Object-Oriented Programming

Tên: Trần Phạm Minh Đức
MSSV: 20226077

Lab 04: Inheritance and Polymorphism

1. Import the existing project into the workspace of Eclipse

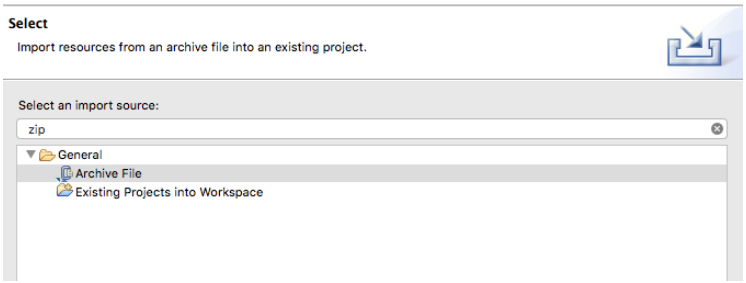


Figure 1. Import existing project

2. Additional requirements of AIMS

- In this section, I will extend the AIMS system to support two new types of media: books and CDs. Each media type will have specific information stored and displayed when users view the details, including information about the title, genre, author or artist, and additional details such as the track list or number of keywords. At the same time, I will implement a content playback feature for CDs and DVDs, ensuring that full information is displayed and checking the duration to notify users if the media cannot be played. In the following chapters, I will specify each requirement and implement each function in detail.

3. Creating the `Book` class

- Package: **`hust.soict.dsai.aims.media`**
- Name: **`Book`**
- Access modifier: **`public`**
- Superclass: **`java.lang.Object`**
- **`public static void main(String[] args)`**: **do not check**
- Constructors from Superclass: **Check**
- All other boxes: **Do not check**

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`**.

```java
public class Book extends Media{
    private List<String> authors = new ArrayList<>();

    public Book(int id, String title, String category, float cost, List<String> authors)
        super(title, category, cost);
        this.setId(id);
        this.authors = authors;
    }
```

Figure 2. Adding fields to Book class

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
- The **`removeAuthor`**(...) method should ensure that the author is present in the **`ArrayList`** before removing
- Reference to some useful methods of the **`ArrayList`** class

```java
    public void addAuthor(String authorName) {
        if (!authors.contains(authorName)) {
            authors.add(authorName);
            System.out.println("Added " + authorName + " to the author list");
        } else {
            System.out.println(authorName + " is already in the author list");
        }
    }

    public void removeAuthor(String authorName){
        if (authors.contains(authorName)){
            authors.remove(authorName);
            System.out.println("Removed " + authorName + " from the author list");
        } else {
            System.out.println(authorName + "is not in the author list");
        }
    }
```

4. Creating the abstract **`Media`** class

- Add fields to the **`Media`** class

```java
package hust.soict.dsai.aims.media;

import java.util.Comparator;

public abstract class Media {
    private int id;
    private String title;
    private String category;
    private float cost;
    public static final Comparator<Media> COMPARE_BY_TITLE_COST = new MediaComparatorByTi
    public static final Comparator<Media> COMPARE_BY_COST_TITLE = new MediaComparatorByCo
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getCategory() {
        return category;
    }
    public void setCategory(String category) {
        this.category = category;
    }
    public float getCost() {
        return cost;
    }
    public void setCost(float cost) {
        this.cost = cost;
    }
    public Media() {}
    public Media(String title, String category, float cost) {
        this.title = title;
        this.category = category;
        this.cost = cost;
    }
```

- Remove fields and methods from **`Book`** and **`DigitalVideoDisc`** classes
- 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.`**

