Taller de programación

Preparándome para el final

- MODULO IMPERATIVO (PASCAL)
  - Códigos
    - Número aleatorio (Random)

```
procedure numRandom (var num: integer);
begin
   Randomize;
   num:= random(100); {valores en el intervalo 0 a 99}
   writeln('el numero aleatorio generado es: ', num);
end;
```

    - Vectores
      - Algoritmos de ordenación

```
procedure seleccion (var v: tVector; dl: integer);
var
```

```
    i,j,pos: integer;
    item: tipoElemento;
begin
    for i:= 1 to (dl-1) do begin {busca el minimo y guarda en pos
        pos:=i;
        for j:= i+1 to dl do
            if ( v[j] < v[pos] ) then
                pos:= j;
        item:=v[pos];
        v[pos]:=v[i];
        v[i]:=item; {intercambia v[i] y v[pos]}
    end;
end;
```

```
procedure insercion (var v: tVector; dl: integer);
var
    i, j: integer;
    actual: tipoElem;
begin
    for i:= 2 to dl do begin
        actual:= v[i];
        j:=i-1;
        while (j > 0) and (v[j] > actual) do begin
            v[j+1]:= v[j];
            j:= j-1;
        end;
        v[j+1]:= actual;
end;
```

▼ Eliminar entre dos valores

```
procedure eliminarEntre (var v: vector; var dl: rangoVector;
                                        cod1, cod2: rangoCod);
var
    dist, cont: integer;
begin
    dist:= 0;
    cont:= 1;
```

```
    while (v[cont].codigo < cod1) do
    begin
        cont:= cont+1;
    end;
    while (v[cont].codigo < cod2) do
    begin
        dist:= dist+1;
        cont:= cont+1;
    end;
    while (cont<dl) do
    begin
        v[(cont-dist)]:=v[cont];
        cont:= cont+1;
    end;
end;
```

▼ Dicotómica

```
function dicotomica (v:vector; dl: integer; codigo: integer) : inte
var
    pri,ult,medio: integer;
begin
    pri:= 1;
    ult:= dl;
    medio:= (pri+ult) div 2;

    while (pri<=ult) and (codigo <> v[medio].codigoID) do begin
        if (codigo < v[medio].codigoID) then
            ult:= ult-1
        else
            pri:= pri + 1;
        medio:= (pri+ult) div 2;
    end;
    if (pri <= ult) and (codigo = v[medio].codigoID) then
        dicotomica:= medio
    else
        dicotomica:= 0
end;
```

▼ Listas

```
Procedure agregarAdelante (var l: lista; r: registro);
var
   nuevo: lista;
begin
   new(nuevo);
   nuevo^.dato:= r;
   nuevo^.sig:=l;
   l:=nuevo;
end;
```

```
Procedure CargarLista (var l:lista);
var
   r: registro;
   ult: lista;
begin
   leerRegistro(r)
   while(r.numero <> -1) do begin
      agregarAtras(l,ult,r)
      leerRegistro(r)
   end;
end;
Procedure agregarAtras(var l: lista; var ult: lista; r: registro);
var
   nuevo: lista;
begin
   new(nuevo);
   nuevo^.dato:= r;
   nuevo^.sig:=nil;
   if(l=nil)then
      l:=nuevo;
   else
      ult^.sig:= nuevo^.sig;
   ult:= nuevo;
end;
```

```
Procedure insertarOrdenado(var l: lista; r: registro);
var
   nuevo, ant, act: lista;
begin
   new(nuevo);
   nuevo^.dato:= r;
   act:= l;
   while (act <> nil) and (act^.dato.codigo < r.codigo) do begin
      ant:= act;
      act:= act^.sig;
   end;
   if (act = l) then
      l:= nuevo
   else
      ant^.sig:= nuevo;
   nuevo^.sig:=act;
end;
```

▼ ABB

▼ Declaración

```
type
   persona = record
      nombre: String;
      dni: integer;
   end;
   arbol = ^ nodo;
   nodo = record
      dato: persona;
      hi: arbol;
      hd: arbol;
   end;
```

▼ Creación

```
procedure crearArbol (var a: arbol)
var
```

```
    num: integer;
begin
    a:= nil;
    readln(num);
    while (num <> -1) do begin
        agregar (a, num)
        readln (num)
end;
procedure agregar (var a: arbol; num: integer);
begin
    if (a = nil) then
    begin
        new(a);
        a^.dato:= num;
        a^.hi:= nil;
        a^.hd:= nil;
    end
    else
    begin
        if (num <= a^.dato) then
            agregar(a^.hi,num)
        else
            agregar(a^.hd,num)
    end;
end;
```

