DATA STRUCTURE — **DEGREE IN COMPUTER SCIENCE ENGINEERING**
English group — **First progress test – November 16th, 2023**

1) SPECIFICATION: 6+6 = 12% of the final mark (continuous assessment)

a) The following Haskell data declarations are useful for modelling natural and integers:

```
data Nat  = Zero | Suc Nat              deriving Show
data Inte = Z    | S Inte | P Inte      deriving Show
```

Specify operation `addabs:: Inte -> Nat -> Nat`, which returns the natural number representing the absolute value of the sum of an integer and a natural number. Although auxiliary operations are not permitted, to simplify the exercise we can assume that the `Inte` number in the first parameter is ALWAYS IN NORMAL FORM. Examples:

```
addabs (P(P(P Z)))   (Suc Zero) → Suc (Suc Zero)
addabs (S(S(S Z)))   (Suc Zero) → Suc (Suc (Suc (Suc Zero)))
```

b) Consider the following Haskell specification:

```
op23::(Ord a)  => [a] ->[a]
op23  (x:y:s)  =  if (x>y) then s++[x] else x:(op23 (y:s))
op23  l        =  l
```

Write the Haskell terms obtained after evaluating the following expressions:

```
op23 [1,2,3]    →
op23 [3,2,1]    →
op23 [1,3,2]    →
```

2) IMPLEMENTATION: 10+13 = 23% of the final mark (continuous assessment)

Complete the templates of the two methods below according to the following guidelines:

- The java code must be EFFICIENT, ITERATIVE, using NO MORE VARIABLES than the ones declared in the templates and applying the `try{…}catch (TADVacioException ex){ex.printStackTrace();}` command when needed;
- Auxiliar methods can not be used except the following ones declared in the `Lista<E>` interface for manipulating objects belonging to all classes implementing such interface:

```
public boolean EsVacia();                          // checks if a list is empty
public void Añade(E e);                            // inserts an element in a list
public E Cabeza() throws TADVacioException;        // returns the last inserted element on a list
public Lista<E> Cola()throws TADVacioException;    // returns a list without its last inserted element
```

- Objects declared of type `LD<E>` can also access the attributes `private int longitud` (number of element of the list) and `private Nodo<E> NodoCabeza` (link the the last inserted node in the list), as well as the constructor of the `Nodo<E>` class `public Nodo(E e, Nodo<E> n) {Info=e; Siguiente=n;}`.

```
a) public LD(Lista<E> l1, LD<E> l2)
   { Nodo<E> aux=null;
          …. . . // COMPLETE THE JAVA CODE HERE                    }
```

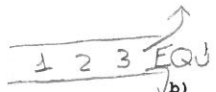
This constructor of the `LD<E>` class generates a list with the elements of two lists `l1` and `l2` after reversing the order of the first one (i.e., `l1`). For instance, if `l1=[1,2,3]` and `l2=[4,5,6]` then, the content of object `this` must be `[3,2,1,4,5,6]`.

```
b) public static <E extends Comparable> Lista<E> op23(Lista<E> l)
   { E x,y;  LD<E> aux1,aux2; aux1=new LD<E>(); aux2=new LD<E>();
          …. . . // COMPLETE THE JAVA CODE HERE
     return new LD<E>(aux1,aux2);                                  }
```

This method implements the previously specified `op23` operation outside all classes implementing the `Lista<E>` interface.

addabs

Stack



Queue

DATA STRUCTURE — **DEGREE IN COMPUTER SCIENCE ENGINEERING**
English group — **First progress test – November 16th, 2023**

SOLUTIONS

```
1) a) addabs:: Inte -> Nat -> Nat
      addabs  Z       x       =   x
      addabs (S x)    y       =   Suc (addabs x y)
      addabs (P x)    (Suc y) =   addabs x y
      addabs (P x)    y       =   Suc (addabs x y)

   b) op23:: (Ord a) => [a] ->[a]
      op23 (x:y:s)= if (x>y) then s ++[x]  else x:(op23 (y:s))
      op23 l      = l

      op23 [1,2,3] → [1,2,3]
      op23 [3,2,1] → [1,3]
      op23 [1,3,2] → [1,3]
```

```
2)
a)
   public LD(Lista<E> l1, LD<E> l2) {
      longitud = l2.longitud;  NodoCabeza = l2.NodoCabeza;
      try { while (!l1.EsVacia()) {
                NodoCabeza = new Nodo(l1.Cabeza(), NodoCabeza);
                l1 = l1.Cola(); longitud++;
            }
      } catch (TADVacioException ex) { ex.printStackTrace(); }
```

```
b)
  public static <E extends Comparable> Lista<E> op23 (Lista<E> l)
{ E x,y;  LD<E> aux1,aux2; aux1=new LD<E>(); aux2=new LD<E>();
    if (l.EsVacia() || (l.Cola().EsVacia())) return l;
    try {
        x = l.Cabeza(); l = l.Cola(); y = l.Cabeza();
        while (!l.Cola().EsVacia() && x.compareTo(y) <= 0 ) {
            aux1.Añade(x);  x = y; y= l.Cabeza();    l = l.Cola();
        }
        if (x.compareTo(l.Cabeza()) > 0 ) {aux2.Añade(x);l= l.Cola();}
        else aux1.Añade(x);
        while (!l.EsVacia()) {
            aux1.Añade(l.Cabeza());   l = l.Cola();
        }
    }catch (TADVacioException e) {return new LD<E>();}
    return new LD<E>(aux1,aux2);
}
```

JAVA CODE FOR TESTING:

```
LD<Integer> l1=new LD<Integer>();l1.Añade(3);l1.Añade(2);l1.Añade(1);
LD<Integer> l2=new LD<Integer>();l2.Añade(6);l2.Añade(5);l2.Añade(4);
System.out.println(" l1=" + l1 + " l2=" + l2 + " new constructor=" +
new LD(l1,l2));

LD<Integer> l3=new LD<Integer>();l3.Añade(3);l3.Añade(2);l3.Añade(1);
System.out.println(" l3=" + l3 + " op23= "+op23(l3));
l3=new LD<Integer>();l3.Añade(1);l3.Añade(2);l3.Añade(3);
System.out.println(" l3=" + l3 + " op23= "+op23(l3));
l3=new LD<Integer>();l3.Añade(2);l3.Añade(3);l3.Añade(1);
System.out.println(" l3=" + l3 + " op23= "+op23(l3));
```

DATA STRUCTURE — **DEGREE IN COMPUTER SCIENCE ENGINEERING**
English group — **First progress test - November 6th, 2020**

1) Consider the following Haskell declarations useful for specifying a three-valued variant of Booleans based on just one NON FREE generator:

```
data Nat                =    Zero        |     Suc Nat
data BoolThreeNat       =    B3N Nat Nat
```

The infinite set of terms built with the **B3N** generator can be partitioned in three equivalence classes representing the following notions of truth:

- *True*: when the first parameter of the term is a **Nat** number strictly bigger than the second one. The canonical representative of this class is **B3N (Suc Zero) Zero**.
- *Maybe*: when the first parameter of the term is a **Nat** number coincides with the second one. The canonical representative of this class is **B3N Zero Zero**.
- *False*: when the first parameter of the term is a **Nat** number strictly smaller than the second one. The canonical representative of this class is **B3N Zero (Suc Zero)**.

Give the set of equations modelling these operations:

a) nfB3N :: BoolThreeNat -> BoolThreeNat, which computes the normal form of a **BoolThreeNat** term without using auxiliary operations.

b) andB3N :: BoolThreeNat -> BoolThreeNat-> BoolThreeNat, which returns the conjunction of two **BoolThreeNat** terms. Calls to **nfB3N** as well as to an auxiliary operation (that must be also specified) are allowed.

Examples:

```
nfB3N (B3N (Suc (Suc Zero))  Zero  )                -> B3N (Suc Zero) Zero
nfB3N (B3N (Suc (Suc Zero)) (Suc (Suc (Suc Zero))) ) -> B3N Zero  (Suc Zero)
nfB3N (B3N (Suc (Suc Zero)) (Suc (Suc Zero))  )     -> B3N Zero  Zero
andB3N (B3N (Suc Zero) (Suc Zero))(B3N (Suc (Suc Zero)) Zero) -> B3N Zero Zero
andB3N (B3N Zero (Suc (Suc Zero)))(B3N (Suc Zero) Zero) -> B3N Zero (Suc Zero)
```

2) Consider the following Haskell specification:

```
queuexam:: Queue Int   ->    Queue Int
queuexam    (q:.x:.y)  =     (q:.y:.x)
queuexam    _          =     queuexam (EQu:.1:.2)
```

a) Write the Haskell terms obtained after evaluating the following expressions:

```
queuexam (EQu:.3)          ---> EQu:.3
queuexam (EQu:.3:.2:.1)    ---> EQu:.3:.1:.2
```

Implement this operation as two different Java methods with the following shapes:

b) **HIGH LEVEL**: `public static Cola<Integer> queuexam (Cola<Integer> q)`
c) *LOW LEVEL*: `public void queuexam ()`

Use only the methods/attributes below accordingly to their **HIGH**/*LOW* labels for each exercise.

```
public interface Cola<E>{
      public boolean EsVacia();                                  -- HIGH
      public void EnCola(E x);                                   -- HIGH
      public E Cabeza() throws TADVacioException;                -- HIGH
      public Cola<E> Resto() throws TADVacioException;           -- HIGH
    }

public class Cola_Dinamica<E> implements Cola<E>{
      public Nodo<E>  NodoCabeza;  public Nodo<E>  NodoFinal;    -- LOW
      public Cola_Dinamica(){NodoCabeza=null;NodoFinal=null;}    -- HIGH
    }

public class Nodo<E> {
      public E Info;  public Nodo<E> Siguiente;                  -- LOW
      public Nodo(E e, Nodo<E> n){Info=e;Siguiente=n;}           -- LOW
    }
```

DATA STRUCTURE — **DEGREE IN COMPUTER SCIENCE ENGINEERING**
English group — **First progress test - November 6th, 2020**

SOLUTIONS

```
1) data BoolThreeNat      =      B3N Nat Nat  deriving Show

a)
nfB3N :: BoolThreeNat -> BoolThreeNat
nfB3N  (B3N Zero Zero)           = B3N Zero Zero
nfB3N  (B3N _ Zero)              = B3N (Suc Zero) Zero
nfB3N  (B3N Zero _)              = B3N Zero (Suc Zero)
nfB3N  (B3N (Suc x) (Suc y))     = nfB3N (B3N x y)

b)
andB3N :: BoolThreeNat -> BoolThreeNat-> BoolThreeNat
andB3N x y =  aux_andB3N  (nfB3N x)  (nfB3N y)

aux_andB3N :: BoolThreeNat -> BoolThreeNat-> BoolThreeNat
aux_andB3N (B3N (Suc Zero) Zero) x                    =  x
aux_andB3N (B3N Zero (Suc Zero)) _                    =  B3N Zero (Suc Zero)
aux_andB3N (B3N Zero Zero)       (B3N Zero (Suc Zero))  =  B3N Zero (Suc Zero)
aux_andB3N _                     _                    =  B3N Zero Zero


2)     queuexam:: Queue Int -> Queue Int
       queuexam (q:.x:.y)  = (q:.y:.x)
       queuexam _ = queuexam (EQu:.1:.2)

a)  queuexam (EQu:.3)            -> (EQu :. 2) :. 1
    queuexam (EQu:.3:.2:.1)      -> ((EQu :. 3) :. 1) :. 2

b)
 public static Cola<Integer>  queuexam (Cola<Integer> q)
  { Cola_Dinamica<Integer> sol=new Cola_Dinamica<Integer>();
     try {
      if (q.EsVacia() || q.Resto().EsVacia())
         { sol.EnCola(2);sol.EnCola(1);}
      else
         { while (!q.Resto().Resto().EsVacia())
             { sol.EnCola(q.Cabeza()); q=q.Resto();}
           Integer x; x= q.Cabeza(); q=q.Resto();
           sol.EnCola(q.Cabeza());sol.EnCola(x);
         }
    }  catch (TADVacioException ex) { System.out.println("TAD lineal vacio');
   return sol;
   }

OTHER VERSION (which perhaps is more elegant)

  public static Cola<Integer> queuexam (Cola<Integer> q)
    {Cola_Dinamica<Integer> sol=new Cola_Dinamica<Integer> );
        try {  while (!q.Resto().Resto().EsVacia())
                    { sol.EnCola(q.Cabeza()); q=q.Resto();}
               Integer x; x= q.Cabeza(); q=q.Resto();
               sol.EnCola(q.Cabeza());sol.EnCola(x); return sol;
            } catch (TADVacioException ex)
               { sol.EnCola(2);sol.EnCola(1); return sol;}
    }

c)
  public void queuexam ()
   {if (NodoCabeza==null || NodoCabeza.Siguiente==null)
       { NodoFinal=new Nodo(1, null);  NodoCabeza=new Nodo(2, NodoFinal); }
    else
       { Nodo<E> aux=NodoCabeza;
         while (aux.Siguiente!=NodoFinal) aux=aux.Siguiente;
         E x; x=aux.Info; aux.Info=NodoFinal.Info;NodoFinal.Info=x;
   }
```

```
op22 (q:.x) s          = x:/(op22 q s)
op22  q   (x:/y:/s)    = y:/ (op22 q(x:/s))
op22  __   __          = ES
```

En Haskell NO desapila

op22 (Equ:.1:.2:.3) (4:.ES)

3:/ (op22 EQu:.1:.2) (4:.ES)) (1)
3:/2:/ (op22(EQu:.1) (4:.ES)) (1)
3:/2:/1:/ (op22(EQu) (4:.ES)) (2)
3:/2:/1:/ES

op22 EQu (4:/5:/6:/ES) (2)
5:/ op22 EQu (4:/6:/ES) (2)
5:/6:/op22 EQu (4:/ES) (2)
5:/6:/ES