5. Creating 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.

```java
public class CompactDisc extends Disc implements Playable{
    private String artist;
    private ArrayList<Track> tracks;

    public CompactDisc(String title, String category, float cost) {
        super(title, category, cost);
    }
    public CompactDisc(String title, String category, float cost, String artist, ArrayList<Track> tracks) {
        super(title, category, cost);
        this.artist = artist;
        this.tracks = tracks;
    }

    public String getArtist() {
        return artist;
    }

    public void addTrack(Track track){
        if (!tracks.contains(track)){
            tracks.add(track);
            System.out.println("Added the input track to the list of tracks");
        }
        else{
            System.out.println("The input track is already in the list of tracks");
        }
    }

    public void removeTrack(Track track){
        if (tracks.contains(track)){
            tracks.remove(track);
            System.out.println("Removed the input track from the list of tracks");
        }
        else{
            System.out.println("The input track doesn't exist in the list of tracks");
        }
    }
```

```java
    public int getLength(){
        int totalLength = 0;
        for (Track track : tracks){
            totalLength += track.getLength();
        }
        return totalLength;
    }

    public void play(){
        System.out.println("Playing this CD\nArtist: " + artist);
        for (Track track : tracks){
            track.play();
        }
    }

    @Override
    public String toString() {
        return "CompactDisc{" +
                "Artist='" + artist + '\'' +
                ", Tracks=" + tracks + '\'' +
                ", ID=" + getId()  +
                ", Title='" + getTitle() + '\'' +
                ", Category='" + getCategory() + '\'' +
                ", Cost=" + getCost() +
                "}\n";
    }
}
```

6. Create the Playable interface

```java
package hust.soict.dsai.aims.media;

public interface Playable {
    public void play();
}
```

- 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`**

```java
package hust.soict.dsai.aims.media;
import hust.soict.dsai.aims.media.Media;

public class DigitalVideoDisc extends Disc implements Playable {
    private String director;
    private int length;
    private static int nbDigitalVideoDiscs = 0;
```

```java
public class CompactDisc extends Disc implements Playable{
    private String artist;
    private ArrayList<Track> tracks;

    public CompactDisc(String title, String category, float cost) {
        super(title, category, cost);
    }
```

- For the **`Track`** class, insert the keywords **`implements Playable`** after the keywords **`public class Track`**

```java
package hust.soict.dsai.aims.media;

public class Track implements Playable{
    private String title;
    private int length;

    public String getTitle() {
        return title;
    }

    public int getLength() {
        return length;
    }

    public Track(String title, int length) {
        this.title = title;
        this.length = length;
    }
```

- 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());
}
```

```java
    public void play() {
        System.out.println("Playing DVD: " + this.getTitle());
        System.out.println("DVD length: " + this.getLength());
    }
```

- Similar additions with the **`Track`** class

- Implement **`play`**() for **`CompactDisc`**

- Since the **`CompactDisc`** class contains a **`ArrayList`** of **`Tracks`**, each of which can be played on its own. The **`play`**() method should output some information about the **`CompactDisc`** to console

- Loop through each track of the arraylist and call **`Track's`** play() method\

```
public void play(){
    System.out.println("Playing this CD\nArtist: " + artist);
    for (Track track : tracks){
        track.play();
    }
}
```

7. Update the `Cart` class to work with `Media`

- Remove the **`itemsOrdered`** array, as well as its add and remove methods.
    - The **`qtyOrdered`** field is no longer needed since it was used to track the number of **`DigitalVideoDiscs`** in the **`itemsOrdered`** array, so remove it and its accessor and mutator (if exist).
    - Add the **`itemsOrdered`** to the **`Cart`** class

a. To create this field, type the following code in the **`Cart`** class, in place of the **`itemsOrdered`** array declaration that you deleted:

`private ArrayList<Media> itemsOrdered = new ArrayList<Media>();`

```
package hust.soict.dsai.aims.cart;
import hust.soict.dsai.aims.media.DigitalVideoDisc;
import hust.soict.dsai.aims.media.Media;

import java.util.ArrayList;

public class Cart {
    public static final int MAX_NUMBER_ORDERED = 20;
    private ArrayList<Media> itemsOrdered = new ArrayList<Media>();
```

- Create **`addMedia`**() and **`removeMedia`**() to replace **`addDigitalVideoDisc`**() and **`removeDigitalVideoDisc`**()

```java
public class Cart {
    public static final int MAX_NUMBER_ORDERED = 20;
    private ArrayList<Media> itemsOrdered = new ArrayList<Media>();
    public void addMedia(Media media){
        if (itemsOrdered.size() == MAX_NUMBER_ORDERED){
            System.out.println("Maximum number of orders exceeded");
            return;
        }
        itemsOrdered.add(media);
        System.out.println("Added the media to the cart");
    }
    public void removeMedia(Media media){
        if (itemsOrdered.contains(media)){
            itemsOrdered.remove(media);
            System.out.println("Removed the media from the cart");
        }
        else{
            System.out.println("The media doesn't exist in the cart");
        }
    }
```

- Update the **`totalCost`**() method

