Object-Oriented Language and Theory

Lecturer: NGUYEN Thi Thu Trang, trangntt@soict.hust.edu.vn
Teaching Assistants: NGUYEN T.T. Giang, giang.ntt194750@sis.hust.edu.vn
VUONG Dinh An, an.vd180003@sis.hust.edu.vn

Lab 4: Basic OOP techniques

In this lab, you will practice with:

- Parameter passing
- Debugging with Eclipse
- Classifier member vs. Instance member
- Re-organizing your project by creating packages to manage classes in Eclipse
- Practicing memory management with String and StringBuffer and other cases

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". Other exercises cover specific Object-Oriented Programming or Java topics.

0 Assignment Submission

For this lab class, you will have to turn in your work twice, specifically:

- **Right after the class**: for this deadline, you should include any work you have done within the lab class.
- **10 PM two days after the class**: for this deadline, you should include your work of all sections of the lab, into a directory namely "**Lab04**" and push it to your **master** branch of the valid repository.

After completing all the exercises in the lab, you have to update the use case diagram and the class diagram of the AIMS project.

Each student is expected to turn in his or her own work and not give or receive unpermitted aid. Otherwise, we would apply extreme methods for measurement to prevent cheating. Please write down answers for all questions into a text file named "**answers.txt"** and submit it within your repository.

1 Import/export a project

- Open Eclipse

- You can import/export a project from/to an archive file. For example, if you want to import from a zip file, you can follow the following steps:

+ Open File -> Import.

+ Type zip to find Archive File if you have exported as a zip file before. You may choose Existing Projects into Workspace if you want to open an existing project in your computer. Ignore this step if the AimsProject is already opened in the workspace.

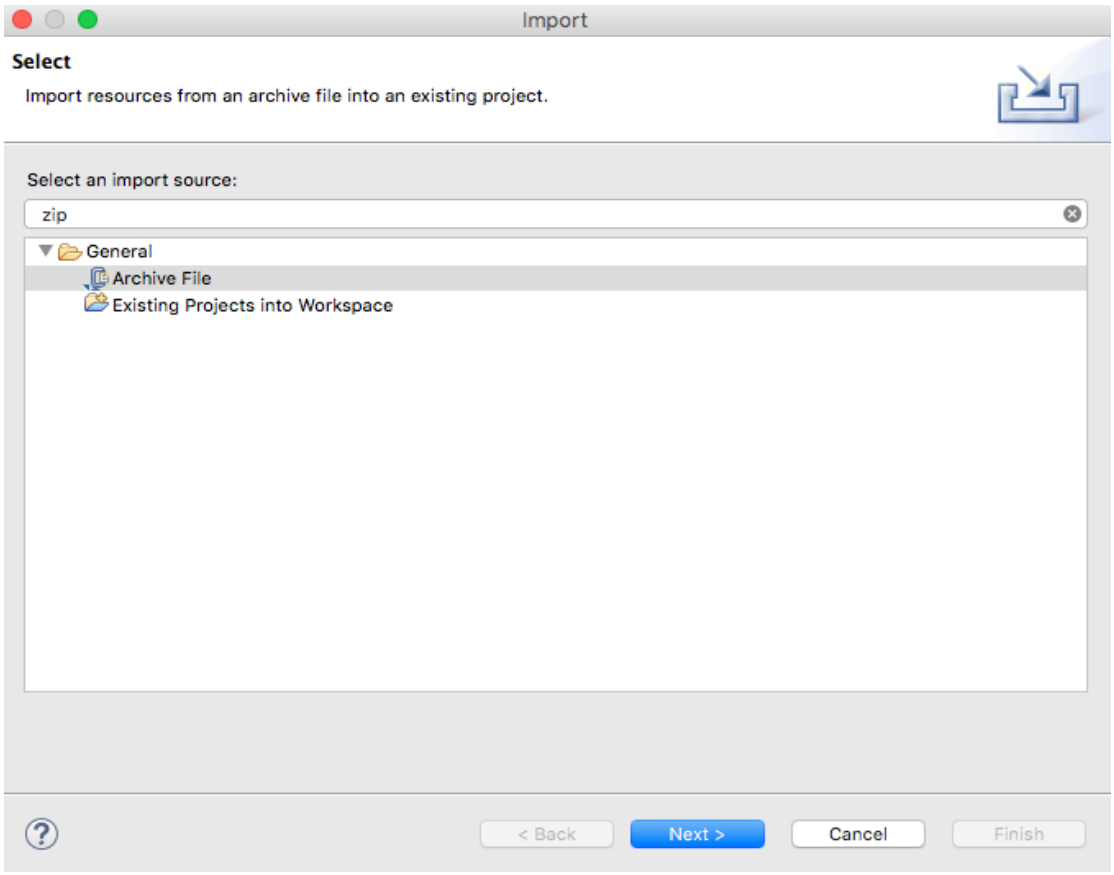

Figure 1. Import from an archive file

- Click Next and Browse to a zip file or a project to open

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 parameters. For this exercise, you will need to temporarily add a setter for the attribute title of the DigitalVideoDisc class.

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

- Check the option for generating the main method in this class like in Figure 1

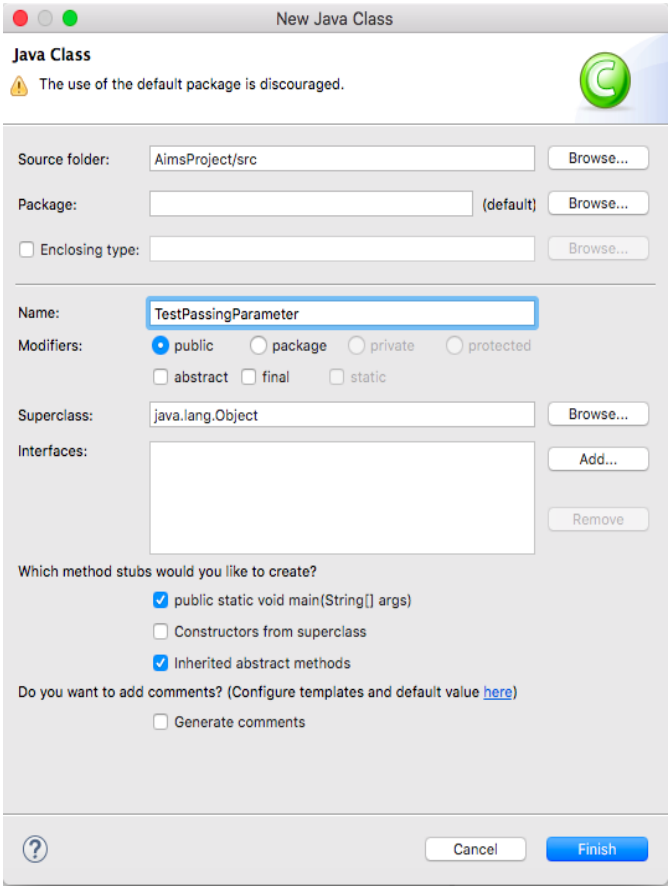

Figure 1. Create TestPassingParameter by Eclipse

In the **`main()`** method of the class, typing the code below in Figure 2:

```
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);
    }
}
```

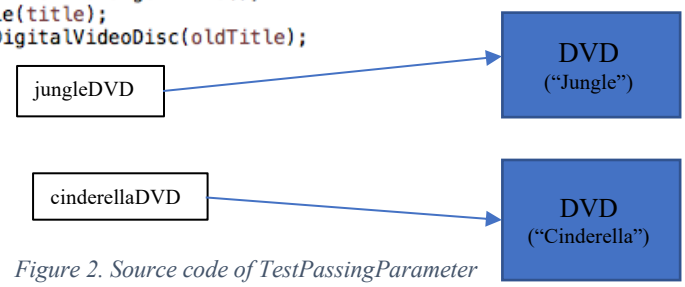

Figure 2. Source code of TestPassingParameter

The result in the console is below:

