CS 46B
Fall 2016
Homework 3

For this assignment you will create a class called Movie and observe its behavior when it is managed by different kinds of “Film Archive” collections.

Due date

This assignment is due on or before 11:59 PM on Monday October 3. Late work will not be accepted.

Getting started

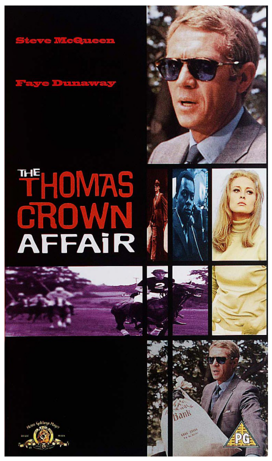
Create a directory called “HW3Workspace” wherever you choose. Start Eclipse. Create a Java project called hw3proj in the workspace. Create a package called movies in the project. All classes and interfaces that you create should go in the movies package.

Class movie

Create a public class `Movie` with private instance variables `String title` and `int year`. The class should declare that it implements the `Comparable<Movie>` interface, and should provide the following:

- A constructor that takes 2 arguments: a String and an int (in that order) for initializing `title` and `year`.
- A method that satisfies the `Comparable<Movie>` interface. Movies should be compared first by title and then by year.

The Maltese Falcon
1941



The Thomas Crown Affair
1968

The Thomas Crown Affair
1999

- An equals() method that is compatible with the method that satisfies the `Comparable<Movie>` interface.
- A toString() method that prints "Movie" followed by 1 space followed by the title followed by 1 space followed by open-parenthesis followed by the year followed by close-parenthesis. Example:

The Maltese Falcon (1941)

- A public static method `getTestMovies()`, which returns an array of 10 unique Movie instances. The 0th and 1st array elements must be 2 movies with the same title but from different years (e.g. The Thomas Crown Affair 1968 and The Thomas Crown Affair 1999, or True Grit 1969 and True Grit 2010). The 2nd and 3rd elements must be 2 movies with different titles but from the same year (e.g. The Martian 2015 and Bridge of Spies 2015). The 4th and 5th elements must be 2 different objects that represent the same movie (same title and same year).
- A hashCode() method. Use the following:

```
public int hashCode()
{
     return title.hashCode() + year;
 }
```

Interface FilmArchive

Create an interface called `FilmArchive` that defines 2 methods:

- getSorted(), which takes no args and returns ArrayList<Movie>. Implementations should return an array list whose members are unique according to deep equality, and sorted according to the criteria in Movie's compareTo() method.
- add(), which takes one arg of type Movie and returns a boolean. If add() is called where the arg already appears in the film archive, the method should return false and otherwise do nothing; if the arg of add() does not yet appear in the archive, it should be added as described below and the method should return true.

3 implementing classes

Create 3 classes, named ListFilmArchive, HashFilmArchive, and TreeFilmArchive, that implement the FilmArchive interface.

Class ListFilmArchive

This class should extend ArrayList<Movie>. In your add() method, check every movie in the array list for deep equality to the arg of add(). If you find a movie that "equals()" the arg, just return false; if you don't find one, add the arg to the array list and return true.

Hint: you are overriding the add() method inherited from the ArrayList superclass. When you detect that the arg movie doesn't appear in the array list, you want to call the superclass' version of add(). Section 9.3 of the textbook explains how to do this.

For the getSorted() method, use a TreeSet<Movie> to do the sorting. You saw how to do this in a recent lecture. First construct a TreeSet<Movie>, passing the ArrayList into the TreeSet constructor. Then construct and return a new ArrayList, passing the TreeSet into the ArrayList constructor.

Class HashFilmArchive

This class should extend HashSet<Movie>. It's ok to add movies to a HashSet because Movie has mutually compatible equals(), and hashCode() methods.

For the add() method, first read the documentation for add() in the java.util.HashSet API page (https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html). Convince yourself that this method does exactly what add() in HashFilmArchive should do. Since the inherited method is acceptable, you don't need to create add() in HashFilmArchive. (But you do need to understand *why* you don't need to).

For the getSorted() method, do something similar to what you did in ListFilmArchive.

Class TreeFilmArchive

This class should extend TreeSet<Movie>.

For the add() method, first read the documentation for add() in the API page for java.util.TreeSet (https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html). Convince yourself that this method does exactly what add() in TreeFilmArchive should do. Since the inherited method is acceptable, you don't need to create add() in TreeFilmArchive.

For the getSorted() method, do something somewhat similar to, but simpler than, what you did in ListFilmArchive and HashFilmArchive.

Testing your 3 FilmArchive classes

To test your classes, you will do what professional programmers do in the real world: Create test inputs, think about what outputs those inputs will cause, run the code, look at the output, and see if your expectations were met. If they were met, that's great, on to the next problem. If not, either your expectations or your implementation were wrong. Thinking this way is the only way to write consistently correct code.

You already have test inputs: the array returned by Movie.getTestMovies(). Be sure that this array meets the requirements described above.

To test ListFilmArchive class, add the following main() method:

```
public static void main(String[] args)
{
     ListFilmArchive archive = new ListFilmArchive();
     for (Movie m: Movie.getTestMovies())
          archive.add(m);
     for (Movie m: archive)
          System.out.println(m);
 }
```

Run ListFilmArchive as an app. Look at the number of movies that were printed out, and their order. Are these what you expected?

To test HashFilmArchive, add the same main() to HashFilmArchive but change "ListFilmArchive" to "HashFilmArchive". Add the same main() to TreeFilmArchive but change "TreeFilmArchive" to "TreeFilmArchive". This is a common way to test individual classes in Java: you give each class a main() method that tests just that class.

Let's do science

So far in this class you have done programming. It's time to do science. Computer science, like all science, includes making observations and looking for patterns. An entire branch of computer science involves determining how slow or fast an algorithm will be. A novel algorithm, even if it's brilliant, isn't worth very much if it won't complete execution on practical data during your lifetime. It isn't enough for software to be correct; it also has to be efficient. Expect a question on that last sentence on your next midterm.

In lecture you learned that inserting into a HashSet is extremely fast, while similar functionality based on an ArrayList is increasingly slow as the list expands. It's time to experience this for yourself.

Create the following class:

```java
package movies;

public class HashAnalyzer
{
	public static void main(String[] args)
	{
		HashFilmArchive hashArchive = new HashFilmArchive();
		for (int i=0; i<100000; i++)
		{
			hashArchive.add(new Movie("Movie" + i, 2016));
			if (i%1000 == 0)
			    System.out.println(i);
		}
	}
}
```

The main() method creates 100,000 fake movies and puts them into a HashFilmArchive. Every 1000 insertions, it prints out a message. When you run the app, the time interval between messages will give you feedback about time performance of the add() method of HashFilmArchive. Now run the app. Do you notice anything about the interval between messages? Is it long or short? Does it increase as the archive grows?

Create a second analyzer class called ListAnalyzer. It should be just like HashAnalyzer, except it should use a ListFilmArchive. Run the app. Now what do you notice about the interval between messages.

On Tuesday or Wednesday Oct 4 or 5 (that is, *after* the deadline for this assignment) post a message to Piazza. State what you observed when you ran HashAnalyzer and ListAnalyzer. Post into the hw3 folder.

Submission

As usual, export your project as a JAR file (not an Executable JAR file) and upload to Canvas.