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 3: Encapsulation and Method Overloading

In this lab, you will practice with:
- Working with Release workflow
- Encapsulation and different techniques for encapsulation
- Class design for use cases related to cart management
- Java Implementation: Creating classes in Eclipse, constructors, getters, setters, and creating instances of classes
- Method overloading

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 the class diagram, the **source code** of all sections of this lab, and the reading assignment in a directory namely "**Lab03**". Note that for the class diagram, submit both the 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**", into the directory "**Lab03**" 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.

Note that all sample codes or diagrams in the lab are only 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 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 merging 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 policy for the release branch. 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 the 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 the `master` and with the `release` branch for the last submitted assignment (e.g., `release/lab05`). In case we have already created a `release` branch for the current week assignment (e.g.,

---

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

release/lab06), 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.

[2] **The latest versions of projects in the 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**. We move the latest versions of all our latest files from previous labs such that they are under the master branch directly. 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 the `master` branch.

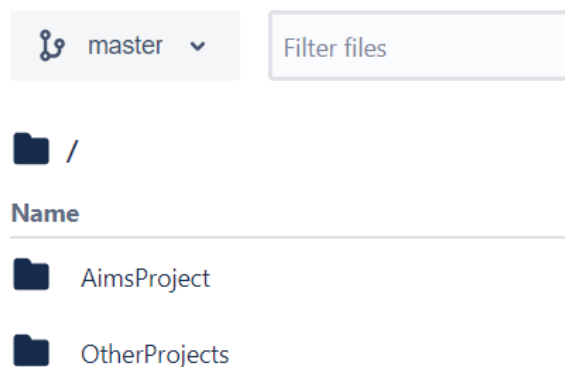The result is shown in the following figure.

Figure 1-Merging result

Hints:
Typical steps for a new branch:

- Create and switch to a new branch (e.g. abc) in the local repo: **git checkout -b abc**
- Make modifications in the local repo
- Commit the change in the local repo: **git commit -m "What you had changed"**
- Create a new branch (e.g. abc) in the remote repo (GitHub through GUI)
- Push the local branch to the remote branch: **git push origin abc**
- Merge the remote branch (e.g. abc) to the master branch (GitHub through GUI)

---

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

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).

For example, in the lab03, there may be 5 main tasks. So, one possible way to apply release flow is to create 5 branches:

- Create a branch **topic/class-diagram-cart-management** for updating the UML class diagram for use cases related to cart management
- Create a branch **topic/dvd** for the implementation of the class DigitalVideoDisc in this lab (including attributes, accessors and mutators, constructors)
- Create a branch **topic/cart** for creating the Cart class to work with DigitalVideoDisc and testing to create a new cart of DVDs
- Create a branch **feature/remove-item-in-cart** for the implementation of the feature remove an item in the cart
- Create a branch **topic/method-overloading** for the exercise on method overloading

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

2.   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, and remove DVDs from the cart as well as calculate the cost. The user can add a maximum of 20 DVDs to 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 1, which includes 3 classes:

- The **Aims** class provides a main() method that 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

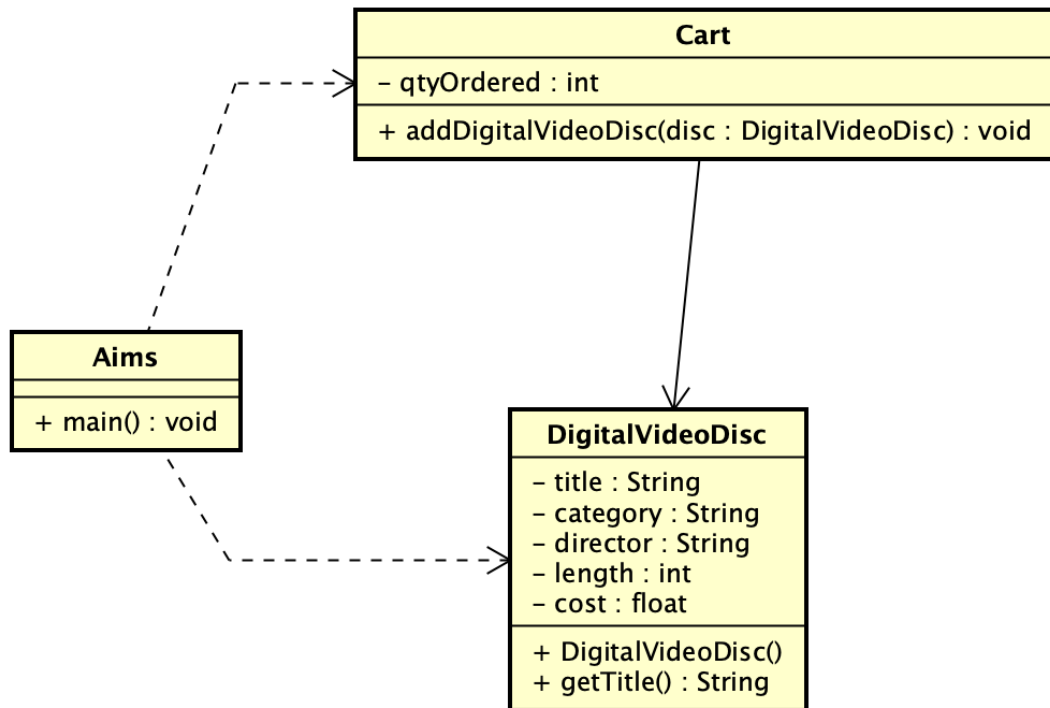***You have to update this class diagram following the below exercises for the final submission.***

