car_2(String brand, String number_plate) {
```

```
        this.brand = brand;
        this.number_plate = number_plate;
    }

    // Getter methods for brand and number_plate

    public String getBrand() {
        return brand;
    }

    public String getNumberPlate() {
        return number_plate;
    }
}

public class Constructor {
    public static void main(String[] args) {
        Car_2 car = new Car_2("Toyota", "ABC123");
        System.out.println(car);
    }
}
```

Ta sẽ phân tích dòng:

```
Car_2 car = new Car_2("Toyota", "ABC123");
```

1. new Car_2(“Toyota”,”ABC123”): Sẽ tạo ra một object Car_2 dựa vào hàm constructor ở trên và lưu object vào vùng nhớ (màu đỏ)
2. Tạo ra vùng nhớ (màu xanh) tên car, lưu địa chỉ của vùng nhớ màu đỏ.

- Từ khóa **this** trong constructor dùng để phân biệt khi tên attribute của class đó và parameter của hàm constructor giống nhau. (optional khi tên attribute và parameter khác nhau.)

Và Java hỗ trợ Multiple Constructor: nghĩa là trong khi tạo object, ta có thể có nhiều constructor để khởi tạo object theo nhiều kiểu khác nhau.

```
class Car_2 {
    private String brand;
    private String number_plate;

    // Constructor for Car_2
    public Car_2(String brand, String number_plate) {
        this.brand = brand;
        this.number_plate = number_plate;
    }
    public Car_2(){
        this.brand = "BMW";
        this.number_plate = "000";
```

```
    }

    // Getter methods for brand and number_plate

    public String getBrand() {
        return brand;
    }

    public String getNumberPlate() {
        return number_plate;
    }
}

public class Constructor {
    public static void main(String[] args) {
        Car_2 car = new Car_2("Toyota", "ABC123");
        Car_2 car_2 = new Car_2();
        System.out.println(car);
        System.out.println(car_2);
    }
}
```

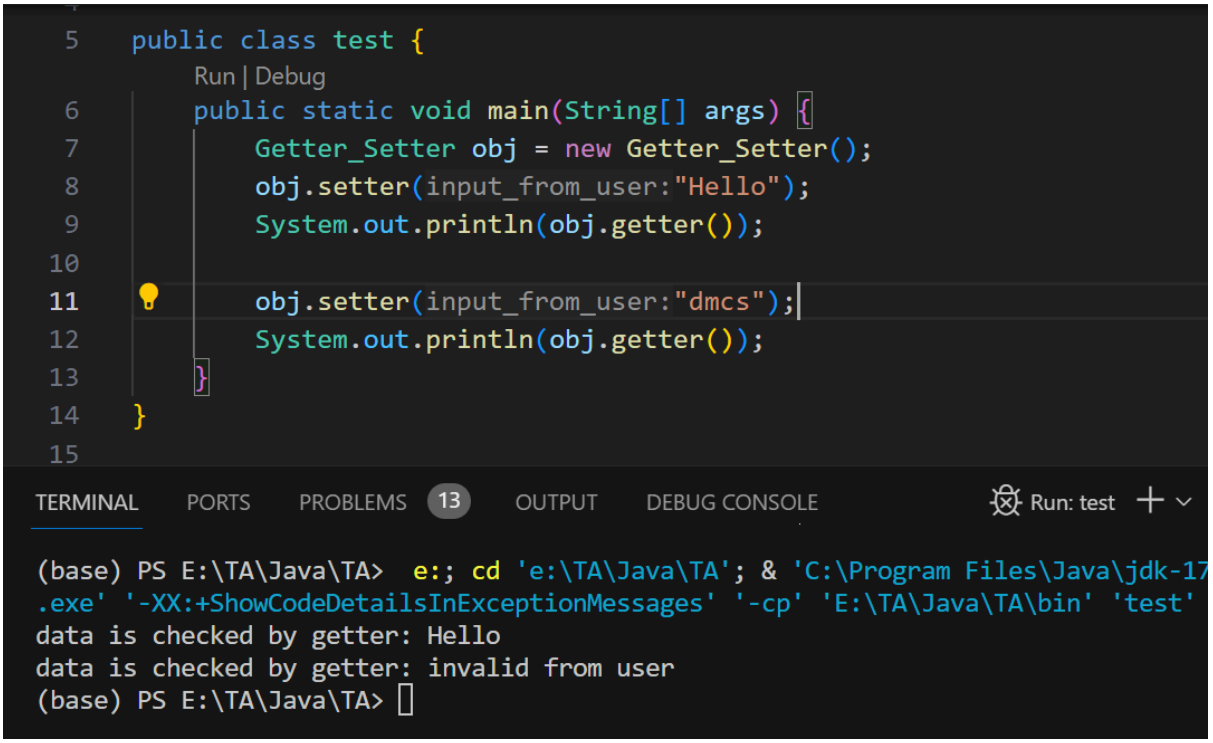
4 Tính chất OOP

Encapsulation (Tính đóng gói): dùng access modifier (public, private, protected) để kiểm soát bảo mật các thành phần trong class. Ví dụ thực tế là dùng hàm getter và setter để kiểm soát input và output của user.

