Taller de Programación

Guia teoria

Módulo 1: Pascal- Imperativo

Clase 1: Ordenación

ARREGLOS - Características

Un arreglo es una estructura de datos compuesta que permite acceder a cada componente por un índice variable. La posición del componente en la estructura se determina mediante el índice. Los arreglos se almacenan en posiciones contiguas de memoria.

- **Homogénea:** Los componentes del arreglo son del mismo tipo.
- **Estática:** La cantidad de elementos en el arreglo se define al inicio y no cambia.
- **Acceso directo:** Se puede acceder directamente a cualquier elemento utilizando su índice.
- **Indexada:** La estructura se accede mediante índices numéricos.
- **Lineal:** Los elementos están dispuestos en una sola dimensión.
- **Dimensión física:** Número total de elementos que puede contener el arreglo.
- **Dimensión lógica:** Número real de elementos presentes en el arreglo.

LISTAS - Características

Una lista es una estructura de datos lineal compuesta por nodos, donde cada nodo contiene el dato y la dirección del siguiente nodo. Las listas se pueden recorrer desde su primer elemento. Los elementos no necesariamente están en posiciones contiguas de memoria. Se requieren operaciones de "new" y "dispose" para agregar o eliminar elementos.

- **Homogénea:** Los elementos de la lista son del mismo tipo.
- **Dinámica:** La cantidad de elementos puede cambiar dinámicamente.
- **Acceso secuencial:** Los elementos se acceden uno tras otro en secuencia.
- **Lineal:** Los elementos se encuentran en una sola línea de conexión.

ARREGLOS - Ordenación

Los algoritmos de ordenación permiten organizar conjuntos de elementos de manera ascendente o descendente. Diversos algoritmos de ordenación están disponibles, y cada uno presenta características distintas en términos de facilidad de implementación, uso de memoria y tiempo de ejecución.

No solo el tiempo de ejecución es relevante al elegir un algoritmo de ordenación, sino también otros factores como la facilidad de implementación, uso de memoria y complejidad de estructuras auxiliares necesarias. La disposición inicial de los datos (ordenados, inversamente ordenados o desordenados) también afecta el rendimiento de los algoritmos.

Algoritmos de ordenación y su orden de ejecución:

- Selección / O(N^2)
- Intercambio / O(N^2)
- Inserción / O(N^2)
- Heapsort / O(N log N)
- Mergesort / O(N log N)
- Quicksort / O(N log N)

Selección: Comparación de elementos, moviendo el menor a la posición actual. Complejidad cuadrática O(N^2).

Intercambio: Comparación y cambio de elementos adyacentes. Complejidad cuadrática O(N^2).

Inserción: Inserta elementos en su posición correcta, desplazando los mayores. Complejidad cuadrática O(N^2).

Heapsort: Organiza elementos en un heap binario. Complejidad O(N log N), eficiente.

Mergesort: Divide en mitades, ordena. Complejidad O(N log N), estable.

Quicksort: Divide usando un pivote, ordena recursivamente. Complejidad O(N log N), rápido.

Ordenación por Selección en Arreglos

La Ordenación por Selección implica buscar y cambiar el elemento mínimo en cada iteración, intercambiándolo con el primer elemento no ordenado. Repite (dimensión lógica - 1) veces.

Proceso Iterativo

- Buscar la posición del mínimo en un rango.
- Intercambiar mínimo con primer no ordenado.
- Repetir en rango no ordenado.

Características

- Implementación sencilla.
- Complejidad cuadrática O(N^2).
- Requiere conocer dimensiones y posiciones.

Ordenación por Inserción en Arreglos

La Ordenación por Inserción implica insertar cada elemento en su posición correcta dentro de un subconjunto ya ordenado, en cada iteración.

Proceso Iterativo

- Seleccionar un elemento no ordenado.
- Insertarlo en su posición dentro del subconjunto ordenado.
- Repetir para todos los elementos no ordenados.

Características

- Implementación más compleja que la selección.
- Complejidad cuadrática O(N^2).
- Rendimiento mejor en datos casi ordenados.
- O(N) si los datos ya están ordenados (ascendente).
- O(N^2) si los datos están en orden descendente.

