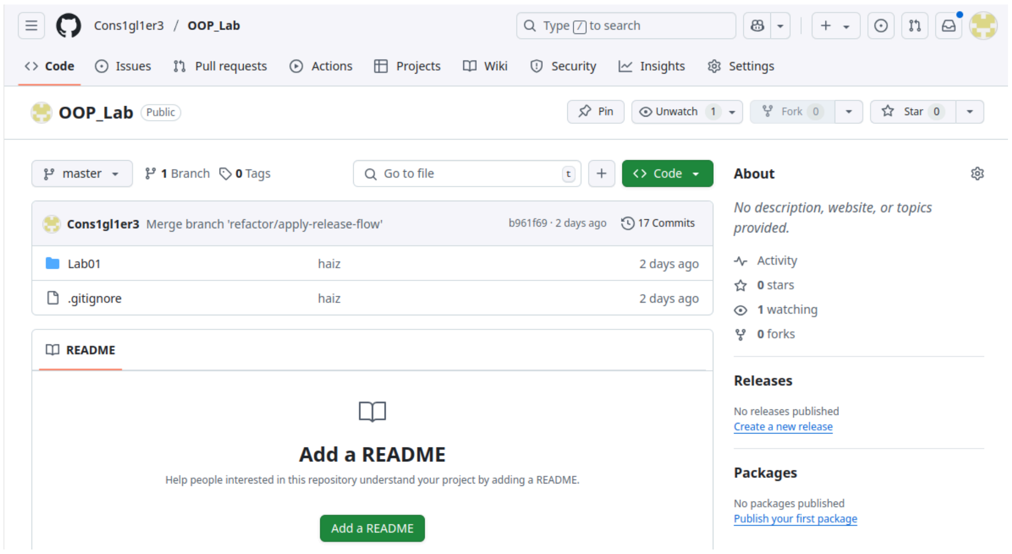
Object-Oriented Programming

Lab 02: Problem Modeling and Encapsulation

Tran Viet Anh - 20226013

1. Branch your repository



Cons1gl1er3 Merge branch 'refactor/apply-release-flow'		b961f69 · 2 days ago · 17 Commits
Lab01	haiz	2 days ago
.gitignore	haiz	2 days ago

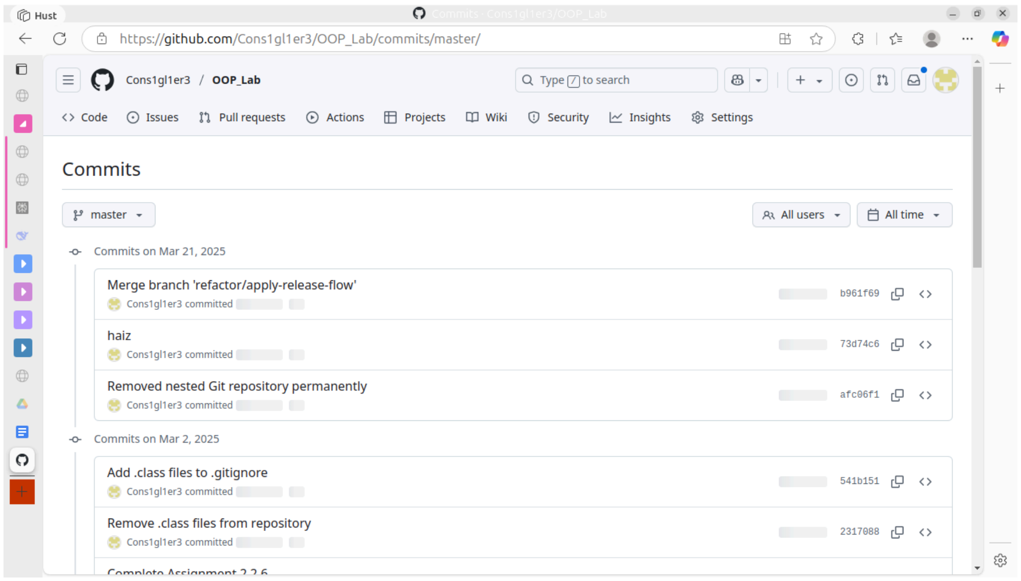
Help people interested in this repository understand your project by adding a README.

No description, website, or topics provided.

No releases published

No packages published

2. Release flow demonstration



Commits

Commits on Mar 21, 2025

Merge branch 'refactor/apply-release-flow' — Cons1gl1er3 committed	b961f69
haiz — Cons1gl1er3 committed	73d74c6
Removed nested Git repository permanently — Cons1gl1er3 committed	afc06f1

