TEST
Software Systems: Programming

course code: 202001024
date: 28 January 2021
time: 13:45 – 17:45

General

- You may use the following (unmarked) materials when making this test:
  - Module manual.
  - Slides of the Programming topics.
  - The book
    David J. Eck. *Introduction to Programming Using Java*. Version 8.1.2, December 2020.
  - A dictionary of your choice.
  - IntelliJ and/or Eclipse.
  - Documentation of Java 11
    `https://docs.oracle.com/en/java/javase/11/docs/api/`

  You are *not* allowed to use any of the following:
  - Solutions of any exercises published on Canvas (such as recommended exercises or old tests);
  - Your own materials (copies of (your) code, solutions of lab assignments, notes of any kind, etc.).
- When you are asked to write Java code, follow code conventions where they are applicable. Failure to do so may result in point deductions. It is recommended that you use IntelliJ and/or Eclipse to write code. Remember you can copy code from Remindo to IntelliJ and/or Eclipse.
- You do *not* have to add Javadoc or comments, unless explicitly asked to do so. Invariants, preconditions and postconditions should be given only when they are explicitly asked.
- You are not allowed to leave the room during the first 30 minutes or the last 15 minutes of the exam.
- Place your student ID card on the table as well as documentation that grants extra time (if applicable).

Question 1 (5 points)

Describe the difference between **overriding** and **overloading** with respect to the method signature. Give an example of overriding and an example of overloading to illustrate your explanation.

Question 2 (7 points)

a. *(4 pts)* Explain the difference between unit testing and integration testing and describe an example of unit testing and an example of integration testing.

b. *(3 pts)* Imagine you find a bug in your program. How do you approach solving the bug if you use **test driven development**?

Question 3 (8 points)

Consider the following Java classes:

```java
public class A {
    private A() {
    }
    public final int a() {
        return 4;
    }
}
public class ExampleClass extends A {
    protected final int a() {
        return b();
    }
    private final int b() {
        return c();
    }
    protected static int c() {
        return 4;
    }
    public final static void main(String[] args) {
        A ex = (A) new ExampleClass();
        (ExampleClass) ex.b();
    }
}
```

This code fragment has four compile-time and/or runtime errors. Explain these errors by referring to what is wrong, why that is wrong, and how it could be fixed.

Question 4 (10 points)

Consider the following situation. You find the source code of the ”firmware” (a software system) running inside a water sensor. It has a web-based interface, protected by a simple password. In the source code, you find the following declaration:

```java
private static final String adminPasswordSHA256 =
        "e5f0d2bc06f4abc1cdcc51b99159a8285b9bf83aeb70e682bb168448d96613eb";
```

From this, you figure that the SHA-256 cryptographic hash function is used to protect the password.

a. *(3 pts)* Why is it a bad idea to use SHA-256 for this purpose? What is a better alternative?

From a reliable source, you hear that the admin password for this device is very simple. The length is always 4 or 5 characters long and only a single upper or lower character is used. So, examples of passwords include: ”AAAA”, ”AAAAA”, ”FFFFF”, ”wwww” and ”ccccc”. Write a small program that tries to find the admin password. To get you started, we provide you with the following two snippets of Java code:

```
public static String hex(byte[] bytes) {
    StringBuilder result = new StringBuilder();
    for (byte aByte : bytes) {
        result.append(String.format("%02x", aByte));
    }
    return result.toString();
}

MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] hash = digest.digest(text.getBytes(StandardCharsets.UTF_8));
```

b. *(7 pts)* Give both your program and the password of the device.

Question 5 (10 points)

Imagine we have a file like the following example:

```
10
930
1116
218
491
682
178
905
192
558
283
851
```

Write a method **void** `processFile(String filename)` that reads all numbers from a file like the example and finds any two numbers a and b such that $a+b=2021$. The method must print these two numbers to standard output. For the example file these would be the numbers 1116 and 905. You don’t need to print all possible numbers a and b, just the first one is enough.

