## Resumen Taller de Programación

# Modulo imperativo

Un arreglo es una estructura de datos compuesta que permite acceder a cada componente por una variable índice, que da la posición de la componente dentro de la estructura de datos. La estructura arreglo se almacena en posiciones contiguas de memoria.

Características: homogénea, estática, acceso directo indexado y lineal.

Una lista es una estructura de datos lineal compuesta por nodos. Cada nodo de la lista posee el dato que almacena la lista y la dirección del siguiente nodo. Toda lista puede recorrerse a partir de su primer elemento. Los elementos no necesariamente están en posiciones contiguas de memoria.

Para generar nuevos elementos en la lista, o eliminar alguno se deben utilizar las operaciones de new y dispose respectivamente.

Características: homogénea, dinámica, lineal y acceso secuencial.

## Algoritmos de ordenación

Un algoritmo de ordenación es un proceso por el cual un conjunto de elementos puede ser ordenado. Existen diferentes algoritmos de ordenación para vectores donde cada uno tiene sus ventajas y desventajas en cuanto a facilidad de escritura, cantidad de memoria requerida y tiempo de ejecución.

### Algoritmo de selección

Dado un arreglo A y una dimensión lógica, el algoritmo repite (dimL-1) veces buscar el elemento mínimo y ubicarlo en la posición correspondiente.

Es decir, en la primera vuelta busca el mínimo desde la posición 1 hasta la dimensión lógica y ubica al mínimo en la primera posición. En la vuelta i busca el mínimo desde la posición i hasta la dimensión lógica y lo ubica en la posición i.

Procedure Ordenar ( var v: tVector; dimLog: indice );

var i, j, p: indice; item : tipoElem;

begin

for i:=1 to dimLog-1 do begin {busca el mínimo y guarda en p la posición}

p := i;

for j := i+1 to dimLog do

if v[ j ] < v[ p ] then p:=j;

{intercambia v[i] y v[p]}

item := v[ p ];

v[ p ] := v[ i ];

v[ i ] := item;

end;

end;

* Es muy sencilla la implementación.
* El tiempo de ejecución en N2, siendo N el tamaño del arreglo.

### Algoritmo de inserción

Es una manera muy natural de ordenar un conjunto de elementos almacenados en un arreglo.

Inicialmente se considera un solo elemento, que obviamente es un conjunto ordenado.

Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento k+1 y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se inserta el elemento k+1 debiendo desplazarse los demás elementos.

Procedure Ordenar ( var v: tVector; dimLog: indice );

var i, j: indice; actual: tipoElem;

begin

for i:=2 to dimLog do begin

actual:= v[i];

j:= i-1;

while (j > 0) y (v[j] > actual) do

begin

v[j+1]:= v[j];

j:= j – 1;

end;

v[j+1]:= actual;

end;

end;

* No es tan sencillo de implementar.
* El tiempo de ejecución en N2, siendo N el tamaño del arreglo.
* Si los datos están ordenados de menor a mayor el algoritmo solo hace comparaciones, por lo tanto, es de orden (n).
* Si los datos están ordenados de mayor a menor el algoritmo hace todas las comparaciones y todos los intercambios, por lo tanto, es de orden (N2). comparaciones.

## Recursión

La recursión es una metodología para resolver problemas. Permite resolver un problema P por resolución de instancias más pequeñas P1, P2, … Pn del mismo problema.

El problema es siempre el mismo, pero debe ir achicándose. Siempre tienen al menos un caso base, en el cual el código a implementar no es recursivo (a veces no debe escribirse código en el caso base, pero siempre existe). Puede haber más de un caso base. El caso base es el que determina el final de la solución recursiva.

### Arboles

Un árbol binario de búsqueda es una estructura de datos jerárquica. Está formada por nodos, donde cada nodo tiene a lo sumo dos hijos y mantienen un orden. El nodo principal del árbol se denomina raíz y los nodos que no tienen hijos se denominan hojas del árbol. Características: homogénea, dinámica, no lineal y acceso secuencial.

