Student Feedback Report

Student Id	999022072
Exam Id	202502071048241
Exam Date	Friday, Feb 7, 2025
Course Id	202401-104824-1
Course Name	INTRODUCTION TO SYSTEMS PROGRAMMING - A
Lecturer	Nazareno AGUIRRE

Open question score	Original Exam Grade	**Final Exam Grade**
95.00	95.00	**95.00**

Summary

Question number	Comments	Actual points	Max points
1		7.00	7.00
2		3.00	8.00
3	Perfect	15.00	15.00
4	Perfect	18.00	18.00
5		17.00	17.00
6		18.00	18.00
7		17.00	17.00

Guangdong Technion
Israel Institute of Technology
广东以色列理工学院

(34)
ID 999022072
Exam 202502071048241

Introduction to Systems Programming - 104824

Exam A
February 7, 2025

Your ID Number:

9	9	9	0	2	2	0	7	2

First name and Surname: Xinyu Ma

Guidelines

1. **Duration: 3 hours**. Use of calculators, personal dictionaries, electronic devices, reference material, personal notes or any other extra material is not allowed.
2. Provide explanatory details for each of your solutions. Answers without appropriate explanations and justifications will receive low or no credit. Provide clear and complete answers in the space provided for answers to each of the exercises.

1. (a) Explain what is a *class*, and what is its relationship to the concept of *object*. What programming elements constitute a class definition? (7p)

   (b) What is a class-based type (also known as an object type)? How does a class-based type differ from a primitive or basic type in Java? (8p)

Answer

7
(1)

(a) class is a blueprint that represents the entity with its properties and methods that work on it.
object is the entity that represents a class.
class is constituted by field, constructor and methods.

3
(2)

(b) class-based type is an instance that represents this class, and it owns all properties the class has and all methods work on this class.
class-based type needs to be created by new, and it's not originally in Java, it needs a class as a blueprint or imported the package.

A class-based type is a type defined by a Java class. When a class is declared, such class defines a type, the type that is associated with all the objects of the class. Basic types, on the other hand, are the few datatypes that are directly represented by values; these include int, boolean, float, etc. A main difference between class-based types and basic types is in how assignment works; assignment to variables of basic types operates as value copying; assignment to variables of class-based types works as reference copying, thus leading, e.g., to aliasing (different variables referring to the same runtime object).

2. Consider the Java implementation of a music playlist, composed of two classes, and shown in Figures 1 and 2. A playlist is represented as a list of track objects. Each track has a title, an artist, and a duration. Your task is to implement method `void removeArtist(String artist)` from class `Playlist`. As described in the comments, this method must remove from the playlist all the tracks of a given artist, the artist received as a parameter of the method. (15p)

```
/**
 * A playlist track, consisting of a title, artist and duration.
 */
public class Track {

    // title of the track
    private String title;

    // artist of the track
    private String artist;

    // duration of the track (in seconds)
    private int duration;

    // constructor of the track
    public Track(String title, String artist, int duration) {
        assert title != null && artist != null && duration > 0;
        this.title = title;
        this.artist = artist;
        this.duration = duration;
    }

    public String getTitle() {
        return this.title;
    }

    public String getArtist() {
        return this.artist;
    }

    public int getDuration() {
        return this.duration;
    }
}
```

Figure 1: A Java class that represents tracks of a playlist.

Answer

```
public void removeArtist(String artist) {
    tracks.removeIf(s -> s.getArtist().equals("artist"));
}
```

15
(3)
Perfect

3. Consider the music playlist implementation given in Figures 1 and 2. Moreover, consider the following method of class `Playlist`, which computes the total duration of the playlist, by adding the durations of the tracks:

```
/**
 * Computes the total duration of the playlist, by adding up the
 * durations of all the tracks in the playlist.
 * @return the total duration of the playlist
 */
public int totalDuration() {
    int duration = 0;
    for (Track t: tracks) {
        duration = duration + t.getDuration();
    }
    return duration;
}
```

Your task is to reimplement this method to achieve exactly the same behavior, but using *streams*, *map* and *reduce*. (18 p)

Answer

```
public int totalDuration() {
return tracks.stream()
        .map(s -> s.getDuration())
        .reduce(0, (acc, dura) -> acc + dura);
}
```

18
(4)
Perfect

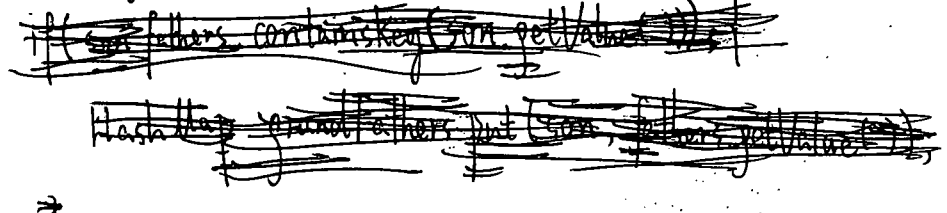
4. Consider the following Java method signature:

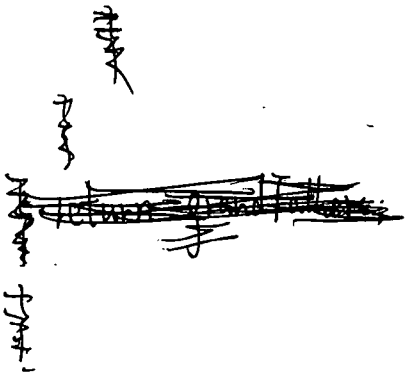
```
import java.util.HashMap;

...
/**
 * Computes the grandfathers mapping of individuals, given the fathers
 * relation represented as a map.
 * @param fathers contains the father of each individual, represented
 * as a map.
 * @return a map containing the grandfather of each individual, given
 * the father mapping received as parameter.
 */
public static HashMap<String, String> grandFather(HashMap<String, String> fathers) {

    ...


}
```