Clase 2: Recursión

La recursión es una técnica poderosa para resolver problemas que pueden subdividirse en instancias más pequeñas del mismo problema. Se basa en la idea de dividir y conquistar, donde un problema se resuelve al dividirlo en subproblemas más simples hasta alcanzar casos base.

Recursión:

- Problemas que se subdividen en subproblemas.
- Se resuelve llamando al mismo módulo (procedimiento o función).
- Evita el uso de bucles e iteradores.

Caso Base:

- Punto donde el problema ya no se puede subdividir más.
- Tiene una solución trivial o directa.

Variantes en Casos Base:

- Algunos casos base requieren acciones específicas.
- En otros casos base no se requieren acciones adicionales.

La recursión es una estrategia efectiva para abordar problemas complejos al descomponerlos en partes más simples. Utiliza llamadas repetidas al mismo módulo, logrando una solución más elegante y concisa en comparación con bucles tradicionales.

Clase 3: Estructura de datos Árbol

Los árboles son estructuras de datos jerárquicas que consisten en nodos interconectados, donde cada nodo puede tener varios hijos.

Características de los Árboles

- **Jerárquicos:** Los árboles siguen una estructura jerárquica, donde los nodos se organizan en niveles. Cada nodo excepto la raíz tiene un padre y cero o más hijos.
- **Dinámicos:** Los árboles son estructuras dinámicas, lo que significa que pueden crecer o disminuir en tamaño durante la ejecución del programa.
- **No Lineales:** A diferencia de las estructuras de datos lineales como los arrays o las listas, los árboles no siguen una estructura lineal. Los nodos pueden tener múltiples conexiones.

Elementos Importantes

- **Raíz:** Es el nodo principal del árbol, desde donde se inicia la jerarquía.
- **Hojas:** Son los nodos que no tienen hijos y se encuentran en el nivel más bajo del árbol.

Creación y Representación

- La construcción de un árbol comienza con la creación de la raíz.
- Un árbol vacío se representa con el valor "nil".
- Los nuevos datos se insertan como hojas, es decir, como nodos que no tienen hijos.

Los árboles son esenciales en programación para representar relaciones complejas y jerarquías de datos. Su estructura jerárquica y dinámica los convierte en una herramienta poderosa para resolver diversos problemas en programación.

Creación de un Árbol

La creación de un árbol implica agregar valores de manera jerárquica siguiendo ciertas reglas. Veamos cómo se agrega una secuencia de valores (9, 18, 22, 19, 7, 50) a un árbol paso a paso:

1. Agregar 9:

- Como el árbol está vacío, el primer valor (9) se convierte en la raíz.
- Se reserva memoria para un nuevo nodo y se asigna el valor 9.
- Los campos de hijos izquierdo (HI) y derecho (HD) se establecen como nil.
- El árbol ahora apunta al nodo con valor 9.

2. Agregar 18:

- Se compara el valor 18 con la raíz (9), y como 18 > 9, se inserta a la derecha.
- Se reserva memoria para un nuevo nodo con valor 18 y se ajustan los enlaces HD y HI de los nodos necesarios.
- El árbol ahora tiene la raíz (9) con HD apuntando a 18.

3. Agregar 22:

- Se compara 22 con la raíz (9), y como 22 > 9, se inserta a la derecha.
- Se compara 22 con 18, y como 22 > 18, se sigue recorriendo hacia la derecha.
- Se compara 22 con 22, lo que significa que debe agregarse a la izquierda.
- Se reserva memoria para un nuevo nodo con valor 22 y se ajustan los enlaces HD y HI según corresponda.
- El árbol ahora tiene la raíz (9) con HD apuntando a 18, y 18 con HD apuntando a 22.

4. Agregar 19:

- Se compara 19 con la raíz (9), y como 19 > 9, se inserta a la derecha.
- Se compara 19 con 18, y como 19 > 18, se inserta a la derecha.
- Se compara 19 con 22, y como 19 <= 22, se inserta a la izquierda.
- Se reserva memoria para un nuevo nodo con valor 19 y se ajustan los enlaces HD y HI según corresponda.
- El árbol ahora tiene la raíz (9) con HD apuntando a 18, y 18 con HD apuntando a 22, y 22 con HI apuntando a 19.