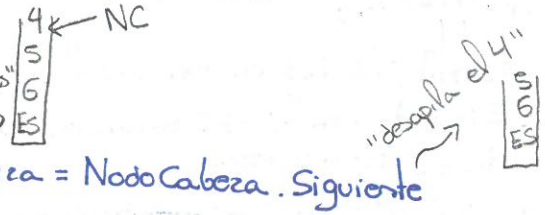
op22 (EQu:.1:.2:.3) (4:/5:/6:/ES)
3:/2:/1:/ op22 EQu (4:/5:/6:/ES) (2)
3:/2:/1:/5:/ op22 EQu (4:/6:/ES) (2)
3:/2:/1:/5:/6:/ Op22 EQu (4:/ES) (2)
3:/2:/1:/5:/6:/ES

a) High level in JAVA

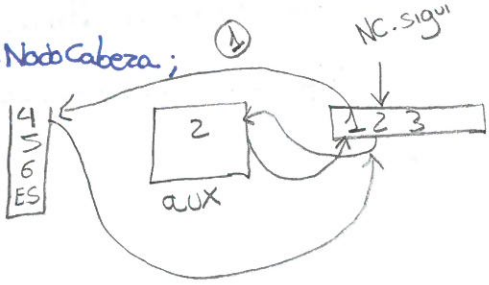
```
public static <E> Pila<E> op22 (Cola<E> q, Pila<E> s){
    Pila<E> s1; Cola<E> q1;
    Pila<E> s1 = Pila<E>s.clone();   { Para no modificar originales
    Cola<E> q1 = Cola<E>q.clone();
try} if(!s1.EsVacia){} s1 = s1.Desapila(){ → Si hay algo NO se puede acceder directamente
    while(!q1.EsVacia()){}
        s1.Apila(q1.Cabeza()) → Introduce el primer elemento de la cola
        q1 = q1.Resto(); → q1 se actualiza con el resto de la cola
    {catch(...)}...{
    return s1;
{
```

b) Low level in JAVA

NodoCabeza → Pila
q.NodoCabeza → Cola




```
public void op22 (CD<E> q){
    Nodo<E> aux=null;
    if (nodoCabeza!=null) NodoCabeza = NodoCabeza.Siguiente
    while(q.NodoCabeza != null)
        }aux = q.NodoCabeza.Siguiente; q.NodoCabeza.Siguiente=NodoCabeza; (1)
         NodoCabeza = q.NodoCabeza; q.NodoCabeza = aux;
         {
```




```
subI  Z   (P Z) → S(subI x y)
subI  Z   (S Z) → P(subI x y)
subI (S Z) Z    → X
subI (P Z) Z    → X
subI (S Z) (S Z) → subI x y { Va quitando "S" y "P" hasta que se desiguales
subI (P Z) (P Z) → subI x y
```

```
mix21:: stack a → Queue a → Queue a
mix21 (x:/s)  (q:.y) = (mix21 (y:/s) q) :.x
mix21 (x:/s)  q      = mix s q
mix21 ES      q      = q
```

(q:.y) → último elemento; ↳ el resto

a) mix21 (1:/2:/ES) (EQu:.3:.4:.5)

1- No modificar la cola
2- Todo de la clase E

(1) mix21 (5:/2:/ES) (EQu:.3:.4) :.1

(1) mix21 (4:/2:/ES) (EQu:.3):.5 :.1

(2) mix21 (3:/2:/ES) EQu :.4:.5:.1

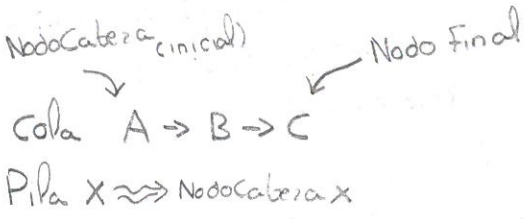
(2) mix21 (2:/ES) EQu

(3) mix21 ES EQu → EQu:.4:.5:.1

b) High level in JAVA

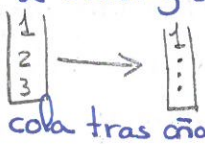
recibe 2 argumentos, Stack y Cola

```
public static <E> Cola <E> mix21 (Pila <E>s, Cola <E> q){
    cola <E> q2 = (Cola <E>) (q.clone()); → No queremos tocar la cola original, creamos una nueva
    if (!s.EsVacia() && !q2.EsVacia()){ → Comprueba que no haya elementos vacíos
        q2 = q2.Resto(); → Toma todos menos la cabeza "EQu:.3:.4:.5"
        q2.EnCola (s.Tope()); → Toma lo de arriba del stack y lo añade a la queue
    }
    return q2;
}
```

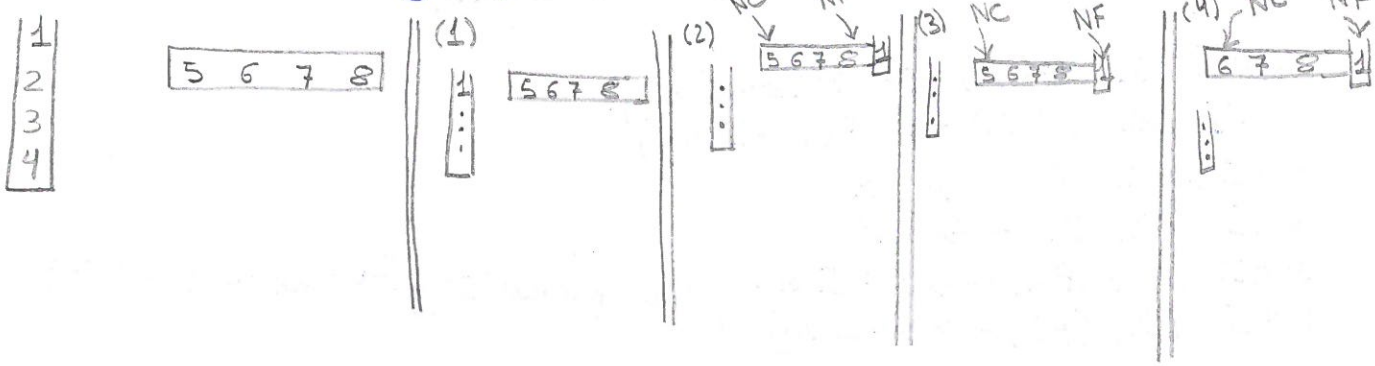


c) Low level in JAVA instancia clase PD<E>

```
public void mix21 (PD<E> s){
    if (s.NodoCabeza != null && Nodocabeza != null ) }→Comprueba no nulo de stack y cola
    (1) s.NodoCabeza.Siguente = null; →Desconecta la cabeza del stack
    (2) NodoFinal.Siguente = s.NodoCabeza; →Actualiza el puntero final a la antigua cabeza del stack
    (3) NodoFinal = NodoFinal.Siguente; → Mete 1er elemento de stack
    (4) NodoCabeza = NodoCabeza.Siguente;
    {
{
```



cola tras añadir



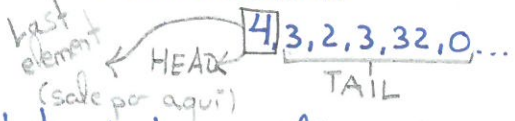
Fundamental Abstract Data Types

✧ List ADT:

Linear sequence of elements with order relation, with a set of operations to manipulate.

Head: 1st element in the list, last introduced. LIFO (Last In First out)

Tail: The rest elements




```
data List a = Nil | Cons a (List a)
```

List a ⟶ [a]

Nil ⟶ [] Empty list

Cons ⟶ : (infix) Free generator (consecutive)

1 : 2 : [] can be expressed as [1,2]

['a', 'b', 'c'] can be expressed as "abc"

_ ⇒ Anom variable (whatever)

- Operations no generators

```
Empty :: [a] → Bool  // Checks if true
Head :: [a] → a      // Returns head
Tail :: [a] → [a]    // Returns tail
```

```
head (h:_) = h
tail (_:t) = t
```

```
Length :: [a] → int
Concatenate :: [a] → [a] → [a]  // Seen as ++
Reverse :: [a] → [a]
```

```
length (h:t) = 1 + length t
concatenate (h:t) s = h : concatenate t s
reverse (h:t) = concatenate (reverse t) [h]
```

```
Last :: [a] → a
Elem :: (Eq a) ⇒ a → [a] → Bool
Extract :: (Eq a) ⇒ a → [a] → [a]
```

```
elem x (h:t) = (x == h) || elem x t
extract x (h:t) = if (x == h) then extract x t else h : extract x t
```

```
Take :: Int → [a] → [a]  // Only "saves" the Int elements from left
Cut :: Int → [a] → [a]   // Deletes int elements from left
```

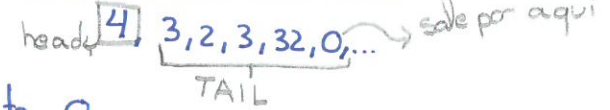
```
take _ [] = []
take n (h:t) = h : take (n-1) t
cut n (h:t) = cut (n-1) t
```

```
Substitute :: (Eq a) ⇒ a → a → [a] → [a]
// substitute x y: changes all X's in [a] by Y's
Sum :: [Int] → Int  // Sum of all numbers
Greater :: [Int] → Int  // greater: :(ord a) ⇒ [a] → a ...
// greater (x:y:t) = if x > y then grt (x:t), else grt (y:t)
```

```
substitute x y (h:t) = if h == x
    then y : substitute x y t
    else h : substitute x y t
```

✧ Queue ADT

FIFO (First In First out)



Put (Put (Put (Put EQ 2) 7) 5) 7 ← Head

↳ EQ :. 2 :. 7 :. 5 :. 7

```
data Queue a = EQ | Put (Queue a) a
EmptyQ :: Queue a → Bool
// EmptyQ EQ = True
// EmptyQ _ = False
```

```
First :: Queue a → a
// First (EQ :. x) = x
// First (c :. x) = First c
```

Recordar que en cola: el primero está al final

```
Rest :: Queue a → Queue a
// rest (EQ :. x) = EQ
// rest (c :. x) = (rest c) :. x
```

```
first (EQ :. 3 :. 5 :. 2 :. 7) ⟶ 3
rest (EQ :. 3 :. 5 :. 2 :. 7) ⟶ (EQ :. 5 :. 2 :. 7)
```

Stack ADT (Pilas)

Based on sequential access, represented vertically

→ TOP (first element)

A "pop" is neccessary to access the remaining elements

3 :/ (2 :/ (5 ... :/ ES)))) → 3 :/ 2 :/ ... :/ ES

Empty S:: Stack a → Bool Top:: Stack a → a (≃ Head) Pop:: Stack a → Stack a (≃ Tail)

→ Se debe desapilar antes de acceder

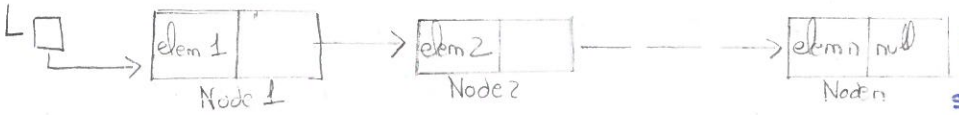
```
elimina :: a → Stack a → Stack a
elimina x ES = ES
elimina x (y :/ p) = if x == y then p, else y :/ (elimina x p)
```

Toca el interior de la pila
↳ + apila x p = x :/ p
↳ elimina x (y :/ p) = if x == y then p
else apila y (elimina x p)

Implementation of lists

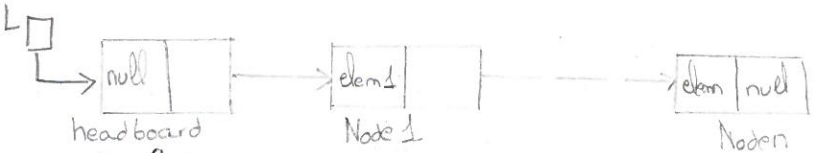
In java, lists are interface List<E>, E is the class of elements in the list.
↳ must be extended for any class implementation list

- Dynamic implementation



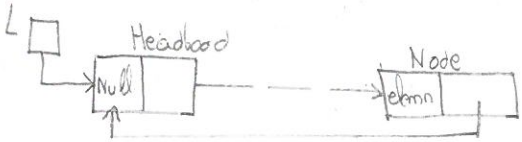
Node 1..., Node N instances of Node<E> class,
L instance of List<E> interface.
Elem 1, elem N... elements of class E to be stored in the list

- With headboard

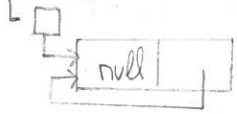


Avoids asymmetry in first node, 1st unique without predecessor

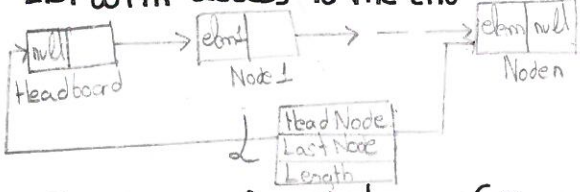
- Circular lists



- Empty list



- List with access to the end



Useful for concatenations or inserting after the last element avoiding to go through all nodes

Dynamic implementation of Queues

2324 First Progress Test

```
data Nat = Zero | Suc Nat
```
→ Números Naturales ⇒ (Num >= 0)

```
data Inte = Z | S Inte | P Inte
```
→ Números enteros (Positivos y negativos)

```
addabs :: Inte → Nat → Nat
```
→ Metemos un entero y un natural, obteniendo otro natural

① `addabs Z x = x` → Siempre será positivo, ya que el primer valor (Inte) podría tomar cualquier valor, pero esta puesto a cero (Z) y el segundo (Nat) sabemos (x >= 0)

Toy example:

`addabs Z (Suc(Zero)) → Suc Zero`

② `addabs (S Z) x = Suc(addabs x y)` → Siempre será positivo por el mismo motivo. La recursividad "despeja" una S, luego Z = x; x = y

