

Student name: Lê Ngọc Anh Vũ

Student ID: 20236014

REPORT LAB 03

4. Update the **Cart** class and **CartTest** class

Write new methods to implement the following functions:

- Create a new method to print the list of ordered items of a cart, the price of each item, and the total price. 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?

```
92*   public void print() {
93       System.out.println("*****CARD*****");
94       System.out.println("Ordered Items:");
95       for(int i = 0; i < qtyOrdered; i++) {
96           System.out.printf("%d. ", i+1);
97           System.out.println(itemsOrdered[i]);
98       }
99       System.out.printf("Total cost: %f\n", totalCost());
00       System.out.println("*****");
01   }
```

toString() method for the **DigitalVideoDisc** class:

```
21*   @Override
22   public String toString() {
23       return "DVD " + getTitle() + " - " + getCategory() + " - "
24           + getDirector() + " - " + getLength() + ": " + getCost() + "$";
25   }
```

- Search for DVDs in the cart by ID and display the search results. Make sure to notify the user if no match is found.

```
103*   public void searchByCartID(int id) {
104       for(int i = 0; i < qtyOrdered; i++) {
105           if(itemsOrdered[i].getId() == id) {
106               System.out.printf("%d - ", itemsOrdered[i].getId());
107               System.out.println(itemsOrdered[i]);
108               return;
109           }
110       }
111       System.out.printf("No match is found for this ID = %d\n", id);
112   }
```

- Search for DVDs in the cart by title and print the results. Make sure to notify the user if no match is found. *Refer to the problem statement in Lab02 for the matching rule(Section 4).*

Suggestion: write a `boolean isMatch(String title)` method in the `DigitalVideoDisc` which finds out if the corresponding disk is a match given the title.

A Boolean `isMatch(String title)` method in the `DigitalVideoDisc` class:

```

27e  public boolean isMatch(String title) {
28      return getTitle().equals(title);
29  }
30

```

The method to search for DVDs in the cart by title and print results:

```

114e public void searchByTitle(String title) {
115     for(int i = 0; i < qtyOrdered; i++) {
116         if(itemsOrdered[i].isMatch(title)) {
117             System.out.printf("%d - DVD - %s - %s - %s - %d: %f$\n",
118                             itemsOrdered[i].getId(),
119                             itemsOrdered[i].getTitle(),
120                             itemsOrdered[i].getCategory(),
121                             itemsOrdered[i].getDirector(),
122                             itemsOrdered[i].getLength(),
123                             itemsOrdered[i].getCost());
124         }
125     }
126     System.out.printf("No mathch is found for this title \'%s\'\n", title);
127 }
128 }
129 }
```

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

The source code is below:

```

6 public class CartTest {
7     public static void main(String[] args) {
8         //Create a new cart
9         Cart cart = new Cart();
10        //Create new dvd objects and add them to the cart
11        DigitalVideoDisc dvd1 = new DigitalVideoDisc("The Lion King",
12            "Animation", "Roger Allers", 87, 19.95f);
13        cart.addDigitalVideoDisc(dvd1);
14
15        DigitalVideoDisc dvd2 = new DigitalVideoDisc("Star Wars",
16            "Science Fiction", "George Lucas", 87, 24.95f);
17        cart.addDigitalVideoDisc(dvd2);
18
19        DigitalVideoDisc dvd3 = new DigitalVideoDisc("Aladin",
20            "Animation", 18.99f);
21        cart.addDigitalVideoDisc(dvd3);
22
23        //Test the print method
24        cart.print();
25        //To-do: Test the search methods here
26        // Search dvd by id that has a match
27        cart.searchByCartID(2);
28        // Search dvd by id that has no match
29        cart.searchByCartID(5);
30        // Search dvd by title that has a match
31        cart.searchByTitle("Aladin");
32        // Search dvd by title that has no match
33        cart.searchByTitle("Captain America");
34    }

```

The result:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with packages like `hust.soict.globalict.aims`, `hust.soict.globalict.aims.cart`, `hust.soict.globalict.aims.media`, and source files `Book.java`, `CompactDisc.java`, `DigitalVideoDisc.java`, `Disc.java`, `Media.java`, `Playable.java`, and `Track.java`.
- CartTest.java:** The code for the test class is visible in the editor.
- Console Output:**

```

The disc The Lion King has been added
The disc Star Wars has been added
The disc Aladin has been added
*****CARD*****
Ordered Items:
1. DVD The Lion King - Animation - Roger Allers - 87: 19.95$
2. DVD Star Wars - Science Fiction - George Lucas - 87: 24.95$
3. DVD Aladin - Animation - null - 0: 18.99$ 
Total cost: 63.889999
*****
```
- Bottom Status Bar:** Shows the date and time (7:21 PM, 4/18/2025), battery level (30°C), and network status.

5. Implement the **Store** class

- Create a **Store** class inside the package: **hust.soict.dsai.aims.store**, 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**

```
1 package hust.soict.globalict.aims.store;
2
3 import hust.soict.globalict.aims.media.DigitalVideoDisc;
4
5 public class Store {
6
7     public static final int MAX_NUMBERS_ORDERED = 100;
8     private DigitalVideoDisc itemsInStore[] = new DigitalVideoDisc[MAX_NUMBERS_ORDERED];
9     private int numDVDs = 0;
10
11     public void addDVD(DigitalVideoDisc dvd) {
12         if(numDVDs < MAX_NUMBERS_ORDERED) {
13             itemsInStore[numDVDs] = dvd;
14             numDVDs++;
15             System.out.println("The disc " + dvd.getTitle() + " has been added in store");
16         }
17         else {
18             System.out.println("The store is almost full");
19         }
20     }
21
22     public void removeDVD(DigitalVideoDisc dvd) {
23         int idx;
24         for(idx = 0; idx < numDVDs; idx++) {
25             if(dvd == itemsInStore[idx]) {
26                 break;
27             }
28         }
29
30         if(idx == numDVDs) {
31             System.out.println("Cannot find disc " + dvd.getTitle() + " in the store");
32             return;
33         }
34
35         for(int i = idx; i < numDVDs - 1; i++) {
36             itemsInStore[i] = itemsInStore[i+1];
37         }
38         numDVDs--;
39         System.out.println("The disc " + dvd.getTitle() + " has been removed from store");
40     }
41 }
```

- Test these two methods in the **StoreTest** class.

As a result, you should have two classes: the **Store** class and the **StoreTest** class inside the **hust.soict.dsai.aims.store** package.

