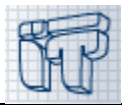


FIRST NAME:
LAST NAME:
NIA:
GROUP:

First midterm exam

Second Part: Problems (7 points out of 10)

Duration: 90 minutes
Highest score possible: 7 points
Date: March 15, 2019

Overall instructions for the exam:

- Books, notes, mobile phones, as well as other electronic devices are not allowed during the exam. Breaking this rule may result in expulsion from the examination
- Complete your personal information before starting the exam.
- The exam must be filled in with blue or black pen. The final version of the exam must not be filled in with pencil.

Problem 1 (5 / 7 points)

A well-known media-services provider has requested the help of our team to improve the performance of their service. They want to reorganize the way they handle the media content. They have defined three basic types of media: *films*, *series* and *episodes*.

- Every media content has two basic characteristics: `name` and `category`.
- Each *film* stores some additional information: duration of the film (in minutes) and the budget of its production. This class is already implemented.
- Each *series* stores the total number of episodes (`numberOfEpisodes`) and an `ArrayList` containing all the episodes (`episodes`). The maximum number of episodes per series is 10.
- Each *episode*, which is an independent media file by itself, has a duration in minutes (`duration`), and an `ID` (a unique identifier assigned in incremental order).

The following piece of code shows some examples of how the objects of this classes are created (our solution must be compatible with this piece of code)

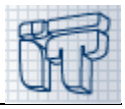
```
Episode episode1 = new Episode("Winter is coming","Action",55);
Episode episode2 = new Episode("The king's road","Action",57);
ArrayList <Episode> episodes = new ArrayList<Episode>();

episodes.add(episode1);
episodes.add(episode2);

try {
        Series series1 = new Series("Game of Thrones","Action", episodes);
} catch (FullSeriesException e) {
        System.err.println("Maximum number of episodes is 10");
        System.exit(-1);
}
```

Section 1.1 (0.75 points)

Implement the class `Media`. This class represents each individual media content offered by the company. The characteristics of this class should be visible only for its subclasses. This class also declares one method: *computeTotalDuration()* which returns the total duration of a media content. Nevertheless, the specific behavior of this method depends on the kind of media we refer to.

Section 1.2 (1 points)

Implement the class `Episode`. This class represents media files which are episodes of a certain series. The characteristics of this class should be visible only from this class. However, a "*SET method"* needs to be



implemented for the *duration* attribute. In this case, the method *computeTotalDuration()* return directly the duration associated to that episode. In the example above, the *episode1* is named "Winter is coming", it belongs to "Action" category and has a duration of 55 minutes.

Section 1.3 (0.5 points)

Implement the class `FullSeriesException`. This class represents the exception which should be thrown when trying to create a series with more than 10 episodes and when trying to add a new episode to a series which already has 10 episodes.

Section 1.4 (2.75 points)

Implement the class `Series`. This class represent media files which are series. The total number of episodes and the `ArrayList` containing the episodes should only be visible from this class. In this case, the method *computeTotalDuration()* returns the sum of the duration of the episodes which are part of the series. You are also requested to program the method that adds a new episode to a series *addNewEpisode()*. In the example above, the *series1* is named "Game of Thrones", it belongs to "Action" category and the episodes which are part of the series are included in the `ArrayList` named *episodes*.

NOTE 1: Take into account the maximum number of episodes per series and make use of the exception you have programmed in the previous section.
NOTE 2: Do not care about importing the class `ArrayList<E>`.
NOTE 3: Some of the methods of `ArrayList<E>`, which may be useful for this section are:

- `boolean add(E e)`
- `void add(int index, E element)`
- `E get(int index)`
- `int indexOf(Object o)`
- `boolean isEmpty()`
- `E set(int index, E element)`
- `int size()`

Problem 2 (2 / 7 points)

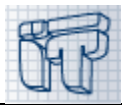
You are employed in the new airline "UC3M Airlines", member of the UniversityWorld Alliance. In this airline, frequent clients can become members its frequent-flier programme UC3M+, where you can get points each time you travel to get discounts, VIP Lounge accesses, priority boarding, etc. A workmate has implemented the class `PointsCalculator`. This class has method to compute the points each client gets for each flight depending on the miles and the flight class. The code of this class is as follows:

```
public class PointsCalculator {
   public static int getPointsFlight(String
flightClass, int miles){
       int points = 0;
       if (miles >= 2000 && miles < 6000){
              points = 200;
       } else if(miles >= 6000 ){
              points = 300;
       } else {
              points = 100;
       }
       if(flightClass.equals("Business")){
           points = points * 2;
       }
       else if(flightClass.equals("Premium
Economy")){
           points = (int)(points * 1.25);
       }
       return points;
       }
}
```

You are asked to develop some initial test methods with limited coverage using the **minimum lines** of code as possible for each test. If any test is impossible to carry out as specified, justify why it is not possible.

a) The first test should achieve a branch coverage of exactly 50% in class PointsCalculator (0.55p)
b) The second test should achieve a method coverage of exactly 50% in class PointsCalculator (0.55p)
c) The third test should achieve a line coverage of 100% in method getPointsFlights(). (0.55 p). Can this test also be used for a black-box test? Justify your answer in 1-2 lines. (0.35p)

NOTE: Do not care about the class declaration and imports. Just implement the three methods.



