Object-Oriented Programming

LECTURER: NGUYEN THI THU TRANG, TRANGNTT@SOICT.HUST.EDU.VN
TEACHING ASSISTANTS: NGUYEN T.T. GIANG, GIANG.NTT194750@SIS.HUST.EDU.VN
VUONG DINH AN, AN.VD180003@SIS.HUST.EDU.VN

Lab 06: Polymorphism

In this lab, you will practice with:

- Polymorphism
- Comparable interface to sort objects within a collection
- Template
- Map

You need to use the project that you did with the previous labs including AimsProject.

0 Assignment Submission

For this lab class, you will have to turn in your work twice, specifically:

- **Right after the class**: for this deadline, you should include any work you have done within the lab class.
- **10 PM two days after the class**: for this deadline, you should include your work on all sections of this lab, and push it to a branch called "***release/lab06***" of the valid repository.

After completing all the exercises in the lab, you have to update the use case diagram and the class diagram of the AIMS project.
Each student is expected to turn in his or her work and not give or receive unpermitted aid. Otherwise, we would apply extreme methods for measurement to prevent cheating. Please write down answers for all questions into a text file named **"answers.txt"** and submit it within your repository.

1 Create the `Playable` interface (skip if done in the previous lab)

Requirement:

Now the user can choose to play some media when browsing the list of media in the store or seeing the current cart. For simplicity, we establish the way the system plays a media as follows: When a CD is played, the system displays the CD information (i.e., CD title and CD length) and plays all the tracks of the CD. To play a track, the system displays the track's name and its length. Similarly, a DVD can also be played, i.e., the system displays the title and length of the DVD. If a DVD or track has a length of 0 or less, the system must notify the user that the track, the DVD or the CD of that track cannot be played.

Solution: Implement **`Playable`** interface

Create the **`Playable`** interface that contains a **`play`**() method. The classes of media types which can be played (i.e CompactDisc, DigitalVideoDisc and Track) will implement this interface.
You can apply Release Flow here by creating a **`topic`** branch for implementing the **`Playable`** interface.

The more detailed instruction is below:

- Create a **`Playable`** interface, and add to it the method prototype: **`public void play();`**
- Save your changes
- Implement the **`Playable`** interface in **`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 the **`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 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
  - Loop through each track of the ArrayList and call **`Track's`** play() method

2 Update **`Aims`** class

- You will update the **`Aims`** class along with all your changes above:
  - When the user is browsing the list of media in the store, the software should allow them to play any media they choose if it is playable (e.g. **`CompactDisc`** or **`DigitalVideoDisc`**)
  - When seeing the current cart, the user should also be able to play any media of the type that is is playable (e.g. **`CompactDisc`** or **`DigitalVideoDisc`**).

3 Unique items in a list

Additional requirement:
The list of tracks on a CD should not contain identical objects. For simplicity, the cart also just contains unique items, i.e the quantity of an item is 1.

Solution: Override **`equals()`** method of the **`Object`** class
In previous labs, you add a media to the cart or a track to a CD. You may use the `contains()` methods to check if an object is already in the list.
When calling this by default, `contains()` will check a variety of conditions in order to confirm that 2 objects are identical. However, you can define your own requirements to perform the verification.

The `contains()` method returns true if the list contains a specific element. More formally, it returns true if and only if the list contains at least one element `e` such that `e` such that `(o==null?e==null:o.equals(e))`. So the `contains()` method actually uses the `equals()` method to check equality.

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

- For the `Media` class: the `id` is equal
- For the `Track` class: the `title` and the `length` are equal

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 dealing 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. `id` for `Media`; `title` and `length` for `Track`). If the passing object is not an instance of `Media`, what happens?

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

4 Sort media in the cart

Requirement:

In this part, you are required to add an additional function for the AIMS system which is sorting the media. In the "View cart" use case of the AIMS system, the user can sort all the items in the cart by title and then category in the alphabet sequence.

Solution: Implement the **`Comparable`** interface and ***`compareTo()`*** method

To sort the list of media, we can implement our sorter using some built-in interfaces and deal with the problems.

4.1 Implement the Comparable interface

To sort media products, implement the `Comparable` interface on `CompactDisc`, `DigitalVideoDisc`, `Track`, and `Book`. Add type arguments to `Comparable`, for example, `Comparable<CompactDisc>`.