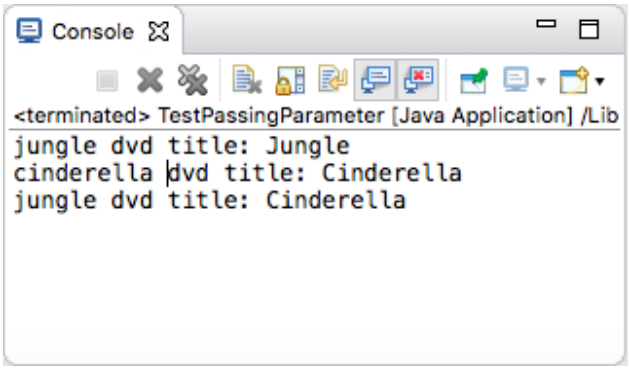

Figure 3. Results(1)

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 questions, you will understand that JAVA is always a pass by value programming language.
Please write a swap() method that can correctly swap the two objects.

3 Use debug run:

3.1 Debugging Java in Eclipse

Video: https://www.youtube.com/watch?v=9gAjIQc4bPU&t=8s

Debugging is the routine process of locating and removing bugs, errors, or abnormalities from programs. It's a must-have skill for any Java developer because it helps to find subtle bugs that are not visible during code reviews or that only happen when a specific condition occurs. The Eclipse Java IDE provides many debugging tools and views grouped in the Debug Perspective to help you as a developer debug effectively and efficiently.

Debug run allows you to run a program interactively while watching the source code and the variables during the execution. A *breakpoint* in the source code specifies where the execution of the program should stop during debugging. **Once the program is stopped you can investigate variables, change their content, etc.**

3.2 Example of debugging run for the `swap` method of `TestPassingParameter`

3.2.1 Setting, deleting & *deactivating* breakpoints:

To set a breakpoint, place the cursor on the line that needs debugging, hold down Ctrl+Shift, and press B to enable a breakpoint. A blue dot in front of the line will appear (Figure 4). Alternatively, you can right-click in the left margin of the line in the Java editor and select Toggle Breakpoint. This is equivalent to double-clicking in the left margin of the line.

```
1
2  public class TestPassingParameter {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          DigitalVideoDisc jungleDVD = new DigitalVideoDisc("Jungle");
7          DigitalVideoDisc cinderellaDVD = new DigitalVideoDisc("Cinderella");
8
9          swap(jungleDVD, cinderellaDVD);
10         System.out.println("jungle dvd title: "+jungleDVD.getTitle());
11         System.out.println("cinderella dvd title: "+cinderellaDVD.getTitle());
12
13         changeTitle(jungleDVD, cinderellaDVD.getTitle());
14         System.out.println("jungle dvd title: "+jungleDVD.getTitle());
15     }
16
17     public static void swap(Object o1, Object o2) {
18         Object tmp = o1;
19         o1 = o2;
20         o2 = tmp;
21     }
22
23     public static void changeTitle(DigitalVideoDisc dvd, String title) {
24         String oldTitle = dvd.getTitle();
25         dvd.setTitle(title);
26         dvd = new DigitalVideoDisc(oldTitle);
27     }
28
```

Figure 4. A breakpoint is set

To delete a breakpoint, toggle the breakpoint one more time. The blue dot in front of the line will disappear (Figure 5).

```
1
2  public class TestPassingParameter {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          DigitalVideoDisc jungleDVD = new DigitalVideoDisc("Jungle");
7          DigitalVideoDisc cinderellaDVD = new DigitalVideoDisc("Cinderella");
8
9          swap(jungleDVD, cinderellaDVD);
10         System.out.println("jungle dvd title: "+jungleDVD.getTitle());
11         System.out.println("cinderella dvd title: "+cinderellaDVD.getTitle());
12
13         changeTitle(jungleDVD, cinderellaDVD.getTitle());
14         System.out.println("jungle dvd title: "+jungleDVD.getTitle());
15     }
16
17     public static void swap(Object o1, Object o2) {
18         Object tmp = o1;
19         o1 = o2;
20         o2 = tmp;
21     }
22
23     public static void changeTitle(DigitalVideoDisc dvd, String title) {
24         String oldTitle = dvd.getTitle();
25         dvd.setTitle(title);
26         dvd = new DigitalVideoDisc(oldTitle);
27     }
28
```

Figure 5. The breakpoint is deleted

To deactivate the breakpoint, navigate to the Breakpoints View and uncheck the tick mark next to the breakpoint you want to deactivate (Figure 6). **The program will only stop at activated breakpoints**.

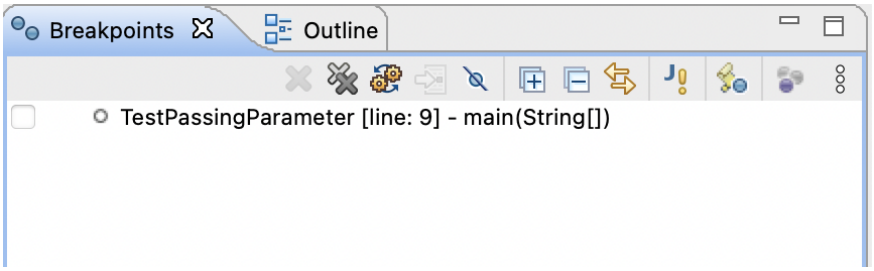

Figure 6. Deactivated breakpoint in Breakpoints View

For this example, we will need to keep this breakpoint, so make sure to set the breakpoint again after practicing with deleting/deactivating it before moving to the next section.

3.2.2 Run in Debug mode:

Select a Java file with the main method that contains the code that you need to debug from Project Explorer. In this example, we choose the **TestPassingParameter.java** file. Right-click and choose Debug As > Java Application (Figure 7).

