FIRST NAME:
LAST NAME:
NIA:
GROUP:

First midterm exam

Second Part: Problems (7 points out of 10)

Duration: 70 minutes
Highest score possible: 7 points
Date: March 23, 2023

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.

Problem 1 (3 / 7 points)

A leading logistics company has asked you to develop a program for managing a warehouse. The classes for this program are the following (assume all getters and setters are already implemented for all classes):

- Product (`Product`). It is a type of good that can be distributed from the warehouse. It has a name (`name`) of type `String`. It also has additional information consisting of the selling price in euros (`pricePerUnit`) and the country in which it has been manufactured (`country`). It also has a category (`category`) which can be identified by one of these four letters: f-FOOD, s-SUPPLIES, e-EQUIPMENTS, and m-MISCELLANY.

- Product list (`ProductList`). It is a data structure that in addition to a list of products that is declared as `ArrayList<Product> list`, stores additional information on all these products, such as their total cost (`totalCost`), the total selling price (`totalPrice`), or the total benefit obtained from its distribution in the market (total selling price minus total cost) (`totalBenefit`). All these values take into account not only the different products on the list but also the number of units available for each of them.

Here you can find some of the methods of class `ArrayList` which you may need:

```
public E get(int index)

public E set(int index,E element)

public boolean add(E e)

public boolean remove(Object o)

public int size()
```

Section 1.1 (0.25 points)

Declare a class `ProductException` and write the code for the constructor. This exception will be used in the following section.

Section 1.2 (0.75 points)

Declare the class `Product` and write the code for a constructor which initializes all the attributes. The constructor should only create the object if `category` has a valid value, throwing a `ProductException` otherwise.

Section 1.3 (1 point)

Write the code for the method `public int countProducts(char category)` in class `ProductList`. This method returns the number of products belonging to the category defined by the argument `category` which are stored in the product list.

Section 1.4 (1 point)

Due to the current high cost of petrol, the store needs to increase the prices of the product which have not been manufactured in Spain. To do this you must write the code for the method `public void updatePrice()` in class `ProductList`. This method must add a 10% extra cost to all products whose country is not Spain. On the other hand, the total price of the products in the product list should also be updated.

Problem 2 (4 / 7 points)

You have been tasked to create a Java program that models a library's payroll system using object-oriented principles. The program should include four classes

- The `LibraryEmployee` class is an abstract class that represents an employee in a library. It contains three private variables: `name, ID, and salary,` which are used to identify and compensate the employee. It has a constructor that takes values for these variables and an abstract method called calculatePay() that calculates the employee's pay.

  It also has getters for name, ID, and salary, and a setter for salary that throws a custom exception SalaryTooLowException if the salary is set below $25,000

- `The LibraryAssistant` class is a subclass of the `LibraryEmployee` that represents an assistant employee in the library. It has instance variables for `name, ID, salary,` and `hoursWorked`, as well as a constructor that takes parameters for each of these variables. The class also implements the abstract calculatePay() method from the parent class to calculate the assistant's pay based on their hourly rate and hours worked.

Section 2.1 (0,5 point)

Declare the class `LibraryEmployee` and the variables for `name, ID, and salary`. Write the code for the constructor, the `setSalary()` and the abstract method `calculatePay()`.

Section 2.2 (0.5 point)

Declare the class `LibraryAssistant` and write the code for method `calculatePay()`

Section 2.3 (1 point)

Implement the class `PayrollSystem` with main method that creates an ArrayList of LibraryEmployee objects, including a Librarian and a LibraryAssistant.

It then attempts to set the Librarian's salary to $20,000, which is below the minimum allowed salary and therefore throws a SalaryTooLowException. The program catches this exception and prints the message.

It´s no necessary implement the class `SalaryTooLowException`

Section 2.4 (2 points)

This code is a set of JUnit test cases for the `LibraryEmployee, Librarian, LibraryAssistant,` and `PayrollSystem` classes.

```
public class LibraryPayrollTest {
    private ArrayList<LibraryEmployee> employees;
    private LibraryEmployee librarian;
    private LibraryEmployee libraryAssistant;

    @BeforeEach
    void setUp() {
        employees = new ArrayList<>();

        librarian = new Librarian("John Smith", 123456, 40000.0, "Adult Fiction");
        employees.add(librarian);

        libraryAssistant = new LibraryAssistant("Jane Doe", 654321, 20.0, 30);
        employees.add(libraryAssistant);
```

In the @BeforeEach method, the test creates an ArrayList of `LibraryEmployee` objects, a Librarian object, and a `LibraryAssistant` object to use in the test cases.

Implement the test cases:

- `testLibrarianDepartment()`: asserts that the Librarian object has the correct department. **(0,5 points)**

```
@Test
    void testLibrarianDepartment() {
        // TO DO}
```

- `testLibraryAssistantHoursWorked()`: asserts that the LibraryAssistant object has the correct hours worked. **(0,5 points)**

```
@Test
    void testLibraryAssistantHoursWorked() {
      // TO DO}
```

- `testSalaryTooLowException()`: tests that the SalaryTooLowException is thrown when attempting to set a salary below $25,000 for a LibraryEmployee object. **(0,5 points)**

```
@Test
  void testSalaryTooLowException() {
    // TO DO}
```

- `testCalculatePay()`: asserts that the calculatePay() method correctly calculates the pay for a Librarian and a LibraryAssistant object. **(0,5 points)**

```
@Test
  void testCalculatePay() {
      // TO DO}
```

REFERENCE SOLUTIONS (several solutions are possible)

PROBLEM 1

Section 1.1 (0.25 points)

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

Evaluation criteria

- 0 if the code makes no sense.
- 0.05 if class declaration is OK.
- 0.2 if constructor is OK.
- Significant errors are subject to additional penalties.