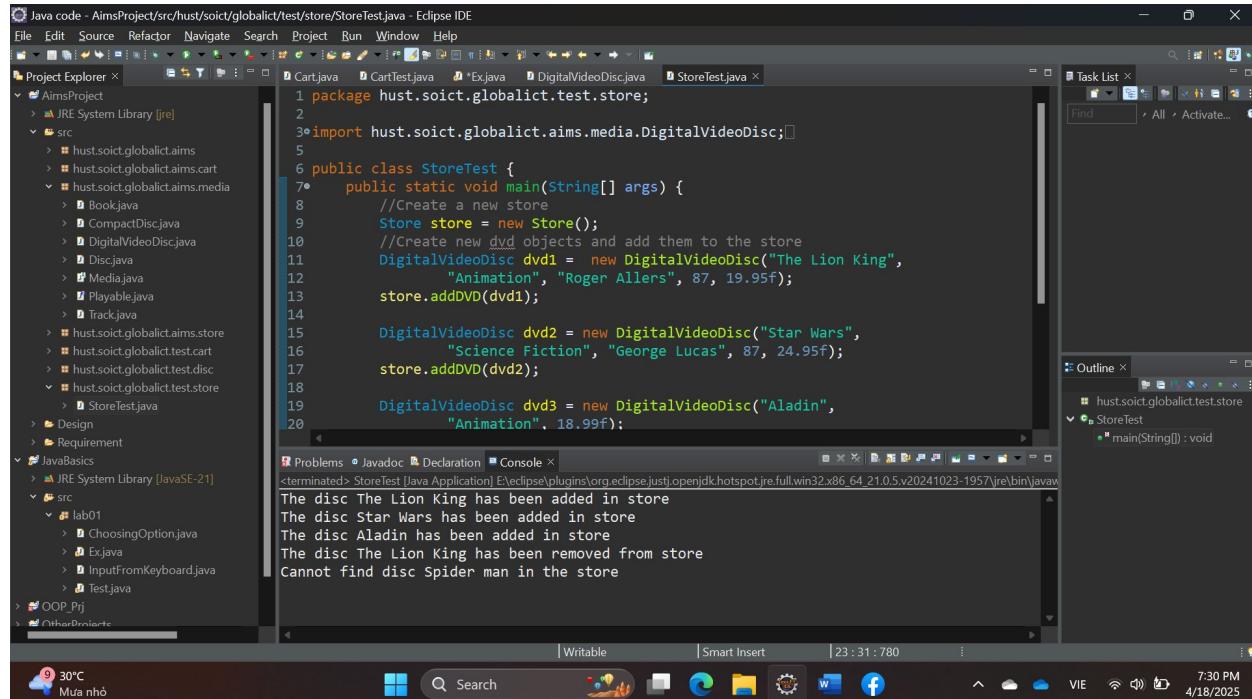
The source code is below:

```

1 package hust.soict.globalict.test.store;
2
3 import hust.soict.globalict.aims.media.DigitalVideoDisc;
4
5
6 public class StoreTest {
7     public static void main(String[] args) {
8         //Create a new store
9         Store store = new Store();
10        //Create new dvd objects and add them to the store
11        DigitalVideoDisc dvd1 = new DigitalVideoDisc("The Lion King",
12            "Animation", "Roger Allers", 87, 19.95f);
13        store.addDVD(dvd1);
14
15        DigitalVideoDisc dvd2 = new DigitalVideoDisc("Star Wars",
16            "Science Fiction", "George Lucas", 87, 24.95f);
17        store.addDVD(dvd2);
18
19        DigitalVideoDisc dvd3 = new DigitalVideoDisc("Aladin",
20            "Animation", 18.99f);
21        store.addDVD(dvd3);
22        //Remove dvd objects from store
23        store.removeDVD(dvd1);
24        //Remove dvd objects that are not available in the store
25        DigitalVideoDisc dvd4 = new DigitalVideoDisc("Spider man",
26            "Science Fiction", 30.07f);
27        store.removeDVD(dvd4);
28    }
29}

```

The result is:



6. String, StringBuilder and StringBuffer

- In the **OtherProjects** project, create a new package **hust.soict.dsai.garbage**. 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**.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with the **ConcatenationInLoops.java** file selected.
- Code Editor:** Displays the Java code for **ConcatenationInLoops**. The code compares the performance of concatenating strings using the **+** operator versus using **StringBuffer** and **StringBuilder**.
- Console:** Shows the output of the program, which prints the time difference between the start and end of the loop execution.
- System Bar:** Shows the date and time as 10:59 PM on 4/19/2025.

```
Java code - OtherProjects/src/hust/soict/globalict/garbage/ConcatenationInLoops.java - Eclipse IDE
File Edit Source Refactor Navigate Project Run Window Help
Project Explorer
  AimsProject
    JRE System Library [jre]
      src
        hust.soict.globalict.aims
        hust.soict.globalict.aims.cart
        hust.soict.globalict.aims.media
        hust.soict.globalict.aims.store
        hust.soict.globalict.test.cart
        hust.soict.globalict.test.disc
        hust.soict.globalict.test.store
      Design
      Requirement
    JavaBasics
    OOP_Pj
  OtherProjects
    JRE System Library [jre]
      src
        hust.soict.globalict.garbage
          ConcatenationInLoops.java
          GarbageCreator.java
          test.txt
    pikachu
ConcatenationInLoops.java
1 package hust.soict.globalict.garbage;
2
3 import java.util.Random;
4
5 public class ConcatenationInLoops {
6     public static void main(String[] args) {
7         Random r = new Random(123);
8         long start = System.currentTimeMillis();
9         String s = "";
10        for(int i = 0; i < 65536; i++) {
11            s += r.nextInt(2);
12        }
13        System.out.println(System.currentTimeMillis() - start); // This prints roughly 4500.
14
15        r = new Random(123);
16        start = System.currentTimeMillis();
17        StringBuilder sb = new StringBuilder();
18        for(int i = 0; i < 65536; i++) {
19            sb.append(r.nextInt(2));
20        }
21        s = sb.toString();
22        System.out.println(System.currentTimeMillis() - start); // This prints 5.
}
Problems Declaration Console
<terminated> ConcatenationInLoops [Java Application] E:\eclipse\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe
515
3
| Writable | Smart Insert | 1:1:0 |
10:59 PM
4/19/2025
```

- 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.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with the **GarbageCreator.java** file selected.
- Code Editor:** Displays the Java code for **GarbageCreator**, which reads all bytes from a file and concatenates them into a string.
- Console:** Shows the output of the program, which prints the time difference between the start and end of the loop execution.
- System Bar:** Shows the date and time as 11:16 PM on 4/19/2025.