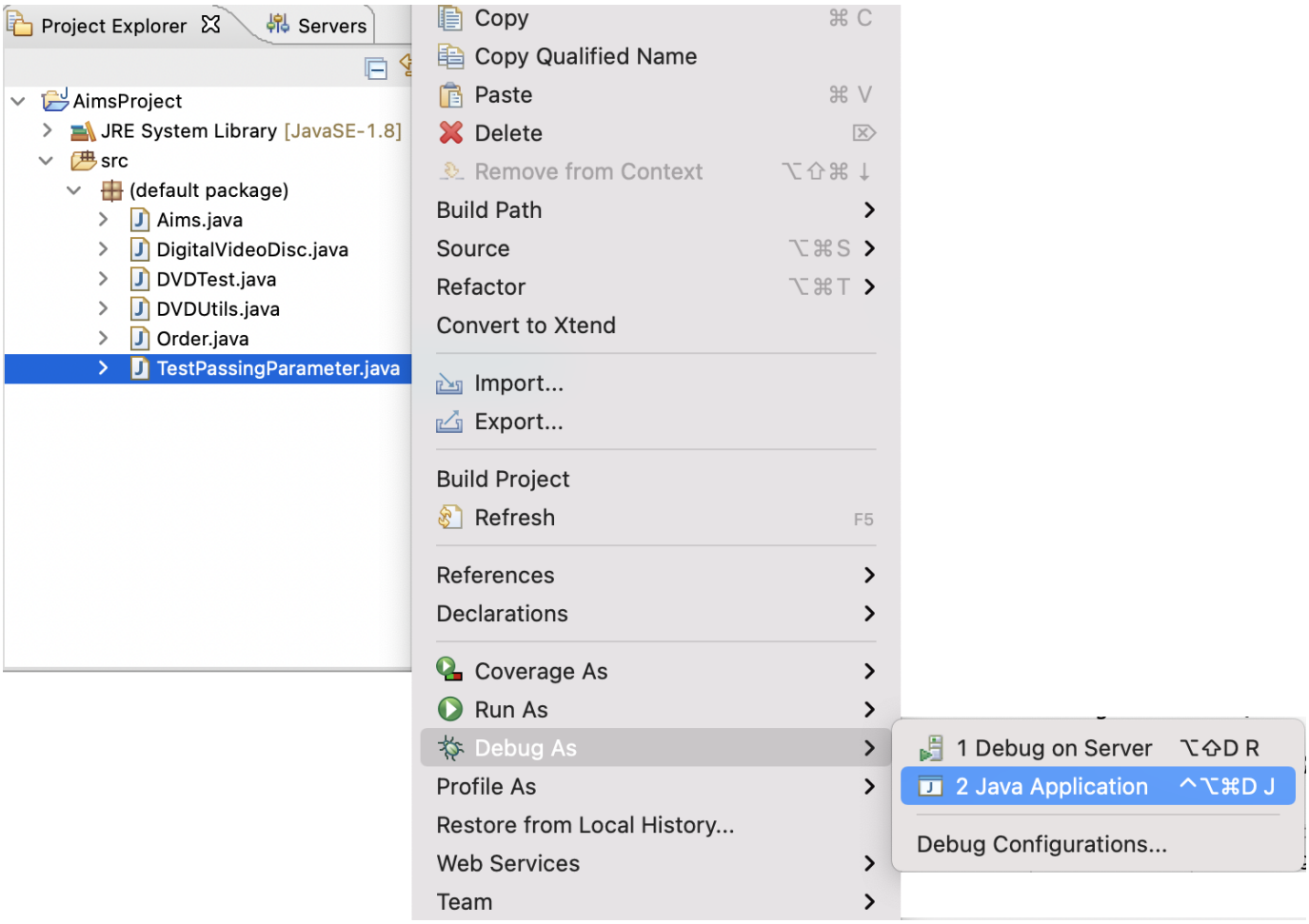

Figure 7. Run Debug from a class

Alternatively, you can select the project root node in the Project Explorer and click the debug icon in the Eclipse toolbar (Figure 8)

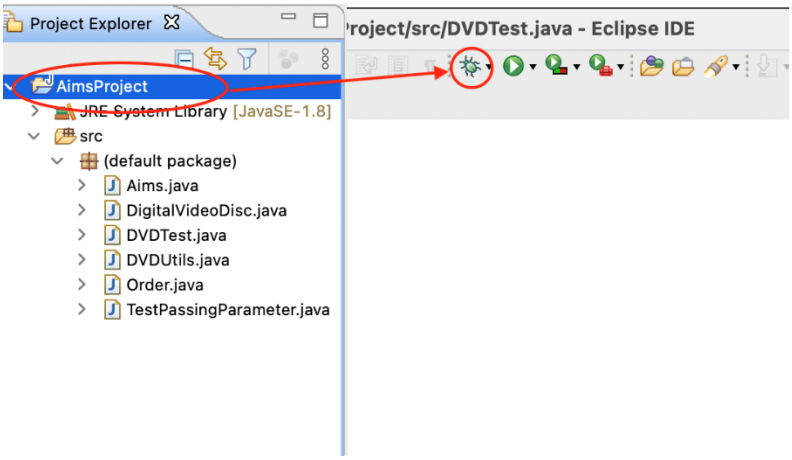

Figure 8. Run debug from a project

The application will now be started with Eclipse attached as a debugger. Confirm to open the Debug Perspective.

3.2.3 Step Into, Step Over, Step Return, Resume:

- In the Debug Perspective, you can observe the Step Into/Over/Return & Resume/Terminate buttons on the toolbar as in Figure 9.

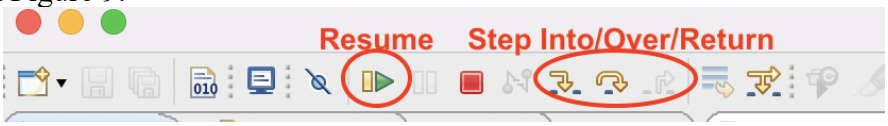

Figure 9. Stepping Commands on the Toolbar in Debug Perspective

- With debugger options, the difference between "Step into" and "Step over" is only noticeable if you run into a function call:

o "Step into" (F5) means that the debugger steps into the function

o "Step over" (F6) just moves the debugger to the next line in the same Java action

- With "Step Return" (pressing F7), you can instruct the debugger to leave the function; this is basically the opposite of "Step into."
- Clicking "Resume" (F8) instructs the debugger to continue until it reaches another breakpoint.

For this example, we need to see the execution of the **`swap`** function, so we choose Step Into. The debugger will step into the implementation of the **`swap`** function in line 18 (Figure 10).

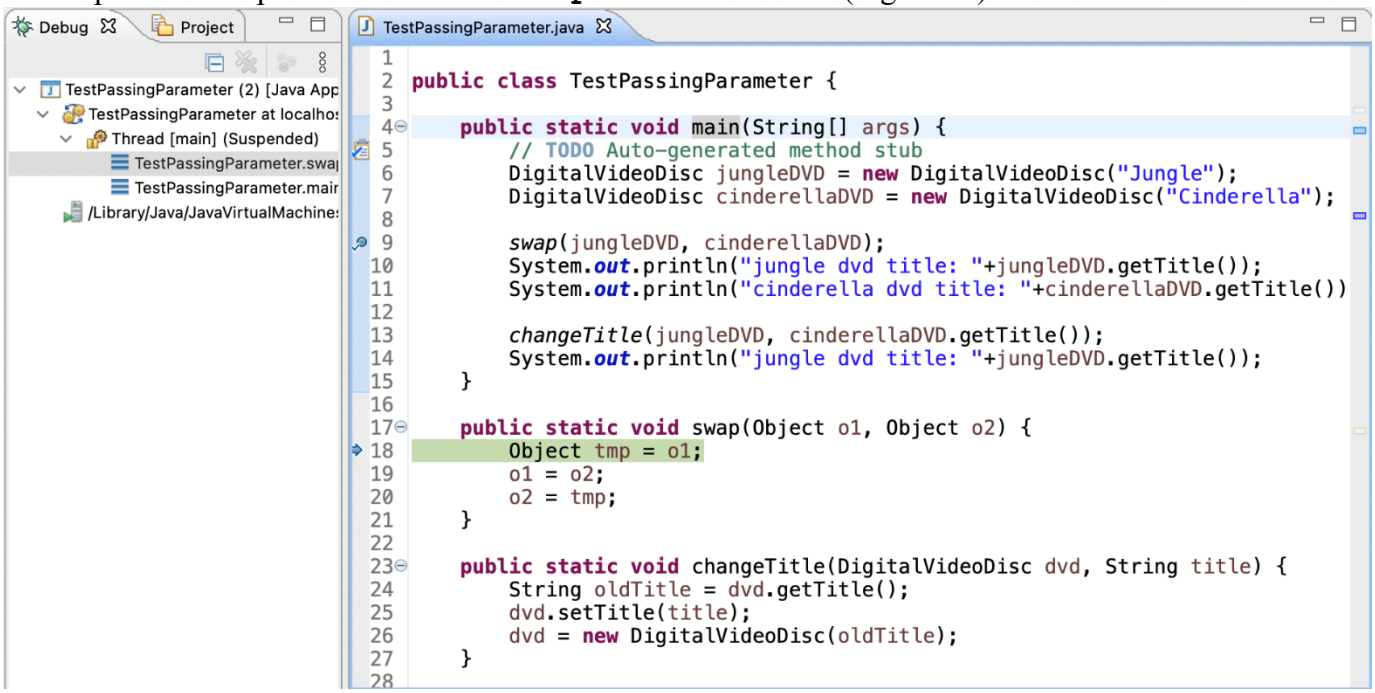

Figure 10. Step into swap function

3.2.4 Investigate value of variables:

We can observe the value of variables & expression in the Variables/Expression View. You can also add a permanent watch on an expression/variable that will then be shown in the Expressions view when debugging is on.

