Object-Oriented Programming

Lab 02: Problem Modeling and Encapsulation

In this lab, you will practice with:

- Working with release flow
- Installing a design tool for UML diagrams: Astah
- Problem Modeling with Use-case diagram
- Encapsulation and different techniques for encapsulation
- Class design for use cases related to cart management
- Java Implementation: Creating classes in Eclipse, constructors, getters and setters, creating instances of classes
- Method overloading
- Parameter passing
- Classifier member vs. Instance member

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.
- **A week before the next class**: For your upcoming deadline, you should prepare the following materials and organize them as described: included **the final use case diagram**, **the final class diagram**, **the source code** of all sections of this lab, and **the reading assignment** in a directory namely "**Lab02**". Note that for **the use case diagram**, submit both **source file (.astah)** and its **exported image (.png)** in the folder namely "**Requirement**". For **the class diagram**, also submit **both source file (.astah)** and its **exported image file (.png)** in the folder namely "**Design**"; for the reading assignment, submit an **image** file of the reading assignment in the folder namely "**ReadingAssignment**", put all into a directory namely "**Lab02**" and push it into a branch namely "**release/lab02**" of the valid repository.

Notice: All sample codes or diagrams in the lab are only the examples/suggestions. You may need to change them to satisfy the requirement. 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. Branch your repository

Day after day, your repository becomes more and more sophisticated, which makes your codes harder to manage. Luckily, a Git workflow can help you tackle this. A Git workflow is a **recipe for how to use Git** to control source code in a consistent and productive manner. Release Flow[1] is a lightweight but

[1] https://docs.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance?view=azure-devops

effective Git workflow that helps teams cooperate with a large size and regardless of technical expertise. Refer to the **Release-Flow-Guidelines.pdf** file for a more detailed guide.

Applying Release Flow is required from this lab forward.

However, we would use a modified version of Release Flow for simplicity.

- We can create as many branches as we need.
- We name branches with meaningful names. See Table 1-Branching policy.
- We had better **keep branches as close to `master` as possible**; otherwise, we could face merge hell.
- Generally, when we merge a branch with its origin, that branch has been history. We usually do not touch it a second time.
- **We must strictly follow the branching policy. Others are flexible.**

Branch	Naming convention	Origin	Merge to	Purpose

`feature` *or* `topic`	`+ feature/feature-name`
	`+ feature/feature-area/feature-name`
	`+ topic/description`
		`master`	`master`	Add **a** new feature or **a** topic

`bugfix`	`bugfix/description`	`master`	`master`	Fix **a** bug
		`feature`	`feature`	
`hotfix`	`hotfix/description`	`release`	`release & master[1]`	Fix **a** bug in a **submitted assignment after deadline**
`refactor`	`refactor/description`	`master`	`master`	Refactor
		`feature`	`feature`	
`release`	`release/labXX`	`master`	`none`	Submit assignment [2]

Table 1: Branching policy

[1] If we want to update your solutions within a week after the deadline, we could make a new hotfix branch (e.g., `hotfix/stop-the-world`). Then we merge the `hotfix` branch with `master` and with `release` branch for the last submitted assignment (e.g., `release/lab01`). In case we already create a `release` branch for the current week assignment (e.g., `release/lab02`), we could merge the `hotfix` branch with the current `release` branch **if need be,** or we can delete and then recreate the current release branch. (fix lại text)

[2] **Latest versions of projects in release branch serve as the submitted assignment**

Let's use Release Flow as our Git workflow and apply it to refactor our repositories.

Step 1: Create a new branch in our local repository. We create a new branch `refactor/apply-release-flow` from our `master` branch.

Step 2: Make our changes, test them, and push them into the previous branch that you have created. We move the latest versions of all our latest files from section 2 and section 6 such that they are under the master branch directly. I suggest moving all files from section 2 and 6 into a folder named 'Lab01'.
See https://www.atlassian.com/git/tutorials/undoing-changes to undo changes in case of problems. To improve the commit message, see https://thoughtbot.com/blog/5-useful-tips-for-a-better-commit-message.
Step 3: Make a pull request for reviews from our teammates[2]. We **skip** this step since we are **solo** in this repository. We, however, had better never omit this step when we work as a team.
Step 4: Merge branches. Merge the new branch `refactor/apply-release-flow` into `master` branch. The result is shown in the following figure.

Hints:

Typical steps for a new branch:

- Create and switch to a new branch (e.g. abc) in the local repo: **git checkout -b <name of the branch which you want to create>.** For example**: git checkout -b refactor/apply-release-flow**
- Make modification in the local repo
- Add all changes in the current directory and its subdirectories to the staging area: **git add .**
- Commit the change in the local repo: **git commit -m "What you had change"**
- Push the local branch to the remote branch: **git push origin <name of the branch which you have created>**
- Merge the remote branch (e.g. refactor/apply-release-flow) to the master branch (Using "Pull request" function in Github)

After completing all the tasks of that week, and merge all branches into master branch, you should create a release/labxx branch from the master in the remote repo (GitHub).

***Notice**: From this step, I assume that you have completed all the tasks from the previous step, such as refactoring the repository format in the previous step on page 3.*

For example, in lab02, there may be 7 main tasks. So, one possible way to apply release flow is to create 7 branches:

- Create a branch **topic/use-case-diagram** for the Use Case diagram you have done in section 5
- Create a branch **topic/class-diagram** for the UML Class diagram you have done in section 6
- Create a branch **feature/initial-aims** for the implementation of the section 7, 8, 9, 10
- Create a branch **feature/manage-cart** for the implementation of the section 11, 12, 13

[2] https://www.atlassian.com/git/tutorials/making-a-pull-request

- Create a branch **topic/method-overloading** for the implementation of the section 14
- Create a branch **topic/passing-parameters** for the implementation of the section 15
- Create a branch **topic/classifier-and-instance-member** for the implementation of the section 16

Refer to the demonstration of Release Flow in the last section of this lab for a more detailed guide.

2. Release flow demonstration

2.1 Hypothesis

We hypothesize that Figure 18 shows the branches of our current remote repository.

Figure 1. Branches of Remote Repository

Now we add a new topic or a new feature to our application. The next section shows us how to apply Release Flow in this hypothesis.

2.2 Demonstration

- **Step 1. Update local repository.**

  Issue the following command and resolve conflicts if any.

  (master) $ git pull

- **Step 2. Create and switch to a new branch in the local repository.**

  (master) $ git checkout -b feature/demonstrate-release-flow

- **Step 3. Make modifications in the local repository.**

- **Step 4. Add all changes in the current directory and its subdirectories to the staging area**

  (feature/demonstrate-release-flow) $ git add .

- **Step 5. Commit the change in the local repository.**

  (feature/demonstrate-release-flow) $ git commit -m “Change files in assignment folder”

- **Step 6. Push the local branch to the remote branch**

  (feature/demonstrate-release-flow) $ git push origin feature/demonstrate-release-flow

- **Step 7. Create a pull request in GitHub GUI (for working in a team only)**
  - Firstly, choose the “Pull requests” tab from the top navigation bar.