5. Agregar 7:

- Se compara 7 con la raíz (9), y como 7 <= 9, se inserta a la izquierda.

- Se reserva memoria para un nuevo nodo con valor 7 y se ajustan los enlaces HD y HI según corresponda.
- El árbol ahora tiene la raíz (9) con HD apuntando a 18, y 18 con HD apuntando a 22, y 22 con HI apuntando a 19, y 9 con HI apuntando a 7.

6. Agregar 50:

- Se compara 50 con la raíz (9), y como 50 > 9, se inserta a la derecha.
- Se compara 50 con 18, y como 50 > 18, se inserta a la derecha.
- Se compara 50 con 22, y como 50 > 22, se inserta a la derecha.
- Se compara 50 con 19, y como 50 > 19, se inserta a la derecha.
- Se compara 50 con 7, y como 50 > 7, se inserta a la derecha.
- Se reserva memoria para un nuevo nodo con valor 50 y se ajustan los enlaces HD y HI según corresponda.
- El árbol ahora tiene la raíz (9) con HD apuntando a 18, y 18 con HD apuntando a 22, y 22 con HI apuntando a 19, y 9 con HI apuntando a 7, y 22 con HD apuntando a 50.

Resumen

La creación de un árbol implica agregar valores siguiendo un orden jerárquico. Cada valor se compara con los nodos existentes y se inserta en el lugar correcto como una hoja. De esta manera, se construye una estructura jerárquica que refleja las relaciones entre los valores de manera eficiente.

Recorrido en Árboles Binarios de Búsqueda (ABB)

El recorrido de un Árbol Binario de Búsqueda (ABB) es una operación fundamental que implica visitar cada nodo del árbol de manera sistemática. Los recorridos son útiles para realizar diversas acciones, como imprimir los valores de los nodos, agregar nodos en otra estructura, modificar nodos, entre otras. Existen tres tipos principales de recorridos en ABB: inorden, preorden y postorden.

Comenzando desde la Raíz

Para realizar cualquier tipo de recorrido en un ABB, es esencial comenzar desde el nodo raíz. La raíz es el punto de partida para explorar todos los demás nodos en el árbol.

Realizando la Acción en el Nodo Actual

Una vez posicionados en un nodo, se realiza la acción deseada con el valor contenido en ese nodo. Esto puede implicar imprimirlo, agregarlo en otra estructura de datos, modificar su valor, etc.

Recorrido Inorden, Preorden y Postorden

- **Inorden:** En el recorrido inorden, se sigue el orden "izquierda - nodo - derecha". Primero se recorre el subárbol izquierdo, luego se realiza la acción en el nodo actual y finalmente se recorre el subárbol derecho.
- **Preorden:** En el recorrido preorden, se sigue el orden "nodo - izquierda - derecha". Primero se realiza la acción en el nodo actual, luego se recorre el subárbol izquierdo y finalmente el subárbol derecho.
- **Postorden:** En el recorrido postorden, se sigue el orden "izquierda - derecha - nodo". Primero se recorren los subárboles izquierdo y derecho, y luego se realiza la acción en el nodo actual.

Realizando la Acción en los Hijos

Después de realizar la acción en el nodo actual, se repite el proceso para sus nodos hijos. En un ABB, los nodos hijos se dividen en dos categorías: hijo izquierdo (HI) y hijo derecho (HD). Se repiten los mismos pasos para cada uno de los hijos, lo que lleva a un recorrido exhaustivo de todo el árbol.

Clase 4: Estructura de datos Arbol - Buscar

Como el árbol está ordenado de tal forma que a la izquierda están los valores menores y a la derecha los mayores, en el recorrido para determinar el mínimo y el máximo no tiene sentido recorrer todo el árbol, se evalúa solo un lado y si no es nil se va directo al último nodo, si es nil, es decir, está vacío, el mínimo o máximo valor será la raíz, según sea el caso.

Para la búsqueda de cualquier tipo de valor siempre debe aprovecharse el orden del árbol.

