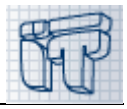


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 180 minutos
Puntuación máxima: 7 puntos
Fecha: 31 mayo 2018

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 (2 puntos)

Tu empresa está desarrollando una aplicación para gestión de excursiones y acaban de incorporarte al proyecto porque el responsable anterior se ha cambiado a otra compañía.

El proyecto llevaba poco tiempo activo, de modo que sólo se ha desarrollado el siguiente código:

- La clase `Participant`, que representa a un participante de una excursión y contiene
  - el nombre del participante (`name`).
  - un constructor que recibe como parámetro el nombre del participante.
- La interfaz `AllergyInfo`, que define un único método, `hasAllergyTo`, que no recibe parámetros y devuelve una cadena de texto con la sustancia que provoca la alergia (por simplicidad se asume que cada persona sólo tiene alergia a una sustancia). Si se trata de una persona no alérgica, el método simplemente devuelve la cadena vacía (“”).

```
public interface AllergyInfo {
      String hasAllergyTo();
}
```

Tu tarea consiste en completar el desarrollo de las clases básicas. Además, durante el análisis de requisitos se detectó que la gestión de posibles alergias de los participantes es una funcionalidad importante para el sistema, de modo que la aplicación deberá tratar esos casos adecuadamente.

NOTA: No está permitido crear métodos ni atributos adicionales a los que se piden en el enunciado (no son necesarios).

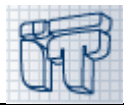
Apartado 1 (1 punto)

Programa la clase `Excursion`, que contiene la siguiente información:

- Una cadena de texto con la descripción (`description`).
- Número de participantes apuntados (`registeredParticipants`).
- Los participantes que se han apuntado a la excursión, que se almacenan en un array de objetos de tipo `Participant`.

Los atributos de la clase no deben ser accesibles desde ninguna otra clase. La clase debe tener únicamente los atributos indicados (no son necesarios más).

Además, la clase `Excursion` debe proporcionar:



- Un constructor que recibe como parámetros la descripción y el número de plazas disponibles. El constructor debe inicializar adecuadamente todos los atributos. Ten en cuenta que inicialmente no habrá ningún participante apuntado a la excursión. El método asume que el número de plazas que se pasa como parámetro es correcto, por lo que no es necesario comprobarlo.
- Un método (`register`) que permite apuntar a un participante a una excursión. Este método recibe como único parámetro el nuevo asistente (de tipo `Participant`), no devuelve ningún resultado y lanza una excepción de tipo `ExcursionException` si ya no quedan plazas disponibles. Puedes suponer que la clase `ExcursionException` ya está programada.

Apartado 2 (0,4 puntos)

Programa la clase `AllergicParticipant`, que representa a un participante de una excursión que sufre algún tipo de alergia. Debe incluir una cadena de texto con información sobre la causa de la alergia (`allergyCause`) y la gravedad de la misma (`severity`). El sistema considera dos niveles de gravedad: *leve* y *severa*, para lo cual la clase `AllergicParticipant` define las constantes `MILD=0` y `SEVERE=1`, respectivamente.

La clase `AllergicParticipant` debe proporcionar también:

- Un constructor que recibe como parámetros el nombre del participante, la sustancia que le provoca la alergia y su gravedad (por simplicidad, asumiremos que el dato de la gravedad será correcto, es decir, se corresponderá con uno de los dos niveles definidos).
- El método `increaseSeverity`, que establece el valor de gravedad a severa (independientemente del valor que tuviera previamente).

Apartado 3 (0,3 puntos)

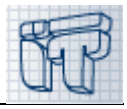
Modifica las clases `Participant` y `AllergicParticipant` para que implementen la interfaz `AllergyInfo`. Recuerda que, en el caso de una persona no alérgica, el método `hasAllergyTo` simplemente devuelve la cadena vacía (`""`); y si se trata de una persona alérgica, devolverá la sustancia responsable.

No es necesario que reescribas las clases. Puedes o bien agregar el código necesario en las clases (si tienes espacio) o bien indicar claramente dónde iría.

