Bachelor in Informatics Engineering	***MODEL A*** Algorithms and Data Structures **30th of June, 2014.** **EXTRAORDINARY CALL**

Name: ....

Surname: ....

PLEASE, READ CAREFULLY THE FOLLOWING INSTRUCTIONS BEFORE START

1. All student data asked in this document must be filled. Please, use a pen to write in this document.
2. The exam includes 5 problems/questions.
3. Only the answer written in this document will be evaluated.
4. This document and all additional sheets must be delivered upon completion.
5. You cannot leave the classroom until the exam has ended.
6. Mobile phones must be disconnected during the exam.
7. The exam will take **2 hours**.
8. **You can consider that all methods included in EDALib classes are public. To make your answers simpler, you can also consider that all attributes in the classes you define are public.**

DO NOT GO BEYOND THIS PAGE until the exam has started.

Problem 1 (3.5 puntos).

Given the following formal specification of the ADT Geometric Sequence:

```
/**
 * ADT Goemetric Sequence specification.
 *  In  a  geometric  progression  each  term  is  obtained  by  multiplying  the
previous one by a given constant factor
 * @author isegura
 */
public interface IGeometricSequence {
	/** returns the factor for the sequence */
	public int getFactor();
	/** computes and returns the next term in sequence */
	public int nextTerm ();
	/** shows the elements of the sequence */
	public void show();
	/** shows the sequence in inverse order */
	public void showInv();
}
```

The goal of the exercise is to develop a class, called **GemoetricSequence**, implementing the given ADT, IGeometricSequence. That class must allow for the storage of the terms in the sequence and the corresponding factor. The student must implement his/her own GeometricSequence class storing all terms in a linked list. In other words, **it is NOT allowed using classes implementing linked lists (that is to say, classes such as SList, DList, ArrayList, LinkedList or Vector are FORBIDDEN). Arrays are also FORBIDDEN**. Nevertheless, using SNode or DNode classes is highly recommended.

a) (0.1 points) Write a constructor method allowing the creation of an empty sequence.

b) (0.25 points) Write a constructor method receiving the sequence factor and the value of the first term in sequence as parameters.

c) (0.25 points) Which attributes must be added to have an adequate representation of an object of the class?

d) (0.1 points) Write the getFactor() method, which returns the value of the factor for the geometric sequence.

```
public class GeometricSequence implements IGeometricSequence {
```

```java
	public GeometricSequence (int factor, int firstTerm) {




	}

	public int getFactor() {



	}
}
```

SOLUTION:

```java
public class GeometricSequence implements IGeometricSequence {
	/** Nodo que almacenará el primer elemento de la sucesión*/
	SNode<Integer> primero=null;
	/** Factor de la sucesión geométrica*/
	int factor;
	/** Almacena el valor del primer término*/
	int primTerm;

	/** Constructor que guarda el factor y crea el primer término de
	la sucesión */
	public GeometricSequence (int factor, int primTerm) {
		this.factor=factor;
		this.primTerm=primTerm;
		primero=new SNode<Integer>(primTerm);
	}

	/** Devuelve el factor de la sucesión */
	public int getFactor() {
		return factor;
	}
```

e) (1 point) Write the nextTerm() method. This method computes and stores the next term in sequence. That term is obtained by multiplying the previous term by the factor. As a recommendation, the method must take into account the situation when the sequence has not been initialized yet (that is to say, when the first element does not yet reference any other node).

```java
public int nextTerm () {
}

SOLUTION:

public int nextTerm() {
        SNode<Integer> anterior = null;
        for (SNode<Integer> nodeIt = primero;
                        nodeIt != null; nodeIt = nodeIt.nextNode) {

                anterior = nodeIt;
        }
        int sigTerm=-1;
        if (anterior == null) {
                sigTerm=primTerm;
                primero=new SNode<Integer>(primTerm);
        } else {
                int antValue=anterior.getElement();
                sigTerm=antValue*factor;
                SNode<Integer> newNode=new SNode<Integer>(sigTerm);
                anterior.nextNode = newNode;
        }
        return sigTerm;
}
```

f) (0.3 points) Write a second constructor that, besides receiving the factor and the first element in sequence as parameters, creates the first n elements in the sequence.

```
public GeometricSequence(int factor, int firstTerm, int n) {
```

```java
}
```