Procedimiento para crear un árbol:

Procedure crear (var A:arbol; num:integer);

Begin

if (A = nil) then

begin

new(A);

A^.dato:= num; A^.HI:= nil; A^.HD:= nil;

end

else

if (num < A^.dato) then crear(A^.HI,num)

else crear(A^.HD,num)

End;

Procedimientos para imprimir pos, pre e in orden:

Procedure preOrden ( a : arbol );

begin

if ( a<> nil ) then begin

write (a^.dato);

preOrden (a^.HI);

preOrden (a^.HD);

end;

end;

Procedure posOrden ( a : arbol );

begin

if ( a<> nil ) then begin

posOrden (a^.HI);

posOrden (a^.HD);

write (a^.dato);

end;

end;

Procedure enOrden ( a : arbol );

begin

if ( a<> nil ) then begin

enOrden (a^.HI);

write (a^.dato);

enOrden (a^.HD);

end;

end;

Buscar un elemento en un árbol:

Function Buscar (a:arbol; x:elemento): arbol;

begin

if (a=nil) then

Buscar:=nil

else if (x = a^.dato) then Buscar:=a

else

if (x < a^.dato) then

Buscar:= Buscar(a^.hi ,x)

else

Buscar:=Buscar(a^.hd ,x)

end;

* Un árbol binario de búsqueda agrega los elementos por sus hojas. Dichos elementos quedan ordenados. Esta operación lleva un tiempo de ejecución de O(log n).
* En un árbol binario de búsqueda la búsqueda de un elemento es de O(log n).

## Merge

La operación de merge consiste en generar una nueva estructura de datos (arreglos, listas) ordenada a partir de la mezcla de dos o más estructuras de datos previamente ordenadas. Las estructuras que se combinan guardan el mismo orden lógico interno (por ejemplo datos ordenados alfabéticamente).

Procedure merge (todos:vectorListas; var nuevo:lista);

Begin

nuevo:= nil;

minimo (todos,minNombre, monto);

while (minNombre <> ‘ZZZ’) do

begin

actual:= minNombre; montoTotal:=0;

while ((minNombre <> ‘ZZZ’) and (minNombre = actual) )do

begin

montoTotal:= montoTotal + monto;

minimo (todos,minNombre,monto);

end;

agregarAtras(nuevo,actual,montoTotal);

end;

End;

Procedimiento mínimo con vector de listas:

Procedure minimo (var todos:estantes; var nomMin:string; var monto:real);

Var

indiceMin,i:integer;

Begin

nomMin:= ‘ZZZ’;

for i:= 1 to cantE do

if (todos[i] <> nil) then

if (todos[i]^.dato.nombre <= nomMin) then begin

indiceMin:= i;

nomMin:= todos[i]^.dato.nombre;

end;

if (nomMin <> ‘ZZZ’) then

begin

//nomMin:= todos[indiceMin]^.dato.nombre;

monto:= todos[indiceMin] ^.dato.monto;

todos[indiceMin]:= todos[indiceMin]^sig;

end;

End;

Procedimiento merge con dos listas:

Procedure minimo(var e1,e2:lista; var min:string);

Begin

min := ‘ZZZ’;

if (e1 <> nil) and (e2 <> nil)then

if (e1^.dato <= e2 ^.dato ) then

begin

min:= e1^.dato;

e1:= e1 ^.sig;

end;

else begin

min:= e2 ^.dato;

e2:= e2 ^.sig;

end

else

if (e1 <> nil) and (e2 = nil) then begin

min:= e1^.dato;

e1:= e1 ^.sig;

end

else

if (e1 = nil) and (e2 <> nil) then begin

min:= e2 ^.dato;

e2:= e2 ^.sig;

end;

end;

## Modulo Concurrente

Un programa concurrente se divide en tareas (2 o más), las cuales se ejecutan al mismo tiempo y realizan acciones para cumplir un objetivo común. Para esto pueden: compartir recursos, coordinarse y cooperar. La comunicación sincrónica es un concepto clave en la Ciencia de la Computación.

Tiene que haber comunicación y sincronización.

Envió de Mensajes (Enviar – Recibir)

* Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos.
* También el lenguaje debe proveer un protocolo adecuado.
* Para que la comunicación sea efectiva los procesos deben “saber” cuándo tienen mensajes para leer y cuando deben transmitir mensajes.