Figure 1. Sample class diagram for use cases related to cart management.

3.   Create Aims class

- Open Eclipse

- Create a new JavaProject named "AimsProject"

- Create Aims class: In the src folder, create a new class named Aims:
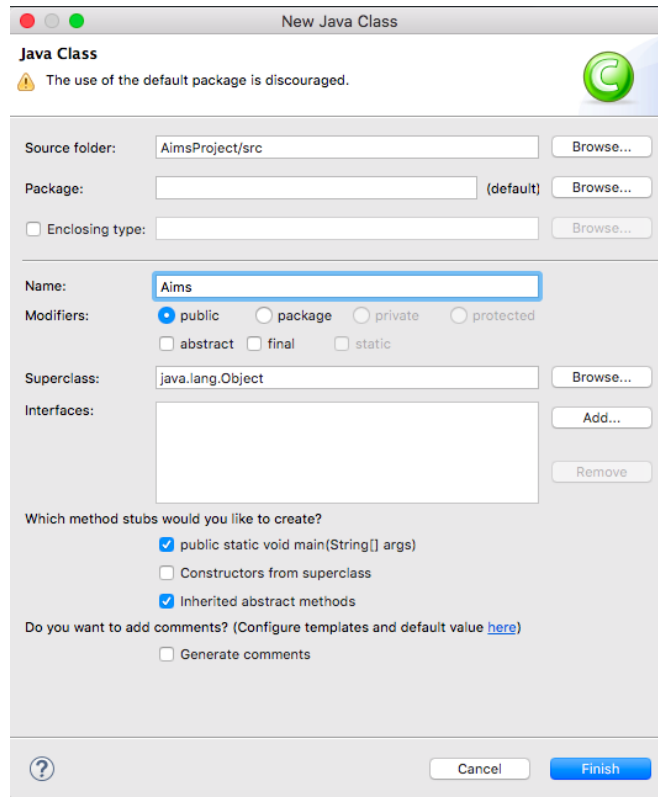
+ Right-click on the folder and choose New -> Class:

Figure 2. 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 3. 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.
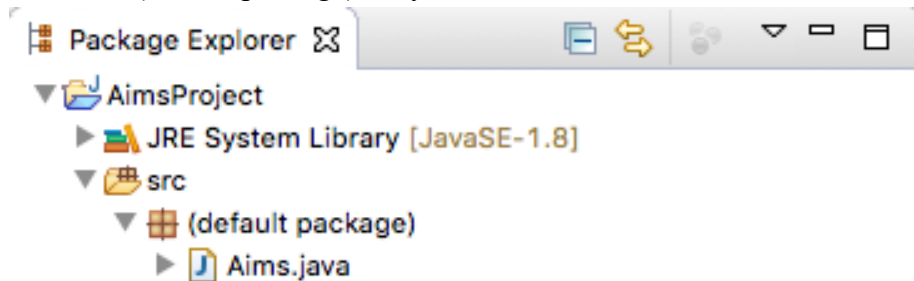

Figure 4. Aims in default package

Page **6** of **17**

+ You can create a package and move the class to this package if you want. In the folder src, 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.

