NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Convocatoria Ordinaria

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 150 minutos
Puntuación máxima: 7 puntos
Fecha: 1 de junio de 2022

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.
- Rellena tus datos personales antes de comenzar a realizar el examen.

Problema 1. OO + testing (3/ 7 puntos)

Una empresa líder en el sector de la logística le ha pedido que desarrolle un programa para la gestión de un almacén. El almacén distribuye productos (`Product`) que son promocionados por diferentes personas (`Influencer`). Las clases de este programa son las siguientes (suponga que todos los métodos `set` y `get` para todos los atributos están ya implementados):

```java
public class Product {

    private int productID;
    private String name;

    private double costPerUnit;
    private double pricePerUnit;

    private Influencer[] influencers;

    public Product(int productID, String name, double costPerUnit,
                   double pricePerUnit, Influencer[] influencers) {...}

    public int numUnits() { }// to do


    public boolean isProfitable(double limit) { }// to do

}
```

```java
public class Person {

    private int id;
    private String name;

    public Person(int id, String name) {...}

}


public class Influencer extends Person{

    private int timesDiscountApplied;

    // constant

    // constructor

}
```

Apartado 1.1 (0,75 puntos).
La clase `Influencer` es una especialización de la clase Persona (`Person`), que además de heredar estructura y comportamiento de cualquier persona tiene una constante de tipo `double` (`DISCOUNT`) con el descuento que puede aplicar sobre un producto y un atributo de tipo `int` (`timesDiscountApplied`) que cuenta el número de veces que se ha aplicado el código descuento, es decir, el número de unidades compradas con ese descuento. El almacén asigna a cada influencer un código descuento del 10%. Se pide declarar la constante e implementar el constructor de la clase `Influencer`.

Apartado 1.2 (0,5 puntos).

Implemente el método `public int numUnits()` de la clase `Product` que devuelve el número total de unidades que han sido compradas utilizando los descuentos proporcionados por todos los Influencer del producto.

Apartado 1.3 (0,75 puntos).

Implemente el método `public boolean isProfitable(double limit)` de la clase `Product` que devuelve true si el beneficio es mayor del límite dado por argumento y false si es menor.

NOTA: Recuerde que el beneficio de un producto se puede calcular a partir del coste del producto, su precio de venta al público y el número de unidades de dicho producto.

NOTA: Aunque no haya implementado el método anterior (`numUnits()`) suponga que está implementado.

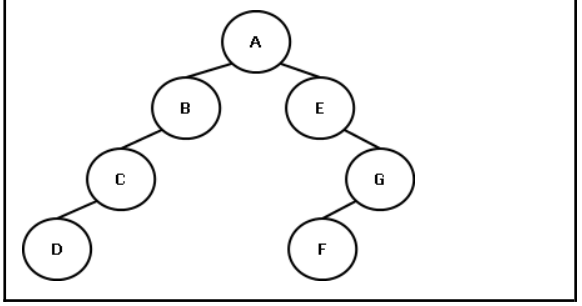
Apartado 1.4 (1 punto)

Codifique el método de testing llamado `testNumUnits()` para el método `numUnits()` de la clase `Product`. El método debe comprobar que el número de unidades vendidas de un producto con dos Influencer, con 10 y 15 descuentos aplicados, es 25.

Problema 2. Estructuras de Datos (3/ 7 puntos)

Dada la clase `TreeMain`, la interfaz `BTree` y la figura mostradas a continuación:

```
public interface BTree<E> {

   static final int LEFT = 0;
   static final int RIGHT = 1;

   boolean isEmpty();

   E getInfo() throws BTreeException;
   void setInfo( E info)  throws BTreeException;

   BTree<E> getLeft() throws BTreeException;
   BTree<E> getRight() throws BTreeException;

}
```




```
public class TreeMain {
 public static void main(String args[]) {

   TreeMain tm = new TreeMain();
   BTree<String> n1 = new LBTree<String>("A");
   BTree<String> n2 = new LBTree<String>("B");
   BTree<String> n3 = new LBTree<String>("C");
   BTree<String> n4 = new LBTree<String>("D");
   BTree<String> n5 = new LBTree<String>("E");
   BTree<String> n6 = new LBTree<String>("F");
   BTree<String> n7 = new LBTree<String>("G");
   BTree<String> tree = n1;
   try {
       n1.insert(n2, BTree.LEFT);
       n1.insert(n5, BTree.RIGHT);
       n2.insert(n3, BTree.LEFT);
       n3.insert(n4, BTree.LEFT);
       n5.insert(n7, BTree.RIGHT);
       n7.insert(n6, BTree.LEFT);

   } catch (BTreeException e) {
       System.out.print(e);
   }
   System.out.println( tm.leafs(tree) );
   System.out.println( tree );
   tm.change(tree, "E", "X");
   tm.change(tree, "A", "X");
   tm.change(tree, "X", "Y");
   System.out.println( tree );
  }

  public int leafs( BTree<String> tree) {...}
  public void change( BTree<String> tree,
                      String s1, String s2) {...}
}
```