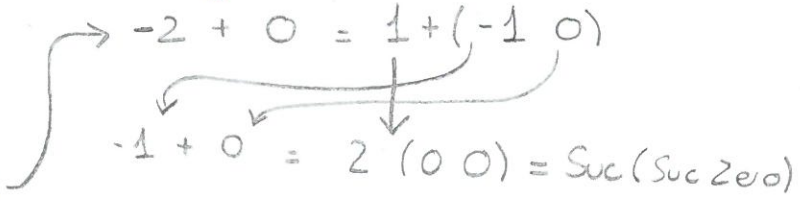
Toy example

`addabs (S(S Z)) (Suc Zero) → (Suc (Suc(Suc Zero)))`

③ `addabs (P Z) (Suc Zero) = addabs x y`

`addabs (P Z) (Zero) = Suc(addabs x Zero)` → Every time the code finds a P, it will substract 1 and add 1

```
addabs :: Inte → Nat → Nat
addabs  Z x = x
addabs (S Z) y = Suc(addabs x y)
addabs (P Z) (Suc Zero) = addabs x y
addabs (P Z) Zero = Suc(addabs x y)
```




```
op23 :: (Orda) → [a] → [a]
op23 (x:y:z) = if (x > y) then s++ [x] else x:(op23 (y:s))
op23 1 : 1
```

"s" concatena el elemento "x"

`op23 [1,2,3]` → `[1,2,3]`

`op23 [3,2,1]` → `[1,3]` → `3 > 2 ; [1] ++ [3] ; [1,3]`

`op23 [1,3,2]` → `[1,3]` → `1 : 1 ; 3 > 2 ; [1] ++ [3] ; [1,3]`

↳ `1 : (op23 [3 : 2 : []])` ⇒ `1 : [] ++ [3]`

(x) (y) (s)

L1 y L2, dif class

Using times :: Nat → Nat → Nat // times a → a → a from practice 1

times (done)

times : a → a → a ## We use a parameter, we obtain one (a)

facto: Nat → Nat ## We introduce one natural, we obtain another one

facto: Zero → SucZero ## If we obtain zero, the result is one → 0! = 1

facto: Suc (x) → times (fact x) (suc x) ## The natural is introduced

↓ Num ≠ 0

↓↓ n! = n · (n-1) recursively

by example:

facto Suc (Suc x) → times (fact (Suc x) Suc (suc x)

2! → 1 · 2

If higher than 2, this result would be save. For example 3 would be 6 · 1 at the end, after 3 · 2

2021) 6)

```
public static <E> Cola<E> mix21 (Pila<E> s, Cola<E> q1)
    Cola<E> q1 = (Cola<E>) q1.clone();
    if(!s.EsVacia() && !q1.esVacia())
      q1 = q1.Resto();
      q1.EnCola(s.Tope());
    {
  return q1;
{
```

1 - Clono
2 - Hay vacías?
3 - Apilo/desapilo lo nuevo
4 - Devuelvo

2022) a)

```
public static <E> Pila<E> op22 (Cola<E> q, Pila<E> s)
  Pila<E> s1 = (Pila<E>) s.clone();
  Cola<E> q1 = (Cola<E>) q.clone();
   if(!s1.EsVacia()) { s1 = s1.Desapila(); {
   while(!q1.EsVacia())
     s1.Apila(q1.Cabeza());
     q1 = q1.Resto();
    {
  return s1;
{
```

1) The following specification of the Boolean ADT is based on two NON free generators, where the second one reflects the classical NAND (negation of the conjunction) operation:

```
data BoolNAND = Yes | NAND BoolNAND BoolNAND
```

The canonical representatives of the two equivalence classes representing the notions of *true* and *false* are, respectively, the terms 'Yes' and 'NAND Yes Yes'. Specify the following operations --whose outputs must be always in normal form-- for normalizing a term of this ADT as well as for evaluating an operation (considered now a modifier, not a generator) which coincides again with the negated conjunction of Booleans:

```
nfBoolNAND  :: BoolNAND -> BoolNAND
nand        :: BoolNAND -> BoolNAND -> BoolNAND
```

Examples:

```
nfBoolNAND (NAND (NAND Yes (NAND Yes Yes)) (NAND Yes Yes))-> Yes
nand  Yes  (NAND (NAND Yes (NAND Yes Yes)) (NAND Yes Yes))->NAND Yes Yes
```

2) Consider the following Haskell specification:

```
mix21 :: Stack a -> Queue a ->  Queue a
mix21 (x:/s)   (q:.y)       =     (mix21 (y:/s) q) :. x
mix21 (x:/s)   q            =     mix21 s q
mix21 ES       q            =     q
```

a) Write the Haskell terms obtained after evaluating the following expressions:

```
mix21 ES              (EQu:.3:.4:.5)   →
mix21 (1:/2:/ES)       EQu             →
mix21 (1:/2:/ES)      (EQu:.3)         →
mix21 (1:/2:/ES)      (EQu:.3:.4:.5)   →
```

Implement this operation as two EFFICIENT Java methods with the following shapes:

b) HIGH LEVEL (use only the methods declared in the Pila and Cola interfaces):

```
public static <E> Cola<E> mix21 (Pila<E> s, Cola<E> q)
```

c) LOW LEVEL (use only the attributes declared in CD, PD and Nodo classes):

```
public void mix21 (PD<E> s)
```

```
public interface Pila<E>{
      public boolean EsVacia();
      public void APila(E e);
      public E Tope() throws TADVacioException;
      public Pila<E> DesaPila()throws TADVacioException;          }
public interface Cola<E>{
      public boolean EsVacia();
      public void EnCola(E x);
      public E Cabeza() throws TADVacioException;
      public Cola<E> Resto() throws TADVacioException;            }
public class PD<E> implements Pila<E>{
      public Nodo<E>  NodoCabeza;
      public PD(){NodoCabeza=null;}                               }
public class CD<E> implements Cola<E>{
  public Nodo<E>  NodoCabeza;  public Nodo<E>  NodoFinal;
  public CD(){NodoCabeza=null;NodoFinal=null;}                    }
public class Nodo<E> {
  public E Info;  public Nodo<E> Siguiente;
  public Nodo(E e, Nodo<E> n){Info=e;Siguiente=n;}                }
```


SOLUTIONS

1) `data BoolNAND = Yes | NAND BoolNAND BoolNAND deriving Sho`

■ Versions using auxiliary operations

```
   → nfBoolNAND, aux_nfBoolNAND:: BoolNAND -> BoolNAND
nfBoolNAND Yes          = Yes
nfBoolNAND (NAND x y)= aux_nfBoolNAND
                              (NAND (nfBoolNAND x) (nfBoolNA D y))

aux_nfBoolNAND(NAND Yes Yes ) = NAND Yes Yes
aux_nfBoolNAND(NAND _  _ )    = Yes

   → nand, aux_nand:: BoolNAND -> BoolNAND -> BoolNAND
nand x y = aux_nand (nfBoolNAND x) (nfBoolNAND y)

aux_nand Yes Yes = NAND Yes Yes
aux_nand _ _     = Yes
```

■ Versions NOT using auxiliary operations

```
   → nfBoolNAND:: BoolNAND -> BoolNAND
nfBoolNAND  Yes                                  = Yes
nfBoolNAND (NAND Yes              Yes)           = (NAND Y S  Yes)
nfBoolNAND (NAND Yes              (NAND Yes  Yes)) = Yes
nfBoolNAND (NAND (NAND Yes  Yes)  Yes)           = Yes
nfBoolNAND (NAND (NAND Yes  Yes)  (NAND Yes  Yes)) = Yes
nfBoolNAND (NAND x                y)             = nfBoo N ND
                                  (NAND (nfBoolNAND x) (nfBo NAND y))

   → nand :: BoolNAND -> BoolNAND -> BoolNAND
nand  x y =  nfBoolNAND   (NAND x y)
```

2)

```
a) mix21 ES            (EQu:.3:.4:.5)   →   EQu :. 3 :. 4 :. 5
   mix21 (1:/2:/ES)     EQu             →   EQu
   mix21 (1:/2:/ES)    (EQu:.3)         →   EQu :. 1
   mix21 (1:/2:/ES)    (EQu:.3:.4:.5)   →   EQu :. 4 :. 5 :. 1
```

```
b) public static <E> Cola<E> mix21 (Pila<E> s, Cola<E> q)
          { Cola<E> q2= (Cola<E>)(q.clone());
            if (!s.EsVacia() && !q2.EsVacia())
                {q2=q2.Resto();q2.EnCola(s.Tope());}
            return q2;
          }
```

```
c) public void mix21 (PD<E> s)
    {  if (s.NodoCabeza!=null  && NodoCabeza!=null)
        {s.NodoCabeza.Siguiente=null;                    (*)
         NodoFinal.Siguiente=s.NodoCabeza;               (**)
         NodoFinal=NodoFinal.Siguiente;
         NodoCabeza=NodoCabeza.Siguiente;
        }
    }
```

(*)(**) Also valid: `NodoFinal.Siguiente=new Nodo<E>(s.NodoCabeza.Info,null);`

1) SPECIFICATION: 10+3 = 13% of the final mark (continuous assessment)

---

a) The following data declaration (based on NON FREE generators) is useful for modelling integers: `data Inte = Z | S Inte | P Inte deriving Show`. WHITOUT using auxiliary operations, specify a substraction operation `subI:: Inte -> Inte -> Inte`, such that: if the inputs are not in normal form, then the output doesn't need to be in normal form but, if the inputs are in normal form, then the output must be also in normal form. Examples:

- `subI (P (P (S Z))) (S (P (S (S Z))))` → `P (P (P (P (S Z))))`
- `subI (P Z) (S (S Z))` → `P (P (P Z))`

b) Consider the following Haskell specification:

```
infixr 5 :/     data Stack a = ES | a :/ (Stack a)  deriving Show
infixl 5 :.     data Queue a = EQu | (Queue a) :. a  deriving Show
      op22:: Queue a -> Stack a -> Stack a
      op22 (q:.x) s             = x:/(op22 q s)
      op22  q     (x:/y:/s)     = y:/(op22 q (x:/s))
      op22  _     _             = ES
```

Write the Haskell terms obtained after evaluating the following expressions:

```
op22  (EQu:.1:.2:.3)   (4:/ES)          → 3:/2:/1:/ES
op22   EQu             (4:/5:/6:/ES)    → 5:/6:/ES
op22  (EQu:.1:.2:.3)   (4:/5:/6:/ES)    → 3:/2:/1:/5:/6:/ES
```

2) IMPLEMENTATION: 11+11 = 22% of the final mark (continuous assessment)

Implement the previous **op22** operation as two EFFICIENT Java methods according to the following guidelines: 1) no more variables than the ones declared in the templates are permitted; 2) the `try{…}catch (TADVacioException ex){ex.printStackTrace();}` command must be appropriately used when needed; 3) the input parameters in the HIGH level method will remain unchanged after calling such method and 4) the LOW level method will be able to alter the content of the input parameter as well as the content of object `this` inside the PD class:

a) HIGH LEVEL (use only the methods declared in the Pila and Cola interfaces):

```
public static <E> Pila<E> op22 (Cola<E> q, Pila<E> s)
      { Pila<E> s1; Cola<E> q1; . . .          }
```

b) LOW LEVEL (use only the attributes of the CD, PD and Nodo classes)

```
public void op22 (CD<E> q)
      { Nodo<E> aux=null;       . . .          }
```

```
public interface Pila<E>{
      public boolean EsVacia();
      public void APila(E e);
      public E Tope() throws TADVacioException;
      public Pila<E> DesaPila()throws TADVacioException;
      public Object clone();                                    }
public interface Cola<E>{
      public boolean EsVacia();
      public void EnCola(E x);
      public E Cabeza() throws TADVacioException;
      public Cola<E> Resto() throws TADVacioException;
      public Object clone();                                    }
public class PD<E> implements Pila<E>{
      public Nodo<E>  NodoCabeza;                               }
public class CD<E> implements Cola<E>{
      public Nodo<E>  NodoCabeza;  public Nodo<E>  NodoFinal;   }
public class Nodo<E> {
      public E Info;               public Nodo<E> Siguiente;    }
```

DATA STRUCTURE — **DEGREE IN COMPUTER SCIENCE ENGINEERING**
English group — **First progress test – November 8th, 2022**

SOLUTIONS

1)

```
a) subI  x      Z =  x
   subI  (S x) (S y)  =  subI x y
   subI  x     (S y)  =  P (subI x y)
   subI  (P x) (P y)  =  subI x y
   subI  x     (P y)  =  S (subI x y)


b) op22  (EQu:.1:.2:.3) (4:/ES)        --> 3 :/ (2 :/ (1 :/ ES))
   op22   EQu   (4:/5:/6:/ES)          --> 5 :/ (6 :/ ES)
   op22  (EQu:.1:.2:.3) (4:/5:/6:/ES) --> 3:/(2:/(1:/(5:/(6:/ES))))
```

2)

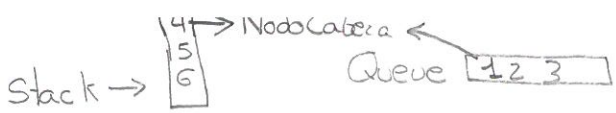
a)

```
public static <E> Pila<E> op22 (Cola<E> q, Pila<E> s)
{ Pila<E> s1= (Pila<E>)s.clone(); Cola<E> q1=(Cola<E>)q.clone();
  try {
      if (!s1.EsVacia()) s1 = s1.DesaPila();
      while (!q1.EsVacia()) {
          s1.APila(q1.Cabeza());
          q1 = q1.Resto();
      }
  }catch (TADVacioException ex) { ex.printStackTrace(); }
       return s1;
}
```

b)

