

# Simulacro Parcial

1. Dado el siguiente fragmento de programa, indicar:

- Indicar cuál es la relación entre `Personaje` y `Pantalla`.
- Indicar si viola algún principio de diseño y, en caso afirmativo, indique cuál(es) y explique por qué lo hace.

```
public interface Personaje {
    int getPosicionX();
    int getPosicionY();
    void caminarALaDerecha();
    void caminarALaIzquierda();
    void saltar();
    void disparar();
}

public class Pantalla {
    public void mostrar(Personaje p) {
        System.out.println("El personaje está en las coordenadas (%d, %d).",
            p.getPosicionX(),
            p.getPosicionY()
        );
    }
}
```

2. Explicar en palabras cuál es la relación entre las clases A y B. ¿Qué diferencia habría si el rombo no estuviera pintado?



3. Dado el siguiente programa:

- Indicar si es factible escribir pruebas unitarias para la función `torresDeHanoi`. En caso negativo indicar cuál es la razón y proponer un cambio para que la función sea *testable*.
- Proponer la menor cantidad de pruebas unitarias que deberían considerarse para cubrir todas las clases de equivalencia posibles.

```
class TDH
{
    public static void torresDeHanoi(int n, char desde_torre, char hacia_torre, char aux_torre)
    {
        if (n < 1) throw new IllegalArgumentException("n debe ser al menos 1");
        if (n > 20) throw new IllegalArgumentException("n es demasiado grande");
        if (n == 1)
        {
            System.out.println("Mover el disco 1 desde la torre " + desde_torre + " a la torre " + hacia_torre);
            return;
        }
    }
}
```

```

        torresDeHanoi(n-1, desde_torre, aux_torre, hacia_torre);
        System.out.println("Mover el disco " + n + " desde la torre " + desde_torre + " a la torre " + hacia_torre);
        torresDeHanoi(n-1, aux_torre, hacia_torre, desde_torre);
    }

    public static void main(String args[])
    {
        int n = 4; // Cantidad de discos
        torresDeHanoi(n, 'A', 'C', 'B'); // A, B y C son los nombres de las torres
    }
}

```

4. Dado el siguiente fragmento de programa, indicar:

- Dibujar el diagrama de clases
- Indicar si está basado en algún patrón de diseño, y cuál.
- Indicar si el código es candidato a ser refactorizado mediante algún otro patrón de diseño y, en caso afirmativo, indicar mediante cuál(es) y explicar cómo lo haría.

```

public interface FabricaDeFiguras {
    Triangulo crearTriangulo();
    Rectangulo crearRectangulo();
}

public interface Triangulo { ... }
public interface Rectangulo { ... }

public class FabricaDeFigurasRojas implements FabricaDeFiguras {
    @Override Triangulo crearTriangulo() { return new TrianguloRojo(); }
    @Override Rectangulo crearRectangulo() { return new RectanguloRojo(); }
}

public class FabricaDeFigurasAzules implements FabricaDeFiguras {
    @Override Triangulo crearTriangulo() { return new TrianguloAzul(); }
    @Override Rectangulo crearRectangulo() { return new RectanguloAzul(); }
}

public class TrianguloRojo implements Triangulo { ... }
public class RectanguloRojo implements Rectangulo { ... }

public class TrianguloAzul implements Triangulo { ... }
public class RectanguloAzul implements Rectangulo { ... }

```

5. Dibujar un diagrama de secuencia resumiendo el siguiente fragmento de programa:

```

class SistemaDeCorreo {
    public bool enviar(Correo correo, Directorio d) {
        String direccionDestino = correo.getDireccionDestino();
        Casilla destino = d.getCasilla(direccionDestino);
        bool recibido = destino.recibirCorreo(correo);
        return recibido;
    }
}

```