REFERENCE SOLUTIONS (several solutions are possible)

PROBLEM 1

Section 2.1 (0.75 points)

```
public abstract class Media {
	protected String name;
	protected String category;

	public Media (String name, String category){
		this.name = name;
		this.category = category;
	}

	public abstract int computeTotalDuration();
}
```

Section 2.2 (1 points)

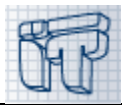
```
public class Episode extends Media{

	private int duration;
	private int ID;
	public static int counter = 1;

	public Episode(String name, String category, int duration)  {
		super(name, category);
		this.duration = duration;
		this.ID = counter++;
	}

	public void setDuration (int duration){
		this.duration = duration;
	}

	public int computeTotalDuration() {
		return duration;
	}
}
```

Section 2.3 (0.5 points)

```
public class FullSeriesException extends Exception {
	public FullSeriesException(String msg) {
		super(msg);
	}
}
```

Section 2.4 (2.75 points)

```
import java.util.ArrayList;
public class Series extends Media {

	private int numberOfEpisodes;
	private ArrayList <Episode> episodes;

	public Series(String name, String category, ArrayList <Episode> episodes) throws
FullSeriesException {
```



```
		super(name, category);
		if(episodes.size()>10){
			throw new FullSeriesException("Series with more than 10 episodes");
		}
		this.numberOfEpisodes = episodes.size();
		this.episodes = episodes;
	}
	public int computeTotalDuration() {
		int totalDuration = 0;
		for(int i = 0; i<episodes.size();i++){
			totalDuration += episodes.get(i).computeTotalDuration();
		}
		return totalDuration;
	}
	public void addNewEpisode(Episode newEp) throws FullSeriesException  {
		if(episodes.size()==10){
			throw new FullSeriesException("Full series");
		}
		episodes.add(newEp);
		this.numberOfEpisodes++;
	}

}
```

Section 1.1 (0.75 points)

- 0.15: Class declaration
- 0.15: Attribute declaration as protected
- 0.25: Constructor initializing the attributes
- 0.20: Abstract method computeTotalDuration()
- Significant errors are subject to additional penalties

Section 1.2 (1 point)

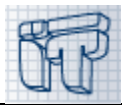
- 0.15: Class declaration extending from `Media`
- 0.15: Static attribute to implement the counter
- 0.40: Constructor calling the constructor of the superclass, initializing duration and updating the ID counter.
- 0.15: Setter needed to retrieve the value of the attribute *duration* from other classes
- 0.15: Method computeTotalDuration()
- Significant errors are subject to additional penalties

Section 1.3 (0.5 points)

- 0.15: Class declaration extending from `Exception`
- 0.35: Constructor calling the constructor of the superclass and passing a string as parameter.
- Significant errors are subject to additional penalties

Section 1.4 (2.75 points)

- 0.25: Class declaration extending from `Media` and attributes (including here the use of Episode as type for the ArrayList)
- 1.00: Constructor
  - 0.25: Signature including `throws`
  - 0.15: Calling the constructor of the superclass
  - 0.30: Checking the number of episodes and throwing the exception (if size>10)
  - 0.15: Calculation of the number of episodes
  - 0.15: Initialization of episodes
- 0.50: Method computeTotalDuration()



  - 0.05: Method signature
  - 0.20: Loop
  - 0.25: Correct access to the duration of each episode and performance of the sum
- 1.00: Method addNewEpisode()
  - 0.25: Method signature including `throws`
  - 0.30: Checking the number of episodes and throwing the exception (if size>10)
  - 0.25: Addition of the element to the ArrayList
  - 0.20: Updating variable *numberOfEpisodes*
- If GET method is implemented in class Episode, penalize 0.1 as it is the same as computeTotalDuration()
- Significant errors are subject to additional penalties

PROBLEM 2

```
@Test
public void testA() { // Branch coverage 50%
	assertEquals(PointsCalculator.getPointsFlight("Premium Economy", 7000), 375);
}

@Test
public void testB() { // Method coverage 50%
	assertEquals(PointsCalculator.getPointsFlight("Business", 3000), 400);
}

@Test
public void testC() { // Line coverage 100% in method
	assertEquals(PointsCalculator.getPointsFlight("Business", 3000), 400);
	assertEquals(PointsCalculator.getPointsFlight("Premium Economy", 7000), 375);
	assertEquals(PointsCalculator.getPointsFlight("Tourist", 500), 100);
}
```

You cannot use this test for a black-box test because it does not cover all the possible equivalence classes (e.g., Business with less than 2000 miles is not tested in the example).

Section 2.2 (1.5 points)

- 0.55 each test
  - 0.05: Method declaration with @Test
  - 0.15: assertEquals are right
  - 0.10: Coverage achieved is correct
  - 0.25: Coverage is achieved with the minimum number of lines. If coverage is incorrect, this part is incorrect regardless the number of lines.
- 0.35 final question in part c. If the answer is correct but not the justification, maximum 0.1
- Significant errors are subject to additional penalties