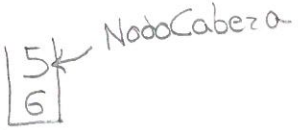
```
public void op22 (CD<E> q)
{ Nodo<E> aux=null;
  if (NodoCabeza!=null)  NodoCabeza=NodoCabeza.Siguiente;
  while (q.NodoCabeza!=null)
     {aux=q.NodoCabeza.Siguiente;q.NodoCabeza.Siguiente=NodoCabeza;
      NodoCabeza=q.NodoCabeza; q.NodoCabeza=aux;}
}
```

```
JAVA CODE FOR TESTING:

CD<Integer> q1=new CD<Integer>();
                q1.EnCola(1); q1.EnCola(2);q1.EnCola(3);
PD<Integer> s1=new PD<Integer>();
                s1.APila(6); s1.APila(5); s1.APila(4);

System.out.println("HIGH op22 (EQu:.1:.2:.3) (4:/5:/6/:ES)  --> 3 :. 2
:. 1 :. 5:/ 6 :/ ES  " + op22(q1,s1));

s1.op22(q1);
System.out.println("LOW op22 (EQu:.1:.2:.3) (4:/5:/6/:ES)  --> 3 :. 2
:. 1 :. 5:/ 6 :/ ES   " + s1);
```



Si stack no vacío

"Desapilamos el 4"




```
public void op22 (CD <E> q)
    Nodo<E> aux = null;
if (nodoCabeza != null)} NodoCabeza = NodoCabeza . siguiente
```

"Mientras haya elementos en cola"

```
while (q.NodoCabeza != null)}
    aux = q.NodoCabeza.Siguiente;    q.NodoCabeza.Siguiente = Nodocabeza;
```

Se desconecta la cabeza de Queue (1)

2 3

BUCLE

```
NodoCabeza = q.NodoCabeza ;    q.NodoCabeza = aux;
```

↳ Se mete el 1 en la pila

{

1/ a) addabs :: Inte → Nat → Nat

addabs Z x = x

addabs (S z) y : Suc (addabs x y

addabs (P z) (Suc y) = addabs x y

addabs (P z) y = Suc (addabs x y)

b) op23 :: (ord a) => [a] → [a]

op23 (x:y:s) = if (x > y) then s ++ [x]
else x : (op23 (y:s))

op23 l = l

op23 [1,2,3] (1)
1 : (op23 (2:3)) (1)
1:2 (op23 (3:_)) (1)
[1:2:3]

op23 [3,2,1]
[3:1] (2)

op23 [1,3,2]
1 : (op23 (3:2)) (2)
1:3
↳ NO hay s

2/ L1 = 1,2,3 L2 = 4,5,6 → [3,2,1,4,5,6] Necesito: 1- Nuevo nodo 2- Amplia long ↳ introduciendo L1 a L2

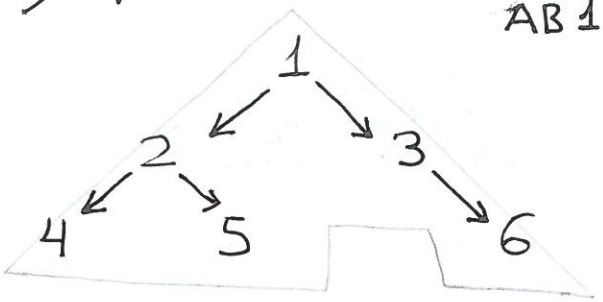
```
public LD (Lista <E> L1, LD<E> L2)}
  longitud = L2.longitud;  NodoCabeza = L2.NodoCabeza;
     try } while (!L1.EsVacia()) }        1
          NodoCabeza = new Nodo (L1.Cabeza(), NodoCabeza);
          { L1 = L1.Cola(); longitud++; 2
     { catch (...) } ... {  → Importantísimo!!! Para recorrer array y quito usado
  {
```

```
public static <E> Pila <E> op22 (Cola <E> q, Pila <E> s)}
   Pila <E> s1; Cola <E> q1;
    s1 = (Pila <E>) s.clone(); q1 = (Cola <E>) q.clone();
    try} if(!s1.EsVacia)} s1 = s1.Desapila(){
      while(!q1.EsVacia)}
      s1.Apila(q1.Cabeza);
      q1 = q1.Resto();
     { catch(...)} ... {
   return s1;
  {
```

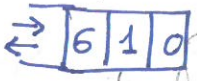
2) a)

Todas son enteros 22-23

AB 1 (AB 2 (AB 4 AV AV) (AB 5 AV AV)) (AB 3 AV (AB 6 AV AV))



Output list: [6,1,0]



→ Y el 0???

↳ el 1??

90% son así: BC → esVacio? Check left recurs → Calling A i; Check right, recurs → Calling Ad { Tarda, el más lleve un hueco (l>r) || (r>l)

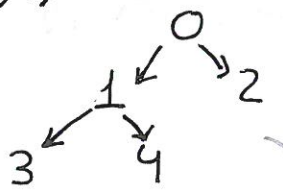
```
public static <E> Lista<Integer> hole(ArbolBinario<E> t){
  int l, r; Lista<Integer> sol, left, right; sol = new LD()<>();
    if(t.EsVacio()) sol.Añade(0); // Indica que está vacío
     else try{ left = hole(t.SubarbolIzqo()); right = hole(t.SubArbolDcho());
                ↓                              ↓
        // El método hole cuenta los nodos, los almacena en left/right
                                  ↳ // Los almacena en la cabeza
          l = left.Cabeza(); r = right.Cabeza()
            if(l<r){sol = left.Cola(); sol.Añade(0);}   Condición previa en enunciado
            else if(l>r){sol = right.Cola(); sol.Añade(1);}
            sol.Añade(l+r+1)
     {catch(TADVacioExcepcion ex){ex.print...();}
     return sol;
     {
```

6 → 1 → primero va a la derecha [1]

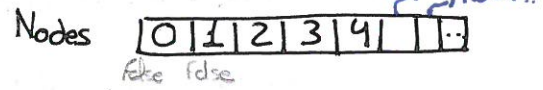
6 → 0 → Luego a la izquierda (del 6) [1, 0]

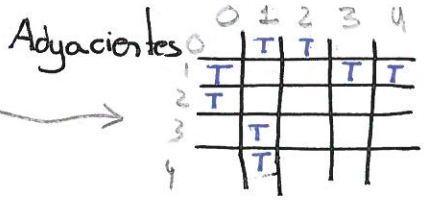
No general trees

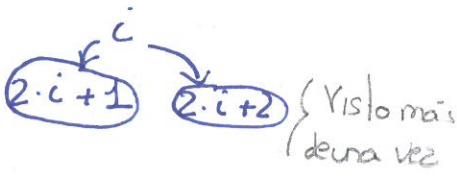
2) b)



NodoGrafo<E>

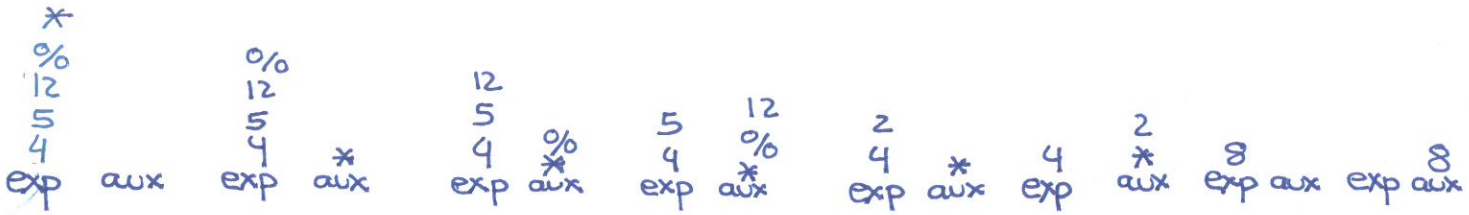




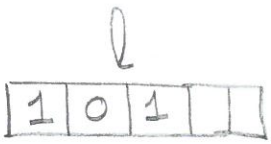


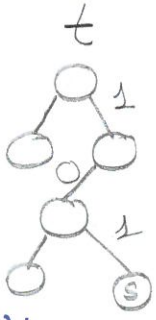
PN (12 % 5) * 4

↳ * (% 12 5) 4 = * % 12 5 4




```
try} while(!exp.EsVacia()){
     if(exp.Tope() instanceof Double && !aux.EsVacia() && aux.tope() instanceof Double){
        x = (Double) exp.tope();
       exp = exp.Desapila();
        y = (Double) aux.tope();
       aux = aux.Desapila();
        op = (String) aux.tope();
       aux = aux.Desapila();
     exp.Apila(OpEval(x, op, y));
    else} aux.Apila(exp.tope());
      exp = exp.Desapila();
    {
  {catch(Tad)}...{
 {return (Double) aux.tope();
{
```






```
     if(!t.EsVacio()){
       try} if(l.EsVacio()){
              if(t.subarbolDer().EsVacio()){ p = t.Raiz();
            else if(l.Cabeza() == 0){
              p = metodo(t.subarbolIzq(), l.Cola()){
            else} p = metodo(t.subarbolDer(), l.Cola()){
         {catch(...)}...{
       {return p;
     {
```

Pasos para crear un Huffman tree

→ Paso lista y árbol

→ Compruebo que el 2º elem. de la lista de árboles no sea nulo

→ Lo es? Devuelvo el árbol único con sol = l.Cabeza(); ↗ Izq ↗ Derc

→ No lo es = Creo ABD con new ABD(raiz1 + raiz2, Arb1, Arb2)

→ Inserto ABD en lista x = insert(ABD, x);

DATA STRUCTURE — **DEGREE IN COMPUTER SCIENCE ENGINEERING**
English group — **Progress test on practices – December 21th 2023**

NAME: _____

Java interfaces that can be used in this exam

```
public interface Lista<E> {
   public boolean EsVacia();
   public void Añade(E x);
   public E Cabeza() throws TADVacioException;
   public Lista<E> Cola() throws TADVacioException;
   public void Concatena(Lista<E> l);
   public Object clone();
}
public interface ArbolBinario<E> extends Arbol<E>
   public boolean EsVacio();
   public E Raiz() throws TADVacioException;
   public ArbolBinario<E> SubArbolIzqdo() throws TADVacioException;
   public ArbolBinario<E> SubArbolDcho() throws TADVacioException;
}
```

PN: 2+((12%5)*3) → +2[(12%5)*3] → +2*[(12%5)3] → +2*%1253

RPN: 2+((12%5)*3) → 2((12%5)*3)+ → 2[(12%5)3)*]+ → 2[(125%3)*]+ → 2 12 5 %3* +

---

1) [Reverse Polish Notation] In this exercise we are concerned with the second practice devoted to evaluate an arithmetic expression stored into a general stack once expressed in RPN (*reverse Polish notation*). Given the arithmetic expression **`2+((12%5)*3)`**, complete the following Java code for initializing a stack representing the same expression in reverse Polish notation:

```
Pila s = new PD();   s.APila( + ); s.APila( * ); s.APila( 3 );
     s.APila( % ); s.APila( 5 ); s.APila( 12 ); s.APila( 2 );
```

In the table below, use only the cells required to represent the contents of stacks **`exp`** and **`aux`** after finishing each iteration when calling the method explained in class with the previous stack:

ITE. 1		ITE. 2		ITE. 3		ITE. 4		ITE. 5		ITE. 6		ITE. 7		ITE. 8		ITE. 9		ITE. 10	
12																			
5		5																	
%		%		%															
3		3		3	5	3			3										
*		*	12	*	12	*	2	*	2		6								
+	2	+	2	+	2	+	2	+	2	+	2		8						
exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	Aux

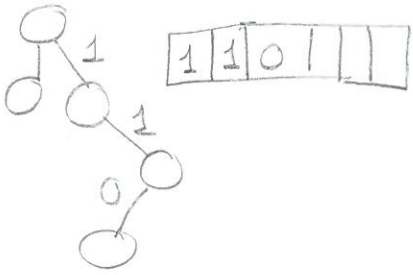
2) [Huffman trees] Complete in the next page the Java code of method **`code01`** which receives a Huffman tree **`t`** with at least two leaves and the ASCII code of a symbol **`s`** and returns a list only containing 0's and 1's representing the Huffman code of **`s`** if such symbol is included in the tree **`t`**. Otherwise, the output must be the list **`[-1]`**. The number of visited nodes in the tree must be minimized.

3) [Floyd's algorithm] A floydij in a weighted directed graph is a path connecting two NON ADJACENT nodes i and j and verifying that its weight is: 1) SMALLER that the weight of any other path between i and j and 2) BIGGER than the double of the weight of its first arrow. Complete in the next page the Java code of method **`floydijs`** which returns the number of floydijs found on a graph represented by its adjacency matrix.

```
public static Lista<Integer> code01 (ArbolBinario<Integer> t, Integer s)
{   Lista<Integer> l, laux; l=new LD(); ArbolBinario<Integer> aux=new ABD<>();









    return l;
}
```

```
public static int floidijs (int[][] adj)
 {int inf = Integer.MAX_VALUE, n = adj.length, i = 0, j = 0, k = 0, num = 0;
  int[][] paw = (int[][])adj.clone();   Lista<Integer>[][] pan = Floyd(paw);









      return num;
   }
```

```
public static int percent(AB<int> t, lista<int> l){
  int p=-1; AB<int> aux= new AB<>();
  if(!t.EsVacio()){
    try{ if(l.EsVacia()){
          if(t.subArbolDer.EsVacio()){ p=t.Raiz(); {
    else if(l.Cabeza()==0){ p= percent(t.subArblzq(), l.Cola()){
     else{ p= percent(t.subArbolDer(), l.Cola()){
```



```
public static ABD<int> huffmanTree(lista<ABD<int>> a){
 try{while(!a.Cola().EsVacia){
   ABD<int>= new ABD(a.Cabeza().Raiz()+ a.Cola().Cabeza().Raiz(), a.Cabeza(), a.Cola().Cabe
   a= a.Cola().Cola();
   a= insert(ABD, a);
 {sol= a.Cabeza();
 {catch(...)}...{
{

public static lista codeO1(AB<int> t, lista l){
 try{if(t.subArbDer().EsVacio()){
      if(t.SubArblzq().Raiz()!=s){l.Añade(-1);
   else{l=codeO1(t.subArblzq(),l)}
      if(l.EsVacia()||l.Cabeza()!=-1){l.Añade(0);
      else{l=codeO1(t.subArbDer,s)}
        if(l.EsVacia()||l.Cabeza()!=-1){l.Añade(1);
        {
 {catch(...)}...{

                          6-(7x2)
                        -6(x72)
                        -6x72

 try{while(!exp.EsVacia()){
     if(exp.Tope() instance of Double && (!aux.EsVacia && aux.Tope instance of Double){
       x=exp.Tope();   y=aux.Tope();   op=aux.Tope();   exp.Apila(eval(x,op,y
     exp=exp.Desapila;  aux=aux.Desapila();  aux=aux.Desapila();
  else{aux.Apila(exp.Tope());
 exp=exp.Desapila;{return (Double) aux.Tope();
{catch(...)}...{
{
```

