**TALLER DE PROGRAMACIÓN**

***Imperativo:***

**Arreglo:**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. Cada componente es almacenado en posiciones contiguas de memoria.

Estática, Homogénea e indexada. Trabaja con:

* *Dimensión Física (DIMF):* Cantidad máxima de elementos que puede almacenar.
* *Dimensión Lógica (DIML)*: Cantidad de elementos en un momento dado.

Siempre DIMF >= DIMF

*Operaciones:*

1. Cargar la estructura.
2. Agregar un elemento.
3. Insertar un elemento.
4. Eliminar un elemento.
5. Recorrer la estructura.
6. Buscar un elemento.
7. Ordenar la estructura.

***Lista:***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 (puntero). 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 new y dispoce respectivamente.

Homogénea, dinámica, lineal y secuencial.

*Operaciones:*

1. Cargar lista vacía.
2. Agregar un elemento adelante.
3. Agregar un elemento atrás.
4. Insertar un elemento.
5. Eliminar un elemento.
6. Recorrer la estructura.
7. Buscar un elemento.

**Algoritmo de ordenación:** es un proceso por el cual un conjunto de elementos puede ser ordenado. La principal ventaja del ordenamiento es la Búsqueda eficiente.

Existen diferentes algoritmos de ordenación para vectores donde cada uno tiene ventajas y desventajas en cuanto a facilidad de escritura, cantidad de memoria requerida y tiempo de ejecución.

* **Selección:** Dado un elemento A y una dimensión lógica DIML, el algoritmo repite DIML-1 veces buscar el elemento mínimo y ubicarlo en la posición correspondiente.

Posición = 0

Repito:

1. Busco el mínimo.
2. Posiciono el mínimo en posición+1 y al elemento que estaba ahí lo pongo en la posición donde extraje el mínimo.
3. DIML-1

Ventajas:

* Es muy sencilla la implementación.
* El tiempo de ejecución en N^2, siendo N el tamaño del arreglo.
* Si los datos están ordenados inicialmente
* **Inserción** (Insert sort)**:** 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 ordenador, 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.

**Recursión:** es una forma de resolver problemas que no está atada a ninguna estructura de control en particular. En otras palabras, es una metodología para resolver problemas.

Permite resolver un problema P por resolución más pequeñas P1, P2, … Pn del mismo problema.

El problema Pi es de la misma naturaleza que el problema original, pero en algún sentido es más simple.

Este procedimiento se hace hasta el momento donde el input que voy reduciendo es lo suficientemente básico para tomar una solución directa (Caso base). Puede haber más de 1 caso base y siempre tiene al menos 1 en el cual el código a implementar no es recursivo. El caso base determina el final de la solución.

El problema siempre es el mismo, solo se va achicando hasta llegar al caso base.

**Árbol binario de búsqueda:** estructura de datos jerárquica formada por nodos, donde cada nodo tiene a lo sumo 2 hijos y mantiene un orden. El nodo principal del árbol se denomina raíz y los nodos que no tienen hijos se denominan hojas. Homogénea, dinámica y no lineal.

El árbol es una estructura donde la utilización de la recursión para trabajar con él simplifica las operaciones.

Una convención: menores a la raíz a la izquierda y mayores a la derecha.

Como el árbol NO es vacío tengo que recorrer desde la raíz hasta el lugar correspondiente respetando el orden. Siempre se inserta en una hoja. Siempre debo generar un espacio para el nuevo dato.

El tiempo de buscar y agregar es de O(log n).

