https://www.eclipse.org/ide/

@Override Para sobreescribir métodos sobre los que se hereda.

type	*set of values*	*common operators*	*sample literal values*
`int`	integers	`+ - * / %`	`99 12 2147483647`
`double`	floating-point numbers	`+ - * /`	`3.14 2.5 6.022e23`
`boolean`	boolean values	`&& || !`	`true false`
`char`	characters		`'A' '1' '%' '\n'`
`String`	sequences of characters	`+`	`"AB" "Hello" "2.5"`

```
public class Polymorphism {
   public static int add(int[] inAr) {
      float total = 0;
      for (int i = 0; i < inAr.length; i++) {
         total = total + inAr[i];
      }
      return ((int)total);
   }

   public static int add(char[] inAr) {
      return 0;
   }

   public static void main(String[] args) {
      int[] intArray = {1, 2, 3, 4, 5};
      char[] charArray = {'a', 'b', 'c'};
      add(intArray);
      add(charArray);
   }
}
```

```
public E desencolar () +
E dato=principioC.dato;
talla--;
principioC=principioC.siguiente;
if (talla==0) finalC=null;
return dato;
/** comprueba si una Pila
```

Cola<Integer> q = new ArrayCola<Integer>();: Se crea una instancia de una cola (queue) que almacena objetos de tipo Integer. La implementación utilizada es ArrayCola, que es una cola respaldada por un array.

Elementos de tipo E

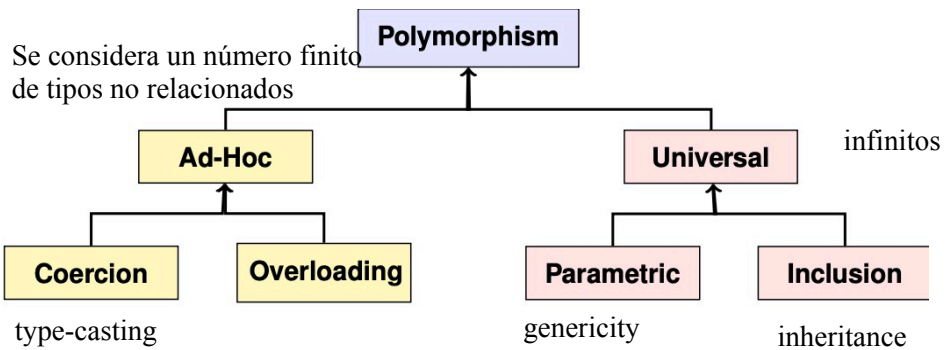
```
public class LEGListaConPI<E> implements ListaConPI<E> {
    public void insertar(E e) {
    NodoLEG<E> nuevo = new NodoLEG<E>(e, ant.siguiente);
    ant.siguiente = nuevo;
    if (nuevo.siguiente == null) ult = nuevo;
    ant = ant.siguiente;
    talla++;
```

```
public class LEGListaConPI<E> implements ListaConPI<E> {

    protected NodoLEG<E> pri, ant, ult;
    protected int talla;
// se implementa 'el anterior al punto de interés al ser más útil
    /** construye una Lista Con PI vacia **/
    public LEGListaConPI() {
        pri = ult = ant = new NodoLEG<E>(null);
        talla = 0;
    }

    /** inserta e en una Lista antes del Elemento que ocupa su PI,
     * que permanece inalterado.
     *
     * @param e Elemento a insertar.
     **/
    public void insertar(E e) {
        NodoLEG<E> nuevo = new NodoLEG<E>(e, ant.siguiente);
        ant.siguiente = nuevo; //PI
        if (nuevo.siguiente == null) ult = nuevo;
        ant = ant.siguiente;
        talla++;
    }
```

El tipo de elemento es E pero nos referimos a el como e

Polimorfismo -> permite el uso de valores de distintos tipos con una interface uniforme

Una función puede ser polifórmica respecto a algunos de sus argumentos (+ funciona para enteros, reales, etc…)
Un tipo de dato puede ser polimórfico respecto al tipo de sus componentes (una lista de distintos tipos de elementos)



Overloading -> muchas funciones tienen el mismo nombre.
(+) :: int -> int -> int

(+) :: int -> float -> float

```
if (tope+1==elArray.length) duplicarArray();// si está llena se aumenta su capacidad al doble
```

```
// duplica el tamagno actual de un array
@SuppressWarnings("unchecked")
protected void duplicarArray() {
    E[] nuevoArray = (E[]) new Object[elArray.length * 2];
    System.arraycopy(elArray, 0, nuevoArray, 0, tope);
    elArray = nuevoArray;
}
```

```
Map<String, Integer> map = new HashMap<String, Integer>();

map.put("valor1", 15);
map.put("valor2", 20);
map.put("valor3", 1000);
map.put("valor4", 1500);
map.put("valor5", 2);

int valor1 = map.get("valor1");

// para saber el número más alto se debe recorrer el mapa
int valorMax = -1;

for (Map.Entry<String, Integer> entry : map.entrySet()) {
    final int valorActual = entry.getValue();

    if (valorActual > valorMax)
        valorMax = valorActual;
}

System.out.println("Valor máximo: " + valorMax);
```

```
tipoDeDato[] nombreDelArreglo = new
tipoDeDato[tamaño];
```

```
elArray = (E[]) new Object[CAPACIDAD_POR_DEFECTO];
```