Programación

EXAMEN RECUPERACIÓN – TEMAS 1-7

1. (1,25p) Tienes un *ArrayList<Integer>* con los siguientes valores: [2, 4, 6, 8, 10].
   a) Utiliza un *Iterator* para eliminar los números mayores a 5 mientras recorres la lista.
   b) ¿Por qué no se puede usar un *for-each* para esto sin lanzar excepción?

2. (1p) Tenemos las clases *Animal* y *Perro*. Ambas implementan el método *hacerSonido():*

```
class Animal {
    public void hacerSonido() {
        System.out.println("El animal está haciendo un sonido!");
    }
}

class Perro extends Animal {
    public void hacerSonido() {
        System.out.println("¡Guau!");
    }
}

Animal a = new Perro();
a.hacerSonido();
```

   a) ¿Qué método se ejecuta y por qué?
   b) ¿Qué pasaría si *hacerSonido()* no estuviera en *Perro* pero sí en *Animal*?

3. (1,25p) Dada esta clase *Persona*:

```
public class Persona {
    public int edad;
}
```

una clase externa modifica la edad así: ***p.edad = -5;***

¿Qué problema representa esto? ¿Cómo aplicarías encapsulación para evitarlo y validar la edad >= 0 correctamente?

4. (1,25p) Considera el siguiente código:

```
int suma = 0;

for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 2; j++) {
        suma += i * j;
        System.out.println("i: " + i + ", j: " + j + ", suma: " + suma);
    }
}

System.out.println("Suma final: " + suma);
```

¿Qué valores van tomando ***i***, ***j***, ***suma*** y las diferentes salidas por pantalla durante la ejecución del programa? Realiza una traza.

5. (1,5p) Tenemos un método que calcula la potencia (b^e):

```
static int potencia(int base, int exponente) {
    if (exponente == 0) return 1;
    return base * potencia(base, exponente - 1);
}

public static void main(String[] args) {
    System.out.println(potencia(2,4));
}
```

   a) ¿Cuántas llamadas recursivas se hacen al invocar a ***potencia(2, 4)***? Haz el seguimiento haciendo uso de una pila de llamadas.

   b) ¿Qué pasaría si el exponente fuera negativo?

6. (1,5p) Supón que estás programando un sistema bancario y necesitas lanzar una excepción cuando un usuario intenta retirar más dinero del que tiene disponible.

   Crea una excepción personalizada llamada *FondosInsuficientesException*, que extienda de *Exception* ò *RuntimeException* y justifica por qué. Invócala desde el siguiente método de ejemplo:

```
public void retirar(double cantidad, double saldo) {
    if (cantidad > saldo) {
        // lanzar excepción
    } else {
        saldo -= cantidad;
    }
}
```

7. (1,25p) ¿Qué imprime este código?

```
int[][] matriz = {
        {5, 3, 2},
        {1, 4, 6},
        {7, 8, 9}
};

int suma = 0, j=1;
for (int i = 0; i < matriz.length; i++) {
    suma += matriz[i][j];
}
System.out.println(suma);
```

¿Cómo modificarías el bucle si quisieras sumar siempre la última fila, sin importar cuántas columnas tenga la matriz?

8. (1p) Explica qué mecanismo tiene *Java* para dar solución a la herencia múltiple de clases, la cual no está permitida.