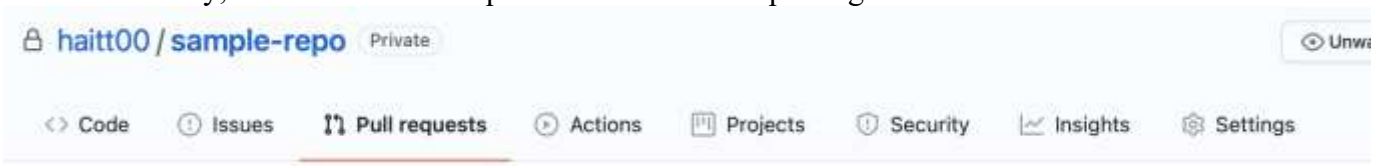


Figure 2. Creation of a Pull Request in GitHub GUI (1/4)

- Secondly, click the button “New pull request” in the top right corner of the interface.

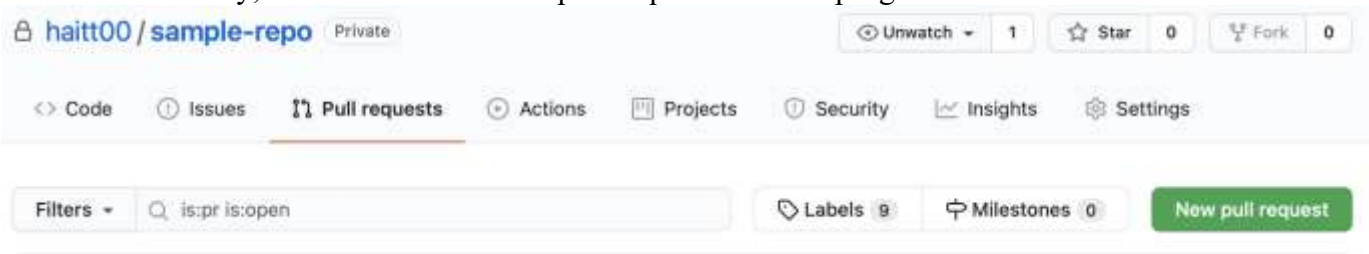


Figure 3. Creation of a Pull Request in GitHub GUI (2/4)

Then, pick the target branch and current branch. Besides, at the bottom of the interface, we can see the changes between the current branch and the target branch. Choose “Create pull request” to the top right.
Note: the target branch will affect the destination branch which we want our branch to merge in the next step.

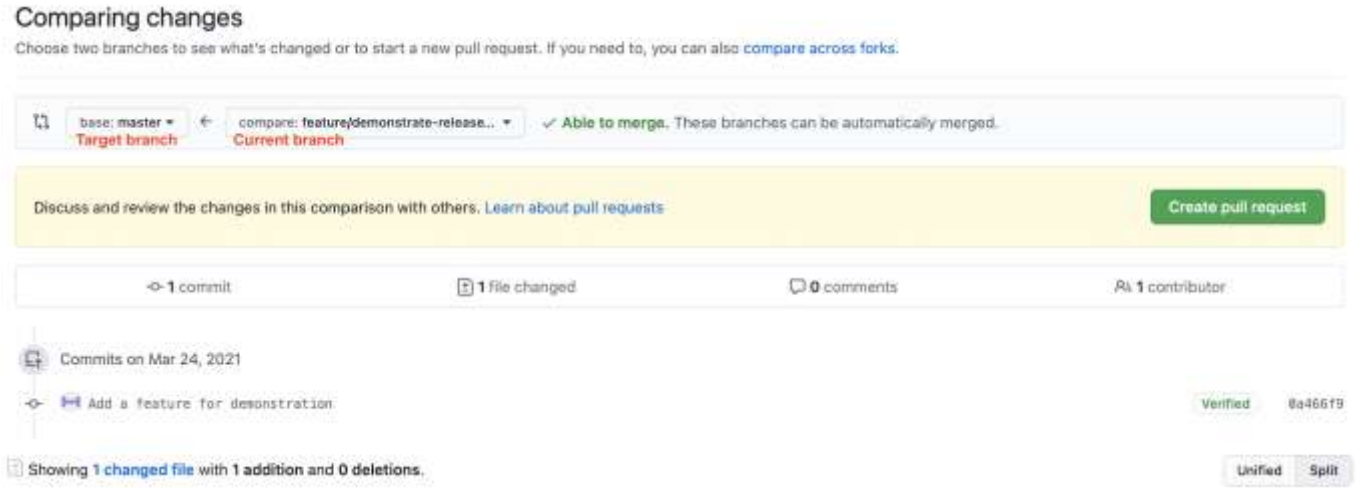
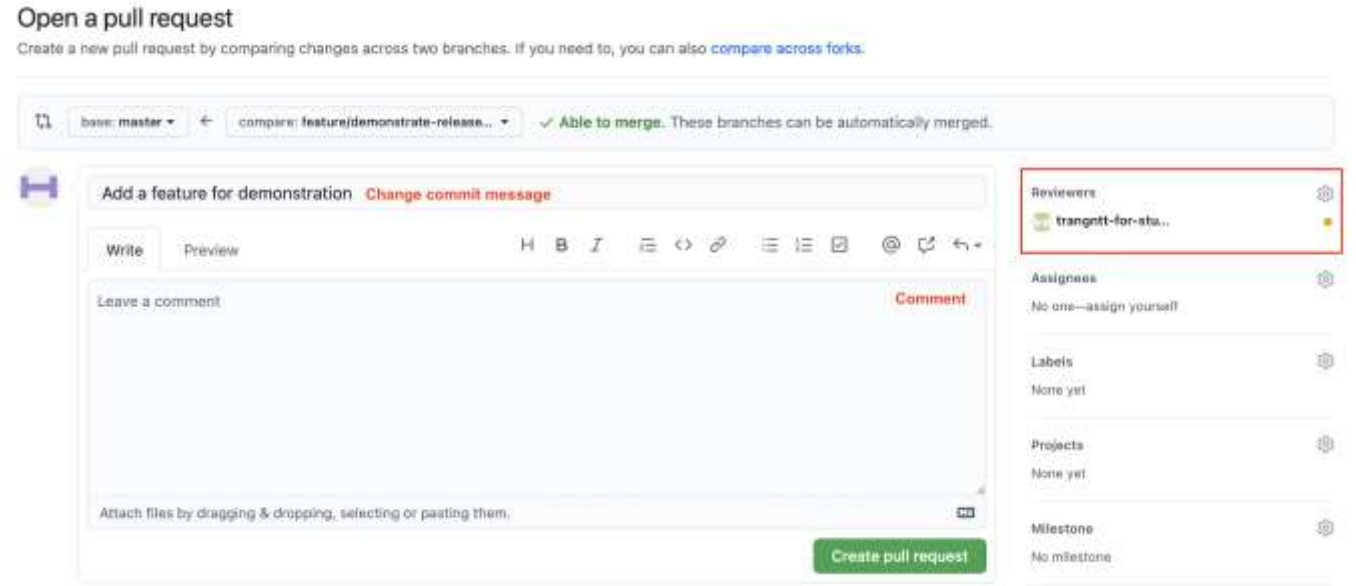


Figure 4. Creation of a Pull Request in GitHub GUI (3/4)

- Lastly, choose reviewers for the pull request. We can also change the commit message, and add comments as we desire. Choose “Create pull request”



The following figure shows the result of our efforts in the dashboard of GitHub. The added reviewers also can see the pull requests in their dashboard. When the changes are viewed, we can merge the branches.

