Programación orientada a objetos

Capítulo 4

Agrupación de objetos

Tutor: Manuel Fernández Barcell
Centro Asociado de Cádiz
http://prof.mfbarcell.es

Tema 4. Agrupar objetos. Semana 4	
1- Librerías de clases 2- Clases genéricas 3- Colecciones de tamaño flexible: ArrayList a. Procesamiento de colecciones b. Estructuras de control: los bucles for-each while c. Acceso mediante índices e iteradores 4- Colecciones de tamaño fijo: Array a. Creación y declaración de arrays b. Uso de arrays c. Estructuras de control: el bucle for	1- Estudiar el capítulo 4 del libro base para la Unidad Didáctica I 2- Realizar los ejercicios en el entorno BlueJ sugeridos en el libro base 3- Definición de estructuras de almacenamiento e implementación de los métodos y constructores necesarios para la realización de la práctica

Tema 4. Se presentan las colecciones de objetos como estructuras de datos útiles en Java. Se desarrollan las técnicas para llevar a cabo recorrido sobre las colecciones haciendo uso de las sentencias de control de flujo de programa en Java.

1- Implementar en Java un recorrido sobre estructuras de almacenamiento del tipo colección.
2- Entender los conceptos de biblioteca, paquete y objeto anónimo.

4.2 La Colección de objetos como abstracción

Las **colecciones de objetos** son objetos que pueden almacenar un número arbitrario de otros objetos.

Muchas aplicaciones requieren agrupar objetos

El número de elementos a agrupar varía

- Adición de elementos
- Eliminación de elementos

Y hay que ser capaces de manipular los elementos de la colección

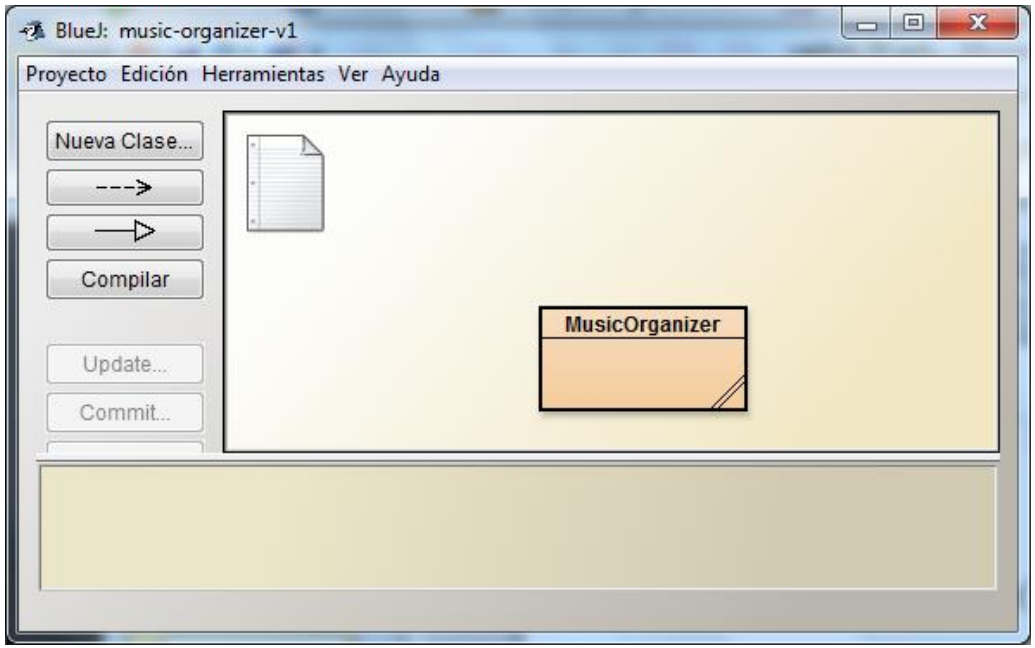
- Búsqueda de elementos
- Consulta y modificación de elementos

Agrupar cosas para referirnos y manejarlas de forma conjunta

- **ArrayList**

4.3 Un organizador para archivos de música

- Music-organizer-v1
  - No vamos a implementar las estructuras de datos para guardar las notas: usaremos librerías
    - Así no hay que hacerlo todo desde cero



Librerías de clases

- Una librería de clases es…
  - Una colección de clases útiles
- En Java una librería de clases es un *package*
  - Como la agrupación de objetos es algo habitual, en Java hay una librería de clases para esto:
    - **package java.util**
- Las librerías o paquetes de Java contiene cientos de clases