```
public class Getter_Setter {
    private String data;
    public Getter_Setter(){}

    public String getter(){
        return "data is checked by getter: "+data;
    }

    public void setter(String input_from_user){
        if (input_from_user.equals("dmcs")){
            data = "invalid from user";
        }
        else{
            data = input_from_user;
        }
    }
}
```



Ta không cho phép người dùng truy cập và chỉnh sửa trực tiếp biến data trong object Getter_Setter bằng private. Và người dùng phải chỉnh sửa và truy cập qua hàm getter và setter mà ta code, từ đó ta sẽ biết được người dùng đang muốn truyền giá trị gì, và hàm của ta sẽ xử lý input từ người dùng.

Inheritance (Tính kế thừa): Về cơ bản thì tính kế thừa sẽ giống với Python, khi mà lớp con kế thừa từ lớp cha sẽ có thể có tất cả các thành phần của lớp cha (các thành phần đó phải là public hoặc protected).

```
class Vehicle {
  protected String brand = "Ford";         // Vehicle attribute
  public void honk() {                     // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar
```

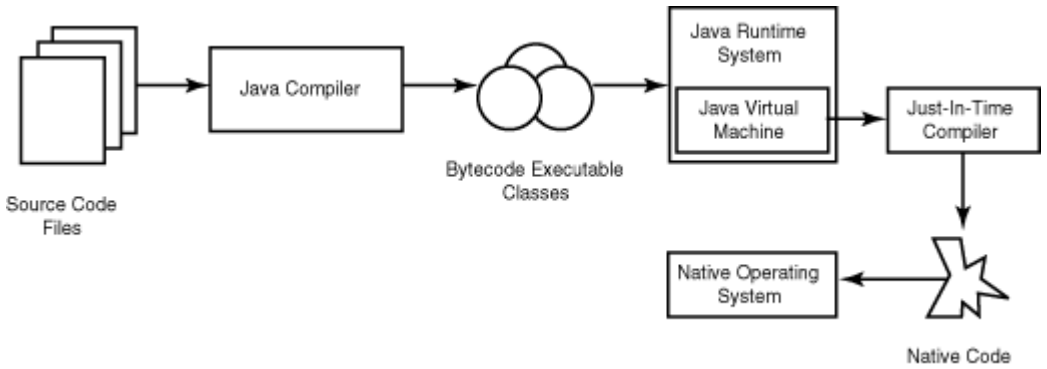
```
object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle
class) and the value of the modelName from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }

}
```

Lớp Car kế thừa từ Vehicle, nên sẽ có luôn biến brand và hàm honk sẵn.

Trả lời câu hỏi: Thứ tự chạy code trong Java đối với đoạn code trên ntn:



1. Sau khi tạo ra file Car.java thì sẽ chạy qua trình compiler (Java Compiler).
2. Trình Compiler sẽ tạo ra hai file bytecode: Car.class, Vehicle.class
3. Sau đó đưa hai file .class vào hệ thống của Java để chạy, sẽ chạy file Car.class
4. Sẽ chạy hàm main trong file Car.class trước tiên.
5. Trong hàm main thì chạy lần lượt các dòng lệnh từ trên xuống dưới của hàm main:
    a. Car myCar = new Car() sẽ liên kết với cấu trúc class của file Car.class để tạo object, và Car thì kế thừa từ Vehicle, nên cũng sẽ liên kết với file Vehicle.class để lấy cấu trúc lớp.
    b. Sau đó chạy tiếp myCar.honk()
    c. …

*Tuy nhiên, Java có thể ngăn chặn việc bạn kế thừa từ một lớp A, khi set lớp A là final

```
final class A{
  ...
}

class B extends A{
  Không thể kế thừa được.
}
```

Polymorphism (Tính đa hình): Những element (gồm attribute và method) ở lớp con có thể được thay đổi (overriding) giá trị, hoặc cách thực thi để đa dạng hơn.

```
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");
  }
}
```

Và đặc biệt một chỗ, Java có thể cho bạn gán object của class con vào tên biến được khai báo là class cha.

```
class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

Object của các class **Pig**, **Animal**, **Dog** đều sẽ là biến tham chiếu, nên các biến **myAnimal**, **myPig**, **myDog** đều không thực sự chứa object của chính nó, mà chỉ chứa địa chỉ dẫn tới object đó. nên Java cho phép các class cha, có thể truy cập tới object của class con.

Ứng dụng thực tế đến từ việc một biến được khai báo có thể sẽ là bất kì các lớp con nào.

```
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Sẽ tạo trước một biến thuộc kiểu dữ liệu Animal
```

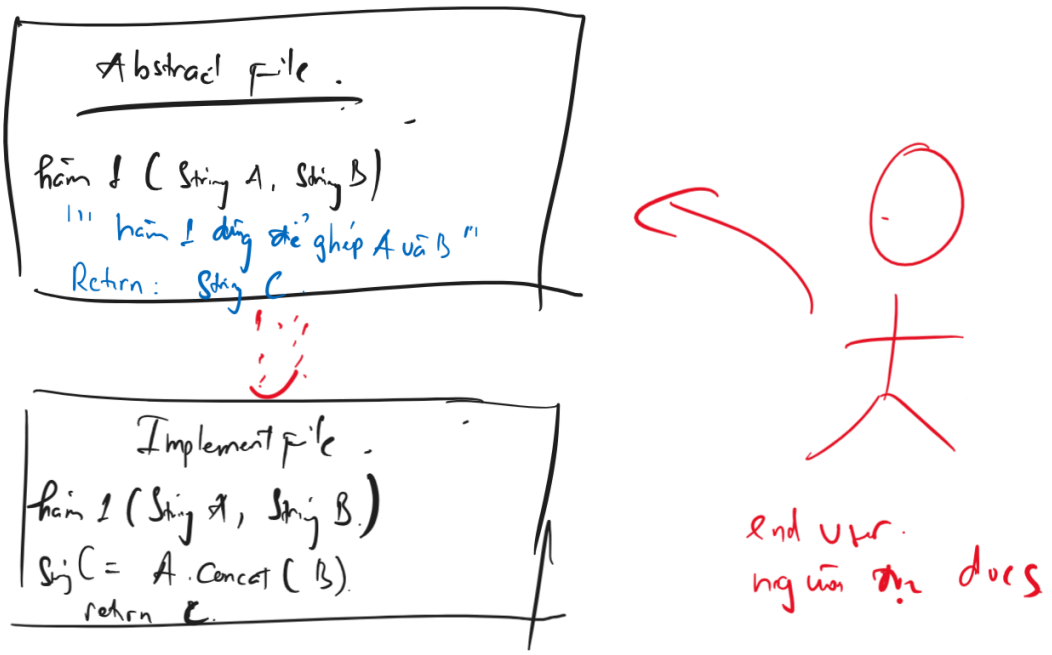
```
        Animal animal;
        System.out.println("Nhập vào object muốn tạo: 1 (Animal), 2 (Pig),
3(Dog)");
        int animal_id = scanner.nextInt();

        if (animal_id==1){
            animal = new Animal();
        }
        else if (animal_id==2){
            animal= new Pig();
        }
        else if (animal_id ==3){
            animal = new Dog();
        }
        else{
            animal = null;
        }
        animal.animalSound();
    }
```

```
Nh?p vào object mu?n t?o: 1 (Animal), 2 (Pig), 3(Dog)
3
The dog says: bow wow
```