Alternatively, place your cursor on any of the variables in the Java action to see its value in a pop-up window.

Open the Variable Perspective and observe the values of variables **`o1`** & **`o2`** (Figure 11). You can click the drop-down arrow to investigate attributes of variables.

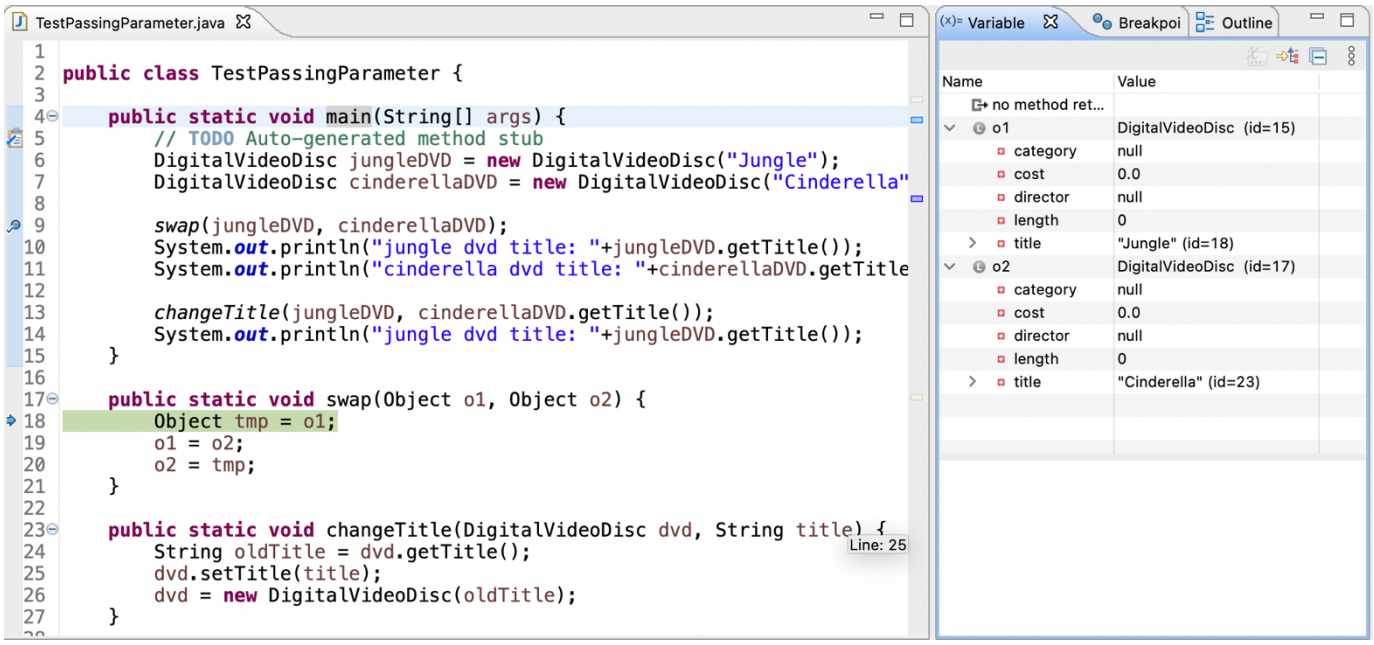

Figure 11. Variables shown in Variable View

Click Step Over and watch the change in the value of variables **`o1`**, **`o2`** & **`tmp`**. Repeat this until the end of the **`swap`** function (Figure 12, Figure 13, Figure 14).