4.4 Utilización de una clase librería

```
import java.util.ArrayList;

/**
 * A class to hold details of audio files.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2011.07.31
 */
public class MusicOrganizer
{
    // An ArrayList for storing the file names of music files.
    private ArrayList<String> files;

    /**
     * Create a MusicOrganizer
     */
    public MusicOrganizer()
    {
        files = new ArrayList<String>();
    }

    /**
     * Add a file to the collection.
     * @param filename The file to be added.
     */
    public void addFile(String filename)
    {
        files.add(filename);
    }

    /**
```

```
    /**
     * Return the number of files in the collection.
     * @return The number of files in the collection.
     */
    public int getNumberOfFiles()
    {
        return files.size();
    }

    /**
     * List a file from the collection.
     * @param index The index of the file to be listed.
     */
    public void listFile(int index)
    {
        if(index >= 0 && index < files.size()) {
            String filename = files.get(index);
            System.out.println(filename);
        }
    }

    /**
     * Remove a file from the collection.
     * @param index The index of the file to be removed.
     */
    public void removeFile(int index)
    {
        if(index >= 0 && index < files.size()) {
            files.remove(index);
        }
    }
}
```

4.4.1 Importación de una clase librería

- El acceso a una clase estándar de librería de Java se obtiene mediante la sentencia ***import***. Ejemplo para ArrayList
  - **import java.util.ArrayList;**
- Las sentencias ***import*** deben ubicarse en el texto de la clase, siempre antes del comienzo de la declaración de la clase.
- Una vez que el nombre de una clase ha sido importado desde un paquete, podemos usar dicha clase tal como si fuera una de nuestras propias clases.
- *import*
  - No es necesario para el paquete java.lang
  - Siempre se asume: *import java.lang.**
- * permite importar todas las clases e interfaces de un paquete
  - **import java.util.*;** // todas las clases e interfaces de util
  - **import java.*;** // ERROR: no vale para paquetes
- Ejemplo: Para importar la clase Applet, hay dos posibilidades:
  - **import java.applet.Applet**; // directamente la clase
  - **import java.applet.*;** // todos los nombres del paquete

Usando el nombre completo:

```
class ImprimeFecha1 {
  public static void main (String[] args) {
        java.util.Date ahora = new java.util.Date();
        System.out.println(ahora);
  }
}
```

Usando la cláusula import:

```
import java.util.Date;
class ImprimeFecha2 {
  public static void main (String[] args) {
        Date ahora = new Date();
        System.out.println(ahora);
  }
}
```

Métodos de ArrayList

- La clase **ArrayList** declara muchos métodos entre ellos:
  - add
    - Almacena un objeto en la lista.
  - size
    - Devuelve el número total de elementos almacenados en la lista.
  - get
    - Devuelve un elemento, sin eliminarlo de la colección.
  - remove
    - Elimina un objeto en la lista.

Principales métodos de ArrayList

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final. Devuelve true.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1

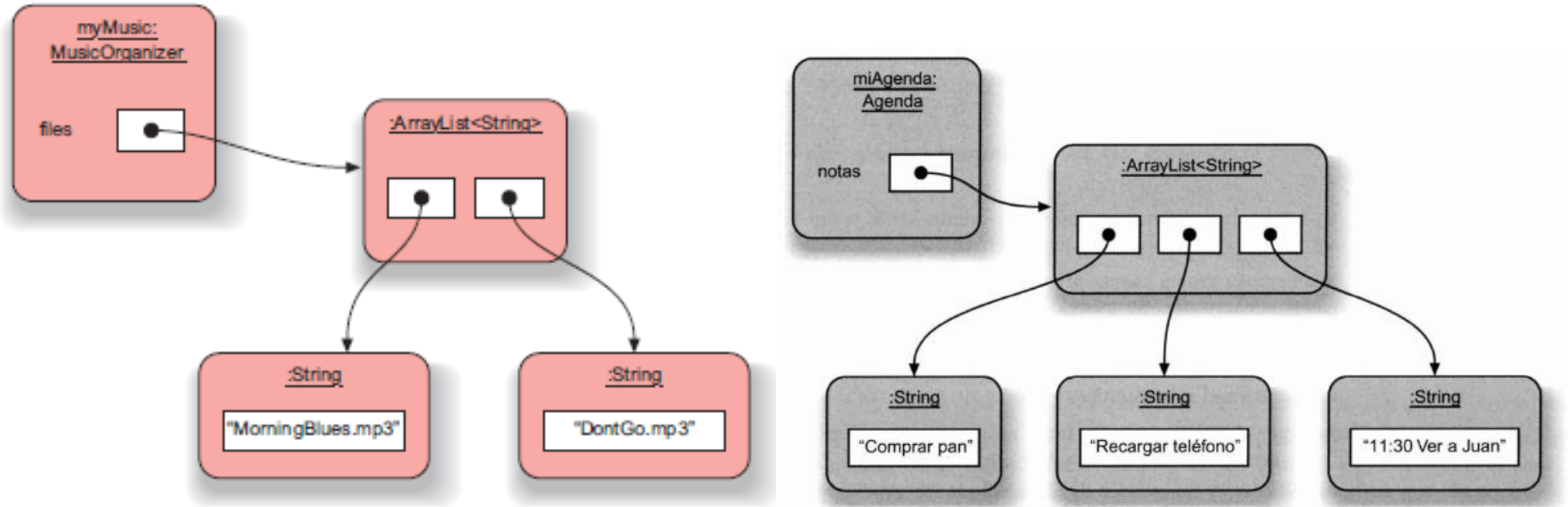
```
public class Notebook
{
    private ArrayList<String> notes;
    ...

    public void storeNote(String note)
    {
        notes.add(note);
    }

    public int numberOfNotes()
    {
        return notes.size();
    }

    ...
}
```

