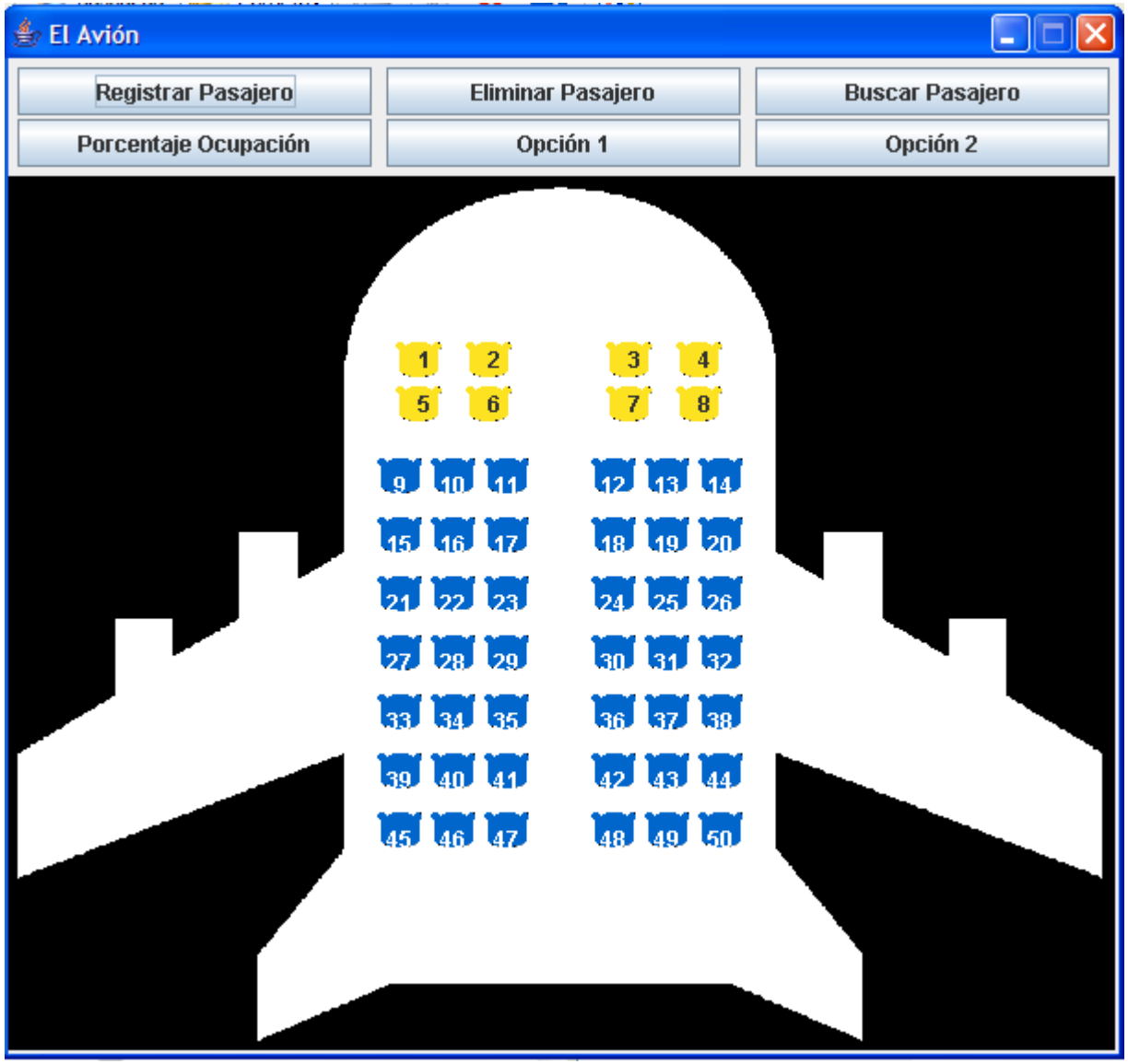
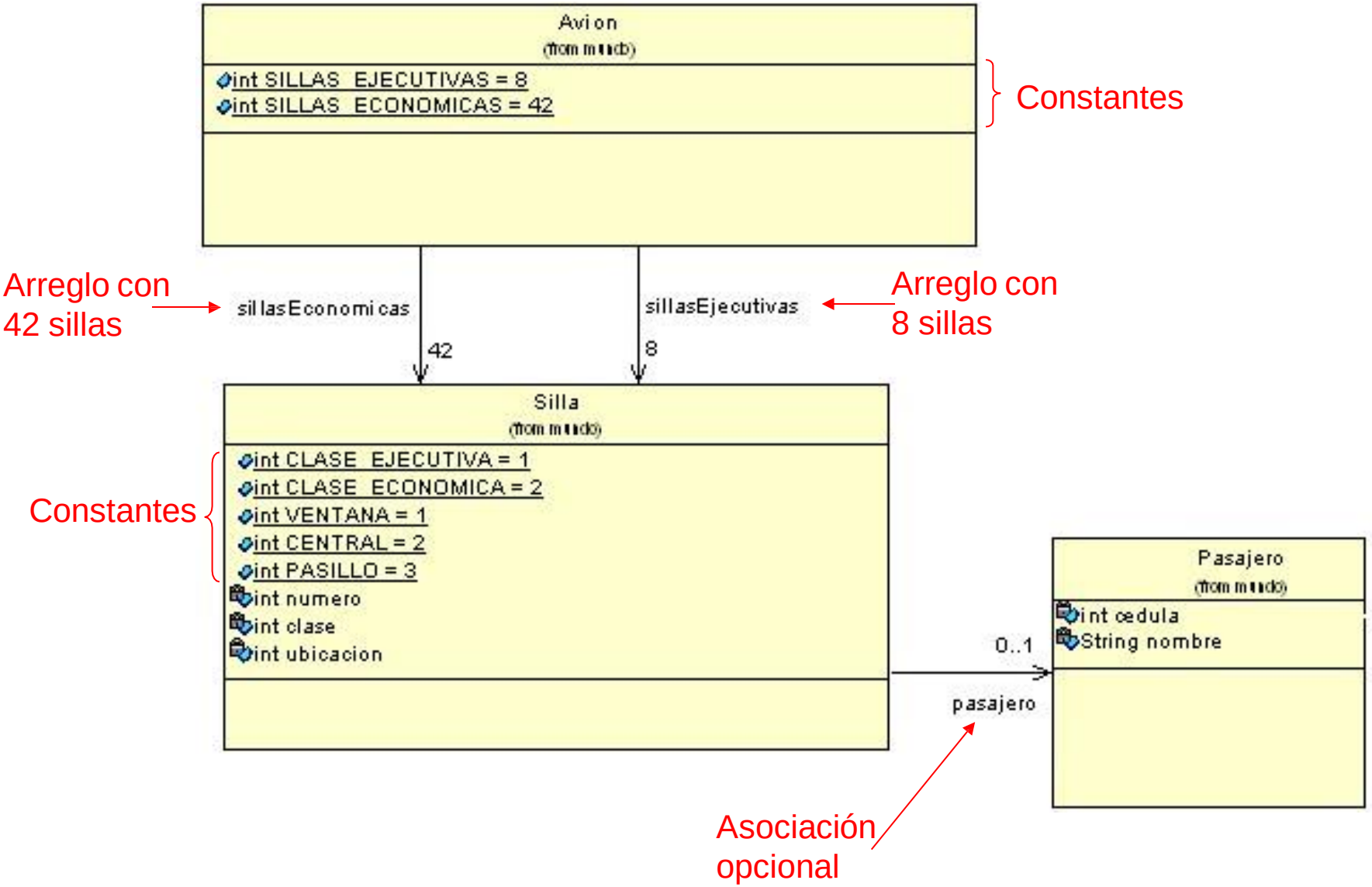
Arreglos de objetos y ArrayLists

Caso de estudio: El Avión

Interfaz usuario



Modelo del Mundo



La clase Avion

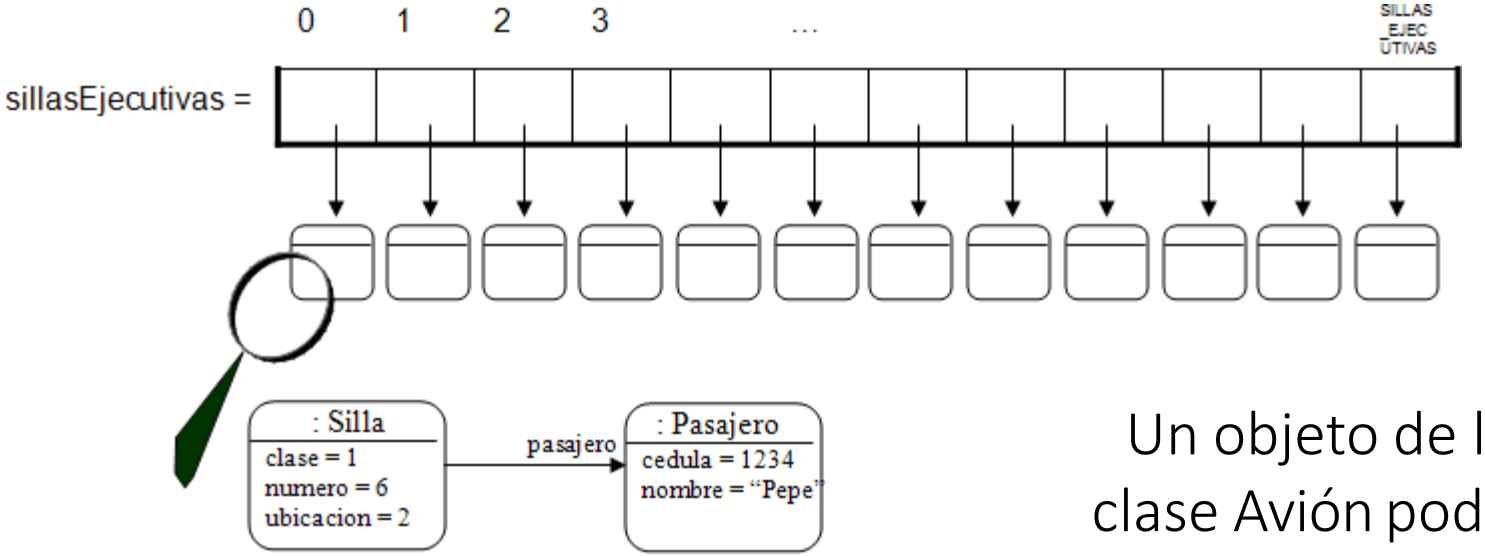
```
public class Avion
{
    //Constantes
     public final static int SILLAS_EJECUTIVAS = 8;
     public final static int SILLAS_ECONOMICAS = 42;

    //Atributos
     private Silla[] sillasEjecutivas;
     private Silla[] sillasEconomicas;
}
```

