Object-Oriented Language and Theory

Lecturer: NGUYEN Thi Thu Trang, trangntt@soict.hust.edu.vn
Teaching Assistant: DO Minh Hieu, hieudominh@hotmail.com

Lab 07: Abstract Class and Interface

* Objectives:

In this lab, you will practice with:

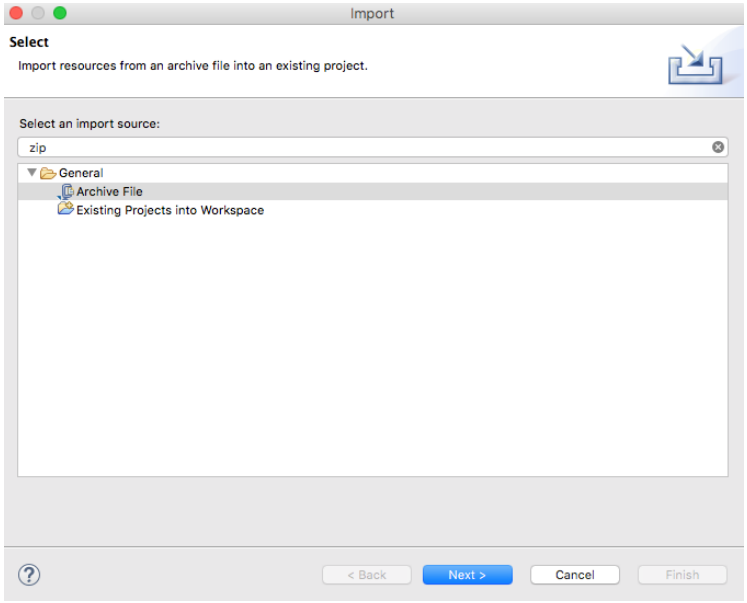
- Creating abstract class, abstract methods
- Creating interface and implement it

You need to use the project that you did with the previous labs including both AimsProject and other projects. Please follow the Release Flow to commit and push to the bitbucket.

1. Import the AimsProject

- Open Eclipse

- Open File -> Import. Type zip to find Archive File if you have exported as a zip file before. You may choose Existing Projects into Workspace if you want to open an existing project in your computer. Ignore this step if the AimsProject is already opened in the workspace.



- Click Next and Browse to a zip file or a project to open

2. Extending the AIMS project to update information of CDs (Compact Discs)

Question: If we keep the class Media as current, what happen if a user create an instance of Media? Not a Book, not a DVD nor CD.

Solution: to prevent the creation of an instance of Media, we should make it as abtract.

You can apply Release Flow here by creating a **`topic`** branch for making the **`Media`** class abstract.

2.1. Make the `Media` as an abstract class

- Open **`Media`** class
- Modify the **`Media`** class and make it **`abstract`**
- Keep three fields of **`Media`**: **`title`**, **`category`**, **`cost`** and their associated methods, but remove the setters. We keep getter methods of these fields so that sub-classes can reuse them.

2.2. 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.
- Make the **`DigitalVideoDisc`** extending the **`Disc`** class. Make changes if need be.
- Create the **`CompactDisc`** extending the **`Disc`** class too. Save your changes.

2.3. Create the `Track` class which models a track on a compact disc and will store information incuding 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

2.4. Open the `CompactDisc` class

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

3. Create 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
- 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
- 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

4. Update the `Aims` class

- You will update the **`Aims`** class to test your changes.
    - Create more choices for your application
    - Update the menu of choices:
        - For the addition of new item to the order, the program should ask for the type: **`Book`**, **`CompactDisc`** or **`DigitalVideoDisc`**
        - For the **`CompactDisc`**, the program should allow to add information of **`Tracks`**
        - When adding a cd/dvd to the order, the user may ask for play them
- You will update the class diagram for the above classes and interfaces.

5. Update the menu of the main method in Aims

In this lab, you will refactor the code to create two kinds of menu: a menu for admin and a menu for user.

5.1. Create command-line menu for admin

Open the Aims class. You will create a prompted menu as following:

```
public static void showAdminMenu() {
            System.out.println("Product Management Application: ");
            System.out.println("--------------------------------");
            System.out.println("1. Create new item");
            System.out.println("2. Delete item by id");
            System.out.println("3. Display the items list");
            System.out.println("0. Exit");
            System.out.println("--------------------------------");
            System.out.println("Please choose a number: 0-1-2-3");
}
```

Now, you have to refactor the code to make available a menu for admin who can create item (DVD, CD, Track, Book) and add them to a list of available items for the store. You have to note that, because of the difference between DVD, CD and Book, the program should allow to create each item in a different way. To do that, you have to create different methods to create items DVD, CD, Track, Book. Thinking to inheritance and abstract methods to implement them.

