NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Primer parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 70 minutos
Puntuación máxima: 7 puntos
Fecha: 25 de marzo de 2021

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.

Ejercicio 1 (2 / 7 puntos)

Dado el siguiente código, que representa una posible jerarquía de clases para tratar los archivos en un sistema operativo, implementa:

1. El constructor en cada una de las clases, teniendo en cuenta que solo debe haber un constructor por clase, que reciba como parámetros todos los valores necesarios para inicializar sus atributos.
2. Implementa el método ***toString()*** en las clases que sea necesario.

Nota: La extensión de un ***FicheroJPG*** es “JPG” y la extensión de un ***FicheroExcel*** es “xlsx”.

```java
public abstract class File{
    private String name;
    private String extension;
    private int size;

}
```

```java
public class JPGFile extends File{
    private float latitude;
    private float longitude;

}
```

```java
public class ExcelFile extends File{
  // No attributes
}
```

Ejercicio 2 (3 / 7 puntos)

Sabiendo que en un almacén se guardan productos para su distribución, se define:

- Un producto es un tipo de bien que puede distribuirse desde el almacén. Consta de su nombre, marca y una variable que indique si el producto a almacenar es contable o incontable.
- Un producto almacenable es una especialización del concepto anterior. Se trata de un producto ya preparado para ser guardado en un almacén con un identificador numérico (int) cuyo valor es único para cada producto y se asigna de manera secuencial en el constructor.
- Un producto perecedero es una especialización de producto y se caracteriza porque tiene una fecha de caducidad (String).

- Los productos tienen un tiempo de vida en el que deben salir del almacén. Este tiempo de vida es soportado por el interfaz siguiente:

```
interface Durable{
    public boolean leaveStore();
}
```

El comportamiento es el siguiente:

1. Los productos (clase ***Product***) no tienen la capacidad para decidir si puede abandonar el almacén, pero sí debe ser un comportamiento que tengan todos los productos que hereden de esta clase. Además, se establece que de la clase ***Product*** no se podrán crear instancias.
2. En un producto almacenable se establece que siempre puede abandonar el almacén.
3. En un producto perecedero se establece que puede abandonar el almacén cuando la fecha no es "" o null.

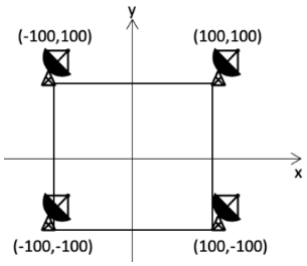
Programa el conjunto de clases que implementan este comportamiento.

Ejercicio 3 (1,5 / 7 puntos)

Teniendo en cuenta la jerarquía de clases del Ejercicio 1, programa una clase TestFile con dos métodos:

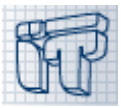
- Un método ***main*** que declare un array llamado ***files*** de tipo ***File*** que contenga 4 ficheros dos de tipo jpg y dos de tipo excel e imprima en pantalla la información de todos ellos.
- Programa el método recursivo: ***public static int totalSize(File[] files, int index)*** que te permita calcular el tamaño de todos los ficheros contenidos en el array files. NOTA: el parámetro int index te permite decir hasta qué posición llegar en el array para que las llamadas recursivas sean cada vez más pequeñas.

Ejercicio 4 (0,5 / 7 puntos)

<table>
<tr>
<td>Se tiene un sistema de radar para la detección de intrusiones no autorizadas en una zona restringida tal y como se muestra en la figura de la derecha ===></td>
<td> 
</td>
</tr>
<tr>
<td>Trabajas para una empresa de consultoría, en el Departamento de Ingeniería del Software para la Defensa y la Seguridad, encargada del desarrollo de los programas capaces de resolver la distancia a la que se encuentra un blanco con respecto a cada una de las antenas del sistema radar.</td>
<td>

```
public class Point {
   private double x;
   private double y;
   public Point(double x, double y) {
       this.x = x; this.y = y;
   }
   /** Returns the distance to another point.*/
   public double distance(Point anotherPoint) {
```

</td>
</tr>
</table>



Dado el siguiente código de la clase que representa a un punto en el plano XY, se pide:

```
        double deltaX; double deltaY;
        deltaX = x - anotherPoint.x;
        deltaY = y - anotherPoint.y;
        return Math.sqrt(
                deltaX * deltaX + deltaY * deltaY
                        );
    }
}
```

Teniendo en cuenta el siguiente código JUnit que testea el método ***public double distance(Point anotherPoint)*** rellena el hueco especificado con **[_____]** para el cálculo de la distancia entre cada una de las antenas y un blanco en la posición (5,-8).