Se pide**, utilizando los métodos del interfaz `BTree`**:

Apartado 2.1 (2 puntos).
Implementar el **método recursivo** `public int leafs (BTree<String> tree)` de la clase `TreeMain` que dado un árbol de Strings (`tree`), devuelve el número de nodos hoja de dicho árbol. Para el árbol del ejemplo el resultado sería 2. (Nota: Recuerda que **debes manejar las excepciones** que lancen los métodos de BTree)

Apartado 2.2 (1 punto).
Implementar el **método recursivo** `public void change (BTree<String> tree, String s1, String s2)` de la clase `TreeMain` que dado un árbol de Strings (`tree`), y dos Strings (`s1` y `s2`), sustituye cada ocurrencia en el árbol del String `s1` por el String `s2`. Para hacerlo deberá recorrer recursivamente el árbol buscando nodos cuyo campo info sea `s1` y sustituirlo por `s2`. (Nota: Recuerda que **debes manejar las excepciones** que lancen los métodos de BTree)

Problema 3. Algoritmos (1/ 7 puntos)

Un clasificador implementado en un router, se encarga de clasificar y ordenar los paquetes de datos según una prioridad marcada en los mismos. Para su implementación, el router ejecuta un software formado por las clases abajo definidas. La clase `PPacket`, representa el paquete y su prioridad y la clase `PPacketList`, es una estructura de `PPackets` donde guarda la prioridad de los mismos y en base a esta, realiza las operaciones definidas. Implemente el método `selectionSort()` que ordene la lista de paquetes según su prioridad de forma ascendente (de menor a mayor).

```
public class PPacket {
   private int priority;
   private PPacket link;

   public PPacket (int initialP, PPacket initialLink) {
      priority = initialP;
      link = initialLink;}

   public PPacket(int initialP) {
         priority = initialP;
         link = null;}

  public int getPriority() {
      return priority;}

   public PPacket getLink() {
      return link;}

   public void setData(int newData) {
       priority = newData;}

   public void setLink(PPacket newLink) {
            link = newLink;}
 }
```

```
public class PPacketList {
   private PPacket first;

 public PPacketList() {
    first = null;}

 public PPacketList(int info) {
  PPacket new_ele = new PPacket(info);
  new_ele.setLink(first);
  first = new_ele;}

 public void add(int info) {
  PPacket new_ele = new POrder(info);
  new_ele.setLink(first);
  first = new_ele; }

 public POrder getFirst( ) {
    return first;
                    }
 public void selectionSort() {
      //Metodo a Implementar
                    }
}
```

SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

PROBLEMA 1

Apartado 1.1 (0,75 puntos)

Rúbrica:

- 0 si no tiene sentido.
- 0,25 por declarar correctamente la constante.
- 0,2 por declarar correctamente el constructor.
- 0,2 por invocar correctamente a super().
- 0,1 por asignar el atributo timesDiscountApplied.

Solución:

```
public static final double DISCOUNT = 0.1;

// solution 1
public Influencer(int id, String name, int timesDiscountApplied){
    super(id, name);
    this.timesDiscountApplied = timesDiscountApplied;
}

// solution 2
public Influencer(int id, String name){
    super(id, name);
    this.timesDiscountApplied = 0;
}
```

Apartado 1.2 (0,5 puntos)

Rúbrica:

- 0 si no tiene sentido.
- 0,2 por recorrer correctamente el array.
- 0,2 por llamar al método get de los objetos del array.
- 0,1 por devolver correctamente el valor.

