Classes and Object

Instructor: DieuNT1

Table of contents

Section 1

OOPs Concepts

What is a Class?

- A class can be considered as a **blueprint** using which you can create as many objects.
- For example, create a class **House** that has three instance variables:

```
public class House {
  String address;
  String color;
  double are;
  void openDoor() {
    // TODO
  }
  void closeDoor() {
    // TODO
  }
}
```

```
public class HouseManagement {
  public static void main(String[] args) {
    House house1 = new House("Duytan", "Blue", 1000);
    House house2 = new House("Tonthatthuyet", "Green", 1200);
    System.out.println(house1.address + "\t" + house1.color +
      "\t" + house1.are);
    System.out.println(house2.address + "\t" + house2.color +
      "\t" + house2.are);
  }
}
```

- This is just a *blueprint*, it does not represent any House
- We have created two objects, while creating objects we provided separate properties to the objects using constructor.

What is an Object

- **Object:** is a bundle of data and its behaviour (often known as methods).
- Objects have two characteristics: They have states and behaviors.
- **Example of states and behaviors**

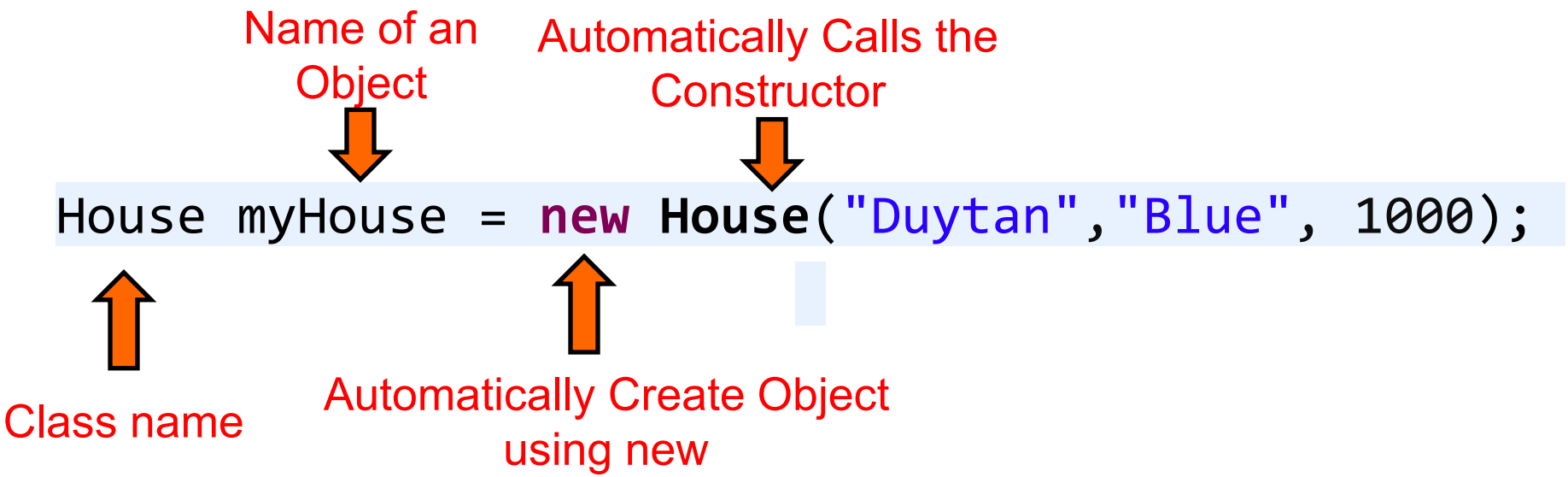
  Object: House

  State: Address, Color, Area

  Behavior: Open door, close door

Creating an Object

- Defining a class does not create an object of that class - this needs to happen explicitly[tường minh]:



- In general, an object must be created before any methods can be called.
  - ✓ the exceptions are *static* methods.

What does it mean to create an object?

```
public class SimpleClass {
    public static void main(String[] args) {
        FooPrinter foo = new FooPrinter();
        foo.print();
        foo.upper();
        foo.print();
    }
}
```

```
Output:
foo
FOO
```

- An object is a chunk of memory:
  - ✓ holds field values
  - ✓ holds an associated object type

Variable Type	Stored In
Static Variables (`static`)	**Method Area** (shared among all objects)
Instance Variables (non-static fields)	**Heap Memory** (inside each object)
Local Variables (inside methods, including `this`)	**Stack Memory** (allocated per method call)

class metadata (including the class structure, method information, static variables, and runtime constants) is stored in the Method Area (a part of the JVM memory).