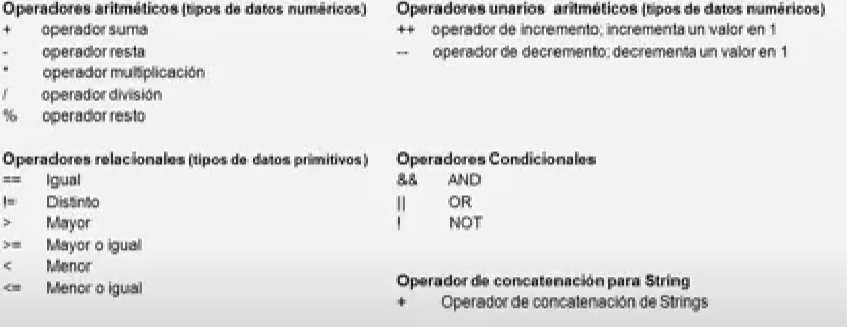
**Marge:** consiste en generar una nueva estructura de datos (arreglo/ lista) ordenada a partir de la mezcla de 2 o más estructuras de datos previamente ordenadas.

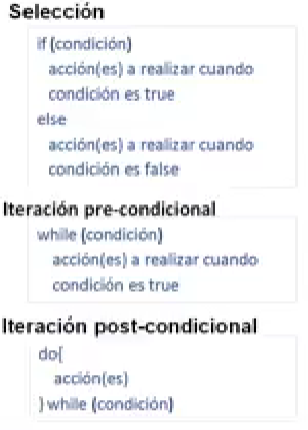
Las estructuras que se combinan guardan el mismo orden lógico interno.

BUSQUEDA DICOTOMICA

***Objetos:***

**JAVA:**

* Lenguaje de propósito general. Paradigma orientado a objetos.
* Permite generar aplicaciones multiplataforma.
* Plataforma Java:
  + Plataforma de desarrollo (JDK: Java Development Kit): incluye compilador, depurador, generador de documentación, etc.
  + Plataforma de ejecución (JRE: Java Runtime Enviroment): incluye componentes para ejecutar aplicaciones Java, entre ellas la JVM (Java Virtual Machine).
* Case sensitive.
* Variables:
  + No toman valor por defecto.
  + CamelCase.
  + nombre = valor.
  + Tipos primitivos: boolean, char, int, double
  + Operadores para tipos primitivos y strings.
* Estructuras de control:



* Vector (Arreglos unidimensionales): Almacenan un número fijo de valores primitivos (objetos). Se accede de forma directa a las posiciones. La DIMF se establece al crearlo y el índice es entero y comienza en 0.
  + Declaración: TipoElemento [] nombreVariable;
  + Creación: nombreVariable = new TipoElemento[DIMF];
  + Acceso a elemento: nombreVariable[Posición];
* Matrices (Arreglos bidimensionales): Colección ordenada e indexada de elementos. Esta estructura de datos compuesta permite acceder a cada campo utilizando 2 índices (fila y columna) que permite ubicar un elemento dentro de la estructura. Homogénea, estática, indexada y lineal.
  + Declaración: TipoElemento [][] nombreVariable;
  + Creación: nombreVariable = new TipoElemento[DIMF1][DIMF2];
  + Acceso a elemento: nombreVariable[Posición1][Posición2];

**Paradigmas de programación:** Indica la manera de estructurar y organizar las tareas de nuestro programa. Los lenguajes de programación suelen ser multiparadigma.

Método = procedimiento.

**Objeto:** “Todo es un objeto”, es una abstracción del mundo real definiendo que lo caracteriza (estado interno) y que 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.

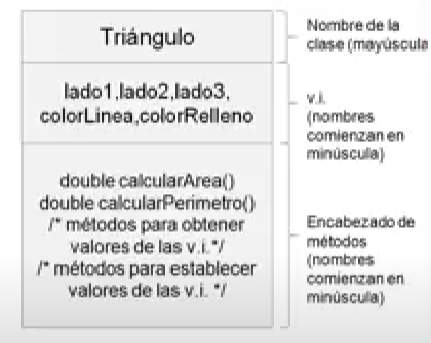
**Envío de Mensaje:** provoca la ejecución del método indicado por el nombre del mensaje. Puede llevar datos (parámetros del método) y/o puede devolver un dato (resultado del método).

**Clase:** describe un conjunto de objetos comunes del mismo tipo. Consta de:

* La declaración de las variables de instancia 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 una instancia de una clase).

