Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

Complementos de Programación (2014/15)
3º Grado en Ingeniería de Tecnologías de Telecomunicación (Esp. Telemática)
22 de Junio de 2015

DECSAI

Normas para la realización del examen: **Duración: 2.5 horas**

- **El único material permitido durante la realización del examen es un bolígrafo azul o negro.**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

1. **(0.5 ptos)** Indica para qué sirve el especificador `static` para datos miembro y métodos de una clase.
2. **(0.5 ptos)** Explica cuáles son los beneficios de ocultar los datos miembro de una clase a otras clases, definiéndolos como datos privados.
3. **(0.5 ptos)** Explica la diferencia entre sobrecarga y sobreescritura de métodos.
4. **(0.5 ptos)** Indica el problema que hay en el siguiente código:

```
1  public class Prueba {
2    public static void main(String[] args) {
3      Object fruta = new Fruta();
4      Object manzana = (Manzana)fruta;
5    }
6  }
7  class Fruta {
8  }
9  class Manzana extends Fruta {
10 }
```

5. **(0.5 ptos)** Supongamos que `sentencia2` causa una excepción en el siguiente bloque `try-catch`:

```
1  public void m1() {
2    try {
3      sentencia1;
4      sentencia2;
5      sentencia3;
6    }
7    catch (Exception1 ex1) {
8    }
9    catch (Exception2 ex2) {
10   }
11   finally{
12     sentencia4;
13   }
14   sentencia5;
15 }
```

Responde a las siguientes preguntas:

(a) ¿Se ejecutará `sentencia3`?

(b) Si la excepción es de tipo `Exception1`, ¿se ejecutará `sentencia4`?

(c) Si la excepción no es de tipo `Exception1` ni `Exception2`, ¿se ejecutará `sentencia4`?

(d) Si la excepción no es capturada en ninguno de los bloques catch, ¿se ejecutará `sentencia5`?

(e) Si la excepción es capturada, ¿se ejecutará `sentencia5`?

6. **(0.5 ptos)** ¿Por qué hay un error de compilación en el siguiente método?

```
1  public void m(int valor) {
2    if (valor < 40)
3      throw new Exception("valor es muy pequeño");
4  }
```

7. **(3.5 puntos)** Se desea crear un programa para contabilizar el número de repeticiones de cada número en una secuencia de números enteros. Por ejemplo, en el caso de la secuencia *9 3 9 3 2 4* existen 4 datos distintos, dos de ellos (el *2* y *4*) se repiten una vez y los otros dos se repiten dos veces. Para resolver el problema, se propone usar dos tipos de datos:

```
class Pareja {
  private int dato;
  private int nVeces;

  public Pareja(int numero){
    dato=numero;
    nVeces=1;
  }
  public int getDato(){
    return dato;
  }
  public int getnVeces(){
    return nveces;
  }
  public void incrementa(){
    nVeces++;
  }
}

class Frecuencias {
  private Pareja[] parejas;
  private int npares;
  public
   // ... interfaz pública de la clase
}
```

El dato miembro `parejas` de la clase `Frecuencias` es un array que contiene `npares` datos de tipo `Pareja` **ordenados por** `dato`, o bien `null` si no hay ninguna pareja. El array debe tener siempre el tamaño justo para contener el número de parejas añadidas al array hasta el momento.

Define los siguientes métodos en la clase `Frecuencias`, sin modificar la clase `Pareja`:

(a) *(0.5 puntos)* El contructor por defecto.

(b) *(1.5 puntos)* Un método que recibe un dato (entero) y lo añade al array. Si el dato ya se había añadido anteriormente, incrementará el contador correspondiente (`nVeces`). Si no, deberá añadir una nueva pareja con dicho entero y el contador con el valor 1. Ten en cuenta que debes hacerlo lo más eficiente posible.

(c) *(1.5 puntos)* Un método para devolver el número de veces que se repite un dato (valor de `nveces` asociado a un `dato`). Lógicamente devolverá cero si no está en ninguna pareja. Ten en cuenta que debes hacerlo lo más eficiente posible.

8. **(3.5 puntos)** Construir un programa para simular el problema de la llegada y salida de coches a un parking de una determinada capacidad (número de plazas de aparcamiento). Al él llega un coche cada cierto tiempo (aleatorio). Cuando un coche llega al parking, se colocará aleatoriamente en una de las plazas libres que tenga. Del mismo modo, del parking sale un coche cada cierto tiempo (aleatorio). En ese caso, debe liberarse aleatoriamente una de las plazas ocupadas en el parking. Las plazas del parking están numeradas desde 0 hasta $capacidad - 1$. El problema se resolverá creando dos hebras, una que hace llegar coches al parking, y otra que saca coches del parking. Si un coche llega al parking, y todas las plazas están ocupadas, la hebra que mete coches en el parking deberá esperar hasta que quede una plaza libre. Si la hebra que saca coches del parking intenta sacar un coche del parking, pero no hay ninguno en él, deberá esperar hasta que haya alguno en el parking.

Cada vez que entra un coche al parking, el programa debe escribir en la salida estándar un mensaje indicando el número de plaza que se ocupa, y el número total de plazas ocupadas actualmente:

```
Entrada de coche a plaza 11. Plazas ocupadas: 7
```

Cada vez que sale un coche del parking, el programa debe escribir en la salida estándar un mensaje indicando la plaza que se libera, y el número total de plazas ocupadas actualmente:

```
Salida de coche de plaza 3. Plazas ocupadas: 6
```

Como mecanismo de sincronización solo estará permitido el uso de métodos `synchronized` y como mecanismo de comunicación entre las hebras que meten y sacan coches del parking podrá usarse únicamente los métodos `wait()`, `notify()` y `notifyAll()` de la clase `Object`.

Para resolver el problema, construiremos 4 clases:

(a) Clase `Parking`: Debe contener los datos miembro necesarios para saber qué plazas están ocupadas en el parking, y cuántas están ocupadas. La capacidad del parking se definirá mediante un parámetro en el constructor de la clase. No se permite el uso de una *cola bloqueante*. Además debe contener un método `entradaCoche()` usado para meter un coche en el parking, y un método `salidaCoche()` para sacar un coche del parking.

(b) Clase `TareaEntradaCoches`: Representa una clase tarea (usada en una hebra) que en un bucle infinito, se encarga de meter coches en el parking en una plaza elegida aleatoriamente entre las que haya libres, y luego espera un tiempo aleatorio (número de milisegundos aleatorio entre 0 y 1000 milisegundos) cada vez que entra un coche.

(c) Clase `TareaSalidaCoches`: Representa una clase tarea (usada en una hebra) que en un bucle infinito, se encarga de sacar coches del parking (elegidos aleatoriamente entre los que hay en el parking) y espera un tiempo aleatorio (número de milisegundos aleatorio entre 0 y 1000 milisegundos) cada vez que sale un coche.

(d) Clase `ProgramaPrincipal`: Contiene la función `main`. Debe crear los objetos necesarios para que funcione el programa y se debe encargar de crear y lanzar las hebras que ejecutan las tareas `TareaEntradaCoches` y `TareaSalidaCoches`. El `Parking` se creará con una capacidad de 100 coches.

9. **(1 pto)** (Pregunta opcional para subir nota) En la clase `MetodosGenericos`, implementa un método genérico que devuelva el mayor elemento de un array de objetos cuyo tipo implementa el interfaz `Comparable`. El tipo de los elementos del array se declarará con un tipo genérico. Pon un ejemplo de un método `main` que muestre cómo se usaría el anterior método para obtener el máximo de un array de objetos `Integer`.