NOMBRE:
APELLIDOS:
NIA:
GRUPO:

# Segundo parcial

## 2ª Parte: Problemas (7 puntos sobre 10)

Duración: 80 minutos
Puntuación máxima: 7 puntos
Fecha: 3 de mayo 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.

## Ejercicio 1. Proyecto (3/ 7 puntos)

La compañía que nos ha encargado el proyecto de gestión de almacenes nos solicita una serie de ampliaciones y modificaciones. Las clases que incluyen estas nuevas especificaciones son las siguientes (asumiendo que los getters y setters están implementados para todas las clases):

- Persona (`Person`). Permite identificar a cualquier persona relevante para la gestión de un almacén. Para identificar a una persona únicamente necesitaremos su número de identificación (`id`), su nombre (`firstName`), apellido (`lastName`) y el email de contacto (`email`).

- Cliente (`Customer`). Especialización de la clase `Person` que representa a los clientes del almacén. Además de los atributos heredados de la clase `Person`, la clase cliente añade el atributo cuenta (`account`) donde se mantiene el valor total de los productos contratados por el cliente.

- Pedido (`Order`). Representa el documento que contiene cada uno de los pedidos que se realizan en el almacén. Para el entorno de nuestro problema los únicos atributos relevantes son: valor total del pedido (`totalPrice`), y el cliente que realiza el pedido (`client`).

- Gestor del almacén (`StoreManager`). Es el cerebro de la aplicación y contendrá toda la lógica del programa. Para el entorno de nuestro problema los únicos atributos relevantes son; el árbol binario de búsqueda (`LBSTree<Customer>`) de clientes (`storeCustomers`) que permite tener registro de los clientes del almacén y la estructura que almacena los pedidos pendientes de ser procesados (`ordersToProcess`), que es una cola de pedidos (`LinkedQueue<Order>`) que implementa la siguiente interfaz:

.

```
public interface Queue<E> {
    boolean isEmpty();
    int size();
    void enqueue (E info);
    E dequeue();
    E front();
```

```
}
```

**Nota:** Para manejar la cola debe utilizar **exclusivamente** los métodos incluidos en la anterior interfaz. No se admitirán soluciones que no cumplan este requisito. Se pide:

**Apartado 1.1 Método compareTo() de la clase Customer (1,5).**

Hacer que la clase Customer implemente la interfaz `Comparable`. Programar el método `compareTo()` de la clase `Customer` que permite comparar dos clientes alfabéticamente en función de su nombre (`firstName`), a igual nombre por apellido (`lastName`) y a igual nombre y apellido por número de identificación (`id`).

NOTA-1: Recuerda que la clase `String` también implementa la interfaz `Comparable` y por tanto, puede usar el método `compareTo()`.

NOTA-2: Recuerda que para que un objeto de tipo `Object` pueda utilizar métodos de una clase que hereda de ella es necesario hacer un casting.

**Apartado 1.2 Método totalValueAccount() de la clase `StoreManager` (1,0).**

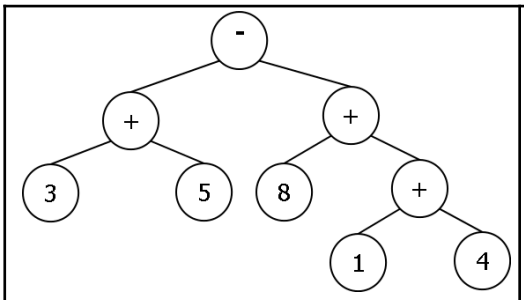
Codifique el método `public double totalValueAccount(Customer customer)` de la clase `StoreManager`. Dicho método debe devolver el valor total de los pedidos pendientes de procesar (`ordersToProcess`) del cliente (`customer`) que se pasa como argumento del método. La cola debe quedar con los elementos en el mismo orden que estaban tras la ejecución del método.

**Apartado 1.3 Método updateAccount() de la clase `StoreManager` (0,5).**

Codifique el método `public void updateAccount(Customer key)` de la clase `StoreManager`. Dicho método debe buscar por `key` a un cliente dentro de `storeCustomers` y si dicho cliente existe, sumar el valor de los pedidos pendientes a su nombre (esto es **`totalValueAccount()`** ) a su cuenta (`account`). Recordar que `LBSTree<Customer>` dispone del método `public BSTree<Customer> search(Comparable c)` y sobre `BSTRee<Customer>` se puede invocar el método `Customer getInfo()`.