Representación gráfica de una clase:



**Instanciación (Creación de objeto):** se realiza enviando un mensaje de creación a la clase. Se reserva espacio para el objeto con la ejecución del código iniciador o constructor. Devuelve la referencia al objeto, se asocia la referencia a una variable que a través de ella podemos enviarle mensajes al objeto.

* Declarar variable para mantener la referencia: nombreDeClase miVariable;
* Enviar a la clase un mensaje de creación y guardar referencia: miVariable = new NombreDeClase(Valores de inicialización);
* Se puede unir los 2 pasos anteriores: nombreDeClase miVariable = new NombreDeClase(Valores de inicialización);

Secuencia de pasos:

* Reserva de Memoria: las variables de instancia se inicializan a valores por defecto o explícitos.
* Ejecución del Constructor: código para inicializar variables de instancia con los valores que envíamos en el mensaje de creación.
* Asignación de la referencia a la variable.

**Programa orientado a objetos:**

* Los programas se organizan como una colección de objetos que cooperan entre sí enviándose mensajes.
* Cada objeto es instancia de una clase.
* Los objetos se crean a medida que se necesitan.
* El usuario le envía un mensaje a un objeto, en caso de que un objeto conozca a otro puede enviarle un mensaje, así los mensajes fluyen por el sistema.
* Cuando los objetos ya no son necesarios se borran de la memoria.

**Desarrollo de SW OO. Pasos:**

* Identificar los objetos a abstraer en nuestra aplicación.
  + Identificar las características relevantes de los objetos.
  + Identificar las acciones relevantes que realizan los objetos.
* Los objetos con características y comportamiento similar serán instancia de una misma clase.

**Referencias:** ubicación en memoria del objeto. Valor default = null. El recolector de basura (garbage collector) libera memoria de objetos no referenciados. Con == se comparan las referencias y con equals el contenido de los objetos.

**Clases:**

public class NombreDeClase{

Declaración del *estado* del objeto (variables);

Declaración de *constructor*(es);

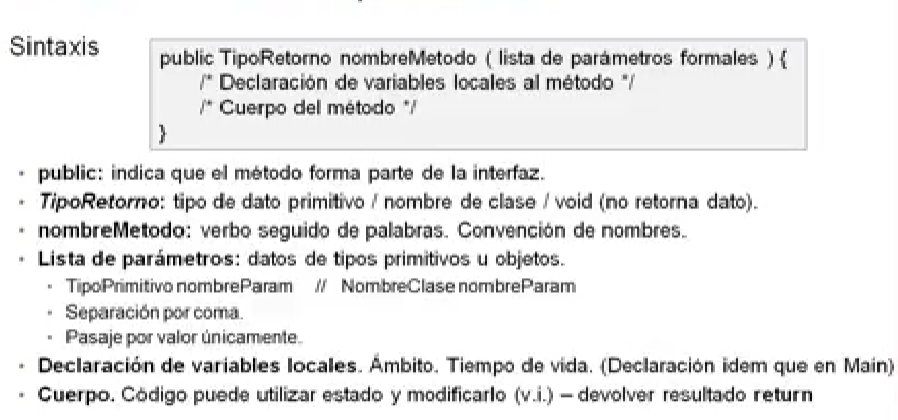
Declaración de *métodos* que implementen acciones;

}

***Estado interno:*** Datos primitivos o Referencia a otros objetos.

La variables de instancia privadas pueden ser accedidas solo dentro de la clase que las declara. Anteponer a la declaración la palabra private para lograr encapsulamiento.

En la declaración del dato se puede dar un valor inicial (inicialización explícita). Cuando no inicializamos toma un valor por defecto.



Parámetros: únicamente pasaje por valor.