Apartado 4 (0,3 puntos)

Programa el método `getAllergies` en la clase `Excursion` que devuelve una cadena de texto con el listado de sustancias a las que son alérgicos los participantes registrados en una determinada excursión. Este método no recibe ningún parámetro.

No es necesario que el método compruebe si hay sustancias repetidas. Es decir, si dos participantes distintos tienen alergia al polen, la cadena "polen" aparecerá dos veces en el listado.



Problema 2 (1 punto)

La clase `InputValidator` ya implementa el método `checkControlLetter` que valida la letra de control del identificador (equivalente al DNI). Es decir, que devuelve `true` si la letra de control es correcta y `false` en caso contrario.

El cálculo de la letra de control se realiza dividiendo el número del identificador `id` (sin letra) entre 23 y el resto se sustituye por una letra que se determina por inspección mediante la siguiente tabla:

RESTO	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
LETRA	T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

Por ejemplo, si el número del identificador es 12345678, dividido entre 23 da de resto 14, luego la letra sería la Z: 12345678Z.

```
/**
 * Valida la letra de control de un identificador (id)
 *
 * @param Identificador del ciudadano
 * @return True si la letra de control es correcta y false si no lo es
 * @throws InputFormatException el id no se compone de un número de exactamente
 *         8 dígitos seguidos de una letra (formato NNNNNNNNL donde N es un
 *         número entre 0 y 9, y L es una letra)
 **/
      public static boolean checkControlLetter(String id) throws
             InputFormatException { ... }
```

Apartado 1 (0,25 puntos)

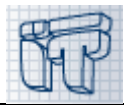
Se pide implementar el método de test que pruebe el método `checkControlLetter` en el caso en el que se lanza la excepción.

NOTA 1: Revisar en detalle la documentación del método para identificar el caso planteado.

NOTA 2: El id 23T no cumple el formato esperado por el método, sino que debería ser 00000023T.

Apartado 2 (0,75 puntos)

Se pide implementar **dos** métodos de test que prueben el método `checkControlLetter`, uno para el caso en el que la validación de la letra del id es correcta y otro para el caso en el que no lo es.



Problema 3 (2 puntos)

Dado el siguiente código:

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

```
public class Node<E> {
   private E info;
   private Node<E> next;

   public Node() {...}
   public E getInfo() {...}
   public void setInfo(E info) {...}
   public Node<E> getNext() {...}
   public void setNext(Node<E> next) {...}
}
```

```
public class LinkedQueue<E>
         implements Queue<E> {
   protected Node<E> top;
   protected Node<E> tail;
   private int size;

   public LinkedQueue() {...}
   public boolean isEmpty() {...}
   public int size() {...}
   public E front() {...}
   public void enqueue (E info) {...}
   public E dequeue() {...}
}
```

Se pide:

Apartado 1 (0,2 puntos)

Implementa la cabecera de la clase `IntegerQueue` sabiendo que especializa el comportamiento de `LinkedQueue<E>` para manejar una cola de objetos de la clase `Integer`.

Apartado 2 (1 punto)

Implementa el método `public void invert()` en la clase `IntegerQueue` que invertirá los elementos de la cola (es decir, los recolocará en orden inverso al original). Puedes utilizar una estructura de datos auxiliar de las estudiadas en clase (hay varias posibilidades), y que puedes suponer ya implementada.

Apartado 3 (0,8 puntos)

Implementa el método `public boolean search(Integer element)` en la clase `IntegerQueue` que devolverá `true` si se encuentra el elemento que recibe como parámetro (`false` en caso contrario).

NOTA: Recuerda que no puedes utilizar == para comparar dos objetos de la clase `Integer`.

Problema 4 (2 puntos)

Para este problema, dispones de la interfaz `BTree<E>` y las clases `LBNode<E>`, `LBTree<E>` y `Point`, que nos servirán para modelar un árbol de puntos. Puedes considerarlas implementadas, excepto el método `isInALeaf`, como se indica a continuación.

