


Paradigma Imperativo (Pascal)

Algoritmo de ordenación: proceso por el cual, un grupo de elementos puede ser ordenado.

¿Por qué es importante una operación de ordenación en arreglos?

Porque hay situaciones donde se nos presenta la información desordenada y la queremos ordenar para luego poder hacer búsquedas más eficientes.

Métodos de ordenación clásicos (más fáciles de implementar y más ineficientes)

- o Selección
 - o Intercambio
 - o Inserción
- 
- Estos métodos se basan en llevar a cada elemento del vector que se está ordenando al lugar adecuado.

Método de ordenación por inserción:

- o Parte de una secuencia de dos ítems y lo ordena.
- o En cada pasada se toma un nuevo ítem y se inserta en la posición adecuada en el arreglo ordenado de su izquierda.

Pseudocódigo:

```
Repetir desde  $i:=2$  hasta  $n$   
  guardar elemento (a ordenar)  
   $j:= i-1$   
  Mientras ( $j > 0$ ) y ( $v[j] > \text{elemento a ordenar}$ )  
     $v[j+1]:= v[j]$   
     $j:= j - 1$   
  guardar elemento en  $v[j+1]$ 
```

Si los datos del vector ya vienen ordenados de menor a mayor, el código no va a hacer el corrimiento de ningún elemento (mejor caso). En cambio si los datos vienen ordenados de mayor a menor se tiene que hacer un corrimiento de todos los elementos (peor caso).

Recursión

Es una forma de programar, y de resolver ciertos problemas. Determina que el problema tenga que ser resuelto, resolviendo el mismo problema pero en instancias más pequeñas.

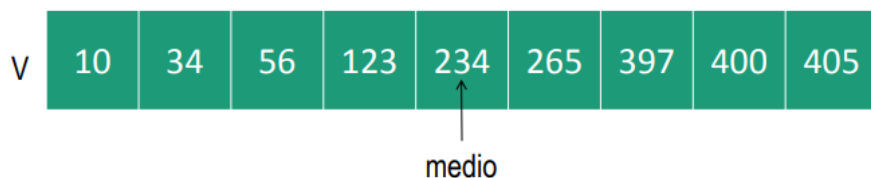
Caso Base: Es una instancia del problema donde no se puede seguir dividiendo el problema. Es una condición que hace que el algoritmo termine. Un algoritmo recursivo sin un caso base no termina nunca.

Caso Recursivo/Autoinvocación: Se tiene que lograr que el problema de alguna manera se reduzca. Se debe garantizar que en un número finito de autoinvocaciones, se alcance el caso base (condición de terminación) para que el algoritmo finalice.

Búsqueda Dicotómica de un valor en el vector

Una forma de buscar un valor en un vector es de manera secuencial, es decir, empezar desde el primer elemento y avanzar uno por uno hasta encontrar el valor buscado o hasta darnos cuenta de que ese valor no existe en el vector. En un vector de pocos elementos parece una tarea sencilla, pero si el vector tiene millones de elementos, la búsqueda secuencial podría ser muy costosa.

Buscar el valor 56 en el vector

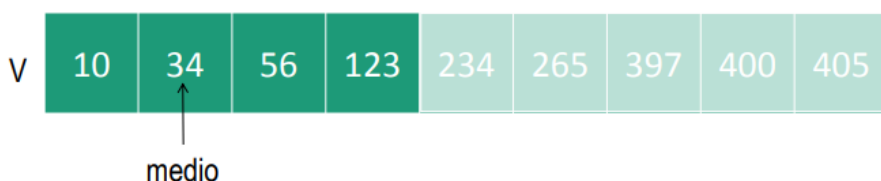


¿Cómo es 56 con respecto a $v[\text{medio}]$?

1. Si es igual, encontré el valor y por lo tanto terminé.
2. Si es menor, busco en la mitad inferior del vector.
3. Si es mayor, busco en la mitad superior del vector.