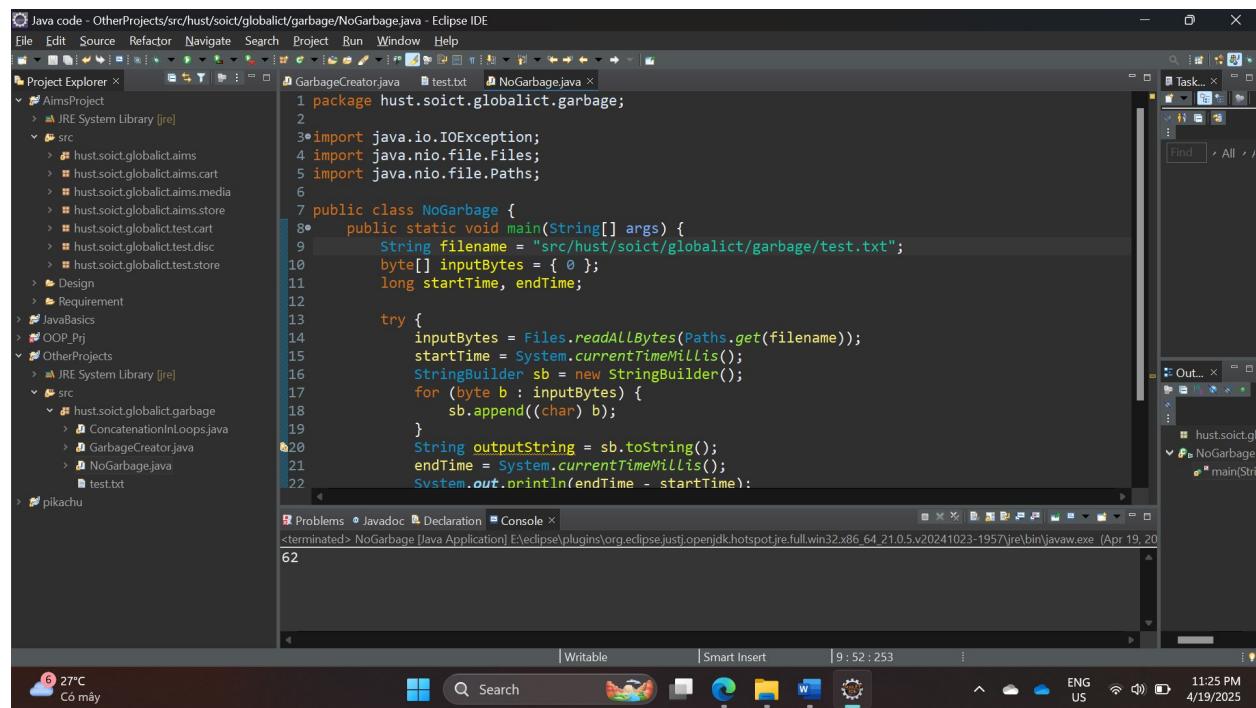
```
Java code - OtherProjects/src/hust/soict/globalict/garbage/GarbageCreator.java - Eclipse IDE
File Edit Source Refactor Navigate Project Run Window Help
Project Explorer
  AimsProject
    JRE System Library [jre]
      src
        hust.soict.globalict.aims
        hust.soict.globalict.aims.cart
        hust.soict.globalict.aims.media
        hust.soict.globalict.aims.store
        hust.soict.globalict.test.cart
        hust.soict.globalict.test.disc
        hust.soict.globalict.test.store
      Design
      Requirement
    JavaBasics
    OOP_Pj
  OtherProjects
    JRE System Library [jre]
      src
        hust.soict.globalict.garbage
          ConcatenationInLoops.java
          GarbageCreator.java
          test.txt
    pikachu
GarbageCreator.java
1 package hust.soict.globalict.garbage;
2
3 import java.io.IOException;
4
5 public class GarbageCreator {
6     public static void main(String[] args) throws IOException {
7         String filename = "src/hust/soict/globalict/garbage/test.txt";
8         byte[] inputBytes = { 0 };
9         long startTime, endTime;
10
11         inputBytes = Files.readAllBytes(Paths.get(filename));
12         startTime = System.currentTimeMillis();
13         String outputString = "";
14         for(byte b : inputBytes) {
15             outputString += (char)b;
16         }
17         endTime = System.currentTimeMillis();
18         System.out.println(endTime - startTime);
19     }
20 }
21
22
Problems Declaration Console
GarbageCreator [Java Application] E:\eclipse\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe (Apr 19, 2025, 11:13:45)
| Writable | Smart Insert | 1:1:0 |
11:16 PM
4/19/2025
```

In the above picture, we can see that the program must hang out in a long time and has not printed the result yet. This is because of the large amount of “garbage”.

For NoGarbage class:

```
1 package hust.soict.globalict.garbage;
2
3 import java.io.IOException;
4 import java.nio.file.Files;
5 import java.nio.file.Paths;
6
7 public class NoGarbage {
8     public static void main(String[] args) {
9         String filename = "src/hust/soict/globalict/garbage/test.txt";
10        byte[] inputBytes = { 0 };
11        long startTime, endTime;
12
13        try {
14            inputBytes = Files.readAllBytes(Paths.get(filename));
15            startTime = System.currentTimeMillis();
16            StringBuilder sb = new StringBuilder();
17            for (byte b : inputBytes) {
18                sb.append((char) b);
19            }
20            String outputString = sb.toString();
21            endTime = System.currentTimeMillis();
22            System.out.println(endTime - startTime);
23        } catch (IOException e) {
24            e.printStackTrace();
25        }
26    }
27 }
```

The result is:



8. Implementation of the **Book** class

- In the Package Explorer view, right-click the project and select New -> Class. Adhere to the following specifications:
 - o Package: **hust.soict.dsai.aims.media**
 - o Name: **Book**
 - o Access modifier: **public**
 - o Superclass: **java.lang.Object**
 - o **public static void main(String[] args): do not check**
 - o Constructors from Superclass: **Check**
 - o All other boxes: **Do not check**
 - o **Add fields to the Book class**
- To store the information about a **Book**, the class requires five fields: an **int** field **id**, **String** fields **title** and **category**, a **float** field **cost** and an **ArrayList** of **authors**. You will want to make these fields private, with public accessor methods for all but the **authors** field.
 - o Instead of typing the accessor methods for these fields, you may use the **Generate Getter and Setter** option in the **Outline** view pop-up menu (i.e., Right Click -> Source -> Generate Getters and Setters...). Note that in reality, not all attributes need to have getter and setter. We only create this when necessary. Getter and setter generator of Eclipse also let you decide which attribute will get getter or setter or both.

```
3*import java.util.ArrayList;□
5
6 public class Book{
7
8     private int id;
9     private String title;
10    private String category;
11    private float cost;
12    private List<String> authors = new ArrayList<String>();
13
14*   public Book() {
15     // TODO Auto-generated constructor stub
16   }
17
18*   public int getId() {
19     return id;
20   }
21
22*   public String getTitle() {
23     return title;
24   }
25
26*   public String getCategory() {
27     return category;
28   }
29
30*   public float getCost() {
31     return cost;
32 }
```