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

   boolean isEmpty();
   E getInfo();
   int size();
   BTree<E> getLeft();
   BTree<E> getRight();

   void insert(BTree<E> tree,int side);
   BTree<E> extract(int side);

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

   boolean isInALeaf(E info);

}
```

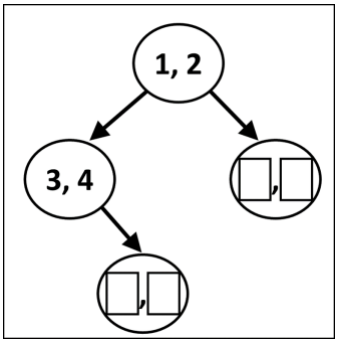
```
public class LBNode<E> {
   private E info;
   private BTree<E> left;
   private BTree<E> right;

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

```
public class LBTree<E>
      implements BTree<E> {
 private LBNode<E> root;

 public LBTree() {
     root = null;}
 public LBTree(E info){...}
            ...
 boolean isInALeaf(E info){
       // APARTADO 1
   }
}
```

```
public class Point {
   public int x;
   public int y;

   public Point() {this(0, 0);}
   public Point(int x, int y) {this.x = x; this.y = y;}
}
```

Apartado 1 (1 punto)

Se te proporciona un fragmento del `main` perteneciente a la clase `MyTree` junto con el esquema del árbol final que se desea generar. Completa **los huecos del código y el contenido de los nodos hoja** con las palabras o números adecuados, de forma que haya una correspondencia entre ambos:




```
public static void main(String[] args) {
   BTree<Point> t1 = new LBTree<Point>(new Point(3, 4));
   BTree<Point> t2 = new LBTree<Point>();
   BTree<Point> t3 = new LBTree<Point>(new Point());
   BTree<Point> t4 = new LBTree<Point>(new Point(          ));

   [      ].insert(t1, [          ]);
   // Create a point p
   [          ] = new Point(1, 6);
   BTree<Point> t5 = new LBTree<Point>(p);

   t5.insert(  [      ]  , BTree.RIGHT);
   t4.insert(t5, BTree.RIGHT);

   p.y = 8;
   t1.insert(t5.extract(  [          ]  ), BTree.RIGHT);
}
```

Apartado 2 (1 punto)

Se pide implementar el **método recursivo** `public boolean isInALeaf (E info)` en la clase `LBTree` que devuelve `true` si el árbol contiene algún nodo hoja con esa `info`. En caso contrario, devolverá `false`.

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

PROBLEMA 1

Apartado 1 (1 punto)

```
public class Excursion {
	private String description;
	private int registeredParticipants;
	private Participant[] participants;

	public Excursion(String desc, int places) {
		this.description = desc;
		this.registeredParticipants = 0;
		participants = new Participant[places];
	}

	public void register(Participant p) throws ExcursionException {
		if (this.registeredParticipants >= participants.length) {
			throw new ExcursionException("No available places");
		} else {
			participants[registeredParticipants] = p;
			registeredParticipants++;
		}
	}
}
```

Apartado 2 (0,4 puntos)

```
public class AllergicParticipant extends Participant {
	public static final int MILD = 0;
	public static final int SEVERE = 1;

	private String allergyCause;
	private int severity;

	public AllergicParticipant(String name, String allergyCause, int severity) {
		super(name);
		this.allergyCause = allergyCause;
		this.severity = severity;
	}

	public void incresaseSensitivity() {
		this.severity = SEVERE;
	}
}
```

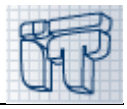
Apartado 3 (0,3 puntos)

En la clase `Participant`:

```
public class Participant implements AllergyInfo {

   // ...

   public String hasAllergyTo() {
       return "";
   }
}
```

En la clase `AllergicParticipant`:

```
public String hasAllergyTo() {
	return allergyCause;
}
```



Apartado 4 (0,3 puntos)

```
public String getAllergies() {
	String res = "";
	for (int i = 0; i < registeredParticipants; i++) {
		res = res + participants[i].hasAllergyTo() + "\n";
	}
	return res;
}
```

Global:

- Restar 0,25 si añaden métodos no pedidos (p.ej. get/set)
- Restar 0,25 si imprimen algún mensaje por consola (ya que no se pide)

(No aplicar la penalización varias veces)

Apartado 1 (1 punto)