Memoria compartida (Bloquear - Desbloquear)

* Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella.
* Lógicamente no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria.
* La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida.

Si esta libre tengo que bloquear – usar – desbloquear.

### Pasaje de mensajes

Envió de mensajes (asincrónico): Un robot después de enviar un mensaje puede continuar su ejecución.

Recepción de mensajes (sincrónico): Un robot que está esperando recibir un mensaje NO puede continuar su ejecución hasta que otro robot no le haya mandado el mensaje.

Asincrónico: El proceso que envía/recibe el mensaje NO espera que se de la comunicación para continuar su ejecución.

Sincrónico: El proceso que envía/recibe el mensaje SI espera que se de la comunicación para continuar su ejecución.

### Memoria compartida

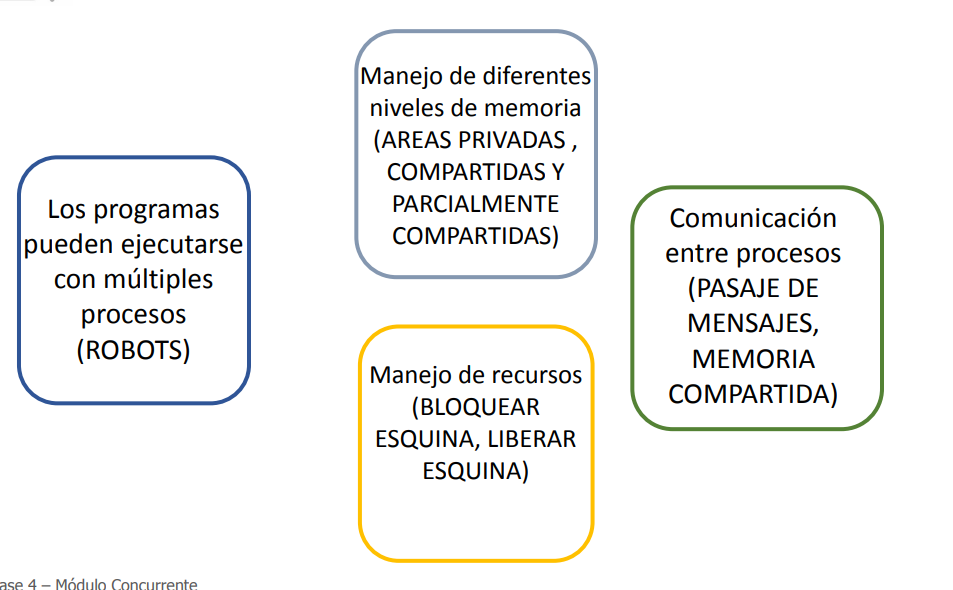
Bloquear un recurso: Dado un recurso DISPONIBLE el programador bloquea dicho recurso para que ningún otro proceso pueda acceder.

Desbloquear un recurso: Dado un recurso BLOQUEADO el programador libera dicho recurso para que cualquier proceso pueda bloquearlo.

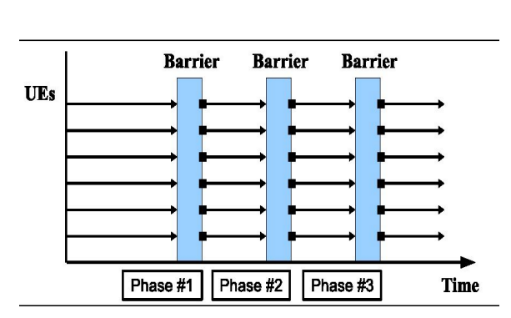
Programador: El control de los accesos al recurso se encuentra a cargo del código desarrollado por el programador.

Sistema operativo: El control de los accesos al recurso se encuentra a cargo del sistema operativo.