```java
public float totalCost() {
    float totalCost = 0;
    for (Media item: itemsOrdered) {
        totalCost += item.getCost();
    }
    return totalCost;
}
```

8. 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`**()

```java
package hust.soict.dsai.aims.store;

import hust.soict.dsai.aims.media.DigitalVideoDisc;
import hust.soict.dsai.aims.media.Media;

import java.util.ArrayList;

public class Store {
    public static int MAX_QUANTITY = 50000;
    private ArrayList<Media> itemsInStore = new ArrayList<Media>();
    public void addMedia(Media media){
        if (itemsInStore.size() == MAX_QUANTITY){
            System.out.println("Maximum quantity exceeded");
            return;
        }
        itemsInStore.add(media);
        System.out.println("Added the media to the store");
    }
    public void removeMedia(Media media){
        if (itemsInStore.contains(media)){
            itemsInStore.remove(media);
            System.out.println("Removed the media from the store");
        }
        else{
            System.out.println("The media doesn't exist in the store");
        }
    }
    public void print(){
        System.out.println("**********STORE**********");
        System.out.println("In-store items:");
        for (Media item: itemsInStore){
            System.out.println(item.toString());
        }
        System.out.println("**************************");
    }

    public Media searchByTitle(String title){
        for (Media item: itemsInStore){
            if (item.isMatch(title)){
                return item;
            }
        }
        return null;
    }
    public Media searchByID(int id){
        for (Media item: itemsInStore){
            if (item.getId() == id){
                return item;
            }
        }
        return null;
    }
}
```

9. Constructors of whole classes and parent classes

- Update the UML class diagram for the **`AimsProject`**. Update the new .astah & .png file in the **`Design`** directory. We can apply Release Flow here by creating a branch, e.g., **`topic/update-class-diagram/aims-project/lab04`**, push the diagram and its image, and then merge with master.
- Which classes are aggregates of other classes? Checking all constructors of whole classes if they initialize for their parts?

- **Order**:
  - Lớp `Order` chứa thuộc tính `itemsOrdered` kiểu `ArrayList<Media>`. Đây là quan hệ tổng hợp vì `Order` quản lý danh sách các đối tượng `Media`.
  - Constructor của `Order` khởi tạo `itemsOrdered`:

```
public Order() {
    this.itemsOrdered = new ArrayList<>();
}
```

  - Kết luận: **`Order` là aggregate class, `Media` là part class.**

- **CompactDisc** (nếu có):
  - Nếu lớp `CompactDisc` chứa danh sách các `Track`, thì nó cũng là một aggregate class. Ví dụ:

```
private ArrayList<Track> tracks;
public CompactDisc() {
    this.tracks = new ArrayList<>();
}
```

  - Kết luận: **`CompactDisc` là aggregate class, `Track` là part class.**

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

1. Constructor của lớp cha `Media`:

- Constructor đã có và hoàn chỉnh:

```
public Media(String title, String category, float cost) {
    this.title = title;
    this.category = category;
    this.cost = cost;
}
```

2. Constructor cho các lớp con:

- **Book:**

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

```
    private String author;