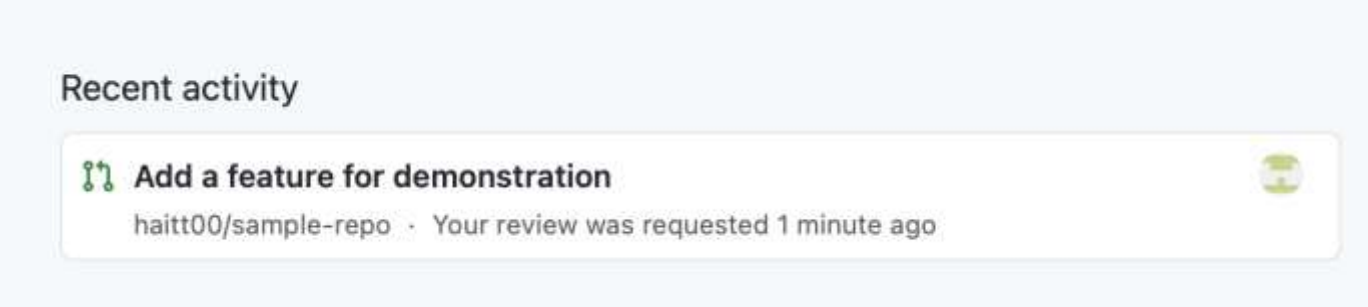


Figure 5. Creation of a Pull Request in GitHub GUI (4/4)

- **Step 8. Merge the new remote branch to the master branch.**
  - Open the pull request.
  - Choose "Merge pull request". You can choose one of several merge options from the drop-down menu

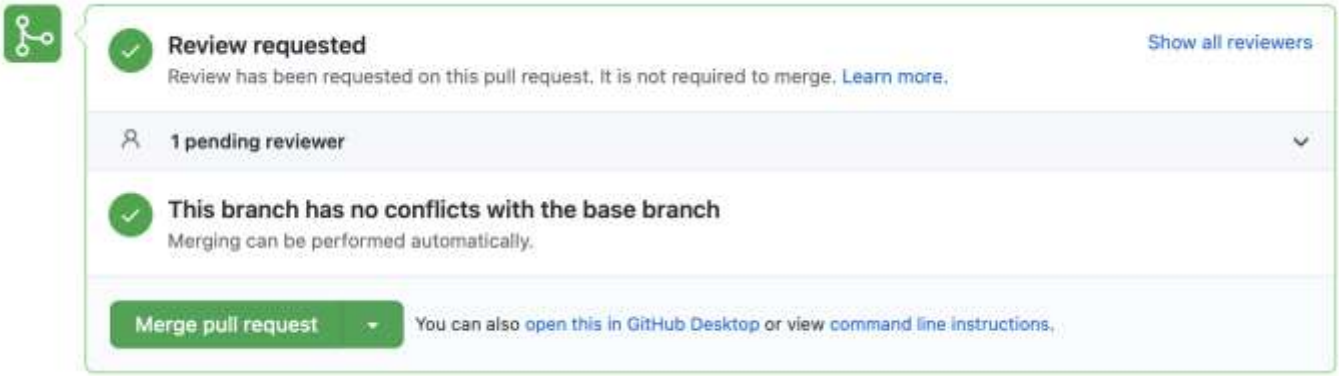


Figure 23. Branch merging (1/3)

  - Lastly, change the commit message if need be. We cannot change the destination branch. Choose "Confirm merge" (as shown in Figure 6)

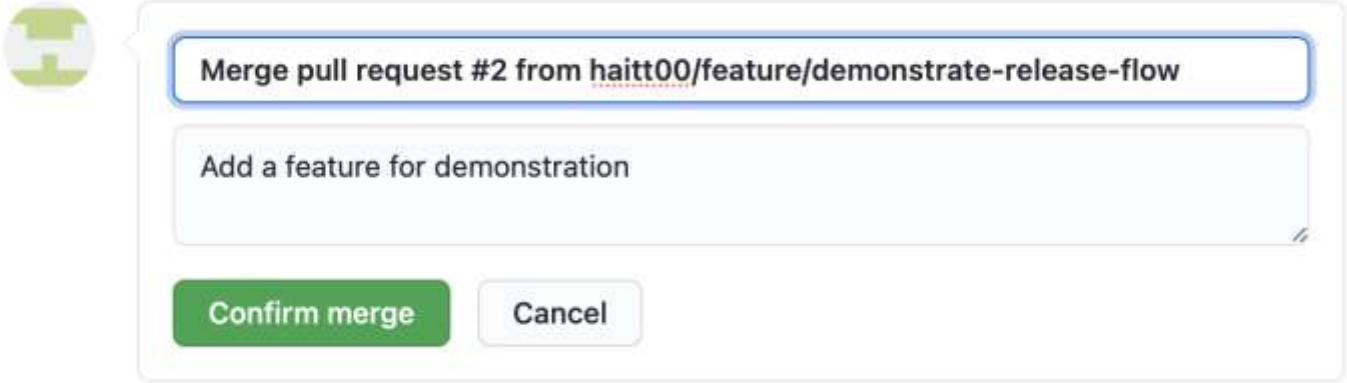
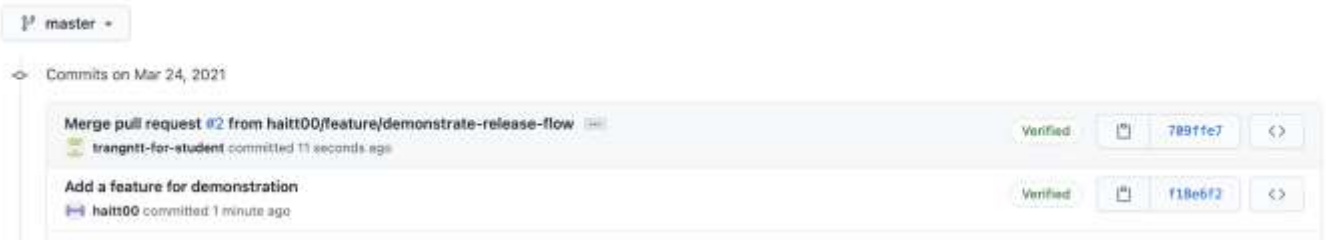


Figure 6. Branch merging (2/3)

The following figure shows the result of our efforts. The changes from the target branch have been merged to the target branch "master".