- All objects of the same type share code
  - ✓ they all have same object type, but can have different field values.

Constructors

- Constructor is a block of code that initializes the newly created object.
  - ✓ Constructor has same name as the class
  - ✓ People often refer constructor as special type of method in Java. It **doesn't have a return type**
- You can create multiple constructors, each must accept different parameters.
- If you don't write any constructor, the compiler will (in effect) write one for you:

thực tế

```
FooPrinter(){}
```

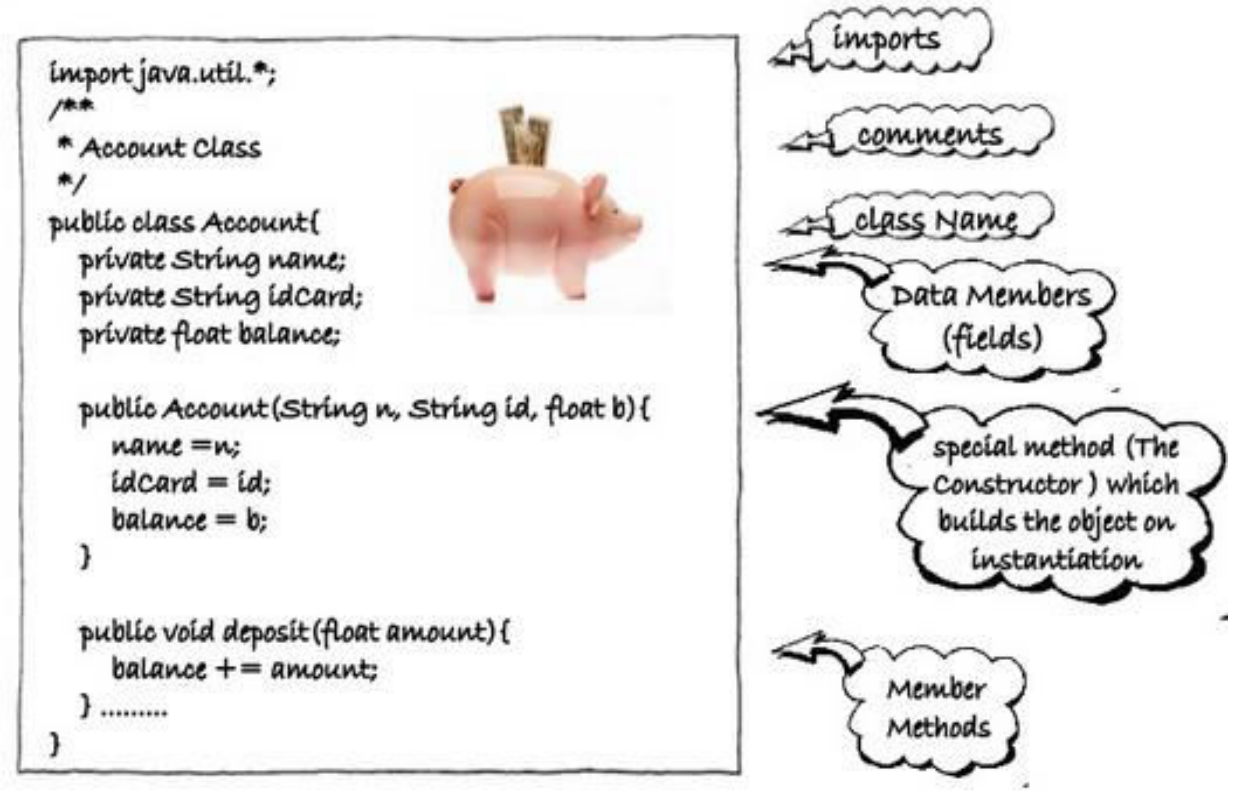
- If you include any constructors in a class, the compiler will not create a default constructor!

nếu tự viết constructor , thì Java ko tạo default constructor

Instance variable (Field)

- Instance variable in java is used by objects to store their states
- **Fields** (data members) can be any primitive or reference type
- Syntax:

  [Access modifier] <Data type> <field_name>;