    public Book(String title, String category, float cost, String author) {
        super(title, category, cost); // Gọi constructor của Media
        this.author = author;
    }
}
```

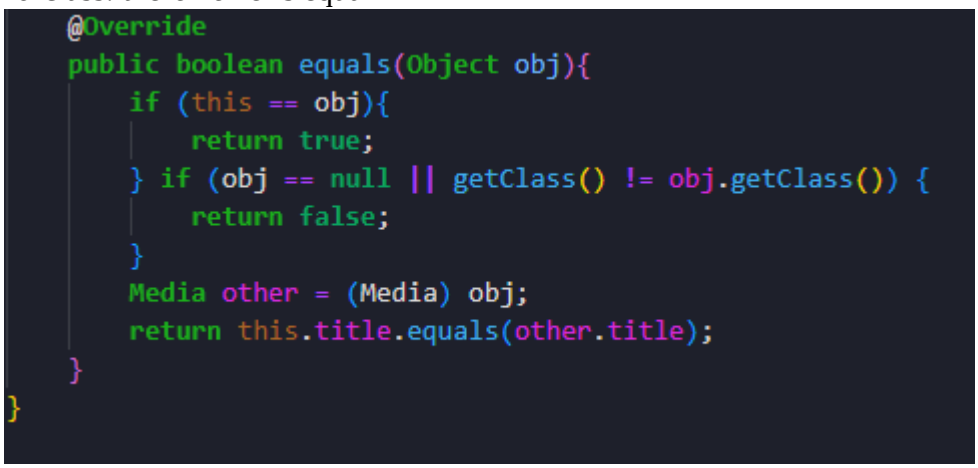
- **CompactDisc:**

```
public class CompactDisc extends Media {
    private String artist;

    public CompactDisc(String title, String category, float cost, String
artist) {
        super(title, category, cost); // Gọi constructor của Media
        this.artist = artist;
    }
}
```

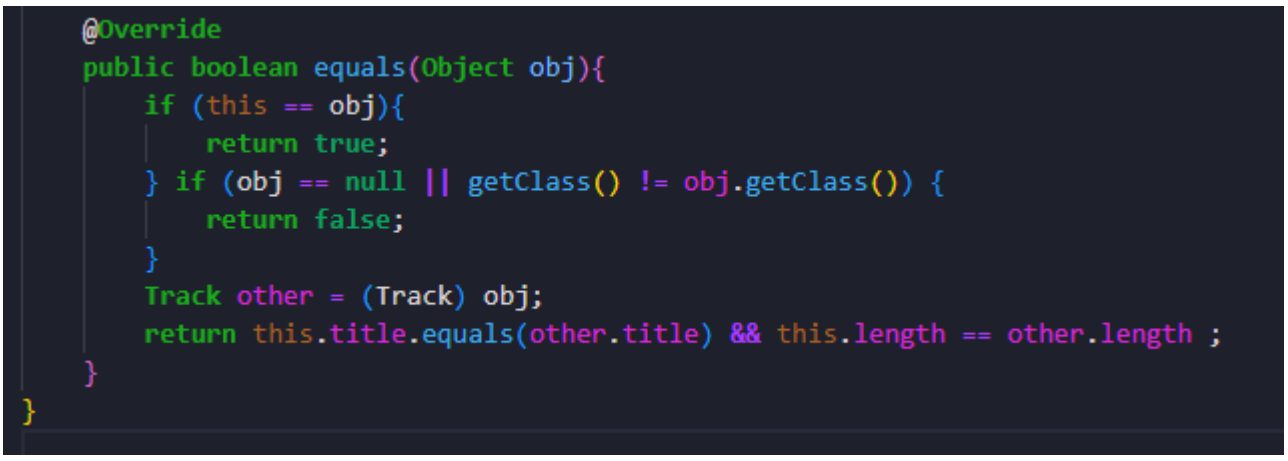
10. Unique item in a list

+ For the `Media` class: the `title` is equal



```
    @Override
    public boolean equals(Object obj){
        if (this == obj){
            return true;
        } if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Media other = (Media) obj;
        return this.title.equals(other.title);
    }
}
```

+ For the `Track` class: the `title` and the `length` are equal



```
    @Override
    public boolean equals(Object obj){
        if (this == obj){
            return true;
        } if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Track other = (Track) obj;
        return this.title.equals(other.title) && this.length == other.length ;
    }
}
```

When overriding the `equals()` method of the `Object` class, you will have to cast the `Object` parameter `obj` to the type of `Object` that you are dea ling with. For example, in the `Media` class, you must cast the `Object` obj to a `Media`, and then check the equality of the two objects' attributes as the above requirements (i.e. `title` for `Media`; `title` and `length` for `Track`).