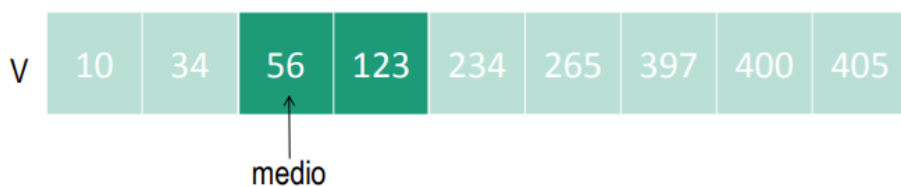
En este caso debemos buscar en la mitad inferior.

Buscar el valor 56 en el vector



¿Cómo es 56 con respecto a $v[\text{medio}]$? 56 es mayor que 34, entonces buscamos en la mitad superior del vector.

Buscar el valor 56 en el vector



¿Cómo es 56 con respecto a $v[\text{medio}]$? Es igual, por lo tanto terminé.

Observamos que la primera vez se trabaja con el vector completo para determinar el punto medio. La siguiente vez, el vector se reduce a la mitad y se determina el punto medio. La siguiente vez, el vector se reduce a la mitad de la mitad y se determina el punto medio. (Recursión).

Buscar (vector, datoABuscar)

```
si el vector "no tiene elementos" entonces
    No lo encontré y termino la búsqueda
sino
    Determinar el punto medio del vector
    Comparar datoABuscar con el contenido del punto medio
    si coincide entonces
        "Lo encontré"
    sino
        si datoABuscar < contenido del punto medio entonces
            Buscar (1era mitad del vector, datoABuscar)
        sino
            Buscar (2da mitad del vector, datoABuscar)
```

El módulo realiza autoinvocaciones a sí mismo (caso recursivo). En cada llamada el tamaño del vector se reduce a la mitad. Hay dos casos bases: cuando el vector no contiene elementos y cuando encuentro el dato a buscar.

A tener en cuenta:

- o Una solución recursiva siempre se va a implementar o bien con un procedimiento o bien con una función.
- o Toda solución recursiva debe tener un caso base (condición de terminación). Y al menos una autoinvocación (caso recursivo) para poder alcanzar el caso base.

Planteo de solución recursiva: ¿Cómo defino el problema en términos de problemas más simples del mismo tipo? ¿Cómo achico el problema en cada llamado recursivo? ¿Qué instancia/s del problema son caso/s base?

Árboles: Se usan para representar relaciones jerárquicas.

Árbol Binario: Es una estructura de datos con las siguientes características:

- o Homogénea: Todos los elementos son del mismo tipo.
- o Dinámica: Su tamaño puede aumentar o disminuir durante la ejecución.
- o No Lineal: Cada elemento puede tener 0, 1, o 2 sucesores.
- o Acceso Secuencial: Para poder llegar a un elemento determinado, tengo que pasar por todos los antecesores.

Propiedades de un árbol binario:

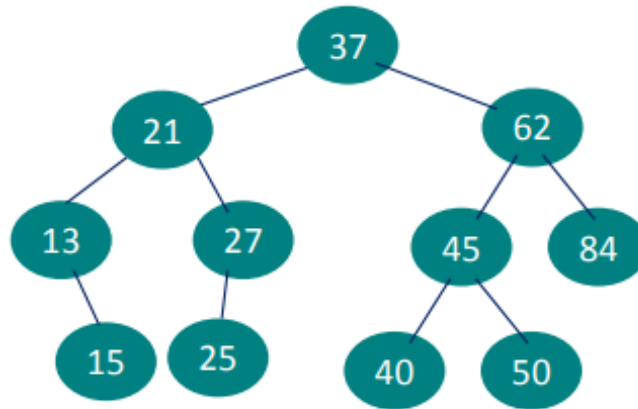
- o Cada elemento del árbol se relaciona con 0, 1, o 2 elementos (hijos).
- o Si el árbol no está vacío, hay un único elemento (raíz) y que no tiene padre (predecesor).
- o Todo otro elemento del árbol posee un único padre y es un descendiente de la raíz (a partir de la raíz puedo llegar al elemento).
- o Raíz: nodo del tope del árbol, es el punto de acceso a la estructura (llevamos un puntero al nodo raíz)
- o Nodo: cada elemento del árbol.
- o Hoja: nodo que no tiene hijos.
- o Nivel de un nodo: número de elementos que hay en el camino desde la raíz hasta el nodo.
- o Subárbol: es el hijo de un nodo con todos sus descendientes.

