Diseño y Programación Orientados a Objetos

1er. Semestre 2025

Certamen I ELO-329 Diseño y Programación Orientada a Objetos (1s2025)

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) Como los números enteros **(Z)** $\mathbb{Z}$ son un subconjunto de los racionales **(Q)** $\mathbb{Q}$, alguien sugiere: *"los números enteros son números racionales con denominador uno"* y propone crear la clase **Enteros** como clase hija de **Racionales**. ¿Está usted de acuerdo? Justifique su respuesta.

R: (2pts) No estoy de acuerdo.

(3pts) Justificación: Si bien se cumple la relación **es un**, no se cumple la relación de **subtipo**.

Ejemplo: Es esperable que pidamos a un número racional que pueda dividirse por dos. En caso de recibir un número entero cuando esperábamos uno racional, digamos 5, y lo dividimos por dos, la respuesta será 2 o 3 según el redondeo y no 2 ½ como se esperaría para un número racional.

B. (5pts) Considere el siguiente programa Java:

```
1.  import java.util.Random;
2.  public class Constructor {
3.     public Constructor(){}
4.     public Constructor (String n){
5.         name = n;
6.     }
7.     public Constructor (String n, int h){
8.         this(n);
9.         height = h;
10.    }
11.
12.    public String toString(){
13.        return "name:"+name+"; height:"+height+"; rut:"+rut+"; RUT:"+RUT;
14.    }
15.
16.    {
17.        Random generator = new Random();
18.        height = generator.nextInt(200); // aleatorio entre 0 y 199
19.        rut=RUT++;
20.    }
21.
22.    static {
23.        RUT=1234;
24.    }
25.
26.    public static void main (String[] args){
27.        Constructor c1= new Constructor();
28.        Constructor c2= new Constructor("Juan");
29.        Constructor c3= new Constructor("Sebastián", 175);
30.        System.out.println(c1+"\n"+c2+"\n"+c3);
31.    }
32.
33.    private String name="Pablo";
34.    private int height; // en cm
35.    private static int RUT;
36.    private int rut;
37. }
```

i. ¿Generaría error(es) de compilación? **R:** (2pts) No genera errores de compilación.

ii. En caso afirmativo, indique las líneas que generaría error y por qué. En caso de que no haya errores, indique cuál podría ser una salida válida del **main** mostrado. **R:** (3pts)

```
name:Pablo; height:67; rut:1234; RUT:1237
name:Juan; height:167; rut:1235; RUT:1237
name:Sebastián; height:175; rut:1236; RUT:1237
```

C. (5pts) Considere el siguiente diseño parcial de clases:

```
class Documento {
    public String imprimir() {
        return "Imprimiendo documento genérico";
    }
}

class Reporte extends Documento {
    public String imprimir() {
        return "Reporte: " + generarContenido();
    }
    public String generarContenido() {
        return "Contenido base del reporte";
    }
}

class ReporteAnual extends Reporte {
    public String generarContenido() {
        return "Contenido del reporte anual";
    }
}
```

Suponga el siguiente código de uso:

```
Documento d = new ReporteAnual();
System.out.println(d.imprimir());
```

Indique qué se imprime por consola. Justifique.

R: (2pts) Se imprime

```
Reporte: Contenido del reporte anual
```

(3pts) Justificación: El método **imprimir()** es sobrescrito en **Reporte**, y **generarContenido()** se resuelve dinámicamente en **ReporteAnual** por ligado dinámico.

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

```
interface Ejecutable {
    void ejecutar();
}

abstract class Tarea implements Ejecutable {
    public abstract void ejecutar();
    public String toString() {
        return this.getClass().getSimpleName();
    }
}

class TareaEnvio extends Tarea {
    public void ejecutar() {
        System.out.println("Enviando...");
    }
}
```

Suponga el siguiente código en el método **main**:

```
Object obj = new TareaEnvio();
System.out.println(obj.toString());
Ejecutable e = (Ejecutable) obj;
e.ejecutar();
```

Indique si el código en el método **main** se ejecuta con errores.

Si el código se ejecuta con errores, explíquelos.

Si el código anterior se ejecuta correctamente, indique la salida y justifique.

R: (2pts) El programa se ejecuta correctamente.

(3pts) Justificación: Aunque **obj** es una referencia de tipo **Object**, apunta a una instancia de **TareaEnvio**, que implementa la interfaz **Ejecutable**. Por lo tanto, el casteo es válido y se puede invocar tanto **toString()** como **ejecutar()** gracias al **principio de sustitución** y al **ligado dinámico**.

E. (5pts) Explique la diferencia entre una clase anidada estática y una clase interna (no estática) en Java. ¿Cuáles son las ventajas de usar clases anidadas en términos de encapsulamiento y organización del código?

R: (3pts)

- Las clases anidadas estáticas no requieren una instancia de la clase externa.
- Las clases internas requieren una instancia y pueden acceder a miembros privados de la clase contenedora. (2pts)
- Las clases anidadas permiten agrupar clases relacionadas, ocultar detalles y mejorar la estructura del código

F. (5pts) Considere el siguiente diseño base:

```
class Calculadora {
    interface Operador {
        double aplicar(int a, int b);
    }

    public double operar(int a, int b, Operador op) {
        return op.aplicar(a, b);
    }
}
```

1. En un método **main**, escriba el código necesario para realizar una división entre dos números utilizando una expresión lambda compatible con el diseño anterior.

2. Considere la excepción **ArithmeticException**. Utilice manejo de excepciones (**throw** y **try-catch**) para lanzar y capturar un error de división por cero.

Hint: En Java, las divisiones no lanzan automáticamente una excepción cuando el divisor es cero. Usted debe lanzar y verificar manualmente la excepción. **R:**

```
// Suponiendo método main dentro de clase Calculadora
 public static void main(String[] args) {
    // Lambda para la operación de división.
    // (2pts) expresión lambda
    Operador division = (a, b) -> { // Basta con lanzar error acá
        if (b == 0) {
            // (1pt) lanzamiento de excepción
            throw new ArithmeticException("División inválida");
            }
        return a / b;
    };

    // (1pt) captura excepción
    try {// Capturando un error de división por cero
        // (1pt) operación
        Calculadora calc = new Calculadora();
        double resultado = calc.operar(10, 0, division);
        System.out.println("Resultado: " + resultado);
    } catch (ArithmeticException e) {// Basta con capturar error acá
        System.out.println("Error: " + e.getMessage());
    }
}
```