Adición de una nota

Devuelve el número de notas
(delegación)

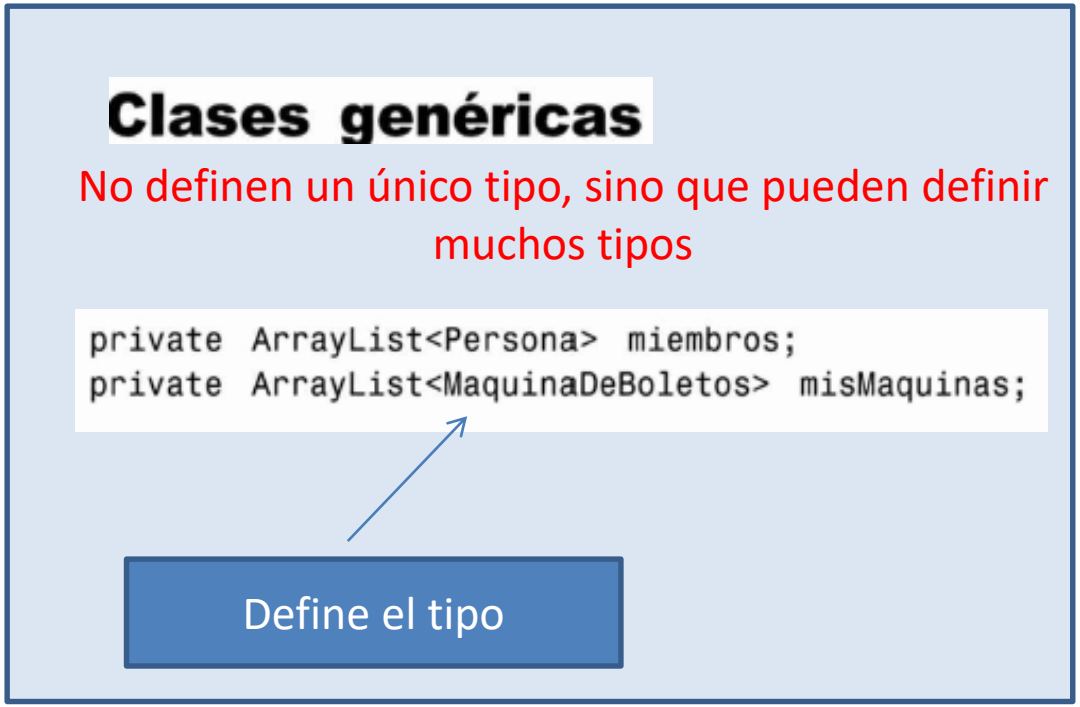
4.5 Estructura de objetos con “colecciones”

- Características importantes de la clase **ArrayList** son:
  - Es capaz de aumentar su capacidad interna tanto como se requiera. Cuando se agregan más elementos, simplemente crea el espacio necesario para ellos.
  - Mantiene su propia cuenta privada de la cantidad de elementos almacenados, que se obtiene mediante **size**.
  - Mantiene el orden de los elementos que se agregan, el método **add** almacena cada nuevo elemento al final de la lista, Posteriormente se pueden recuperar en el mismo orden.



4.6 Clases genéricas

- Es una clase que **requiere que se especifique un segundo tipo** como parámetro cuando se usa para declarar campos u otras variables.
- Las colecciones están implementadas como tipos genéricos o parametrizados
- Las clases genéricas no definen un tipo único en Java, sino potencialmente muchos tipos
- El tipo de parámetro indica de qué queremos que sea la lista:
  - ArrayList<Person>
  - ArrayList<TicketMachine>
  - etc