Módulo 2: Java - POO

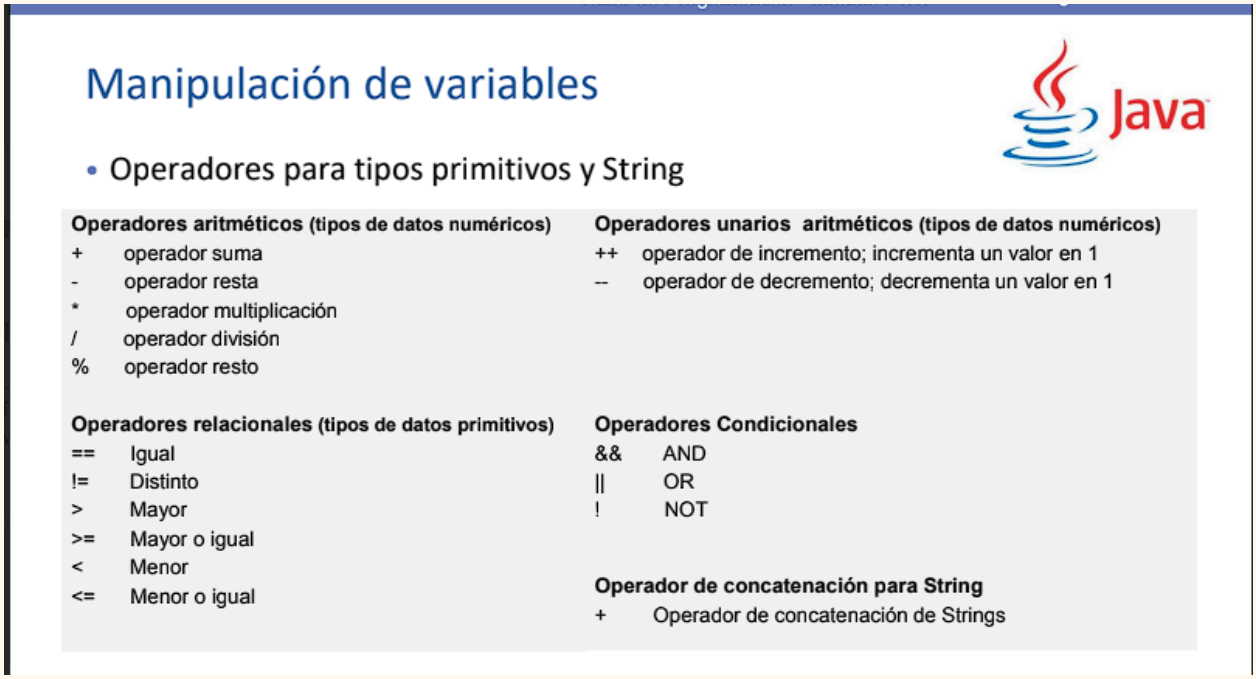
Clase 1: Introducción a Java y matrices

```
public class NombreAplicacion {
    public static void main(String[] args) {
        /* Código */
    }
}
```

- 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)



Clase 2: Introducción a POO en Java

Paradigmas de Programación:

Los paradigmas de programación indican la forma en que estructuramos y organizamos las tareas en un programa. Los lenguajes de programación suelen ser multiparadigma, y en este curso se aborda la Programación Orientada a Objetos (POO).

Conceptos Básicos de POO:

Objeto:

- Es una abstracción de un objeto del mundo real.
- Define su estado interno (datos/atributos) y comportamiento (acciones que puede realizar).
- Todo es considerado un objeto en POO.

Mensaje:

- El envío de un mensaje provoca la ejecución de un método asociado al nombre del mensaje.
- Puede incluir datos como parámetros y devolver un resultado.

Clase:

- Describe un conjunto de objetos que comparten características comunes.
- Incluye declaración de variables de instancia (estado) y codificación de métodos (comportamiento).

Instanciación (Creación de Objeto):

- Se realiza enviando un mensaje de creación a la clase.
- Implica reservar espacio para el objeto y ejecutar el código inicializador o constructor.
- Devuelve la referencia al objeto, que se asocia a una variable para interactuar con él.

Programación Orientada a Objetos en Java: Fundamentos Esenciales

Organización del Programa:

- Los programas se componen de objetos que interactúan mediante el envío de mensajes.
- Cada objeto es una instancia de una clase.
- Los objetos se crean según la necesidad y se eliminan de la memoria cuando ya no son necesarios.
- Los mensajes fluyen entre objetos, permitiendo la comunicación en el sistema.