SOLUTION:

```java
public GeometricSequence (int factor, int firstTerm, int n) {
       this.factor=factor;
       this.primTerm=firstTerm;
       for (int i=0; i<=n;i++) nextTerm();
}
```

g) (0.5 points) Write the method show().

```java
public void show() {



}
```

SOLUTION:

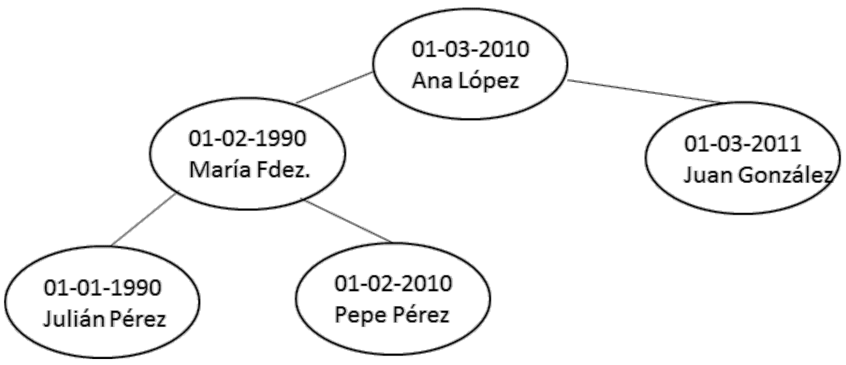
```java
public void show() {
              System.out.print("Sucesión : ");

              for (SNode<Integer> nodeIt = primero;
                            nodeIt != null; nodeIt = nodeIt.nextNode) {
                     System.out.print(nodeIt.getElement()+" ");
              }
              System.out.println();
       }
```

h) (1 point) Write the showInv() method. It shows the elements of the geometric sequence in reverse order. NOTE: if needed, it is possible to use abstract data types such as stacks, queues, etc.

```java
public void showInv() {
        }

SOLUTION:

        public void showInv() {
                SStack<Integer> pila=new SStack<Integer>();
                for (SNode<Integer> nodeIt = primero;
                                nodeIt != null; nodeIt = nodeIt.nextNode) {
                        pila.push(nodeIt.getElement());
                }
                System.out.print("Sucesión en orden inverso: ");
                while (!pila.isEmpty()) {
                        System.out.print(pila.pop()+ " ");
                }
                ();
}
```

Problem 2 (3,5 points)

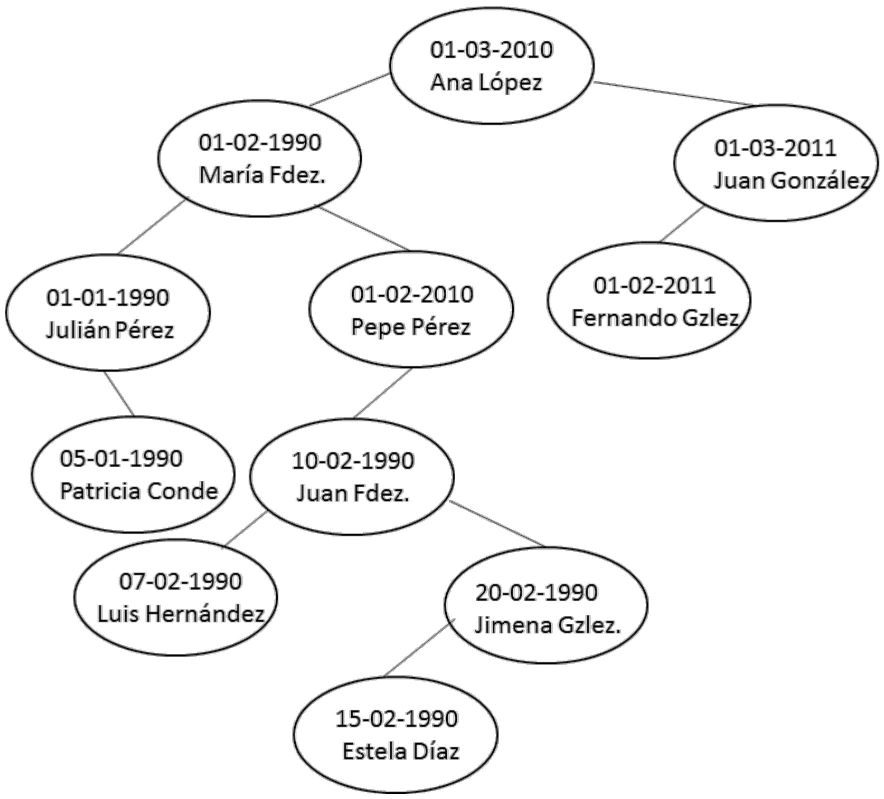
My family is a kind of special and, instead of remembering my ancestors using a genealogical tree, this year they have preferred storing all the information in a Binary Search Tree (BST). They have realized that, in that way, it is easier searching the relatives who lived in a given time period. In the following figure it is represented part of my family tree, where the key is the birth date and each element in a node is the proper name of the corresponding person:



a) **(0,5 points)** Depict the resulting tree after inserting the following relatives in the tree shown in the previous figure. IMPORANT: the order given in the list is important and must be followed:

- <10-02-1990, Juan Fdez.>
- <20-02-1990, Jimena González>
- <15-02-1990, Estela Díaz>
- <05-01-1990, Patricia Conde>
- <01-02-2011, Fernando González>
- <07-02-1990, Luis Hernández>

SOLUTION:



b) **(0,5 points)** Write the header for the class "FamilyTree". This class allows for the storing of a binary search tree. For this purpose, EDALib library must be used (in particular, the BSTree class). Dates will be represented through the "Date" data type in Java and the names will be represented using "String".

SOLUTION:

```
public class FamilyTree extends BSTree<Date, String> {

}
```

c) **(1 points)** Implement a recursive algorithm "firstRelative" returning the name of the first person born in the family (according to the information stored in the tree).

IMPORTANT NOTICE: In order to develop the different implementations requested in this exercise, remember that the following methods are available:

- Class BSTNode
  - *node.**hasLeftChild()***
  - *node.**hasRigtChild()***

SOLUTION:

```
public String firstRelative(){
      return primerFamiliar (root);
}

public String firstRelative (BSTNode<Date, String> nodo){
      if (nodo != null){
            if (nodo.hasLeftChild())
                  return firstRelative (nodo.getLeftChild());
            else return nodo.getElement();
      }
      return null;
}
```

d) **(1.5 points)** Implement the recursive algorithm "ancestors" that prints in console the name of all the relatives born before a given date, ordered according to the most recent date. Remember that you can use the method ***<date>.compareTo(<date>)***, where, given two dates f1 and f2, f1.compareTo(f2) will return:

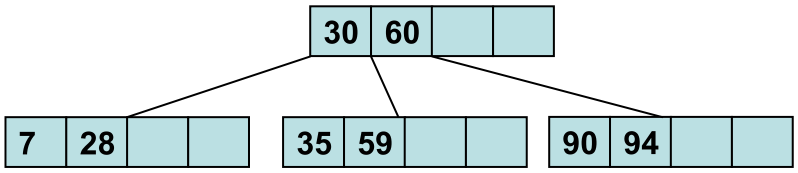
a. a negative value if f1<f2;
b. 0 if f1=f2
c. a number >0 if f1>f2.

SOLUTION:

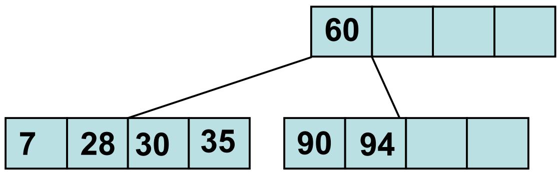
```
   public void ancestors(Date fecha) {
         ancestors (root, fecha);
   }



   public void ancestors(BSTNode<Date, String> nodo, Date fecha) {
         if (nodo != null) {
               // POST-ORDER traversal

               // right child
               If ((nodo.hasRightChild()) &&
                   (nodo.getKey().compareTo(fecha) < 0))
                          ancestors(nodo.getRightChild(), fecha);

               // node
               if (nodo.getKey().compareTo(fecha) < 0)
                          System.out.println(nodo.getElement());

               // Left child
               if ((nodo.hasLeftChild()) &&
                    (nodo.getKey().compareTo(fecha) < 0))
                          ancestors(nodo.getLeftChild(), fecha);
         }
   }
```

Problem 3 (1 point)

