Diseño y Programación Orientados a Objetos

1er. Semestre 2024

Certamen I ELO-329 Diseño y Programación Orientada a Objetos (1s2024)

Nombre: _____
Paralelo: _____

En este certamen usted no podrá hacer preguntas. Si algo no está claro, indíquelo en su respuesta, haga una suposición razonable y resuelva conforme a ella.

Primera parte, **sin apuntes** (30 minutos):

Primera pregunta (30 pts.)

A. (5pts) En el contexto del desarrollo de software orientado a objetos, considere la siguiente frase:

```
"Salvo excepciones, la interacción con un objeto solo debería realizarse a
través de sus métodos"
```

Explique qué implica esta afirmación. ¿Cuáles son las posibles consecuencias de interactuar con los atributos de un objeto directamente, en lugar de hacerlo a través de sus métodos?

R: *La afirmación habla del principio de encapsulamiento en programación orientada a objetos. Este promueve el acceso a los datos de un objeto exclusivamente mediante métodos públicos. Esto permite controlar cómo se accede y se modifica el estado interno del objeto, aumentando la seguridad restringiendo el acceso a los atributos internos según lo requiera el programador.*

Interactuar directamente con los atributos de un objeto puede llevar a problemas de seguridad y aumentar el riesgo de errores, ya que expone detalles internos del objeto.

B. (5pts) Considere el siguiente fragmento de código Java:

```java
public class FirstSample {

    public static void main(String[] args){
        System.out.println("Hola Mundo!");
    }
}
```

Para el método **main**, describa en detalle el propósito o función de cada elemento en su declaración:

public: **R:** *Este modificador de acceso indica que el método es accesible desde cualquier lugar del código.*

static: **R:** *Significa que el método pertenece a la clase, y no a una instancia específica de la clase.*

void: **R** *Especifica que el método no retorna ningún valor.*

main: **R** *Es el nombre del método que la JVM busca como punto de inicio del programa.*

String[] args: **R** *Es un arreglo que almacena los argumentos de línea de comando pasados al programa.*

C. (5pts) Considere el siguiente código Java

```
double r=2.0;
Figura a= new Circulo(r);
Figura b= new Rectangulo(1.0, 3.0); // ancho, alto

System.out.println("Área círculo: "+a.calcularArea() + ", Área rectángulo: " +
b.calcularArea());
```

Cree la clase **Figura** con un método abstracto **calculaArea()**. Luego cree las clases **Circulo** y **Rectangulo** para que el código anterior entregue el área para cada instancia.

Hint: para las constante **PI**, puede usar **Math.PI**.

R: *código de solución*

```
abstract class Figura {
    abstract double calcularArea();
}

class Círculo extends Figura {
    private double radio;

    public Círculo(double radio) {
        this.radio = radio;
    }

    @Override
    double calcularArea() {
        return Math.PI * radio * radio;
    }
}

class Rectángulo extends Figura {
    private double ancho;
    private double altura;

    public Rectángulo(double ancho, double altura) {
        this.ancho = ancho;
        this.altura = altura;
    }

    @Override
```

```
    double calcularArea() {
        return ancho * altura;
    }
}
```

D. (5pts) Considere las siguientes clases en código Java:

```
class Vehiculo {
    private String marca;
    protected String modelo;

    public Vehiculo(String marca, String modelo) {
        this.marca = marca;
        this.modelo = modelo;
    }

    public void mostrarDetalles() {
        System.out.println("Marca: " + marca);
        System.out.println("Modelo: " + modelo);
    }
}
```

```
public class Main {

    public static void main(String[] args){
        Vehiculo auto = new Auto("Toyota", "Corolla", 4);  // marca, modelo,
número de puertas.
        System.out.println("Detalles del auto:");
        auto.mostrarDetalles();
    }
}
```

Cree la clase **Auto**, de manera que el código **Main** mostrado compile y su ejecución arroje la siguiente salida.

```
Detalles del auto
Marca: Toyota
Modelo: Corolla
Número de Puertas: 4
```

R: *código de solución*

```
class Auto extends Vehiculo {
    private int numeroPuertas;
```

```
    public Auto(String marca, String modelo, int numeroPuertas) {
        super(marca, modelo);
        this.numeroPuertas = numeroPuertas;
    }

    @Override
    public void mostrarDetalles() {
        super.mostrarDetalles();
        System.out.println("Número de Puertas: " + numeroPuertas);
    }
}
```

E. (5pts) El siguiente programa calcula el índice de masa corporal:

```
public class Calculator {
    private int peso;
    public Calculator(int p) {
        peso = p;
    }

    interface DoubleMath {
        double operation(double a);
    }

    public double operate(double a, DoubleMath op) {
        return op.operation(a);
    }

    public static void main(String args[]){
        int peso = Integer.parseInt(args[0]);
        float altura = Float.parseFloat(args[1]);
        Calculator myApp = new Calculator(peso);
        System.out.println("Su indice de masa corporal es: " +
            myApp.operate(altura,
                h -> peso/(h*h))); // Línea a cambiar. Linea previa se debe
mantener
    }
}
```

¿Compila este programa sin errores ni warnings ? En caso de generar error o warning, indique cambios necesarios para superarlos.

Reemplace la línea indicada en el código para conseguir el mismo resultado sin usar expresiones lambda.

R:

a) *Ese código no genera errores ni warnings*

b)

```
public class CalculatorNoLambda {
    private int peso;
    public CalculatorNoLambda(int p) {
        peso = p;
    }
    interface DoubleMath {
        double operation(double a);
    }
    public double operate(double a, DoubleMath op) {
        return op.operation(a);
    }
    public static void main(String args[]) {
        int peso = Integer.parseInt(args[0]);
        float altura = Float.parseFloat(args[1]);
        CalculatorNoLambda myApp = new CalculatorNoLambda(peso);
        System.out.println("Su indice de masa corporal es: " +
        myApp.operate(altura,
            new DoubleMath() {
                public double operation(double a) {
                    return peso/(a*a);
                }
            }
        ));
    }
}
```

F. (5pts) Se tienen las siguientes clases:

```
public class OuterClass {

    private int a;
    public OuterClass(int a) {
        this.a=a;
    }

    public static class StaticClass {
        private int b;
        public StaticClass (int b) {
            this.b=b;
        }
        //   otros métodos
    }

    public class InnerClass {
        private int c;
        public InnerClass(int c){
            a=this.c=c;
        }
        // otros métodos

    }
```

```
    //   otros métodos
    public static void main(String args[]) {

        //aquí se pide crear las instancias de cada clase

    }
}
```

a) ¿Qué código en el método main permite crear una instancia de cada una de las clases dadas: **OuterClass** y sus dos clases anidadas?

b) ¿Es su solución única? Justifique.

R: a)

```
public static void main(String args[]) {
    OuterClass oc= new OuterClass(3);
    StaticClass sc= new StaticClass(5);
    InnerClass oi= oc.new InnerClass(-1);
    //....
}
```

b) *No, también pudo ser*

```
public static void main(String args[]) {
   OuterClass oc= new OuterClass(3);
   OuterClass.StaticClass sc= new StaticClass(5);
   OuterClass.InnerClass oi= oc.new InnerClass(-1);
   //...
}
```