# United International University
## Department of Computer Science and Engineering

CSE 1115: Object Oriented Programming  Mid: Summer 2025

Total Marks: 30  Time: 1 hour and 30 minutes

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

---

*Answer all five questions. The numbers on the right of the questions denote their marks.*

1. Consider the following Code:

**Cineplex.java**

```
class Seat {
    private int row;
    private int number;
    private String tag;

    public Seat(int row, int number, String tag) {
        this.row = row;
        this.number = number;
        this.tag = tag;
    }

    public void setTag(String tag) {
        this.tag = tag;
    }

    @Override
    public String toString() {
        return "R" + row + "-S" + number + "(" + tag + ")";
    }
}
```

(a) Make a 2D array called **hall** inside the main function in **Cineplex** class with **three(3)** rows: row 0 has 2 seats, row 1 has 3, row 2 has 1. (2)

(b) Use nested loops to fill it with new Seat(r, c, "Row" + r + ",Seat" + c), then iterate over the whole array to print each seat with System.out.println(hall[r][c]). (this uses toString()). (2)

(c) Create a Seat class reference variable temp that refers to **hall[1][2]**, update its tag to "**VIP**" and print hall[1][2]. (2)

**Expected Output**

```
R0–S0(Row0, Seat0)   R0–S1(Row0, Seat1)
R1–S0(Row1, Seat0)   R1–S1(Row1, Seat1)   R1–S2(Row1, Seat2)
R2–S0(Row2, Seat0)
R1–S2(VIP)
```

2. Write the output of the following program (6)

**FirstClass.java**

```
public class FirstClass {
    int a = initA();
    static int x = 10;
    FirstClass() {
        this(SecondClass.b);
        System.out.println("FirstClass: no-
            arg ctor");
    }
    FirstClass(int a) {
        System.out.println("FirstClass: one-
            arg ctor, a=" + a);
    }
    static {
        System.out.println("FirstClass:
            static block");
    }
    public static void main(String[] args) {
        System.out.println("MAIN: start");
        FirstClass f1 = new FirstClass();
        FirstClass f2 = new FirstClass(
            SecondClass.b);
        f1.display();
        SecondClass.display();
        SecondClass s = new SecondClass();
        s.display();
        System.out.println("MAIN: end");
    }
    final void display() {
        System.out.println("FirstClass:
            display(), a=" + a + ", x=" + x)
            ;
    }
    static {
        System.out.println("FirstClass:
            static block again");
    }
    int initA() {
        return 2;
    }
}
```

```
class SecondClass {
    static int b;
    int y = initY();
    {
        System.out.println("SecondClass:
            instance block");
    }
    SecondClass() {
        b = 8;
        System.out.println("SecondClass: no
            -arg ctor");
    }
    SecondClass(int b) {
        this.b = b;
        System.out.println("SecondClass:
            one-arg ctor, b="+b);
    }
    static void display() {
        System.out.println("SecondClass:
            display(), b=" + b);
    }
    static {
        b = initB();
        System.out.println("SecondClass:
            static block");
    }
    static int initB() {
        System.out.println("SecondClass:
            static var b init");
        return 9;
    }
    int initY() {
        return 7;
    }
}
```

3. Consider the given code:

**Inheritance.java**

```
class Person {
    private String name ;
    private int age; /// cannot change

    public Person(String name, int age) {
        this.name = name ;
        this.age = age ;
    }
}

class Student extends Person{
    private String studentId; /// cannot change
}

public class Inheritance {
    public static void main(String[] args) {
        Student student = new Student("Harun",17,"S102"); /// cannot change
        GraduateStudent graduateStudent = new GraduateStudent("Alif", 23, "S123"
            , "AI in Healthcare"); /// cannot change
        graduateStudent.age = 25;
        System.out.println("Age: " + graduateStudent.age);
        System.out.println("Student ID" + graduateStudent.studentId);
        System.out.println("Thesis Title: " + graduateStudent.ThesisTitle);
    }
}
```

(a) Modify the **name** attribute so that it can only be inherited by subclasses. Also, ensure that **age** cannot be set to a negative value. (1)

(b) Write necessary constructor(s) and methods in the **Student** class. Use **super** where necessary. (1)

(c) Create another class named **GraduateStudent**, which is a child of Student with an additional attribute thesisTitle (private, String). Write the necessary constructor(s) and methods. (2)

(d) Inside the **main** method, add/rewrite necessary code so that there is no error and the expected output is generated. (2)

**Expected Output**

```
25
S123
AI in Cybersecurity
```

4. Observe the following code and make necessary changes to generate the expected output: (6)

**ColorPrinter.java**

```
class Printer {
  private String brand;

  public Printer(String brand) {
     this.brand = brand;
  }
  /// this method cannot be changed
  public void print(String doc) {
     System.out.println("Printing document: " + doc);
  }
  // Add your code here
}

public class ColorPrinter extends Printer {
  private String color;

  // Add your code here

  public static void main(String[] args) {
    Printer p = new Printer("Epson");
    p.print("Assignment.pdf", 3); /// 1st call

    ColorPrinter cp = new ColorPrinter("Canon", "Red");
    cp.print("Report.pdf"); /// 2nd call
  }
}
```

**Expected Output**

```
Output of 1st call: (this line will not be printed)
Printing 3 copies of the document: Assignment.pdf
Printing copy 1
Printing copy 2
Printing copy 3
Output of 2nd call: (this line will not be printed)
Printing document: Report.pdf
Red Color mode enabled
```

5. Consider the following Code:

**Shape.java**

```
class Shape {
    protected String className = "Shape";
    String getClassName() {
        return this.className;
    }
}

class Rectangle extends Shape {
    double width, height;
    Rectangle(double width, double height) {
        this.className = "Rectangle";
        this.width = width;
        this.height = height;
    }
    protected double getArea() {
        return width * height;
    }
}

class Square extends Rectangle {
    Square(double side) {
        super(side, side);
    }
    @Override
    String getClassName() {
        return "Square";
    }
}
```

**Main.java**

```
public class Main {
    public static void main(String[] args) {
        Shape godel = new Shape();
        Shape escher = new Rectangle(2, 3);
        Rectangle bach = new Square(4);
        System.out.println("0: " + godel.nonExistingMethod()); // Error
            !
        System.out.println("1: " + godel.getClassName());
        System.out.println("2: " + escher.getClassName());
        System.out.println("3: " + bach.getClassName());
        System.out.println("4: " + ((Shape) escher).getClassName());
        System.out.println("5: " + ((Square) escher).getClassName());
        System.out.println("6: " + ((Square) bach).getClassName());
        Shape[] shapeArray = {godel, escher, bach};
        printTotalArea(shapeArray);
    }
    static void printTotalArea(Shape[] shapesArray) {
        double totalArea = 0;
      //complete the code
        System.out.println("Total Area is : " + totalArea);
    }
}
```

(a) Examine the **seven(7) println** statements and write their outputs. If a statement results in either a compile-time or run-time error, skip its output and instead list its number along with the reason in the following format: **0: ERROR, method shape.nonExistingMethod() does not exist** (3)

(b) Implement the **printTotalArea** method that accepts an array of various shape objects and returns the total area of all objects that define an area. **Shape objects should be ignored in the area calculation.** (3)