▼ Recorrido

```
procedure enOrden (a: arbol);
begin
    if (a <> nil) then
    begin
        enOrden(a^.hi);
        write(a^.dato); // o cualquier otra acción
        enOrden(a^.hd);
    end;
end;
```

```
procedure preOrden (a: arbol);
begin
   if (a <> nil) then
   begin
      write(a^.dato); // o cualquier otra acción
      enOrden(a^.hi);
      enOrden(a^.hd);
   end;
end;

procedure postOrden (a: arbol);
begin
   if (a <> nil) then
   begin
      enOrden(a^.hi);
      enOrden(a^.hd);
      write(a^.dato); // o cualquier otra acción
   end;
end;
```

▼ Mínimos y máximos

```
function minimo (a:arbol): integer;
begin
   if(a^.hi = nil) then
      minimo:= a^.dato;
   else
      minimo:= minimo(a^.hi);
end;

function maximo (a:arbol): integer;
begin
   if(a^.hd = nil) then
      maximo:= a^.dato;
   else
      maximo:= maximo(a^.hd);
end;
```

```
function minimoNodo (a:arbol): integer;
begin
   if(a = nil) then
      minimoNodo:= nil
   else
      if(a^.hi = nil) then
         minimoNodo:= a;
      else
         minimoNodo:= minimoNodo(a^.hi);
end;

function maximoNodo (a:arbol): integer;
begin
   if(a = nil) then
      maximoNodo:= nil
   else
      if(a^.hd = nil) then
         maximoNodo:= a;
      else
         maximoNodo:= maximoNodo(a^.hd);
end;
```

▼ Búsqueda

```
function buscar(a:arbol; x: integer): boolean;
begin
   if(a=nil) then
      buscar:= false
   else begin
      if (a^.dato = x) then
         buscar:= true
      else begin
         if (x > a^.dato) then
            buscar:= buscar(a^.hd, x)
         else
            buscar:= buscar(a^.hi, x)
```

```
      end;
   end;
end;

function buscarNodo(a:arbol, x: integer): arbol;
begin
   if (a=nil) then
      buscarNodo:= nil;
   else begin
      if (a^.dato = x) then
         buscarNodo:= a;
      else begin
         if (x > a^.dato) then
            buscarNodo:= buscarNodo(a^.hd,x)
         else
            buscarNodo:= buscarNodo(a^.hi,x)
      end;
   end;
end;
```

▼ Entre dos valores

```
procedure entreDosValores(a: arbol; cod1: integer; cod2: integ
                                    valor: integer; var cant: integer
begin
   if (a<>nil) then begin
      if (cod1 >= a^.dato.codigo) then
         entreDosValores(a^.hd,cod1,cod2,valor,cant)
      else begin
         if (cod2 >= a^.dato.codigo) then begin
            entreDosValores(a^.hi,cod1,cod2,valor,cant);
            cant:= cant+1; // o cualquier otra acción
            entreDosValores(a^.hd,cod1,cod2,valor,cant);
         end;
      end;
   end;
end;
```

▼ Importante para no desaprobar

Nunca recorrer una estructura por completo si no es necesario.

▼ MODULO OBJETOS (JAVA)

▼ BASES DE LA POO

▼ Conceptos

▼ Objeto

▼ ¿Qué es?

Abstracción de un objeto del mundo real, definiendo qué lo caracteriza
(estado interno) y qué acciones sabe realizar (comportamiento).

▼ Estado interno

Compuesto por datos/atributos que caracterizan al objeto y relaciones con otros objetos con los cuales colabora. Se implementan a través de variables de instancia.

▼ Comportamiento

Acciones o servicios a los que sabe responder el objeto. Se implementan a través de métodos de instancia que operan sobre el estado interno. Los servicios que ofrece al exterior constituyen la interfaz.

▼ Envio de mensaje

Envío de Mensaje: provoca la ejecución del método indicado por el nombre del mensaje.
• Puede llevar datos (parámetros del método)
• Puede devolver un dato (resultado del método)

▼ Clase

Un objeto se crea a partir de una clase (el objeto es instancia de una clase). Cuando creo una clase se dice que la estoy instanciando.

