Exam DIT042 / DIT043:
Object-Oriented Programming

2023-10-23 - PM (Lindholmen)

Examiner:	Francisco Gomes de Oliveira Neto
Questions:	+46 31 772 69 51 Francisco will visit the exam hall at ca 15:00 and ca 16:30.
Results:	Will be posted within 15 working days.
Grades:	00–49 points: U (Fail) 50–69 points: 3 (Pass) 70–84 points: 4 (Pass with merit) 85–100 points: 5 (Pass with distinction)
Allowed aids:	No aids (books or calculators) are allowed.

Please observe carefully the instructions below. Not following these instructions will result in the deduction of points.

- Write clearly and in legible English (illegible translates to "no points"!). Answers that require code must be written using the Java programming language.
- Motivate your answers, and clearly state any assumptions made. Your own contribution is required.
- Before handing in your exam, **number and sort the sheets in task order!** Write your anonymous code and page number on every page! The exam is anonymous, **do not** leave any information that would reveal your name on your exam sheets.
- When answering what is being printed by the program, make sure to write as if the corresponding answer were being printed in the console.
- You can assume that all classes in the code presented in this exam are in the **same package**. Similarly, **you can assume** that all classes used in the code (Exceptions, Lists, Streams, etc.) are properly imported in their respective file.
- For **all class diagrams** in your exam, write all methods and attributes relevant to your code, including **constructors, getters and setters**.

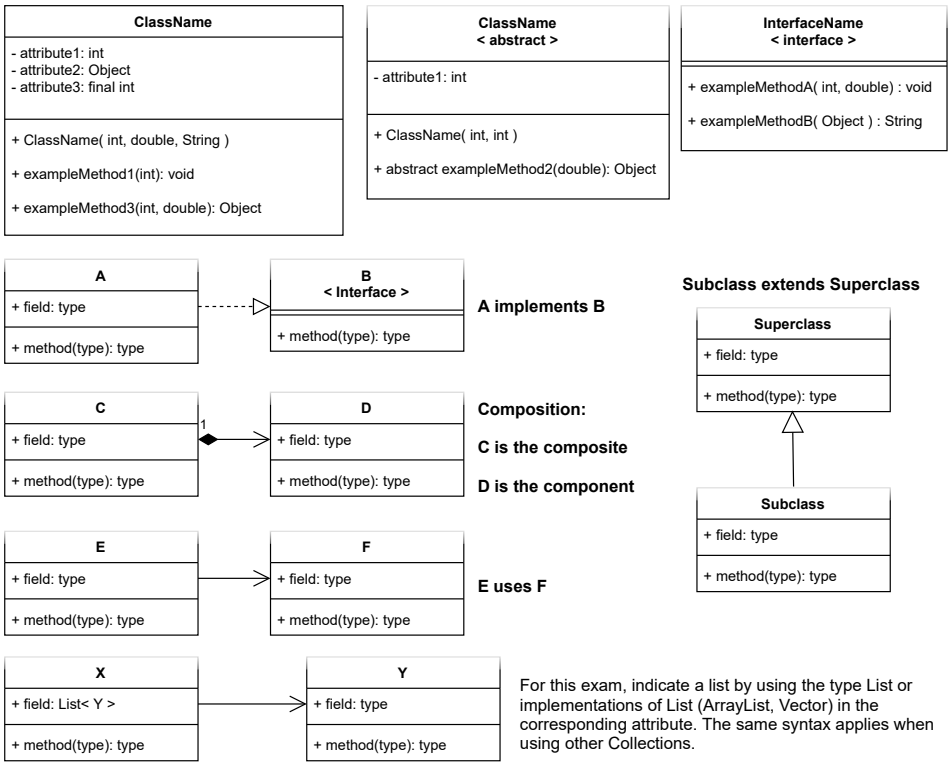

Figure 1: The elements of a class diagram. For this exam, you do not need to represent Exceptions in your diagrams. Using '-' indicates that the attribute or method are private, whereas '+' means public.

Table 1: List of useful methods from the List interface and String class.