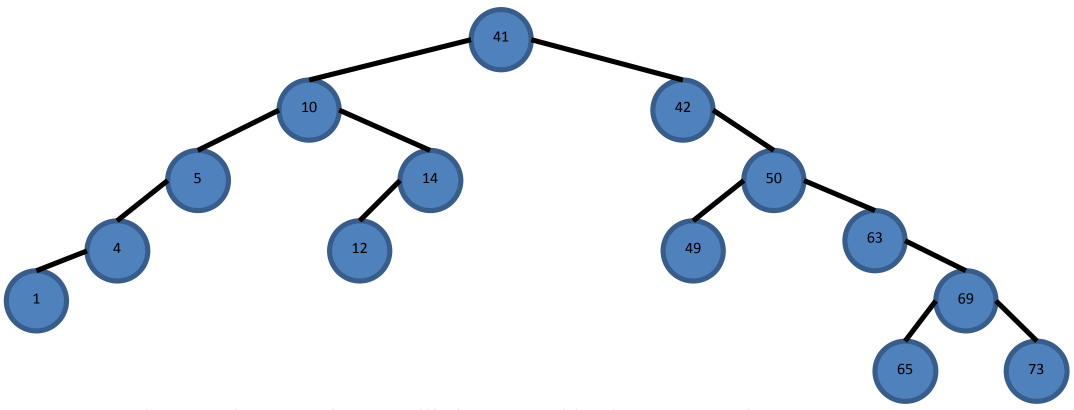
a) **(0,30 points)** Taking into account the given B Tree, draw the resulting tree when deleting the key **59.**



SOLUTION:

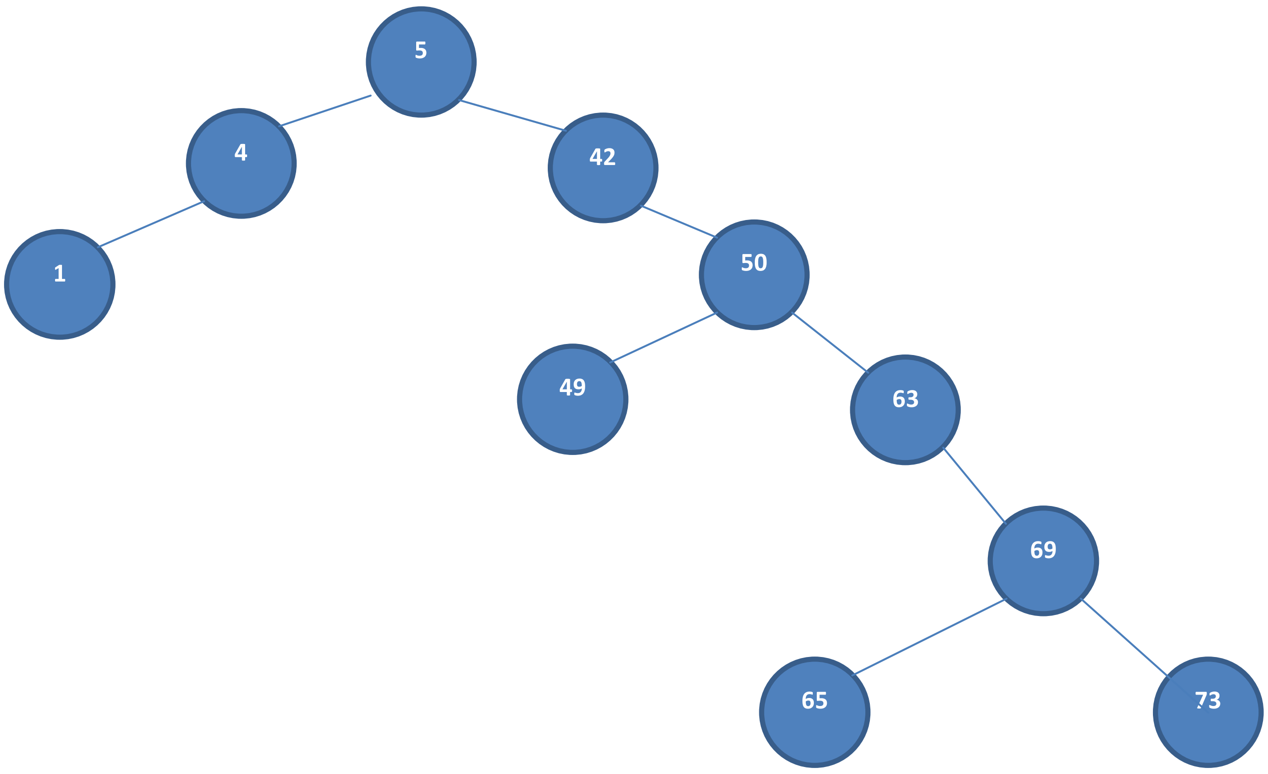


b) **(0,40 points)** Using as a starting point the given binary search tree (where the key is equal to the stored element), remove 4 times the root of the tree. Depict only the resulting tree (you can include the needed intermediate operations in auxiliary sheets).

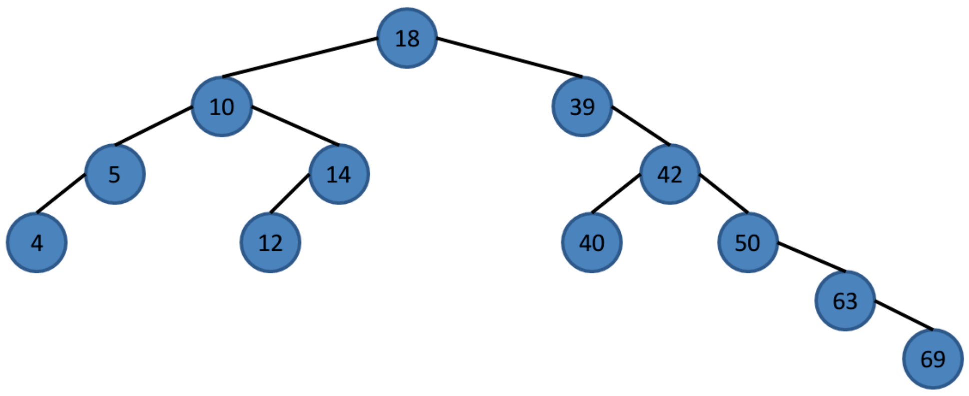


Note: In the removing operation we will always consider the ancestor rule.

SOLUTION:

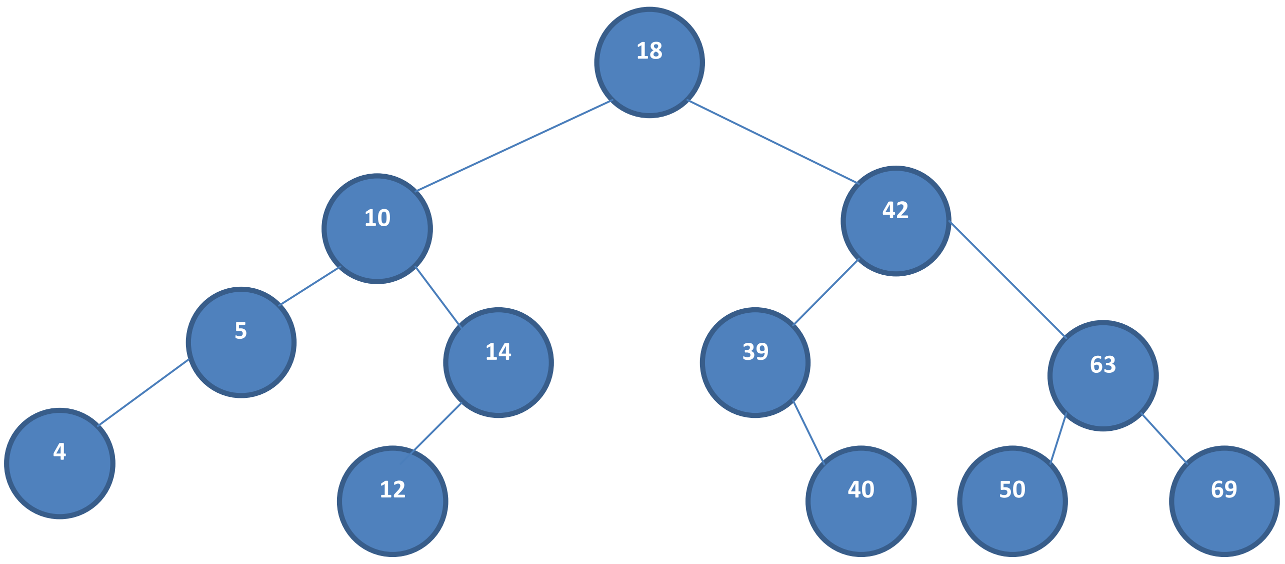


c) **(0, 40 points)** Obtain the balanced tree for the resulting tree in the previous paragraph. Draw only the final (you can include the needed intermediate operations in auxiliary sheets).