- 0,1: Declaración de clase
- 0,1: Declaración de atributos, incluyendo private
- 0,1: Signatura del constructor (importante el no recibir parámetros no indicados en el enunciado)
- 0,1: Inicialización de atributos (int/String) en el constructor
- 0,1: Inicialización del array en el constructor
- 0,1: Signatura método register, con throws
- 0,1: Comprobar condición del número plazas
- 0,1: Lanzar excepción correctamente (throw + new). 0 si sólo se realiza bien una de las dos cosas
- 0,1: Guardar el participante en la posición correcta del array
- 0,1: Incrementar registeredParticipants
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (0,4 puntos)

- 0,1: Declaración de la clase, incluyendo extends
- 0,1: Declaración de las constantes
- 0,1: Llamada a super
- 0,1: Método increaseSeverity (todo correcto, incluido el uso de la constante)
- Restar 0,1 si hay errores en la declaración atributos o resto constructor (signatura con parámetros correctos y asignación de atributos)
- Los errores significativos están sujetos a sanciones adicionales

Apartado 3 (0,3 puntos)

- 0,1: Implements en la declaración de la clase padre (en la hija no es necesario)
- 0,1: Método correcto en la clase padre. Devolver null en vez de cadena vacía es un error (restaría los 0,1)
- 0,1: Método correcto en la clase hija
- Los errores significativos están sujetos a sanciones adicionales

Apartado 4 (0,3 puntos)

(Siempre que tenga sentido en conjunto)

- 0,1: Inicializar, concatenar y devolver adecuadamente el String
- 0,1: Bucle correcto
- 0,1: Llamar al método sobre cada participante
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 2 (1 punto)

```
import static org.junit.Assert.*;
import org.junit.Test;
public class InputValidatorTest {

	// SECTION 1
	@Test(expected = InputFormatException.class)
	public void testIDInputFormat() throws InputFormatException {
		InputValidator.checkControlDigit("0892315RRRR");
	}

	// SECTION 2
	@Test
	public void testIDOk() throws InputFormatException {
		assertTrue(InputValidator.checkControlDigit("00000023T"));
		//assertEquals(true, InputValidator.checkControlDigit("00000023T"));
	}

	@Test
	public void testIDKO() throws InputFormatException {
		assertFalse(InputValidator.checkControlDigit("08923155R"));
		//assertEquals(false, InputValidator.checkControlDigit("08923155R"));
	}
 }
```

Global:

- Restar en total (ambos apartados) 0,1 si no se define en los métodos de test que se lanza excepción
- Restar en total 0,5 si no se invoca correctamente como método estático al método de prueba o se instancia la clase InputValidator
- Restar en total 0,75 si se usa el parámetro delta (números reales) en el assert
- No tenemos en cuenta los imports
- Es indiferente el assert que se use mientras sea correcto

Apartado 1 (0,25 puntos)

- No tenemos en cuenta si no se prueban todos los posibles casos que lanzan excepción, con probar con un id que no cumpla el formato es suficiente
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (0,75 puntos)

- Restar 0,5 si sólo se prueba uno de los casos (id válido o inválido)
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 3

Apartado 1 (0,2 puntos)

```
public class IntegerQueue extends LinkedQueue<Integer> {
}
```

Apartado 2 (1 punto)

```
public void invert(){ // with Stack
    LinkedStack<Integer> le = new LinkedStack<Integer>();
```

```
    // delete elements from the queue and it save them in the stack
    while (!this.isEmpty()){ // size() >= 2 is also OK
        le.push(this.dequeue());
    }

    // delete elements from the stack and save them in the queue
    while (!le.isEmpty()){
        this.enqueue(le.pop());
    }
} // invert()


public void invert(){ // Alternative solution with recursion
    if (!this.isEmpty()){ // size() >= 2
      Integer element = this.dequeue();
      invert();
      this.enqueue(element);
    }
}

public void invert(){ // Alternative solution with ArrayList
    ArrayList<Integer> le = new ArrayList<Integer>();
    while (!this.isEmpty()){ // size() >= 2
        le.add(this.dequeue());
    }
    for (int i = le.size()-1; i >= 0; i--){
        this.enqueue(le.get(i));
    }
} // invert()

public void invert(){ // Alternative solution with array
    Integer le[] = new Integer[this.size()];
    int i = 0;
    while (!this.isEmpty()){ // size() >= 2
        le[i++] = this.dequeue();
    }
    while (i>0) {
        this.enqueue(le[--i]);
    }
} // invert()
```