* No debo bloquear recursos sino es necesario.
* Debo liberar un recurso lo más pronto posible



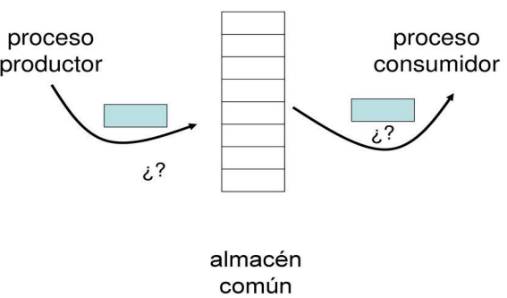
### Sincronización por barrera

Múltiples procesos se ejecutan concurrentemente, hasta que llegan a un punto especial, llamado barrera. Los procesos que llegan a la barrera, deben detenerse y esperar que todos los procesos. Cuando todos los procesos alcanzan la barrera, podrán retomar su actividad (hasta finalizar o hasta alcanzar la próxima barrera). Para esto los procesos deben avisar que llegaron. Ejemplos: La vacunación por COVID. Un partido de paddle.

### Passing the battom

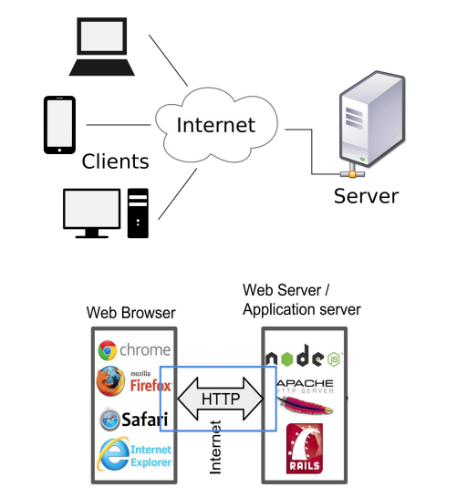
Múltiples procesos se ejecutan en paralelo. Sólo un proceso a la vez, el que posee el testigo (baton), se mantiene activo. Cuando el proceso activo completa su tarea, entrega el baton a otro proceso. El proceso que entregó el baton queda a la espera hasta recibirlo nuevamente. El proceso que recibió el baton completa su ejecución. Al completar su tarea, pasará el baton a otro proceso. Para esto los procesos deben tener una forma de comunicarse con el otro proceso. Ejemplos: La producción de un producto. Una carrera de postas.

### Productor / Consumidor

Existen dos tipos de procesos: Productores: trabajan para generar algún recurso y almacenarlo en un espacio compartido. Consumidores: utilizan los recursos generados por los productores para realizar su trabajo. Para esto los procesos deben coordinar donde almacenan los datos los productores, cuando saben los consumidores que hay datos. Ejemplos: Corrección de parciales Cualquier sistema de producción.

### Servidor / Cliente

Los procesos se agrupan en dos categorías: Servidores: permanecen inactivos hasta que un cliente les solicita algo. Cuando reciben una solicitud, realizan su tarea, entregan el resultado y vuelven a “dormir” hasta que otro cliente los despierte Clientes: realizan su trabajo de manera independiente, hasta que requieren algo de un servidor. Entonces realizan una solicitud a un proceso servidor, y esperan hasta que reciben la respuesta. Cuando esto sucede, el cliente continúa su trabajo. Para esto los procesos cliente deben realizar sus pedidos y el servidor debe administrar cómo los atiende. Ejemplos: Cualquier navegador (cliente) que accede a un página (servidor) Cualquier empresa



### Master / Slave

Los procesos se agrupan en dos categorías: Maestro: deriva tareas a otros procesos (trabajadores) Esclavos: realizan la tarea solicitada y envían el resultado al jefe, quedando a la espera de la siguiente tarea Para esto, el proceso jefe determina cuántos trabajadores necesita, cómo les reparte la tarea, cómo recibe los resultados. Ejemplos: Buscar valores en un arreglo Remisería.

# Orientación a Objetos

• Main = “Programa principal”. { } delimita el cuerpo.

• Sentencias de código separadas por punto y coma (;).

• Se recomienda indentar el código para facilitar su lectura.