((3 + (5 - 1)) × (15 % 4)) + 1 ((3 + 4) × (3)) + 1

PN

((3 + - (5 1)) * (% (15 4)) + 1

(+3 - 5 1) * (% 15 4) + 1

*(+3 - 5 1 % 15 4) + 1

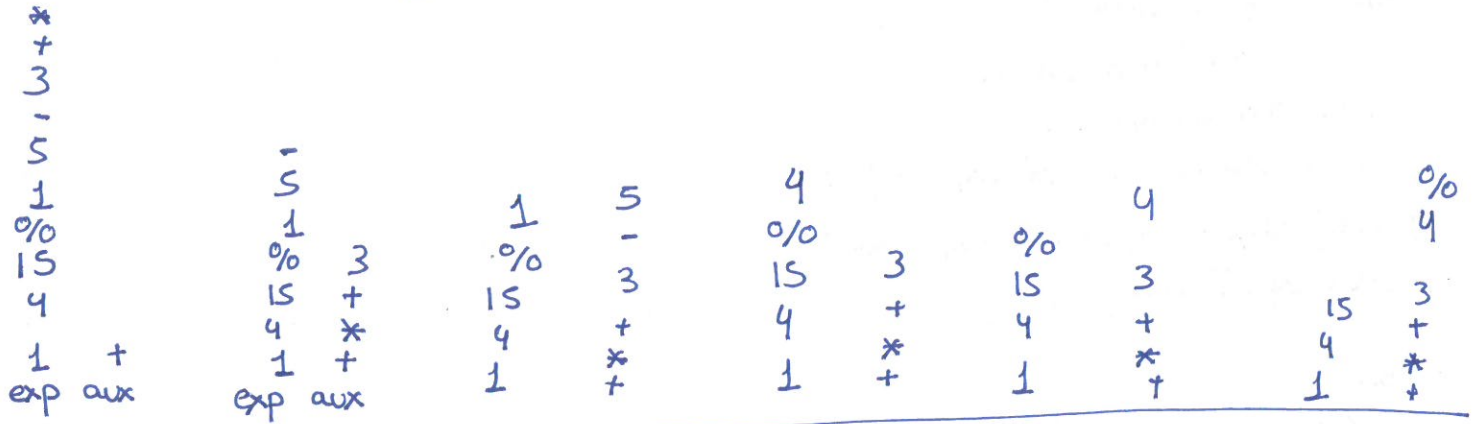
+ * + 3 - 5 1 % 15 4 1

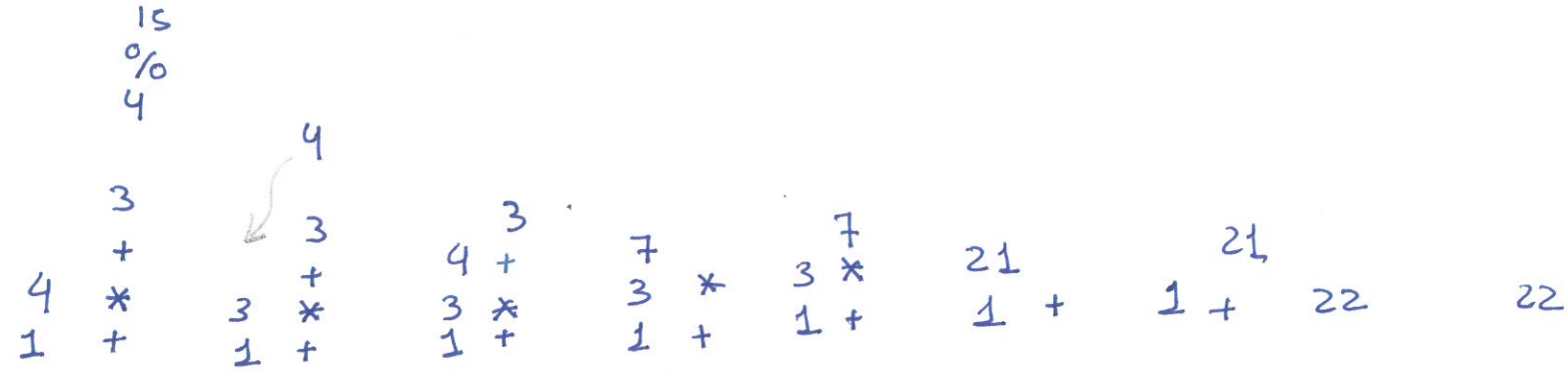
RPN

((3 + (5 1) -) * (15 4) %)) + 1

(3 5 1 - +) * (15 4 %)) 1 +

3 5 1 - + 15 4 % * 1 +





```
public static Double PNrepeat(Pila exp){
Pila aux = new PD(); Double x, y; String op;
try{ while(!exp.EsVacia()){
    if(exp.Tope() instance Of Double && (!aux.EsVacia() && (aux.Tope() intance of Double)))
        x = (Double)aux.Tope();
      aux = aux.Desapila();
       y = (Double) exp.Tope();
     exp = exp.Desapila();
      op = (String) aux.Tope();
     aux = aux.Desapila();
      exp = exp.Apila(eval(x, op, y))
   {else}      aux.Apila(exp.Tope());
    exp = exp.Desapila());
   {
  {
{catch(TADVacio Exception ex)} sout("ex");{
```

exp: 3 aux: 3

{

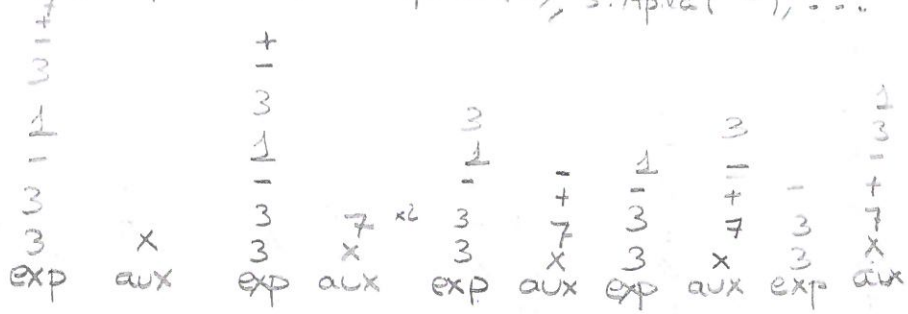
7x((3-1) + (3-3))

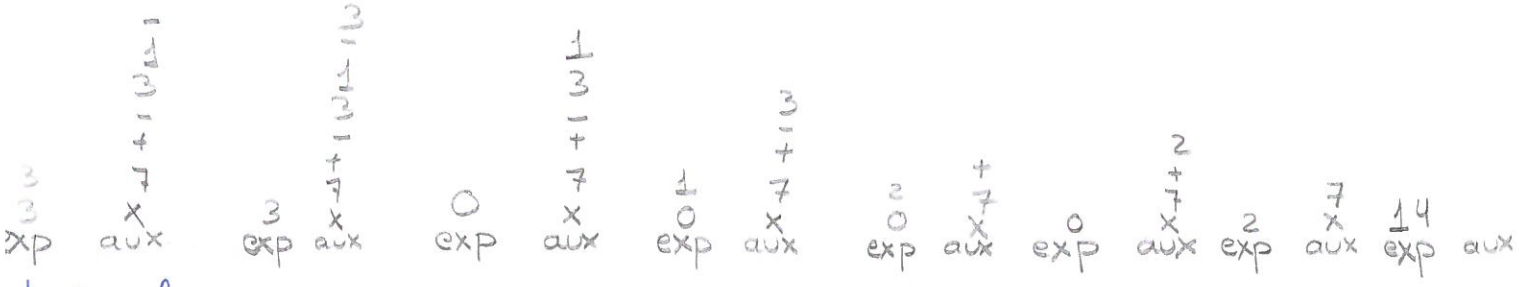
X7((-3 1) + (- 3 3))

X 7(+(- 3 1)(-3 3))

X7 + - 31 - 33

s.Apila(3); s.Apila(3); s.Apila(-); ...






```
try} while(!exp.EsVacia()){
    if(exp.TopeInstance of Double && (!aux.EsVacia() && aux.tope() instance of Double)){
       x = (Double) exp.Tope();
      exp = exp.Desapila();
      y = (Double) aux.Tope();
     aux = aux.Desapila();
     op = (String) aux.Tope();
    aux = aux.Desapila();
   exp.Apila(eval(x, op, y))
  {else} aux.Apila(exp.Tope());
    exp = exp.Desapila();
   {
{catch(...)}...{                    {return (Double) exp.tope();
```

exp: 14 aux

Huffman 2023 | codeO, recieves Huffman tree "t" and ASCII code of symbol "s"; returns lists of 0 and 1 the Huffman code of "s" if included. If not, [-1]

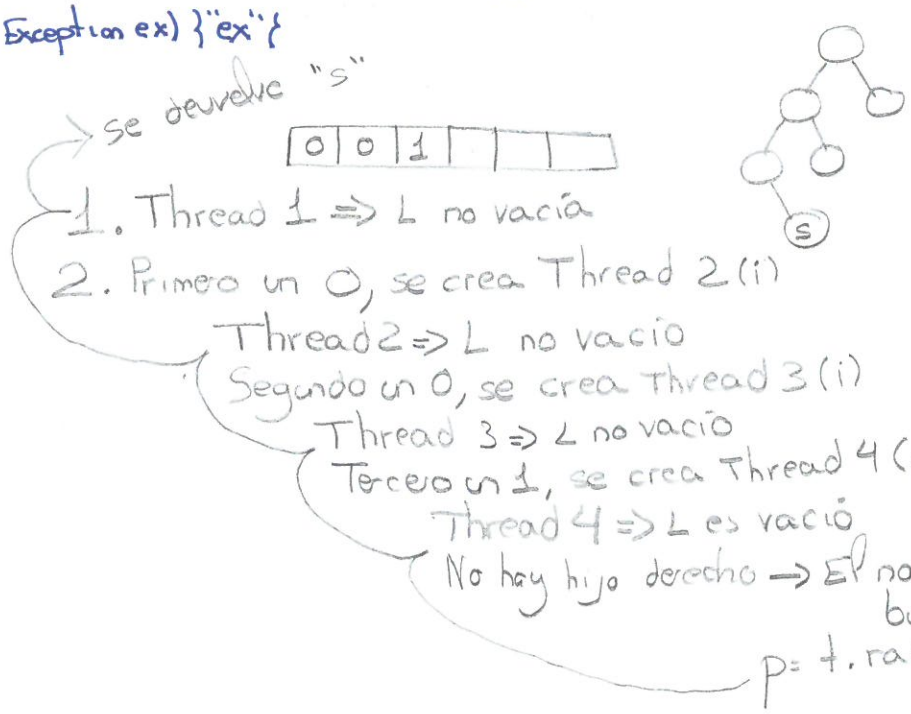
```
public static Lista<Integer> codeO1(ArbolBinario<Integer> t, Integer s){
   Lista<Integer> l, laux; l = new LD();
   ArbolBinario<Integer> aux = new ABD();
    try{ if (t.subArbolDer.EsVacio()){
            if (t.subArbolIzq.Raiz() != s){ l.Anade(-1);{
   else{ l = codeO1(t.SubArbolIzq(), s);
      if(l.EsVacia() || l.Cabeza() != -1)) l.Añade(0);
      else{ l = codeO1(t.subArbolDer(), s);
        if(l.EsVacia() || l.Cabeza != -1)){ l.Añade(1);
        {
   {catch(--)}...
  { return l;
{
```

tdv
tiR s -1

Huffman 2022 | percent. receives Huffman tree and a list of 0s and 1s. If the list corresponds to the Huffman code of a symbol in the tree, return %, else, -1

```
public static int percent(ArbolBinario<Integer> t, lista<Integer> l){
   integer p = -1; AB<Integer> aux = new AB<>();
   if(!t.EsVacio()){
"final" } try{ if(l.EsVacia()){1
          if (t.SubArbolDcho().EsVacio()){ p = t.Raiz(); {
     else if (l.cabeza() == 0){
          p = percent(t.subArbolIzq(), l.cola()); {2
      else{ p = percent(t.subArbolDer(), l.cola()); {3
   {catch(TADVacio Exception ex){"ex"{
  { return p;
  {
```

Hemos llegado al final del listado, ese era el símbolo



DATA STRUCTURE	DEGREE IN COMPUTER SCIENCE ENGINEERING
English group	Progress test on practices – December 20th 2023

SOLUTIONS

1)

```
Pila s = new PD(); s.APila(3); s.APila(5); s.APila(12); s.APila('%');
s.APila('*'); s.APila(2); s.APila('+');
```

ITE. 1		ITE. 2		ITE. 3		ITE. 4		ITE. 5		ITE. 6		ITE. 7		ITE. 8		ITE. 9		ITE. 10	
2																			
'*'		'*'							12										
'%'		'%'		'%'			'%'		'%'				2						
12		12		12	'*'	12	'*'		'*'		'*'		'*'						
5		5	2	5	2	5	2	5	2	2	2		2		2				
3	'+'	3	'+'	3	'+'	3	'+'	3	'+'	3	'+'	3	'+'	6	'+'	8			8
exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	a