- Next, create **addAuthor(String authorName)** and **removeAuthor(String authorName)** for the **Book** class
- The **addAuthor(...)** method should ensure that the author is not already in the **ArrayList** before adding

```

34•    public void addAuthor(String authorName) {
35        for(int i = 0; i < authors.size(); i++) {
36            if(authors.get(i).equals(authorName)) {
37                System.out.printf("This name of author \'%s\' has already added!\n", authorName);
38                return;
39            }
40        }
41        authors.add(authorName);
42        System.out.printf("This name of author \'%s\' has added successfully!\n", authorName);
43    }

```

- The **removeAuthor(...)** method should ensure that the author is present in the **ArrayList** before removing

```

45•    public void removeAuthor(String authorName) {
46        if(authors.contains(authorName)) {
47            authors.remove(authorName);
48            System.out.printf("This name of author \'%s\' has already removed!\n", authorName);
49            return;
50        }
51        System.out.printf("This name of author \'%s\' cannot be removed!\n", authorName);
52    }

```

9. Implementation of the abstract **Media** class

At this point, the **DigitalVideoDisc** and the **Book** classes have some fields in common namely id, title, category and cost. Here is a good opportunity to create a common superclass between the two, to eliminate the duplication of code. This process is known as refactoring. You will create an abstract class called **Media** which contains these fields and their associated get and set methods.

Please follow these steps to create the **Media** class in the project:

- In the **Package Explorer** view, right click to the project and select New -> Class. Adhere to the following specifications for the new class:
 - Package: **hust.soict.dsai.aims.media**
 - Name: **Media**
 - Access Modifier: **public, abstract**
 - Superclass: **java.lang.Object**
 - Constructors from Superclass: Check
 - **public static void main (String[] args):** do not check
 - All other boxes: Do not check
- Add fields to the **Media** class
 - To store the information common to the **DigitalVideoDisc** and the **Book** classes, the **Media** class requires four private fields: **int id, String title, String category** and **float cost**
 - You will want to make public accessor methods for these fields (by using **Generate Getter and Setter** option in the **Outline** view pop-up menu)

```

1 package hust.soict.globalict.aims.media;
2
3 public abstract class Media {
4
5     private int id;
6     private String title;
7     private String category;
8     private float cost;
9
10    public Media() {
11        // TODO Auto-generated constructor stub
12    }
13
14    public int getId() {
15        return id;
16    }
17
18    public String getTitle() {
19        return title;
20    }
21
22    public String getCategory() {
23        return category;
24    }
25
26    public float getCost() {
27        return cost;
28    }
29
30    public void setId(int id) {
31        this.id = id;
32    }
33
34    public void setTitle(String title) {
35        this.title = title;
36    }
37
38    public void setCategory(String category) {
39        this.category = category;
40    }
41
42    public void setCost(float cost) {
43        this.cost = cost;
44    }
45
46 }

```

- Remove fields and methods from **Book** and **DigitalVideoDisc** classes
 - o Open the Book.java in the editor
 - o Locate the Outline view on the right-hand side
 - o Select the fields id, title, category, cost and their accessors & mutators (if exist)
 - o Right click the selection and select Delete from the pop-up menu
 - o Save your changes
- Do similarly for the **DigitalVideoDisc** class and move it to the package **hust.soict.dsai.aims.media**. Remove the package **hust.soict.dsai.aims.disc**.

After doing that you will see a lot of errors because of the missing fields

Extend the **Media** class for both **Book** and **DigitalVideoDisc**

```

public class Book extends Media
public class DigitalVideoDisc extends Media

```

Save your changes.

```
public class Book extends Media{
```

```
3 public class DigitalVideoDisc extends Media implements Playable{
```

10. Implementation of the **CompactDisc** class

As with **DigitalVideoDisc** and **Book**, the **CompactDisc** class will extend **Media**, inheriting the **id**, **title**, **category** and **cost** fields and the associated methods.

10.1. Create the **Disc** class extending the **Media** class

- The **Disc** class has two fields: **length** and **director**
- Create **getter** methods for these fields
- Create constructor(s) for this class. Use **super()** if possible.

```
1 package hust.soict.globalict.aims.media;
2
3 public class Disc extends Media{
4
5     private int length;
6     private String director;
7
8•    public Disc(String title) {
9        super();
10       setTitle(title);
11    }
12
13•   public Disc(String title, String category, float cost) {
14        super();
15        setTitle(title);
16        setCategory(category);
17        setCost(cost);
18    }
19
20•   public Disc(String title, String category, String director, float cost) {
21        super();
22        this.director = director;
23        setTitle(title);
24        setCategory(category);
25        setCost(cost);
26    }
27
28•   public Disc(String title, String category, String director, int length, float cost) {
29        super();
30        this.director = director;
31        setTitle(title);
32        setCategory(category);
33        setCost(cost);
34        this.length = length;
35    }
36
37•   public int getLength() {
38        return length;
39    }
40
41•   public String getDirector() {
42        return director;
43    }
44
45 }
```

- Make the **DigitalVideoDisc** extending the **Disc** class. Make changes if need be.

```

1 package hust.soict.globalict.aims.media;
2
3 public class DigitalVideoDisc extends Disc implements Playable{
4
5•   public DigitalVideoDisc(String title) {
6     super(title);
7   }
8
9•   public DigitalVideoDisc(String title, String category, float cost) {
10    super(title, category, cost);
11  }
12
13•  public DigitalVideoDisc(String title, String category, String director, float cost) {
14    super(title, category, director, cost);
15  }
16
17•  public DigitalVideoDisc(String title, String category, String director, int length, float cost) {
18    super(title, category, director, length, cost);
19  }
20
21• @Override
22  public String toString() {
23    return "DVD " + getTitle() + " - " + getCategory() + " - "
24      + getDirector() + " - " + getLength() + ": " + getCost() + "$";
25  }
26
27•  public boolean isMatch(String title) {
28    return getTitle().equals(title);
29  }

```

- Create the **CompactDisc** extending the **Disc** class. Save your changes.

```

1 package hust.soict.globalict.aims.media;
2
3•import java.util.ArrayList;
4
5
6 public class CompactDisc extends Disc{
7
8   private String artist;
9   private List<Track> tracks = new ArrayList<Track>();
10
11•  public CompactDisc(String title, String artist) {
12    super(title);
13    this.artist = artist;
14  }
15 }

