Exam DIT042 / DIT043: Object-Oriented Programming

2023-01-04 - 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.
Reviewing:	2023-01-20 11:00–12:00,
	Jupiter building, 4th Floor, Room 400, Lindholmen

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**.

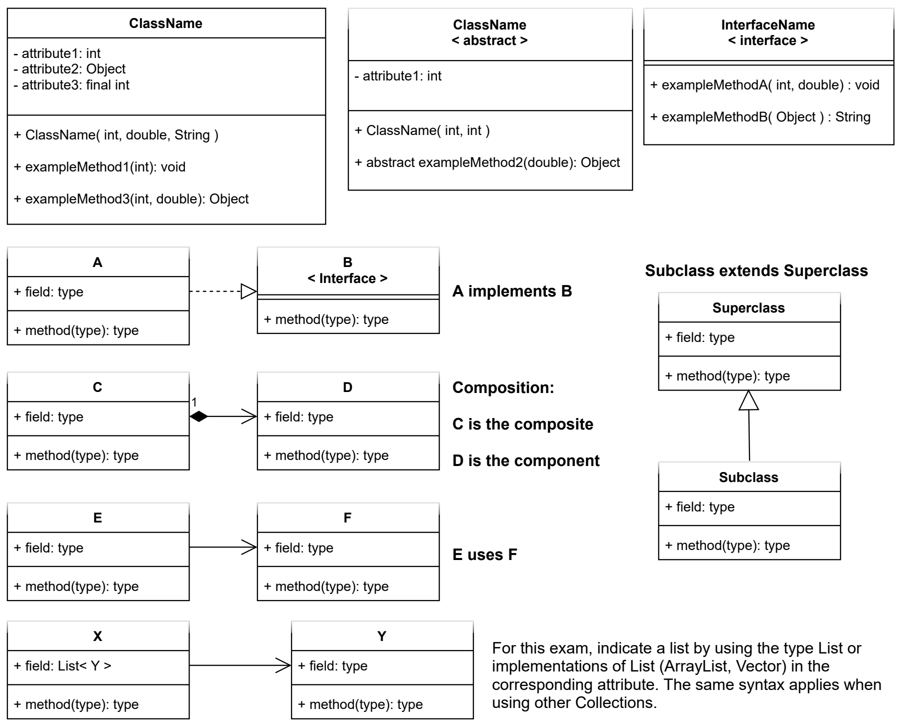

Figure 1: The elements of a class diagram. For this exam, you do not need to represent Exceptions in your diagrams.

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	endsWith(String s): boolean	Returns a boolean indicating whether the current string ends with the specified string ′s′.
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: 20 pts]: Based on the code below, answer the questions:

Q1.1 [8 pts]: Regarding the class `PointMain`: Your team members have different opinions regarding the relationship between the methods `mixContent(Point p1, Point p2)` and `mixContent(int p1, int p2).` Alex says that one method **overrides** the other. Charlie says that one method **overloads** the other. Who is correct, Alex or Charlie? Justify your answer by (i) explaining each concept, and (ii) specifying which one is used in the class below.

Q1.2 [12 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.

```java
public class Point {
  int x;
  int y;
  static int total = 0;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
    total = total + this.y;
  }
  public String toString() {
    return "x= " + x + " y= " + y;
  }
}
public class PointMain {
  public static void mixContent(Point p1, Point p2) {
    int tempX = p1.x;
    int tempY = p1.y;
    p1.x = p2.x;
    p1.y = p2.y;
    p2.x = tempX;
    p2.y = tempY;
  }
  public static void mixContent(int p1, int p2) {
    p1 = p1 % 2 != 0 ? p1 + 5 : p1 + 10;
    p2 = p1 + p2;
    System.out.println("p1 = " + p1 +  " p2 = " + p2);
  }
  public static void main(String[] args) {
    Point p1 = new Point(-2, 3);
    Point p2 = new Point(7 , 8);

    p1.total = p2.total + 10;
    Point p3 = new Point(6, 1);

    mixContent(p1, p3);
    mixContent(p2, p3);


    System.out.println(p1);
    System.out.println(p2);
    System.out.println(p3);
    System.out.println("total= " + p3.total);

    mixContent(p1.x, p3.x);
    p2.total = p1.total + 5;
    System.out.println("total= " + p1.total);

    p1 = p2;
    p2 = p3;
    p3 = p1;
    System.out.println(p1);
    System.out.println(p2);
    System.out.println(p3);

    mixContent(p2, p3);
    p2 = new Point(p2.x, p3.y);
    System.out.println(p1);
    System.out.println(p2);
    System.out.println(p3);
  }
}
```