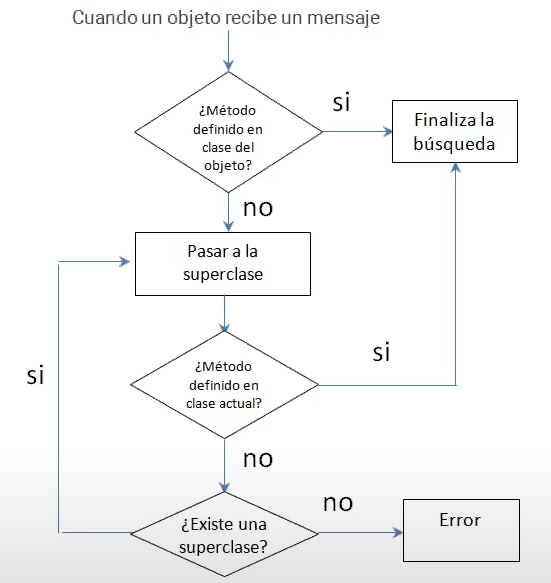
* Parámetro dato primitivo: Parámetro formal recibe copia del valor del parámetro actual. Si se modifica el parámetro formal no altera el parámetro actual.
* Parámetro objeto: Parámetro formal recibe copia de la referencia del parámetro actual. Si se modifica el estado interno del objeto parámetro formal el cambio es visible en el parámetro actual.

***Constructor:*** se ejecuta tras alocar el objeto e inicializar las variables de instancia (por defecto o explícitamente). Si la clase no declara ningún constructor, Java incluye uno sin parámetros y sin código (constructor nulo).

Sobrecarga: cuando hay varios constructores para la clase. Java identifica cual está siendo invocado por el número y tipo de parámetros.

**Herencia:** es un mecanismo que permite que una clase **herede** características y comportamientos (atributos y métodos) de otra clase (clase padre o superclase). A su vez la clase hija define sus propias características y comportamientos. La principal ventaja es la reutilización de código.

Jerarquía de clases:



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

Los métodos abstractos son métodos para los cuales se define su encabezado, pero no se los implementa. Las clases que heredan de una superclase abstracta deberán implementar sus métodos abstractos.

**Polimorfismo:** objetos de clases distintas responden al mismo mensaje de distinta forma. Esto permite realizar código gémerico y reusable.

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

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

El principal beneficio de la POO es que podemos producir SW 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 SW. Cada clase se pude 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.

***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 se pueden compartir recursos, coordinarse y cooperar.

Comunicación y Sincronización: conceptos claves de la ciencia de la computación. La necesidad de trabajar de manera concurrente genera cambios en HW y SW.

Formas de comunicar procesos entre sí:

* Envió de mensajes: es necesario establecer un canal (lógico o físico) para transmitir la información. Como en todos los envíos de mensaje se establece un protocolo adecuado que define las reglas. Para que la comunicación sea efectiva los procesos deben saber cuándo tienen mensajes para leer y cuando deben transmitir mensajes.

Siempre hay un emisor y un receptor, esta comunicación puede ser:

* + Asincrónico (envío): un robot después de enviar el mensaje puede continuar la ejecución. El mensaje puede incluir un solo valor como contenido. La recepción siempre es sobre una variable.
  + Sincrónico (recepción): un robot que está esperando recibir un mensaje NO puede continuar su ejecución hasta que otro robot no le haya mandado el mensaje. RecibirMensaje(var,\*) no implica que en \* esté almacenado el número del robot q hizo el envío.

El proceso que envía/recibe el mensaje NO espera que se dé la comunicación para continuar su ejecución. El mensaje puede incluir un solo valor como contenido. La recepción siempre es sobre una variable.

* Memoria compartida: se comparte una memoria y se trabaja con el bloqueo, hay un bloque de memoria compartido en el que no pueden operar 2 o más procesos obligando a bloquear y liberar. La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida. los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre residentes en ella.

**Áreas:**

* Compartida: cualquier robot puede circular por la misma.
* Privada: solo puede haber en ella un único robot.
* Parcialmente compartida: se debe seleccionar que subconjunto de robots pueden circular en la misma.