Object-Oriented Language and Theory

Bui Thi Mai Anh, anhbtm@soict.hust.edu.vn
Nguyen Thi Thu Trang, trangntt@soict.hust.edu.vn

Lab 4: Some techniques in Class Building

* Objectives:

In this lab, you will practice with:

- Method overloading
- Parameter passing
- Classifier member vs. Instance member

This lab also concentrates on the project that you did with the previous lab. You continue using Eclipse to implement "AIMS: An Internet Media Store" - A system for creating orders of CDs, DVDs and books. Other exercises cover specific Object-Oriented Programming or Java topics.

1. Working with method overloading

Method overloading allows different methods to have **same name** but different signatures where signature can differ by **number** of input parameters or **type** of input parametere or **both.**

1.1 Overloading by differing types of parameter

- Open Eclipse

- Open the JavaProject named "`AimsProject`" that you have created in the previous lab.

- Open the class `Order.java`: you will overload the method **`addDigitalVideoDisc`** you created last time.

+ The current method has one input parameter of class **`DigitalVideoDisc`**

+ You will create new method has the same name but with different type of parameter.

```
addDigitalVideoDisc(DigitalVideoDisc [] dvdList)
```

This method will add a list of DVDs to the current order.

+ You should always verify the number of items in the current order to assure the quantity below the maximum number.

+ Inform users the list of items that cannot be added to the current order because of full ordered items

+ Try to add a method **`addDigitalVideoDisc`** which allows to pass an arbitrary number of arguments for dvd. Compare to an array parameter. What do you prefer in this case?

1.2. Overloading by differing the number of parameters

- Continuing focus on the `Order` class

- Create new method named `addDigitalVideoDisc`

+ The signature of this method has two parameters as following:

```
addDigitalVideoDisc(DigitalVideoDisc dvd1,
                    DigitalVideoDisc dvd2)
```

+ You also should verify the number of items in the current order to assure the quantity below the maximum number.

+ Inform users if the order is full and print the dvd(s) that could not be added

2. Passing parameter

- Question: ***Is JAVA a Pass by Value or a Pass by Reference programming language?***

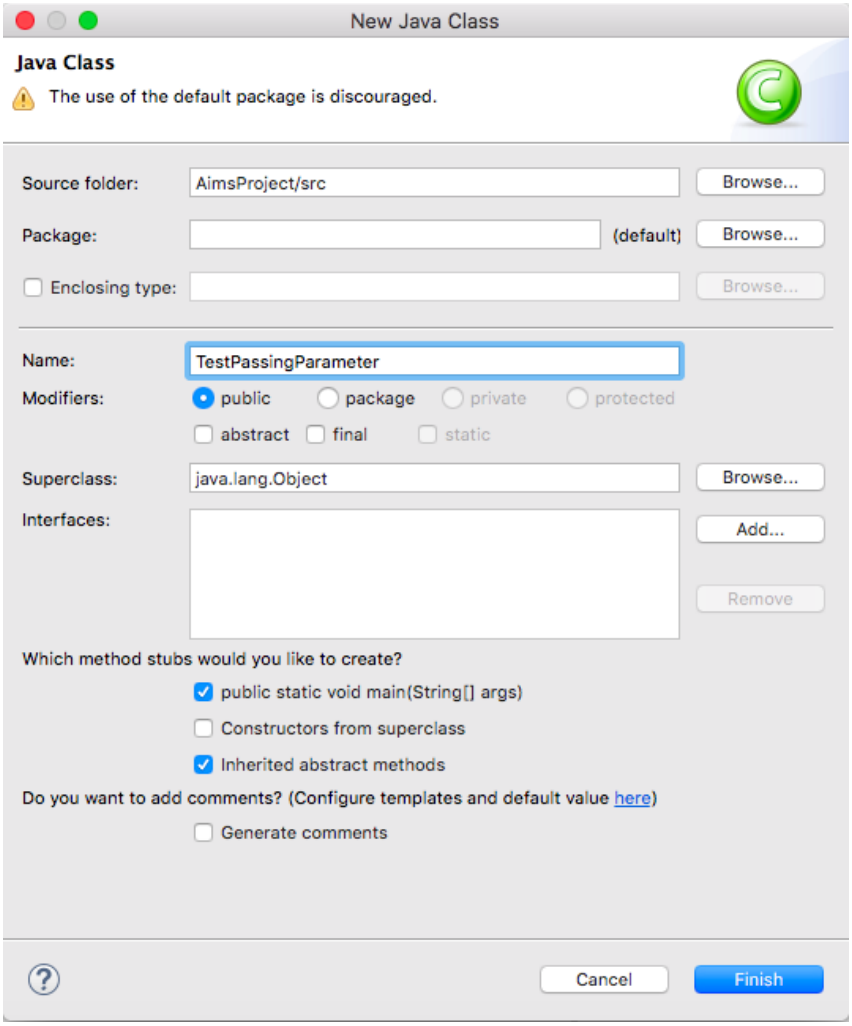
First of all, we recall what is meant by **pass by value** or **pass by reference**.

- Pass by value: The method parameter values are copied to another variable and then the copied object is passed to the method. That's why it's called pass by value
- Pass by reference: An alias or reference to the actual parameter is passed to the method. That's why it's called pass by reference.