La clase Silla

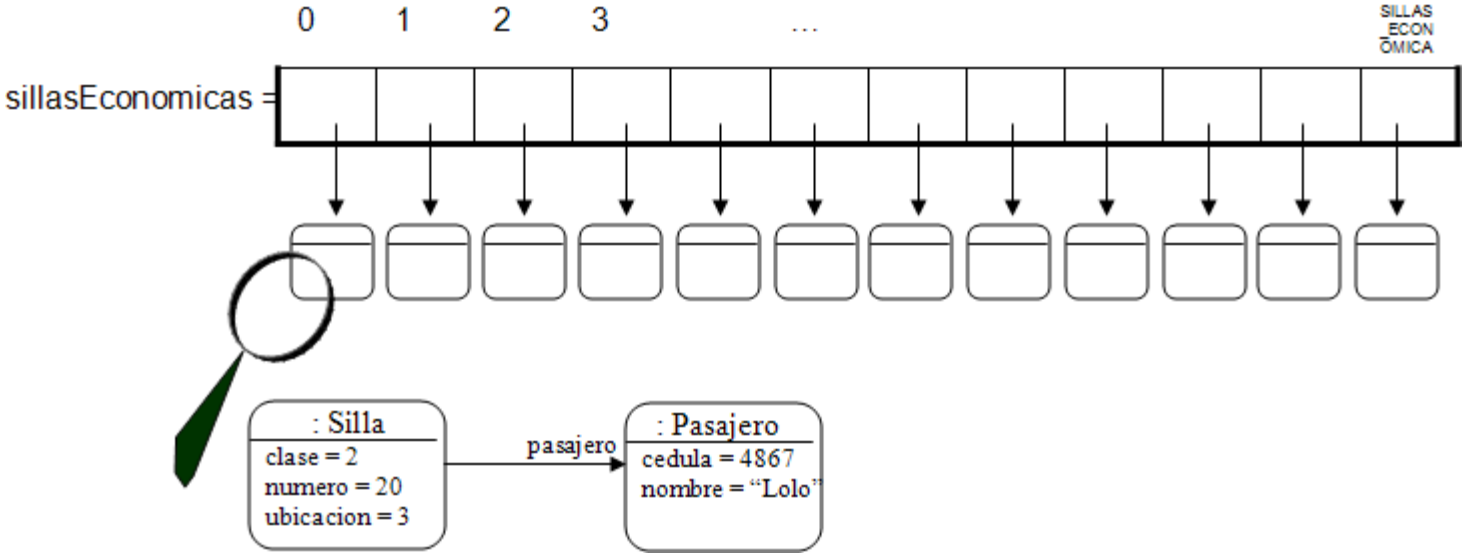
```
public class Silla
{
    // Constantes
   public final static int CLASE_EJECUTIVA = 1;
   public final static int CLASE_ECONOMICA = 2;
   public final static int VENTANA = 1;
   public final static int CENTRAL = 2;
   public final static int PASILLO = 3;

    // Atributos
    private int numero;
    private int clase;
    private int ubicacion;
    private Pasajero pasajero;
}
```

La clase Pasajero

```
public class Pasajero
{
    // Atributos
    private int cedula;
    private String nombre;
}
```



Un objeto de la clase Avión podría verse así



Tarea No.6 – Hacer la declaración de la clase Pasajero

```
public class Pasajero
{
    //---------------------------------
    // Atributos
    //---------------------------------




    //---------------------------------
    // Constructor
    //---------------------------------
    public Pasajero( int unaCedula, String unNombre )
    {



    }
```

Tarea No.6 – Hacer la declaración de la clase Pasajero

```
public class Pasajero
{
    //---------------------------------
    // Métodos
    //---------------------------------
    public int darCedula( )
    {



    }

    public String darNombre( )
    {



    }
}
```

Tarea No.7 – Completar la declaración de la clase Silla

```
public class Silla
{
    //----------------------------------------
    // Constantes
    //----------------------------------------
    public final static int CLASE_EJECUTIVA = 1;
    public final static int CLASE_ECONOMICA = 2;
    public final static int VENTANA = 1;



    //----------------------------------------
    // Atributos
    //----------------------------------------
    private int numero;
    private int clase;
    private int ubicacion;
    private Pasajero pasajero;
```

Tarea No.7 – Completar la declaración de la clase Silla

```
public class Silla
{
    public Silla( int num, int clas, int ubica )
    {
        numero = num;
        clase = clas;
        ubicacion = ubica;
        pasajero = null;
    }

    public void asignarPasajero( Pasajero pas )
    {




    }
```

Tarea No.7 – Completar la declaración de la clase Silla

```
public void desasignarSilla ( )
{



}


public boolean sillaAsignada( )
{







}

public int darNumero( )
{



}
```

Tarea No.7 – Completar la declaración de la clase Silla

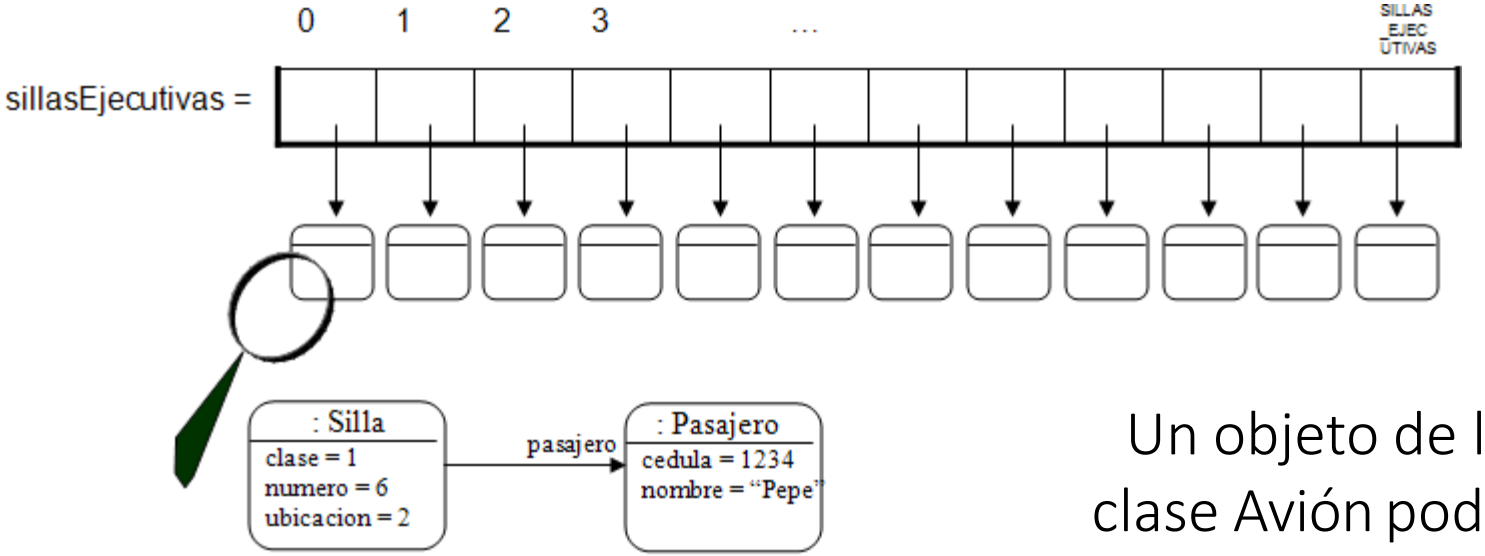
```
 public int darClase( )
 {


 }


pubic int darUbicacion( )
{



}

 public Pasajero darPasajero( )
 {


 }
```

La clase Avion

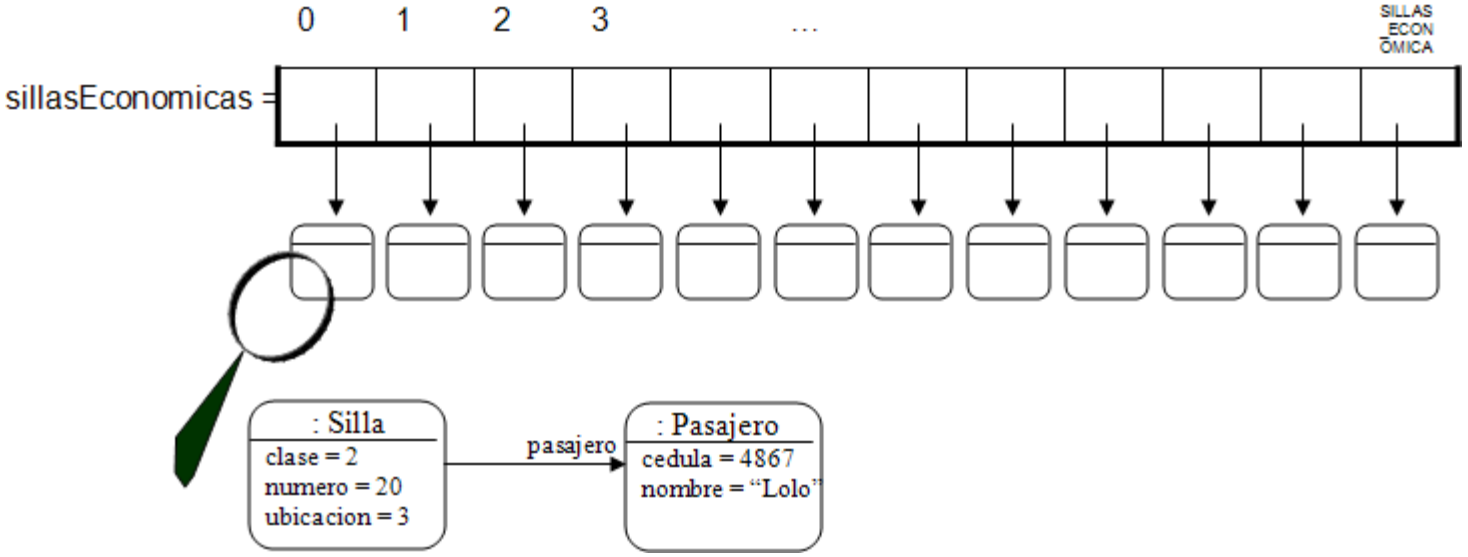
```
public class Avion
{
    //Constantes
     public final static int SILLAS_EJECUTIVAS = 8;
     public final static int SILLAS_ECONOMICAS = 42;

    //Atributos
     private Silla[] sillasEjecutivas;
     private Silla[] sillasEconomicas;
}
```