Desarrollo de Software Orientado a Objetos:

- Identificación de objetos a abstraer.
- Identificación de características relevantes y acciones que realizan los objetos.
- Agrupación de objetos con características y comportamientos similares en una misma clase.

Objetos en Java:

Clases y Creación de Objetos:

- Java proporciona bibliotecas de clases para crear objetos de uso común.
- La creación de objetos implica enviar un mensaje de creación a la clase utilizando "new".
- Ejemplos incluyen la clase String y otras clases de utilidad.

Instanciación (Creación de Objeto):

- Se declara una variable para mantener la referencia al objeto.
- Se envía el mensaje de creación a la clase y se guarda la referencia.
- Los pasos en la instanciación incluyen reserva de memoria, ejecución del constructor y asignación de la referencia.

Referencias y Comparaciones:

- Las referencias apuntan a la ubicación en memoria del objeto.
- Se puede asignar "null" a una referencia para indicar que no apunta a ningún objeto.
- Las comparaciones de objetos se realizan mediante referencias y contenido.

Envío de Mensajes a Objetos en Java:

Sintaxis y Ejemplos:

- El envío de mensajes se realiza utilizando la sintaxis: objeto.nombreMétodo(...).
- Ejemplo de envío de mensajes a un objeto en Java.

Clase 3: POO en Java

- Sintaxis

```
public class NombreDeClase {
    /* Declaración del estado del objeto*/
    /* Declaración de constructor(es) */
    /* Declaración de métodos que implementan acciones */
}
```

Declaración del comportamiento.

Sintaxis

```
public TipoRetorno nombreMetodo ( lista de parámetros formales ) {
    /* Declaración de variables locales al método */
    /* Cuerpo del método */
}
```

- **Public**: indica que el método forma parte de la interfaz.
- **TipoRetorno:** tipo de dato primitivo / nombre de clase / void (no retorna dato).
- **nombreMetodo:** verbo seguido de palabras. Convención de nombres.
- **Lista de parámetros:** datos de tipos primitivos u objetos.
    - **TipoPrimitivo** nombreParam // NombreClase nombreParam
    - Separación por coma.
    - Pasaje por valor únicamente.
- **Declaración de variables locales.** Ámbito. Tiempo de vida. (Declaración idem que en Main)
- **Cuerpo.** Código puede utilizar estado y modificarlo (v.i.) – devolver resultado return

Declaración del comportamiento. Parámetros

- **Parámetro de dato primitivo:** el parámetro formal recibe una copia del parámetro actual. Si se modifica el parámetro formal, no se altera al parámetro actual.
- **Parámetro objeto:** el parámetro formal recibe una copia del parámetro actual. Si se modifica el estado interno del objeto, se verá reflejado en el parámetro actual. Si se modifica la referencia del objeto del parámetro formal, no se altera la referencia del parámetro actual.

Creación de objetos

Declarar variable para mantener la referencia:

- NombreDeClase miVariable;

Enviar a la clase el mensaje de creación:

- miVariable= new NombreDeClase();

Se puede unir los dos pasos anteriores:

- NombreDeClase miVariable= new NombreDeClase();

Secuencia de pasos en la creación:

- Reserva de Memoria. Las variables de instancia se inicializan a valores por defecto o explícito (si
- hubiese).
- Ejecución del Constructor (código para inicializar variables de instancia con los valores que
- enviamos el mensaje de creación).
- Asignación de la referencia a la variable.

Declaración de constructores

- Se ejecuta tras alocar al objeto e inicializar sus variables instancias
- Si la clase no declara a un constructor. Java incluye uno sin parámetros ni código llamado **constructor nulo.**

Interacciones entre objetos

- Un objeto puede formar puede ser una variable instancia de otro.

This

- Es un método que envía un mensaje al propio objeto, se usa para invocar a otros métodos del propio objeto.