```

*10.2. Create the **Track** class which models a track on a compact disc and will store information including the **title** and **length** of the track*

- Add two fields: **String title** and **int length**
- Make these fields **private** and create their **getter** methods as **public**
- Create constructor(s) for this class.
- Save your changes

```

1 package hust.soict.globalict.aims.media;
2
3 public class Track {
4
5   private String title;
6   private int length;
7
8•  public Track(String title, int length) {
9    this.title = title;
10   this.length = length;
11  }
12
13•  public String getTitle() {
14    return title;
15  }
16•  public int getLength() {
17    return length;
18  }
19 }

```

10.3. Update the **CompactDisc** class

- Add 2 fields to this class:
 - a **String** as **artist**
 - an **ArrayList** of **Track** as **tracks**
- Make all these fields **private**. Create a public **getter** method for only **artists**.
- Create constructor(s) for this class. Use super() if possible.

```
1 package hust.soict.globalict.aims.media;
2
3 import java.util.ArrayList;
4
5 public class CompactDisc extends Disc implements Playable{
6
7     private String artist;
8     private List<Track> tracks = new ArrayList<Track>();
9
10    public CompactDisc(String title, String artist) {
11        super(title);
12        this.artist = artist;
13    }
14
15    public CompactDisc(String title, String category, float cost, String artist) {
16        super(title, category, cost);
17        this.artist = artist;
18    }
19
20    public CompactDisc(String title, String category, String director, float cost, String artist) {
21        super(title, category, director, cost);
22        this.artist = artist;
23    }
24
25    public CompactDisc(String title, String category, String director, int length, float cost, String artist) {
26        super(title, category, director, length, cost);
27        this.artist = artist;
28    }
29
30
31    public String getArtist() {
32        return artist;
33    }
34
35
36
37
38
39
40 }
```

- Create methods **addTrack()** and **removeTrack()**
 - The **addTrack()** method should check if the input track is already in the list of tracks and inform users
 - The **removeTrack()** method should check if the input track existed in the list of tracks and inform users

```
31    public void addTrack(Track track) {
32        for(int i = 0; i < tracks.size(); i++) {
33            if(tracks.get(i).equals(track)) {
34                System.out.println("This track is already existed in the list!");
35                return;
36            }
37        }
38        tracks.add(track);
39        System.out.println("This track is already added!");
40    }
```

```

42•    public void removeTrack(Track track) {
43        if(tracks.contains(track)) {
44            tracks.remove(track);
45            System.out.println("This track is already removed!");
46        }
47        else {
48            System.out.println("This track does not exist in the list!");
49        }
50    }

```

- Create the **getLength()** method
 - Because each track in the CD has a length, the length of the CD should be the sum of lengths of all its tracks.
- Save your changes

```

52•    public int getLength() {
53        int sum = 0;
54        for(int i = 0; i < tracks.size(); i++) {
55            sum += tracks.get(i).getLength();
56        }
57        return sum;
58    }

```

11. Implementation of the Playable interface

The **Playable** interface is created to allow classes to indicate that they implement a **play()** method.

You can apply Release Flow here by creating a **topic** branch for implementing **Playable** interface.

- Create **Playable** interface, and add to it the method prototype: **public void play();**
- Save your changes

```

1 package hust.soict.globalict.aims.media;
2
3 public interface Playable {
4
5     public void play();
6 }

```

- Implement the **Playable** with **CompactDisc**, **DigitalVideoDisc** and **Track**
 - For each of these classes **CompactDisc** and **DigitalVideoDisc**, edit the class description to include the keywords **implements Playable**, after the keyword **extends Disc**
 - For the **Track** class, insert the keywords **implements Playable** after the keywords **public class Track**
- Implement **play()** for **DigitalVideoDisc** and **Track**
 - Add the method **play()** to these two classes
 - In the **DigitalVideoDisc**, simply print to screen:


```
public void play() {
    System.out.println("Playing DVD: " + this.getTitle());
    System.out.println("DVD length: " + this.getLength());
}
```
 - Similar additions with the **Track** class

For **DigitalVideoDisc** class:

```

31*     @Override
32     public void play() {
33         // TODO Auto-generated method stub
34         if(this.getLength() <= 0) {
35             System.out.println("Cannot play this DVD!");
36         }
37         else {
38             System.out.println("Playing DVD: " + this.getTitle());
39             System.out.println("DVD length: " + this.getLength());
40         }
41     }

```

For Track class:

```

20*     @Override
21     public void play() {
22         // TODO Auto-generated method stub
23         if(length <= 0) {
24             System.out.println("Cannot play this track!");
25         }
26         else{
27             System.out.println("Playing track: " + title);
28             System.out.println("Track length: " + length);
29         }
30     }

```

- Implement **play()** method for **CompactDisc**
 - o Since the **CompactDisc** class contains an **ArrayList** of **Tracks**, each of which can be played on its own. The **play()** method should output some information about the **CompactDisc** to console
 - o Loop through each track of the arraylist and call **Track's** **play()** method

```

54*     @Override
55     public void play() {
56         // TODO Auto-generated method stub
57         System.out.println(getTitle() + " - " +
58             getDirector() + " - " +
59             getCategory() + " - " +
60             artist + " - " +
61             super.getLength() + ": " +
62             getCost());
63         for(int i = 0; i < tracks.size(); i++) {
64             tracks.get(i).play();
65         }
66     }

```

12. Update the **Cart** class to work with **Media**

You must now update the **Cart** class to accept not only **DigitalVideoDisc** but also **Book** and **CompactDisc**. Currently, the **Cart** class has methods:

- `addDigitalVideoDisc()`
- `removeDigitalVideoDisc()`.

You could add more methods to add and remove `Book` and `CompactDisc`, but since `DigitalVideoDisc`, `Book` and `CompactDisc` are all subclasses of type `Media`, you can simply change `Cart` to maintain a collection of `Media` objects. Thus, you can add a `DigitalVideoDisc`, or a `Book`, or a `CompactDisc` using the same methods.