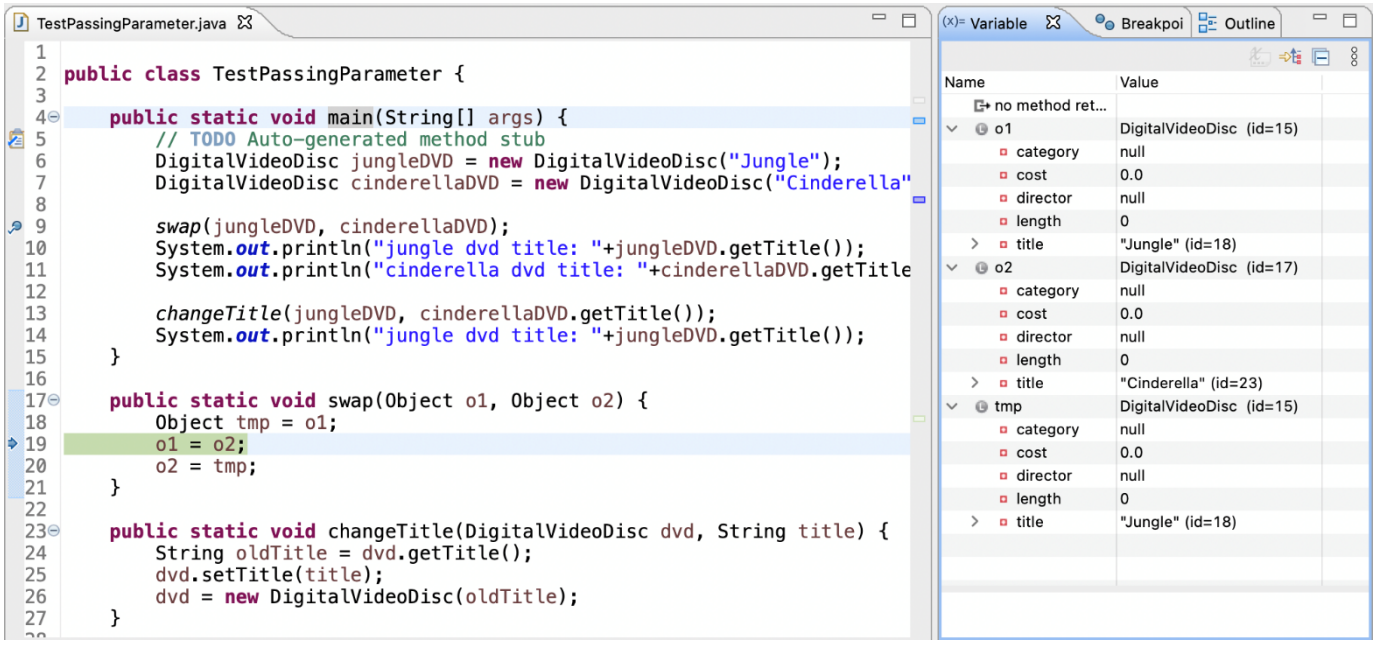

Figure 12. Step over line 18 of swap function

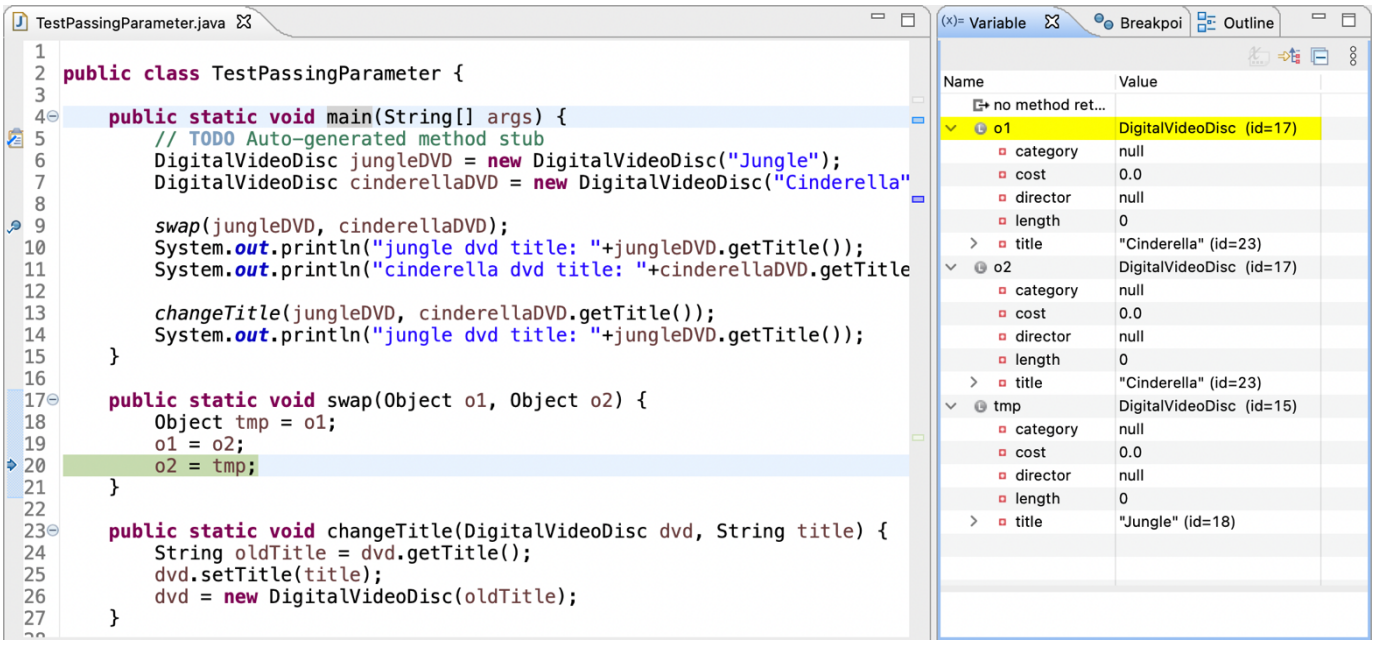

Figure 13. Step over line 19 of swap function

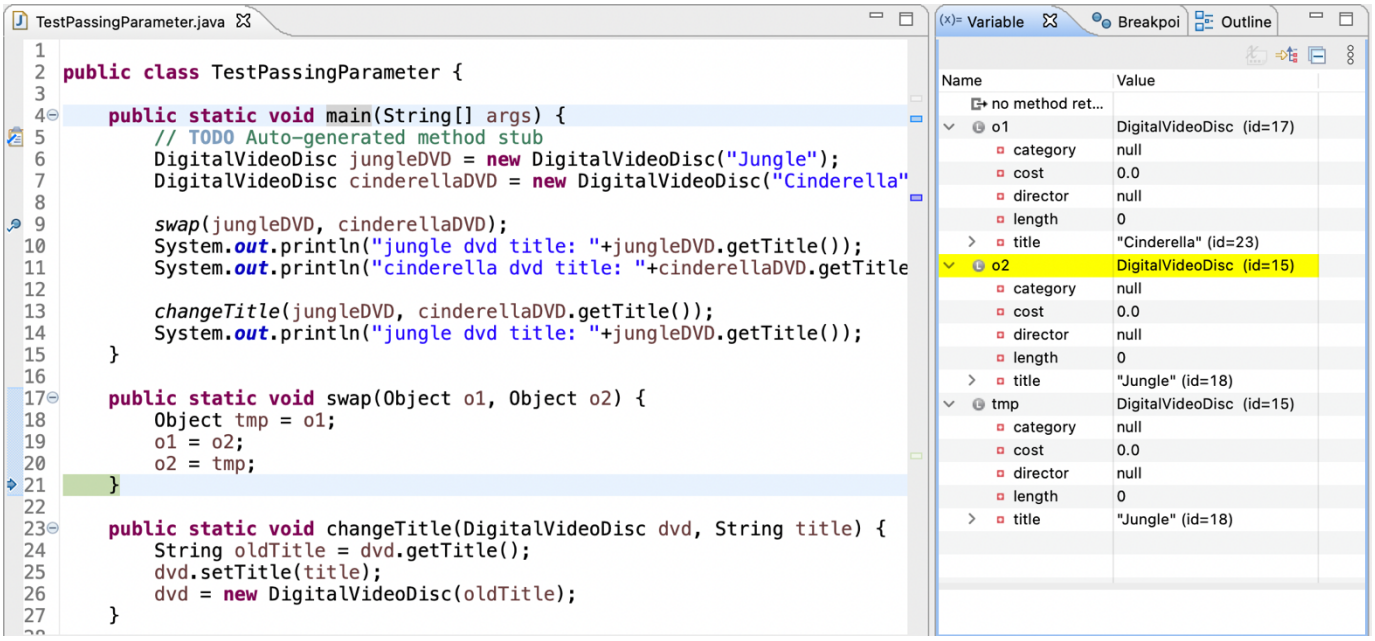

Figure 14. Step over line 20 of swap function

3.2.5 Change value of variables:

In the Variable Perspective, you can also change the value of the variable while debugging.
Click Step Return so the debugger returns from the **`swap`** function back to the line after the call to it. (Figure 15)