  Ejemplo: **this.getNombre();**

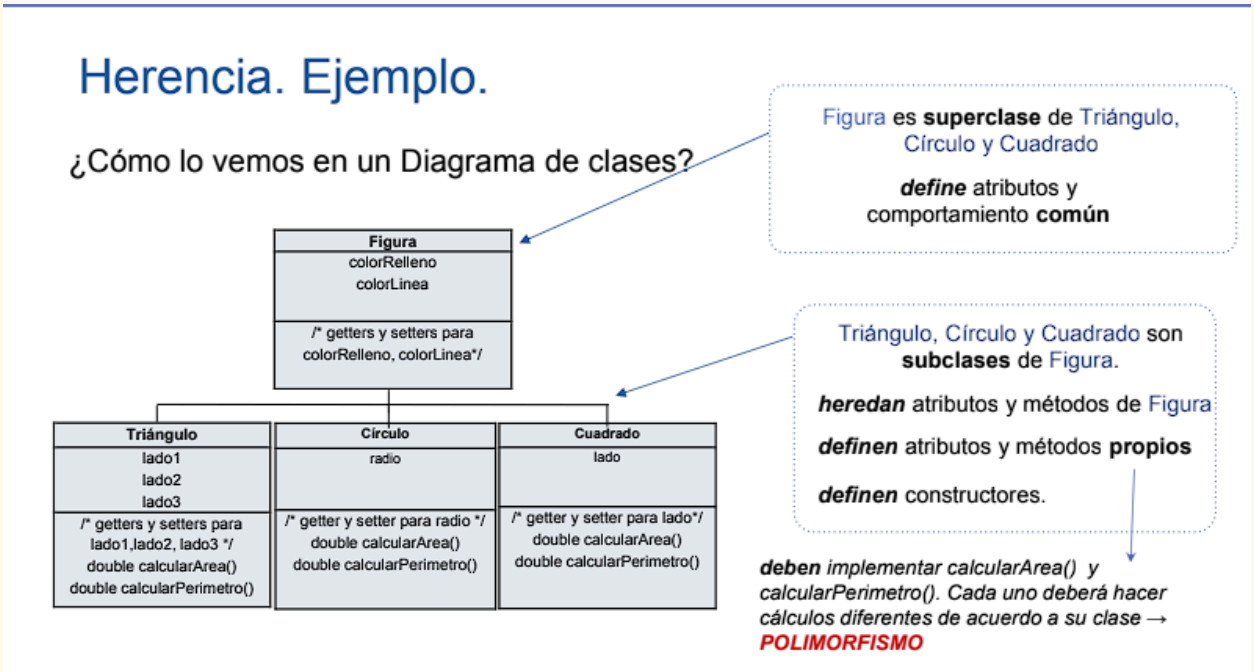
Clase 4: Herencia

Hasta ahora hemos trabajado en la creación de programas en los cuales declaramos clases con un conjunto de características (**atributos**) y comportamientos (**métodos**). A

A partir de esas clases hemos creado **objetos**.

Vimos que los objetos pueden interactuar entre ellos a través del **envío de mensajes** y que para realizar su tarea el objeto puede delegar trabajos en otro objeto que puede ser parte de él mismo o no.

Herencia: Es un 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 sus propias características y comportamiento.



Los atributos declarados en una superclase como privados no son accesibles en sus subclases.

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

Clase abstracta

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 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 hereden de una superclase abstracta deberán implementar sus métodos abstractos.

Declaración de clase abstracta: *anteponer **abstract** a la palabra class.*

```
public abstract class NombreClase {
      /* Definir atributos */
      /* Definir métodos no abstractos (con implementación) */
      /* Definir métodos abstractos  (sin implementación) */
}
```

Resumen del módulo: 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 nombre (selector) sintácticamente idénticos. Esto 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

- **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 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.

Módulo 3: Programación concurrente (R- info)

Clase 1: Evolución de arquitecturas, conceptos de concurrencia

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.

Cualquier lenguaje que brinde concurrencia debe proveer mecanismos para comunicar y sincronizar procesos.

Características

Comunicación: los procesos involucrados realizan un pasaje de mensajes y comparten memoria.

- **Pasaje de mensajes:** 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:** 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.

Ambiente CMRE (R-Info)

Características:

- **Robots:** se permite declarar múltiples robots.
- **Áreas:** distintos tipos (privadas, compartidas y parcialmente compartidas).
- **Comunicación:** permite el intercambio de mensajes entre los robots.
- **Sincronización:** permite bloquear o desbloquear recursos compartidos (esquinas).