Class	**Method specification**	**Description**
List	add(E e): boolean	Appends the specified element to the end of the collection.
List	contains(Object o): boolean	Returns true if this collection contains the element.
List	get(int index): Object	Returns the element at the specified index.
List	isEmpty(): boolean	Returns true if this collection contains no elements.
List	size(): int	Returns the number of elements in this collection.
String	length(): int	Returns the number of characters in the string.
String	isEmpty(): boolean	Returns a boolean value indicating whether the string is empty.
String	contains(String s): boolean	Returns a boolean indicating whether the specified string 's' is a substring of the current string.
String	startsWith(String s): boolean	Returns a boolean indicating whether the current string starts with the specified string 's'.
String	charAt(int i): char	Returns the character at position 'i' in the String.
String	substring(int start, int end): String	Returns a subset from the string with characters from the start index until the end index (not included).

Question 1 [Total: 15 pts]: Write **exactly** what is printed when running the Java program below. **Important:** The **ordering and the formatting is important**. If you want to skip printing lines, you should number your printed lines to indicate what you skipped.

```
1  public class Point {
2      static int x;
3      int y;
4      int z;
5      public Point(int x, int y, int z) {
6          this.x = x;
7          this.y = y;
8          this.z = z;
9      }
10     public String toString() {
11         return "x= " + x + " y= " + y + " z = " + z;
12     }
13 }
14 public class PointMain {
15     public static void mixContent(Point p1, Point p2) {
16         int tempX = p1.x;
17         int tempY = p1.y;
18         int tempZ = p1.z;
19         p1.x = p2.x;
20         p1.y = p2.y;
21         p1.z = p2.z;
22         p2.x = tempX;
23         p2.y = tempY;
24         p2.z = tempZ;
25     }
26
27     public static void mixContent(int p1, int p2) {
28         p1 = p1 % 2 == 0 ? p1 + 5 : p2 + 2;
29         p2 = p1 + p2;
30         System.out.println("p1 = " + p1 +  " p2 = " + p2);
31     }
32
33     public static void main(String[] args) {
34         Point p1 = new Point(1, 3, 4);
35         Point p2 = new Point(7, 8, 10);
36
37         p1.x = p2.x + 10;
38         System.out.println(p2);
39
40         Point p3 = new Point(6, 1, 0);
41
42         mixContent(p1, p3);
43         mixContent(p2, p3);
44
45         System.out.println(p1);
46         System.out.println(p2);
47         System.out.println(p3);
48
```

```
49
50          mixContent(p1.y, p3.z);
51          p1.x = p2.x + 10;
52          p1.z = p2.z + 5;
53          System.out.println(p1);
54
55          p2 = p3;
56          p3 = p1;
57          p1 = p2;
58          System.out.println(p1);
59          System.out.println(p2);
60          System.out.println(p3);
61
62          mixContent(p2, p3);
63          System.out.println(p1);
64          mixContent(p2.z, p3.z);
65
66          p2 = new Point(p2.y, p3.y, p1.y);
67          System.out.println(p1);
68          System.out.println(p2);
69          System.out.println(p3);
70
71          p3 = new Point(2, 6, 7);
72          mixContent(p1.x, p3.y);
73      }
74  }
```

Question 2 [Total: 40 pts]: You are hired by a company that creates a platform for competitive online gaming. Your predecessor created the code for the Player abstractions. Using your knowledge of Object-oriented programming and design answer the questions below:

Specification (Player): All players have a username to represent their ID in the system, a name, their preferred role, a number of victories and losses. The roles can be: Attacker, Healer or Defender. When creating players, we must specify the username, the name and their preferred role. The victories and losses always begin at zero. Once created, the username of the Player cannot change. Victories and losses should never be negative, and can only change by increasing 1 to their value when the player wins or loses a game. Two players are equal if they have the same username. When printed, the players should show: `<name> (V: <victories>, L: <losses>) Role: <role>`

```
1  public class Player {
2
3    private final String username;
4    private String name;
5    private int victories;
6    private int losses;
7    private String role;
8
```

```
9   public Player(String username, String name, int victories,
10                int losses, String role) throws Exception {
11    if(username.isEmpty() || name.isEmpty() || role.isEmpty()) {
12      throw new Exception("Player name, username or role cannot be empty.");
13    }
14    this.username = username;
15    this.name = name;
16    this.victories = victories;
17    this.losses = losses;
18    this.role = role;
19  }
20
21  public String getUsername()  { return username;}
22
23  public String getName()  { return name;}
24  public int getVictories()  { return victories;}
25  public int getLosses()  { return losses;}
26  public String getRole()  { return role;}
27
28  public void setName(String name)  { this.name = name;}
29  public void setVictories(int victories)  { this.victories = victories;}
30  public void setLosses(int losses)  { this.losses = losses;}
31  public void setRole(String role)  { this.role = role;}
32
33  // TODO: Implement the equals method.
34 }
```

2.1 [7 pts]: Implement the equals method for the Player class.