## Ejercicio 2. Árboles (4/ 7 puntos)

Tenemos una estructura de árbol binario con objetos tipo `String` (`BTree<String>`) que representa expresiones matemáticas, para simplificar el problema sólo contemplaremos operaciones de sumas y restas entre enteros. Por ejemplo, la expresión (3+5) - ( 8 + (1 + 4 ) ) estará representada por el siguiente árbol binario y nuestro árbol `BTree<String>` implementa la siguiente interfaz:




```
public interface BTree {
    static final int LEFT = 0;
    static final int RIGHT = 1;

    boolean isEmpty();
    E getInfo();
    BTree getLeft();
    BTree getRight();
    void insert(BTree tree, int side);
    BTree extract(int side);
    String toString(); // pre-order
    int size();
    int height();
    boolean equals(BTree tree);
    boolean find(BTree tree);
}
```

NOTA: Recuerda que se dispone del método `int Integer.parseInt( String s)` para convertir un String que represente un número entero a su valor entero.

Se pide implementar, de forma recursiva, los siguientes métodos:

**Apartado 2.1 Método isValid() (1,5).**

Implementar el método `boolean isValid( BTree<String> ope )` que devuelve `True` si la expresión representada por el árbol `ope` es una expresión válida o `False` en caso contrario. Una expresión válida será aquella que cumple que; Todo nodo suma ("+") o resta ("-") tiene que tener dos hijos no nulos, y todo nodo entero (String que representa un valor entero positivo ) tiene que ser un nodo hoja.

**Apartado 2.2 Método eval() (1,5).**

Implementar el método `int eval( BTree<String> ope )` que devuelve el valor entero resultado de evaluar la expresión representada por `ope`. Por ejemplo para la expresión (3+5) - ( 8 + (1 + 4 ) ), representada por el árbol de la figura que aparece en el enunciado del ejercicio, el resultado devuelto es -5.

**Apartado 2.3 Método print() (1,0).**

Implementar el método `String print( BTree<String> ope )` que devuelve la representación en formato `String` de la expresión representada por el árbol `ope`. Por ejemplo, para el árbol representado en la figura que aparece en el enunciado del ejercicio, el resultado sería el `String` "(3+5) - ( 8 + (1 + 4 ) )"

# SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

## Ejercicio 1. Proyecto (3/ 7 puntos)

- Apartado 1.1 Método compareTo() de la clase Customer (1,5 puntos):

```
public int compareTo(Object o) {

    int result;
    Customer c = (Customer) o;

    if ( this.getFirstName().compareTo(c.getFirstName() ) > 0 ){
        result = 1;
    } else if ( this.getFirstName().compareTo(c.getFirstName() ) < 0 ) {
        result = -1;
    } else {
        if ( this.getLastName().compareTo(c.getLastName() ) > 0 ) {
           result = 1;
        } else if ( this.getLastName().compareTo(c.getLastName() ) < 0 ) {
           result = -1;
        } else {
            if ( this.getId() > c.getId() ) {
                result = 1;
            } else if ( this.getId() < c.getId()  ) {
                result = -1;
            } else {
                result = 0;
            }
        }
    }
    return result;
}
```

Criterios de evaluación

- 0 si no tiene sentido
- 0,1 devolver el valor de retorno
- 0,2 Hacer el casting de Object a Customer
- 0,2 llamada correcta a compareTo para firstName
- 0,2 llamada correcta a compareTo para lastName
- 0,2 llamada correcta a < > para el Id
- 0,2 ramas firstName
- 0,2 ramas lastName
- 0,2 ramas Id

- Apartado 1.2 Método totalValueAccount() de la clase StoreManager (1,0 puntos).

```
public double totalValueAccount(Customer customer) {

    double total = 0.0;


    for (int i = 0; i < ordersToProcess.size(); i++) {

          Order order = ordersToProcess.dequeue();

          if ( customer.compareTo(  order.getClient() ) == 0 ) {
               total = total +  order.getTotalPrice();
           }

          ordersToProcess.enqueue(order);
      }


    return total;

}
```

Criterios de evaluación

- 0 si el código no tiene sentido.
- 0,2 si la cola se recorre correctamente.
- 0,2 por desencolar y encolar los elementos.
- 0,3 por comparar correctamente los elementos de tipo Customer.
- 0,2 si consigue que la cola no quede modificada al terminal método (encolando o creando una cola auxiliar).
- 0,1 si se devuelve el resultado correcto.
- Si no se usan los métodos de la interfaz la calificación máxima es 0,75.
- Los errores significativos están sujetos a penalizaciones adicionales.