Método constructor de la clase Avion

```
public Avion( )
   {
      // Crea las sillas ejecutivas y economicas
      sillasEjecutivas = new Silla[SILLAS_EJECUTIVAS];
      sillasEconomicas = new Silla[SILLAS_ECONOMICAS];

      sillasEjecutivas[ 0 ] = new Silla( 1, Silla.CLASE_EJECUTIVA, Silla.VENTANA );
      sillasEjecutivas[ 1 ] = new Silla( 2, Silla.CLASE_EJECUTIVA, Silla.PASILLO );
      sillasEjecutivas[ 2 ] = new Silla( 3, Silla.CLASE_EJECUTIVA, Silla.PASILLO );
      sillasEjecutivas[ 3 ] = new Silla( 4, Silla.CLASE_EJECUTIVA, Silla.VENTANA );
      sillasEjecutivas[ 4 ] = new Silla( 5, Silla.CLASE_EJECUTIVA, Silla.VENTANA );
      sillasEjecutivas[ 5 ] = new Silla( 6, Silla.CLASE_EJECUTIVA, Silla.PASILLO );
      sillasEjecutivas[ 6 ] = new Silla( 7, Silla.CLASE_EJECUTIVA, Silla.PASILLO );
      sillasEjecutivas[ 7 ] = new Silla( 8, Silla.CLASE_EJECUTIVA, Silla.VENTANA );

      sillasEconomicas[ 0 ] = new Silla( 9, Silla.CLASE_ECONOMICA, Silla.PASILLO );
      sillasEconomicas[ 1 ] = new Silla( 10, Silla.CLASE_ECONOMICA, Silla.CENTRAL );
      …
}
```



Un objeto de la clase Avión podría verse así



Preguntas frecuentes…

- Cómo se hace el llamado a un método de un objeto que está en el arreglo?

  R// sillasEjecutivas[3].sillaAsignada( )

- Los objetos que están en el arreglo se pueden guardar en una variable ?

  R// SI.

  Silla sillaTemp = sillasEjecutivas [0];

Volvamos al arreglo de notas …

Contar cuántas notas están por encima de 3.0 (o sea cuántos estudiantes pasaron)

```
public int cuantosPasaron ( )
{
    int contador = 0;

    for ( int i = 0;  i < notas.length; i++ )
    {
        if ( notas [ i ] >= 3.0 )
            contador++;
    }

    return contador;
}
```

FACIL !!!

Pasemos al avión …

Contar cuántas sillas ejecutivas están libres (o sea no asignadas)

- En qué se parece al caso anterior (cuantosPasaron) ?

- En qué se diferencia?
  - Cuál es el arreglo en este problema ?

  - Qué estamos buscando ?

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{




}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{




}
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
      int contador = 0;




}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{




}
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
      int contador = 0;




}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
      int contador = 0;




}
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
    int contador = 0;

    for ( int i = 0; i < notas.length; i++ )









}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
    int contador = 0;









}
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
    int contador = 0;

    for ( int i = 0; i < notas.length; i++ )









}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
    int contador = 0;

    for ( int i = 0; i < sillasEjecutivas.length; i++ )









}
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
    int contador = 0;

    for ( int i = 0; i < notas.length; i++ )
    {
      if ( notas [ i ] >= 3.0 )
        contador++;
    }




}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
    int contador = 0;

    for ( int i = 0; i < sillasEjecutivas.length; i++ )




}
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
      int contador = 0;

      for ( int i = 0; i < notas.length; i++ )
      {
         if ( notas [ i ] >= 3.0 )
             contador++;
      }




}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
      int contador = 0;

       for ( int i = 0; i < sillasEjecutivas.length; i++ )
       {
          if ( sillasEjecutivas[ i ].sillaAsignada( ) == false )
              contador++;
       }




 }
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
    int contador = 0;

    for ( int i = 0; i < notas.length; i++ )
    {
      if ( notas [ i ] >= 3.0 )
          contador++;
    }

   return contador;



}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
    int contador = 0;

    for ( int i = 0; i < sillasEjecutivas.length; i++ )
    {
      if ( sillasEjecutivas[ i ].sillaAsignada( ) == false )
          contador++;
    }



}
```

Transformación …

Caso de las notas del curso (método de la clase Curso)

```
public int cuantosPasaron ( )
{
    int contador = 0;

    for ( int i = 0; i < notas.length; i++ )
    {
       if ( notas [ i ] >= 3.0 )
          contador++;
    }

   return contador;



}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
    int contador = 0;

    for ( int i = 0; i < sillasEjecutivas.length; i++ )
    {
       if ( sillasEjecutivas[ i ].sillaAsignada( ) == false )
          contador++;
    }

    return contador;



}
```

Ahora con variables auxiliares …

- Caso de las notas del curso
- (método de la clase Curso)

```
public int cuantosPasaron ( )
{
    int contador = 0;
    double notaAux;

    for ( int i = 0; i < notas.length; i++ )
    {
        notaAux = notas[ i ];
        if ( notaAux >= 3.0 )
            contador++;
    }

    return contador;

}
```

Caso avión (método de la clase Avion)

```
public int cuantasSillasEjecutivasLibres ( )
{
    int contador = 0;
    Silla sillaAux;

    for ( int i = 0; i < sillasEjecutivas.length; i++ )
    {
        sillaAux = sillasEjecutivas[ i ];
        if ( sillaAux.sillaAsignada( ) == false )
            contador++;
    }

    return contador;

}
```

En la clase AVION:

Contar cuántas sillas de clase económica tienen ubicación VENTANA

```
public int contarVentanasEconomica ( )
{




 }
```

En la clase AVION:

Contar cuántas sillas de clase económica tienen ubicación VENTANA

```
public int contarVentanasEconomica ( )
{
    int contador = 0;
     Silla sillaAux;

    for ( int i = 0;  i < sillasEconomicas.length; i++ )
     {
        sillaAux = sillasEconomicas[ i ];
        if ( sillaAux.darUbicacion( ) == Silla.VENTANA)
            contador++;
     }

    return contador;
```

En la clase AVION:

Contar cuántas sillas de clase económica tienen una ubicación dada (que se recibe como parámetro)

```
public int contarSillasEconomicasEnUbicacion ( int ubi )
    {




     }
```

En la clase AVION:

Contar cuántas sillas de clase económica tienen una ubicación dada (que se recibe como parámetro)

```
public int contarSillasEconomicasEnUbicacion ( int ubi )
    {
    int contador = 0;
     Silla sillaAux;

    for ( int i = 0;  i < sillasEconomicas.length; i++ )
     {
        sillaAux = sillasEconomicas[ i ];
        if ( sillaAux.darUbicacion( ) == ubi )
            contador++;
     }

    return contador;
```

En la clase AVION:

Informar (verdadero o falso) si un pasajero dado se encuentra en la clase ejecutiva del avión (la cédula del pasajero se recibe como parámetro)

public boolean existePasajeroEjecutivo (int ced)

AYUDAS:

1. Es un patrón de recorrido parcial
2. Utilice el método darPasajero( ) de la clase Silla para obtener el pasajero de una silla
3. Recuerde que una silla puede o no estar asignada (una silla está asignada cuando su pasajero existe. Una silla NO está asignada cuando su pasajero no existe (es null))
4. Utilice el método darCedula( ) de la clase Pasajero, para obtener el número de cédula del pasajero de una silla

En la clase AVION:

Informar (verdadero o falso) si un pasajero dado se encuentra en la clase ejecutiva del avión (la cédula del pasajero se recibe como parámetro)

```
public boolean existePasajeroEjecutivo( int ced )
{




}
```

En la clase AVION:

Asignar al pasajero que se recibe como parámetro una silla en clase económica que esté libre (en la ubicación pedida). Si el proceso tiene éxito, el método retorna verdadero. En caso contrario, retorna falso

```
public boolean asignarSillaEconomica( int ubicación, Pasajero
    pasajero)
{









}
```

En la clase AVION:

Anular la reserva en clase ejecutiva que ten´´ia el pasajero con la cédula dada (la cédula del pasajero se recibe como parámetro). Retorna verdadero si el proceso tiene éxito.

