

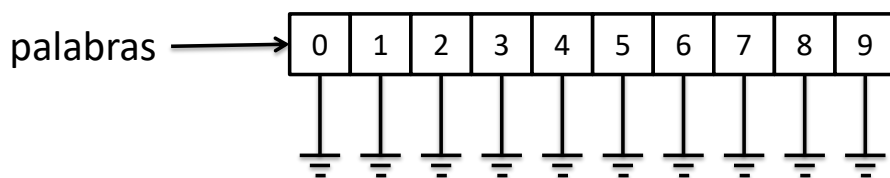
# Contenedores (I)


Metodologia i programació orientada  
a Objectes

## Guardar colecciones de objetos

- Muchas veces tenemos que guardar colecciones de objetos o tipos básicos → Vectores

```
String palabras[] = new String[10];
```

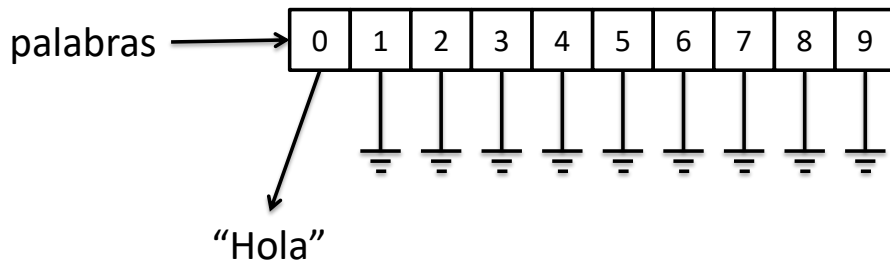


 == null

# Guardar colecciones de objetos

- Muchas veces tenemos que guardar colecciones de objetos o tipos básicos → Vectores

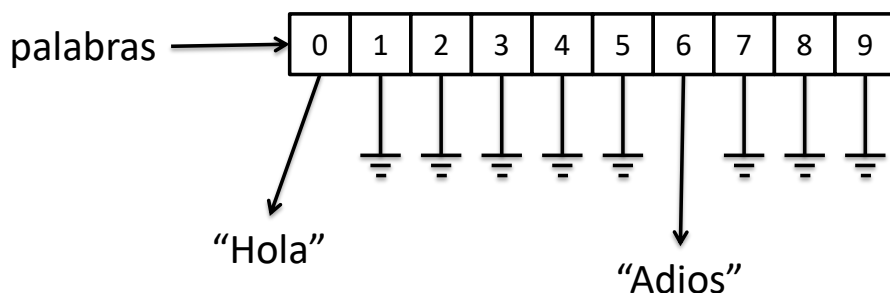
```
String palabras[] = new String[10];  
palabras[0] = "Hola";
```



# Guardar colecciones de objetos

- Muchas veces tenemos que guardar colecciones de objetos o tipos básicos → Vectores

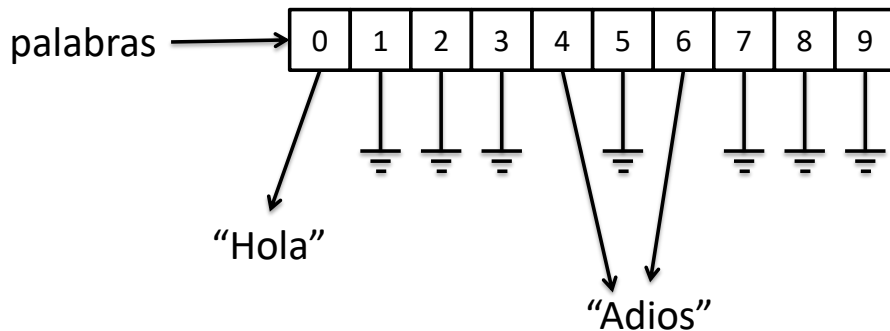
```
String palabras[] = new String[10];  
palabras[0] = "Hola";  
palabras[6] = "Adios";
```



# Guardar colecciones de objetos

- Muchas veces tenemos que guardar colecciones de objetos o tipos básicos → Vectores

```
String palabras[] = new String[10];  
palabras[0] = "Hola";  
palabras[6] = "Adios";  
palabras[4] = palabras[6];
```