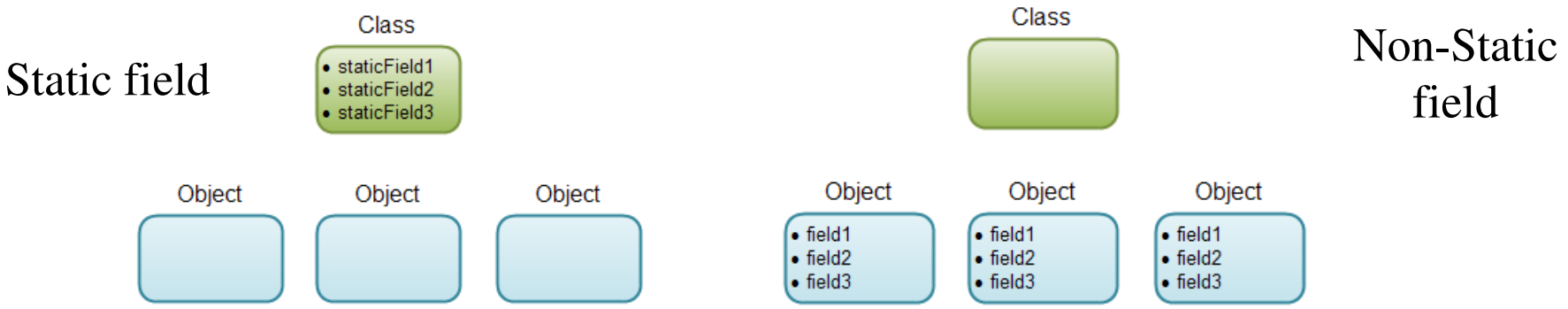


Instance method

- Instance method are methods which require an object of its class to be created before it can be called.
- Access modifiers: same idea as with fields.
  - ✓ `private/protected/public/no modifier`:
- No access modifier:
  - ✓ `abstract:` no implementation given, must be supplied by subclass.
  - ✓ `final`: the method cannot be changed by a subclass (no alternative implementation can be provided by a subclass).

Static fields

- Fields declared static are called class fields (class variables).
  - ✓ others are called *instance fields*.
- There is only one copy of a static field, no matter how many objects are created.



Static fields Examples

```
class Student {
    int rollno;
    String name;
    static String college;
    static {
        college = "ITS";
        System.out.println("Static block");
    }

    Student(int rollno, String name) {
        this.rollno = rollno;
        this.name = name;
        System.out.println("Constructor block");
    }

    void display() {
        System.out.println(rollno + " " + name + " " + college);
    }

    static void changeCollege() {
        college = "FU";
    }
}
```

```
public static void main(String args[]) {
       // Student.changeCollege();
        Student s1 = new Student(111, "Karan");
        Student s2 = new Student(222, "Aryan");
 Student.changeCollege();
        s1.display();
        s2.display();
}
```

```
111 Karan FU
222 Aryan FU
```

Static methods

- Static methods are the methods in Java that can be called without creating an object of class.
  - ✓ Instance method can access the instance methods and instance variables directly.
  - ✓ Instance method can access static variables and static methods directly.
  - ✓ Static methods can access the static variables and static methods directly.
  - ✓ Static methods can't access instance methods and instance variables directly.
- **Syntax:**

```
static return_type method_name();
```

Final Fields

- The keyword **final** means: once the value is set, it can never be changed.
  - ✓ They must be ***static*** if they belong to the ***class***.
  - ✓ ***Not be static*** if they belong to the ***instance*** of the class.
- Typically used for constants:

```
private static final int MAX_LAST_NAME_LENGTH = 255; // belongs to the type
private final String firstName; // belongs to the instance
private final String lastName; // belongs to the instance
```

- **Important Note:**
  - ✓ A **final variable** that is not initialized at the time of declaration is known as **blank final variable**.
    - We can **initialize** blank final variable **in constructor**. chỉ trong constructor mà thôi
  - ✓ A **static final variable** that is **not initialized** at the time of declaration is known as static blank final variable.
    - It can be **initialized** only in **static block**.

because the initialization must happen at a time when the class is loaded into memory, before any instances of the class are created or the variable is accessed.