Clase 2: Pasaje de mensajes

Dos procesos

- **Envío de mensajes:** Un proceso prepara un mensaje y selecciona uno o varios destinatarios para que lo reciban.
- **Recepción de mensajes:** Un proceso recibe un mensaje de un proceso determinado, o puede recibirlo de cualquiera de los procesos con los que interactúa.

Dos modos

- **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.

En R-Info

El envío de mensajes es asincrónico, es decir, el robot que envía el mensaje lo hace y sigue procesando sin esperar que el robot receptor lo reciba.

La recepción de mensajes es sincrónica, es decir, el robot que espera un mensaje NO sigue procesando hasta que recibe el mensaje.

```
EnviarMensaje(valor,variableRobot)

EnviarMensaje(variable,variableRobot)
```

```
RecibirMensaje(variable,variableRobot)
```

RecibirMensaje(variable, *): * indica “cualquier robot” que envíe un mensaje al receptor con el dato del mismo tipo que variable.

Clase 3: Memoria compartida

Bloquear recurso: Dado un recurso compartido (por 2 o más procesos) que está DISPONIBLE se bloquea ese recurso para que otro proceso no pueda accederlo.

-Consideraciones

- Puede realizarlo el programador o el Sistema Operativo
- Sólo se bloquea un recurso libre. Si el recurso ya está bloqueado no se debe intentar hacerlo.
- Hay que bloquear un recurso cuando puede ser accedido por dos o más procesos de un programa.

Liberar recurso: Dado un recurso compartido (por 2 o más procesos) BLOQUEADO el programador libera dicho recurso para que cualquier proceso pueda bloquearlo.

-Consideraciones

- Puede realizarlo el programador o el Sistema Operativo
- Sólo se libera un recurso ocupado. Si el recurso no está bloqueado no se debe intentar hacerlo.
- Hay que liberar un recurso cuando puede ser accedido por dos o más procesos de un programa.

```
BloquearEsquina(avenida,calle)

   BloquearEsquina(2,8)
      BloquearEsquina(posAv+1,posCa)
      BloquearEsquina(ave,ca)
```

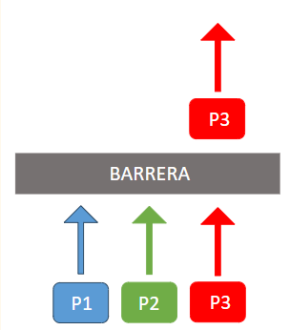
```
LiberarEsquina(avenida,calle)

   LiberarEsquina(2,8)
      LiberarEsquina(posAv+1,posCa)
      LiberarEsquina(ave,ca)
```

Clase 4: Algoritmos y arquitecturas concurrentes

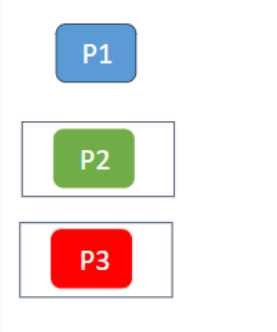
Tipos de problemas reales – Sincronización por barrera

- 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.



Tipos de problemas reales – Passing the baton

- Múltiples procesos se ejecutan en concurrente.
- 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.

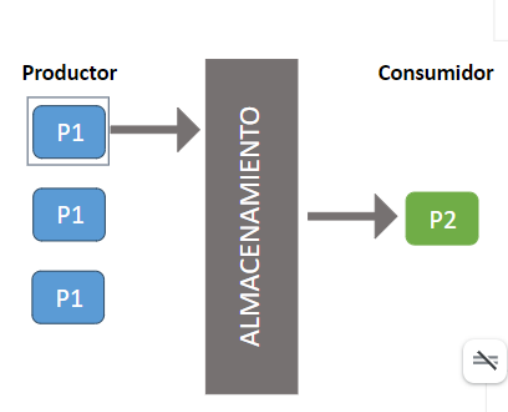


Tipos de problemas reales – Productor/consumidor

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.

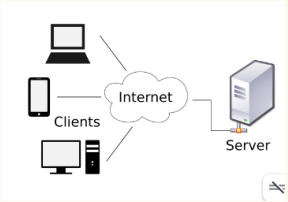


Tipos de problemas reales – Servidor/Cliente

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 como los atiende.



Tipos de problemas reales – 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.