## Inconvenientes de los vectores

- Los objetos que se guardan en él deben estar asociados a un índice/posición del vector.
- Hay referencias a null: posibilidad de errores.
- Tamaño fijo
  - Podemos llegar a llenarlo y quedarnos sin espacio para añadir más objetos.
  - Si creamos un vector demasiado grande, desperdiciaremos mucha memoria.
- Usar vectores como listas de objetos es complejo
  - Inserción
  - Eliminación
  - Búsqueda
  - Contabilización
  - Dimensionado

# Solución: contenedores

- Son clases que Java nos proporciona para tratar de manera fácil las colecciones de objetos.
  - Esconden la complejidad que llevan dentro.
  - Proporcionan un conjunto de sencillos métodos para insertar, borrar, buscar, leer...
- En este curso trataremos tres tipos, según las funciones que proporcionen
  - Conjuntos
  - Listas
  - Diccionarios

## Conjuntos

- Contenedores de objetos sin orden alguno.
- Un mismo objeto no puede estar repetido en un mismo contenedor.
- Permiten insertar objetos, pero no leerlos directamente
  - Uso de iteradores para “recorrer” el conjunto
- Clases que lo implementan
  - HashSet, TreeSet...
  - Estas clases solo se diferencian en su estructura interna, pero se usan exactamente igual ("interfaz" Set).
- Ejemplos

```
Set<String> conjunto1 = new HashSet<>();
conjunto1.add("Hola");
conjunto1.add("Adios");
Set<Complex> conjunto2 = new TreeSet<>();
conjunto2.add(new Complex(1,1));
```

# Conjuntos: utilización

- Tipo de datos **paramétrico**: se debe especificar entre < y > la clase de los objetos que guardará
  - Si no se especifica, asumirá que se puede guardar cualquier tipo de datos.
- Métodos comunes
  - `add(objeto)` → añade un nuevo objeto al conjunto.
  - `contains(objeto)` → devuelve “true” si el objeto que se le pasa como parámetro ya está contenido
  - `size()` → Retorna el numero de objetos
  - `remove(objeto)` → elimina el objeto especificado
  - `clear()` → elimina todos los elementos del conjunto
  - `iterator()` → devuelve un iterador al objeto

Más info: <http://download.oracle.com/javase/6/docs/api/java/util/Set.html>

## Estructura “for each”

- El 99% de las veces que querramos recorrer un conjunto (también sirve para Arrays y servirá para Listas) usaremos la estructura de control “for each”

```
Set<String> frases = new HashSet<>();
```

```
frases.add("Perro ladrador, poco mordedor");  
frases.add("Más vale pájaro en mano que ciento volando");  
frases.add("No por mucho madrugar amanece más temprano");
```

```
for(String f : frases) {  
    System.out.println(f);  
}
```

# Iteradores

- Son unos objetos que van “saltando” por todos los elementos de una lista o conjunto, de tal manera que podemos acceder a todos ellos.

```
Set<String> hs = ....  
.....  
Iterator<String> it = hs.iterator();  
while(it.hasNext()) {  
    String texto = it.next();  
    // usar “texto” según se requiera  
}
```

## Ejemplo de Conjuntos + iteradores (I)

```
public class Alumno {  
    private String nombre;  
    private String dni;  
  
    public Alumno(String nombre, String dni) {  
        this.nombre = nombre;  
        this.dni = dni;  
    }  
    public String toString() {  
        return nombre + “. DNI: “ + dni;  
    }  
}
```

## Ejemplo de Conjuntos + iteradores (II)

```
public class Aula {  
    private Set<Alumno> matriculados = new HashSet<>();  
  
    public void matricula(Alumno alumno) {  
        matriculados.add(alumno);  
    }  
  
    public void listadoEnPantalla() {  
        Iterator<Alumno> iterador = matriculados.iterator();  
        while(iterador.hasNext()) {  
            Alumno al = iterador.next();  
            System.out.println(al.toString());  
        }  
    }  
}
```

## Ejemplo de Conjuntos + *for each*

```
public class Aula {  
    private Set<Alumno> matriculados = new HashSet<>();  
  
    public void matricula(Alumno alumno) {  
        matriculados.add(alumno);  
    }  
  
    public void listadoEnPantalla() {  
        for(Alumno al : matriculados) {  
            System.out.println(al.toString());  
        }  
    }  
}
```