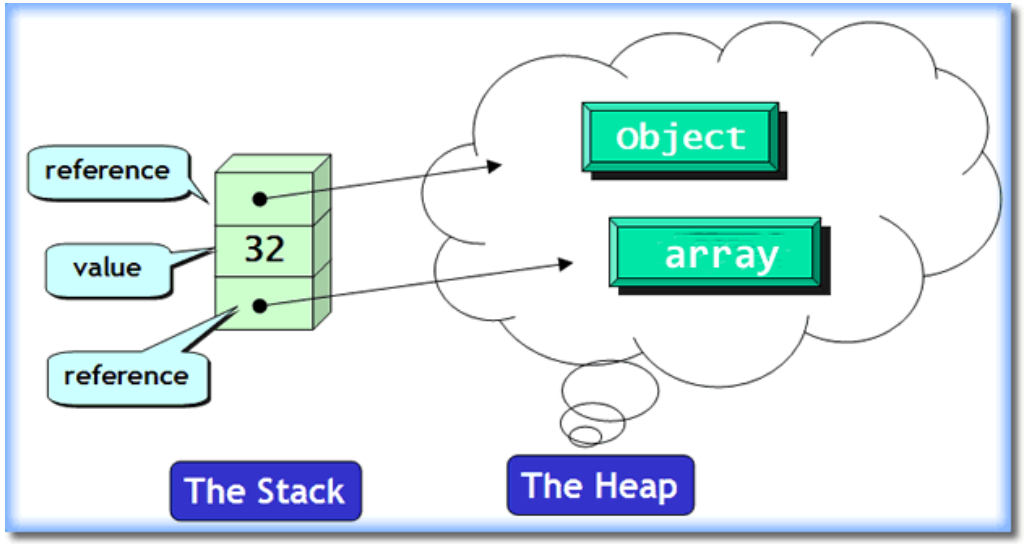
Section 2

HEAP SPACE VS STACK MEMORY

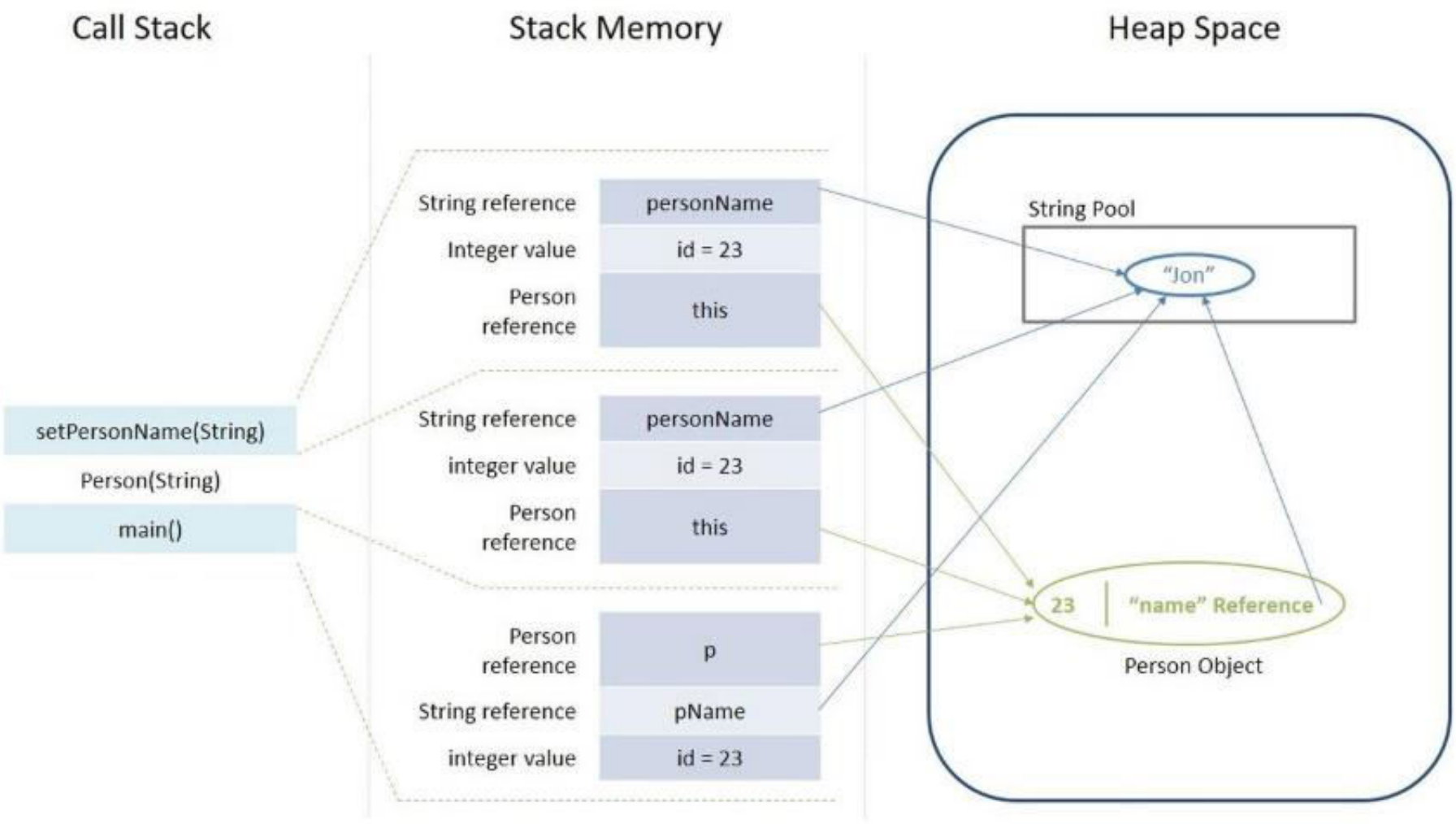
Introduction

- To run an application in an optimal way, JVM divides memory into stack and heap memory.
  - ✓ **Declare new variables and objects, call new method, declare a *String* or perform similar operations**

    ➔ JVM designates memory to these operations from either Stack Memory or Heap Space.