3. UML & Astah

The Unified Modeling Language (UML) is a family of graphical notations, backed by single metamodel, that help in describing and designing software systems, particularly software systems built using the object-oriented style (Fowler, 2003).

Astah is a design tool which supports UML. To get Astah UML, go to http://astah.net/student-license-request, fill the form, and send the request. Then follow the 3 steps in the redirected page http://astah.net/student/thank-you. See use case diagram with Astah at http://astah.net/manual/422-usecase-diagram.
Try to draw several UML diagrams in the Astah UML, such as class diagram, activity diagram, sequence diagram.

4. Problem Statement of AIMS Project

Given the below problem statement of AIMS Project. Please read carefully the statement to ensure that you understand clearly the software that you will design and implement from this lab. If you have any problems, please do not hesitate to ask the professor or TA.

There might be a future where Tiki and Sendo are in talks over a potential merger to contend other e-commerce platforms and especially those who have foreign backers. The merger of these two firms would create a Ti-do company, where "Ti" is from Tiki, and "do" is from Sendo, which means a billion-dollar company in Vietnamese. That firm, Ti-do company, would like you to help them create a brand-new system for the AIMS project (AIMS stands for An Internet Media Store). Currently, there is only one type of media: Digital Video Disc (DVD).

Customers can browse the list of DVDs available in the store, the display order is based on their added date, from latest to oldest. When a customer wants to search for DVDs to add to cart, he or she can choose one of three searching options. The software will display a list of all matches (latest DVDs first) with all their information. He or she can also choose to play a specific DVD. The software will play a DVD (a demo part). If a DVD has the length 0 or less, the system must notify the customer that the DVD cannot be played.

- When a customer searches for DVDs by title, he or she provides a string of keywords. If any DVD has the title containing any word in the string of keywords, it is counted as a match. Note that the comparison of words here is case-insensitive.
- When a customer searches for DVDs by category, he or she provides the category name. If any DVD has the matching category (case-insensitive), it is counted as a match.
- When a customer searches for DVDs by price, he or she provides either the minimum and maximum cost, or just the maximum cost.

Customers can view the detailed information of a DVD from the list of DVDs. He/she can add a DVD to a cart from a list of DVDs or the detail screen.

When a customer wants to see the current cart, the system displays all the information of the DVDs, along with the total cost. Customers may listen to a DVD (a demo part) in the cart before confirming to place an order. Customers can sort all DVDs in the cart by title or by cost:

- Sort by title: the system displays all the DVDs in the alphabet sequence 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 increasing the title.

Customers can update the quantity of a DVD in a cart or remove a DVD from a cart. To increase consumer demand for the product and grow sales, customers are allowed to have an item for free which is randomly picked out in the cart by the system. Customers can filter DVDs in the cart by providing either its ID or title. If the item is found, display information of the found item in the cart. Or else, notify the customer the item is not found in the current cart.

The customer can request to place an order when they are seeing the current cart. For simplification, he/she does not need to log in to place an order. The application will prompt the customer to enter the delivery information and delivery instructions. The software will then calculate the delivery fee based on the total mass of the order & the delivery location. Then, it will display to the customer the invoice including the DVD list, total cost before VAT, total cost after VAT, and the delivery fee. The customer can then proceed to pay for the order. Currently, only one payment method – i.e. credit card – is allowed by connecting to a card association system for checking the validation of the card or performing the pay transaction. After the transaction, the AIMS software will display all the detailed information such as transaction ID, card owner, transaction amount, transaction message, balance, transaction date to the customer. The order will be in pending state and the information of the order & the transaction will be sent to the customer's email.

The store manager needs to log in to the system to navigate to the management mode. He/she can see the list of pending orders, then can pick any order to see its details to approve or reject the order. The store manager can add new DVDs to the store. He or she must provide all information about the new DVD, including its ID, title, category, director, length, and the cost. Additionally, the manager can also remove DVDs from the store.

5. Use case diagram

Based on the problem statement in section 4, please draw the use case diagram using Astah UML for the AIMS project.

A use case diagram is one of the UML diagrams to captures dynamic behaviors, i.e., the *pattern of state change over time* (https://www.cs.uct.ac.za/mit_notes/software/htmls/ch05s08.html). The use case diagram illustrates the relations among use cases. A **use-case model** describes a software's functional requirements in terms of use cases. The use-case diagram is a model of the software's intended functions and its environment and serves as a contract between the customer and the developers. Because it is a very powerful planning instrument, the use-case diagram is generally used in all phases of the development cycle. For better understanding, see https://www.uml-diagrams.org/use-case-diagrams.html.

To draw the use case diagram, you must identify the actors and use cases. A sample use case diagram is illustrated in Figure 7 with one use case: place an order. *You have to identify more use cases to update the use case diagram for the final submission.*

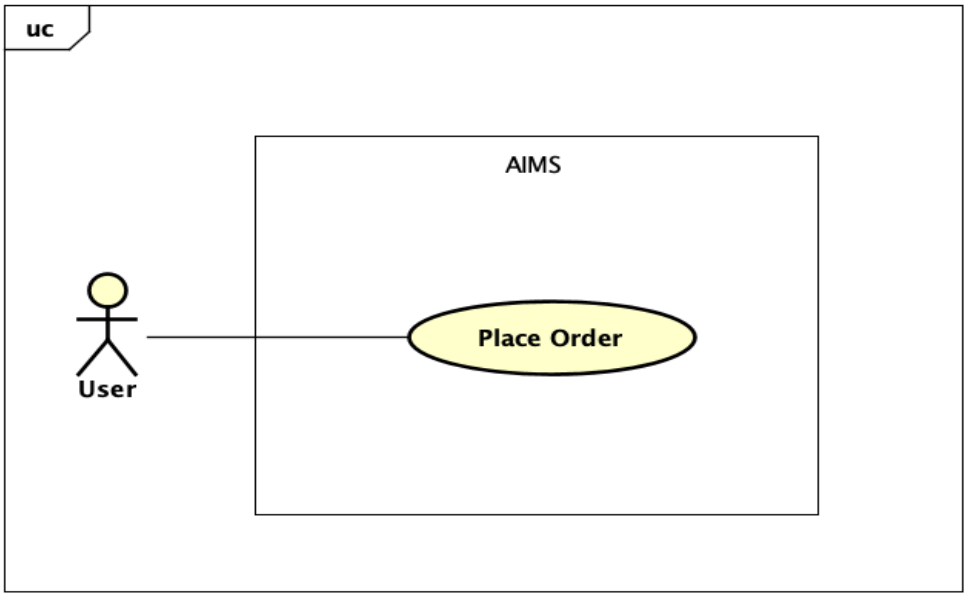


Figure 7. Sample of use case diagram for AIMS Project.