Commits on Mar 2, 2025

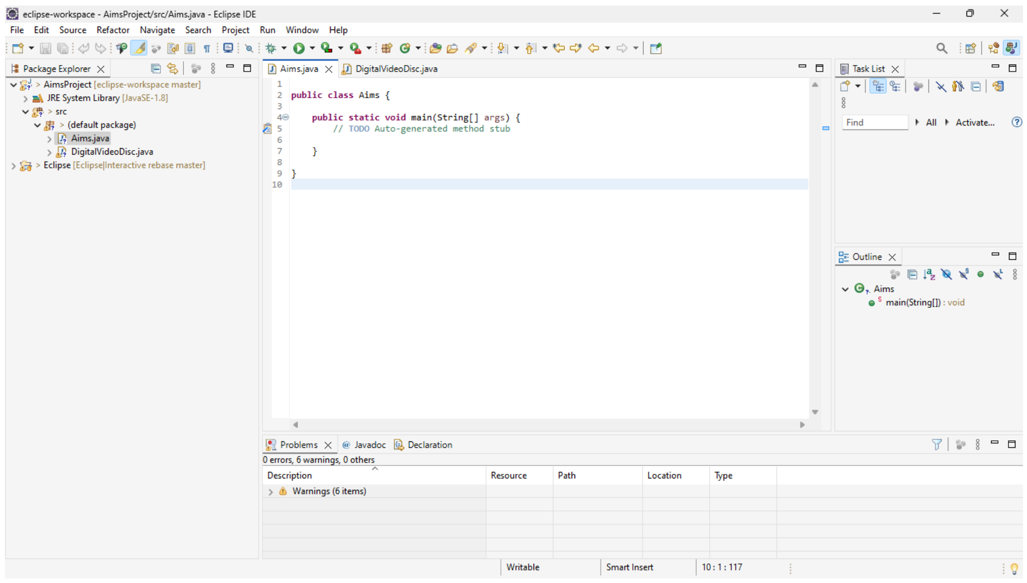
Add .class files to .gitignore — Cons1gl1er3 committed	541b151
Remove .class files from repository — Cons1gl1er3 committed	2317088
Complete Assignment 2.2.6	

3. UML & Astah

4. Problem Statement of AIMS Project
5. Use case diagram

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

7. Create Aims class

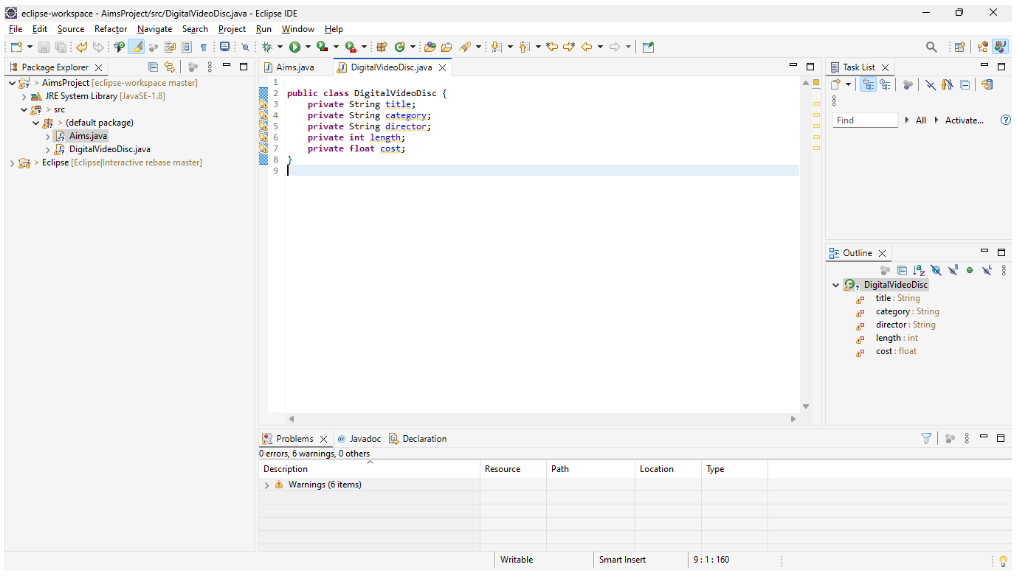

eclipse-workspace - AimsProject/src/Aims.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
AimsProject [eclipse-workspace master]
JRE System Library [JavaSE-1.8]
src
(default package)
Aims.java
DigitalVideoDisc.java
Eclipse [Eclipse|Interactive rebase master]
Aims.java
DigitalVideoDisc.java
Task List
Find
All
Activate...
Outline
Aims
main(String[]) : void
Problems
Javadoc
Declaration
Writable
Smart Insert
10 : 1 : 117

```java

public class Aims {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

```
0 errors, 6 warnings, 0 others
```

Description	Resource	Path	Location	Type
Warnings (6 items)				

8. Create the DigitalVideoDisc class and its attributes



eclipse-workspace - AimsProject/src/DigitalVideoDisc.java - Eclipse IDE