- Remove the `itemsOrdered` array, as well as its add and remove methods:
- Create `addMedia()` and `removeMedia()` to replace `addDigitalVideoDisc()` and `removeDigitalVideoDisc()`

```

1 package hust.soict.globalict.aims.cart;
2
3 import java.util.ArrayList;
4 import hust.soict.globalict.aims.media.DigitalVideoDisc;
5 import hust.soict.globalict.aims.media.Media;
6
7 public class Cart {
8
9     private ArrayList<Media> itemsOrdered = new ArrayList<Media>();
10
11    public void addMedia(Media media) {}
12
13    public void removeMedia(Media media) {}
14
15    public float totalCost() {}
16
17    public void display() {}
18
19 }

```

`addMedia()` method:

```

11    public void addMedia(Media media) {
12        if(itemsOrdered.contains(media)) {
13            System.out.println("This media is already in the cart!");
14        }
15        else {
16            itemsOrdered.add(media);
17            System.out.println("This media has been added in the cart");
18        }
19    }

```

`removeMedia()` method:

```

21    public void removeMedia(Media media) {
22        if(itemsOrdered.contains(media)) {
23            itemsOrdered.remove(media);
24            System.out.println("This media has been removed from the cart");
25        }
26        else {
27            System.out.println("This media does not exist in the cart");
28        }
29    }

```

- Update the **totalCost()** method

```

31•    public float totalCost() {
32        float sum = 0;
33        for(int i = 0; i < itemsOrdered.size(); i++) {
34            sum+=itemsOrdered.get(i).getCost();
35        }
36        return sum;
37    }

```

13. Update the **Store** class to work with **Media**

- Similar to the **Cart** class, change the **itemsInStore[]** attribute of the **Store** class to **ArrayList<Media>** type.
- Replace the **addDigitalVideoDisc()** and **removeDigitalVideoDisc()** methods with **addMedia()** and **removeMedia()**

```

1 package hust.soict.globalict.aims.store;
2
3•import java.util.ArrayList;
4 import hust.soict.globalict.aims.media.Media;
5
6 public class Store {
7
8     private ArrayList<Media> itemsInStore = new ArrayList<Media>();
9

```

addMedia() method:

```

10•    public void addMedia(Media media) {
11        if(itemsInStore.contains(media)) {
12            System.out.println("This media is already in the store!");
13        }
14        else {
15            itemsInStore.add(media);
16            System.out.println("This media has been added in the store");
17        }
18    }

```

removeMedia() method:

```

20•    public void removeMedia(Media media) {
21        if(itemsInStore.contains(media)) {
22            itemsInStore.remove(media);
23            System.out.println("This media has been removed from the store");
24        }
25        else {
26            System.out.println("This media does not exist in the store");
27        }
28    }

```

14. Constructors of whole classes and parent classes

- Write constructors for parent and child classes. Remove redundant setter methods if any

For Media class:

```
10*     public Media(int id, String title, String category, float cost) {  
11         this.title = title;  
12         this.category = category;  
13         this.cost = cost;  
14     }
```

For Book class:

```
10*     public Book(int id, String title, String category, float cost) {  
11         super(id, title, category, cost);  
12     }
```

For Disc class:

```
8*     public Disc(int id, String title, String category, float cost) {  
9         super(id, title, category, cost);  
10    }  
11  
12*    public Disc(int id, String title, String category, String director, float cost) {  
13        super(id, title, category, cost);  
14        this.director = director;  
15    }  
16  
17*    public Disc(int id, String title, String category, String director, int length, float cost) {  
18        super(id, title, category, cost);  
19        this.director = director;  
20        this.length = length;  
21    }
```

For DigitalVideoDisc class:

```
5*     public DigitalVideoDisc(int id, String title, String category, float cost) {  
6         super(id, title, category, cost);  
7     }  
8  
9*     public DigitalVideoDisc(int id, String title, String category, String director, float cost) {  
10        super(id, title, category, director, cost);  
11    }  
12  
13*    public DigitalVideoDisc(int id, String title, String category, String director, int length, float cost) {  
14        super(id, title, category, director, length, cost);  
15    }
```

For CompactDisc class:

```
11*    public CompactDisc(int id, String title, String category, String artist, float cost) {  
12        super(id, title, category, cost);  
13        this.artist = artist;  
14    }  
15  
16*    public CompactDisc(int id, String title, String category, String artist, String director, float cost) {  
17        super(id, title, category, director, cost);  
18        this.artist = artist;  
19    }  
20  
21*    public CompactDisc(int id, String title, String category, String artist, String director, int length, float cost) {  
22        super(id, title, category, director, length, cost);  
23        this.artist = artist;  
24    }
```

For Track class:

```

8•    public Track(String title, int length) {
9        this.title = title;
10       this.length = length;
11    }

```

15. Unique item in a list

To make sure the list of media in cart or list of tracks in a CD should not contain identical objects, we can override the `equals()` method of the `Object` class

- Please override the boolean `equals(Object o)` of the `Media` and the `Track` class so that two objects of these classes can be considered as equal if:
 - o For the `Media` class: the `title` is equal

```

16•  @Override
17  public boolean equals(Object obj) {
18      if(obj == null) {
19          return false;
20      }
21      Media media = (Media)obj;
22      return (media.title == null) ? false : this.title.equals(media.getTitle());
23  }
24

```

- o For the `Track` class: the `title` and the `length` are equal

```

13•  @Override
14  public boolean equals(Object obj) {
15      if(obj == null) {
16          return false;
17      }
18      Track track = (Track)obj;
19      if(track.title == null) {
20          return false;
21      }
22      return this.title.equals(track.title) && this.length == track.length;
23  }

```

16. Polymorphism with `toString()` method

This exercise gives an illustration for polymorphism at behavior level.

Recall that for each type of media, we have implemented a `toString()` method that prints out the information of the object. When calling this method, depending on the type of object, corresponding `toString()` will be performed.

`toString()` method in `Book` class:

```

34•  public String toString() {
35      return "Book: [" + getId() + "] - " + getTitle() + " - " + getCategory() + " - " + getCost() + "$";
36  }
37
38 }

```

`toString()` method in `CompactDisc` class:

```

59     public String toString() {
60         return "CompactDisc: (ID " + getId() + " - " + getTitle() + " - " + getCategory() + artist + " - " +
61             getDirector() + " - " + getLength() + " - " + getCost() + "$";
62     }

```

toString() method in DigitalVideoDisc class:

```

17•    public String toString() {
18     return "DVD: (ID " + getId() + " - " + getTitle() + " - " + getCategory() + " - "
19         + getDirector() + " - " + getLength() + ": " + getCost() + "$";
20   }