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

import java.util.stream.Stream;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

class PointTest {
        //A continuación se parametrizan las 4 antenas a probar,
        //con sus respectivas distancias al blanco.
        private static Stream<Arguments> addFixture() {
                return Stream.of(Arguments.of(new Point(100, 100),143.836713),
                                Arguments.of(new Point(-100, 100),150.6286825),
                                Arguments.of(new Point(-100, -100),139.6030086),
                                Arguments.of(new Point(100, -100),132.2459829));
        }

        @ParameterizedTest
        @MethodSource("addFixture")
        public void testDistancePoint(Point antenna, double distance) {
                // Este es el punto que representa al blanco.
                **[_____]**
                // Ahora se realiza la prueba del método
                assertEquals(target.distance(antenna),distance,0.0000001);
        }
}
```

Ejercicio 1. OOP Fácil. 2 Puntos

Ejercicio 1. Rúbrica.

Apartado 1. 1 punto. Constructores

0,8 puntos :: Implementación correcta de constructores con uso de super() correcto.
0,2 puntos :: Inicialización correcta de las extensiones de los archivos.

Apartado 2. 1 punto. toString()

0,8 puntos :: implementación correcta toString() en Fichero y FicheroJPG.
0,2 puntos :: uso correcto de super().

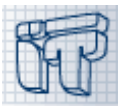
Ejercicio 1. Solución.

```java
package p1_2021_71;
public abstract class File{
    private String name;
    private String extension;
    private int size;

    public File(String name, String extension, int size) {
        this.name = name;
        this.extension = extension;
        this.size = size;
    }
    public String toString() {
        return this.name + "." + this.extension + " size:" + size;
    }
    public int getSize() {
        return size;
    }
}
```

```java
package p1_2021_71;

public class ExcelFile extends File{
  public ExcelFile(String name, int size) {
    super(name, "xlsx", size);
  }
}
```



```java
package p1_2021_71;

public class JPGFile extends File{
    private float latitude;
    private float longitude;

    public JPGFile(String name, int size, float latitude, float longitude) {
        super(name, "jpg" , size);
        this.latitude = latitude;
        this.longitude = longitude;
    }
    public String toString() {
        return super.toString() +  " latitude: " + latitude +  " longitude: " + longitude;
    }
}
```

Ejercicio 2. OOP. PROYECTO

Ejercicio 2. Rúbrica

3 puntos

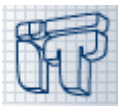
- [0,5 puntos]:: Jerarquía de clases correcta.
- [0,5 puntos]:: Clase Producto Abstracta de forma correcta.
- [0,5 puntos]:: Implements Durable en Producto correcto.
- [0,5 puntos]:: Método abstract en Producto correcto.
- [0,5 puntos]:: AbandonarAlmacen() correcto en ProductoAlmacenable.
- [0,5 puntos]:: AbandonarAlmacen() correcto en ProductoPerecedero.

Ejercicio 2. Soluciones

```java
public abstract class Product implements Durable{
   protected String name;
   protected String mark;
   protected boolean contable;

   public abstract boolean leaveStore();

   public Product(String name, String mark, boolean contable){
      this.name = name;
      this.mark = mark;
      this.contable = contable;
   }
}
```



```
class StorableProduct extends Product{
    private int productID;
    private static int auxID=0;

    public StockableProduct(String name, String mark, boolean contable){
        super(name, mark, contable);
        auxID++;
        this.productID = auxID;
    }


    public boolean leaveStore()
    {
        return true;
    }

}

class PerishebleProduct  extends Product{
    protected String expiryDate;

    public  boolean leaveStore()
    {
        return (
                (this.expiryDate == null) ||
                (this.expiryDate.equals(""))
               );
    }

}

interface Durable{
    public boolean leaveStore();
}
```

Ejercicio 3. Recursividad. Rúbrica. 1,5 puntos

Ejercicio 3.Solución

- 0.5 Método main
  - 0,3 declaración y creación del array
  - 0,2 recorrido del array y llamada a toString()
- 1 Método recursivo
  - 0,6 caso recursivo
  - 0,3 caso base
  - 0,1 llamada al método recursivo desde el main

```
1   public class TestFile {
        Run | Debug
2       public static void main(String[] args) {
3           File[] files = {new JPGFile("portrait", 3, 23, 45),
4                           new JPGFile("landscape", 2, 34, 12),
5                           new ExcelFile("income", 5),
6                           new ExcelFile("expenses", 7)};
7           for (int i = 0; i < files.length; i++) {
8               System.out.println(files[i]);
9           }
10          System.out.println(totalSize(files, files.length-1));
11      }
12      public static int totalSize(File[] files, int index) {
13          int result = 0;
14          if(index==0) {
15              result = files[0].getSize();
16          }else {
17              result = files[index].getSize() + totalSize(files, index-1);
18          }
19          return result;
20      }
21  }
```

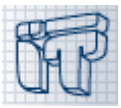
Ejercicio 4. Testing

0,5 puntos

Ejercicio 4. Rúbrica

0,5 puntos por crear correctamente el punto.

Ejercicio 4. Soluciones



```java
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

import java.util.stream.Stream;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

class PointTest {
    //A continuación se parametrizan las 4 antenas a probar,
    //con sus respectivas distancias al blanco.
    private static Stream<Arguments> addFixture() {
        return Stream.of(Arguments.of(new Point(100, 100),143.836713),
                Arguments.of(new Point(-100, 100),150.6286825),
                Arguments.of(new Point(-100, -100),139.6030086),
                Arguments.of(new Point(100, -100),132.2459829));
    }

    @ParameterizedTest
    @MethodSource("addFixture")
    public void testDistancePoint(Point antenna, double distance) {
        // Este es el punto que representa al blanco.
        Point target = new Point(5, -8);
                // Ahora se realiza la prueba del método
        assertEquals(target.distance(antenna),distance,0.0000001);
    }
}
```