United International University (UIU)

Dept. of Computer Science & Engineering (CSE)

Midterm Exam, Trimester: Fall 2023

Course Code: CSE-1115, Course Title: Object Oriented Programming

Total Marks: 30, Duration: 1 Hour 45 Minutes

Any examinee found adopting unfair means will be expelled from the trimester / program as per UIU disciplinary rules.

Q1: (a) Logarithms are mathematical functions of the form $\log_b x$ which consists of two parts: base (int b) and argument (double x). The logarithmic value is calculated using the formula: [3+3]

$$\log_b x = \frac{\log_e x}{\log_e b}$$

where $\log_e x$ can be evaluated by Java code as Math.log(x).

You have to write a class **Logarithm** and include the **member** variables b and x**,** different types of **constructors** and a function **myfunc()** to evaluate $\log_b x$ such that the **Main** generates the output provided in the following table:

public class Main {
public static void main(String[] args)
{
Logarithm log1 = new Logarithm(2, 9);
Logarithm log2 = new Logarithm(log1);
Logarithm log3 = new Logarithm();
System.*out*.println(log1.b +" "+log1.x+" "+log1.myfunc());
System.*out*.println(log2.b +" "+log2.x+" "+log2. myfunc());
System.*out*.println(log3.b +" "+log3.x+" "+log3. myfunc());
}
}
	Output
	
	2 9.0 3.0
	
	2 9.0 3.0
	
	0 0.0 0.0

(b) What is the output of following java codes:

```
public class Animal {
   public String color;
   public String name;
   public Animal() {
      System.out.println("Default animal");
      color = "Unknown";
   }
   public void showNameColor() {
      System.out.println("Color is: "+ color+" Name is: "+
name);  }
   {
       System.out.println("Animal instance initialization ");
   }
}
```

```
public class Pokemon extends Animal
{
   public String name = "Pikachu";
   public String color = "Red";
}
public class AnimalTest {
   public static void main(String[] args) {
      Animal defaultAnimal = new
                Animal();
      Animal pk = new Pokemon();
      defaultAnimal.showNameColor();
      pk.showNameColor();
   }
}
```

Q2: (a) Consider the following codes [3+3]

```
public class BankAccount {
    public String name;
    private String account_id;
    private double balance;
    BankAccount(String name,double balance,
                    String gender){
        this.name=name;
        this.balance=balance;
        this.account_id=gender+"-"+name;
    }
    //Add necessary codes here
}
```

```
class Test{
    public static void main(String[] args) {
        BankAccount b=new
                        BankAccount("Mr.Rahman",1000, "M");
        System.out.println("Account id:"+b.account_id);
        System.out.println("balance before:"+b.balance);
        b.balance = b.balance - 2000;
    }
}
```

Rewrite the codes after correcting the errors by adding necessary getter/setter methods to access private variables. ***Do not change access modifiers*** of member variables. While updating the balance, a condition that ***balance cannot be less than 0*** should be considered.

(b) Consider the following codes:

```
public class PizzaShop {
    private int pizza_price=320;
    private int drink_price=40;
    private int fries_price=100;
    PizzaShop(){
        // Write necessary codes here
    }
 //Write necessary codes here
}
```

```
class Order{
    public static void main(String[] args) {
        PizzaShop p=new PizzaShop();
        p.order(2,4);
        p.order(1,2,1);
        p.order(3);
    }
}
```

Write the necessary **missing codes** so that the following output is produced when the program runs:

```
Welcome to pizza shop
You ordered 2 pizzas, 4 drinks
Total bill: 800 taka
You ordered 1 pizzas, 2 drinks, 1 fries
Total bill: 500 taka
You ordered 3 pizzas
Total bill: 960 taka
```

Q3: Consider the class named *Vehicle*. [1+1+2 +1+1]

```
class Vehicle {
    private String make;
    private String model;
    public Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }
    public void start() {
        System.out.println("[Vehicle] The vehicle is starting.");
    }
```

```
    public void stop() {
        System.out.println("[Vehicle] The vehicle is stopping.");
    }
    public void drive() {
        System.out.println("[Vehicle] The vehicle is moving.");
    }
}
```

Create a class named *Car* that inherits the Vehicle class.