```

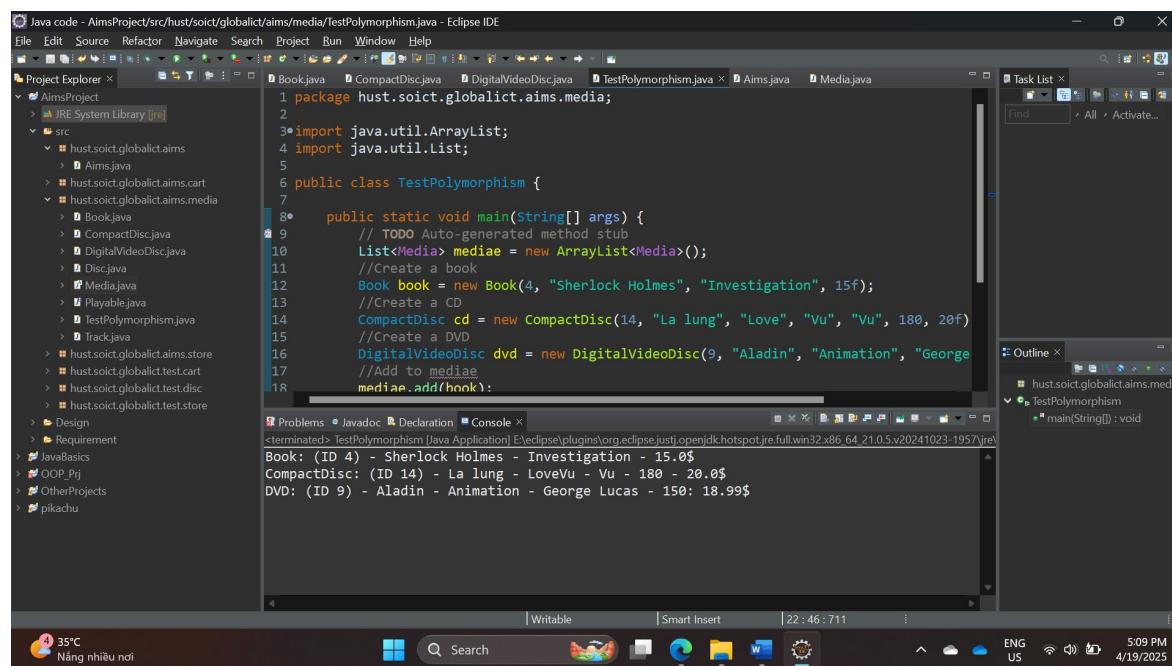
- Create an ArrayList of Media, then add some media (CD, DVD or Book) into the list.
 - Iterate through the list and print out the information of the media by using toString() method.
- Observe what happens and explain in detail.

```

1 package hust.soict.globalict.aims.media;
2
3•import java.util.ArrayList;
4 import java.util.List;
5
6 public class TestPolymorphism {
7
8•    public static void main(String[] args) {
9        // TODO Auto-generated method stub
10       List<Media> mediae = new ArrayList<Media>();
11       //Create a book
12       Book book = new Book(4, "Sherlock Holmes", "Investigation", 15f);
13       //Create a CD
14       CompactDisc cd = new CompactDisc(14, "La lung", "Love", "Vu", "Vu", 180, 20f);
15       //Create a DVD
16       DigitalVideoDisc dvd = new DigitalVideoDisc(9, "Aladin", "Animation", "George Lucas", 150, 18.99f);
17       //Add to mediae
18       mediae.add(book);
19       mediae.add(cd);
20       mediae.add(dvd);
21       for(Media m:mediae) {
22           System.out.println(m.toString());
23       }
24   }
25 }

```

Result:



17. Sort media in the cart

As mentioned before, when seeing the current cart, the user can sort the items in the cart by title or by cost:

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

Here, we can use **Comparator** to allow multiple sorting ways of **Media**:

Note: The `Comparator` interface is a comparison function, which imposes a total ordering on some collection of objects. Comparators can be passed to a sort method (such as `Collections.sort()`) to allow precise control over the sort order.

Please open the Java docs to see the information of this interface.

- Create two classes of comparators, one for each type of ordering

```
public class MediaComparatorByCostTitle implements Comparator<Media>
public class MediaComparatorByTitleCost implements Comparator<Media>
```

- Implement the `compare()` method of each comparator class to reflect the ordering that we want, either by title then cost, or by cost then title. You may utilize the method `Comparator.thenComparing()` to sort using multiple fields.

```
1 package hust.soict.globalict.aims.media;
2
3 import java.util.Comparator;
4
5 public class MediaComparatorByTitleCost implements Comparator<Media> {
6
7     @Override
8     public int compare(Media o1, Media o2) {
9         // TODO Auto-generated method stub
10        if(o1.getTitle().equals(o2.getTitle())){
11            return o1.getCost() > o2.getCost() ? -1 : 1;
12        }
13        return o1.getTitle().compareTo(o2.getTitle());
14    }
15
16 }
17
18 }
```

```
1 package hust.soict.globalict.aims.media;
2
3 import java.util.Comparator;
4
5 public class MediaComparatorByCostTitle implements Comparator<Media> {
6
7     @Override
8     public int compare(Media o1, Media o2) {
9         // TODO Auto-generated method stub
10        if(o1.getCost() < o2.getCost()) {
11            return 1;
12        }
13        if(o1.getCost() > o2.getCost()) {
14            return -1;
15        }
16        return o1.getTitle().compareTo(o2.getTitle());
17    }
18
19 }
20
```

Test 2 types of sort:

```
9•     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         List<Media> mediae = new ArrayList<Media>();
12         //Create a book
13         Book book = new Book(4, "Sherlock Holmes", "Investigation", 15f);
14         //Create a CD
15         CompactDisc cd = new CompactDisc(14, "La lung", "Love", "Vu", "Vu", 180, 20f);
16         //Create a DVD
17         DigitalVideoDisc dvd = new DigitalVideoDisc(9, "Aladin", "Animation", "George Lucas", 150, 18.99f);
18         //Add to mediae
19         mediae.add(book);
20         mediae.add(cd);
21         mediae.add(dvd);
22         Book book1 = new Book(7, "Conan", "Investigation", 15f);
23         mediae.add(book1);
24         CompactDisc cd1 = new CompactDisc(12, "La lung", "Love", "Vu", "Vu", 200, 30f);
25         mediae.add(cd1);
26         Collections.sort(mediae, Media.COMPARE_BY_COST_TITLE);
27         System.out.println("Sort by cost then title:");
28         for(Media m:mediae) {
29             System.out.println(m.toString());
30         }
31         Collections.sort(mediae, Media.COMPARE_BY_TITLE_COST);
32         System.out.println("\nSort by title then cost:");
33         for(Media m:mediae) {
34             System.out.println(m.toString());
35         }
36     }
37 }
```