```
public boolean anularReservaEjecutivo( int ced )
{




}
```

La clase TiendaLibros

```
public class TiendaLibros
{
    //Atributos
     private ArrayList catalogo;
     private CarroCompras carrito;
}
```

Contenedoras de tamaño variable o vectores (ArrayList)

- Caso de estudio: Tienda de Libros

Contenedoras de tamaño variable o vectores

- Se usan cuando NO conocemos el tamaño de la contenedora (cantidad de elementos que se van a guardar)

- Para implementarlas en Java es necesario utilizar una clase de Java que se llama ArrayList (se encuentra en el paquete java.util)

Diferencias entre arreglos y vectores

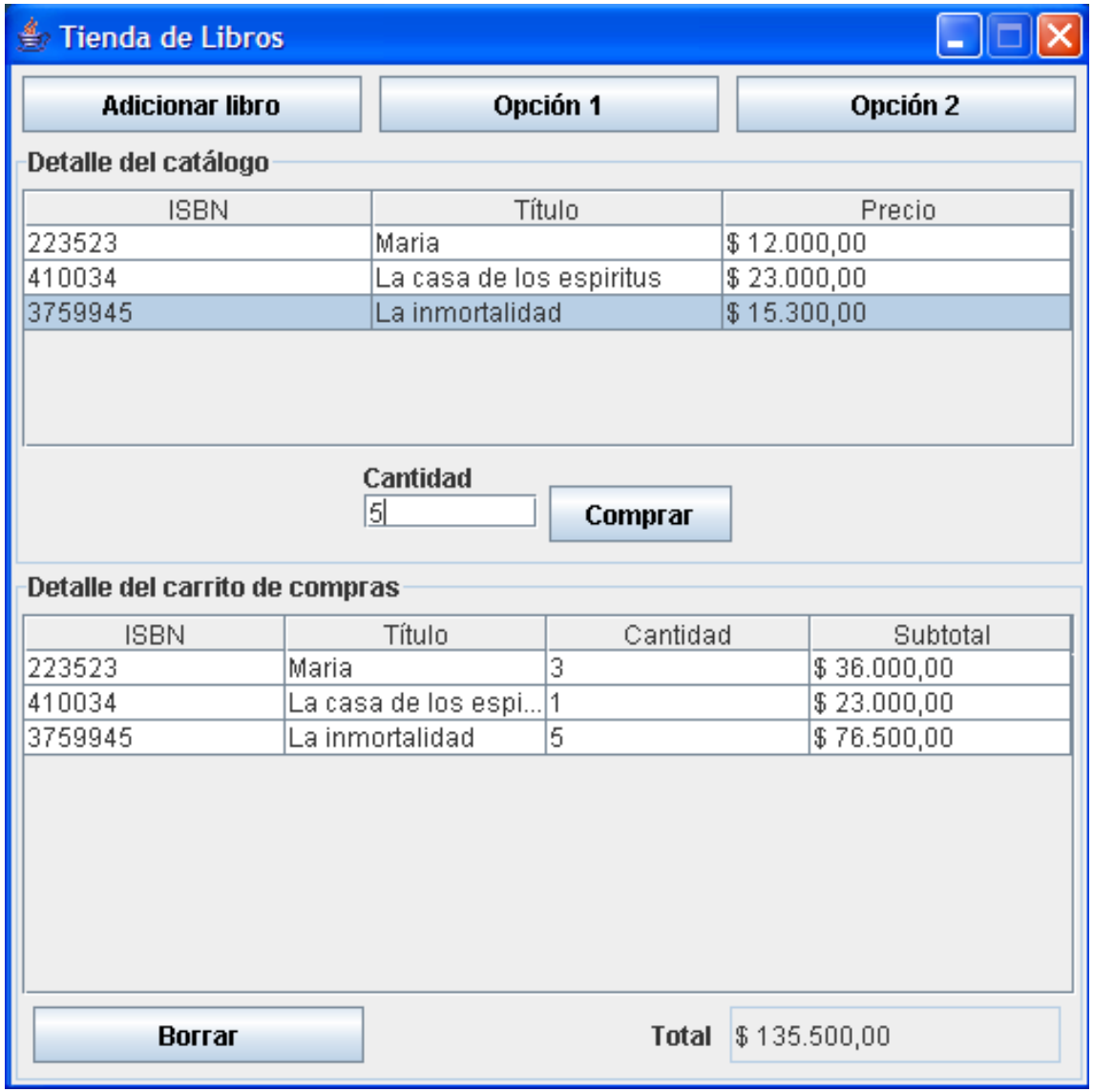
ARREGLOS

- Tamaño fijo
- Permite almacenar elementos de tipo simple (int, float, double) y objetos
- Para implementarlos en java, existe una sintaxis especial:
  - [ ]: para obtener un elemento del arreglo
  - length: para conocer la longitud

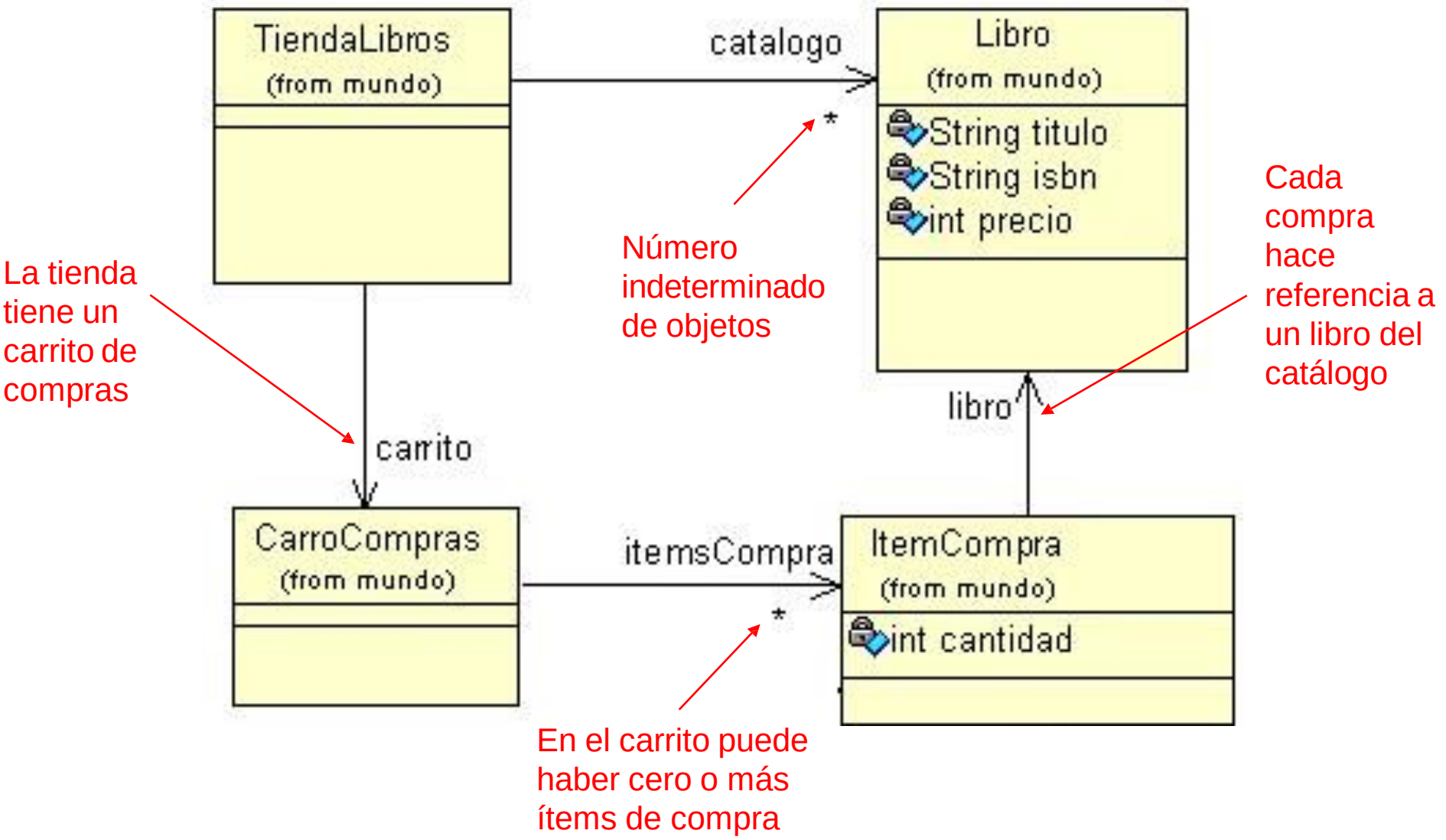
VECTORES

- Tamaño variable
- NO permite almacenar elementos de tipo simple (int, float, double), SOLO objetos
- Para implementarlos en java debemos utilizar una clase especial de java (ArrayList).
- Para manipular un vector se utiliza entonces la misma sintaxis que utilizamos para manejar cualquier otra clase (llamado a métodos)

Interfaz usuario



Modelo del Mundo



La clase Libro

```
public class Libro
{
    // Atributos
    private String titulo;
    private  String  isbn;
    private int precio;
}
```

La clase CarroCompras

```
public class CarroCompras
{
    // Atributos
    private ArrayList itemsCompra;
}
```

La clase ItemCompra

```
public class ItemCompra
{
    // Atributos
    private Libro libro;
    private int cantidad;
}
```

Declaración de un vector

```
import java.util.*;
```

Se importa el paquete de java donde se encuentra la clase ArrayList

```
public class TiendaLibros
{
    private ArrayList catalogo;

    private CarroCompras carrito;

}
```

Declaración de un vector

```
import java.util.*;

public class TiendaLibros
{
    private ArrayList catalogo;

    private CarroCompras carrito;

}
```

catalogo es un vector

Declaración de un vector

```
import java.util.*;

public class TiendaLibros
{
    private ArrayList catalogo;

    private CarroCompras carrito;

}
```

carrito es un objeto de la clase CarroCompras

Declaración de un vector

```
import java.util.*;
```

Se importa el paquete de java donde se encuentra la clase ArrayList

```
public class CarroCompras
{
    private ArrayList itemsCompra;

}
```

itemsCompra es un vector

Inicialización y tamaño de un vector

- public TiendaLibros( )
- {
  - catalogo = new ArrayList( ); carrito = new CarroCompras( );
- }

← En el método constructor de la clase TiendaLibros

Inicialización y tamaño de un vector