Árbol binario de búsqueda: hay un orden entre los elementos.

Cada nodo tiene un valor que:

- o Es mayor que el valor de todos los nodos del subárbol izquierdo.
- o Es menor que el valor de todos los nodos del subárbol derecho.

Es decir, si nos paramos en un nodo, a su izquierda se encuentran valores menores y a su derecha valores mayores.



Si estoy buscando un elemento determinado, la operación de búsqueda siempre empieza desde la raíz. Comparo el elemento que estoy buscando con el nodo donde estoy parado. Si es igual, lo encontré, si es menor puede estar en el subárbol izquierdo, y si es mayor en el subárbol derecho.

ABB – Operación de insertar un dato;

- o Al principio el árbol está vacío (puntero a la raíz a = nil).
- o Siempre se inserta a nivel hoja (respetando el criterio de orden).

```
Procedure Insertar(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 (a^.dato>num) then
      insertar(a^.HI,num)
    else
      insertar(a^.HD,num);
    end;
  end;
end;
```

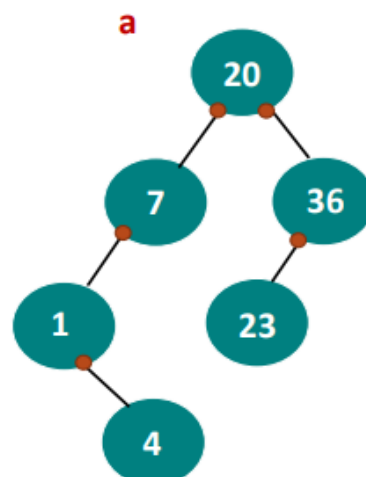


ABB – Recorridos

Los distintos recorridos permiten desplazarse a través de todos los nodos del árbol de tal forma que cada nodo sea visitado una y solo una vez. Existen varios métodos que se diferencian en el orden que se visitan los nodos:

- o Recorrido En – Orden (subárbol izquierdo – raíz – subárbol derecho).
- o Recorrido Pre – Orden (raíz - subárbol izquierdo – subárbol derecho).
- o Recorrido Post – Orden (subárbol izquierdo – subárbol derecho – raíz).

Recorridos Acotados: Se usa en ocasiones determinadas donde necesitamos mostrar los datos que están comprendidos entre dos valores determinados del árbol por ejemplo.

Merge: Consiste en generar una nueva estructura de datos (arreglos, listas) ordenando 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.

Al estar todas las estructuras ordenadas con el mismo criterio, la estructura final resultante que las combina, queda ordenada de la misma forma.

Paradigma Orientado a Objetos (Java)

Paradigmas de Programación: indica la manera de estructurar y organizar las tareas de nuestro programa.

Paradigma Imperativo: El programador le ordena a la computadora una serie de pasos que tiene que seguir para resolver un determinado problema. Esos pasos los organiza en módulos.

Paradigma Orientado a Objetos: Es otra forma de organizar nuestro programa. Varios objetos interactúan entre sí para resolver un problema.

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

Ejemplo:

Triangulo = Objeto.

Características (E.I) = Lado1, Lado2, Lado3, ColorDeLinea, ColorDeRelleno.

Comportamiento (Acciones) = Calcular Área, Calcular Perímetro, etc.

Ante un problema, identificamos los objetos que existen en ese problema y luego debemos identificar cuáles son las características más relevantes y cuáles acciones tiene que saber hacer ese objeto. Los objetos con características y comportamiento similar serán instancia de una misma clase.

Estado Interno: Compuesto por datos/atributos que caracterizan al objeto y relaciones con otros objetos con los cuales colabora. Se implementa 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 constituye la interfaz.

Los métodos pueden recibir datos externos al objeto. Con estos datos y con el estado interno hace algún cálculo y puede devolver o no un resultado.