6. UML Class Diagram for use cases related to cart management

The system needs to create a new cart for the user, where it will keep information on the DVDs that the user wants to buy. The user can add, remove DVDs from the cart as well as calculate the cost. The user can add a maximum of 20 DVDs into one cart. The cart with its information and behaviors is modeled with the **Cart** class. When the user adds a DVD to the cart, the system must also create a new DVD based on the information that the user provides. This information can be displayed whenever the user decides to see it. The DVD with its information and functions is modeled with the DVD class. Finally, the application needs an entry point for displaying to and taking input from the user (via a command-line interface), which will be the Aims class. A sample class diagram is illustrated in Figure 8, which includes 3 classes:

- The **`Aims`** class which provides a `main()` method which interacts with the rest of the system
- The **`DigitalVideoDisc`** class which stores the title, category, cost, director and length
- The **`Cart`** class to maintain an array of these `DigitalVideoDisc` objects

You have to update this class diagram following the below exercises for the final submission.

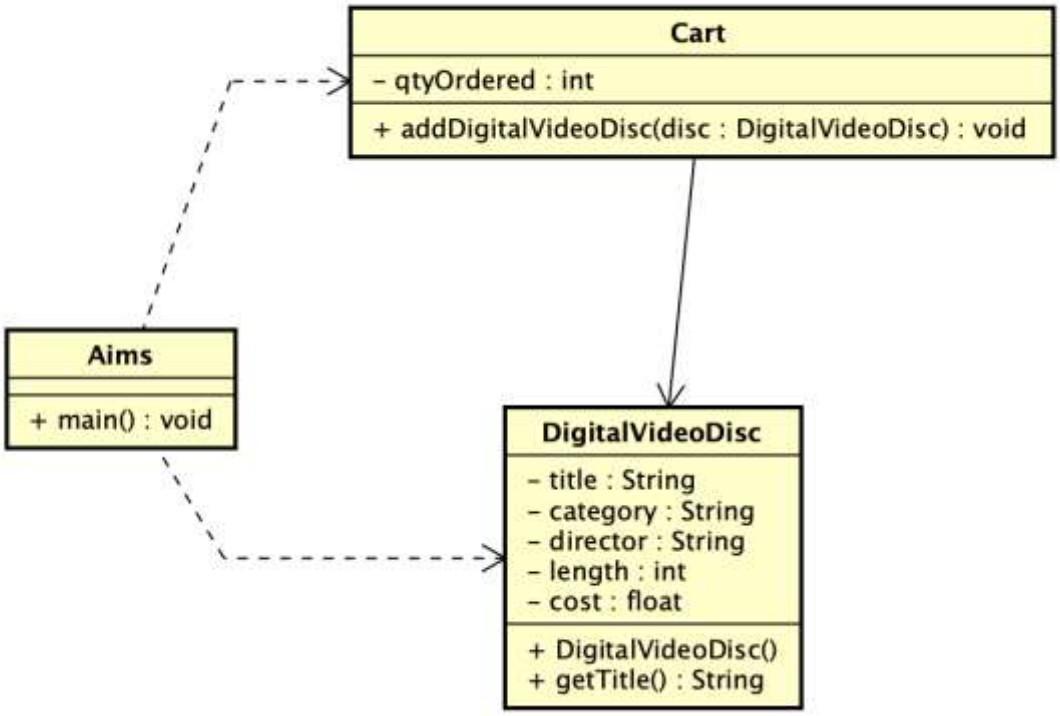


Figure 8. Sample class diagram for use cases related to cart management (you have to update it).

7. Create Aims class

- **Open Eclipse**
- **Create a new JavaProject named "AimsProject"**
- **Create Aims class:** In the src folder, create a new class named Aims:
  - Right click on the folder and choose New -> Class:

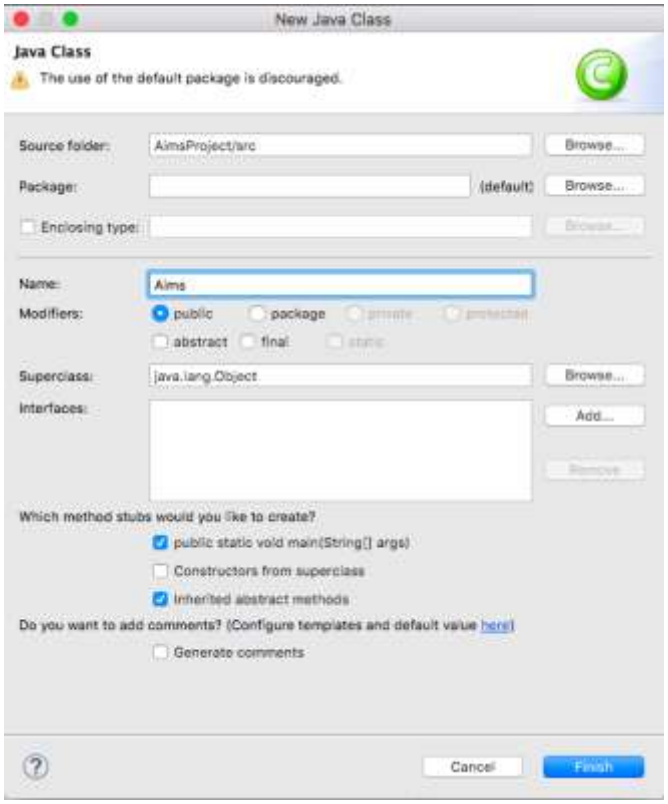


Figure 9. Create Aims class by Eclipse

- You may need to check the option "`public static void main(String[] args)`". This will automatically generate the main function in the class Aims.java as the following result.

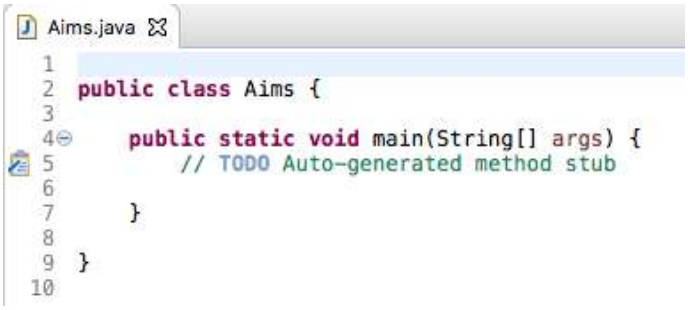


Figure 10. Generated code for Aims class

- Because you did not choose any package for the Aims class, Eclipse then displays the icon package and mentions (default package) for your class.

Figure 11. Aims in default package

- You can create a package and move the class to this package if you want. In the folder scr, a sub-folder will be created (with the name of the package) to store the class. Do it yourself and open the src folder to see the result.