4.   Create the DigitalVideoDisc class and its attributes

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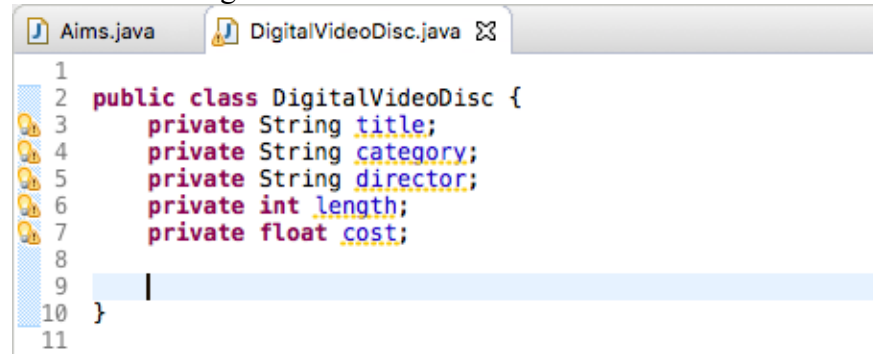Open the source code of the DigitalVideoDisc class and add some attributes below:

Figure 5. DigitalVideoDisc class

5.   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 6)
- **Choose the attributes that need getters/setters**
    - **For each of them, choose the dropdown arrow next to the tick box and choose to generate only setter, getter, or both.**
      <u>Suggestion</u>: 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 about the DVDs when the user sees the current cart, all the **getters** are chosen. (Figure 7)
- **Choose the option "public" in the Access modifier**
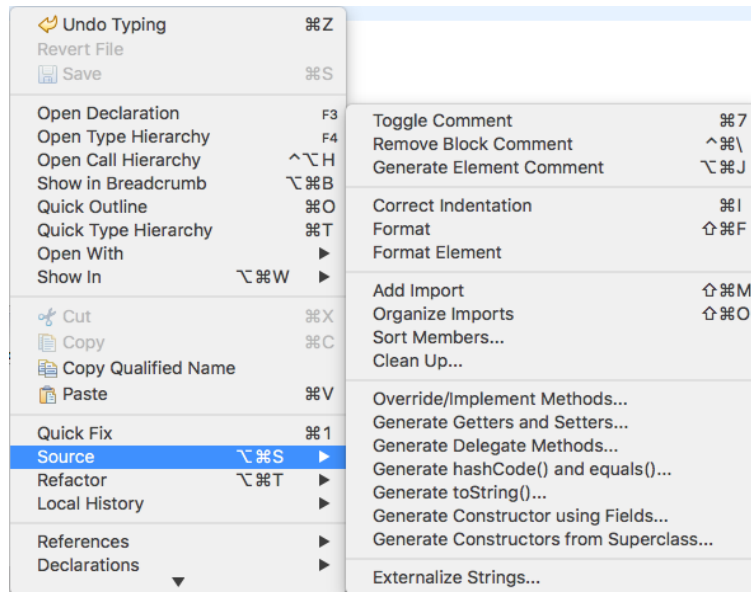- **Click Generate**
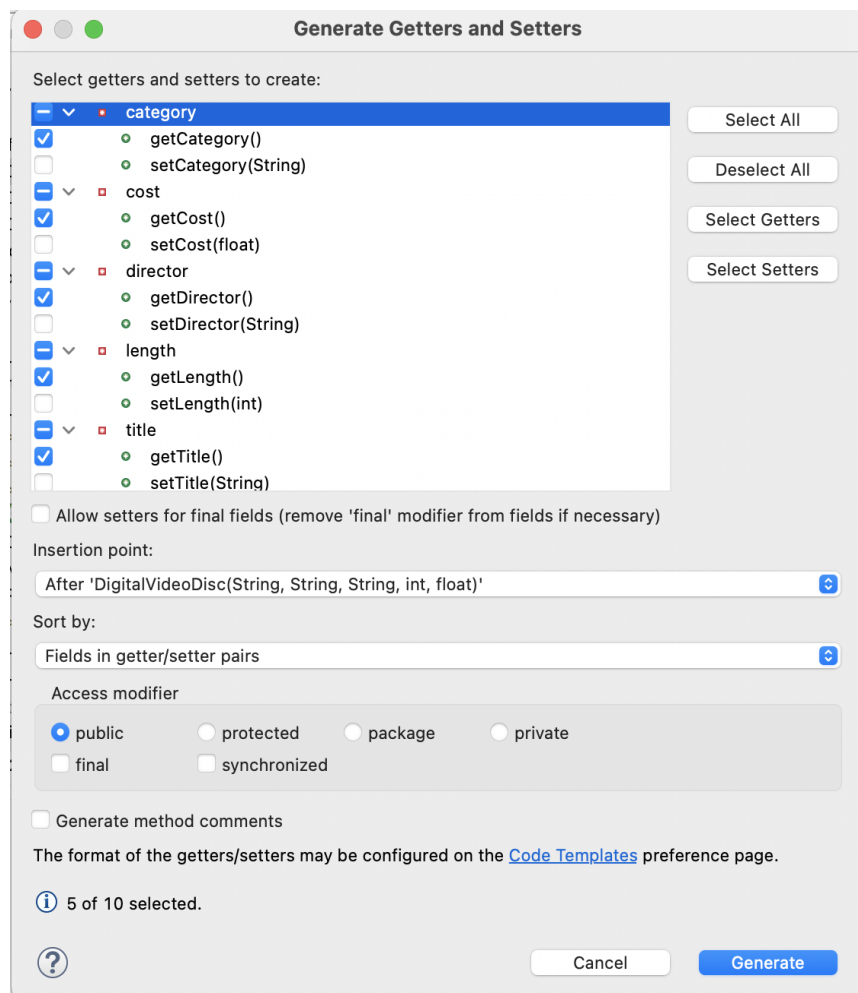
Figure 6. Generate getters & setters by Eclipse