Note: The `Comparable` interface is part of the Java class library. It is in java.lang package, so no import statement is needed. Please open the Java docs to see the information of this interface. Which method(s) do you need to implement from this interface?

+ For each of `CompactDisc`, `DigitalVideoDisc`, `Track`, and `Book`, edit the class description to include the `Comparable` interface after the implements keyword in the class declaration. For example, the class declaration for `DigitalVideoDisc` becomes:

```java
public class DigitalVideoDisc extends Disc implements Playable, Comparable<DigitalVideoDisc>
```

Note: When you save your classes, you will now receive an error in the Tasks view. This is because classes that implement the `Comparable` interface must implement the `compareTo` method (from `Comparable`) - and this has not yet been completed.

We can apply Release Flow here by creating a topic branch for `Comparable` interface implementation.

*4.2 Implementing **`compareTo()`** method*

- For each of `CompactDisc`, `DigitalVideoDisc`, `Track`, and `Book`, create a method called `compareTo()`. This method will compare two items by title, and if the titles are the same, it will compare the categories in the alphabet sequence, then return the result.

- Each method will have the signature corresponding to the class. For example, the signature for the `compareTo()` method of `CompactDisc` class is: `public int compareTo(CompactDisc obj)`

- When implementing the `compareTo()` method of the `Comparable` interface, you often simply use the `compareTo()` method on one or several of the fields of the class. For example, in the `DigitalVideoDisc` class, you return a value based on the result of `compareTo()` on the `title` and then the `category` fields of the two objects.

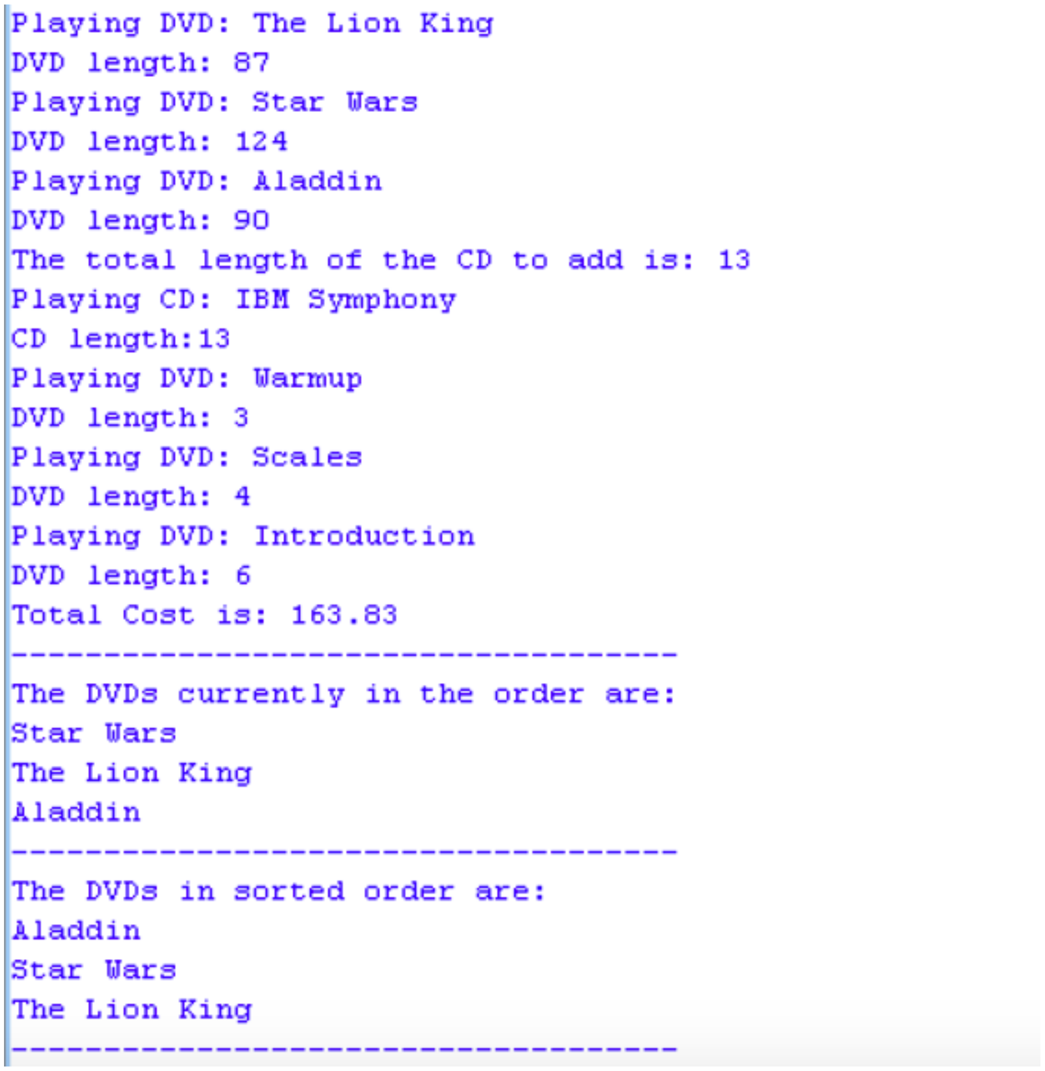
- Since `title` and `category` are fields of `Media` and three of the above `Comparable` classes (`CompactDisc`, `DigitalVideoDisc`, and `Book`) extend `Media`, you can create the same `compareTo()` methods for them. Hence, you should think about the fact that we can move the `compareTo()` method to the `Media` class for re-use, and only the `Media` class needs to implement the `Comparable` interface. If you do that, you can freely remove all implementations of the `compareTo()` method for the `Comparable` interface from the `CompactDisc`, `DigitalVideoDisc`, and `Book` classes.