```java

public class DigitalVideoDisc {
    private String title;
    private String category;
    private String director;
    private int length;
    private float cost;
}

```

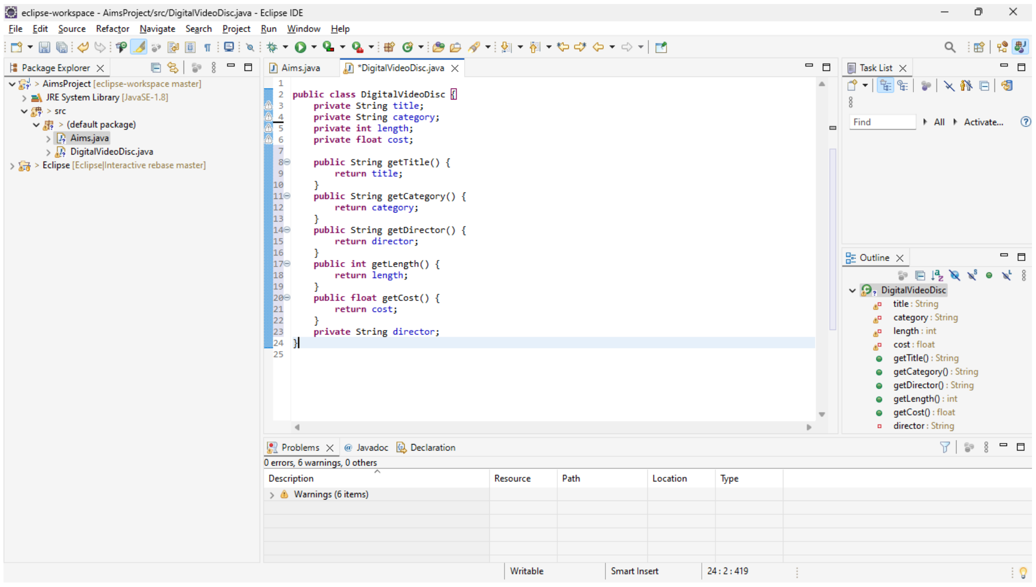
Outline: DigitalVideoDisc — title : String, category : String, director : String, length : int, cost : float

0 errors, 6 warnings, 0 others

Warnings (6 items)

Writable | Smart Insert | 9 : 1 : 160

9. Create the DigitalVideoDisc class and its attributes



eclipse-workspace - AimsProject/src/DigitalVideoDisc.java - Eclipse IDE

```java

public class DigitalVideoDisc {
    private String title;
    private String category;
    private int length;
    private float cost;