Figure 7. Choose appropriate accessors

```java
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 8. 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 mind map. 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.

6.   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 a class by a no-argument 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 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 using fields (Figure 9) then select the fields (Figure 10) to generate constructor methods.
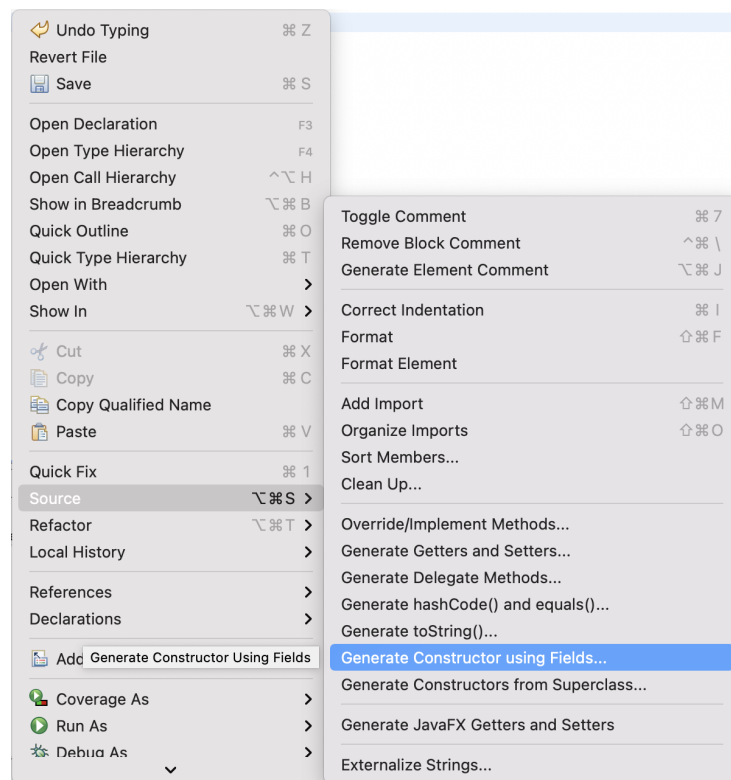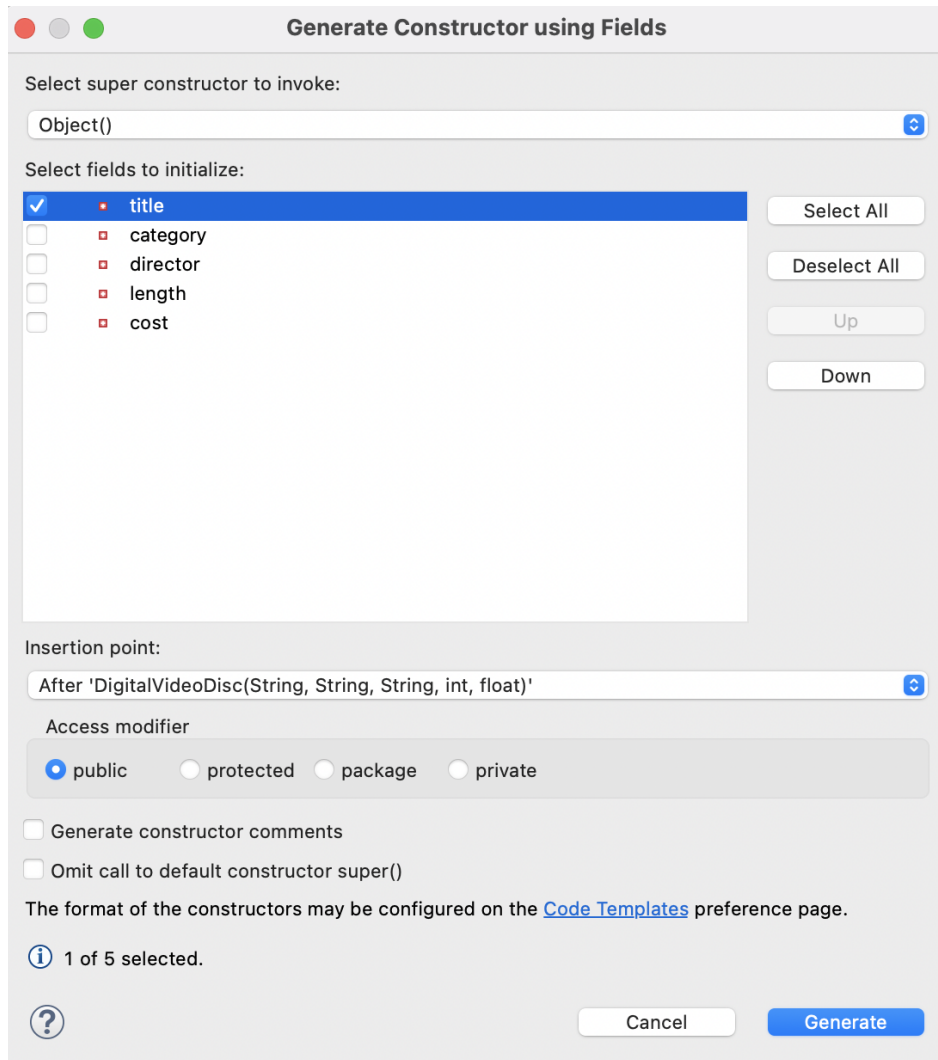

Figure 9. Generating constructor using fields

Figure 10. Setup for generating constructor using fields

The result is:

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

Figure 11. A sample code for a generated constructor

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

7.   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.

Add a field as an array to store a list of **DigitalVideoDisc**.