• Comentarios:

• De líneas múltiples /\* Esto es un comentario \*/

• De línea única // Este es un comentario

• Case-sensitive (sensible a las mayúsculas y minúsculas)

### Uso de Lector (funcionalidad definida en PaqueteLectura.Lector)

import PaqueteLectura.Lector; // Importar funcionalidad para lectura

public class Demo05Entrada {

public static void main(String[] args) {

System.out.println("Ingrese nombre");

String nombre = Lector.leerString();

}

### Generación de datos aleatoria Uso de GeneradorAleatorio (funcionalidad definida en PaqueteLectura.GeneradorAleatorio)

import PaqueteLectura.GeneradorAleatorio; // Importar funcionalidad Generador Aleatorio public class Demo06Generador {

public static void main(String[] args) {

GeneradorAleatorio.iniciar(); //Inicia el generador aleatorio System.out.println(GeneradorAleatorio.generarInt(10)); //Genera un int entre 0 y 9

Objeto: abstracción de un objeto del mundo real, que define qué lo caracteriza (estado interno) y qué acciones sabe realizar (comportamiento). Entidad que combina en una unidad

**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.

**Encapsulamiento** (ocultamiento de información) Se oculta la implementación del objeto hacia el exterior. Desde el exterior sólo se conoce la interfaz del objeto. Facilita el mantenimiento y evolución del sistema ya que no hay dependencias entre las partes del mismo.

Una **clase** describe un conjunto de objetos comunes (mismo tipo). Consta de:

• La declaración de las v.i. que implementan el estado del objeto.

• La codificación de los métodos que implementan su comportamiento.

• Un objeto se crea a partir de una clase (el objeto es instancia de una clase).

La **instanciación** se realiza enviando un mensaje de creación a la clase.

• Reserva de espacio para el objeto.

• Ejecución el código inicializador o constructor

• Devuelve la referencia al objeto.

• Asociar la referencia a una variable (a través de ella podemos enviarle mensajes al objeto).

Secuencia de pasos en la instanciación (creación de objeto):

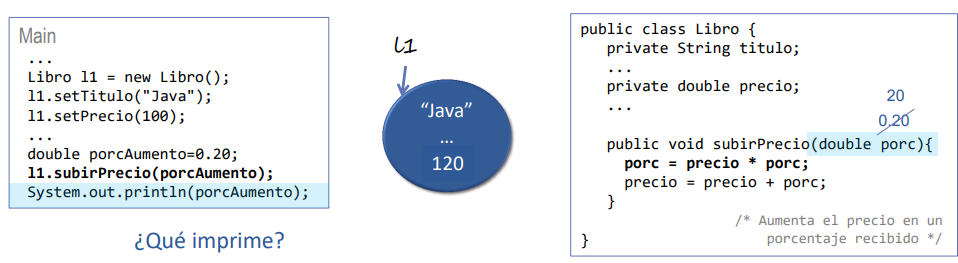
1. Reserva de Memoria. Las variables de instancia se inicializan a valores por defecto o explícitos.
2. Ejecución del Constructor. Inicia las variables de instancia con los valores que enviamos en el mensaje de creación.
3. Asignación de la referencia a la variable.

## Envío de mensaje al objeto. Funcionamiento parámetro por valor

### Parámetro dato primitivo

• Parámetro formal recibe una copia del parámetro actual (dato primitivo).

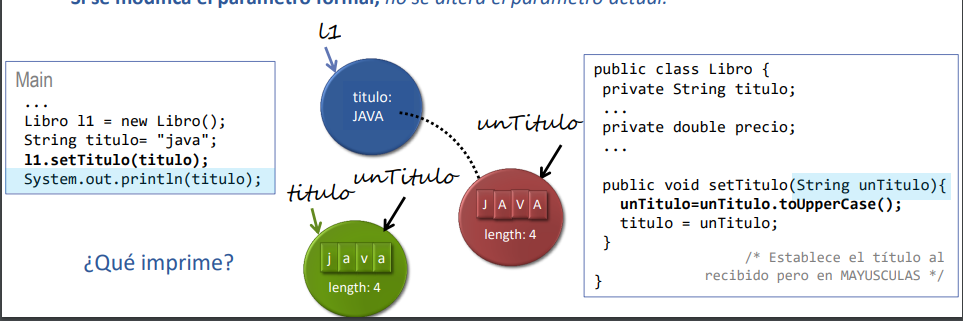
• Si se modifica el parámetro formal, no se altera el parámetro actual.