Encapsulamiento (ocultamiento de información): desde afuera del objeto no voy a conocer como se llaman sus variables de instancia ni como están implementados sus métodos. Solo conozco la interfaz (Todas las acciones que yo le puedo pedir a un objeto que realice)

Todas las acciones que yo le puedo pedir a un objeto que realice forman parte de la interfaz del objeto. Si yo quiero que un determinado objeto realice una acción, le debo enviar un mensaje, con el nombre de la acción o método que quiero que ejecute. El mensaje puede llevar datos (parámetros del método) y puede devolver un dato (resultado del método).

Clase: Una clase describe un conjunto de objetos comunes (del mismo tipo).

Consta de:

- o La declaración de las v.i. que implementan el estado del objeto.
- o 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).

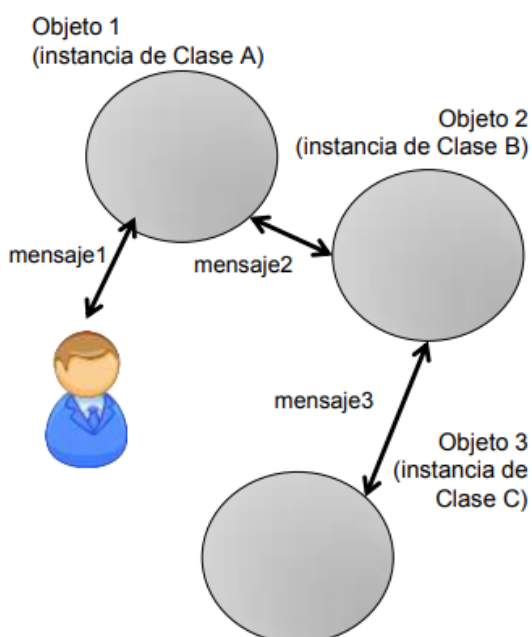
La instanciación se realiza enviando un mensaje de creación a la clase. Ejemplo:

New Triangulo (10, 10,10,"amarillo","violeta")

- o Reserva de espacio para el objeto.
- o Ejecución el código inicializador o constructor (el constructor puede tomar valores pasados en el mensaje de creación. Inicializa el objeto (vi.s) con valores recibidos)
- o Una vez ejecutado el código constructor, se devuelve la referencia al objeto. (ubicación en memoria del objeto)
- o Asociar la referencia a una variable (a través de ella podemos enviarle mensajes al objeto).

Los programas se organizan como una colección de objetos que cooperan entre sí enviándose mensajes.

- o Cada objeto es instancia de una clase.
- o Los objetos se crean a medida que se necesitan.
- o 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.
- o Cuando los objetos ya no son necesarios se borran de la memoria.



Pasos a la hora de realizar un problema:

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

Instanciación de una clase:

- o Declarar variable para mantener la referencia: NombreDeClase miVariable; Triangulo T
- o Enviar a la clase el mensaje de creación y guardar referencia:
miVariable = new NombreDeClase (valores para inicialización);
T = new Triangulo (10, 10, 10, rojo, azul)

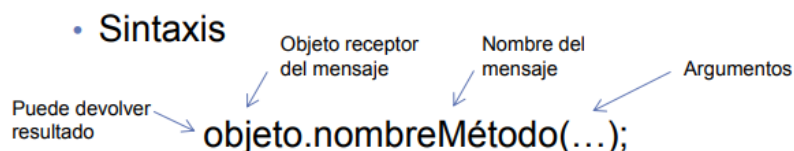
Se puede unir los dos pasos anteriores:

- › NombreDeClase miVariable= new NombreDeClase (...);
- › Triangulo T = new Triangulo (10, 10, 10, rojo, azul).