```java
public class Cart {

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

}
```

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.

8.   Create a Cart 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.

```java
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());

}
```

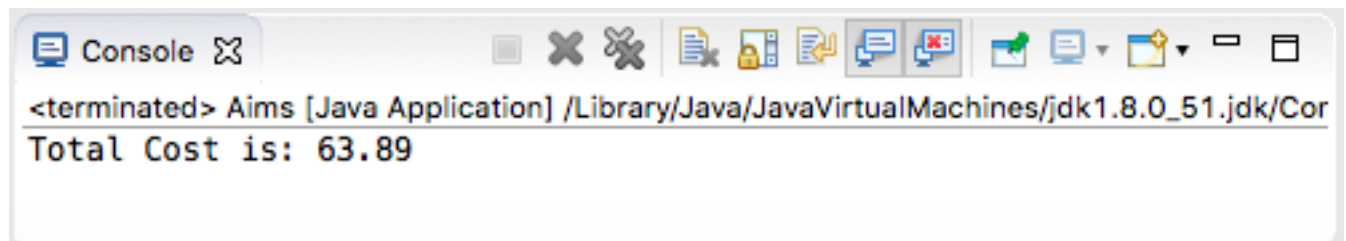*Figure 13. Sample code of the Aims class*

The result should be:

Figure 14. Results for creating a cart of digital video discs.

9.   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.

10.   Working with method overloading

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

10.1. Overloading by differing types of parameters

- **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?

10.2. 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)

11. Release flow demonstration

11.1. Hypothesis

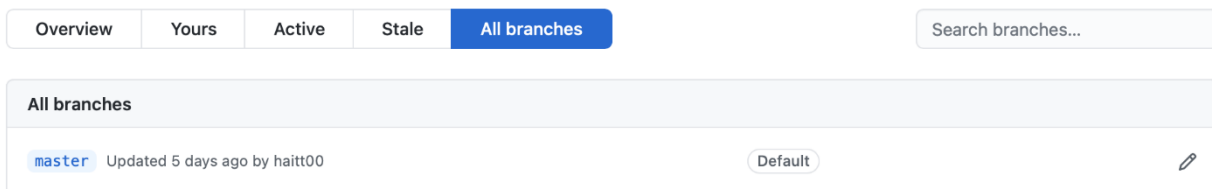We hypothesize that Figure 4 shows the branches of our current remote repository.

Figure 4. 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.

11.2. 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)
```