2)

```
public static int percent ( ArbolBinario<Integer> t,
                            Lista<Integer> l)
{   Integer p=-1; ArbolBinario<Integer> aux=new ABD<>();
    if (!t.EsVacio())
    try{ if (l.EsVacia())
            {if (t.SubArbolDcho().EsVacio()) p=t.Raiz();}
         else if (l.Cabeza()==0)
                    p=percent(t.SubArbolIzqdo(),l.Cola());
              else   p=percent(t.SubArbolDcho(),l.Cola());
    }catch(TADVacioException e)  {System.out.println(e);}
    return p;
}
```

3)

```
public static Lista<Integer> cycle(int[][] adj) {
    int inf=Integer.MAX_VALUE, minweight=Integer.MAX_VALUE,
        n= adj.length, i=0, j=0;
    Lista<Integer> sol=new LD<Integer>();
    Lista<Integer>[][] path = Floyd(adj);
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            if (i!=j && adj[i][j]!=inf && adj[j][i]!=inf
                && (adj[i][j]+adj[j][i])<minweight)
              { minweight=adj[i][j]+adj[j][i];
                sol=(Lista<Integer>)(path[i][j].Cola().clone());
                sol.Concatena(path[j][i].Cola()); }
   return sol;
}
```

Polish Notation

2+((12%5)*3)

PN

2 + ((12 %o 5) x 3)

+2((12 %o 5) x 3)

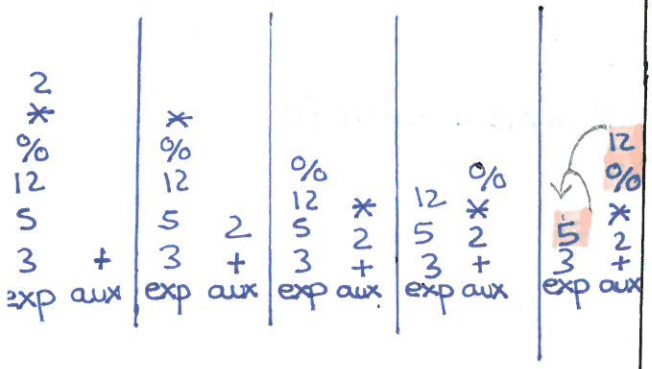
+2(x(12%o5)3)

+2(x(%o 12 5)3)

+2 x %o 12 5 3

Se apila de derecha a izquierda

s.apila(3); s.apila(5); s.apila(12); s.apila(%o); s.apila(*); s.apila(2); s.apila(+);



RPN (Reverse)

2+((12 %o 5) x 3)

2((12%o5)*3) +

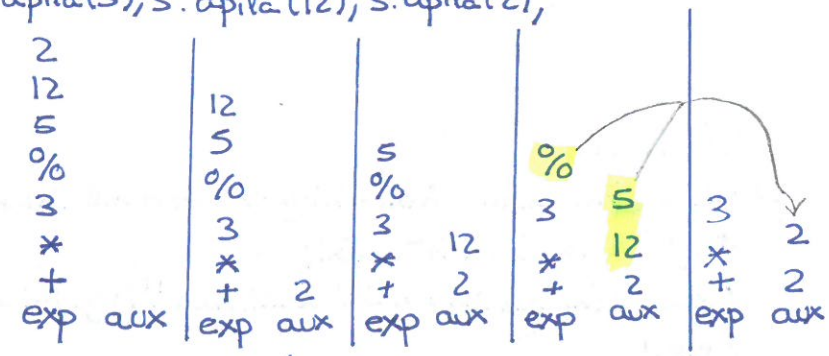
2((12%o5)3)* +

2((12 5) %o 3) * +

2 12 5 %o 3 * +

Se apila de derecha a izquierda

s.apila(+); s.apila(*); s.apila(3); s.apila(%o); s.apila(5); s.apila(12); s.apila(2);



Se va desapilando en aux, cuando en aux hay dos núm seguidas y un operando en exp, se opera y se guarda en aux

Código

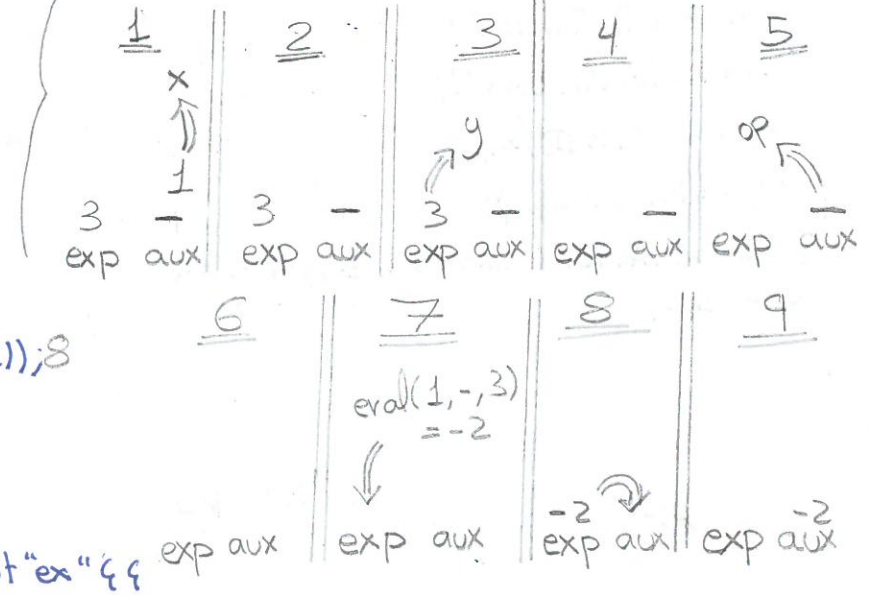
Pila exp ⇒ tiene la expresión en PN, ex: +2*34 ⇒ 2+(3x4)

Double ⇒ guarda números; String ⇒ guarda operandos

Proceso: Se recorre exp de tope a abajo, evaluando posibilidades. Aux: pila que ayuda

```
public static Double PNvalue(Pila exp){
  Pila aux = new PD(); Double x, y; String op;
  try{while(!exp.EsVacia()){
    if(exp.Tope() instanceOf Double && (!aux.EsVacia() && aux.Tope() instanceOf Double)){
      x = (Double) aux.Tope(); 1
      aux = aux.Desapila(); 2
      y = (Double) exp.Tope(); 3
      exp = exp.Desapila 4
      op = (String) aux.Tope(); 5
      aux = aux.Desapila(); 6
    exp.Apila(eval(x, op, y)); 7
    }else aux.Apila(exp.Tope()); 8
      exp = exp.Desapila(); 9
  }
  return (Double) aux.Tope()
}catch(TADVacioException e){} sout "ex" }}
```

¿Qué hace?

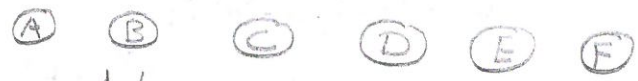


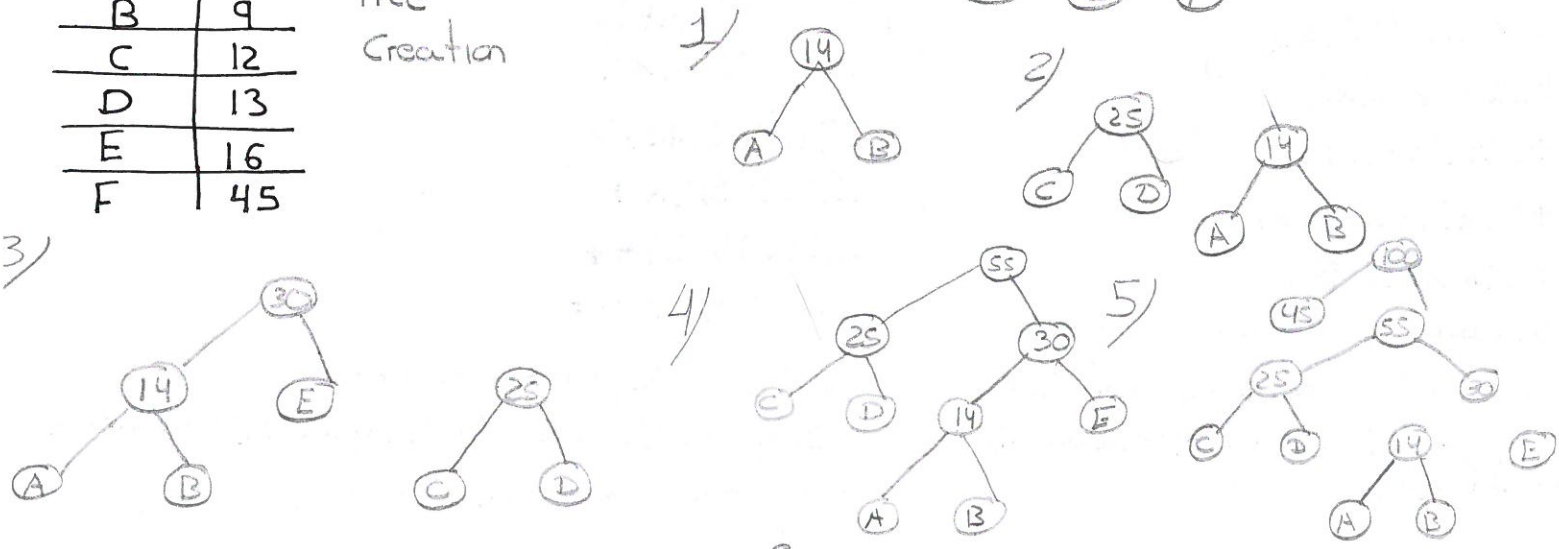
Huffman Trees

Simbolo	Freq
A	5
B	9
C	12
D	13
E	16
F	45

Tree Creation

De menor a mayor





Método insert

```
public static Lista<ABD<Integer>> insert(ABD<Integer> x, Lista<ABD<Integer>> y){
  if(y.EsVacia()){y.Añade(x);
  }else if(x.Raiz()<=y.Cabeza().Raiz()){y.Añade(x);
  }else{
    ABD<Integer> c = y.Cabeza();
     y = y.Cola();
     y = insert(x,y)
    y.Añade(c);
    return y;
  }
```

Método HuffmanTree

Lista ORDENADA creciente

```
public static <ABD<Integer>> HuffmanTree(Lista<ABD<Integer>> x){
   ABD<Integer> sol= new ABD<Integer>;
   try{while(!x.Cola().EsVacia()){} -> Mientras que al menos queden 2 árboles en la lista (pa poder unirlos)
     ABD= new ABD<integer>(x.Cabeza().Raiz()+x.Cola().Cabeza().Raiz(), (suma de Cabezas
     x.Cabeza(), -> El 1er elem. Hijo izq
     x.Cola().Cabeza); -> El 2º elem. Hijo der.
     X=x.Cola().Cola(); -> Pasa de los dos siguientes ya que fueron procesados
    x=insert(ABD,x); -> Va a insert para asegurar orden creciente
   {sol=x.Cabeza();
  {Catch(TADvacioException ex){}sout(ex){
  return sol;
 {
```

DATA STRUCTURE	DEGREE IN COMPUTER SCIENCE ENGINEERING
English group	Progress test on practices – December 20th 2023

NAME: _____

Java interfaces that can be used in this exam

```
public interface Lista<E> {
   public boolean EsVacia();
   public void Añade(E x);
   public E Cabeza() throws TADVacioException;
   public Lista<E> Cola() throws TADVacioException;
   public void Concatena(Lista<E> l);
   public Object clone();
}
public interface ArbolBinario<E> extends Arbol<E>
   public boolean EsVacio();
   public E Raiz() throws TADVacioException;
   public ArbolBinario<E> SubArbolIzqdo() throws TADVacioException;
   public ArbolBinario<E> SubArbolDcho() throws TADVacioException;
}
```

(12%5)*3 → *(12%5)3 → *%12 5 3

+ * 9 / 2 3 12 □ 5

1) [Polish notation] In this exercise we are concerned with the second practice devoted to evaluate an arithmetic expression stored into a general stack once expressed in PN (*Polish notation*). Given the arithmetic expression **2+((12%5)*3)**, complete the following Java code for initializing a stack representing the same expression in Polish Notation: +2*%12 5 3

```
Pila s = new PD();   s.APila( 3 ); s.APila( 5 ); s.APila( 12 );
     s.APila( % ); s.APila( * ); s.APila( 2 ); s.APila( + );
```

In the table below, use only the cells required to represent the contents of stacks **exp** and **aux** after finishing each iteration when calling the method explained in class with the previous stack:

ITE. 1		ITE. 2		ITE. 3		ITE. 4		ITE. 5		ITE. 6		ITE. 7		ITE. 8		ITE. 9		ITE. 10	
2																			
*		*						12											
%		%		%			%		%				2						
12		12		12	*	12	*		*		*		*						
5		5	2	5	2	5	2	5	2	2	2		2		2				
3	+	3	+	3	+	3	+	3	+	3	+	3	+	6	+	8			8
exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux

2) [Huffman trees] Complete in the next page the Java code of method **percent** which receives a Huffman tree with al least two leaves and a list only containing 0's and 1's. If such list corresponds to the Huffman code of a symbol in the tree, then the method must return the percentage of such symbol. Otherwise, the output will be -1.