This static method takes as a parameter a map from strings to strings, containing parenthood information: for a set of names of individuals (the keys), the map provides their corresponding fathers (the values). Your task is to implement this method, which builds a map with the *grandfathers* relationship: if an individual x has father y, and y has father z, then x has grandfather z. (17 p)

Answer

```
public static HashMap<String, String> grandFather(HashMap<String,
String> fathers) {
                                        // there exists a space
    HashMap<String, String> grandFathers = new HashMap<String, String>();
    for(String son: fathers.keySet()) {
```





```
    if (fathers.containsKey(fathers.getValue(son))) {
        grandFathers.put(son, fathers.getValue(fathers.getValue(son)));
    }
  }
  return grandFathers;
}
```

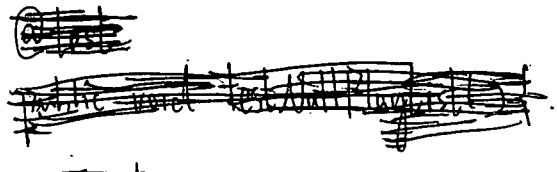
17
(5)

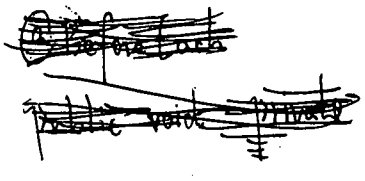
5. Consider the music playlist implementation given in Figures 1 and 2. Moreover, consider the following method of class `Playlist`, which returns the first track of the playlist:

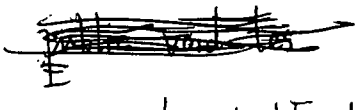
```
/**
 * Returns the first track of the playlist.
 * If the playlist is empty, it returns null.
 */
public Track firstTrack() {
    if (tracks.isEmpty()) {
        return null;
    }
    else {
        return tracks.get(0);
    }
}
```

Write two JUnit unit tests to test the behavior of method `firstTrack()`. If you need to add some method or methods to `Playlist` or `Track` in order to complete the tests, do so as part of this exercise. (18 p)

Answer







```
public class testFirstTrack() {
    private Playlist playlist;

    @BeforeEach
    public testFirstTrack() {
        playlist = new Playlist();
    }
```

```
@test
private void testNullPlayList() {
    assertEquals(null, playlist.firstTrack());
}

@test
private void testFilledPlaylist() {
    Track track1 = new Track(a, Jay Chou, 340);
    Track track2 = new Track(b, Eason, 770);
    playlist.addTrack(track1);
    playlist.addTrack(track2);
    assertEquals(track1, playlist.firstTrack());
}
}
```

missing quotes

18
(6)

6. Consider the Java implementation of posts for a social network application. As part of this implementation, we have classes `Post` and `MessagePost`, shown in Figures 3 and 4. The `toString()` method of class `Post` shows the post information in the following format:

```
<id>: <title> (submitted <timeStamp>)
```

Your task is to implement method `toString()` of class `MessagePost` so that the message post information is shown in the following format:

```
<id>: <title> (submitted <timeStamp>)
Contents: <message>
```

Implement this method. Do not modify class `Post`. (17 points)

Answer

```
public String toString( ){
    String output = super.toString( );
    StringBuilder stringBuilder = new StringBuilder( );
    stringBuilder.append(output);
    stringBuilder.append("\nContents: ");
    stringBuilder.append(this.message);
    return stringBuilder;
}
```

17
(7)

```
import java.util.ArrayList;

/**
 * A music playlist, consisting of a list of tracks.
 */
public class Playlist {

    //the list of tracks
    private ArrayList<Track> tracks;

    //default constructor
    public Playlist() {
        tracks = new ArrayList<Track>();
    }

    /**
     * Adds a track to the end of the playlist
     */
    public void addTrack(Track newTrack) {
        // assume this method is implemented.
    }

    /**
     * Removes from the playlist all the tracks of a given artist.
     * All tracks whose artist is the parameter of the method must be
     * removed from the playlist. All other tracks (of artists other
     * than the parameter) must be maintained in the playlist.
     * @param artist is the artist to remove from the playlist.
     */
    public void removeArtist(String artist) {
        //TODO: Implement this method
    }

}
```

Figure 2: A Java class representing music playlists.

```
/**
 * Class Post represents a general post of a social network.
 */
public class Post {

    // id of the post
    private int id;

    // timestamp of the post
    private int timeStamp;

    // title of the post
    private String title;

    /**
     * Constructor of class Post
     */
    public Post(int id, int timeStamp, String title) {
        assert id >= 0 && timeStamp >= 0 && title != null;
        this.id = id;
        this.timeStamp = timeStamp;
        this.title = title;
    }

    /**
     * Produces a string representation of the state of the object
     */
    public String toString() {
        String output = id + ": " + title + " (submitted " + timeStamp + ")";
        return output;
    }

}
```

Figure 3: A Java class that represents general posts of a social network application.

```
/**
 * Class MessagePost represents a message post of a social
 * network. This class extends the general post class.
 */
public class MessagePost extends Post {

    // message text
    private String message;

    /**
     * Constructor of class MessagePost
     */
    public MessagePost(int id, int timeStamp, String title, String message) {
        super(id, timeStamp, title);
        assert message != null;
        this.message = message;
    }


    /**
     * Produces a string representation of the state of the object
     */
    public String toString() {
        //TODO implement this method

    }

}
```

Figure 4: A Java class representing message posts of a social network application.