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

- o Reserva de Memoria. Las variables de instancia se inicializan a valores por defecto o explícito (si hubiese). Ej. Por defecto, todas las variables enteras a 0, las booleanas a F, etc.
- o Ejecución del Constructor (código para inicializar variables de instancia con los valores que enviamos en el mensaje de creación).
- o Asignación de la referencia a la variable.
Cuando una referencia vale null, significa que esa referencia no está apuntando a ningún objeto válido.
String saludo1 = new String ("hola");
String saludo2 = new String ("hola");
Asignación: copia referencias. saludo1 = saludo2 (se copia en saludo 1 la referencia de saludo2, ahora ambos apuntan a un mismo objeto)
Recolector de basura: libera memoria de objetos no referenciados.
Comparación de objetos con == y !=, comparan referencias
System.out.println(saludo1==saludo2);
Comparación del contenido de objetos por ejemplo, enviar mensaje equals al objeto, pasando como argumento el objeto a comparar: saludo2.equals (saludo1) compara el string contenido en saludo2 con el string contenido en saludo1 y devuelve true o false.

Envío de Mensaje:

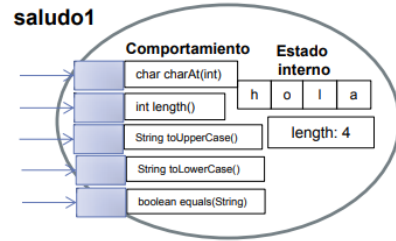


• Ejemplo

```
public class Demo01EnvioMensaje {
    public static void main(String[] args) {
        String saludo1 = new String("hola");
        System.out.println(saludo1.length()); //Imprime 4
        System.out.println(saludo1.charAt(0)); //Imprime h
        System.out.println(saludo1.toUpperCase().equals("HOLA")); //Imprime true
    }
}
```

1) Envío de msg toUpperCase a saludo1
Devuelve un objeto String

2) Envío msg equals al objeto retornado por
saludo1.toUpperCase()



Regla de precedencia:
los mensajes se
ejecutan de izq a der

Métodos getters y setters: métodos que permiten almacenar o modificar un valor de un atributo para una clase (setters) y permiten obtener el valor almacenado en cada uno de los atributos (getters).

Declaración de clases:

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

Declaración del estado

Estado interno:

- o Datos de tipos primitivos: double precio;
- o Referencias a otros objetos: String título;

Anteponer a la declaración la palabra private para lograr encapsulamiento (ocultamiento de la información). Las v.i.s. privadas pueden ser accedidas sólo dentro de la clase que las declara.

En la declaración del dato se puede dar un valor inicial (inicialización explícita). Los datos correspondientes al estado toman un valor por defecto cuando no se inician explícitamente.

Declaración del comportamiento:

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

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

Declaración del constructor: Inicializa el estado interno de un objeto al momento de instanciarlo.

Objetivo: inicialización de las v.i.s.

Si la clase no declara ningún constructor, Java incluye uno sin parámetros y sin código (constructor nulo).

Instanciación de objeto:

```
NombreClase objeto= new NombreClase (lista de parámetros actuales);
```

Sobrecarga: puede haber varios constructores para la clase.

Java identifica cuál está siendo invocado por el número y tipo de sus parámetros.

Referencia This: Es una referencia hacia la propia instancia. Poniendo this.nombreMétodo (parámetros) El objeto que está ejecutando (this) se enviará un mensaje a sí mismo. El método a ejecutar se busca a partir de la clase de la cual es instancia el objeto.

Herencia

La Herencia es el mecanismo que permite que la clase herede características y comportamiento (atributos y métodos) de otra clase (clase padre o superclase). A su vez, la clase define características y comportamiento propio.

Ventaja: reutilización de código, evita la replicación de características y comportamiento común.

Ejemplo: se define lo común en una clase figura (superclase) y las clases triángulo, círculo y cuadrado lo heredan (subclases).

Las subclases heredan atributos y métodos de la figura, definen atributos y métodos propios, y definen constructores.

Polimorfismo: cualidad que tienen objetos de distintas clases de responder al mismo mensaje de manera distinta.