```
public TiendaLibros( )
{
    catalogo = new ArrayList( );
    carrito = new CarroCompras( );
}
```

- Se usa la misma sintaxis que para crear un objeto de cualquier clase (new)

- No se dice cuántos elementos va a tener el vector. El tamaño inicial es 0 (cero).

- No se dice de que tipo son los objetos que se van a guardar en él

Para conocer el número de elementos que hay en un vector

- isEmpty( ) - Es un método que retorna verdadero si el vector no tiene elementos, y falso en caso contrario.
- size( ) – Es un método que retorna el número de elementos que hay en el vector

En la clase …	La expresión …	Se interpreta …
TiendaLibros	catalogo.size( )	
TiendaLibros	catalogo.size( ) == 10	
TiendaLibros	catalogo.isEmpty( ) == true ó catalogo.isEmpty( )	
CarroCompras	itemsCompra.size( )	

Patrón de recorrido total

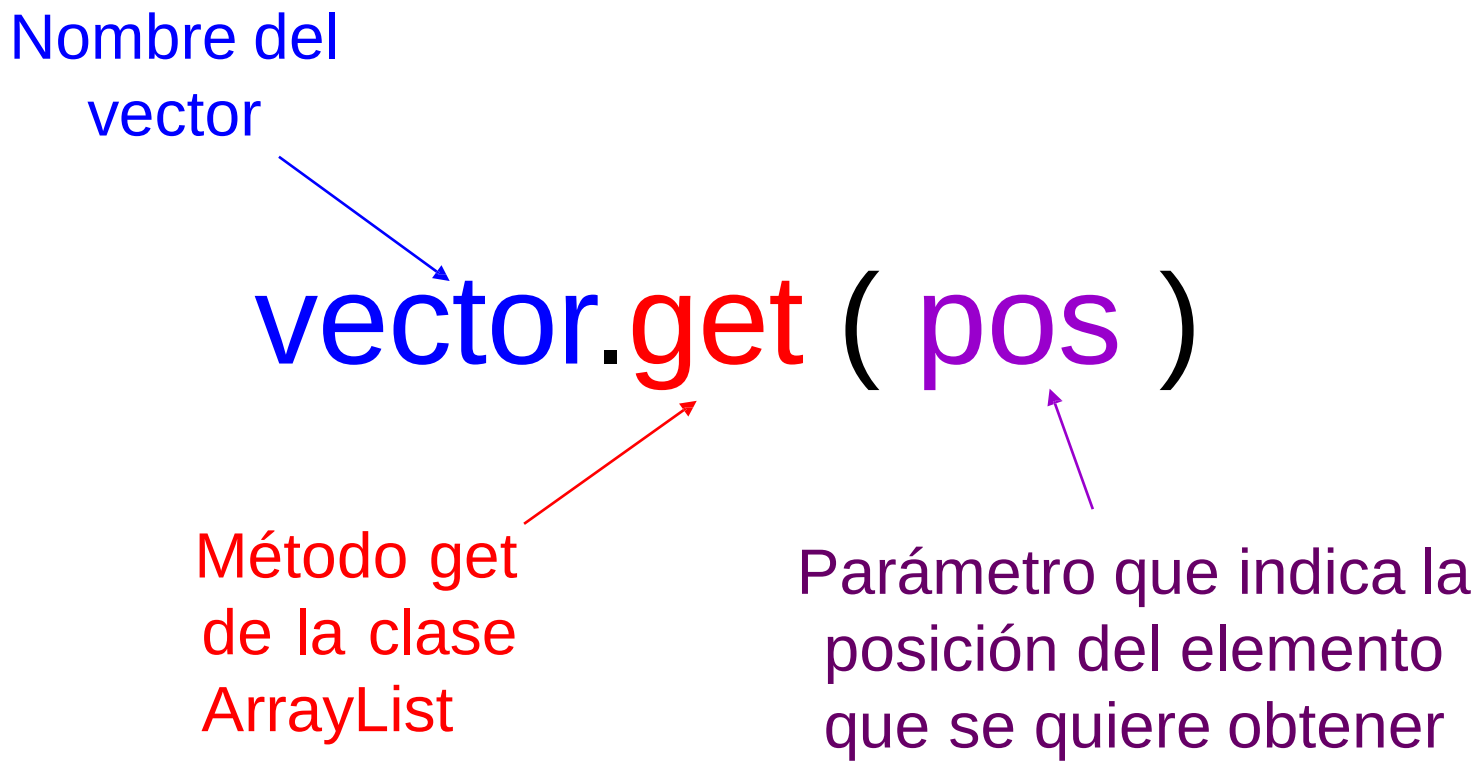
En arreglos

```
for ( int i = 0; i < notas.length; i++)
{
    <cuerpo del ciclo>
}
```

En vectores

```
for ( int i = 0; i < catalogo.size( ); i++)
{
    <cuerpo del ciclo>
}
```

Acceso a los elementos de un vector



Ejemplo en la clase TiendaLibros

```
public int inventario( )
{
    int sumaPrecios = 0;
    Libro libroAux;

    for ( int i = 0; i < catalogo.size( ); i++)
    {
        libroAux = ( Libro ) catalogo.get( i );
        sumaPrecios += libroAux.darPrecio( );
    }
    return sumaPrecios;
}
```


Variable auxiliar o temporal de tipo Libro

Ejemplo en la clase TiendaLibros

```
public int inventario( )
{
    int sumaPrecios = 0;
    Libro libroAux;

    for ( int i = 0; i < catalogo.size( ); i++)
    {
        libroAux = ( Libro ) catalogo.get( i );
        sumaPrecios += libroAux.darPrecio( );
    }
    return sumaPrecios;
}
```

Recorrido total hasta el final del vector catalogo

Ejemplo en la clase TiendaLibros

```
public int inventario( )
{
    int sumaPrecios = 0;
    Libro libroAux;

    for ( int i = 0; i < catalogo.size( ); i++)
    {
        libroAux = ( Libro ) catalogo.get( i );
        sumaPrecios += libroAux.darPrecio( );
    }
    return sumaPrecios;
}
```

Obtención del elemento de la posición i del vector

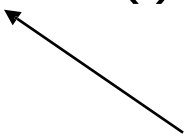
Ejemplo en la clase TiendaLibros

```
public int inventario( )
{
    int sumaPrecios = 0;
    Libro libroAux;

    for ( int i = 0; i < catalogo.size( ); i++)
    {
        libroAux = ( Libro ) catalogo.get( i );
        sumaPrecios += libroAux.darPrecio( );
    }
    return sumaPrecios;
}
```

Se especifica el tipo del objeto

Ejemplo en la clase TiendaLibros

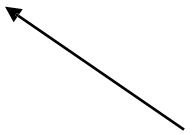
```
public int inventario( )
{
    int sumaPrecios = 0;
    Libro libroAux = null;

    for ( int i = 0; i < catalogo.size( ); i++)
    {
        libroAux = ( Libro ) catalogo.get( i );
        sumaPrecios += libroAux.darPrecio( );
    }
    return sumaPrecios;
}
```



Llamado al método darPrecio del libroAux

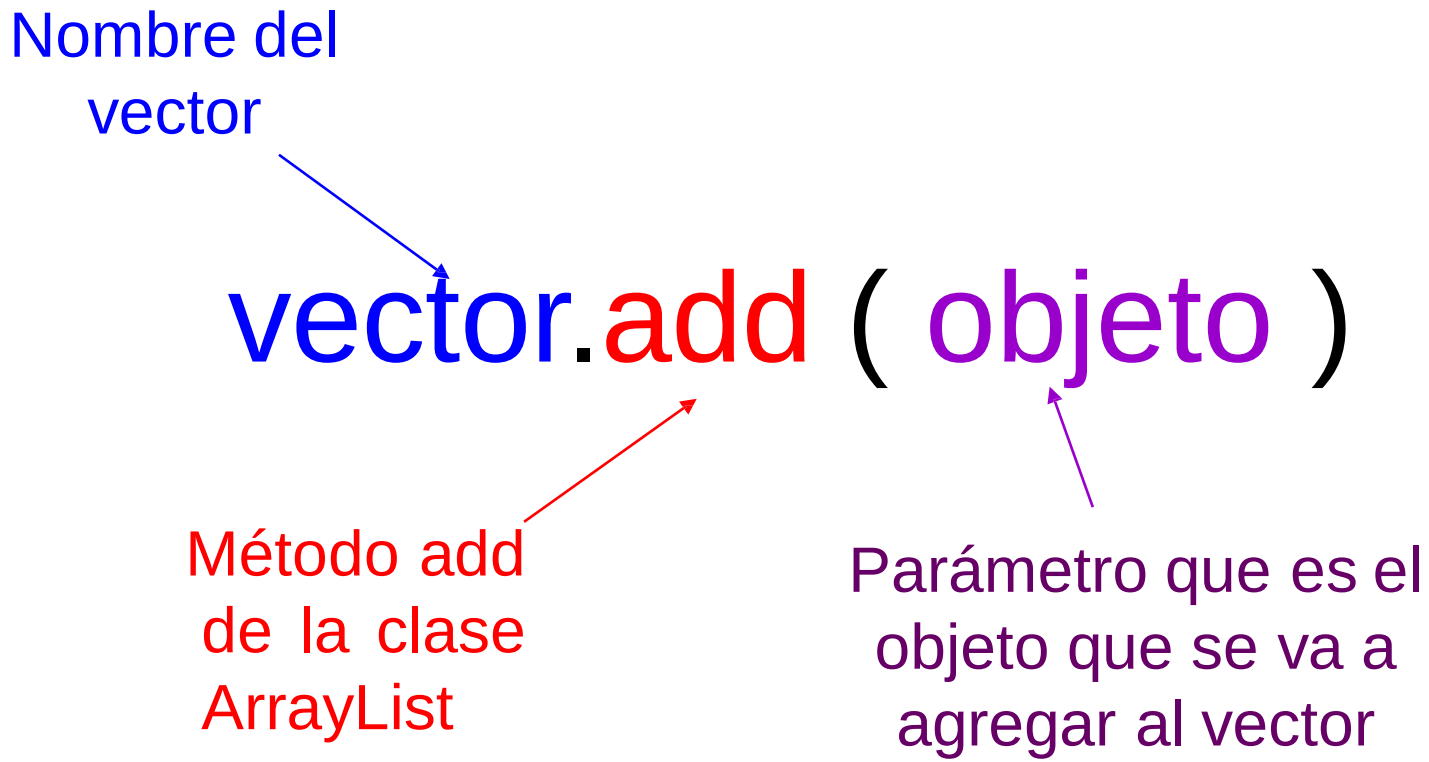
Ejemplo en la clase TiendaLibros