```
    @Override
    public boolean equals(Object obj){
        if (this == obj){
            return true;
        } if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Media other = (Media) obj;
        return this.title.equals(other.title);
    }
}
```

If the passing object is not an instance of `Media`, what happens?
Nếu đối tượng được truyền vào không phải là kiểu cần kiểm tra (`Media` hoặc `Track`), phương thức sẽ trả về `false`.

Note: We can apply Release Flow here by creating a topic branch for the override of `equals()` method.

11. Polymorphism with toString() method

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.

- 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.

- Khi duyệt qua danh sách và gọi `toString()`, phương thức cụ thể của từng lớp (`Book`, `DigitalVideoDisc`, `CompactDisc`) sẽ được thực thi.
- Điều này minh họa tính đa hình: một lời gọi phương thức (`toString()`) sẽ thực hiện hành vi khác nhau dựa trên kiểu đối tượng thực tế.

Sample code:

```java
public static void main(String[] args) {
    CompactDisc cd = new CompactDisc("Thriller", "Music", 51.5f);
    DigitalVideoDisc dvd = new DigitalVideoDisc("Contratiempo", "Crime", "Oriol Paulo", 106, 30.0f);
    Book book = new Book(5, "Harry Potter", "Fantasy", 11.4f,  new ArrayList<>(Arrays.asList("J.K Rowling")));
    mainStore.addMedia(cd);
    mainStore.addMedia(dvd);
    mainStore.addMedia(book);

    while (true) {
        showMenu();
        int choice = getIntegerInput("");
        switch (choice) {
            case 1:
                viewStore();
                break;
            case 2:
                updateStore();
                break;
            case 3:
                seeCurrentCart();
                break;
            case 0:
                System.out.println("Closing");
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
```

Figure 3. Polymorphism

12. Sort media in the cart

```java
package hust.soict.dsai.aims.media;

import java.util.Comparator;

public abstract class Media {
    private int id;
    private String title;
    private String category;
    private float cost;
    public static final Comparator<Media> COMPARE_BY_TITLE_COST = new MediaComparatorByTitleCost();
    public static final Comparator<Media> COMPARE_BY_COST_TITLE = new MediaComparatorByCostTitle();
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
```

```java
package hust.soict.dsai.aims.media;

import java.util.Comparator;

public class MediaComparatorByCostTitle implements Comparator<Media> {
    @Override
    public int compare(Media m1, Media m2) {
        int costComparison = Float.compare(m1.getCost(), m2.getCost());
        if (costComparison != 0) {
            return costComparison;
        } else {
            return m1.getTitle().compareTo(m2.getTitle());
        }
    }
}
```

nsProject > src > main > java > hust > soict > dsai > aims > media > J MediaComparatorByTitleCost.java

```java
package hust.soict.dsai.aims.media;

import java.util.Comparator;

public class MediaComparatorByTitleCost implements Comparator<Media> {
    @Override
    public int compare(Media m1, Media m2) {
        int titleComparison = m1.getTitle().compareTo(m2.getTitle());
        if (titleComparison != 0) {
            return titleComparison;
        } else {
            return Float.compare(m2.getCost(), m1.getCost());
        }
    }
}
```

Nếu sử dụng giao diện `Comparable` thay vì `Comparator`:

- **Lớp nào cần triển khai giao diện `Comparable`?**
  - Lớp `Media` nên triển khai `Comparable`, vì nó là lớp cơ sở chứa các thuộc tính chung cần được sắp xếp.
- **Phương thức `compareTo()` trong các lớp này cần được triển khai như thế nào để phản ánh thứ tự mong muốn?**
  - Triển khai `compareTo()` để:
    - Sắp xếp theo **title**, sau đó là **cost**.
    - Hoặc sắp xếp theo **cost**, sau đó là **title**.

- **Có thể có hai quy tắc sắp xếp khác nhau (title rồi cost, hoặc cost rồi title) nếu dùng `Comparable` không?**
    - Không, `Comparable` chỉ hỗ trợ **một quy tắc sắp xếp tự nhiên** (natural ordering). Để có nhiều quy tắc, bạn cần sử dụng `Comparator`.
- **Nếu `DVD` có quy tắc sắp xếp khác (title, rồi length giảm dần, rồi cost), bạn sẽ thay đổi mã như thế nào?**
    - Ghi đè phương thức `compareTo()` riêng cho `DVD`.

13. Create a complete console application in the Aims class

(Trong file code)