Step 3. Make modifications in the local repository.

Step 4. Commit the change in the local repository.
(feature/demonstrate-release-flow)

Step 5. Create a new branch in the remote repository (GitHub through GUI).
-   Firstly, under the "Code" tab of the top navigation bar, choose the drop-down button with the branch name (in this case "master") on the top left.
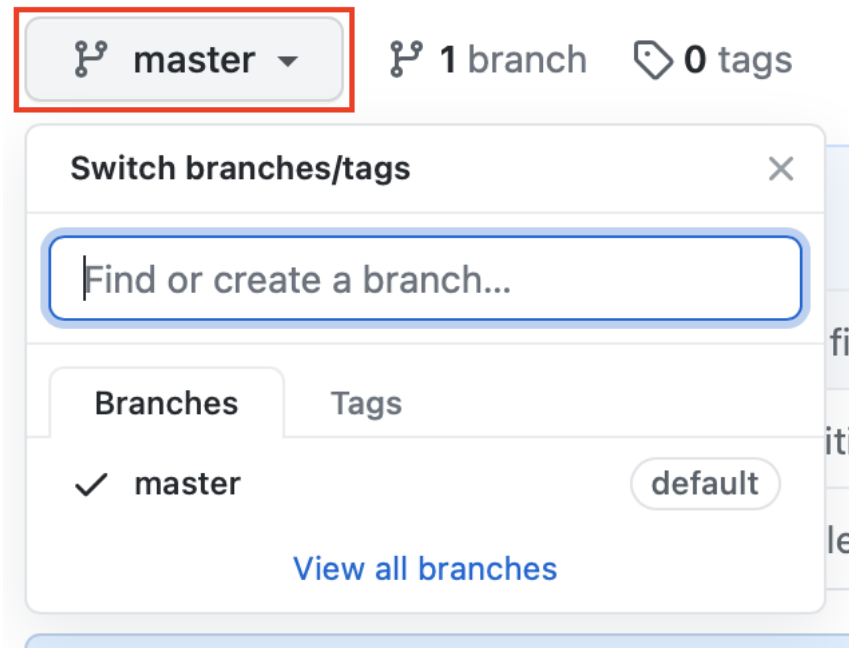

Figure 5. Branch Creation in GitHub GUI (1/3)

-   Secondly, enter the new branch name "feature/demonstrate-release-flow" into the text field and click "Create branch: feature/demonstrate-release-flow from 'master'".
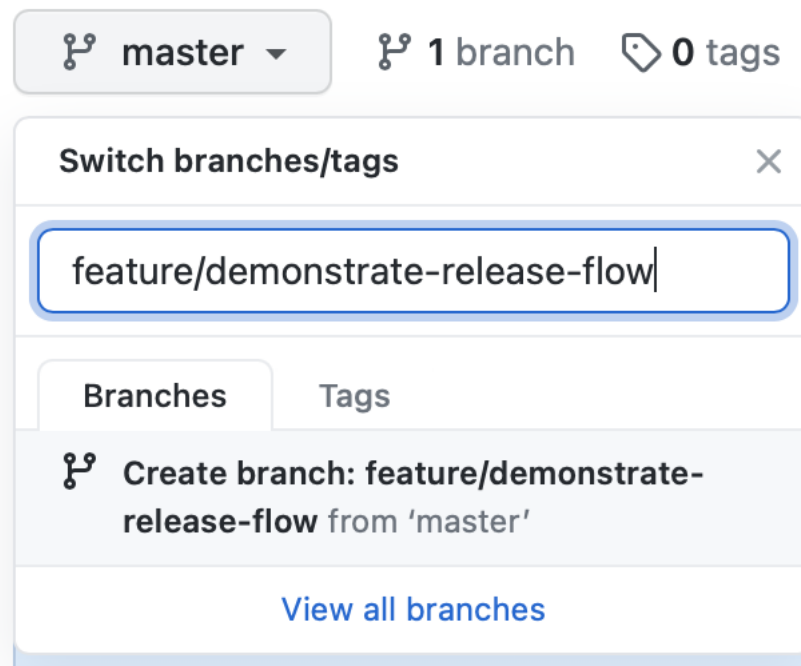