```
public int inventario( )
{
    int sumaPrecios = 0;

    for ( int i = 0; i < catalogo.size( ); i++)
    {
        sumaPrecios += ( Libro ) (catalogo.get( i )).darPrecio( );
    }
    return sumaPrecios;
}
```

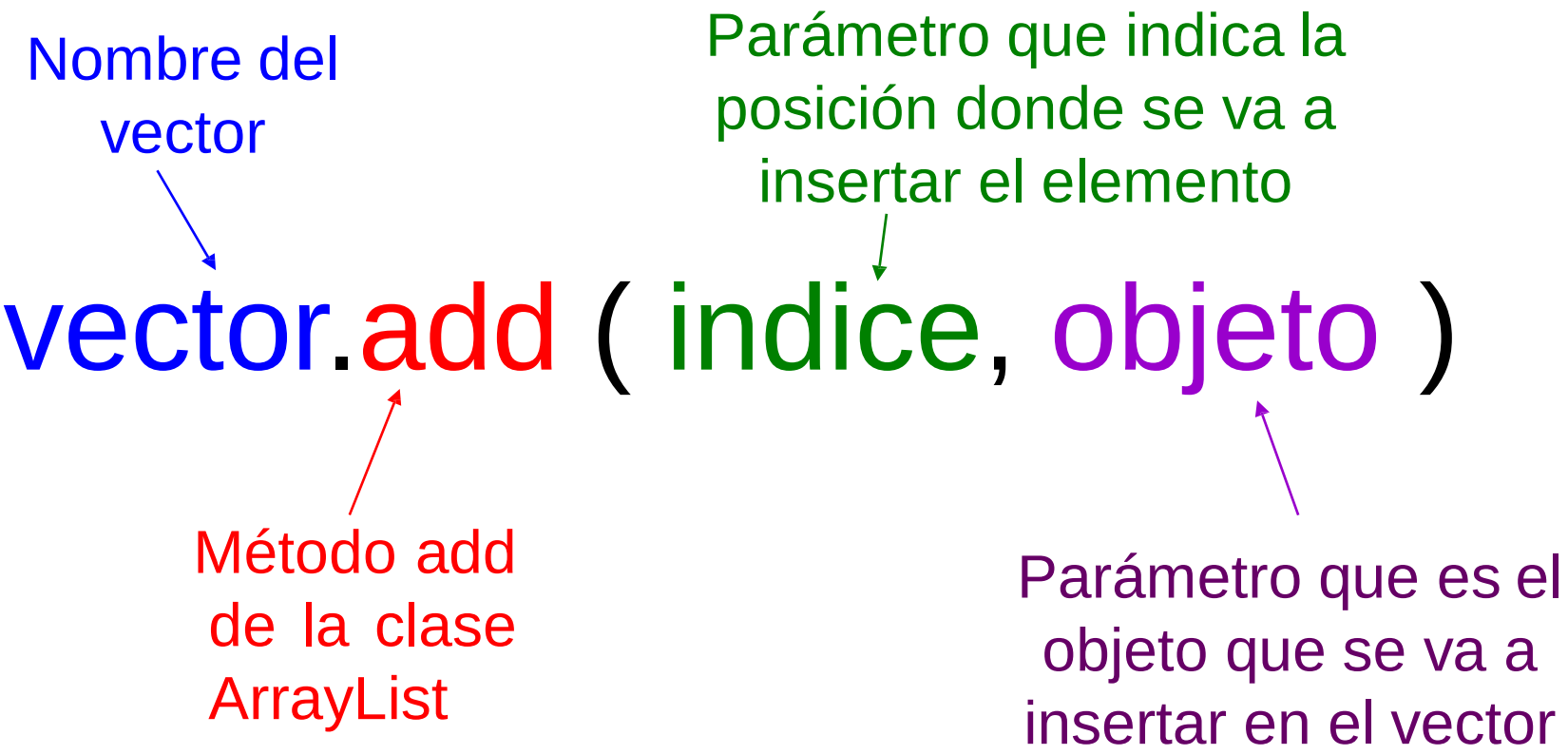


Llamado al método darPrecio del libroAux

Agregar un elemento al final de un vector



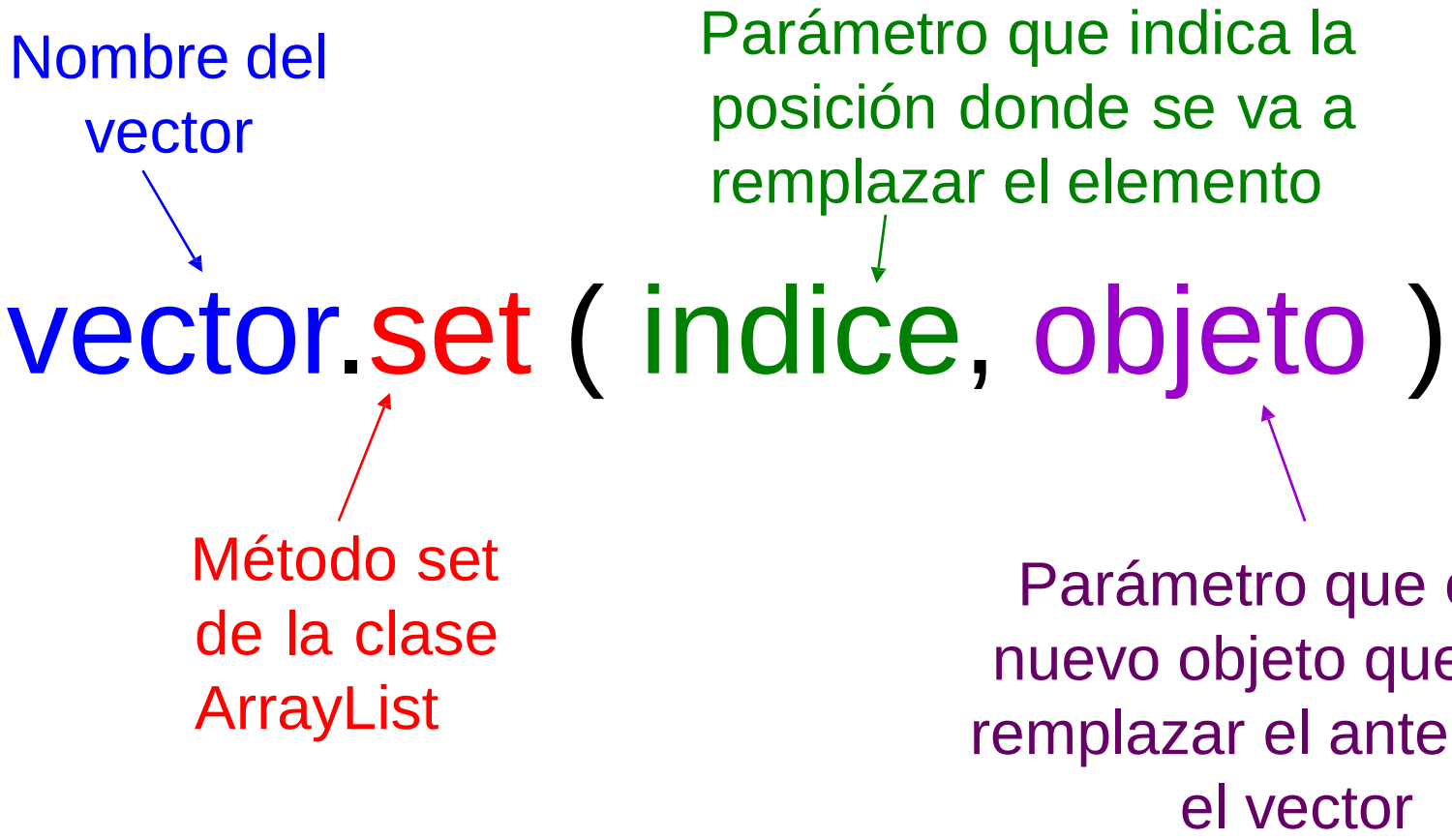
Insertar un elemento en una posición dada dentro de un vector



Ejemplo en la clase TiendaLibros

```
public void agregarTresLibros( )
{
    Libro libroAux1 = new Libro("Cien años de soledad", "98657", 15000);
    Libro libroAux2 = new Libro("La inmortalidad", "23758", 23800);
    Libro libroAux3 = new Libro("El perfume", "5759308", 42100);

