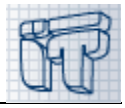


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 180 minutos
Puntuación máxima: 7 puntos
Fecha: 30 mayo 2019

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.
- El examen debe rellenarse con bolígrafo azul o negro. No está permitido entregar el examen con lapicero.

Problema 1 (2 puntos)

Se quiere hacer una aplicación para un evento deportivo que permita imprimir un listado de voluntarios con su número de identificación, la posición que se les asigna en el evento (*"press"*, *"floaters"*, *"ticketing"*, *"grandstand"*, *"protocol"*), su turno de trabajo (mañana "*Shift: M*" o tarde "*Shift: A*") y sus datos personales: la edad, el sexo y un identificador (DNI en el caso de ser español) tal y como se muestra a continuación:

```
Volunteer number: 1, Position: press, Shift: M, Age: 24, Gender: F, ID: 00000001-R
Volunteer number: 2, Position: grandstand, Shift: M, Age: 56, Gender: F, ID: 00000002-W
Volunteer number: 3, Position: ticketing, Shift: A, Age: 43, Gender: M, ID: 00000003-A
```

Los voluntarios además recibirán un ticket de comida si son voluntarios de mañana y de cena si son del turno de tarde. Los voluntarios asignados al puesto de *ticketing* (venta de entradas) no podrán recibir ticket de cena porque la taquilla cierra a las 18:00. Para realizar esta aplicación se proporciona:

- El código de la clase `Person` que modela toda la información personal de los voluntarios (edad-*age*, sexo-*gender* y dni-*id*)
- La interfaz `Position`, que declara el método `selectPosition`() que devuelve un String con el puesto al que ha sido asignado el voluntario.

```
public class Person {
  private int age;
  private char gender;
  private String id;
  public Person(int age, char gender, String id) {
    this.age = age;
    this.gender = gender;
    this.id = id;
  }
  public String toString() {
    return "Age: " + age + ", Gender: " + gender +
      ", ID: " + id;
   }
}
```

```
public interface Position {

String[] positions =
{"press", "floaters", "ticketing",
"grandstand", "protocol"};

String selectPosition();
}
```

Apartado 1 (0,2 puntos)

Declare la interfaz `TicketPrinter` y su método `restaurantTickets()` que no recibe parámetros, devuelve un String y puede lanzar una excepción de tipo `TicketException`.

Apartado 2 (1,5 puntos)

Implemente la clase `Volunteer` que hereda de la clase `Person` e implementa las interfaces `Position` y `TicketPrinter`, y contiene los atributos y métodos necesarios para modelar el estado y comportamiento de un voluntario. Para hacerlo se pide:

(A) Declarar **4 atributos** que no son accesibles desde ninguna otra clase y **2 constantes** que sí podrán ser accedidas desde cualquier clase y son:

- `numTotal`. Número entero que comienza en 1 y se irá incrementando cada vez que se apunte un nuevo voluntario. Será el mismo para todos los voluntarios que se vayan a crear.
- `numVolunteer`. Número entero que se asigna de forma consecutiva a cada voluntario, es decir, cada voluntario tendrá un `numVolunteer` diferente.
- `position`. Atributo de tipo String que guardará la posición asignada.
- `shift`. Atributo de tipo carácter (char) que indica el turno, y tomará los valores de las constantes que siguen.
    - *MORNING*: tomará el valor de 'M' y representa el turno de mañana.
    - *AFTERNOON*: tomará el valor de 'A' y representa el turno de tarde.

(B) Implementar el **método** `selectPosition()` de la interfaz `Position` que asigna de forma aleatoria a cada uno de los voluntarios uno de los cinco puestos definidos (*"press"*, *"floaters"*, *"ticketing"*, *"grandstand"*, *"protocol"*). **NOTA**: Para hacerlo haga uso de los métodos `random` y `round` de la clase `Math` que se proporcionan a continuación:

- `public static double random()  // Returns a double value between [0,1]`
- `public static long round(double a) // Returns the closest long to the argument, with ties rounding up`

(C) Implementar un **constructor** de la clase `Volunteer` que reciba como parámetros la edad, el sexo, el id y el turno y asigne valor al resto de los atributos teniendo en cuenta las siguientes consideraciones:

- Deberá inicializar adecuadamente los atributos `numTotal` y `numVolunteer`.
- Para asignar la posición del voluntario se deberá llamar al método `selectPosition()` implementado anteriormente.
- Para inicializar el atributo `shift` que se refiere al turno, será necesario hacer una comprobación dentro del constructor, llamando al método `boolean checkParameter(char shift)` de la clase `Volunteer`. Si el valor es correcto se asignará directamente y si el valor no es correcto, tomará por defecto el valor de MORNING o 'M'. **NOTA**: No es necesario que implemente el método `checkParameter`; puede invocarlo asumiendo que está implementado correctamente y que funciona devolviendo *true* si el turno toma alguno de los valores posibles y *false* si el valor no es correcto.