8. Create the DigitalVideoDisc class and its attributes

Make sure that the option for the main method is not checked.
Open the source code of the DigitalVideoDisc class and add some attributes below:

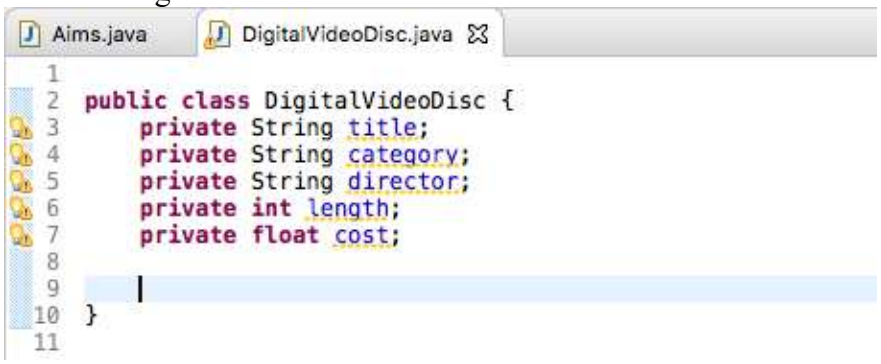


Figure 12. DigitalVideoDisc class

9. Create accessors and mutators for the class DigitalVideoDisc

To create setters and getters for private attributes, you can create methods to allow controlled public access to each of these private variables. Eclipse allows you to do this automatically. However, in many cases, you are not allowed to create accessors and mutators for all attributes, but depending on the business. E.g. in a bank account, the balance cannot be modified directly through a mutator, but should be increased or decreased through credit or debit use cases.

- **Right click anywhere in the source file of DigitalVideoDisc.**
- **Choose Source, then choose Generate Getters and Setters** (Figure 13)
- **Choose the attributes that needs getters/setters**
  - **For each of them, choose the dropdown arrow next to the tick box and choose to generate only setter, getter, or both.**

Suggestion: To choose the appropriate getters/setters, one should examine carefully the requirements of the system. In the case of Aims, based on the description of the system, we can decide the appropriate accessor methods for each attribute of the DVD class as follows: Firstly, there is no use case that requires the change of the attributes of a DVD after it is added, so we eliminate all the **setters**. Secondly, since the system needs to display all information of the DVDs when the user sees the current cart, all the **getters** are chosen. (Figure 14)

- **Choose the option "public" in the Access modifier**
  - **Click Generate**

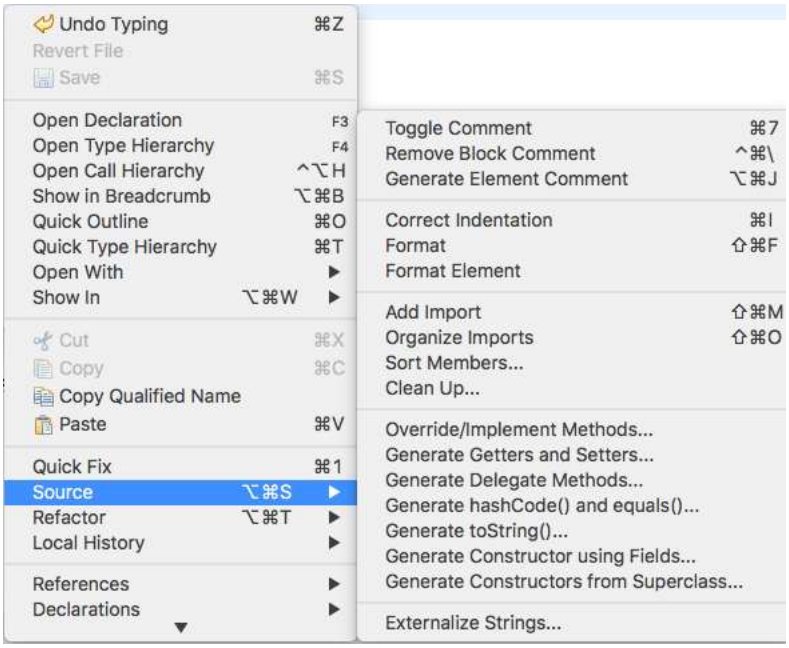


Figure 13. Generate getters & setters by Eclipse

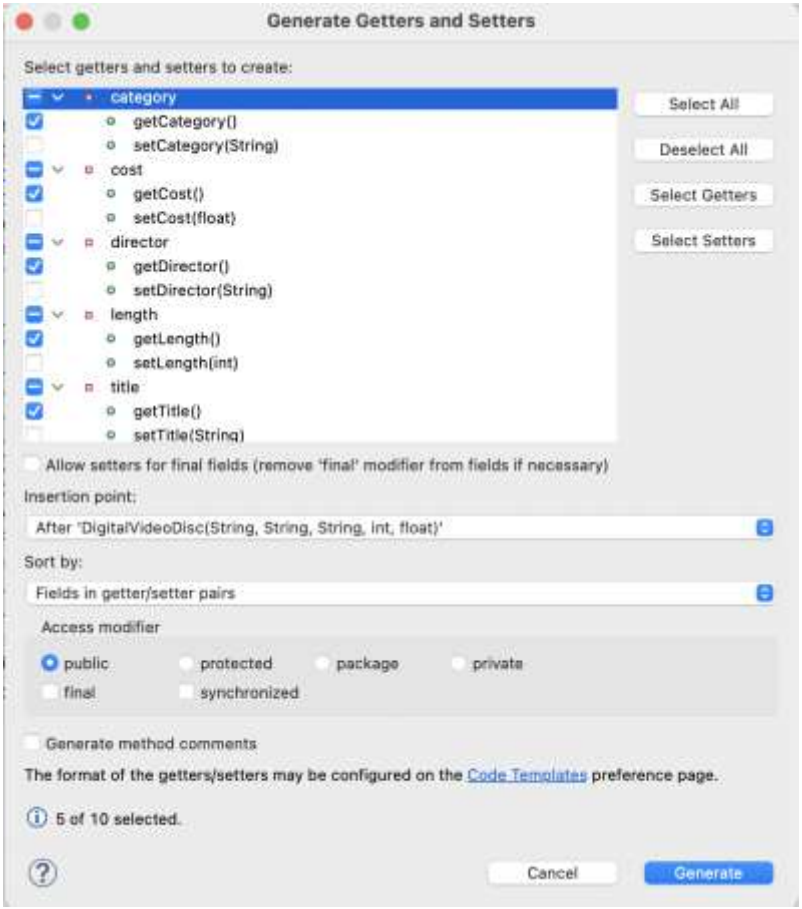


Figure 14. Choose appropriate accessors

```
public String getTitle() {
    return title;
}
public String getCategory() {
    return category;
}
public String getDirector() {
    return director;
}
public int getLength() {
    return length;
}
public float getCost() {
    return cost;
}
```

Figure 15. Generated accessors

Reading Assignment: When should accessor methods be used?