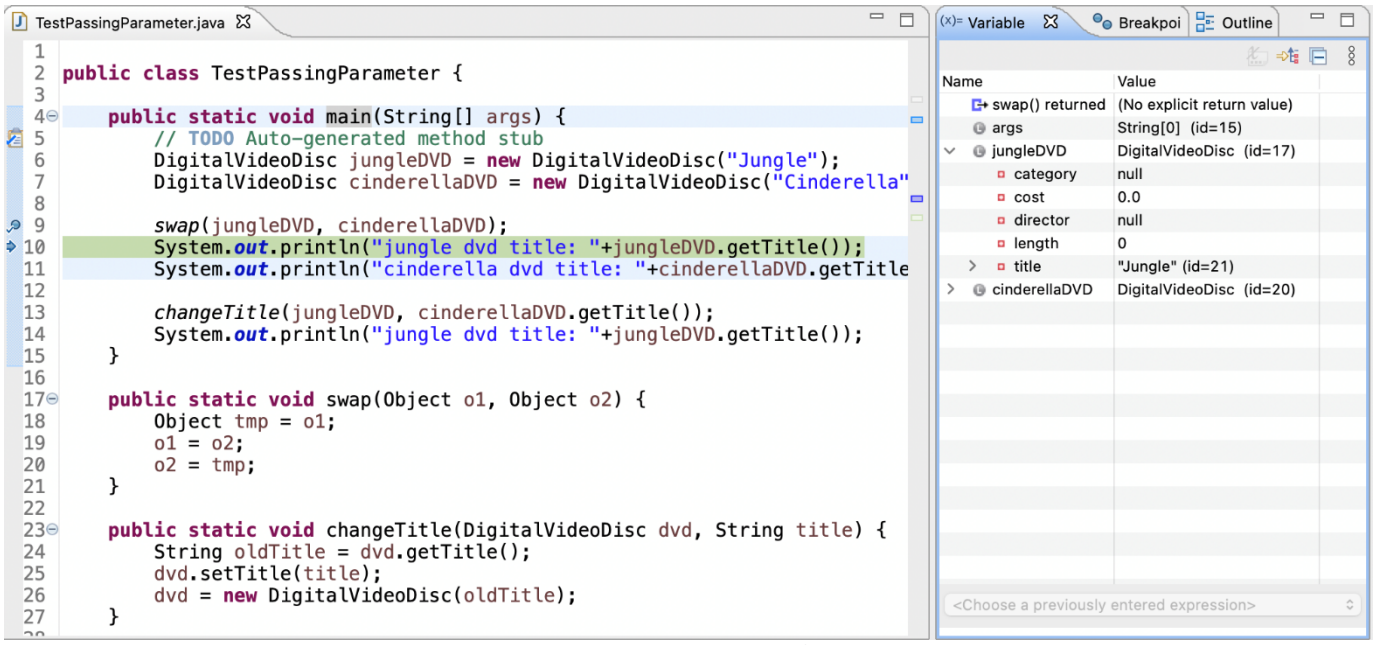

Figure 15. Step return to main function

The variable **`jungleDVD`** still has a title attribute with value "Jungle". You can change this value by clicking on it and change it to "abc", for example (Figure 16).

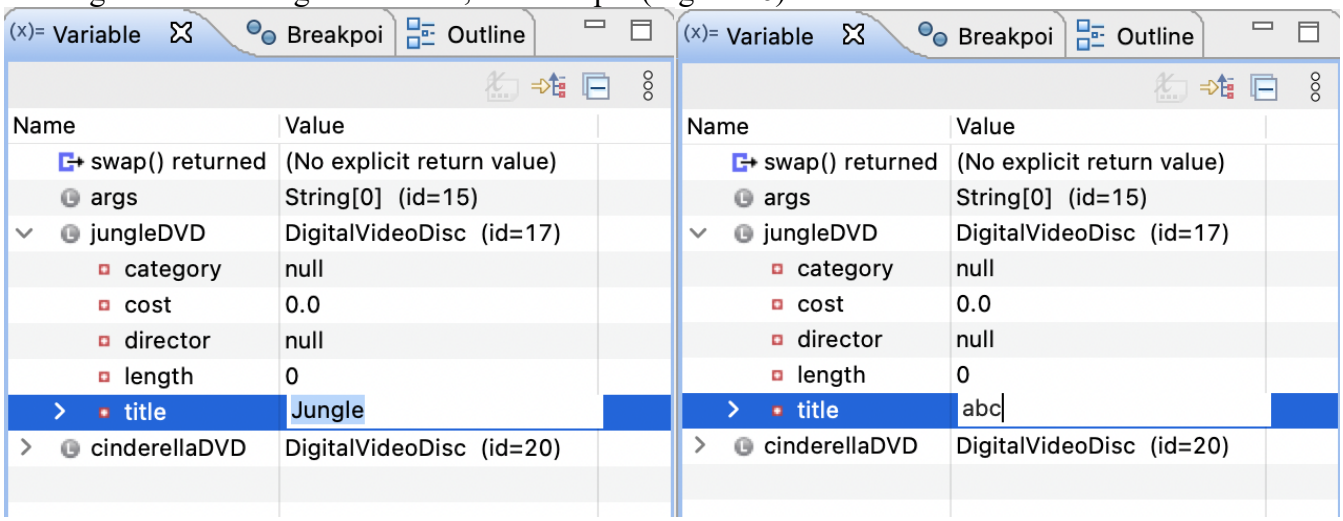

Figure 16. Change title of jungleDVD

Click Step Over and see the result in the output in the Console (Figure 17)

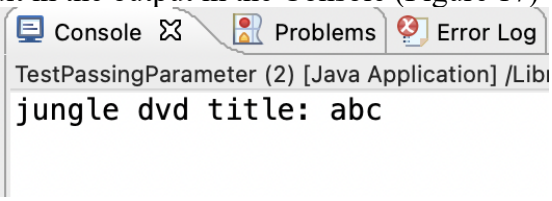

Figure 17. Results(2)

4 Classifier Member and Instance Member

- Classifier/Class member: static
  - Defined in a class of which a single copy exists regardless of how many instances of the class exist.
  - Objective: to have variables that are **common** to all objects

    - Any object of a class can change the value of the class variable; that's why you should always be careful 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 DigitalVideoDisc class:

- You should note that this class only has instance variables: **`title`**, **`category`**, **`director`**, **`length`**, **`cost`**.

- You add a new **`LocalDate`** (Javadocs link) private instance variable named "**`dateAdded`**" to store the date when the DigitalVideoDisc is added.
- This instance variable has a unique value to each instance of the **`DigitalVideoDisc`** class and must be initialized inside the constructor method of the **`DigitalVideoDisc`**.

- Now, we know that each DVD has a unique id assigned by the system. One simple way to manage all the ids is to give them out to new DVDs as consecutively incremented values. In order to do this, we must keep track of the number of DVDs created.
- Create a class attribute named "**`nbDigitalVideoDiscs`**" in the class **`DigitalVideoDisc`**
- Create an instance attribute named "**`id`**" in the class **`DigitalVideoDisc`**

```
private static int nbDigitalVideoDiscs = 0;
```

- Each time an instance of the **`DigitalVideoDisc`** class is created, the **`nbDigitalVideoDiscs`** should be updated. Therefore, you should update the value for this class variable inside the constructor method and assign the appropriate value for the **`id`**.

5 Create a new class DVDUtils:

- This class includes public static methods:
    + Compare two DVDs (by cost)
    + Compare two DVDs (by title)
    + Sorting a number of DVDs (by cost)
    + Sorting a number of DVDs (by title)
- Write a **`DVDTest`** class to test the two methods, the main responsibility of this class you is the following:
    + Create sample DVDs
    + Test the method on the sample DVD

A code snippet is provided below for the class **`DVDTest`** in Figure 18:

```
public class DVDTest {
    public static void main(String[] args) {
        DigitalVideoDisc dvd1 = new DigitalVideoDisc("AAAA", "aaaa", "aaaa", 1, 5.6f);
        DigitalVideoDisc dvd2 = new DigitalVideoDisc("BBBB", "bbbb", "bbbb", 2, 5.3f);
        DigitalVideoDisc dvd3 = new DigitalVideoDisc("CCCC", "cccc", "cccc", 3, 5f);
        DigitalVideoDisc dvd4 = new DigitalVideoDisc("DDDD", "dddd", "dddd", 4, 7.1f);
        DigitalVideoDisc dvd5 = new DigitalVideoDisc("EEEE", "eeee", "eeee", 5, 3.3f);

        System.out.println(DVDUtils.compareByCost(dvd1, dvd2));
        System.out.println(DVDUtils.compareByTitle(dvd5, dvd3));

        DigitalVideoDisc[] sorted = DVDUtils.sortByCost(new DigitalVideoDisc[] {dvd1, dvd2, dvd3, dvd4, dvd5});
        System.out.println("sort by cost: ");
        for (int i = 0; i < sorted.length; i++) {
            System.out.println(sorted[i].toString());
        }

        sorted = DVDUtils.sortByTitle(new DigitalVideoDisc[] {dvd1, dvd2, dvd3, dvd4, dvd5});
        System.out.println("sort by title: ");
        for (int i = 0; i < sorted.length; i++) {
            System.out.println(sorted[i].toString());
        }
    }
}
```

Figure 18. Source code of DVDTest

6 Open the Cart class

Write new methods to implement the following functions:

- Sort the DVDs in the cart by cost and print the result
- Sort the DVDs in the cart by title and print the result
- Search for DVDs in the cart by ID and display the search results. Make sure to notify the user if no match is found.
- Create a new method to print the list of ordered items of a cart, the price of each item, and the total price. The order of DVDs should be alphabetical, then by cost (decreasing), then by length (decreasing). Format the outline as below:

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

Suggestion: Write a **toString()** method for the **DigitalVideoDisc** class. What should be the return type of this method?