*4.3 Testing the **`compareTo()`** method*

- In **`hust.soict.dsai.test.media`** or **`hust.soict.hedspi.test.media`** package, create a new class for testing, e.g., `TestMediaCompareTo`.
- Create a collection (for example, an `ArrayList`) and add some `Media` objects to it (for example, some `DigitalVideoDiscs` as the sample code below). Write more codes for other `Media` types.
- Iterate through the entries in the collection and output them.
- Use the `Collection.sort()` method to sort the entries in the collection, and output them again:
  ```
  java.util.Collection collection = new java.util.ArrayList();
  ```
  They should now be in sorted order based on the `compareTo()` method that you created for that class:
- Below is a suggestion:

```
// Add the DVD objects to the ArrayList
collection.add(dvd2);
collection.add(dvd1);
collection.add(dvd3);

// Iterate through the ArrayList and output their titles
// (unsorted order)
java.util.Iterator iterator = collection.iterator();

System.out.println("-----------------------------------");
System.out.println ("The DVDs currently in the order are: ");

while (iterator.hasNext()) {
      System.out.println
      (((DigitalVideoDisc)iterator.next()).getTitle());
}
   // Sort the collection of DVDs - based on the compareTo()
   // method
   java.util.Collections.sort((java.util.List)collection);

   // Iterate through the ArrayList and output their titles -
   // in sorted order
   iterator = collection.iterator();

   System.out.println("-----------------------------------");
   System.out.println ("The DVDs in sorted order are: ");

   while (iterator.hasNext()) {
         System.out.println
         (((DigitalVideoDisc)iterator.next()).getTitle());
   }

   System.out.println("-----------------------------------");
```
Figure 1. TestMediaCompareTo source code

- Execute your program (click the Run button as before).
- The output in the Console view may like the



```
Playing DVD: The Lion King
DVD length: 87
Playing DVD: Star Wars
DVD length: 124
Playing DVD: Aladdin
DVD length: 90
The total length of the CD to add is: 13
Playing CD: IBM Symphony
CD length:13
Playing DVD: Warmup
DVD length: 3
Playing DVD: Scales
DVD length: 4
Playing DVD: Introduction
DVD length: 6
Total Cost is: 163.83
------------------------------------
The DVDs currently in the order are:
Star Wars
The Lion King
Aladdin
------------------------------------
The DVDs in sorted order are:
Aladdin
Star Wars
The Lion King
------------------------------------
```

Figure 2. Output of TestMediaCompareTo

5 Sort media in the cart in multiple ways

Requirement:

In the “View cart” use case of the AIMS system, the user can choose to sort all the items in the cart either by title or by cost:

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

Solution: Use **`Comparator`** to allow multiple sorting ways of **`Media`**:

In reality, there are many cases when users want to sort products based on multiple criteria. The previous exercise with the `Comparable` interface showed you how to implement sorting with one requirement only. In fact, when using the `Comparable` interface, we can only define ONE order for the class, with the `compareTo()` method. To allow multiple orderings of `Media`, we need to use `Comparator`.

