FIRST NAME:
LAST NAME:
NIA:
GROUP:

Second Part: Problems (7 points out of 10)

Duration: 180 minutes
Highest score possible: 7 points
Date: May 31, 2018

Overall instructions for the exam:
- Books, notes, mobile phones, as well as other electronic devices are not allowed during the exam. Breaking this rule may result in expulsion from the examination
- Complete your personal information before starting the exam.

Problem 1 (2 points)

Your company is developing an application for excursion management and you have just joined the project because the previous manager has moved to another company.

The project has been active only for a short time, so only the following code has been developed:
- The class `Participant`, which represents an excursion participant and contains
  - the name of the participant (`name`).
  - a constructor whose only parameter is the name of the participant.
- The interface `AllergyInfo`, which defines one method, `hasAllergyTo`. This method does not receive parameters and returns a string with the substance that causes the allergy (for simplicity it is assumed that each person can have allergy to one substance at most). If it is a non-allergic person, the method simply returns the empty string (“”).

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

Your task is to complete the development of the basic classes. In addition, during the requirements analysis it was found that the management of potential allergies of the participants is an important functionality for the system, so that the application will have to deal with these cases appropriately.

NOTE: You are not allowed to create methods or attributes beyond those requested below (they are not really needed).

Section 1 (1 point)

Program the class `Excursion`, which contains the following information:

- A string with the description of the excursion (`description`).
- The number of participants who registered for the excursion (`registeredParticipants`).
- The participants who have registered for the excursión, which are stored in an array of objects of type `Participant`.

The attributes of the class must not be accessible from any other class. The class must only have the attributes specified above (no further attributes are needed).

In addition, the class `Excursion` must provide:

- A constructor which receives as parameters the description of the excursion and the number of available places. The constructor must properly initialize all attributes. Please note that initially there will be no registered participants for the excursion. The constructor assumes that the number of available places passed as parameter is correct, so it is not necessary to check it.
- A method (`register`) which allows registering a participant for an excursion. This method receives as its only parameter the new participant (of type `Participant`), does not return anything and throws an exception of type `ExcursionException` if there are no more available places. You can assume that the class `ExcursionException` is already developed.

Section 2 (0.4 points)

Program the class `AllergicParticipant`, which represents an allergy-suffering participant on an excursion. This class should have a string with information about the cause of the allergy (`allergyCause`) and the severity of the allergy (`severity`). The application shall consider two levels of severity: mild and severe, for which the class `AllergicParticipant` defines the constants `MILD=0` and `SEVERE=1`, respectively.

The class `AllergicParticipant` must also provide:

- A constructor which receives as parameters the name of the participant, the substance that causes the allergy, and the severity of the allergy (for simplicity, you can assume that the information about the severity is correct, that is, it will correspond to one of the two levels defined in the application).
- The method `increaseSeverity`, which sets the severity value to "severe" (regardless of its previous value).

Section 3 (0.3 points)

Modify the classes `Participant` and `AllergicParticipant` so that they implement the interface `AllergyInfo`. Remember that, in the case of a non-allergic person, the method `hasAllergyTo` simply returns the empty string (`""`); and if it is an allergic person, this method will return the substance which causes the allergy.

You don't have to rewrite the classes. You can either add the necessary code in the classes (if you have space) or clearly indicate where the new code would be placed.

Section 4 (0.3 points)

Program the method `getAllergies` in the class `Excursion` which returns a string listing the substances to which registered participants are allergic on a given tour. This method does not receive any parameters.

The method does not need to check for repeated substances. That is, if two different participants have pollen allergies, "pollen" will appear twice in the list.

Problem 2 (1 point)

The class InputValidator already implements the method checkControlLetter which validates the control letter of an identifier (equivalent to the Spanish National Identifier or DNI). That is, it returns true if the control letter is correct and false otherwise.

The calculation of the control letter is done by dividing the identifier id (without the letter) by 23 and the remaining is replaced by a letter that is determined by inspection using the following table:

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

For example, if the identifier number is 12345678, divided by 23 results in the remainder 14, then the letter would be Z: 12345678Z.

```
/**
 * Validates the control letter of an identifier (id)
 *
 * @param Identifier
 * @return True if the control letter is correct and false otherwise
 * @throws InputFormatException if the id does not contain exactly 8 digits
 *         followed by a letter (format NNNNNNNNL where N is a
 *         number between 0 and 9, and L is a letter)
 **/
     public static boolean checkControlLetter(String id) throws
            InputFormatException { ... }
```

Section 1 (0.25 points)

Program a method which tests the method checkControlLetter in the case when it throws an exception.