    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;
    }
    private String director;
}

```

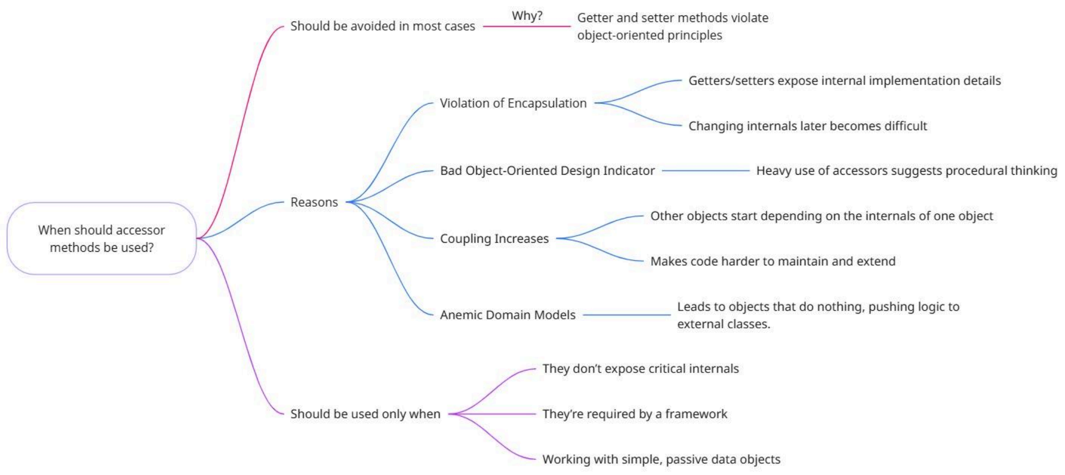
Outline: DigitalVideoDisc — title : String, category : String, length : int, cost : float, getTitle() : String, getCategory() : String, getDirector() : String, getLength() : int, getCost() : float, director : String

0 errors, 6 warnings, 0 others

Warnings (6 items)

Writable | Smart Insert | 24 : 2 : 419

<u>Reading Assignment</u>: When should accessor methods be used?


When should accessor methods be used?
Should be avoided in most cases
Why?
Getter and setter methods violate object-oriented principles
Reasons
Violation of Encapsulation
Getters/setters expose internal implementation details
Changing internals later becomes difficult
Bad Object-Oriented Design Indicator
Heavy use of accessors suggests procedural thinking
Coupling Increases
Other objects start depending on the internals of one object
Makes code harder to maintain and extend
Anemic Domain Models
Leads to objects that do nothing, pushing logic to external classes.
Should be used only when
They don't expose critical internals
They're required by a framework
Working with simple, passive data objects

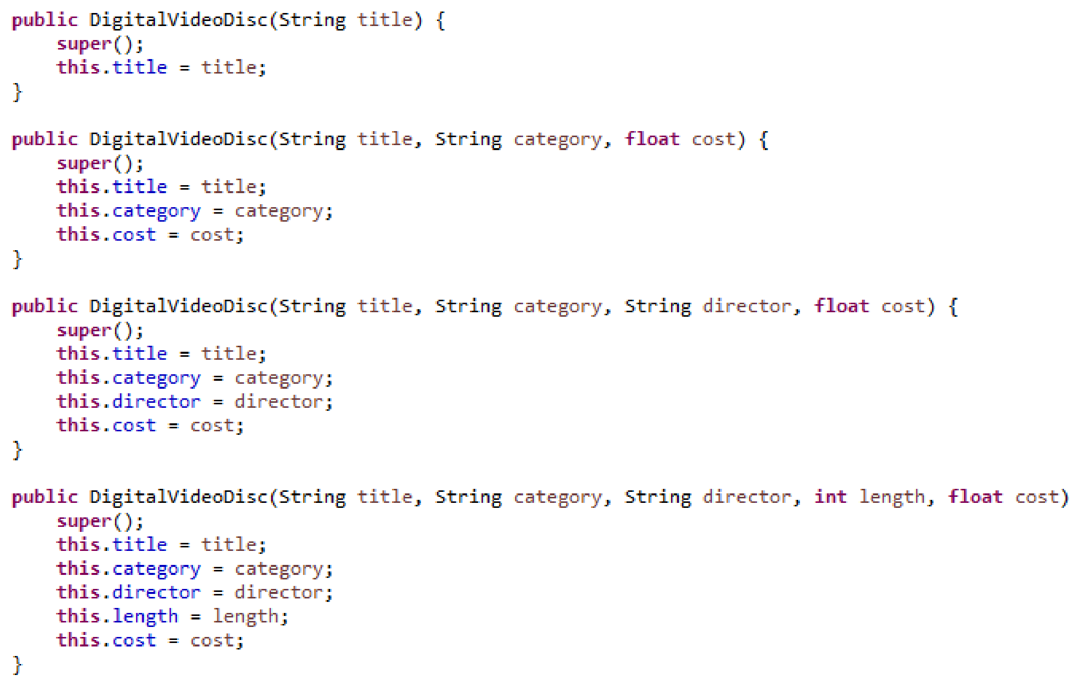
10. Create Constructor method

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?

Answer: No! In Java, although constructor overloading is allowed, each constructor must have a unique signature. That is, the number and types of the parameters must differ.

- If we try to create constructors with the same parameter types, Java will not be able to distinguish between them and will throw an error.
- Java does not allow constructors with the same parameter types unless they differ by number or type of parameters.

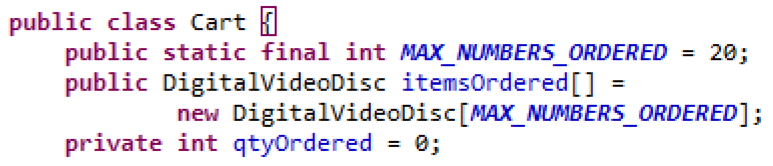
In this part, you will create yourself constructor method for DigitalVideoDisc for different purposes:



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

public DigitalVideoDisc(String title, String category, float cost) {
    super();
    this.title = title;
    this.category = category;
    this.cost = cost;
}

public DigitalVideoDisc(String title, String category, String director, float cost) {
    super();
    this.title = title;
    this.category = category;
    this.director = director;
    this.cost = cost;
}

public DigitalVideoDisc(String title, String category, String director, int length, float cost)
    super();
    this.title = title;
    this.category = category;
    this.director = director;
    this.length = length;
    this.cost = cost;
}
```

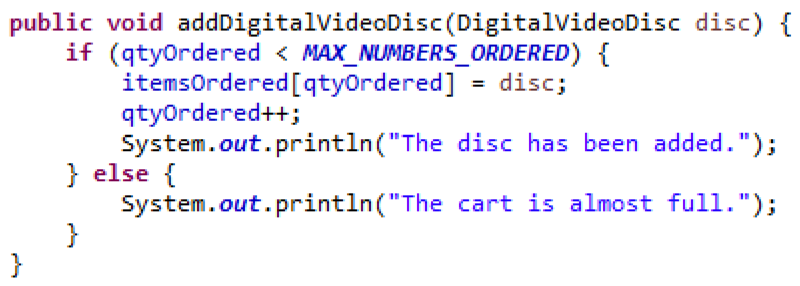
11. Create the Cart class to work with DigitalVideoDisc

a. Create a field name qtyOrdered in the Cart class which stores this information:



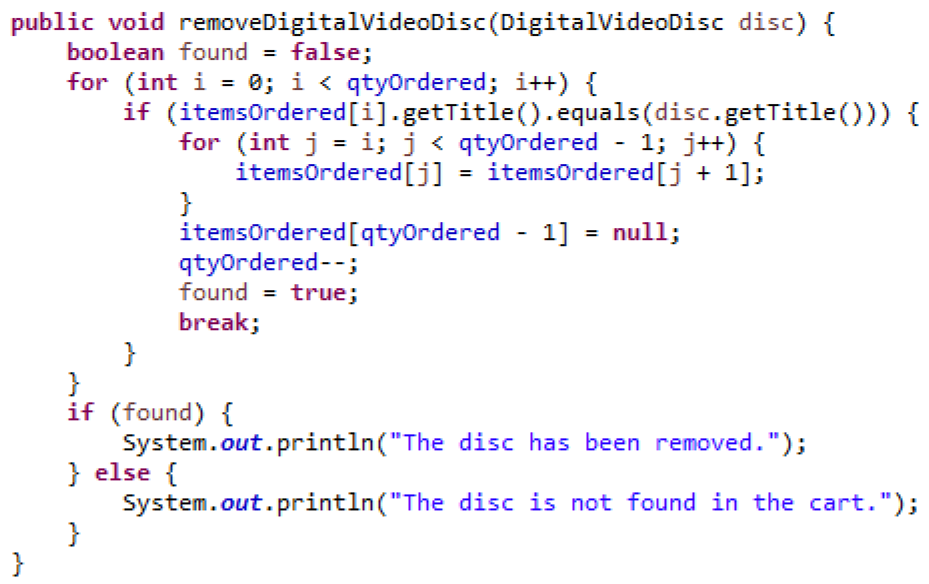
```
public class Cart {
    public static final int MAX_NUMBERS_ORDERED = 20;
    public DigitalVideoDisc itemsOrdered[] =
            new DigitalVideoDisc[MAX_NUMBERS_ORDERED];
    private int qtyOrdered = 0;
```

b. 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:



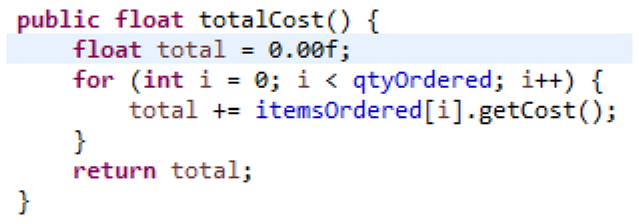
```
public void addDigitalVideoDisc(DigitalVideoDisc disc) {
    if (qtyOrdered < MAX_NUMBERS_ORDERED) {
        itemsOrdered[qtyOrdered] = disc;
        qtyOrdered++;
        System.out.println("The disc has been added.");
    } else {
        System.out.println("The cart is almost full.");
    }
}
```

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



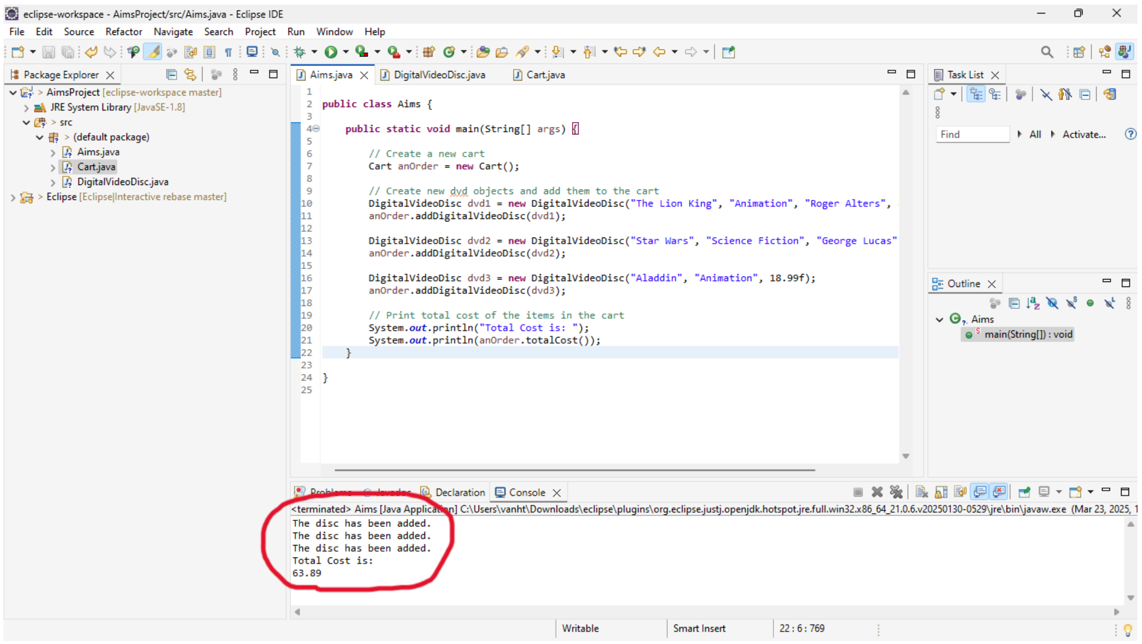
```java
public void removeDigitalVideoDisc(DigitalVideoDisc disc) {
    boolean found = false;
    for (int i = 0; i < qtyOrdered; i++) {
        if (itemsOrdered[i].getTitle().equals(disc.getTitle())) {
            for (int j = i; j < qtyOrdered - 1; j++) {
                itemsOrdered[j] = itemsOrdered[j + 1];
            }
            itemsOrdered[qtyOrdered - 1] = null;
            qtyOrdered--;
            found = true;
            break;
        }
    }
    if (found) {
        System.out.println("The disc has been removed.");
    } else {
        System.out.println("The disc is not found in the cart.");
    }
}
```

d. 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.



```java
public float totalCost() {
    float total = 0.00f;
    for (int i = 0; i < qtyOrdered; i++) {
        total += itemsOrdered[i].getCost();
    }
    return total;
}
```