Section 1.2 (0,75 points)

```
public class Product {
	private String name;
	private char category;
	private String country;
	private double pricePerUnit;

	// f-FOOD, s-SUPPLIES, e-EQUIPMENT, m-Miscellanea
	public static final char FOOD = 'f';
	public static final char SUPPLIES = 's';
	public static final char EQUIPMENTS = 'e';
	public static final char MISCELLANY = 'm';

	public Product(String name, char category, String country,
			double pricePerUnit) throws ProductException {

		this.name = name;
		this.country = country;
		this.pricePerUnit = pricePerUnit;

		switch (category) {
		case FOOD:
		case SUPPLIES:
		case EQUIPMENTS:
		case MISCELLANY:
			this.category = category;
			break;
		default:
			throw new ProductException(
					"The category must be: 'f' FOOD, 's' SUPPLIES, 'e'
EQUIPMENT, 'm' Miscellanea");
		}
	}

}
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.1 if class declaration is OK.
- 0.1 if the attributes declaration is OK (they must be private).
- 0.2 if the setting of the values of all the attributes is OK.
- 0.2 if the exception's throw is OK.
- 0.15 if the constructor throws the exception in the declaration, or a block try-catch is declared.
- Significant errors are subject to additional penalties.

Section 1.3 (1 point)

```
public int countProducts(char category) {

	int total = 0;

	for (int i = 0; i < list.size(); i++) {
		if (list.get(i).getCategory() == category)
			total++;
	}

	return total;
}
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.25 if the loop to traverse through the ArrayList is OK.
- 0.25 if the access of the elements in the ArrayList is OK.
- 0.25 if the comparison of category is OK.
- 0.25 if the counter is correctly incremented and returned.
- Significant errors are subject to additional penalties.

Section 1.4 (1 point)

```
public void updatePrice() {

	for (int i = 0; i < list.size(); i++)
		if (!list.get(i).getCountry().equals("Spain")) {
			Product current = list.get(i);

			current.setPricePerUnit(current.getPricePerUnit() * 1.1);
			totalPrice = totalPrice + current.getPricePerUnit();
		}
	}
```

Evaluation criteria:

- 0 if the code makes no sense.
- 0.25 if the loop to traverse through the ArrayList is OK.
- 0.25 if the access of the elements in the ArrayList is OK.
- 0.25 if the comparison of the String is OK (0 if equals is not used to compare the Strings).
- 0.25 if the total price is correctly updated.
- Significant errors are subject to additional penalties.

PROBLEM 2

Section 2.1 (0.75 points)

```
abstract class LibraryEmployee {
    private String name;
    private int ID;
    private double salary;

    public LibraryEmployee(String name, int ID, double salary) {
        this.name = name;
        this.ID = ID;
        this.salary = salary;
    }

    public void setSalary(double salary) throws SalaryTooLowException {
        if (salary < 25000.0) {
            throw new SalaryTooLowException("Salary cannot be set below $25,000");
        }
        this.salary = salary;
    }

    public abstract double calculatePay();
}
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.15 if class and variables declaration is OK.
- 0,25 if constructor is OK.
- 0,25 if set Salary is OK
- 0,1 if abstract method is OK
- Significant errors are subject to additional penalties.

Section 2.2 (0.5 points)

```
public class LibraryAssistant extends LibraryEmployee {
    private int hoursWorked;

    @Override
    public double calculatePay() {
        return getSalary() * hoursWorked;
    }
}
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.25 if class and variables declaration is OK.
- 0,25 if method calculatePay() is OK
- Significant errors are subject to additional penalties.

Section 2.3 (1 points)

```
public class PayrollSystem {
    public static void main(String[] args) {
        ArrayList<LibraryEmployee> employees = new ArrayList<>();
        employees.add(new Librarian("Jane Smith", 1234, 40000.0, "Reference"));
        employees.add(new LibraryAssistant("John Doe", 5678, 20, 30));

        try {
            employees.get(0).setSalary(20000.0);
        } catch (SalaryTooLowException e) {
            System.out.println(e.getMessage());
        }

    }
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.25 if ArrayList is created OK
- 0,25 if the object Librarian and LibraryAssistant is created OK
- 0,5 manage the Exception is OK try and catch
- Significant errors are subject to additional penalties

Section 2.4 (2 points)

```
    @Test
        void testLibrarianDepartment() {
            assertEquals("Adult Fiction", ((Librarian) librarian).getDepartment());
        }
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.25 if call assertEquals with correct parametres
- 0,25 if cast and call method getDepartament()
- Significant errors are subject to additional penalties

```
        @Test
        void testLibraryAssistantHoursWorked() {
            assertEquals(30, ((LibraryAssistant) libraryAssistant).getHoursWorked());
        }
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.25 if call assertEquals with correct parametres
- 0,25 if cast and call method getDepartament()
- Significant errors are subject to additional penalties

```
@Test
void testSalaryTooLowException() {
    Exception exception = assertThrows(SalaryTooLowException.class, () -> {
        librarian.setSalary(20.0);
    });}
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.25 if call assertThrows with correct parametres
- 0,25 if cast and call method setSalary() with the correct parameter
- Significant errors are subject to additional penalties

```
@Test
void testCalculatePay() {
    assertEquals(600.0, libraryAssistant.calculatePay());}
```

Evaluation criteria

- 0 if the code makes no sense.
- 0.25 if call assertEquals with correct parametres
- 0,25 if cast and call method calculatePay()
- Significant errors are subject to additional penalties