Ejemplo: Cuando instancio en el Main un objeto;

```
Cuadrado C = new Cuadrado (10,"rojo","negro")
```

Ese objeto que cree, cuenta con las variables de instancias definidas en la clase cuadrado y con las heredadas de la superclase. Le puedo mandar los mensajes que están definidos en su clase, y también todos los mensajes que herede.

```
System.out.println (C.calcularArea ());
```

Siempre la búsqueda del método empieza en la clase de la cual es instancia el objeto. Es decir, que como C es instancia de la clase cuadrado, ese mensaje lo empieza a buscar desde la clase cuadrado. Como el método está implementado en esa clase, se va a ejecutar y después vuelve el control al programa principal.

```
System.out.println (C.getColorRelleno ());
```

El método a ejecutar se empieza a buscar de la clase a la que el objeto es instancia (clase cuadrado). El método no está definido en esta clase, entonces lo busca en la clase superior (figura), el método getColorRelleno (); se encuentra definido en la clase figura, entonces se ejecuta y después vuelve el control al programa principal.

Palabra clave extends.

```
Public class NombreSubclase extends nombreSuperclase {  
  
/* Definir atributos propios */  
  
/* Definir constructores propios */  
  
/* Definir métodos propios */}
```

La subclase hereda:

- o Atributos declarados en la superclase. Como son privados son accesibles sólo en métodos de la clase que los declaró. En la subclase accederlos a través de getters y setters heredados.
- o Métodos de instancia declarados en la superclase.

La subclase puede declarar: Atributos / métodos / constructores propios

Clase abstracta: es una clase que no puede ser instanciada (no se pueden crear objetos a partir de ella).

Uso: define características y comportamiento común para un conjunto de clases (subclases).

Puede definir **métodos abstractos** (sin implementación) que deben ser implementados por las subclases. Se definen para obligar a todas las subclases a que implementen ese método.

Ejemplos: La clase Figura es abstracta. Figura puede declarar métodos abstractos calcularArea /calcularPerimetro. Declarándolos como métodos abstractos en la clase figura, cuando el programador vaya a hacer, por ejemplo, la clase cuadrado, como obligación va a tener que implementar ese método.

Declaración de clase abstracta: anteponer abstract a la palabra class;

```
public abstract class NombreClase
```

Declaración de método abstracto: sólo se pone el encabezado del método (sin código) anteponiendo abstract al tipo de retorno;

```
public abstract TipoRetorno nombreMetodo (lista parámetros formales);
```

```
public abstract class Figura{  
    ....  
    public abstract double calcularArea();  
    public abstract double calcularPerimetro();  
}
```

Super (...) Invoco al constructor de la superclase. Al declarar un constructor en la superclase esta invocación debe ir como primera línea.

Con super.método () empezamos a buscar a partir de la clase superior a la clase instancia.

Blinding dinámico: se determina en tiempo de ejecución el método a ejecutar para responder a un mensaje.

Ventaja: Código genérico, reusable.

Programación Concurrente (R-Info)

Concurrencia: capacidad de ejecutar múltiples actividades de manera simultánea.

El 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:

Los procesos concurrentes pueden tener la necesidad de comunicarse, y la necesidad de sincronizarse.

Comunicación:

- o **Memoria compartida** (bloquear-desbloquear): Los procesos intercambian información sobre la memoria compartida o actúan cordialmente sobre datos residentes en ella.
Los procesos no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria.
- o **Envío de mensajes (enviar-recibir)**: El lenguaje debe proveer un protocolo adecuado (origen, destino, contenido). Los procesos deben saber cuándo tienen mensajes para leer y cuando deben transmitir mensajes.

Pasaje de mensajes:

- o **Asincrónico**: El proceso que recibe/envía el mensaje, no espera que se dé la comunicación para continuar su ejecución.
- o **Sincrónico**: El proceso que envía/ recibe el mensaje, sí espera a que se dé la comunicación para continuar su ejecución.

Pasaje de mensajes en CMRE (misma cantidad de envíos que de recepción):

- › Envío de mensajes (Asincrónico): un robot después de enviar un mensaje puede continuar su ejecución.
- › Recibir mensajes (Sincrónico): un robot que está esperando recibir un mensaje no puede continuar su ejecución hasta que el otro robot no le haya mandado el mensaje.

Comunicación en CMRE:

- › Memoria compartida:
 - Bloquear: dado un recurso disponible el programador bloquea dicho recurso para que ningún otro proceso pueda acceder.
 - Desbloquear: dado un recurso bloqueado el programador libera dicho recurso para que cualquier proceso pueda bloquearlo.