3) [Floyd's algorithm] A cycle in a graph is a path starting and ending in the same node and the rest of nodes are not repeated. Complete in the next page the Java code of method **cycle** which receives the adjacency matrix of a graph and, after applying the Floyd's algorithm on it, returns a list with all the nodes (without repetitions) involved on the cycle with the lowest weight in the graph. If there are no cycles in the graph, the output will be an empty list.

```
public static int percent (ArbolBinario<Integer> t, Lista<Integer> l)
{    Integer p=-1; ArbolBinario<Integer> aux=new ABD<>();
     if (!t.EsVacio())
     try{




     }catch(TADVacioException e)  {System.out.println(e);}
     return p;
}
```

```
public static Lista<Integer> cycle(int[][] adj)
{    int   inf=Integer.MAX_VALUE, minweight=Integer.MAX_VALUE,
           n= adj.length, i=0, j=0, k=0;
     Lista<Integer> sol=new LD<Integer>();
     Lista<Integer>[][] path = Floyd(adj);




        return sol;
}
```

SOLUTIONS

1)

```
Pila s = new PD(); s.APila("+"); s.APila("*"); s.APila(3.0);
s.APila("%"); s.APila(5.0); s.APila(12.0); s.APila(2.0);
```

ITE. 1		ITE. 2		ITE. 3		ITE. 4		ITE. 5		ITE. 6		ITE. 7		ITE. 8		ITE. 9		ITE. 10	
12.0																			
5.0		5.0																	
"%"		"%"		"%"															
3.0		3.0		3.0	5.0	3.0			3.0										
"*"		"*"	12.0	"*"	12.0	"*"	2.0	"*"	2.0		6.0								
"+"	2.0	"+"	2.0	"+"	2.0	"+"	2.0	"+"	2.0	"+"	2.0		8.0						
exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux	exp	aux

2)

```
public static Lista<Integer> code01(ArbolBinario<Integer> t, Integer s)
{ Lista<Integer> l,laux; l=new LD();
  ArbolBinario<Integer> aux = new ABD<>();
  try {
    if (t.SubArbolDcho().EsVacio())
              { if (s != t.SubArbolIzqdo().Raiz()) l.Añade(-1);}
    else { l = code01(t.SubArbolIzqdo(), s);
           if ((l.EsVacia() || l.Cabeza() != -1)) l.Añade(0);
           else {  l = code01(t.SubArbolDcho(), s);
                   if ((l.EsVacia() || l.Cabeza() != -1)) l.Añade(1);
           }
       }
  }catch(TADVacioException e) { System.out.println(e); }
  return l;
}
```

3)

```
public static int floidijs(int[][] adj) {
int inf = Integer.MAX_VALUE, n = adj.length, i = 0, j =0, k =0, num = 0;
int[][] paw =(int[][]) adj.clone(); Lista<Integer>[][] pan =Floyd(paw);
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      try {
        if (i != j && adj[i][j] == inf && paw[i][j] != inf &&
            (2 * adj[i][pan[i][j].Cola().Cabeza()]) < paw[i][j]) num++;
      }catch(TADVacioException e)  {System.out.println(e);}
  return num;
}
```

if [(Hileaf i) > (Hileaf d)] = Hileaf i

if [(Hileaf i) < (Hileaf d)] = Hileaf d

dec24(AB 1(ABSAV(ABSAV AV))AV) [7] =

= dec24 i (dec24 d l) = dec24 i (dec24 AV [7]) = dec24 i [7]

= dec24(ABSAV(ABSAV AV) [7] = dec24 i (dec24 d l) =

= dec24 i (dec24(ABSAV AV) [7]) = dec24 i (8:7) = dec24 i [8:7]

= dec24 AV [8:7] = [8:7]

```
if(!t.EsVacio()){
  hq) (!t1.subarbolIzq.EsVacio()){
        i++;
```

t.nodes(t1.subarbolIzq())

```
sol--;
if(-sol%(2i+1)==0){
  sol=Math.abs(sol);}
```

sol

0 0.1

0 0 1

0 0 0 0.2

0 0 0 0 0 0 0

2i+1

Nodo[0]: new NodoCirculo(x, Rand(), false)

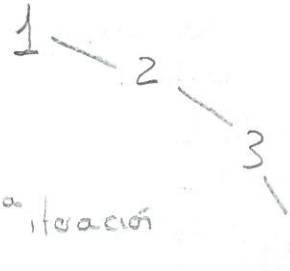
[numnodos - 1] [numnodos]

Graphs: High Level

Low only GNDE => static

```
public static <E> boolean IsConnected (Grafo <E> g){
  Lista <E>, nodes, visited, pending; nodes = g.Nodos();
  integer tovisit = nodes.Longitud(); if tovisit == 0) return true
  try{ pending.Add( nodes.Cabeza()) · 1º elem. de la lista
    while (!pending.EsVacia() && tovisit > 0)}
    E x = pending.Cabeza() pendingg = pending.Cola();
    if (!visited.EstaContenida(x))} visited.Add(x); tovisit--; pending.Concatena(g.Adyacentes(x)
```

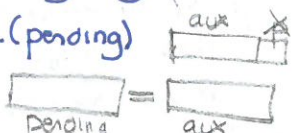
Listas



Solo para la 1ª iteración

x.visitado = true

Lista <E> aux = g.Adyacente(x)
aux.Concatena(pending)
pending = aux



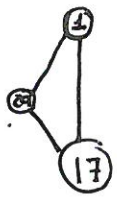

```
{catch(...)}
  return tovisit == 0;
```

GNDE => Solo este tiene el visitado Atrib

High ++ Low Level

```
public static <E> boolean IsConected( GNDE <E> g)}
  Lista <E> Nodes,        pending; nodes[] = g.Nodos;
  int tovisit = nodes.Longitud
    if(x.visitado
  try } pending.Add(nodes.Cabeza());
    while(!pending.Esvaca() && tovisit > 0)
  E x = pending.Cabeza(); pending = pending.Cola();
    if( !visited.EstaContenida(x))} visited.Add(x); - toVisit--; pending.Concatena(g.Ady(x))
```

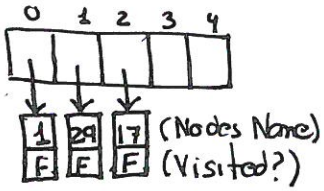
Graphs



Limited size:

MAX_nodes = 5

numNodes = 3



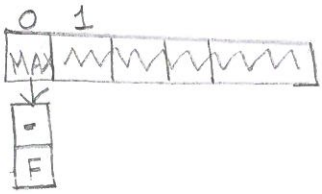
	0	1	2	3	4
0		T	T		
1	T		T		
2	T	T			
3					
4					

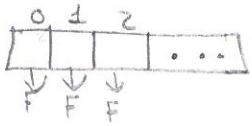
En BST el mínimo está en

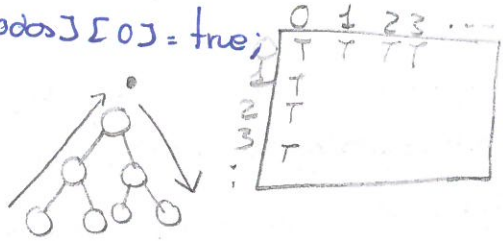
b) Java Iterative inside GNDE class

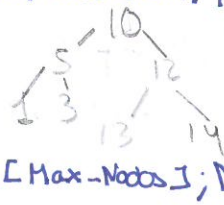
Mixheap → Mayor / Menor

```
public GNDE(ArbolBinario<Integer> t){
  numnodos = 0; Adyacencias = new boolean[Max_Nodos][Max_Nodos];
  Nodos = new NodoGrafo[Max_Nodos]; Integer x; ArbolBinario<Integer> y, z;
try{ if(!t.EsVacio()){
      Nodos[0] = NodoGrafo(t.Raiz(), false);
      t = mixheap(t.subArbIzq(), t.subArbDer());
      numnodos++;
   }
   while(!t.EsVacio()){
     Nodos[numnodos] = NodoGrafo(t.Raiz(), false);
     t = mixheap(t.subArbIzq(), t.subArbDer());
     Adyacencia[0][numnodos] = Adyacencia[numnodos][0] = true;
     numNodos++
   }
 }{ catch(catch TADVacioException ex) sout(ex); }{
```

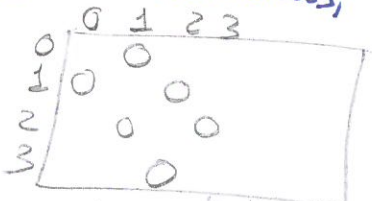









```
public GNDE(ArbolBinario<E> t){
  Adyacencia = New Boolean[Max_Nodos][Max_Nodos]; Nodos = new NodoGrafo[Max_Nodos];
  numnodos = 0; ArbolBinario<E> aux;
  if(!t.EsVacio())
    try{ Nodos[0] new NodoGrafo(t.Raiz(), false);
         nodografo++; aux = t.subArbIzq();
         while(!aux.EsVacio()){
           Nodos[numnodos] = new NodoGrafo(aux.Raiz(), false);
           Adyacencia[numnodos-1][numnodos]. Ady[numnodos][numnodos-1] = true;
           numNodo++; aux = t.subGrafo();
    if(!t.subArbIzq().EsVacio()) }
```



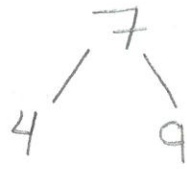
→ Está corriendo el con el anterior

Unit 4. Trees

Árbol: Colección de nodos conectados jerárquicamente
- **Raíz:** Nodo principal del árbol
- **Hijo:** Nodo conectado a otro previo
- **Hojas:** Nodos sin hijos
- **Subárbol:** Cualquier nodo + sus descendientes

Nivel: Profundidad de un nodo, raíz en el nivel 1 (o 0)
Altura: Nivel más profundo del árbol
Recorrido: Proceso de visitar todos los nodos
- **Preorden:** Raíz, subárbol izq, subárbol der.
- **Inorden:** Subárbol izq, raíz, subárbol der.
- **Postorden:** Subárbol izq, subárbol der, raíz.

```
empty :: Tree a -> Bool
root :: Tree a -> a
leftsubtree // rightsubtree :: Tree a -> Tree a
```




```
mytree = BT 7 (BT 4 EBT EBT) (BT 9 EBT EBT)
```

↳
```
mytree = BT 7
           (BT 4
              EBT
              EBT)
           (BT 9
              EBT
              EBT)
```

Vacío ↩

```
data Tree a = EBT | BT a (Tree a) (Tree a)
```
Nodo con raíz a subárbol izq y der.

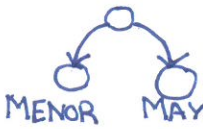
1-Cont. long Está conectado?

Meto nodos, `nodes = g.Nodes();`
Tomo cabeza, `pareja = nodes.Cabeza();`
Miro si reconoce

Implementar recorridos

```
inorder :: Tree a -> [a]
inorder EBT = []
inorder (BT r i d) = inorder i ++ [r] ++ inorder d

preorder :: Tree a -> [a]
preorder EBT = []
preorder (BT r i d) [r] ++ preorder i ++ preorder d

postorder :: Tree a -> [a]
postorder EBT = []
postorder (BT r i d) = postorder i ++ postorder d ++ [r]
```

Árboles de búsqueda (Binary search trees)

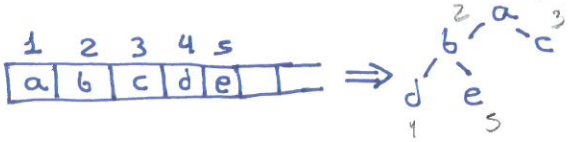


→ Queremos introducir X
- → if X < r = izq
- → if X > r = der.

```
insert x EBT = BT x EBT EBT
insert x (BT r i d)
  | x < r   BT r (insert x i) d
  | x > r   BT r i (insert d)
```

```
Search x EBT = False
Search x (BT r i d)
  | x == r => True
  | x < r = BT r (search i) d
  | x > r = BT r i (search d)
```

```
height EBT = 0
height (BT _ i d) = 1 + max (height i) (height d
```



2i ← 0 → 2i+1

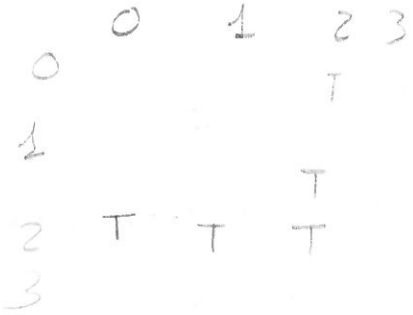
2i+1 ← 1 → 2i+2

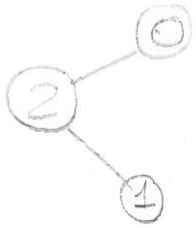
AVL HeightAVL

```
HeightAVL :: Tree a -> int
HeightAL EBT = 0
HeightAVL (BT _ i d) = left = HeightAVL i ; right = HeightAVL d;
 if left == -1 || right == -1 || abs(left - right) > 0
    then -1
 else 1 + max(left) (right
```

El árbol es t

```
InvertadoBT :: Tree a -> [a]
InvertadoBT EBT = 0  // No hay na
InvertadoBT (BT _ i d) = left = InvBT i ; right = Inv BT d;
  if left - right != 0 = 0  // No es completo
  else  r:(BT + i) : (BT + d)
```



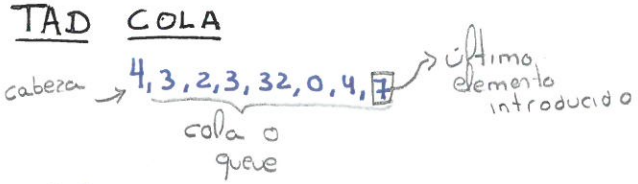


Practica de Floyd

Implement `public static Lista<Integer>[][] Floyd(int[][] adj)`

Shortest path in a directed graph

TAD COLA



Sigue la idea de FIFO, first In, first Out
La cabeza fue el primero metido y el primero en salir

data Queue a = EQ | Put (Queue a) a
(Free generaters)

Put == :.