Từ đây, Java cho phép "đa hình" (thay đổi) giá trị của biến tham chiếu trong quá trình chạy code.

Abstraction (Tính trừu tượng): Giảm bớt sự phức tạp cho người dùng end user, những người đọc thư viện. Họ chỉ cần biết input, output của một method trong class đó là cái gì bằng cách xem file abstraction và docs string mà không cần phải vào source code để xem cách nó được code như thế nào.



Trong Java, abstract file sẽ là: abstract class và interface. Cả hai loại này đều không thể tạo object của riêng nó.

1. **Abstract class:**

```
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Trong Abstract class sẽ có thể có chứa abstract method, và abstract method thì chỉ là hàm khai báo thôi, không có nội dung. Nội dung sẽ được các class con **bắt buộc** ghi vào. Abstract class có thể chứa method thường, và nội dung của method này sẽ được kế thừa cho các lớp con.

2. **Interface**

```
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a
body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Trong Interface thì chỉ có hàm abstract thôi, nên khi implements từ interface thì tất cả các class con phải ghi nội dung của tất cả các hàm có trong interface vào.

Và điểm mạnh ở interface là tính đa kế thừa:

```
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}
```

```java
class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}

class Main {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
  }
}
```

Lớp DemoClass được kế thừa từ cả hai Interface, cho nên nó sẽ có cả hai method của hai class.

***Tổng kết:**

- Sử dụng Abstract class khi có những hàm chưa có nội dung cụ thể mà sẽ được viết ra vào thời gian tới. Nhiệm vụ là khai những hàm đó trước cho dev viết body sau.
- Sử dụng Interface khi có tính trừu tượng cao, lên kế hoạch chi tiết tất cả các method cho các dev viết body. Ngoài ra hỗ trợ đa kế thừa sẽ giúp mở rộng đối tượng hơn.
  -> Với 2 ví dụ trên, thì thường các function sẽ được lên plan gồm (input và output là gì), sau đó viết abstract code (có thể là interface hoặc abstract class) và phần body sẽ được các dev viết tiếp qua các class con kế thừa từ abstract code.

- Abstraction có thể sẽ được kết hợp với inheritance và polymorphism để mở rộng đối tượng và ngăn chặn việc khởi tạo object từ class cha (là class abstraction)

Bài tập về OOP:

1. Write a Java program to create a class called "Person" with a name and age attribute. Create two instances of the "Person" class, set their attributes using the constructor, and print their name and age.

2. Write a Restaurant system project: (Live - coding)

- Name of restaurant
- Address of restaurant
- List of menu of restaurant: ArrayList<Menu>, tạo luôn một class Menu
    - Menu consists:
        - id
        - Time.
        - List of food. List<String>{“thit kho”,”ca kho to”}
        - List of cost of food. List<int> {30000, 40000}
- Order(menu_id,list of food in menu) -> return cost of dish.

  List<String> {“thit kho”, “thit kho”}