(D) Implementar el **método** `toString()` que devuelve la información del voluntario usando el siguiente formato:

```
Volunteer number: <numVolunteer>, Position: <position>, Shift: <shift>, Age: <age>, Gender: <gender>, ID: <id>
```

(E) Implementar el **método** `restaurantTickets()` de la interfaz `TicketPrinter`, que debe devolver una cadena de texto indicando el tipo de ticket para el restaurante que recibe cada voluntario. En el caso de voluntarios del turno de mañana devolverá *"Lunch ticket"*, y para los voluntarios del turno de tarde *"Dinner ticket"*. Además el método deberá lanzar una excepción de tipo `TicketException` con el mensaje de error *"Invalid shift"* si el puesto asignado es venta de entradas (*"ticketing"*) en turno de tarde, ya que estos voluntarios cierran la taquilla a las 18:00h y no tienen derecho a cena. **NOTA**: No es necesario que implemente la clase `TicketException`; puede utilizarla asumiendo que está correctamente implementada.

Apartado 3 (0,3 puntos)

Complete el código del método `main()` de la clase `PrintVolunteerList` para que se imprima la lista de voluntarios, y además imprima el ticket de comida correspondiente llamando al método `restaurantTickets()`.

```
public class PrintVolunteerList {

	public static void main(String[] args) {
		Volunteer v1 = new Volunteer(24, 'F', "00000001-R", Volunteer.MORNING);
		Volunteer v2 = new Volunteer(56, 'F', "00000002-W", Volunteer.AFTERNOON);
		Volunteer v3 = new Volunteer(43, 'M', "00000003-A", Volunteer.AFTERNOON);

		ArrayList<Volunteer> volunteers = new ArrayList<Volunteer>();
		volunteers.add(v1);
		volunteers.add(v2);
		volunteers.add(v3);

		// APARTADO 3. COMPLETAR
}
```

Problema 2 (2 puntos)

Se dispone de las clases `MyBasicLinkedList<E>` y `Node<E>`, que tienen implementados los métodos que se proporcionan a continuación:

```
public class MyBasicLinkedList<E> {
   private Node<E> first;
   public void setFirst(Node<E> first){...}
   public Node<E> getFirst(){...}
   public boolean isEmpty(){...}
   public void insert(E info){...}
   public E extract(){...}
   public int size(){...}
   public int numberOfOccurrences(E info){...}
   public MyBasicLinkedList<E> intersection
     (MyBasicLinkedList<E> list2){//Apartado 2}
}
```

```
public class Node<E> {
   private E info;
   private Node<E> next;
   public Node(E info){this.info = info;}
   public E getInfo(){...}
   public Node<E> getNext(){...}
   public void setInfo(E info){...}
   public void setNext(Node<E> next){...}
}
```

Apartado 1 (0,25 puntos)

Programe una clase `MyBasicLinkedListException` que herede de la clase `Exception` y que simplemente tenga un constructor que reciba un mensaje de tipo *String*.

Apartado 2 (1,75 puntos)

Programe el método `public MyBasicLinkedList<E> intersection(MyBasicLinkedList<E> list2) throws MyBasicLinkedListException`

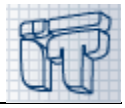
La signatura del método deberá ser respetada en la solución del ejercicio. Este método recibe un objeto de la clase `MyBasicLinkedList<E>` y devuelve una lista cuyos elementos son el resultado de la intersección entre dos listas, es decir, los elementos comunes. En dicha lista resultante **no importa** el orden de los elementos, y **no debe haber elementos repetidos**.

Para hacerlo puede utilizar el método `numberOfOccurrences` de la clase `MyBasicLinkedList` (el cual devuelve el número de veces que `info` está en la lista) asumiendo que está implementado correctamente (el resto de los métodos de ambas clases, salvo `intersection`, también están implementados correctamente y pueden usarse si los necesita).

Por ejemplo, si tenemos las listas L1 y L2, y suponiendo que el campo de información fuera numérico:

`L1: 1 2 3 3`

`L2: 4 3 5 2 6 2 3 3 2 9`



El resultado a devolver en la lista sería: 2 3. La lista con los elementos 3 2 también sería válida, pues no importa el orden en la lista resultante.