Note: In the removing operation we will always consider the ancestor rule.

SOLUTION:



Problem 4 (1 point)
Given a binary search tree with integers, write a recursive method to traverse the tree printing those elements whose grandfather is an even number.

SOLUTION:

```
public static<E> void mostrarAbuelosPar(BinTree<Integer>tree) {
      mostrarAbuelosPar(tree.rootNode);
}

public static<E> void mostrarAbuelosPar(BinTreeNode<Integer>node){

      if (node==null) return;

      if (node.parent!=null&&node.parent.parent!=null&&
                  node.parent.parent.getElement()%2==0) {

            System.out.println(node.getElement());
      }

      mostrarAbuelosPar(node.leftChild);
      mostrarAbuelosPar(node.rightChild);
}
```

Problem 5 (1 point)

Say if the following assertions are true or false. Justify your answer:

1. To define an inheritance relation with a given class, the reserved word ‘extends’ must be used.

   Solution: [✓ True] ☐ False

   Justify:

Yes, to define an inheritance relation, marking the superclass from which a subclass inherits is a must. For example:

public class B extends A {...}

2. Static methods are those requiring the creation of an object through the reserved word 'new' to be able to invoke them.

   Solution: [ ] True [X] False

   Justify:

   On the contrary, invoking a static method does not require a previous call. Static methods are linked to a specific instance of the class.

3. Overloading an operator can only be done if there exists an inheritance relation with a parent class.

   Solution: [ ] True [X] False

   Justify:

   It is not that way, overloading a method allows for the definition of several headers for the same method. The only condition is for those headers having different number of parameters and/or different data types for those parameters.

4. A singly linked list is formed by a sequence of nodes. Each node stores references or pointers to ancestor and successor nodes.

   Solution: [ ] True [x] False

   Justify:

   In a singly linked list, each node is linked only with the successor.

5. A recursive algorithm must always be defined as static (through the reserved word 'static') to be able to invoke it from another class, regardless of the class where the recursive method is defined.

   Solution: [ ] True [X] False

   Justify:

   A method being static or not has nothing to do with recursion.

6. One of the conditions for an algorithm to have a temporal complexity, O, with order n, is that there must be loops making a sequential traversing of input data.

   Solution: [X] True [ ] False

   Justify:

If the algorithm has a loop, we can be sure that the order of the algorithm is, at least, O(n). If there are two nested loops, it would have a quadratic complexity.

7. An algorithm to search in a binary search tree has complexity O(n).

   Solution: ☐ True [X] False

   Justify:

   In a binary search tree, the search space is divided by 2 in each recursive call. The sequence ½, ¼, 1/8, ... has *log n* as limit.

8. For a recursive method to work properly there must be a *recursive case* where the method invokes itself.

   Solution: [X] True ☐ False

   Justify:

   Yes, there must be a recursive call and, also, a base case to assure the end of the recursive process.

9. Given an interface IList, the following Java code should produce a compilation error:

   ```
   IList<Integer> mylist = new IList<Integer>();
   ```

   Solution: [X] True ☐ False

   Justify:

   It is not possible to create an instance for an interface. Interfaces are only "templates" and do not include code.

10. Given the following recursive algorithm to obtain the factorial of a number where all factors are stored in a singly linked list:

```
static long recFactorial (int n) {

        SList<Integer> list = new SList<Integer>();

        if (n == 0) {
           list.add(n);
           return 1;
        }else {
           list.add(n);
           return n * recFactorial(n - 1);
        }
}
```

When the base case is reached, the SLIst ‘list’ will keep all used factors. So, for example, if n=5, ‘list’ will contain the following values: 5,4,3,2,1,0.

Solution: True [✓ False]

Justify:

It must be noticed that in each call, a new ‘list’ is being created and the scope for this list is the current invocation so, that list will not exist for the next recursive call.