EQ : vacía cola
empty queue

EQ :.2 :.7 :.5 :.7

≈ Operaciones ≈

- EmptyQ :: Queue a → Bool
  EmptyQ EQ = true

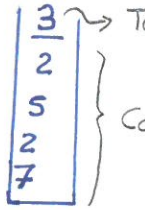
- first :: Queue a → a
  ① first (EQ :. x) = x
  ② first (c :. x) = first c

EXAMPLE: first (EQ :.3 :.5 :.2 :.7) → ②→2 / first (EQ :.3 :.5 :.2 ... ; first (EQ :.3) → ①→3 (②→5)

- rest :: Queue a → Queue a
  ① rest (EQ :. x) = EQ
  ② rest (c :. x) = (rest c) :. x

EXAMPLE: rest (EQ :.3 :.5 :.2 :.7) (①) → EQ :.5 :.2 :.7

TAD STACK (O pila)



Para poder acceder a los elementos de abajo se debe desapilar

ES : PV : pila vacía

data Pila a : PV | a :/ (Pila a)

:/ apila o push
PV pila vacía

EJEMPLO: 3 :/ 2 :/ 5 :/ 2 :/ 7 :/ ES

- top :: Stack a → a Like head
- pop :: Stack a → Stack a Like tail

⚠ Típico error ⚠
elimina :: a → Stack a → Stack a
elimina x ES = ES
elimina x (y :/ p) if x == y then p else [y :/ (elimina x p)] → Elimina de abajo sin desapilar

Podemos añadir
"apilar :: a → Stack a → Stack a
apilar x p = x :/ "

Quedando ⟹

elimina :: a → Stack a → Stack a
elimina x ES = ES
elimina x (y :/ p) if x == y then p else apila y (elimina x p)

Tema 3.
Listas. Pilas. Colas

Tipo abstracto de datos (TADs):

→ Lineal: vacío o al menos un elemento siguiendo un orden
↳ Arrays, listas, pilas, colas

→ No lineal: conjuntos, árboles, grafos

TAD LISTA

4, 3, 2, 3, 32, 0, 4, 7 → La cabeza es el último elemento introducido
Cabeza ↳ Cola
Sigue una estructura LIFO (Last IN First Out)

Todas son de un mismo tipo pero no de tipo fijo → Genérica
Si se mezclan tipos → general

Definiciones Haskell

data Lista a = Nil | cons a (Lista a)
Nil: lista vacía
Cons: operaciones generadoras libres

Lista a = [a]
Nil = []
Cons " = (infijo)

Operaciones no generadoras

Cons 1 (Cons 2 Nil) = 1 : (2 : [])
['a', 'b', 'c'] = "abc"

- empty :: [a] → Bool — empty [a] → false — empty _ = False
- head :: [a] → a — head [] → error "Empty list" — head (h : _) = h
- tail :: [a] → a — tail [] → error "Empty list" — tail (_ : t) = t

- lenght :: [a] → int — lenght [] = 0 — lenght (h : t) = 1 + lenght of t
- concatenate :: [a] → [a] → [a] — concatenate [] s = s — concatenate (h : t) s = (h : concatenate t s)
  ↳ also seen as ++ [a]
  "Links two lists"
- reverse :: [a] → [a]
  ↳ "Same elements but in inverted order"
  ↳ Example: reverse [1,2,3] } h = 1, t = [2,3]
  reverse [] = []
  reverse (h : t) concatenate (reverse t) [h]
  reverse [3] is [] ++ [3] = [3]; reverse [2,3] is [3] ++ [2] = [3,2];
  reverse [1,2,3] is [3,2] ++ [1] = [3,2,1]
- last :: [a] → a
  "Returns last element of a list" — last [x] = x — ([x] == to x : [], list with 1 element) — last (_ : t) = last t
- elem :: (Eq a) ⇒ a → [a] → Bool
  "If element in list" "a belongs to Eq class to compare"
- extract :: (Eq a) ⇒ a → [a] → [a] — Extract x [] = [] — extract x (h : t) = if (x == h) then extract t, else h : extract x t
- take :: Int → [a] → [a] — take 0 [] = [] — take n (h : t) = h : take (n - 1) t → ya que h se "salva"
- cut :: Int → [a] → [a]
  ↳ "Contrario a take" — cut 0 l = l — cut n (h : t) = cut (n - 1) t → h se fue
- substitute :: (Eq a) ⇒ a → a → [a] → [a] — substitute x y (h : t) if (h == x) then y substitute x y t / else h : substitute x y t
  ↳ "Changes all x in l by y"
- sum :: [int] → Int — sum [] = 0 — sum (h : t) = h + sum t
- greater :: (Ord a) ⇒ [a] → a — greater [x] = x — greater (x : y : t) = if x > y then greater (x : t) else greater (y : t)

23-24

1st Progress test

```
op23 :: (ord a) => [a] -> [a]   Doy una lista ordenada y recibo una lista
op23 (x:y:s) = if (x>y) then s++[x] else x:(op23 (y:s))
op23 1 = 1
```

a) op23 [1,2,3] → [1,2,3]

b) op23 [3,2,1] → [1,3]

(3>2) → 1++[3] ⇒ [1,3]

op23 1 = 1 → 1

c) op23 [1,3,2] → [1,3]

(3>2) → []++[3] = [3]

try{... { catch (TADVacioException ex)}ex.printStackTrace();{

In Lista<E>:

```
public boolean EsVacia();
public void Añade(E e);
public E Cabeza() throws TADVacioException;
public Lista<E> Cola() throws TADVacioException;
```

In LD<E>:

```
private int longitud;
private Nodo<E> NodoCabeza
```

In Nodo<E>:

```
public Nodo(E e, Nodo<E> n)}Info = e; Siguiente = n;{
```

a) public LD(Lista<E> L1, LD<E> L2)} ⇒ two lists, mix them but firstly we need to reverse L1

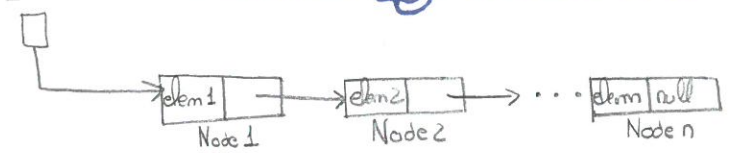
```
Nodo<E> aux = null
public LD(Lista<E> L1, LD<E> L2)}
longitud = L2.longitud; NodoCabeza = L2.NodoCabeza;
try}while (!L1.esVacia())}
    NodoCabeza = new Nodo(L1.Cabeza(), NodoCabeza);
    L1 = L1.Cola(); longitud++;
(
{ catch (TADVacioException ex)} ex.printStackTrace();{
```

b) public static <E extends Comparable> Lista<E> op23 (Lista<E> L)

```
}E x,y ;LD<E> aux1, aux2; aux1 = new LD<E>(); aux2 = new LD<E>();
if (L.EsVacia() || (l.Cola().EsVacia())) return L;
try}x = l.Cabeza(); l = l.Cola(); y = l.Cabeza();
while (!l.Cola().EsVacia() && x.CompareTo(y) <= 0)}
    aux1.añade(x); x = y; y = l.Cabeza(); l = l.cola();{
if (x.compareTo(l.Cabeza()) > 0) }aux2.añade(x); l = l.Cola();{
else aux1.Añade(x);
while (!l.EsVacia())}
    aux1.Añade(l.Cabeza()); l = l.cola();{
{ catch (TADVacioException e)} return new LD<E>();{
return new LD<E>(aux1, aux2);{
```

Implementación de listas

interface List<E> → clase de los elementos de la lista

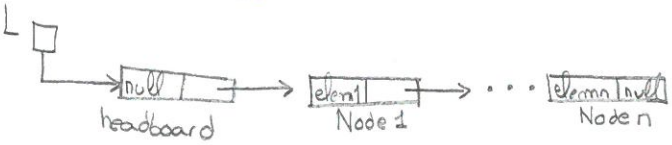


Nodes from Node<E> class

L instance of List<E> interface

elemns to be stored

→ With headboard



Para evitar asimetría con el primer nodo, ya que sería el único sin predecesor y obligaría a modificar algunos

1) insert::

Void Insert(E x, E y) throws TADVacioException. Primero y, luego x

insert :: a → a → Stack a → Stack a

(Eq a)

insert y x EmptyS = ES

insert y x = if x == a then Node y (Node a rest) else Node a (insert x y)

TEORIA

80%

TEMA 3
Listas, Pilas, Colas

1
0
2

• TEMA 4
Árboles

1
0
2

TEMA 5
Grafos.

20% HASKELL

50% JAVA.

PRACTICAS

20%

JAVA decto ororo...

Gráficos.

1 o 2 ejercicios o cuestiones,

20% JAVA.

Not. Polaca — PILAS
Huffman — ARBOLES
Floyd. — GRAFOS

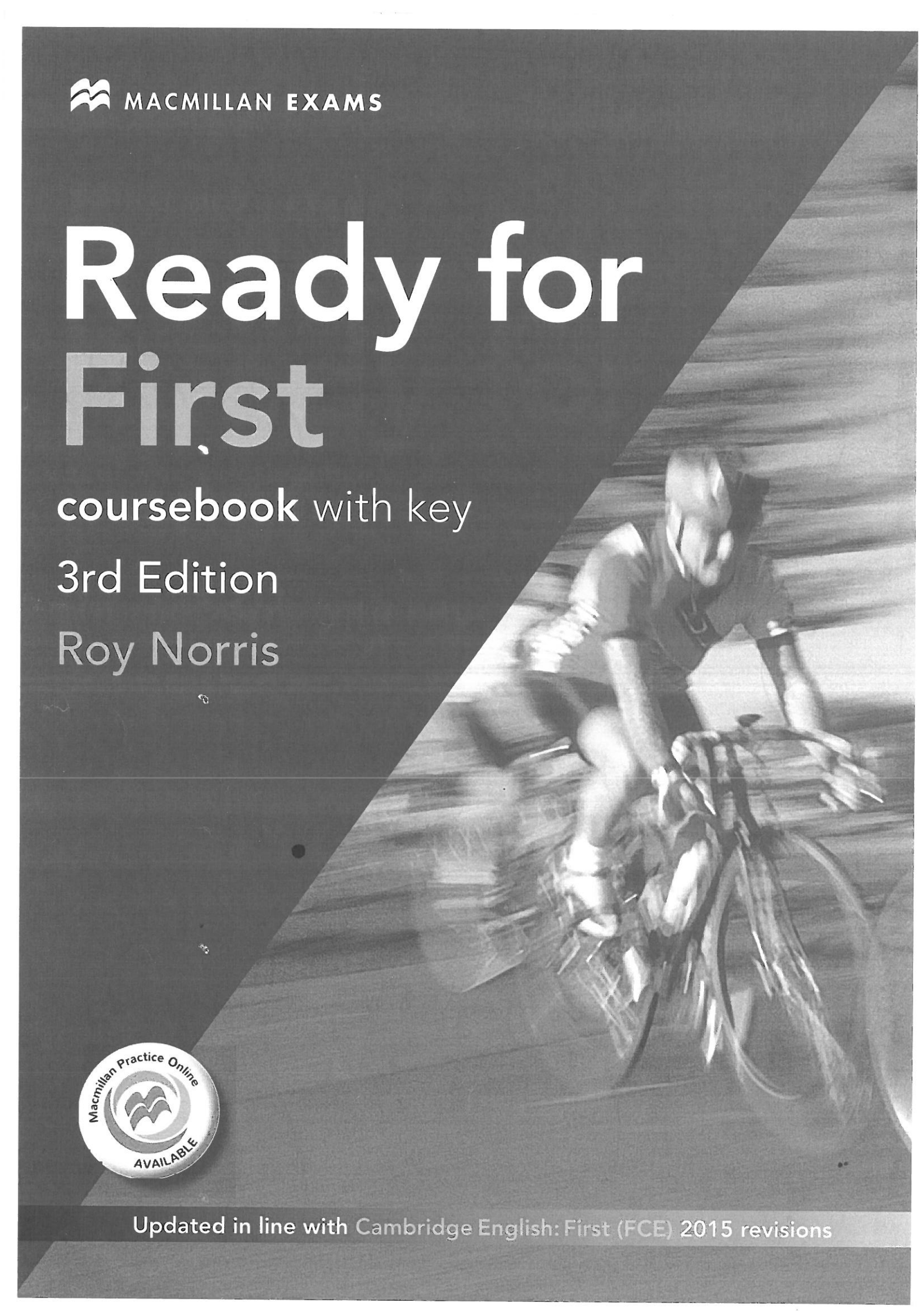
MACMILLAN EXAMS

Ready for First

coursebook with key

3rd Edition

Roy Norris



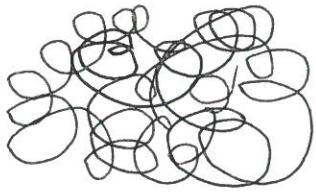
Macmillan Practice Online AVAILABLE

Updated in line with Cambridge English: First (FCE) 2015 revisions

```
public void Inserta(E x, E y) throws TADVacioException {
 Nodo<E> aux;
boolean encontrado: false;
if (EsVacia()) throw new TADVacioException

if (Nodo Cabeza.Info.equals(x)){
    Nodo Cabeza = new Nodo<E>(y, NodoCabeza);
     longitud++;
{
else}
    aux: NodoCabeza;
     while ((aux.Siguiente != null) && !encontrado)
        if (aux.Siguiente.Info.equals(x)){
            aux.Siguiente = new Nodo(y, aux.Siguiente);
            longitud++;
            encontrado:: true;
          { else} aux = aux.Siguiente;
          {
 {
```

Estr. lineales Java

↳ árboles al final



/

Arboles → Nada de genericos, solo binarios y sus variaciones

Grafos Haskell → NO

Numeros no mucho en los exámenes

Bajo nivel no llamo otras funciones

Alto nivel no punteros

Proyectos Java Temas 3, 4, 5
Practicos

Parcial 1 → 3
Parcial 2 → 4,5 1a 22-23 NO, que lleva grafos

13 Animal magic

Vocabulary 1: The Arts

1 Both words in each of the pairs below can be used in combination with one of the words in the box. Write an appropriate word from the box in each of the spaces. There is an example at the beginning (**0**).

novel	opera	~~concert~~	painting	stone	classical	gallery

0 open-air *concert* / jazz

1 _____ / ballet music

2 _____ / singer house

3 _____ / sculpture statue

4 portrait _____ / art

5 abstract _____ / priceless

6 detective _____ / historical

2 Which people do you associate with each of the following areas of the arts?

theatre	music	literature	art	opera	ballet	sculpture

Example:
Theatre: *actor, actress, director, cast, playwright, audience*

Check your answers in the Wordlist on page 208.

Reading and Use of English Part 6

Gapped text

1 Look at the work of art in the photograph and discuss the following questions.

What type of person might want to own such a work of art?
Where might they put it on display?