NOTE 1: Review in detail the documentation of the method as explained above to identify the case when an exception is thrown

NOTE 2: The id 23T does not meets the format expected by the method but should be 00000023T.

Section 2 (0.75 points)

Program **two methods** to test the method checkControlLetter, one for the case in which the control letter is correct, and another one for the case in which it is not.

Problem 3 (2 points)

Given the following code:

```
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() {...}
}
```

You are requested to do the following:

Section 1 (0.2 points)

Implement the header of class `IntegerQueue` knowing that this class is a specialization of class `LinkedQueue<E>` to store in a queue objects of class `Integer`.

Section 2 (1 point)

Implement the method `public void invert()` in the class `IntegerQueue` which will invert the elements of the queue (i.e., rearranging them in reverse order). You can use an auxiliary data structure of those studied in this course (there are several possibilities), which you can suppose already implemented, to solve this problem.

Section 3 (0.8 points)

Implement the method `public boolean search(Integer element)` in the class `IntegerQueue` wich returns `true` if the element received as a parameter is found in the queue (`false` otherwise).

NOTE: Remember that you cannot use == to compare to objects of class `Integer`.

Problem 4 (2 points)

For this problem, you have the interface `BTree<E>` and the classes `LBNode<E>`, `LBTree<E>` and `Point`, which will be used to model a tree of points. You can assume they are all implemented, except for the method `isInALeaf`, as shown below.

```java
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);

}
```

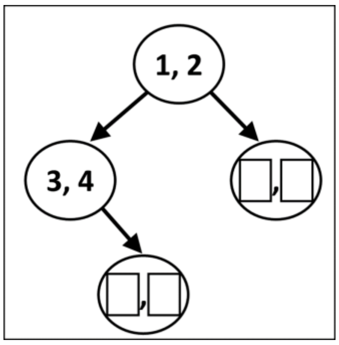
```java
public class LBNode<E> {
   private E info;
   private BTree<E> left;
   private BTree<E> right;

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

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

 public LBTree() {
     root = null;}
 public LBTree(E info){...}

          ...

 boolean isInALeaf(E info){
       // SECTION 1
   }
}
```

```java
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;}
}
```

Section 1 (1 point)

You have part of the code of the `main` method belonging to the class `MyTree` along with the schema of the final tree to be created on the right. **Fill in the gaps in the code and the contents of the leaf nodes in the schema** with the appropriate words or numbers, so that there is a match between the two:



```java
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);
}
```

Section 2 (1 point)

Implement the **recursive method** `public boolean isInALeaf (E info)` in the class `LBTree` which returns `true` if the tree contains any leaf node with that `info`. Otherwise, the method will return `false`.

REFERENCE SOLUTIONS (Several solutions may be valid for each of the problems)

PROBLEM 1

Section 1 (1 point)

```
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++;
		}
	}
}
```

Section 2 (0.4 points)

```
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;
	}
}
```

Section 3 (0.3 points)