Question 2 [Total: 45 pts]: You were hired to write the code for a software that records temperature data over the year. Based on your knowledge of algorithms and OOP, answer the questions below. Your algorithms will be evaluated based on correctness, readability and efficiency:

Specification: The WeatherRecorder stores a list of TemperatureData that, in turn, represents a temperature data point. A data point has a temperature value (in Celsius) and the corresponding month (String) when the data was collected. Once measured, an individual temperature data point cannot be changed. **Important:** You can also assume that the string values for months will never be **null**.

Q2.1 [5 pts]: The code below has a problem (bug). Explain: i) What is the problem in the code?; ii) What are the consequences of this problem in the execution of the program? You can use the main method to illustrate your explanation; iii) How to fix the problem?

```java
public class TemperatureData {

  final double temperature;
  final String month;

  public TemperatureData(double temperature, String month) {
    this.temperature = temperature;
    this.month = month;
  }

  public String toString(){
    return String.format("%s: %.2f", month, temperature);
  }
}

public class WeatherRecorder {
  static final String[] MONTHS = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

  private List<TemperatureData> temperatures;

  public boolean registerData(double temperature, String month){
    // implement this method ...
  }
  public String getSmallestAbsDifference(){
    // Implement this method ...
  }
  public String getLowestTemperatureData(){
    // Implement this method ...
  }

  public double getTempAt(int index) throws Exception {
    if(index < 0 || index >= temperatures.size()) {
      throw new Exception(temperatures.size() + " measures registered. " +
                          "Index " + index + " is out of bounds.");
    } else {
      return temperatures.get(index).temperature;
    }
  }
}

public class WeatherMain {
  public static void main(String[] args) {
    try{
      WeatherRecorder sensor = new WeatherRecorder();

      System.out.println(sensor.registerData( 22.0, "Jun")); // true
      System.out.println(sensor.registerData( 30.0, "Jun")); // true
      System.out.println(sensor.registerData(-20.0, "Dec")); // true
      System.out.println(sensor.registerData( 13.0, "Mun")); // <- false.
      System.out.println(sensor.registerData(  5.0, "Jan")); // true
      System.out.println(sensor.registerData(  0.0, "Feb")); // true


      System.out.println(sensor.getTempAt(3));               // 5.0
      System.out.println(sensor.getLowestTemperatureData()); // "Dec: -20.00"

      // Output: "Diff: 5.0. Between Jan: 5.00 and Feb: 0.00"
      System.out.println(sensor.getSmallestAbsDifference());

      System.out.println(sensor.registerData( -5.0, "Nov")); // creates a tie!

      // Tied case! Output: "Diff: 5.0. Between Jan: 5.00 and Feb: 0.00"
      System.out.println(sensor.getSmallestAbsDifference());

      System.out.println(sensor.registerData(  5.0, "Oct")); // true

      // Output: "Diff: 0.0. Between Jan: 5.00 and Oct: 5.00"
      System.out.println(sensor.getSmallestAbsDifference());

    }catch(Exception exception){
      System.out.println(exception.getMessage());
    }
  }
}
```

Important: The questions below assume that you fixed the problem in Q2.1.

Q2.2 [8 pts]: Implement the method `registerData(double temperature, String month)`: Creates and add a temperature data point to the list of temperature **if** the specified month matches one of those in the array `MONTHS`. Returns a boolean indicating whether the temperature measure was successfully created and stored.

Q2.3 [5 pts]: Implement the method `getLowestTemperatureData()`: Returns a string representation of the month that had the lowest temperature measured. The string should be formatted: `"<month>: <temperature>"`. Should return an empty string if no temperatures have been registered yet.

Q2.4 [10 pts]: Implement the method `getSmallestAbsDifference()`: Returns a string with the **smallest absolute difference** of temperatures registered. You should indicate the corresponding temperature data with that difference. Some important points:

- Use the absolute difference since $|a - b| == |b - a|$. In Java, `Math.abs(a - b)` returns the $|a - b|$.
- Return the following string: ”`Diff: <smallestDiff>. Between <data1> and <data2>`". See examples in the main method.
- In case there are less than two temperature data registered, your method should throw an exception with the message: ”Invalid request. We require more than 2 temperatures registered.”