Read the following article and find the best possible answer to the above question: Holub, Allen. "Why getter and setter methods are evil" *JavaWorld*, 5 Sep. 2003, https://www.infoworld.com/article/2073723/why-getter-and-setter-methods-are-evil.html
You should expand your research to other sources as well. For the response, give a summary of your findings in the form of a mindmap. You can draw this mind map by hand and take a picture of your work or use any online tools. In both cases, the accepted format for the image file is one of the following: .png, .jpg, .jpeg and .pdf.

10. Create Constructor method

By default, all classes of Java will inherit from the **java.lang.Object**. If a class has no constructor method, this class in fact uses the constructor method of **java.lang.Object**. Therefore, you can always create an instance of class by a no-arguments constructor method. For example:

```
DigitalVideoDisc dvd1 = new DigitalVideoDisc();
```

In this part, you will create yourself constructor method for DigitalVideoDisc for different purposes:

- Create a DVD object by title
- Create a DVD object by category, title and cost
- Create a DVD object by director, category, title and cost
- Create a DVD object by all attributes: title, category, director, length and cost

Each purpose will be corresponding to a constructor method. By doing that, you have practiced with the overloading method.

Question:

- If you create a constructor method to build a **DVD** by title then create a constructor method to build a **DVD** by category. Does JAVA allow you to do this?

Eclipse also allows you to automatically generate constructor methods by field. Just do the same as generating getters and setters. Right click anywhere in the source file, Choose Source, Choose Generate constructors by fields (Figure 16) then select the fields (Figure 17) to generate constructor methods.

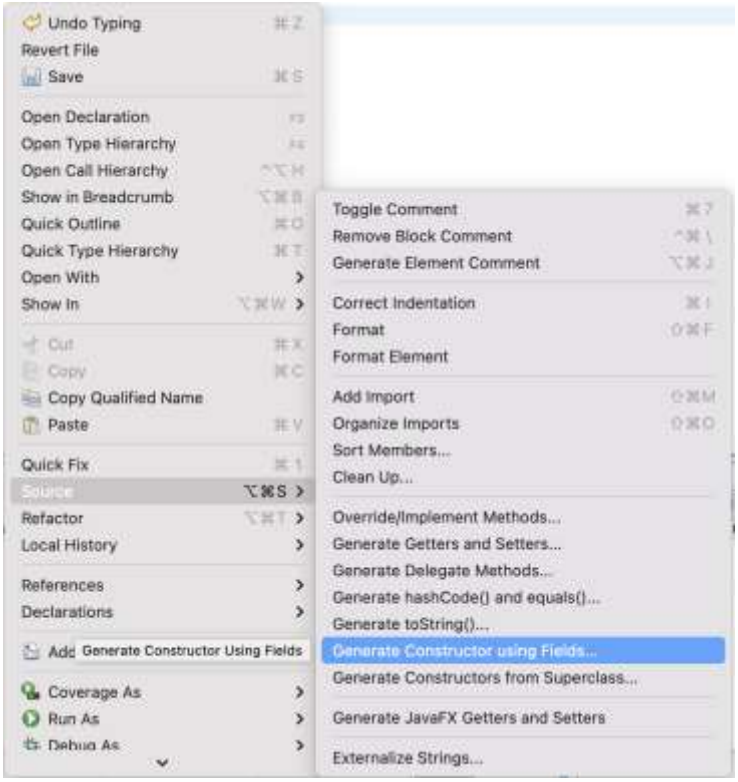


Figure 16. Generating constructor using fields

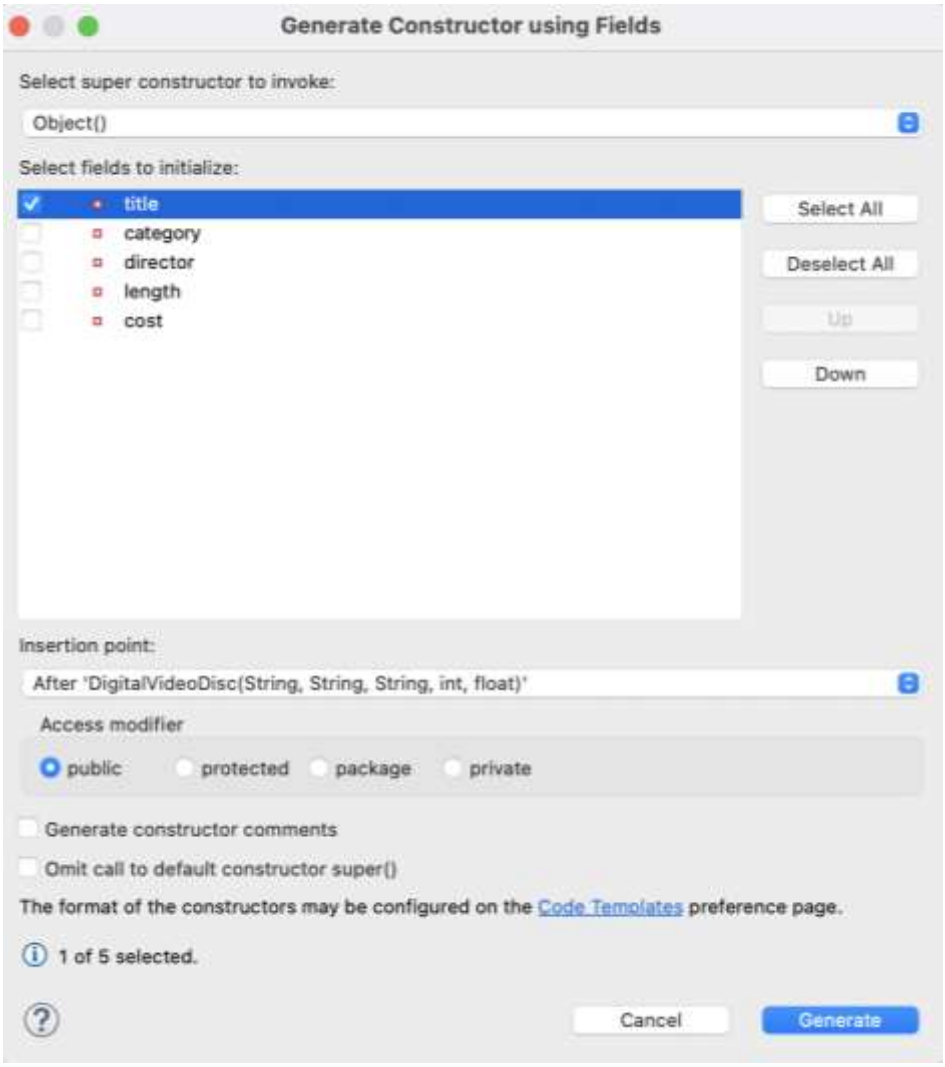


Figure 17. Setup for generating constructor using fields

The result is:

```
public DigitalVideoDisc(String title) {
    super();
    this.title = title;
}
```

Figure 18. A sample code for a generated constructor

This is how you create the first constructor method. Similarly, you will create by yourself the others.