- Apartado 1.3 Método updateAccount() de la clase StoreManager (0,5 puntos).

```
public void updateAccount(Customer key) {

    double value = this.totalValueAccount(key);

    BSTree<Customer> result = this.storeCustomers.search( key );

    if ( result != null ) {
      Customer c = result.getInfo();
      c.setAccount(  c.getAccount() + value  );
    }

}
```

Criterios de evaluación

- 0 si el código no tiene sentido.
- 0,1 por llamar correctamente el método totalValueAccount().
- 0,1 por llamar correctamente el método search().
- 0,1 por llamar correctamente el método getInfo().
- 0,2 por actualizar correctamente el valor del atributo account.

## Ejercicio 2. Árboles (4/ 7 puntos)

- Apartado 2.1 Método isValid() (1,5 puntos):

```
public boolean isValid( BTree<String> ope )
        throws NumberFormatException, BTreeException {

    if( ope.isEmpty() )
        return false;

    if( ope.getInfo().equals("+") ||
            ope.getInfo().equals("-") ) {

            return isValid( ope.getLeft() ) && isValid( ope.getRight() );

    } else {


        if (  ope.getLeft().isEmpty() &&  ope.getRight().isEmpty() ) {

            if ( Integer.parseInt( ope.getInfo() ) >= 0 )
                    return true;

            };

            return false;

        }
}
```

Criterios de evaluación

- 0 si el código no tiene sentido.
- 0,2 por evaluar si el árbol está vacío.
- 0,3 por identificar nodo operador (“+” o “-”) y nodo operando.
- 0,4 por hacer las llamadas recursivas en caso de nodo operador (“+” o “-”).
- 0,3 por comprobar que nodo operando es nodo hoja.
- 0,3 por comprobar si el operando es un entero positivo.
- Los errores significativos están sujetos a penalizaciones adicionales.
- No considerar el manejo o no de excepciones.

- Apartado 2.2 Método eval() (1,5 puntos):

```
public int eval( BTree<String> ope )
        throws NumberFormatException, BTreeException {

    if( ope.isEmpty() )
        return 0;

    if( ope.getInfo().equals("+") ) {
         return eval( ope.getLeft() ) +
                eval( ope.getRight() );
    } else if ( ope.getInfo().equals("-") ) {

         return eval( ope.getLeft() ) -
                eval( ope.getRight() );

    } else {

        return   Integer.parseInt( ope.getInfo() );

    }
}
```

Criterios de evaluación

- 0 si el código no tiene sentido.
- 0,1 por evaluar si el árbol está vacío.
- 0,3 por identificar nodo operador (“+” o “-”) y nodo operando.
- 0,2 por hacer las llamadas recursivas en caso de nodo operador suma.
- 0,15 por sumar el resultado de las llamadas recursivas en caso de operador suma.
- 0,2 por hacer las llamadas recursivas en caso de nodo operador resta
- 0,15 por restar correctamente el resultado de las llamadas recursivas en caso de operador resta.
- 0,1 por identificar nodo operando.
- 0,2 por devolver correctamente el valor entero del operando.
- Los errores significativos están sujetos a penalizaciones adicionales.
- No considerar el manejo o no de excepciones.

- Apartado 2.3 Método print() (1,0 puntos):

```
public String print( BTree<String> ope )
        throws BTreeException {

    String out = "";

    if( ope.getInfo().equals("+") ||
         ope.getInfo().equals("-") ) {

        if (! ope.getLeft().isEmpty() &&
               (ope.getLeft().getInfo().equals("+")  ||
                ope.getLeft().getInfo().equals("-") ) )
            out = "( " + print( ope.getLeft() ) + " )";
        else
            out = print( ope.getLeft() ) + " ";

        out = out + ope.getInfo();

        if  (! ope.getRight().isEmpty() &&
                (ope.getRight().getInfo().equals("+")  ||
                 ope.getRight().getInfo().equals("-") ) )
            out = out + "( " + print( ope.getRight() ) + " )";
        else
            out = out + print( ope.getRight() ) + " ";


    } else {
        out = ope.getInfo();
    }

    return out;
}
```

Criterios de evaluación

- 0 si el código no tiene sentido.
- 0,3 por identificar nodo operador (“+” o “-”) y nodo operando.
- 0,3 por hacer las llamadas recursiva en orden correcto en caso de nodo operador; izquierda, añadir operador, derecha
- 0,2 por identificar cuando hay que utilizar paréntesis.
- 0,2 por devolver operando.
- Los errores significativos están sujetos a penalizaciones adicionales.
- No considerar el manejo o no de excepciones.