Heap Space vs Stack Memory



Section 3

PARAMETERS

Parameters

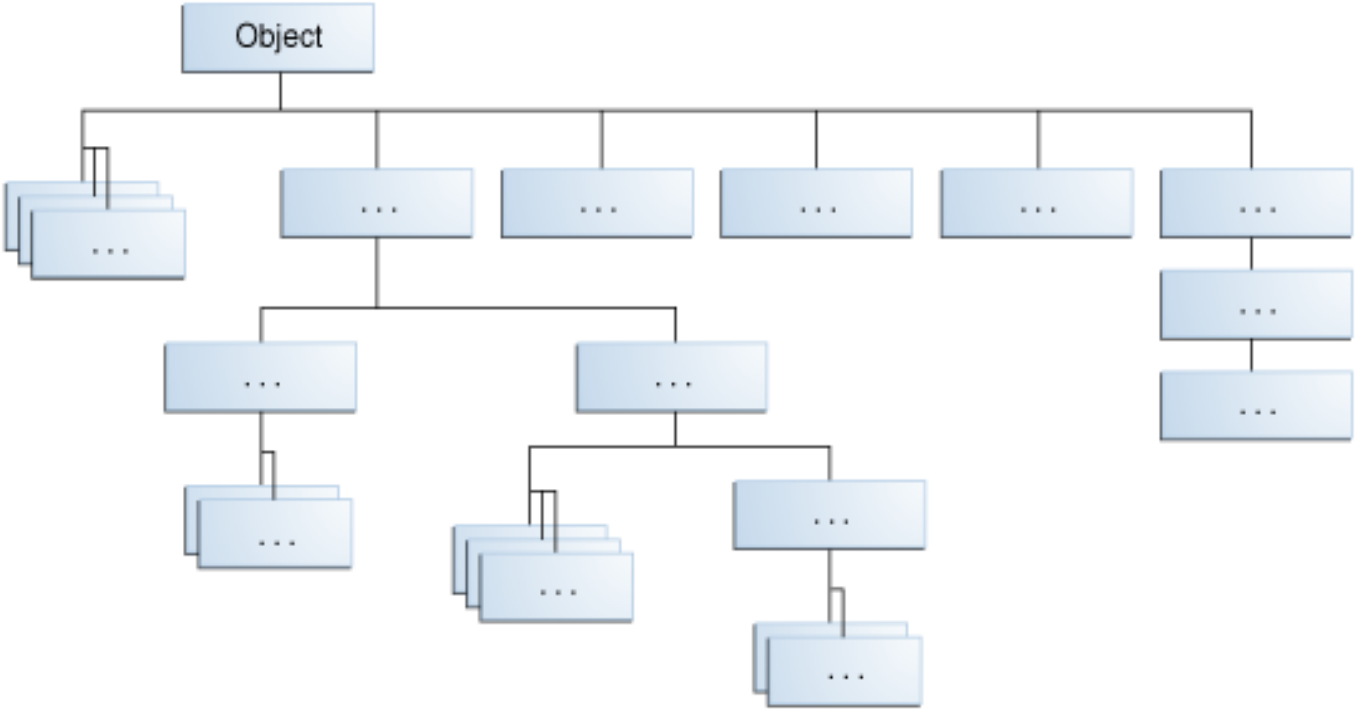
- Parameters (also called arguments) is variable that declare in the method definition.
- Parameters are always classified as "variables" not "fields".
- Two ways to pass arguments to methods
  - ✓ Pass-by-value
  - ✓ Pass-by-reference

Value and Reference Parameters

- Pass-by-value
  - ✓ Copy of argument's value is passed to called method
- Pass-by-reference
  - ✓ Caller gives called method direct access to caller's data
  - ✓ Called method can manipulate this data
  - ✓ Improved performance over pass-by-value

The class Object

- Granddaddy of all Java classes.
- All methods defined in the class Object are available in every class.
- Any object can be cast as an `Object`. nghĩa là upcasting



Summary

- OOPs Concepts
- Heap Space vs Stack Memory
- Method Parameters

Thank you