- In the **`CartTest`** class, write codes to test all methods you have written in this exercise. You should create sample DVDs and carts, like this code snippet:

```
public class CartTest {
    public static void main(String[] args) {
            //Create a new cart
            Cart cart = new Cart();

            //Create new dvd objects and add them to the cart
            DigitalVideoDisc dvd1 = new DigitalVideoDisc("The Lion King",
                    "Animation", "Roger Allers", 87, 19.95f);
            cart.addDigitalVideoDisc(dvd1);

            DigitalVideoDisc dvd2 = new DigitalVideoDisc("Star Wars",
                    "Science Fiction", "George Lucas", 87, 24.95f);
            cart.addDigitalVideoDisc(dvd2);

            DigitalVideoDisc dvd3 = new DigitalVideoDisc("Aladin",
                    "Animation", 18.99f);
            cart.addDigitalVideoDisc(dvd3);

            //Test the print method
            cart.print();
            //To-do: Test the search methods here

    }
}
```

7 Reorganize your projects

- Rename project, use packages and reorganize all hands-on labs and exercises from the Lab01 up to now.

+ For renaming or moving an item (i.e. a project, a class, a variable…), right-click on the item, choose Refactor -> Rename/Move and follow the steps.

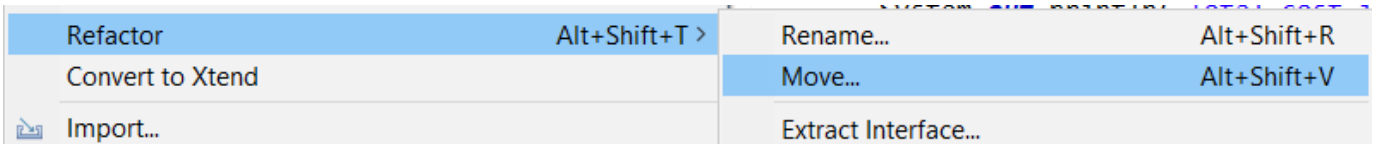

Figure 2. Refactoring

+ For creating a package, right-click to the project (or go to menu File) and choose New -> Package. Type the full path of the package including parent packages, separated by a dot.

- Keep the text file for answering questions in the lab, the "**Requirement**" & "**Design**" folders should be moved inside the root folder of **AimsProject**, next to its **src/** and **bin/** folder.

- The **structure of your labs** should be at least as below. You can create sub-packages for more efficiently organizing your classes in both projects and all listing packages. All the exercises of Lab 01 should be put in the corresponding package of one project - the OtherProjects project.

For Global ICT

+ AimsProject
hust.soict.globalict.aims.disc.DigitalVideoDisc
hust.soict.globalict.aims.cart.Cart
hust.soict.globalict.aims.Aims

```
    hust.soict.globalict.aims.utils.DVDUtils
    hust.soict.globalict.test.utils.DVDTest
    hust.soict.globalict.test.disc.TestPassingParameter
    hust.soict.globalict.test.cart.CartTest

+ OtherProjects
    hust.soict.globalict.lab01
```

For DS-AI class

```
+ AimsProject
    hust.soict.dsai.aims.disc.DigitalVideoDisc
    hust.soict.dsai.aims.cart.Cart
    hust.soict.dsai.aims.Aims
    hust.soict.dsai.aims.utils.DVDUtils
    hust.soict.dsai.test.utils.DVDTest
    hust.soict.dsai.test.disc.TestPassingParameter
    hust.soict.dsai.test.cart.CartTest
+ OtherProjects
    hust.soict.dsai.lab01
```