Question 6 (10 points)

Imagine we are playing a guessing game. In this question, we are going to guess a number between 0 and 100. We make two classes, `Guesser` and `Challenger`. The Challenger comes up with a random number between 0 and 100. The guesser must repeatedly try to guess the number, and the reply is either 0 (you guessed correct), 1 (your guess was too high) or -1 (your guess was too low).

a. *(5 pts)* Implement a class `Challenger` which gets a random number between 0 and 100 (including 0 but excluding 100) in the constructor. You can use the `Random.nextInt` method to obtain such a number. Implement a method **int** `guess(`**int** `number)` which returns -1, 0 or 1 according to the description.

b. *(5 pts)* Implement a class `Guesser` which can solve the challenge. Implement a recursive method `int solve(Challenger challenger, int lowest, int highest)` which guesses a number `x` where `lowest <= x && x <= highest`. Your implementation must be recursive and may not contain any loops.

Question 7 (20 points)

In this question, we are concerned with ordering the perfect pizza. Assume we have the following toppings:

```
public enum Toppings {
    ONIONS,
    MOZZARELLA,
    SALAMI,
    MUSHROOMS,
    SALMON,
    OLIVES,
    PINEAPPLE,
    SPINACH,
    PEPPERS,
    CHILI,
    GARLIC
}
```

We want to calculate the **subjective value** of a pizza: how good do we estimate that a pizza with certain toppings would taste? Example rules are:

- contains both onions and salami: multiply value by $1.25\times$
- contains mushrooms: multiply value by $1.2\times$
- contains pineapple: multiply value by $0.2\times$

If we consider a pizza with onions, salami and pineapple, and we start with an initial value of 10.0, application of the rules would result in the value 12.5 after applying the first rule, then 12.5 after applying the second rule, and finally 2.5 after the third rule.

a. *(2 pts)* Define an interface `ToppingRule` with a method `double apply(Collection<Topping> toppings, double value)`. The purpose of the `apply` method will be to check if the rule matches the given toppings, and if so, to apply the rule to the given value, returning the result.

b. *(3 pts)* Define an implementation of `ToppingRule` called `OneToppingRule` that checks for **one** topping and applies a (`double`) modifier if the topping is present. Don't forget to include an appropriate constructor.

c. *(3 pts)* Define an implementation of `ToppingRule` called `TwoToppingsRule` that checks of **two** toppings and applies a (`double`) modifier if both toppings are present. Don't forget to include an appropriate constructor.

d. *(3 pts)* Implement a class `ToppingRules`. This class must have some data structure to store your `ToppingRule`s. Implement a method `void addRule(ValueModifier rule)` which you will use to add your rules.

e. *(3 pts)* Implement a method in the `ToppingsRules` class called `double evaluate(Collection<Topping> toppings)` that applies the rules to a collection of toppings and returns the result. As the initial value, use the number 10.

f. *(6 pts)* A new pizza place has opened right around the corner, and it allows selecting exactly three toppings from the list of toppings. Write a program that finds the best combination of toppings, according to the rules added to a `ToppingsRules` object and prints the result to standard output. You must include the given example rules above, but are free to add additional rules of your choice.

Question 8 (30 points)

You are asked to make a management system for managing the inventory of pizza toppings.

a. *(7 pts)* Implement a class `Inventory` that maintains the number of 'units' of each topping. Initially, there are 10 units of each topping available. Implement a method `void refill(Topping topping, int amount)` that adds the given amount of units of the given topping in the inventory. Implement a method `boolean take(Topping topping)` which tries to take one unit of the specified topping, if possible. It must return `true` if successful, and `false` otherwise.

b. *(5 pts)* Define appropriate preconditions and postconditions for `refill`. Pay attention that your postcondition should describe the change to the state of the `Inventory` object after a call to `refill`.

c. *(5 pts)* After some soul searching, you decide that you actually want to throw an exception if there is no topping available, instead of returning `false`. Define a new exception `NoToppingsLeftException` for this purpose and modify `take` to return nothing and instead throw the exception if no toppings are available. Give both the newly defined exception and the modified `take` method as your answer.

d. *(7 pts)* You suddenly remember that the inventory object will be manipulated by many different threads. Modify your implementation to ensure that no race conditions occur, i.e., `refill` and `take` always run as expected. Explain why your implementation is thread-safe and describe what can happen without your modifications. Give the modified `take` and `refill` methods as your answer.

e. *(6 pts)* Modify `take` to only return when a topping has been taken. That means that if there is no topping available, your implementation should wait until the topping has been refilled by another thread. Use an appropriate method to do this. Give the modified `take` and `refill` methods as your answer.