▼ Herencia

Es un mecanismo que permite que una clase herede características y comportamiento (atributos y métodos) de otra

clase (superclase). A su vez, la clase hija define sus propias caracteristicas y comportamiento.

Ventaja: Reutilización del codigo.

▼ Clases y métodos abstractos

Una clase abstracta es una clase que no puede ser instanciada (no se pueden crear objetos de esta clase). Define características y comportamiento común para otras clases (sus subclases). Puede definir métodos abstractos que DEBEN ser implementados en sus subclases.

▼ Super

▼ ¿Qué es?

Es una palabra clave de java, que sirve para enviarse un mensaje a uno mismo a partir de la clase que está arriba.

▼ ¿Cuándo lo utilizo?

Cuando un método está definido en la misma clase con el mismo nombre, y quiero dejar en claro que quiero usar el método de la superclase. Es incorrecto usar el super si no tengo un metodo con el mismo nombre.

Se utiliza comunmente para invocar al constructor de la superclase (debe ir en la primera linea) y para el toString.

```
Constructor:
public abstract class Figura{
   private String colorRelleno, colorLinea;

   public Figura(String unCR, String unCL){
      setColorRelleno(unCR);
      setColorLinea(unCL);
   }
 public String toString(){
      String aux = "Área: " + this.CalcularArea() + CR:" + get
      "CL:" + getColorLinea();
   return aux;     }
```

```
    public abstract double calcularArea();

    }

public class Cuadrado extends Figura{
    private double lado;

    public Cuadrado(double unLado, String unColorR, String
        super(unColorR,unColorL);  // INVOCA AL CONSTRUC
        setLado(unLado);
}
// Metodos CalcularArea y CalcularPerímetro
    public String toString(){
        String aux = super.toString() // INVOCA AL toString DE
        +  "Lado:" + getLado();
    return aux;
}
}
```

▼ Encapsulamiento

Permite construir componentes autónomos de software, independientes de los demas componentes.

La independencia se logra ocultando detalles internos (implementación) de cada componente.

Entonces, una vez encapsulado del componente se puede ver sólo su interfaz.

▼ Polimorfismo

Objetos de clases distintas pueden responder a mensajes sintácticamente idénticos. Esto permite crear codigo altamente reusable.

▼ Binding dinámico

Mecanismo por el cuál se determina en tiempo de ejecución el método a ejecutar para responder un mensaje.

▼ Importante para no desaprobar

1. nunca recorrer un vector entero. siempre hasta la dimensión lógica.

2. nunca recorrer una matriz entera. Para esto puede ser necesario crear un vector de dimensiones lógicas.
3. si creo una matriz de objetos, aclarar que java inicializa en null con un comentario //.
4. no instanciar objetos de más.

▼ MODULO CONCURRENTE (R-INFO)

1. Leer bien el enunciado. Hay tiempo de sobra.
2. Declarar bien las áreas.
3. Bloquear una esquina de conflicto, justo antes de ir (sin acciones / código en el medio).
4. Liberar la esquina SIEMPRE, apenas vuelva de ella.
5. USAR LA CONCURRENCIA. Por ejemplo, si el robot1 tiene que ir a depositar 1000 papeles a la esquina (10,10) y el robot2 tiene que depositar solo 1 papel en la esquina (10,10), el robot1 debe depositar DE A UNO sus papeles, bloqueando (antes de ir) y liberando (después de volver a su esquina segura) la esquina así el robot2 no espera a que deposite los 1000 papeles.

   ▼ Solución correcta

```
robot tipo1
comenzar
    mientras (HayPapelEnLaBolsa)
        BloquearEsquina(10,10)
        Pos(10,10)
        depositarPapel
        Pos(1,1) // esquina segura
        LiberarEsquina(10,10)
fin
robot tipo2
comenzar
    BloquearEsquina(10,10)
```

```
        Pos(10,10)
        depositarPapel
        Pos(2,1)
        LiberarEsquina(10,10)
    fin
```

▼ Solución incorrecta

```
    robot tipo1
    comenzar
        BloquearEsquina(10,10)
        Pos(10,10)
        mientras(HayPapelEnLaBolsa)
            depositarPapel
        Pos(1,1)
        LiberarEsquina(10,10)
    fin
    robot tipo2
    comenzar
        BloquearEsquina(10,10)
        Pos(10,10)
        depositarPapel
        Pos(2,1)
        LiberarEsquina(10,10)
    fin
```