Apartado 3 (0,8 puntos)

```
public boolean search(Integer element) {
      Node<Integer> current = this.top;
      while (current != null) {
            // if (current.getInfo().equals(element))
            if (current.getInfo().compareTo(element) == 0)
                  return true;
            current = current.getNext();
      }
      return false;
}
```

Apartado 1 (0,2 puntos)

- 0,1: Extends correcto
- 0,1: Indicar correctamente el tipo en el genérico

- 0: En cualquier otro caso (excluido el penalizar por sobrescribir el constructor por defecto)
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1 punto)

- 0,2: Elegir correctamente una estructura de datos auxiliar
- 0,2: Manejo correcto de la estructura de datos cola
- 0,2: Manejo correcto de la estructura de datos auxiliar
- 0,4: Inversión correcta de la cola
- Los errores significativos están sujetos a sanciones adicionales
- Al existir varias soluciones, en cualquier caso, se otorga el punto completo si se deja la cola correctamente invertida

Apartado 3 (0,8 puntos)

- 0,2: Inicialización correcta de las variables del bucle
- 0,2: Recorrido correcto por el bucle
- 0,2: Comparación y encontrar correctamente el elemento
- 0,2: Finalizar el bucle de forma correcta, devolviendo el valor adecuado
- Los errores significativos están sujetos a sanciones adicionales

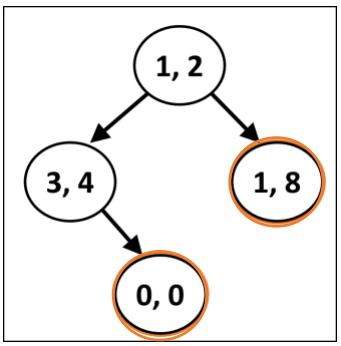
En caso de que se optase por una solución recursiva, aplicaríamos la siguiente rúbrica:

- 0,2: Definición correcta de método auxiliar recursivo
- 0,2: Definición correcta del caso base
- 0,2: Llamada recursiva correcta
- 0,2: Devolución correcta del valor encontrado/no encontrado
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 4

Apartado 1 (1 punto)

```
public static void main(String[] args) {
      BTree<Point> t1 = new LBTree<Point>(new Point(3, 4));
      BTree<Point> t2 = new LBTree<Point>();
      BTree<Point> t3 = new LBTree<Point>(new Point());
      BTree<Point> t4 = new LBTree<Point>(new Point( 1, 2 ));

      t4 .insert(t1, BTree.LEFT );

      // Create a point p
      Point p  = new Point(1, 6);
      BTree<Point> t5 = new LBTree<Point>(p);

      t5.insert( t3 , BTree.RIGHT);
      t4.insert(t5, BTree.RIGHT);

      p.y = 8;
      t1.insert(t5.extract( BTree.RIGHT ), BTree.RIGHT);

}
```



Apartado 2 (1 punto)

```
public boolean isInALeaf (E info) {
      if (isEmpty()) {
```

```
            return false;
        } else if (getLeft().isEmpty() && getRight().isEmpty()){//else if(size() == 1)
              return getInfo().equals(info);
        } else {
            return getLeft().isInALeaf(info) || getRight().isInALeaf(info);
        }
}
```

Apartado 1 (1 punto)

- 0,1: Por cada hueco en el código rellenado correctamente (6 huecos)
- 0,2: Por cada nodo del árbol rellenado correctamente (2 nodos). Es decir, 0,1 por coordenada
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1 punto)

- 0,25: Comprobación del árbol vacío, para devolver falso
- 0,35: Comprobación de la hoja para devolver cierto o falso dependiendo de si coincide o no con la información a buscar
- 0,4: Caso en el que el nodo buscado no haya sido encontrado aún:
    - 0,1: Comprobación de la rama izquierda
    - 0,1: Comprobación de la rama derecha
    - 0,2: Devolución del resultado de ambas ramas combinado con un OR
- Los errores significativos están sujetos a sanciones adicionales