Por otra parte, el método debe lanzar la excepción `MyBasicLinkedListException` en el caso de que la lista a devolver fuera vacía. Además, las listas originales deben conservar los mismos elementos que tenían (y en el mismo orden) al finalizar la ejecución del método.

Problema 3 (2 puntos)

Para el desarrollo del problema, se proporciona la interfaz `BTree<E>` y las clases `LBNode<E>` y `LBTree<E>`, que permite modelar un árbol binario. Además, se presenta la clase `BinaryTreeExample`, la cual está compuesta por un `main` y por un método llamado `sumEvenNumbers`.

Nota: No se van a tener en cuenta las excepciones en ningún caso.

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

  boolean isEmpty();
  E getInfo();
  BTree<E> getLeft();
  BTree<E> getRight();
  void insert(BTree<E> tree, int side)
  BTree<E> extract(int side);

  String toStringPreOrder();
  String toStringInOrder();
  String toStringPostOrder();
  String toString();

  int size();
  int height();
  boolean equals(BTree<E> tree);
  boolean find(BTree<E> tree);
}
```

```
class LBNode<E> {
  private E info;
  private BTree<E> left;
  private BTree<E> right;

  LBNode(E info, BTree<E> left, BTree<E> right) {
    this.left = left;
    this.right = right;
    this.info = info;
   }

   E getInfo() {return info;}
   void setInfo(E info) {this.info = info;}

   BTree<E> getLeft() {  return left;     }
   void setLeft(BTree<E> left) {this.left = left;}

   BTree<E> getRight() {return right;}
   void setRight(BTree<E> right) {this.right = right;}
}
```

```
public class LBTree<E> implements
BTree<E>{
  private LBNode<E> root;
  public LBTree() {
    root = null;
  }
  public LBTree(E info) {
    root = new LBNode<E>(info,
           new LBTree<E>(),
           new LBTree<E>());
  }
  ...
}
```

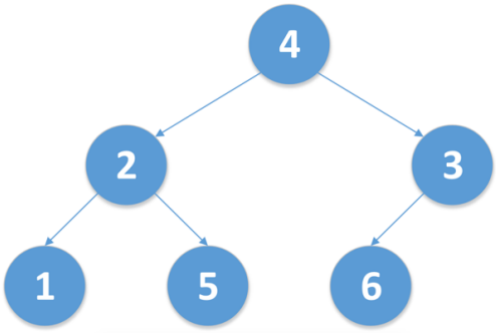
```
public class BinaryTreeExample {
   public static void main(String args[]) {
        BTree<Integer> n1 = new LBTree<Integer>(1);
        BTree<Integer> n2 = new LBTree<Integer>(2);
        BTree<Integer> n3 = new LBTree<Integer>(3);
        BTree<Integer> n4 = new LBTree<Integer>(4);
        BTree<Integer> n5 = new LBTree<Integer>(5);
        BTree<Integer> n6 = new LBTree<Integer>(6);

        // APARTADO 1: El código solicitado se debe
escribir a continuación de este comentario.

        System.out.println("Sum even numbers = " +
                           sumEvenNumbers(n4));
    }
    public static int sumEvenNumbers(BTree<Integer> tree){
// APARTADO 3  }
}
```

Apartado 1 (0,5 puntos)

Complete el método `main` de la clase `BinaryTreeExample` para crear el siguiente árbol binario (ver página siguiente) que almacena información de tipo `Integer`, a partir de los nodos creados que se proporciona en el código, empleando las clases mostradas anteriormente.

Apartado 2 (0,3 puntos)

Del árbol binario proporcionado en el Apartado 1, indicar la secuencia de los recorridos pre-order, in-order y post-order.



Apartado 3 (1,2 puntos)

Implementar el método `sumEvenNumbers` de la clase `BinaryTreeExample`, qué permite sumar el valor de aquellos nodos del árbol cuyo valor sea par. Dicho método se debe **implementar de forma recursiva**; cualquier otra implementación no será puntuada.

NOTA: La sentencia `System.out.println("Sum even numbers = " + sumEvenNumbers(n4));` debe mostrar por pantalla el siguiente resultado (al aplicarse sobre el árbol de ejemplo):

```
Sum even numbers = 12
```

Problema 4 (1 punto)

Dado el siguiente código:

```
import java.util.ArrayList;
public class BubbleSort {
public static void main(String[] args){
        ArrayList<Integer> a = new ArrayList<Integer>();
        a.add(2);
        a.add(5);
        a.add(4);
        a.add(6);
        a.add(8);
        a.add(3);
        a.add(1);
        a.add(9);
        a.add(7);
        System.out.println("Elements before sorting: ");
        System.out.println(a);
        System.out.println("Elements After sorting (in Descending order): ");
        bubbleSort(a);
        System.out.println(a);}
}
```

Implementa el método `bubbleSort` (que no debe devolver nada y debe ser estático) para que ordene el ArrayList en orden **descendente.**

Ejemplo:

```
Elements before sorting:
[2, 5, 4, 6, 8, 3, 1, 9, 7]
Elements After sorting (in Descending order):
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

SOLUCIONES DE REFERENCIA (Varias soluciones a cada uno de los problemas son posibles)

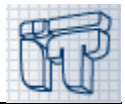
PROBLEMA 1

Apartado 1 (0,2 puntos)

```
public interface TicketPrinter {
	String restaurantTickets() throws TicketException;
}
```

Apartado 2 (1,5 puntos)

```
public class Volunteer extends Person implements Position, TicketPrinter {

	private static int numTotal;

	private int numVolunteer;
	private String position;
	private char shift;


	public static final char MORNING = 'M';
	public static final char AFTERNOON= 'A';


	public Volunteer(int age, char gender, String id, char shift) {
		super(age, gender, id);
		if (checkParameter(shift)){
			this.shift = shift;
		} else {
			this.shift = MORNING;
		}
		this.position = selectPosition();
		numTotal++;
		this.numVolunteer = numTotal;
	}

	public String toString() {
		return "Volunteer number: " + numVolunteer + ", Position: " + position +
", " + "Shift: " + shift + ", " + super.toString();
	}


	public String selectPosition() {
		int p = (int) Math.round(Math.random() * 4);
		return positions[p];
	}

	public String restaurantTickets() throws TicketException {
		if (shift == MORNING) {
		  return "Lunch ticket";
		}
		else if (shift == AFTERNOON && !this.position.equals("ticketing")) {
		  return "Dinner ticket";
		}
		else {
		  throw new TicketException("Invalid shift");
		}
	}
```



```
	public boolean checkParameter(char shift) { // De referencia
		if (shift == MORNING || shift == AFTERNOON) {
				return true;
		}
		return false;
	}
}
```

Apartado 3 (0,3 puntos)

```
for(int i = 0; i<volunteers.size(); i++) {
	System.out.println(volunteers.get(i));
	System.out.println(volunteers.get(i).restaurantTickets());
}
```

Apartado 1 (0,2 puntos)

- 0,2: Declaración de interfaz y método abstracto
  - Si no demuestra conocimiento de interfaz porque pone abstract en la declaración o implementa el método, o pone {} en vez de ;, entonces 0
  - No penalizar si pone public en el método aunque al ser una interfaz no es necesario porque todos sus métodos lo son
  - Penalizar 0,05 si no se pone el throws
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1,5 puntos)

- 0,1: Declaración de la clase
- 0,05: Declaración de la variable estática numTotal
- 0,1: Declaración de la variables numVolunteer, position y shift
- 0,1: Declaración de las dos constantes
- 0,5. Constructor
  - 0,1: Declaración
  - 0,1: Manejo e invocación a super()
  - 0,1: Manejo y asignación de la variable shift
  - 0,1: Manejo y asignación de la variable position
  - 0,05: Manejo y asignación del atributo estático
  - 0,05: Manejo y asignación del atributo numVolunteer
- 0,15: Método toString()
  - No penalizar si lo hace con más líneas de código de las necesarias
- 0,20: Método selectPosition()
  - 0,05: Declaración del método
  - 0,10: Obtención de la posición usando Math.random
  - 0,05: Devolución de la posición
- 0,30: Método restaurantTickets()
  - 0.05: Declaración
  - 0,25: Condiciones para imprimir tickets de comida (0,05), de cena (0,1) o lanzar excepción (0,1)
- Los errores significativos están sujetos a sanciones adicionales

Apartado 3 (0,3 puntos)

- 0,1: Si se hace bien el recorrido y los límites del bucle for
- 0,1: Si imprime la lista de voluntarios
- 0,1: Si imprime los tickets restaurante
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 2 (2 puntos)

Apartado 1 (0,25 puntos)

```
public class MyBasicLinkedListException extends Exception {
     public MyBasicLinkedListException(String msg){
           super(msg);
     }
}
```