### Parámetro objeto

• Parámetro formal recibe una copia del parámetro actual (referencia al objeto).

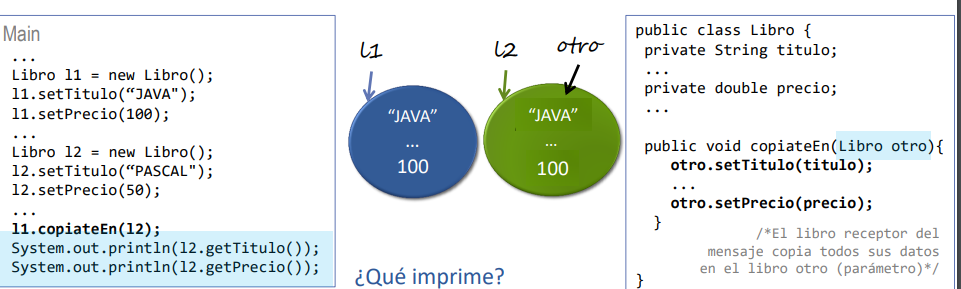
• Si se modifica el parámetro formal, no se altera el parámetro actual.



### Parámetro objeto

• Parámetro formal recibe una copia del parámetro actual (referencia al objeto).

• Si se modifica el estado del parámetro formal, el cambio es visible desde el parámetro actual.



Constructor: código que se ejecuta tras reservar memoria para el objeto e iniciar las v.i. al valor por defecto o explícito. Toma valores pasados en el mensaje de creación. Su objetivo es iniciar el objeto.

* Al instanciar el objeto (new) se ejecutará un sólo constructor.

• Java identifica cuál ejecutar según el número y tipo de parámetros pasados al instanciar.

• No puede haber dos constructores con igual número y tipo de parámetros.

## Herencia

Mecanismo que permite que una clase herede características y comportamiento

(atributos y métodos) de otra clase (clase padre o superclase). A su vez, la clase hija define

características y comportamiento propio.

• La subclase hereda atributos y métodos de instancia declarados en la superclase.

• Nota: los atributos declarados en una superclase como privados no son accesibles directamente en sus

subclases. Para accederlos en una subclase se deben usar los getters y setters heredados.

• La subclase puede declarar atributos, métodos y constructores propios.

**Clase abstracta**: es una clase que no puede ser instanciada (no se pueden crear objetos a partir de ella). Se usa para definir características y comportamiento común para un conjunto de clases (subclases). Puede definir métodos abstractos (sin implementación) que deben ser implementados por las subclases.

## Resumen de conceptos de POO

• Encapsulamiento: permite construir componentes autónomos de software, es decir independientes de los demás componentes. La independencia se logra ocultando detalles internos (implementación) de cada componente. Una vez encapsulado, el componente se puede ver como una caja negra de la cual sólo se conoce su interfaz.

• Herencia: permite definir una nueva clase en términos de una clase existente. La nueva clase hereda automáticamente todos los atributos y métodos de la clase existente, y a su vez puede definir atributos y métodos propios.

• Polimorfismo: objetos de clases distintas pueden responder a mensajes con selector (nombre) sintácticamente idéntico de distinta forma. Permite realizar código genérico, altamente reusable.

• Binding dinámico: mecanismo por el cual se determina en tiempo de ejecución el método (código) a ejecutar para responder a un mensaje.

## Beneficios de la POO

Producir software que sea:

• Natural. El programa queda expresado usando términos del problema a resolver, haciendo que sea más fácil de comprender.

• Fiable. La POO facilita la etapa de prueba del software. Cada clase se puede probar y validar independientemente.

• Reusable. Las clases implementadas pueden reusarse en distintos programas. Además, gracias a la herencia podemos reutilizar el código de una clase para generar una nueva clase. El polimorfismo también ayuda a crear código más genérico.

• Fácil de mantener. Para corregir un problema, nos limitamos a corregirlo en un único lugar.