Figure 6. Branch Creation in GitHub GUI (2/3)

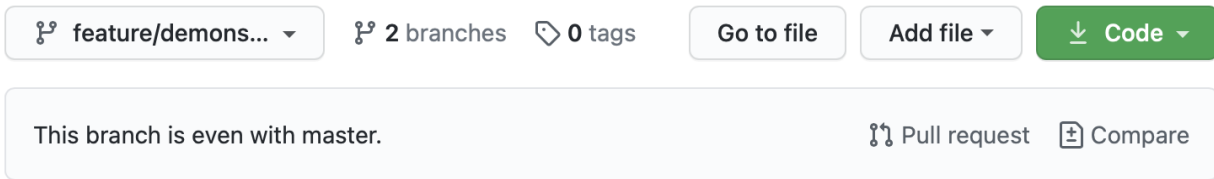- The following figure shows the result of our efforts in this step.

Figure 7. Branch Creation in GitHub GUI (3/3)

Step 6. Push the local branch to the remote branch (feature/demonstrate-release-flow)

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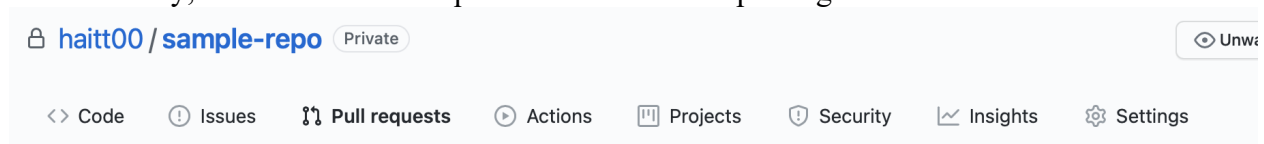- Firstly, choose the "Pull requests" tab from the top navigation bar.

Figure 8. 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.
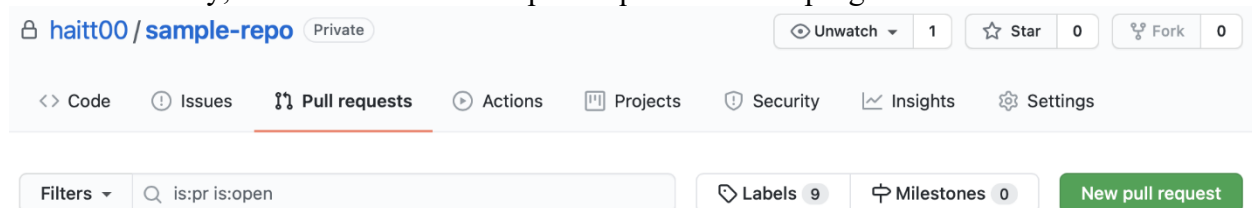

Figure 9. Creation of a Pull Request in GitHub GUI (2/4)

- 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 branches to be merged in the next step.

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base: master ▾   ←   compare: feature/demonstrate-release... ▾   ✓ **Able to merge.** These branches can be automatically merged.
Target branch   **Current branch**

Discuss and review the changes in this comparison with others. Learn about pull requests   **Create pull request**

-○- **1** commit   ⊞ **1** file changed   ▢ **0** comments   ⋈ **1** contributor

Commits on Mar 24, 2021

-○-   Add a feature for demonstration   Verified   0a466f9

⊞ Showing **1 changed file** with **1 addition** and **0 deletions**.   Unified   Split

- *Figure 10. 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"