Solución:

```
public int numUnits() { // to do
    int units = 0;
    for(int i =0; i<influencers.length; i++) {
        units += influencers[i].getTimesDiscountApplied();
    }

    return units;
}
```

Apartado 1.3 (0,75 puntos)

Rúbrica:

- 0 si no tiene sentido.
- 0,25 por tener en cuenta el descuento de la clase Influencer.
- 0,25 por calcular correctamente el beneficio.
- 0,25 por devolver correctamente el valor teniendo en cuenta el límite.

Solución:

```
public boolean isProfitable(double limit) { // to do
    boolean result = false;

    double benefit = numUnits()*(pricePerUnit*(1-Influencer.DISCOUNT)-costPerUnit);
    if (benefit > limit){
        result = true;
    } else {
        result = false;
    }

    return result;
}
```

Apartado 1.4 (1 punto)

Rúbrica:

- 0 si no tiene sentido.
- 0,2 por usar la anotación @Test.
- 0,2 por instanciar los objetos Influencer.
- 0,1 por crear el array Influencers.
- 0,2 por instanciar el objeto Product.
- 0,3 por usar correctamente assertEquals.

Solución:

```
@Test
public void testNumUnits(){
    Influencer influencer = new Influencer(1, "influencer", 10);
    Influencer influencer2 = new Influencer(2, "influencer2", 15);
    Influencer[] influencers = {influencer, influencer2};
    Product product = new Product(0, "clothes", 1.5, 3.0, influencers);
    assertEquals(product.numUnits(), 25);
}
```

Problema 2. Estructuras de Datos (3/ 7 puntos)

Apartado 2.1 (2,0 puntos):

Rúbrica:

- 0 si no se hace recursivo.
- 0,1 por evaluar si el árbol está vacío.
- 0,2 inicialización y devolución del resultado correcto
- 0,2 manejo de excepciones (try/catch). Considerar correcto también si dentro del catch ponen un System.out.println.
- 0,5 rama del if
    - 0,25 identificar nodo hoja
    - 0,25 asignación del resultado.
- 0,5 rama izquierda
- 0,5 rama derecha

Solución:

```
public int leafs( BTree<String> tree) {

    int result = 0;

    try {

        if( tree.isEmpty() )
            result = 0;
        else
         if ( tree.getLeft().isEmpty() &&
                    tree.getRight().isEmpty() )  {
              result = 1;
            } else {
              result = leafs( tree.getLeft() ) + leafs( tree.getRight() );
            }
     } catch (BTreeException e) {
         e.printStackTrace();
     }

    return result;
}
```

Apartado 2.2 (1,0 puntos):

Rúbrica:

- 0 si no se hace recursivo..
- 0,1 por evaluar si el árbol está vacío.
- 0,1 manejo de excepciones (try/catch). Considerar correcto también si dentro del catch ponen un System.out.println.
- 0,2 rama del if.
  - 0,1 localizar s1
  - 0,1 Sustituir s1 por s2
- 0,3 llamada recursiva por la izquierda.
- 0,3 llamada recursiva por la derecha.

Solución:

```
public void change( BTree<String> tree, String s1, String s2) {

    try {

        if( ! tree.isEmpty() ) {

           if( tree.getInfo().equals(s1)) {
               tree.setInfo( s2 );
           }

           change( tree.getLeft(), s1, s2);
           change( tree.getRight(), s1, s2);

        }

     } catch (BTreeException e) {
         e.printStackTrace();
     }

}
```

Problema 3. Algoritmos (1/ 7 puntos)

Rúbrica:

- 1 método selectionSort.
  - 0,25 primer bucle correcto.
  - 0,25 segundo bucle correcto.
  - 0,25 bloque if correcto.
  - 0,25 Si hace el intercambio de forma correcta (swap)

Solución:

```
public void selectionSort()      {

for (PPacket node1 = first; node1 != null; node1 = node1.getLink()) {
       PPacket min = node1;
       for (PPacket node2 = node1; node2 != null; node2 = node2.getLink()) {
             if (min.getPriority() > node2.getPriority()) {
                   min = node2;
                   }
              }
       PPacket temp = new PPacket(node1.getPriority(), null);
       node1.setData(min.getPriority());
       min.setData(temp.getPriority());
           }
   }
```