Now, you will practice with the **`DigitalVideoDisc`** class to test how JAVA passes parameter.

Create a new class named **`TestPassingParameter`** in the current project

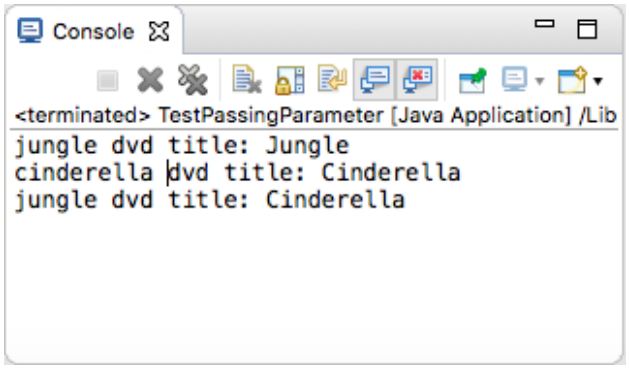
- Check the option for generating the main method in this class.



In the **`main()`** method of the class, typing the code below:

```
public class TestPassingParameter {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        DigitalVideoDisc jungleDVD = new DigitalVideoDisc("Jungle");
        DigitalVideoDisc cinderellaDVD = new DigitalVideoDisc("Cinderella");

        swap(jungleDVD, cinderellaDVD);
        System.out.println("jungle dvd title: " + jungleDVD.getTitle());
        System.out.println("cinderella dvd title: " + cinderellaDVD.getTitle());

        changeTitle(jungleDVD, cinderellaDVD.getTitle());
        System.out.println("jungle dvd title: " + jungleDVD.getTitle());
    }

    public static void swap(Object o1, Object o2) {
        Object tmp = o1;
        o1 = o2;
        o2 = tmp;
    }

    public static void changeTitle(DigitalVideoDisc dvd, String title) {
        String oldTitle = dvd.getTitle();
        dvd.setTitle(title);
        dvd = new DigitalVideoDisc(oldTitle);
    }

}
```

The result in console is below:



To test whether a programming language is passing by value or passing by reference, we usually use the **`swap`** method. This method aims to swap an object to another object.

- After the call of **`swap(jungleDVD, cinderellaDVD)`** why does the title of these two objects still remain?
- After the call of **`changeTitle(jungleDVD, cinderellaDVD.getTitle())`** why is the title of the JungleDVD changed?

After finding the answers to these above question, you will understand that JAVA is always a pass by value programming language.

3. Classifier Member and Instance Member

- Classifier/Class member:
    - Defined in a class of which a single copy exists regardless of how many instance of the class exist.
    - Objective: to have variables that are **common** to all objects
    - Any object of class can change the value of a class variable that's why you should always be carefull with the side effect of class member
    - Class variables can be manipulated without creating an instance of the class

- Instance/Object member:
    - Associated with only objects
    - Defined inside the class but outside of any method
    - Only initialized when the instance is created
    - Their values are unique to each instance of a class
    - Lives as long as the object does

Open the `Order` class:

- You should note that there are 2 instance variables
    - **`itemsOrdered`**
    - **`qtyOrdered`**

- You add a new instance variable named "**`dateOrdered`**" to store the date-time the ordered created.

- Add getter/setter methods for this instance variable

- This variable instance has a unique value to each instance of the **`Order`** class and should be initialized inside the constructor method of the **`Order`**.

- Now we suppose that, the application only allows to make a limited number of **`orders.`** That means: if the current number of orders is over this limited number, users cannot make any new order.

- Create a class attribute named "**`nbOrders`**" in the class **`Order`**

- Create also a constant for limited number of **`orders`** per user for this class

```
public static final int MAX_LIMITTED_ORDERS = 5;

private static int nbOrders = 0;
```

- Each time an instance of the **`Order`** class is created, the **`nbOrders`** should be updated. Therefore, you should update the value for this class variable inside the constructor method and check if **`nbOrders`** is below to the **`MAX_LIMITTED_ORDERS`**.

- Creating a new method to printing the list of ordered items of an order, the price of each item, the total price and the date order. Formatting the outline as below:

**********Order********** ********
Date: [date-order]
Ordered Items:
1. DVD - [Title] - [category] - [Director] - [Length]: [Price] $
2. DVD - [Title] - ...
Total cost: [total cost]
****

- In the main method of the class Aims:
    - Creating different orders
    - For each order, add different items (DVDs) and print the order to the screen
    - Write some code to test what you have done in the main method

4. Open the project of mydate in the previous lab:

- In the **`MyDate`** class:
    + Write overloading setter methods for **`setX`**, where **`x`** can be **`day`**, **`month`** and **`year`** in **`String`** (their names instead of their values in number, such as "second", "September", "twenty nineteen")
    + Write **`print()`** method in My Date to print the String version of the current date
    + Write another **`print`** method which allows users can print a date with a specified format, such as "**`dd/mm/yyyy`**" or "**`yy-mm-dd`**" or even **`String`** values for **`day`**, **`month`** or **`year`**. You can look at the **`DateFormat`** in Java for reference.
- Create a new class naming **`DateUtils`** which includes public static methods:
    + Compare two dates
    + Sorting a number of dates
- In the **`DateTest`** class, write codes to test all methods you've wrote in this exercise.