12. Create Carts of DigitalVideoDiscs



```java

public class Aims {

    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 Alters",
        anOrder.addDigitalVideoDisc(dvd1);

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

        DigitalVideoDisc dvd3 = new DigitalVideoDisc("Aladdin", "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());
    }

}
```

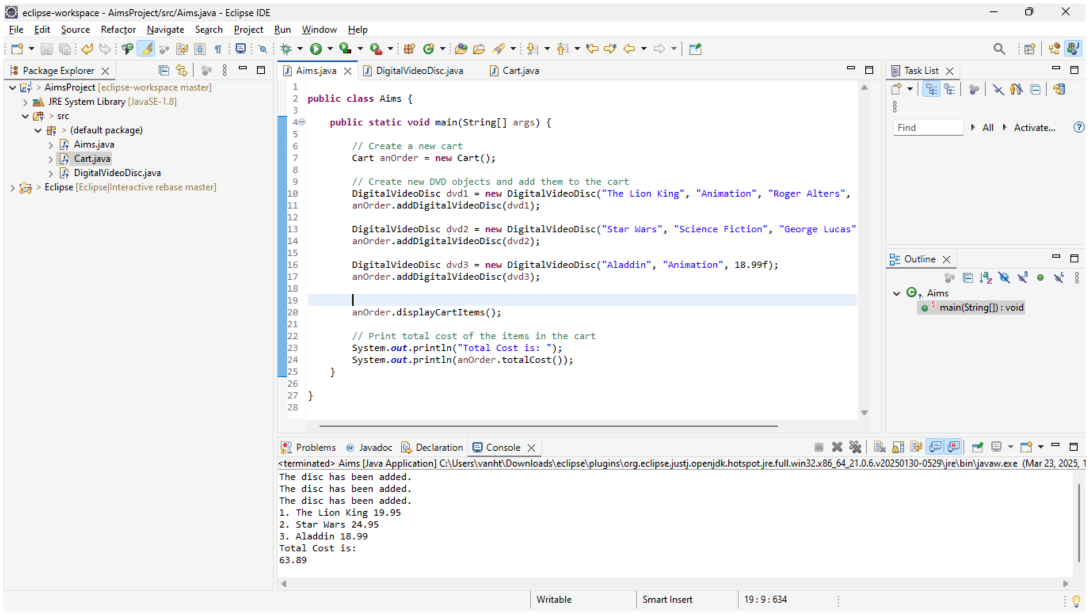
```
<terminated> Aims [Java Application] C:\Users\vanht\Downloads\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe (Mar 23, 2025, 1
The disc has been added.
The disc has been added.
The disc has been added.
Total Cost is:
63.89
```

Exercise: 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:

Answer: The following code is in class Cart:

```java
public void displayCartItems() {
    for (int i = 0; i < qtyOrdered; i++) {
        System.out.println((i + 1) + ". " + itemsOrdered[i].getTitle() +
                " " + itemsOrdered[i].getCost());
    }
}
```

Result:



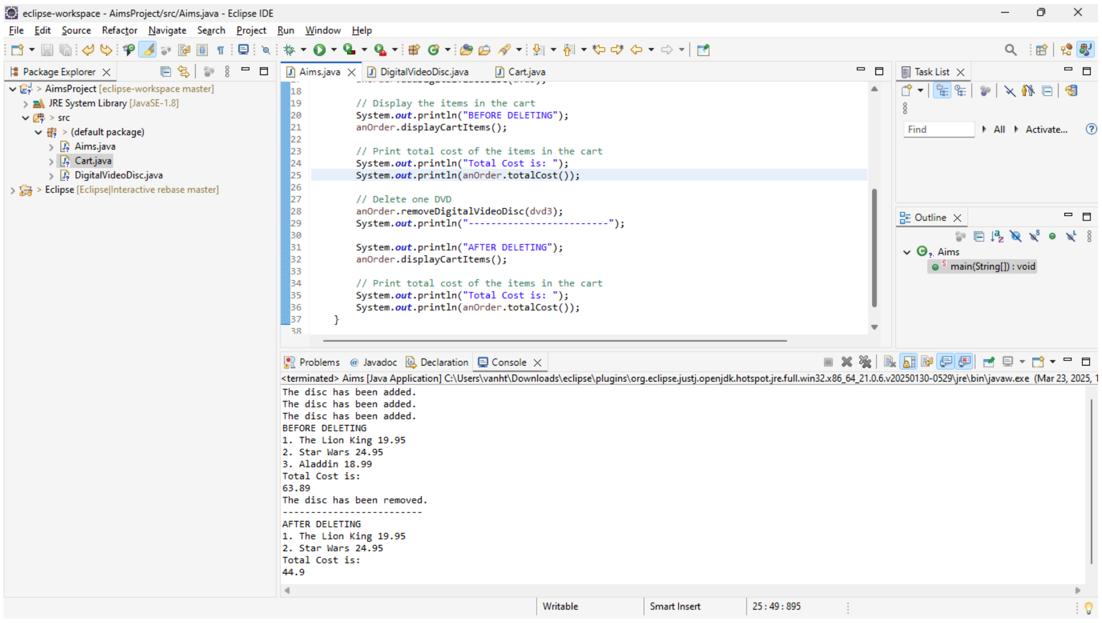
```java
public class Aims {

    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 Alters",
        anOrder.addDigitalVideoDisc(dvd1);

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

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


        anOrder.displayCartItems();

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

}
```

```
<terminated> Aims [Java Application] C:\Users\vanht\Downloads\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe (Mar 23, 2025, 1
The disc has been added.
The disc has been added.
The disc has been added.
1. The Lion King 19.95
2. Star Wars 24.95
3. Aladdin 18.99
Total Cost is:
63.89
```