11. Create the Cart class to work with DigitalVideoDisc

The **Cart** class will contain a list of **DigitalVideoDisc** objects and have methods capable of modifying the list.

```
public class Cart {

    public static final int MAX_NUMBERS_ORDERED = 20;
    private DigitalVideoDisc itemsOrdered[] =
            new DigitalVideoDisc[MAX_NUMBERS_ORDERED];

}
```

Figure 19. Sample code of the Cart class

Add a field as an array to store a list of **DigitalVideoDisc**.
To keep track of how many DigitalVideoDiscs are in the cart, you must create a field named **qtyOrdered** in the Cart class which stores this information.
Create the method **addDigitalVideoDisc(DigitalVideoDisc disc)** to add an item to the list. You should check the current quantity to assure that the cart is not already full

- Create the method **removeDigitalVideoDisc(DigitalVideoDisc disc)** to remove the item passed by argument from the list.

Create the **totalCost()** method which loops through the values of the array and sums the costs of the individual **DigitalVideoDiscs**. This method returns the total cost of the current cart.
Note that your methods should interact with users. For example: after adding it should inform the user: "**The disc has been added**" or "**The cart is almost full**" if the cart is full.
Now you have all the classes for the application. Just practice with them in the next section.

12. Create Carts of DigitalVideoDiscs

The **`Aims`** class should create a new Cart, and then create new DVDs and populate the cart with those DVDs. This will be done in the **`main()`** method of the Aims class. Do the following code in your main method and run the program to test.

```
public static void main(String[] args) {

    //Create a new cart
    Cart anOrder = 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);
    anOrder.addDigitalVideoDisc(dvd1);

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

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

    //print total cost of the items in the cart
    System.out.println("Total Cost is: ");
    System.out.println(anOrder.totalCost());

}
```

Figure 20. Sample code of the Aims class

The result should be:

Figure 21. Results for creating a cart of digital video discs.

You are asked to write more codes for Aims and/or Cart classes to display the cart items (sequence number, title and cost for each item in one line) before the total cost.

For example:

1	The Lion King	19.95
2	Star Wars	24.95
3	Aladin	18.99
	Total Cost	**63.89**

13. Removing items from the cart

You have to write code in your main method to test the **removeDigitalVideoDisc(DigitalVideoDisc disc)** method of the Cart class and check if the code is successfully run (add/remove items and display again the cart after updating).

14. Working with method overloading

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

14.1 Overloading by differing types of parameter

- Open Eclipse
- Open the JavaProject named "**AimsProject**" that you have created in the previous lab.
- **Open the class Cart.java:** you will overload the method `addDigitalVideoDisc` you created last time.
  - The current method has one input parameter of class `DigitalVideoDisc`
  - You will create a new method that has the same name but with different types of parameters.

    `addDigitalVideoDisc(DigitalVideoDisc [] dvdList)`

    This method will add a list of DVDs to the current cart.
  - Try to add a method `addDigitalVideoDisc` which allows to pass an arbitrary number of arguments for dvd. Compared to an array parameter. What do you prefer in this case?

14.2 Overloading by differing the number of parameters

- Continuing focus on the **`Cart`** class
- Create new method named **`addDigitalVideoDisc`**
    - The signature of this method has two parameters as following:
      `addDigitalVideoDisc(DigitalVideoDisc dvd1,DigitalVideoDisc dvd2)`

15. 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 22.

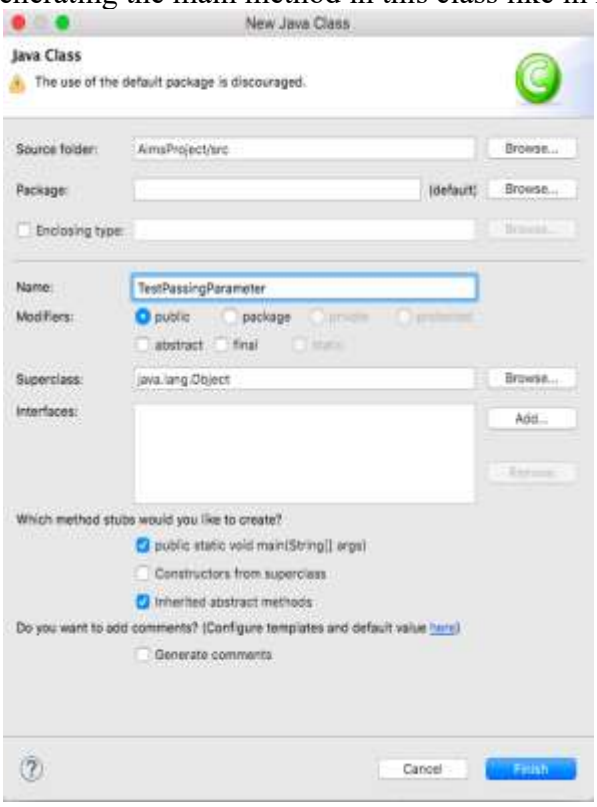


Figure 22. Create TestPassingParameter by Eclipse

- In the **`main()`** method of the class, typing the code below in Figure 23:

```java
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);
    }

}
```

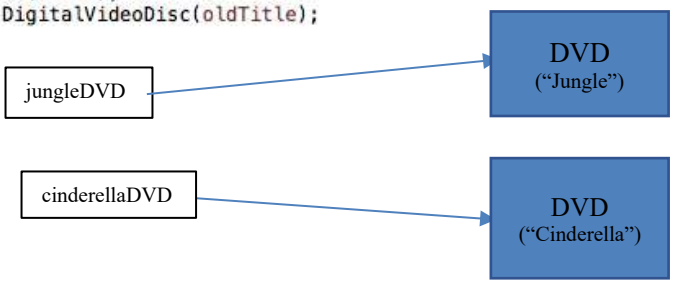


Figure 23. Source code of TestPassingParameter.

- The result in console is below:

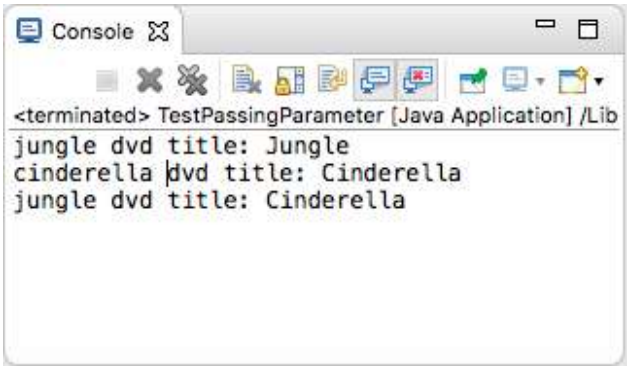


Figure 24. Results(1)

- 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.

Question:

- 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.

16. Classifier Member and Instance Member

- Classifier/Class member:
  - 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 class can change the value of a 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

Now, you open the **DigitalVideoDisc** class:

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

- 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`**.

17. References

James Rumbaugh, Ivar Jacobson, and Grady Booch (2004). *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education