En la clase `Participant`:

```
public class Participant implements AllergyInfo {

   // ...

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

En la clase `AllergicParticipant`:

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

Section 4 (0.3 points)

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

Global:

- Subtract 0.25 if you add additional methods (e.g., get/set)
- Subtract 0.25 if a message is printed on screen (as it is not requested)

(The penalty must not be applied multiple times)

Section 1 (1 point)

- 0.1: Class declaration
- 0.1: Attributes declaration, including private
- 0.1: Signature of the constructor (parameters not indicated in the statement shall not be received)
- 0.1: Attribute initialization (int/String) in the constructor
- 0.1: Initialization of the array in the constructor
- 0.1: *register* method signature, with throws
- 0.1: Checking the condition of the number of places
- 0.1: Throwing the exception correctly (throw + new). 0 if only one of the two words is included.
- 0.1: Setting the participant in the correct position of the array
- 0.1: Increasing registeredParticipants
- Significant errors are subject to additional penalties

Section 2 (0.4 points)

- 0.1: Class declaration, including extends
- 0.1: Constants declaration
- 0.1: Call to the superclass with super
- 0.1: Method increaseSeverity (all correct, including the use of the constant)
- Subtract 0.1 if there are errors in the declaration of attributes or in the rest of the constructor (signature with correct parameters and attribute assignment).
- Significant errors are subject to additional penalties

Section 3 (0.3 points)

- 0.1: Implements in the class declaration of the parent class (in the child class it is not needed)
- 0.1: Correct method in the parent class. Returning null instead of empty string is an error (0 points)
- 0.1: Correct method in the child class
- Significant errors are subject to additional penalties

Section 4 (0.3 points)

(As long as it makes sense)

- 0.1: Initializing, concatenating and returning the String properly
- 0.1: Correct loop
- 0.1: Calling the method on each participant
- Significant errors are subject to additional penalties

PROBLEM 2 (1 point)

```java
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:

- Subtract in total (both sections) 0.1 if the exception that is thrown is not defined in the test methods
- Subtract a total of 0.5 if the method to be tested is not correctly invoked as a static method, or if the InputValidator class is instantiated.
- Subtract a total of 0.75 if the delta parameter (real numbers) is used in the assert
- Imports are not taken into account
- It is irrelevant which assert is used as long as the code is correct.

Section 1 (0.25 points)

- We do not take into account if not all possible cases that throw an exception are tested, with a case in which the id that does not meet the format is enough
- Significant errors are subject to additional penalties

Section 2 (0.75 points)

- Subtract 0.5 if only one of the cases is tested (valid or invalid id)
- Significant errors are subject to additional penalties

PROBLEM 3

Section 1 (0.2 points)

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

Section 2 (1 point)

```java
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()
```

Section 3 (0.8 points)

```
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;
}
```

Section 1 (0.2 points)

- 0.1: Extends correct
- 0.1: Indicating correctly the type in the generic

- 0: In any other case (excluding the penalty for overriding the default constructor)
- Significant errors are subject to additional penalties

Section 2 (1 point)

- 0.2: Choosing an auxiliary data structure correctly
- 0.2: Correct handling of the queue data structure
- 0.2: Correct handling of the auxiliary data structure
- 0.4: Correct queue inversión
- Significant errors are subject to additional penalties
- As there can be several solutions, in any case, the full score is granted if the information in the queue is correctly inverted.

Section 3 (0.8 points)

- 0.2: Correct initialization of loop variables
- 0.2: Correct loop traversal
- 0.2: Comparison and finding the correct element
- 0.2: Ending the loop correctly, returning the correct value
- Significant errors are subject to additional penalties

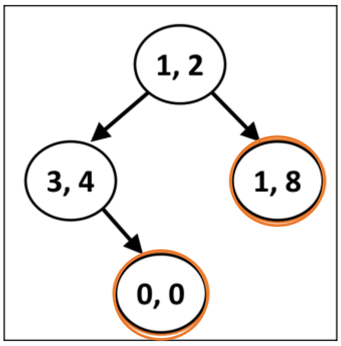
In the event that a recursive solution is chosen, we apply the following criteria:

- 0.2: Correct definition of recursive auxiliary method
- 0.2: Correct definition of the base case
- 0.2: Correct recursive call
- 0.2: Correct return of the found/not found value
- Significant errors are subject to additional penalties

PROBLEM 4

Section 1 (1 point)

```java
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);

}
```



Section 2 (1 point)

```java
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);
      }
}
```

Section 1 (1 point)

- 0.1: For every gap in the code filled correctly (6 gaps)
- 0.2: For each node of the tree filled correctly (2 nodes). That is, 0.1 per coordinate (x,y)
- Significant errors are subject to additional penalties

Section 2 (1 point)

- 0.25: Checking the empty tree and returning false
- 0.35: Checking the lead to return true or false depending on whether it matches or not the information to be searched
- 0.4: Case in which the searched node has not been found yet:
    - 0.1: Checking the left branch
    - 0.1: Checking the right branch
    - 0.2: Returning the result of both branches combined with an OR
- Significant errors are subject to additional penalties