    catalogo.add( libroAux1 );
    catalogo.add( libroAux2 );
    catalogo.add( 1, libroAux3 );
}
```

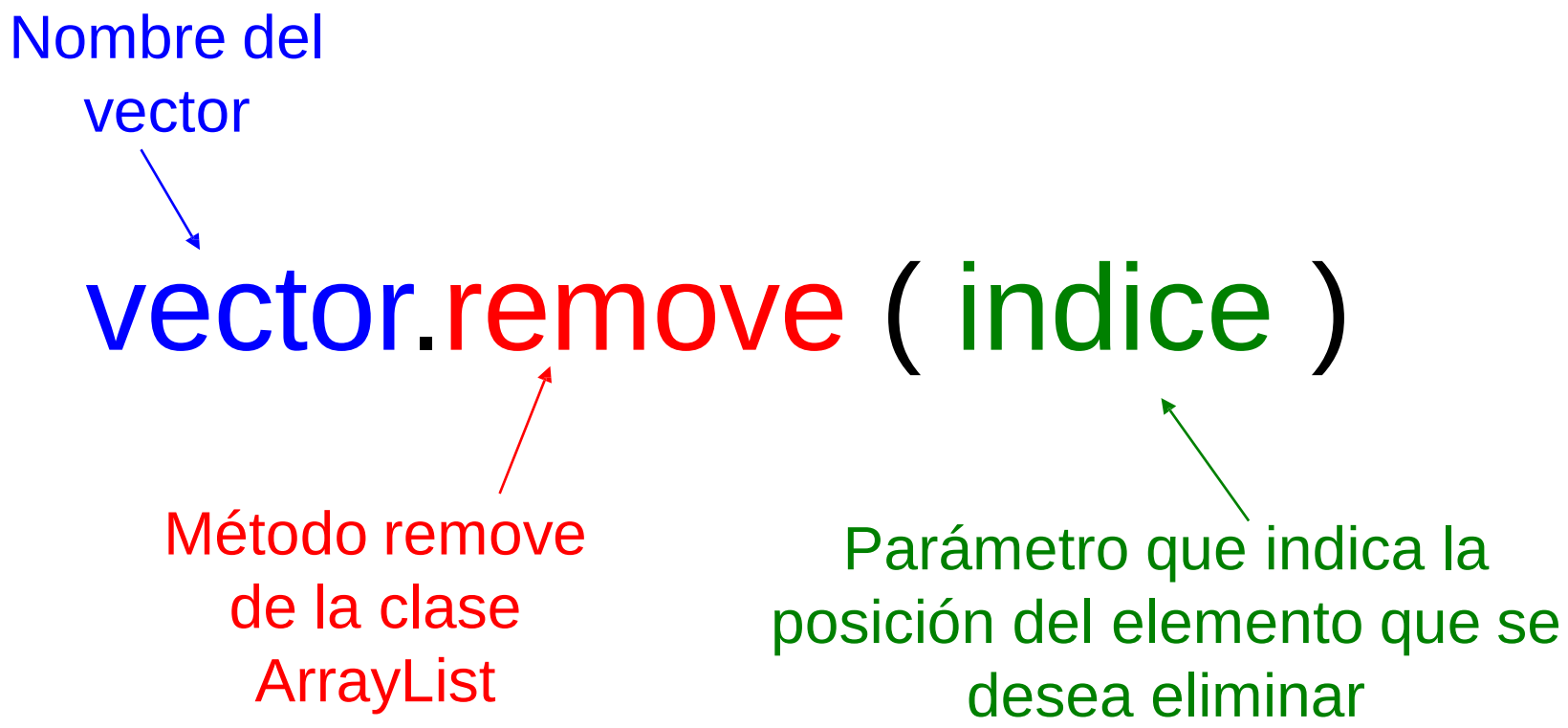
Remplazar un elemento en un vector



Ejemplo en la clase TiendaLibros

```
public void intercambiar( int pos1, int pos2 )
{
    Libro libroAux1;
    Libro libroAux2;

    libroAux1 = (Libro) catalogo.get( pos1 );
    libroAux2 = (Libro) catalogo.get( pos2 );
    catalogo.set( pos1, libroAux2 );
    catalogo.set( pos2, libroAux1 );
}
```

Eliminar un elemento de un vector



Tarea No. 10 - En la clase TiendaLibros:

Localizar un libro en el catalogo, dado su ISBN (que se recibe como parámetro). Si no lo encuentra, el método debe retornar null

```
public Libro buscarLibro ( String isbn)
{




}
```

Tarea No. 10 - En la clase TiendaLibros:

Adicionar un libro en el catálogo, si no existe ya un libro con ese ISBN. Utilice el método anterior para identificar el caso en el cual ya hay un libro con ese ISBN. El libro que se desea adicionar se recibe como parámetro.

```
public void adicionarLibroCatalogo ( Libro nuevoLibro)
{









}
```

Tarea No. 11 - En la clase CarroCompras:

Adicionar una cantidad de ejemplares de un libro al pedido actual. El método debe considerar el caso en el que el libro ya se encuentre en el pedido, caso en el cual solo debe incrementar el número de ejemplares. Si el libro no se encuentra, el método debe crear un nuevo item ItemCompra. El libro que se desea adicionar, así como la cantidad se reciben como parámetros.

```
public void adicionarCompra ( Libro lib, int cant)
{



}
```

Tarea No. 11 - En la clase CarroCompra:

Calcular el monto total de la compra del usuario. Para esto debe tener en cuenta el precio de cada libro y el número de ejemplares de cada uno que hay en el pedido.

```
public int calcularValorTotalCompra ( )
{




}
```

Tarea No. 11 - En la clase CarroCompra:

Eliminar del pedido el libro que tiene el ISBN dado como parámetro. Si no hay ningún libro con ese ISBN, el método no hace nada.

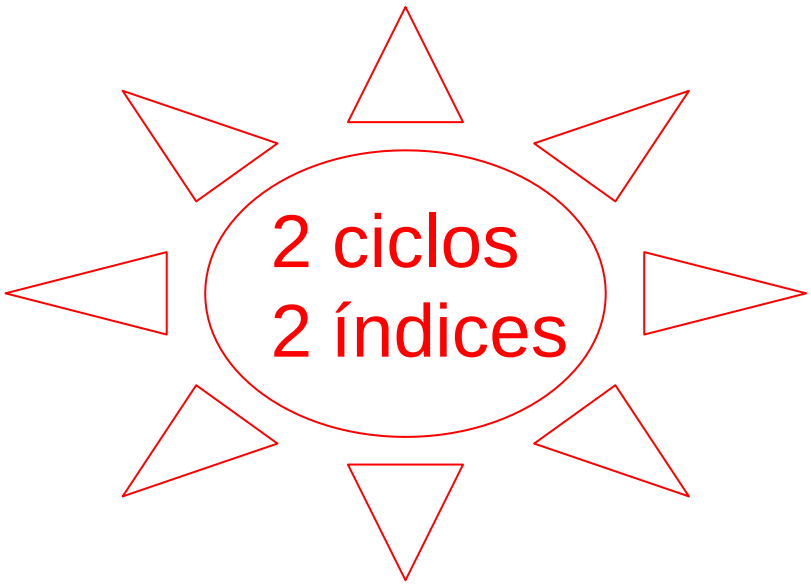
```
public void borrarItemCompra( String isbn)
{




}
```

Patrón de doble recorrido

- Se usa cuando por CADA ELEMENTO del arreglo se debe hacer un recorrido COMPLETO
- Ejemplos:
  - Encontrar la nota que aparece un mayor número de veces en el curso
  - …
  - …
  - …

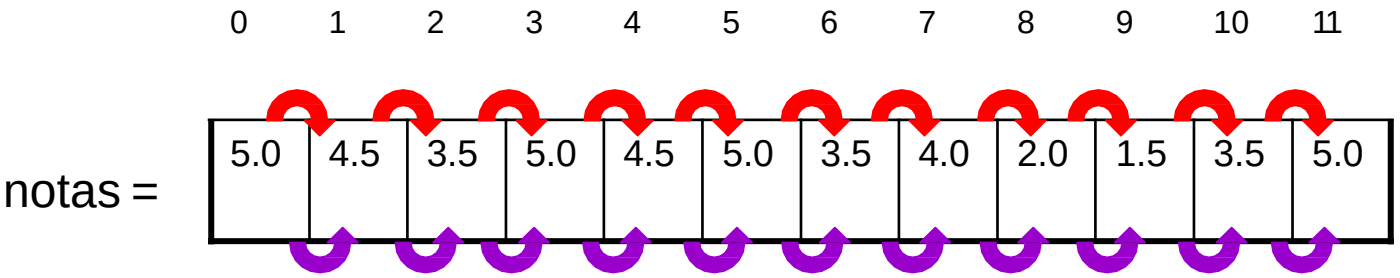
Patrón de doble recorrido

```
for ( int indice1 = 0; indice1 < arreglo.length; indice1++)
{
    for (int indice2 = 0; indice2 < arreglo.length; indice2++)
     {
         <cuerpo del ciclo interno>
     }
     <cuerpo del ciclo externo>
}
```



Patrón de doble recorrido

```
for ( int indice1 = 0; indice1 < arreglo.length; indice1++)
{
    for (int indice2 = 0; indice2 < arreglo.length; indice2++)
     {
        <cuerpo del ciclo interno>
     }
     <cuerpo del ciclo externo>
}
```



indice1: avanza lentamente

Indice2: avanza rápidamente. Cada vez que índice1 se incrementa en una unidad, indice2 recorre TODO el arreglo

Ejemplo

Encontrar la nota que aparece un mayor número de veces en el curso

```
public double masVecesAparece( )
{
    Qué hacer ???
}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
5.0	4.5	3.5	5.0	4.5	5.0	3.5	4.0	3.5	1.5	3.5	5.0

Ejemplo

Encontrar la nota que aparece un mayor número de veces en el curso

```
public double masVecesAparece( )
{
   double notasMasVecesAparece = 0.0;
   for ( int i = 0; i < notas.length; i++)
   {
      for (int j = 0; j < notas.length; j++)
    {
        <cuerpo del ciclo interno>
    }
    <cuerpo del ciclo externo>
   }
   return notaMasVecesAparece;
}
```

El resultado lo vamos a dejar en una variable llamada notasMasVecesAparece, la cual retornamos al final del método

Ejemplo

Encontrar la nota que aparece un mayor número de veces en el curso

```
public double masVecesAparece( )
{
   double notasMasVecesAparece = 0.0;
   for ( int i = 0; i < notas.length; i++)
   {
      for (int j = 0; j < notas.length; j++)
     {
         <cuerpo del ciclo interno>
     }
     <cuerpo del ciclo externo>
   }
   return notaMasVecesAparece;
}
```

Armamos la estructura del método a partir del esqueleto del patrón.

