Object-Oriented Language and Theory

Bui Thi Mai Anh, anhbtm@soict.hust.edu.vn

Lab 7: Abstract Class and Interface

* Objectives:

In this lab, you will practice with:

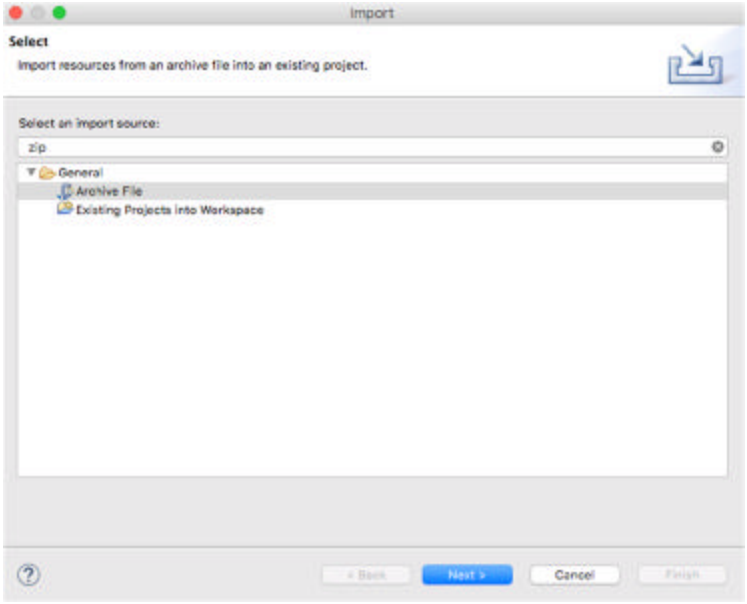
- Creating abstract class, abstract methods
- Creating interface and implement it
- Using Threads

You need to use the project that you did with the previous labs including both AimsProject and other projects.

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 allow the ordering of CDs (Compact Discs)

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

2.1. Make the `Media` 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

2.2. Create the Disc class extending the Media class

- The **Disc** class has two fields: **length** and **director**
- Create **getter**/**setter** methods for these fields
- Make the **DigitalVideoDisc** extending the **Disc** class
- Create the **CompactDisc** extending the **Disc** class

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

- Open the **Track** class
- Add two fields: **String title** and **int length**
- Make these fields **private** and create their **getter** and **setter** methods as **public**
- Save your changes

2.4. Open the CompactDisc class

- Add 3 fields to this class:
  - a **String** as **artist**
  - an **int** field as **length**
  - an **ArrayList** of **Track** as **tracks**
- Make all these fields as **private**
- Create **getter** and **setter** methods for only **artist**, make them **public**
- 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 total length 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

- Add to the Playable interface the method prototype: **`public void play();`**
- Save your changes
- Implement the **`Playable interface`**
  - Implement **`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());
}
```

  - For the **`Track`** class, you do the same
- 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 console 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

5. Practice with `Threads`

changes.

A class called **`MemoryDaemon`** runs as a daemon thread, tracking the memory usage in the system.

- The **MemoryDaemon** class implements the **java.lang.Runnable** interface
- Implement the **run**() method
- Add the field **long memoryUsed** to the **MemoryDaemon** class
- Initialize this field to 0. It keeps track of the memory usage in the system.
- Add code to the **run**() method to check memory usage as the program runs

Create a loop which will log the amount of memory used as the **Aims.main**() method executes. You will make use of the **java.lang.Runtime** class, which has a static method **getRuntime**().

```
public void run(){
   Runtime rt = Runtime.getRuntime();
   long used;

   while (true) {
      used = rt.totalMemory() - rt.freeMemory();
      if (used != memoryUsed) {
        System.out.println("\tMemory used = " + used);
        memoryUsed = used;
      }
   }
}
```

- Save your changes

5.2. Updating the Aims class

The **Aims** class will create a new **MemoryDaemon** object and run it as a daemon thread

- Open the **main**() method of **Aims** class
- Add code to create a new **MemoryDaemon** object
- Use this object in the constructor of the **Thread** class to create a new **Thread** object
- Using the **setDaemon**() method to indicate that this thread is a daemon thread, add code to start the thread.
- Save your changes

Run the program and see the changes**...**