As you know, in the previous lab, we have created these methods in each class: Media – Book – DVD – CD as the below example of the DigitalVideoDisc class:

```
public Media createMediaItem() {
            Scanner sc = new Scanner(System.in);
            System.out.println("Title of the item: ");
            String title = sc.nextLine();
            System.out.println("Category of the item: ");
            String category = sc.nextLine();
            System.out.println("Price of the item: ");
            float cost = Float.parseFloat(sc.nextLine());
            System.out.println("Director: ");
            String director = sc.nextLine();
            System.out.println("Length: ");
            long length = Long.parseLong(sc.nextLine());
            return new DigitalVideoDisc(title, category, cost, director, length);
}
```

You can see that this method allow to create a media item, all the subclasses of Media classes have implemented this method to create the corresponding item (CD, DVD or Book). Now we move these methods outside of these classes to refactor them (the main reason for this comes from the fact that we have constructors to create instances of a class, so if an user want to create instance of this class but don't use its constructor, it is better to do it outside of the class).

```
package hust.soict.hedspi.aims.media.factory;

import hust.soict.hedspi.aims.media.Media;

public interface MediaCreation {
```

```
    public Media createMediaFromConsole();

}
```

You create a package named

- `hust.soict.hedspi.aims.media.factory`
- `hust.soict.globalict.aims.media.factory` for ICT class

You create an interface named `MediaCreation` and its method which allow to create an Media item. Then you will create the following classes in this package who implement this interface:

- `DVDCreation`
- `CDCreation`
- `TrackCreation`
- `BookCreation`

Implement the corresponding method for each class, as example below:

```
package hust.soict.hedspi.aims.media.factory;

import java.util.Scanner;

import hust.soict.hedspi.aims.disc.DigitalVideoDisc;
import hust.soict.hedspi.aims.media.Media;

public class DVDCreation implements MediaCreation {

    @Override
    public Media createMediaFromConsole() {
        // TODO Auto-generated method stub
        Scanner sc = new Scanner(System.in);
        System.out.println("Media Title: ");
        String title = sc.nextLine();
        System.out.println("Media Category: ");
        String category = sc.nextLine();
        System.out.println("Media Price: ");
        float cost = Float.parseFloat(sc.nextLine());
        System.out.println("DVD Director: ");
        String director = sc.nextLine();
        System.out.println("DVD Length: ");
        long length = Long.parseLong(sc.nextLine());
        return new DigitalVideoDisc(title, category,
                                    cost, director, length);
    }
}
```

Do the same task for all the other classes.

Now in the AIMS class, you create a static method to create a media item

```
public static Media createMedia(MediaCreation mc) {
        return mc.createMediaFromConsole();
    }
```

Then, you create a method to show the creation menu, which allows admin to choose to create Book, DVD or CD, in this method you use the above method to create the kind of item which admin wants (for example, if admin choose 1 to create DVD, so we call the method `createMedia(new DVDCreation())`, and so on).

Finally, test the program for admin.

5.2. Create command-line menu for user

You have to also refactor the previous code to create a menu for user who can make an order, add item from the list of available items of the store, remove items from order, display the order information, this method call showUserMenu().

```
public static void showUserMenu() {
            System.out.println("Welcome to AIMS Store: ");
            System.out.println("--------------------------------");
            System.out.println("1. Create new order");
            System.out.println("2. Search for an item from the list by title");
            System.out.println("3. Add item to order by id (id in the list of
available items of the store");
            System.out.println("4. Remove item from order by id (id in the
order)");
            System.out.println("5. Display the order information");

            System.out.println("0. Exit");
            System.out.println("--------------------------------");
            System.out.println("Please choose a number: 0-1-2-3-4-5");
}
```

You have to note that, when user chooses 2, the program will take the title of the item the user types and return a list of corresponding items from the list of available items of the store including: id – [kind] - title – category - … while [kind] should be: DVD – CD or Book and the followed detailed information is corresponding to each kind of item.

When user choose 3, the program will allow user to add the corresponding item (by the id return from the 2-option) to the order. When user choose 4, the program allows to remove the corresponding item from the list of ordered items (so the id is the id in this list). The original items from the list of available items of the store aren't deleted.

----------END----------