Utilizamos las variables **i** y **j** para llevar los índices de cada uno de los ciclos

Ejemplo

Encontrar la nota que aparece un mayor número de veces en el curso

```
public double masVecesAparece( )
{
   double notasMasVecesAparece = 0.0;
   for ( int i = 0; i < notas.length; i++)
   {
      for (int j = 0; j < notas.length; j++)
     {
         <cuerpo del ciclo interno>
     }
     <cuerpo del ciclo externo>
   }
   return notaMasVecesAparece;
}
```

Hay dos problemas para resolver:

1) Contar el número de veces que aparece en el arreglo el valor que está en la casilla **i**
2) Encontrar el mayor valor entre los calculados por el primer problema

Ejemplo

Encontrar la nota que aparece un mayor número de veces en el curso

```
public double masVecesAparece( )
{
    double notaMasVecesAparece = 0.0;

    for ( int i = 0; i < notas.length; i++)
    {
      for (int j = 0; j < notas.length; j++)
      {
        <contar cuantas veces aparece }
          la nota buscada (notas [ i ])>
      }
      <verificar si es la nota mas frecuente>
    }
     return notaMasVecesAparece;
}
```

Hay dos problemas para resolver:

1) Contar el número de veces que aparece en el arreglo el valor que está en la casilla **i**
2) Encontrar el mayor valor entre los calculados por el primer problema

Primer problema: contar el número de veces que aparece en el arreglo el valor que está en la casilla i

```
public double masVecesAparece( )
{
   double notasMasVecesAparece = 0.0;
   for ( int i = 0; i < notas.length; i++)
   {
      double notaBuscada = notas[ i ];
      int contador = 0;

       for (int j = 0; j < notas.length; j++)
       {
           if ( notas[ j ] == notaBuscada )
             contador++;
       }
     <verificar si es la nota mas frecuente>
   }
   return notaMasVecesAparece;
}
```

Dejamos en la variable notaBuscada, la nota para la cual queremos contar el número de ocurrencias. La inicializamos con la nota de la casilla i.

Primer problema: contar el número de veces que aparece en el arreglo el valor que está en la casilla i

```
public double masVecesAparece( )
{
   double notasMasVecesAparece = 0.0;
   for ( int i = 0; i < notas.length; i++)
   {
      double notaBuscada = notas[ i ];
      int contador = 0;

       for (int j = 0; j < notas.length; j++)
       {
           if ( notas[ j ] == notaBuscada )
             contador++;
       }
     <verificar si es la nota mas frecuente>
   }
   return notaMasVecesAparece;
}
```

Usamos una segunda variable llamada contador para acumular allí el número de veces que aparece la nota buscada.

Primer problema: contar el número de veces que aparece en el arreglo el valor que está en la casilla i

```
public double masVecesAparece( )
{
   double notasMasVecesAparece = 0.0;
   for ( int i = 0; i < notas.length; i++)
   {
      double notaBuscada = notas[ i ];
      int contador = 0;

       for (int j = 0; j < notas.length; j++)
       {
          if ( notas[ j ] == notaBuscada )
            contador++;
       }
     <verificar si es la nota mas frecuente>
   }
   return notaMasVecesAparece;
}
```

En el ciclo interno, vamos a contar cuántas veces aparece la nota buscada. Se incrementa el contador cada vez que notaBuscada sea igual a notas[j].

Segundo problema: Encontrar el mayor valor entre los calculados por el primer problema

```
public double masVecesAparece( )
{
   double notaMasVecesAparece = 0.0;
   int numeroVecesAparece = 0;
   for ( int i = 0; i < notas.length; i++)
   {
        double notaBuscada = notas[ i ];
        int contador = 0;
         for (int j = 0; j < notas.length; j++)
        {
                    if ( notas [ j ] == notaBuscada )
                      contador++;
        }
       if ( contador > numeroVecesAparece)
       {
          notasMasVecesAparece = notaBuscada;
          numeroVecesAparece = contador;
       }
    }
    return notaMasVecesAparece;
}
```

Usamos esta variable para indicar la nota que hasta el momento aparece mas veces

Segundo problema: Encontrar el mayor valor entre los calculados por el primer problema

```
public double masVecesAparece( )
{
    double notaMasVecesAparece = 0.0;
    int numeroVecesAparece = 0;
    for ( int i = 0; i < notas.length; i++)
    {
         double notaBuscada = notas[ i ];
         int contador = 0;
          for (int j = 0; j < notas.length; j++)
         {
                        if ( notas [ j ] == notaBuscada )
                          contador++;
         }
        if ( contador > numeroVecesAparece)
        {
            notasMasVecesAparece = notaBuscada;
            numeroVecesAparece = contador;
        }
     }
     return notaMasVecesAparece;
}
```

Usamos esta variable para contar cuántas veces aparece la nota notaMasVecesAparece

Segundo problema: Encontrar el mayor valor entre los calculados por el primer problema

```
public double masVecesAparece( )
{
    double notaMasVecesAparece = 0.0;
    int numeroVecesAparece = 0;
    for ( int i = 0; i < notas.length; i++)
    {
          double notaBuscada = notas[ i ];
          int contador = 0;
           for (int j = 0; j < notas.length; j++)
          {
                        if ( notas [ j ] == notaBuscada )
                          contador++;
          }
         if ( contador > numeroVecesAparece)
         {
             notasMasVecesAparece = notaBuscada;
             numeroVecesAparece = contador;
         }
     }
     return notaMasVecesAparece;
}
```

Definimos el caso en el cual debemos cambiar el acumulado: si encontramos un valor que aparezca mas veces que el que teníamos hasta el momento (contador > numeroVecesAparece) debemos actualizar los valores de las variables

Conclusión

- Para resolver un problema que implique doble recorrido:
  - Primero debemos identificar los dos problemas que queremos resolver (uno con cada ciclo)
  - Luego, debemos tratar de resolverlos independientemente, usando los patrones de recorrido total o parcial
- Si para resolver el problema, se necesita un tercer ciclo anidado, debemos escribir métodos separados que ayuden a resolver cada problema individualmente (nivel 4), porque la solución directa es muy compleja y propensa a errores.

Tarea No. 5: Calcular una nota del curso tal que la mitad de las notas sean menores o iguales a ella. Si hay varias que lo cumplan, puede retornar cualquiera.

```
public double notaMediana ( )
{




}
```

Ahora con la Tienda de Libros

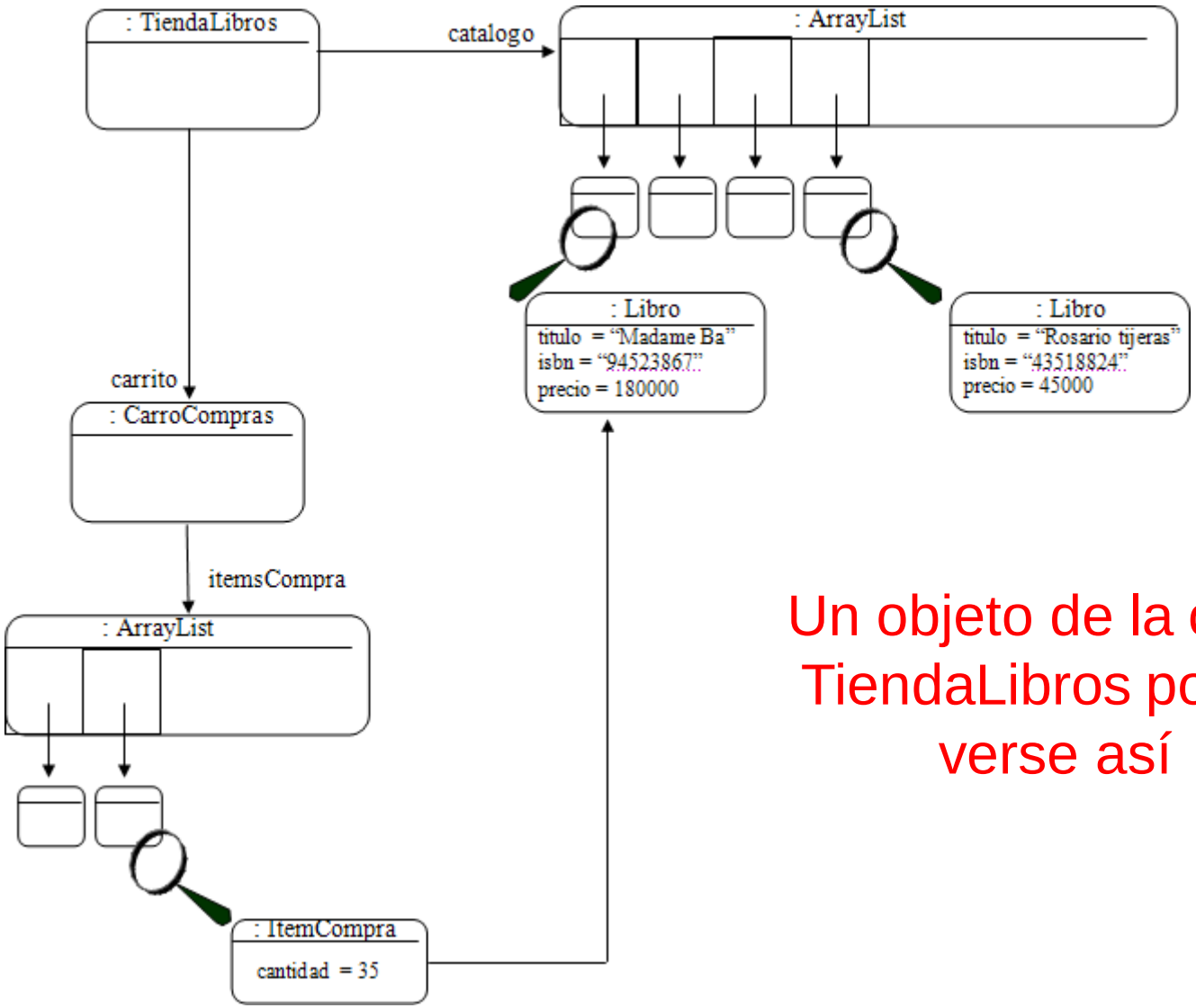
Tarea: Informar (verdadero o falso) si en el catálogo de la tienda de libros hay dos libros con el mismo título

```
public boolean hayDobleTitulo( )
{
    Qué hacer ???
}
```

Ahora con la Tienda de Libros

Tarea: Informar (verdadero o falso) si en el catálogo de la tienda de libros hay dos libros con el mismo título

```
public boolean hayDobleTitulo( )
{




}
```



Un objeto de la clase TiendaLibros podría verse así