Result:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with packages like JRE System Library [jre], src, hust.soict.globalict.aims, hust.soict.globalict.aims.cart, hust.soict.globalict.aims.media, hust.soict.globalict.test.cart, hust.soict.globalict.test.disc, hust.soict.globalict.test.store, Design, Requirement, JavaBasics, OOP_Prj, and OtherProjects.
- Code Editor:** Displays the Java code for `TestPolymorphism.java`. It includes imports for `Media`, `CompactDisc`, `DigitalVideoDisc`, and `Book`. The code creates instances of these classes and sorts them using two different comparison strategies: `COMPARE_BY_COST_TITLE` and `COMPARE_BY_TITLE_COST`.
- Console Output:** Shows the execution results:

```
Sort by cost then title:
CompactDisc: (ID 12) - La lung - LoveVu - Vu - 200 - 30.0$
CompactDisc: (ID 14) - La lung - LoveVu - Vu - 180 - 20.0$
DVD: (ID 9) - Aladin - Animation - George Lucas - 150: 18.99$
Book: (ID 7) - Conan - Investigation - 15.0$
Book: (ID 4) - Sherlock Holmes - Investigation - 15.0$

Sort by title then cost:
DVD: (ID 9) - Aladin - Animation - George Lucas - 150: 18.99$
Book: (ID 7) - Conan - Investigation - 15.0$
CompactDisc: (ID 12) - La lung - LoveVu - Vu - 200 - 30.0$
CompactDisc: (ID 14) - La lung - LoveVu - Vu - 180 - 20.0$
Book: (ID 4) - Sherlock Holmes - Investigation - 15.0$
```
- Bottom Status Bar:** Shows system information including the date (4/19/2025), time (6:06 PM), battery level (18:16), and network status.

18. Create a complete console application in the Aims class

```

1 package hust.soict.globalict.aims;
2
3 import java.util.Scanner;
4
5 public class Aims {
6
7     private Store store = new Store();
8     private Cart cart = new Cart();
9
10    public static void showMenu() {
11        System.out.println("AIMS: ");
12        System.out.println("-----");
13        System.out.println("1. View store");
14        System.out.println("2. Update store");
15        System.out.println("3. See current cart");
16        System.out.println("4. Exit");
17        System.out.println("-----");
18        System.out.println("Please choose a number: 0-1-2-3");
19    }
20
21    public static void storeMenu() {
22        System.out.println("Options: ");
23        System.out.println("-----");
24        System.out.println("1. See a media's details");
25        System.out.println("2. Add a media to cart");
26        System.out.println("3. Play a media");
27        System.out.println("4. See current cart");
28        System.out.println("0. Back");
29        System.out.println("-----");
30
31        System.out.println("Please choose a number: 0-1-2-3-4");
32    }
33
34    public static void mediaDetailsMenu() {
35        System.out.println("Options: ");
36        System.out.println("-----");
37        System.out.println("1. Add to cart");
38        System.out.println("2. Play");
39        System.out.println("0. Back");
40        System.out.println("-----");
41        System.out.println("Please choose a number: 0-1-2");
42    }
43
44    public static void cartMenu() {
45        System.out.println("Options: ");
46        System.out.println("-----");
47        System.out.println("1. Filter media in cart");
48        System.out.println("2. Sort media in cart");
49        System.out.println("3. Remove media from cart");
50        System.out.println("4. Play a media");
51        System.out.println("5. Place order");
52        System.out.println("0. Back");
53        System.out.println("-----");
54        System.out.println("Please choose a number: 0-1-2-3-4-5");
55    }
56
57    public void chooseMainMenu() {
58        showMenu();
59
60        Scanner in = new Scanner(System.in);
61        int opt = in.nextInt();
62        if(opt == 1) {
63            viewStore();
64        }
65        else if(opt == 2) {
66            updateStore();
67        }
68        else if(opt == 3) {
69            cart.display();
70            cartMenu();
71            seeCurrentCart();
72        }
73        else {
74            System.exit(0);
75        }
76    }
77
78    public void viewStore() {
79        store.displayItems();
80        storeMenu();
81        Scanner in = new Scanner(System.in);
82        int opt = in.nextInt();
83        if(opt == 1) {
84            System.out.print("Enter title of media: ");
85            String title = in.nextLine();
86            store.seeMediaDetail(title);
87            mediaDetailsMenu();
88            seeMediaDetails();
89        }
90    }
91
92}

```

```

93     }
94     else if(opt == 2) {
95         Media media = null;
96         cart.addMedia(media);
97     }
98     else if(opt == 3){
99         CompactDisc media = null;
100        media.play();
101    }
102    else if(opt == 4) {
103        cart.display();
104    }
105    else {
106        chooseMainMenu();
107    }
108 }
109
110 public void seeMediaDetails() {
111     Scanner in = new Scanner(System.in);
112     int opt = in.nextInt();
113     DigitalVideoDisc media = null;
114     if(opt == 1) {
115         cart.addMedia(media);
116     }
117     else if(opt == 2) {
118         media.play();
119     }
120     else {
121         viewStore();
122     }
123
124 public void updateStore() {
125     Scanner in = new Scanner(System.in);
126     System.out.println("Add or Remove?");
127     String type = in.nextLine();
128     if(type.equals("Add")) {
129         Media media = null;
130         store.addMedia(media);
131     }
132     else {
133         Media media = null;
134         store.addMedia(media);
135     }
136 }
137
138 public void seeCurrentCart() {
139     Scanner in = new Scanner(System.in);
140     int opt = in.nextInt();
141     if(opt == 1) {
142         System.out.println("filter with id or title:");
143         String type = in.nextLine();
144         cart.filterCart(type);
145     }
146     else if(opt == 2) {
147         System.out.println("sort by title or cost:");
148         String type = in.nextLine();
149         cart.sortMedia(type);
150     }
151     else if(opt == 3) {
152         Media media = null;
153
154         cart.removeMedia(media);
155     }
156     else if(opt == 4) {
157         CompactDisc media = null;
158         media.play();
159     }
160     else if(opt == 5) {
161         System.out.println("Ordering successfully!");
162         cart.clear();
163     }
164     else {
165         chooseMainMenu();
166     }
167 }
168
169 public static void main(String[] args) {
170     Aims aims = new Aims();
171     aims.chooseMainMenu();
172 }
173 }
```

Java code - AimsProject/src/hust/soict/globalict/aims/Aims.java - Eclipse IDE

```

File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console
terminated> Aims [Java Application] E:\eclipse\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64.21.0.5.v20241023-1957\jre\bin\javaw.exe (Apr 19, 2025, 10:04:57 PM – 10:05:12 PM elapsed: 0:00:15.068) [pid: 7280]
AIMS:
-----
1. View store
2. Update store
3. See current cart
0. Exit

Please choose a number: 0-1-2-3
1
There is no item in the store
Options:
-----
1. See a media's details
2. Add a media to cart
3. Play a media
4. See current cart
0. Back

Please choose a number: 0-1-2-3-4
4
Total Cost 0.00

```

Không khí: Cực tệ
Thứ Ba Tuần Tối

- Update the UML class diagram for the **AimsProject**.