Note: The `Comparator` interface is a comparison function, which imposes a total ordering on some collections 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 MediaComparatorByTitleCost implements Comparator<Media>
```

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

- Implement the `compare()` method 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.
- Add the comparators as attributes of the `Media` class:

```
public class Media {

    public static final Comparator<Media> COMPARE_BY_TITLE_COST =
            new MediaComparatorByTitleCost();
    public static final Comparator<Media> COMPARE_BY_COST_TITLE =
            new MediaComparatorByCostTitle();
```

- Pass the comparator into `Collections.sort`:

```
java.util.Collection.sort(collection,Media.COMPARE_BY_TITLE_COST);
```

or

```
java.util.Collection.sort(collection,Media.COMPARE_BY_COST_TITLE);
```

6 Polymorphism with `toString()` method

This exercise gives a better illustration of polymorphism at the behavior level.

Recall that when a user sees the details of a media in the store, the information displayed depends on the type of media, specifically:

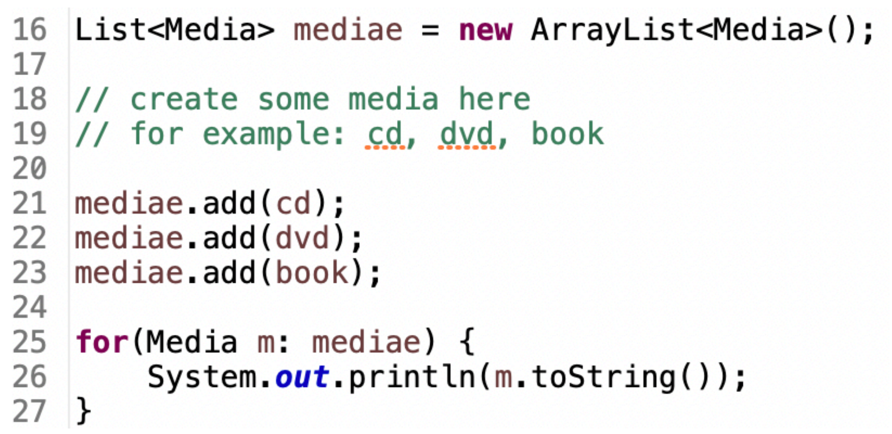
- For books, the system shows their title, category, author list, and content length (i.e., the number of tokens).
- For CDs, the system displays the CD title, category, artist, director, CD length, and the cost for the CD and then displays the information of all the tracks on that CD.
- For DVDs, the system displays DVD title, category, director, DVD length, and the cost for the DVD.

In each class type of media, we implemented a different `toString()` method to print out the information of the object. When calling this method, depending on the type of object, corresponding `toString()` will be performed and different information will be displayed.

Question:

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

Sample code:



```java
List<Media> mediae = new ArrayList<Media>();

// create some media here
// for example: cd, dvd, book

mediae.add(cd);
mediae.add(dvd);
mediae.add(book);

for(Media m: mediae) {
    System.out.println(m.toString());
}
```

Figure 3. Polymorphism sample code

7 Counting the frequency of **`Book`** content with **`Map`**

- We can apply Release Flow here by creating a feature branch for book content processing.
- Add an attribute `String content` for `Book` with two additional attributes:
    - A sorted list `List<String> contentTokens`:
      Hint: you may use `String.split(String separator)` to split the content by the separator, where the separator can be a regex (regular expression).
    - A sorted map `Map<String,Integer> wordFrequency`
- Write a method `processContent()` calculating the following information of the book content. This method is called when the content of the book is set/changed.
    - Split the content into tokens by spaces or punctuations, then sort these tokens from a to z and set them to the `contentTokens` attribute list
    - Count the frequency of each token, sort by tokens from a to z and set it to the `wordFrequency` attribute map *(Hint: use `TreeMap` to get an ordered map).*
- Override the method `toString()` to return all information of `Book`: all values of `Book` attributes, the content length (i.e. the number of tokens), the token list, and the word frequency of the content.
- Write the `BookTest` class in a test package to test all the above methods and display information of `Book`.