- In case of a tie (i.e., two data points have the same absolute temperature difference), you should print the first pair found.
- See the main methods for two examples indicating the expected output of the method.

Q2.5 [5 pts]: Your boss asks you to implement a method `removeAt(int index)` that will remove the temperature data point at the specified index. Your colleague Jane says that such method cannot be implemented because the TemperatureData object is immutable. Do you agree or disagree with Jane? Justify your answer.

Q2.6 [7 pts]: What is the relationship between the WeatherRecorder and TemperatureData. Justify your answer by referring to lines of code that reflect that relationship.

Q2.7 [5 pts]: Your boss wants your program to save the temperature information in files but you cannot modify the TemperatureData class. Your colleague Alex, Charlie and Sam have suggestions. Alex suggests you to use an Object output stream to save the list of temperature data points in a file, Charlie suggests you to use a text stream for saving the data points as strings, and Sam suggests you to use a Scanner. For each suggestion, **explain and justify** if they are appropriate solutions.

Question 3 [Total: 35 pts]: SOLID is a set of five principles to improve the design of OO programs. The five principles are: **S**ingle-responsibility principle, **O**pen-closed principle, **L**iskov substitution principle, **I**nterface segregation principle, and the **D**ependency inversion principle. Using your knowledge of OOP and the SOLID principles, answer the question below.

Specification: The code below is part of a game. This game has champions that have a name and health points (HP). The HP of the champions can never be less than or equal to zero, or greater than the set maximum for all champions (100 HP). A champion can also receive damage or heal its HP based on a specified amount. Besides the default Champion, there are also special types of champions with different outcomes on their healing and damage behavior. Check the `toString()` method to see how Champions are printed.

Q3.1 [5 pts]: Your colleague Charlie says that the class Champion below is not well encapsulated. Do you agree or disagree with Charlie? Justify your answer.

```java
public class Champion {
    private static final int MAX_HP = 100;
    private int currentHealth;
    private String name;
    private final String type;

    public Champion(String name, String type) {
        this.currentHealth = MAX_HP;
        this.name = name;
        this.type = type;
    }

    public void receiveDamage(int amountDamage){
        if(type.equals("Tank")) {
            amountDamage = amountDamage / 2;
        }
        if(amountDamage >= currentHealth) {
            this.currentHealth = 0;
        } else {
            this.currentHealth = this.currentHealth - amountDamage;
        }
    }

    public void healHero(int amountHealed){
        if(type.equals("Assassin")) {
            amountHealed = amountHealed / 2;
        }
        int afterHeal = amountHealed + currentHealth;
        this.currentHealth = afterHeal >= MAX_HP ? MAX_HP : afterHeal;
    }

    public String toString() {
        return name + " has " + this.currentHealth + " HP left.";
    }
    public int getCurrentHealth() {
        return currentHealth;
    }
    public void setCurrentHealth(int currentHealth) {
        this.currentHealth = currentHealth;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

There are special types of champions. Besides the default type, the code above implements two special types of champions: Tank and Assassin. Your boss asks you to implement the highly anticipated Undead champion. All special types of champion behaviour are specified below:

- **Tank** champions reduce the incoming damage amount by half.
- **Assassin** champions heal only for health of the specified amount.
- **Undead** champions will heal whenever receiving damage, and will receive damage whenever healing. The amount of damage/heal is not changed for Undead champions. When printed, an undead champion will also have the word ”Undead” before their name. For instance: `"Undead Francisco has 35 HP left."`

Q3.2 [6 pts]: Which of the SOLID patterns can be used to improve the Champion class. You must: (i) justify your choice, and (ii) describe what changes can improve the design of the Champion class.

Q3.3 [10 pts]: Choose **other two** principles from SOLID and, for each, (i) explain that principle, and (ii) indicate how we can identify whether that principle is being followed or violated. (**Note:** Must be two principles **different** than the principle chose in Q3.2)

Q3.4 [7 pts]: Write the class diagram of your improved design. You must consider your modifications to include the Undead champion type.

Q3.5 [7 pts]: Write the code for your abstraction of the Undead champion.