Apartado 2 (1,75 puntos)

```
public MyBasicLinkedList<E> intersection(MyBasicLinkedList<E> list2) throws
MyBasicLinkedListException{
    MyBasicLinkedList<E> result = new MyBasicLinkedList<E>();
    Node<E> aux = this.getFirst();

    // También válido for(int i=0; i<this.size(); i++){
    while (aux != null) {
        if ((list2.numberOfOccurrences(aux.getInfo()) != 0) &&
          (result.numberOfOccurrences(aux.getInfo()) == 0))
            result.insert(aux.getInfo());
        aux = aux.getNext();
    }

    // También válido if (result.size()==0)
    if (result.isEmpty())
        throw new MyBasicLinkedListException("Empty intersection!");
    return result;
}
```

Apartado 1 (0,25 puntos)

- 0 si la solución planteada no tiene sentido, o en general no se sabe hacer
- 0,1: Declarar correctamente la clase con la herencia de Exception
- 0,15: Hacer correctamente el constructor
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1,75 puntos)

- 0 si la solución planteada no tiene sentido, o en general no se sabe hacer
- 0,1: Declarar e instanciar correctamente la lista a devolver
- 0,1: Acceder correctamente al primer elemento para recorrer la lista
- 0,25: Declarar el bucle correctamente (tantas iteraciones como elementos tenga una de las listas y condición de parada correcta)
- 0,5: Comprobar correctamente si el elemento actual está en la otra lista y no está repetido en la lista resultante (0,25 cada condición)
- 0,25: Insertar correctamente el elemento si procede en la lista resultado
- 0,15: Avanzar correctamente al siguiente elemento
- 0,25: Lanzar correctamente la excepción en el caso de que la lista resultado sea vacía
- 0,15: Devolver correctamente la lista resultado
- Penalizar con -0,2 si se modifica alguna de las listas
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 3

Apartado 1 (0,5 puntos)

```
n2.insert(n1, BTree.LEFT);
n2.insert(n5, BTree.RIGHT);
n3.insert(n6, BTree.LEFT);
n4.insert(n2, BTree.LEFT);
n4.insert(n3, BTree.RIGHT);
```

Apartado 2 (0,3 puntos)

```
Pre-order = 4 2 1 5 3 6
In-order = 1 2 5 4 6 3
Post-order = 1 5 2 6 3 4
```

Apartado 3 (1,2 puntos)

```
public static int sumEvenNumbers(BTree<Integer> tree) {
	if (tree.isEmpty()) {
		return 0;
	} else if (tree.getInfo() % 2 == 0) {
		return sumEvenNumbers(tree.getLeft()) +
			sumEvenNumbers(tree.getRight()) + tree.getInfo();
	} else {
		return sumEvenNumbers(tree.getLeft()) +
		    sumEvenNumbers(tree.getRight());
	}
}
```

Apartado 1 (0,5 puntos)

- 0,5: Insert realizados correctamente independientemente del segundo argumento del insert(0,1 por cada insert)
- -0,2: Si es incorrecto el segundo argumento de los insert. Segundo argumento puede tomar los siguientes valores únicamente (BTree.LEFT o BTree.RIGHT) o (0 o 1)
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (0,3 punto)

- 0,1 por cada recorrido correcto
- Los errores significativos están sujetos a sanciones adicionales

Apartado 3 (1,2 punto)

- 0,3: Si comprueba si el árbol está vacío
- 0,2: Si comprueba si la información del nodo es par
- 0,4: Por el primer caso recursivo que entra en caso de si es par
- 0,3: Por el caso contrario (else) donde el nodo no tiene información par
- Si el método no se implementa de forma recursiva, entonces 0
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 4

```
public static void  bubbleSort (ArrayList<Integer> a) {
      for (int i = 0; i < a.size(); i++) {
            for (int j = 0; j < a.size() - i - 1; j++) {
                  if (a.get(j).comparteTo(a.get(j + 1)) < 0) {
                      Integer temp = a.get(j);
                      a.set(j, a.get(j + 1));
                      a.set(j + 1, temp);
                  }
            }
      }
}
```

PROBLEMA 4

- 0,1: Declaración correcta del método.
  - Penalizar 0,1 si no ponen void y/o static o sino ponen el argumento o si se equivocan en el tipo del ArrayList
- 0,2: Primer bucle for
  - Penalizar 0,1 si ponen length en lugar de size()
  - Si los límites no están bien definidos, entonces 0
- 0,2: Segundo bucle for
  - Penalizar 0.1 si ponen length en lugar de size()
  - Si los límites no están bien definidos, entonces 0
- 0,3. Condicional if
  - Penalizar 0,3 si ponen en el if la ordenación ascendente
  - Penalizar 0,2 si ponen mal los índices del get
- 0,2: Líneas dentro del if
  - Penalizar 0.1 por el mal uso de set y asignación de índices
  - Penalizar 0.1 por el mal uso de get y asignación de índices