13. Removing items from the cart

Exercise: 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).

Result:



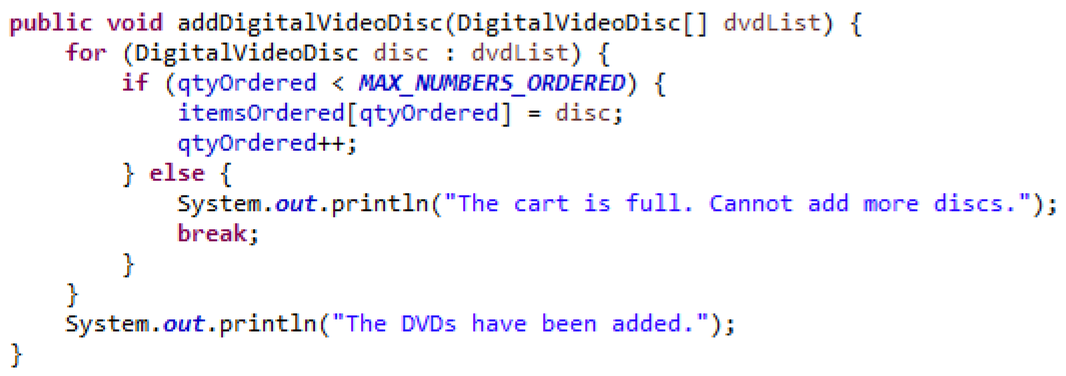
```java

        // Display the items in the cart
        System.out.println("BEFORE DELETING");
        anOrder.displayCartItems();

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

        // Delete one DVD
        anOrder.removeDigitalVideoDisc(dvd3);
        System.out.println("------------------------");

        System.out.println("AFTER DELETING");
        anOrder.displayCartItems();

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

```
<terminated> Aims [Java Application] C:\Users\vanht\Downloads\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe (Mar 23, 2025, 1
The disc has been added.
The disc has been added.
The disc has been added.
BEFORE DELETING
1. The Lion King 19.95
2. Star Wars 24.95
3. Aladdin 18.99
Total Cost is:
63.89
The disc has been removed.
------------------------
AFTER DELETING
1. The Lion King 19.95
2. Star Wars 24.95
Total Cost is:
44.9
```

14. Working with method overloading

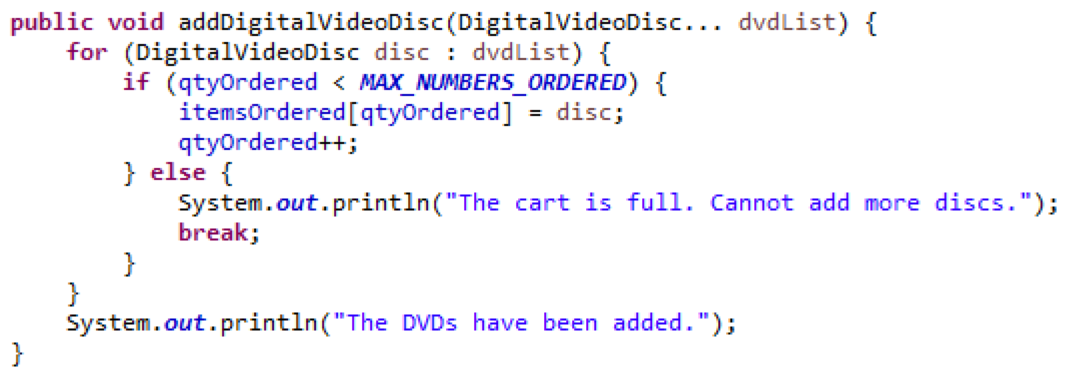
14.1 Overloading by differing types of parameter

a. 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.



```
public void addDigitalVideoDisc(DigitalVideoDisc[] dvdList) {
    for (DigitalVideoDisc disc : dvdList) {
        if (qtyOrdered < MAX_NUMBERS_ORDERED) {
            itemsOrdered[qtyOrdered] = disc;
            qtyOrdered++;
        } else {
            System.out.println("The cart is full. Cannot add more discs.");
            break;
        }
    }
    System.out.println("The DVDs have been added.");
}
```

b. Try to add a method addDigitalVideoDisc which allows to pass an arbitrary number of arguments for dvd.