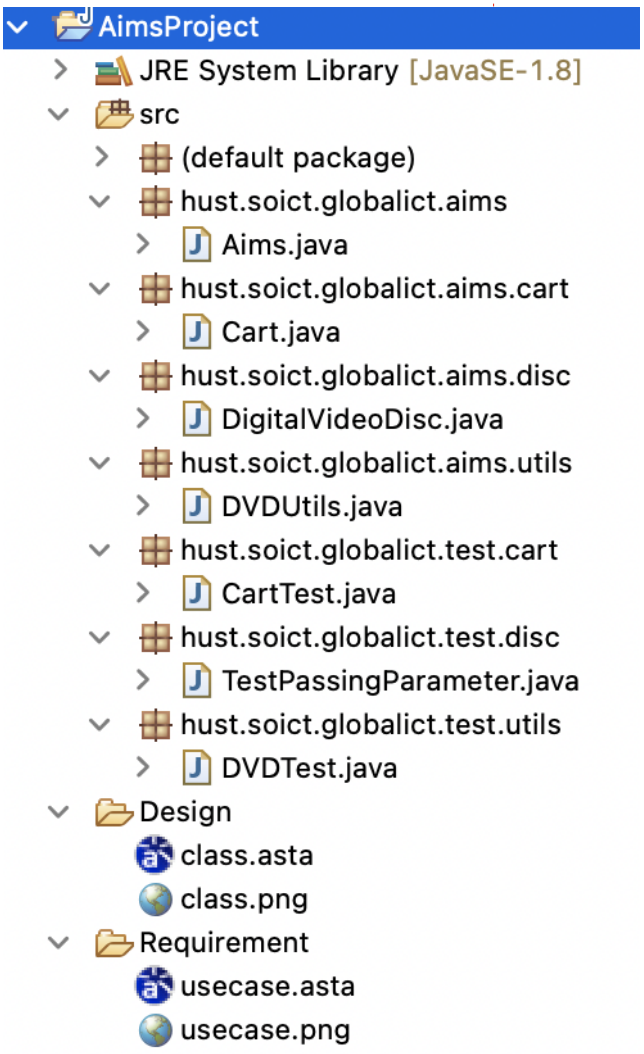

Figure 3. Recommended Structure for Global ICT

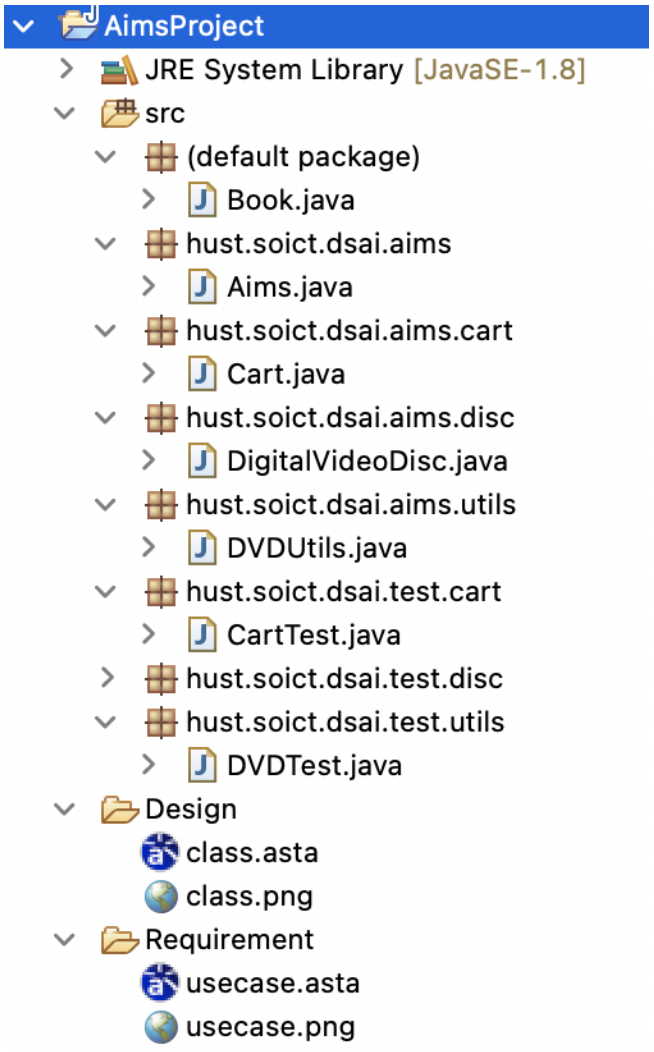

Figure 4. Recommended Structure for DS-AI

8 AimsProject project - search a dvd in cart

- In the **`DigitalVideoDisc`** class, write a **`boolean isMatch(String title)`** method which finds out (case insensitive) if the corresponding disk of the current object contains any token of **`title`**. Remember that if the **`title`** has multiple tokens (e.g. "Harry Potter"), the method still returns **`true`** if the disc has a **`title`** including at least one of the tokens (e.g. "Harry" or "Potter").
- Open the **`Cart`** class, write a new method to search for DVDs in the cart by title, and print the results. Make sure to notify the user if no match is found.

9 AimsProject - test new features

- Create **`DiskTest`** class in the package **`hust.soict.[globalict|dsai].aims`** to test the methods in section 3.

10 AimsProject – implement the `Store` class

- Create a **`Store`** class, which contains one attribute **`itemsInStore[]`** – an array of DVDs available in the store.
- To add and remove DVDs from the store, implement two methods called **`addDVD`** and **`removeDVD`**
- Test these two methods in the **`StoreTest`** class.

11 Create a complete console application in the Aims class

In the **`main`** method of **`Aims`**, you will now implement a complete console application, by first creating an instance of the **`Store`** class, and then providing a list of functionality through a menu that the user can interact with. For the home interface, you will create the main menu as the following:

```
public static void showMenu() {
            System.out.println("AIMS: ");
            System.out.println("--------------------------------");
            System.out.println("1. View store");
            System.out.println("2. Update store");
            System.out.println("3. See current cart");
            System.out.println("0. Exit");
            System.out.println("--------------------------------");
            System.out.println("Please choose a number: 0-1-2-3");
}
```

- From the main menu, if the user chooses option "**View store**", the application will display all the DVDs in the store, and a menu as follows:

```
public static void storeMenu() {
            System.out.println("Options: ");
            System.out.println("--------------------------------");
            System.out.println("1. See a DVD's details");
            System.out.println("2. Add a DVD to cart");
            System.out.println("3. See current cart");
            System.out.println("0. Back");
            System.out.println("--------------------------------");
            System.out.println("Please choose a number: 0-1-2-3");
}
```

- The option "**See a DVD's details**" will ask the user to enter the title of the DVD, and then display the information of that DVD. Please remember to check the validity of the title. In the information display, the system also allows the user to add that DVD to their cart.
- The option "**Add a DVD to cart**" will ask the user to enter the title of the DVD that he/she sees on the screen (the list of DVDs in the store), then add the media to his/her cart. Please remember to check the validity of the title. After adding a DVD to the cart, the system will display the number of DVDs in the current cart.

- From the main menu, if the user chooses the option "**Update store**", the application will allow the user to add a DVD to or remove a DVD from the store
- From the main menu, if the user chooses the option "**See current cart**", the application will display the information of the current cart and a menu as follows:

```
public static void cartMenu() {
        System.out.println("Options: ");
        System.out.println("--------------------------------");
        System.out.println("1. Filter DVDs in cart");
        System.out.println("2. Sort DVDs in cart");
        System.out.println("3. Remove DVD from cart");
        System.out.println("4. Place order");
        System.out.println("0. Back");
        System.out.println("--------------------------------");
        System.out.println("Please choose a number: 0-1-2-3-4");
}
```

The "**Filter DVDs in cart"** option should allow the user to choose one of two filtering options: by id or by title.

The "**Sort DVDs in cart**" option should allow the user to choose one of two sorting options: by title or by cost.

- Sort by title: the system displays all the DVDs in alphabetical order by title. In case they have the same title, the DVDs having the higher cost will be displayed first.
- Sort by cost: the system displays all the DVDs in decreasing cost order. In case they have the same cost, the DVDs will be ordered by title in alphabetical order.

Note: When the user chooses option "**Place order**", the system is designated to move on to the Delivery Information gathering & Payment step. However, for simplicity, within the scope of this lab course, when the user chooses this option, we only need to notify the user that an order is created, and then empty the current cart.

12 String, StringBuilder and StringBuffer

- In the **`OtherProjects`** project, create a new package **`hust.soict.globalict.garbage`** for ICT or **`hust.soict.dsai.garbage`** for DS-AI. We work with this package in this exercise.

- Create a new class **`ConcatenationInLoops`** to test the processing time to construct **`String`** using **`+`** operator, **`StringBuffer`**, and **`StringBuilder`**.

```java
public class ConcatenationInLoops {
    public static void main(String[] args) {
        Random r = new Random(123);
        long start = System.currentTimeMillis();
        String s = "";
        for (int i = 0; i < 65536; i++)
            s += r.nextInt(2);
        System.out.println(System.currentTimeMillis() - start);  // This prints roughly 4500.

        r = new Random(123);
        start = System.currentTimeMillis();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 65536; i++)
            sb.append(r.nextInt(2));
        s = sb.toString();
        System.out.println(System.currentTimeMillis() - start);  // This prints 5.
    }
}
```

Figure 5. ConcatenationInLoops

For more information on **`String`** concatenation, please refer to https://redfin.engineering/java-string-concatenation-which-way-is-best-8f590a7d22a8.

- Create a new class **`GarbageCreator`**. Create "garbage" as much as possible and observe when you run a program (it should let the program hang or even stop working when there is too much "garbage"). Write another class **`NoGarbage`** to solve the problem.

Some suggestions:

- Read a text/binary file to a **`String`** without using **`StringBuffer`** to concatenate String (only use **`+`** operator). Observe and capture your screen when you choose a very long file
- Improve the code using **`StringBuffer.`**

The following piece of code is a suggestion for your implementation

```java
        String filename = "test.exe"; // test.exe is the name or path to an executable file
        byte[] inputBytes = { 0 };
        long startTime, endTime;

        inputBytes = Files.readAllBytes(Paths.get(filename));
        startTime = System.currentTimeMillis();
        String outputString = "";
        for (byte b : inputBytes) {
            outputString += (char)b;
        }
        endTime = System.currentTimeMillis();
        System.out.println(endTime - startTime);
```

Figure 6. Sample code for GarbageCreator

Change the code in lines 14-17 above to use StringBuffer instead of the "+" operator to build the string and observe the result.

```
14          StringBuilder outputStringBuilder = new StringBuilder();
15          for (byte b : inputBytes) {
16              outputStringBuilder.append((char)b);
17          }
```

Figure 7. New code in line 14-17