The Car class must have the following attributes/methods:

- An additional attribute named *numberOfDoors* (data type: int, access modifier: private).
- A *constructor* that receives the values of make (String), model (String), numberOfDoors (int) as arguments and initializes the attributes.
- A *method* that overrides the method *drive()* of the parent class. This method invokes *drive()* of the parent class first and then prints "[Car] The car is moving.".
- Another method named *honk()* (with access modifier: public and return type: void) that prints "[Car] Honk! Honk!"
- Now, consider the class named Main and write the output.

```
public class Main {
    public static void main(String[] args) {
        Vehicle car1 = new Car("make001", "model001", 4);
        car1.drive();
        car1.honk();
    }
}
```

Q4: Think about your own startup **Uthao** which is a ride sharing program using Java. Now due to our country's rules and regulations, vehicles on your platform must abide by the ***speedLimit*** at 80 km/h. [2+1.5+1.5+1]

(a) *Create a class* name **Ride** that contains an attribute ***speedLimit*** which

- Cannot get changed by its child classes
- Cannot get changed once assigned
- Will contain an Integer value
- Must be **static**

Now create the following child classes of **Ride:**

- Bike
- Car
- Microbus

Each child of Ride (that is, Bike, Car and Microbus) will receive a penalty for exceeding the ***speedLimit***.

All the child classes have the following

initial_speed and ***acceleration***:

- Bike: initial_speed 20 km/h , acceleration 2 km/h
- Car: initial_speed 40 km/h, acceleration 10 km/h
- Microbus: initial_speed 15 km/h, acceleration 5 km/h

Now do the following tasks:

(b) Create a method named ***getHighestAccelerationTime()*** (return type: double) which will find out the time needed for a ride to reach the ***speedLimit*** by using the formula :$v = u + at$

(c) Create a method named ***calculateFine(int hour)*** (return type: double) which will calculate the fine for each vehicle by following the implementation below:

- **Bike:** Base fine will be 50 TK and for each hour exceeding **speedLimit**, it will add an extra 100 TK
- **Car:** Base fine will be 100 TK and for each hour exceeding **speedLimit**, it will add an extra 150 TK
- **Microbus:** Total fine will be 3000 TK for just exceeding **speedLimit**.

(d) Find out the output of the following Code:

```
public class Uthao {
    public static void main(String[] args) {
        Ride car = new Car();
        System.out.println(car.calculateFine( hour: 10));
    }
}
```

Q5: (a) Can you define an abstract method in a non-abstract class? Provide a reason for your answer. [1+1+

(b) An abstract class cannot contain variables. Do you agree with the statement? Provide a reason for your answer. 2+1 +1]

Suppose that you are tasked with modeling a library system for various types of reading materials. You will be working with books, magazines. Each type of reading material has different properties. Thus, you are required to accomplish the following tasks:

(c) Write an **abstract class** named **ReadingMaterial** and

i. Add the following **private fields**: title (String), author (String), and year (int).
ii. Add a **constructor** that initializes these fields: (title: String, author: String, year: int).
iii. Define an **abstract method** named **displayDetails()** which should be implemented in subclass (es) to display the details of the reading material.

(d) Create a **subclass** named **Book** that inherits from **ReadingMaterial** and

i. Add an **additional private field** named genre (String) to hold the genre of the book.
ii. Add a **constructor** that takes all parameters (title, author, year, genre) and sets them properly.
iii. Now **override** the "**displayDetails()**" method to display the details of the book, including its title, author, year, and genre.

(e) Create a subclass named **"Magazine"** that inherits from **ReadingMaterial** and

i. Add an **additional private field** named issueNumber (int) to hold the issue number of the magazine.
ii. Add a **constructor** that takes all parameters (title, author, year, issueNumber) and sets them properly.
iii. Now **override** the **"displayDetails()"** method to display the details of the magazine, including its title, author, year, and issue number.