2.2 [8 pts]: Is the Player class well encapsulated? Justify your answer.

2.3 [15 pts]: Write two suggestions to improve the Player class. Explain how your code improves or fixes problems in the Player abstraction.

Customer requests: The customer asks for a new type of player: a Ranked Player. The Ranked player are similar to the regular player, but Ranked Players should have a victory rate showing how often they win instead of losing (`rate = victories / total matches`). The rate must be **truncated** to two decimal digits.
Ranked players are classified based on their victory rate using the categories below (note: the term 'between' means that both values are included in the interval). When printed, the players should show: `<name> (V: <victories>, L: <losses>). Victory rate: <rate> Rank: <rank>`

- Diamond rank: Between 95% and 100% rate.
- Gold rank: Less than 95% and greater than or equal to 70%.
- Silver rank: Less than 70% and greater than or equal to 50%.
- Bronze rank: Less than 50% and greater than or equal to 20%.

- Noob rank: Less than 20% victory rate. When created, all ranked players should be in the Noob rank.

2.4 [10 pts]: Write the code for the Ranked Player abstraction. When reusing members from the Player class, you can assume that your suggestions mentioned in Q2.3 have been implemented in the Player class.

Question 3 [Total: 35 pts]: You are asked to expand the online gaming software by adding a Leaderboard that stores and operates with players. Your team's architect designed the Leaderboard (see diagram in Figure 2)). Your should finish the implementation of the Leaderboard, and integrate it with the Players abstractions.

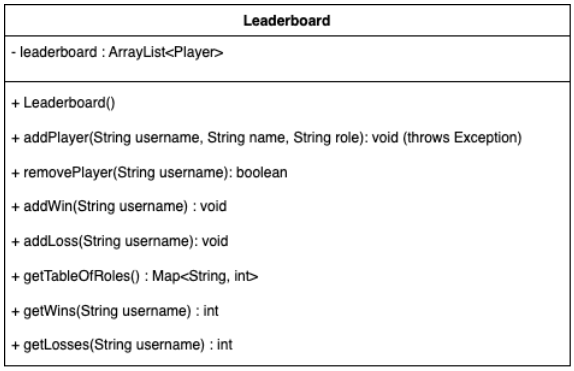

Figure 2: Class diagram for the Leaderboard abstraction.

Specification (Leaderboard): The leaderboard stores the players registered in the game. We can add player objects, or remove them from the leaderboard based on their username. The leaderboard also triggers changes in a player's number of victories or losses. The leaderboard should also return a table (as a HashMap) including the number of players registered in each of the roles. For instance, if there are 3 attacker players, 2 healers and no defenders, the resulting map should be as shown below:

Healers →	2
Attacker →	3

3.1 [5 pts]: Is the Leaderboard class well encapsulated? Justify your answer.

3.2 [10 pts]: Write the code for the method `getTableOfRoles()`.

3.3 [15 pts]: Write a complete class that connects your Leaderboard to the Player and Ranked Player (Question 2). You **must also** justify the choice of **each** relationship between the classes in your design.

3.4 [5 pts]: Explain what changes are required in the code to allow your Leaderboard objects to be saved in files.

Question 4 [Total: 10 pts]: (From the book Cracking the Coding Interviews). **String compression:** Implement a function to perform basic string compression using the counts of repeated characters. For example, the string `"aabcccccaaa"` as input, the method would return `"a2b1c5a3"`. If the "compressed" string would not become smaller than the original string, your method should return the original string. You can assume that the String is non-empty, not null and has only lowercase letters (a–z).

The code will be evaluated based on correctness, readability, and efficiency. Below we include a few more examples of input and expected output.

- `compress("axccccccccc") = "a1x1c8"`.
- `compress("axccc") = "axccc"`.
- `compress("hihihihi") = "hihihihi"`.
- `compress("wwweeeeeerrrrrk") = "w3e6r5k1"`.