```
public void addDigitalVideoDisc(DigitalVideoDisc... dvdList) {
    for (DigitalVideoDisc disc : dvdList) {
        if (qtyOrdered < MAX_NUMBERS_ORDERED) {
            itemsOrdered[qtyOrdered] = disc;
            qtyOrdered++;
        } else {
            System.out.println("The cart is full. Cannot add more discs.");
            break;
        }
    }
    System.out.println("The DVDs have been added.");
}
```

Prefer: Pass an arbitrary number of arguments.

14.2 Overloading by differing the number of parameters

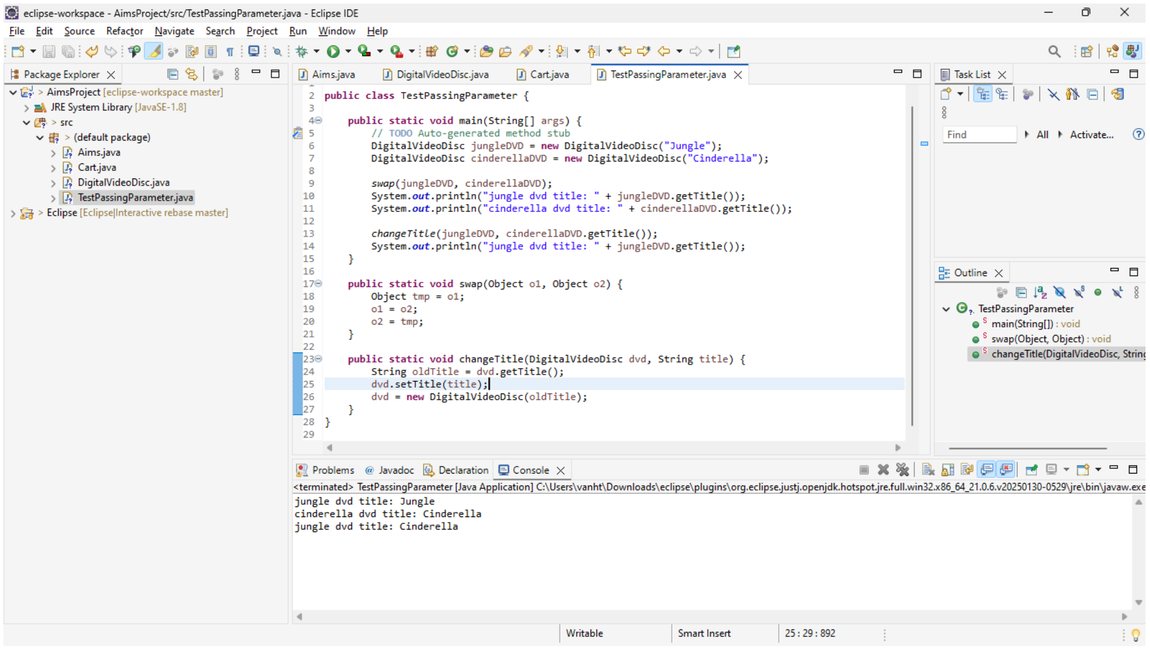
a. Create a new method named addDigitalVideoDisc. The signature of this method has two parameters as following:
addDigitalVideoDisc(DigitalVideoDisc dvd1,DigitalVideoDisc dvd2)

```java
public void addDigitalVideoDisc(DigitalVideoDisc dvd1, DigitalVideoDisc dvd2) {
    if (qtyOrdered < MAX_NUMBERS_ORDERED) {
        itemsOrdered[qtyOrdered] = dvd1;
        qtyOrdered++;
    } else {
        System.out.println("The cart is full. Cannot add more discs.");
        return;
    }

    if (qtyOrdered < MAX_NUMBERS_ORDERED) {
        itemsOrdered[qtyOrdered] = dvd2;
        qtyOrdered++;
    } else {
        System.out.println("The cart is full. Cannot add more discs.");
    }

    System.out.println("The DVDs have been added.");
}
```

15. Passing parameter

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

Answer: Java is passed by value, which means when passing a variable to a method, the method receives a copy of the value of that variable, not the original one.



```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);
    }
}
```

```
<terminated> TestPassingParameter [Java Application] C:\Users\vanht\Downloads\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe
jungle dvd title: Jungle
cinderella dvd title: Cinderella
jungle dvd title: Cinderella
```

Question: After the call of swap(jungleDVD, cinderellaDVD) why does the title of these two objects still remain?

Answer: The swap method only swaps the local references o1 and o2 within the method. Since Java passes object references by value, the original references (jungleDVD and cinderellaDVD) outside the method are not affected, and their titles remain unchanged.

Question: After the call of changeTitle(jungleDVD, cinderellaDVD.getTitle()) why is the title of the JungleDVD changed?

Answer: The changeTitle method modifies the jungleDVD object directly by changing its title to the one passed from cinderellaDVD. Since jungleDVD is passed by reference, the original jungleDVD object is modified within the method.

16. Classifier Member and Instance Member

Check out the source code.