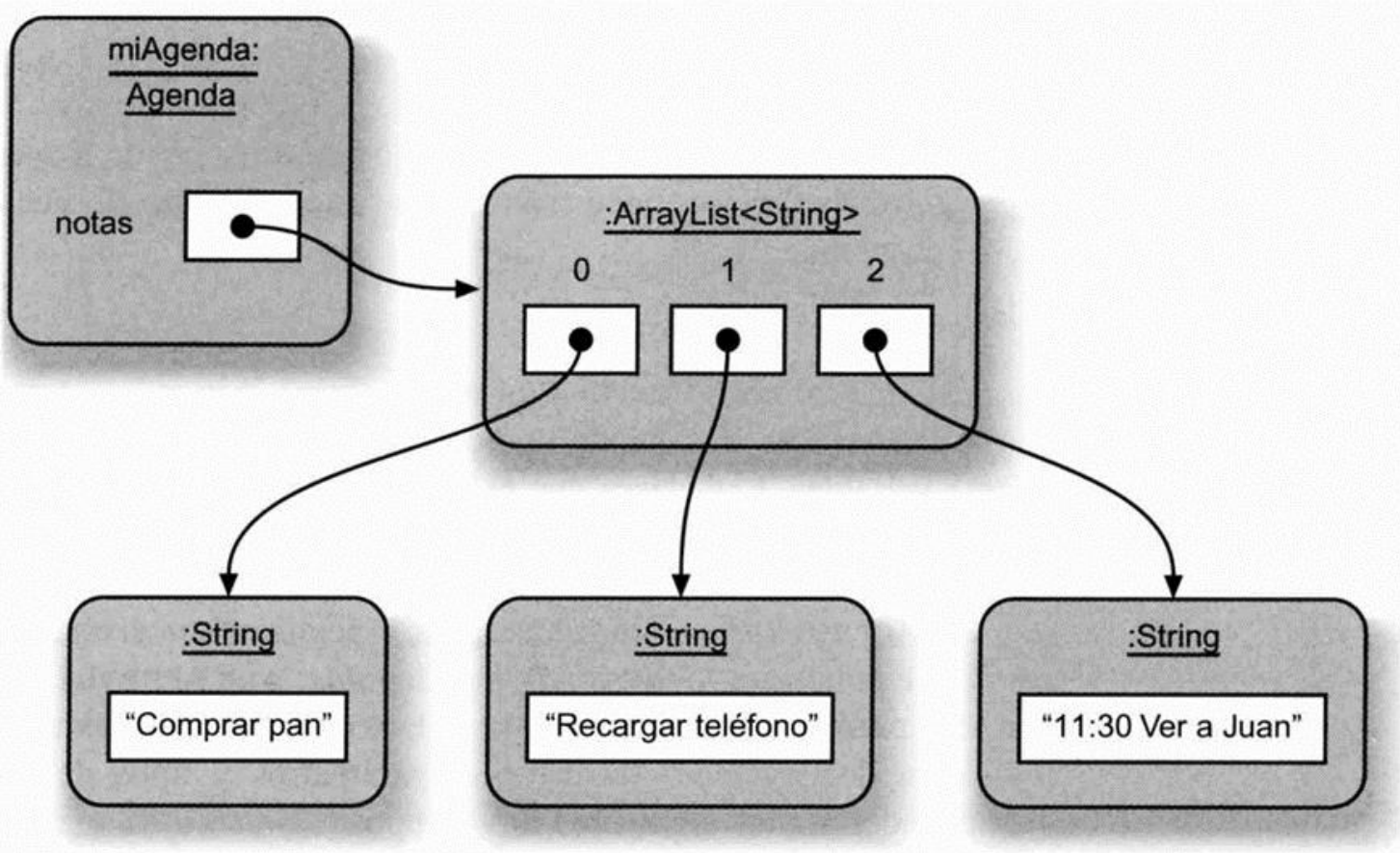


Notación Diamante

- Para instanciar un nuevo objeto de una clase genérica, se necesita especificar de nuevo el tipo completo con el tipo de elemento entre los símbolos de menor y de mayor, seguido de los paréntesis para la lista de parámetros:
  - peliculas = new ArrayList<String>();
- Desde la versión 7 de Java el compilador puede inferir el tipo parametrizado del objeto que se está creando a partir del tipo de la variable a la que se está realizando la asignación.
- Se conoce como notación diamante:
  - **peliculas = new ArrayList<>();**

4.7 Numeración dentro de las colecciones

- Los elementos almacenados en las colecciones tienen una numeración implícita o posicionamiento que comienza a partir de cero.
- La posición que ocupa un objeto en una colección se conoce como su índice.
  - El primer elemento que se agrega a una colección tiene por índice al número 0.
  - El segundo tiene al número 1.
  - Y así sucesivamente hasta size - 1.
  - Los métodos que se utilicen deben asegurar que el valor de su parámetro se encuentre dentro del rango de valores de índice permitidos **[0...size()-1**



Cuidado: si usted no es cuidadoso, podría intentar acceder a un elemento de una colección que está fuera de los índices válidos del `ArrayList`. Cuando lo haga, obtendrá un mensaje del error denominado *desbordamiento*. En Java, verá un mensaje que dice *IndexOutBoundsException*.

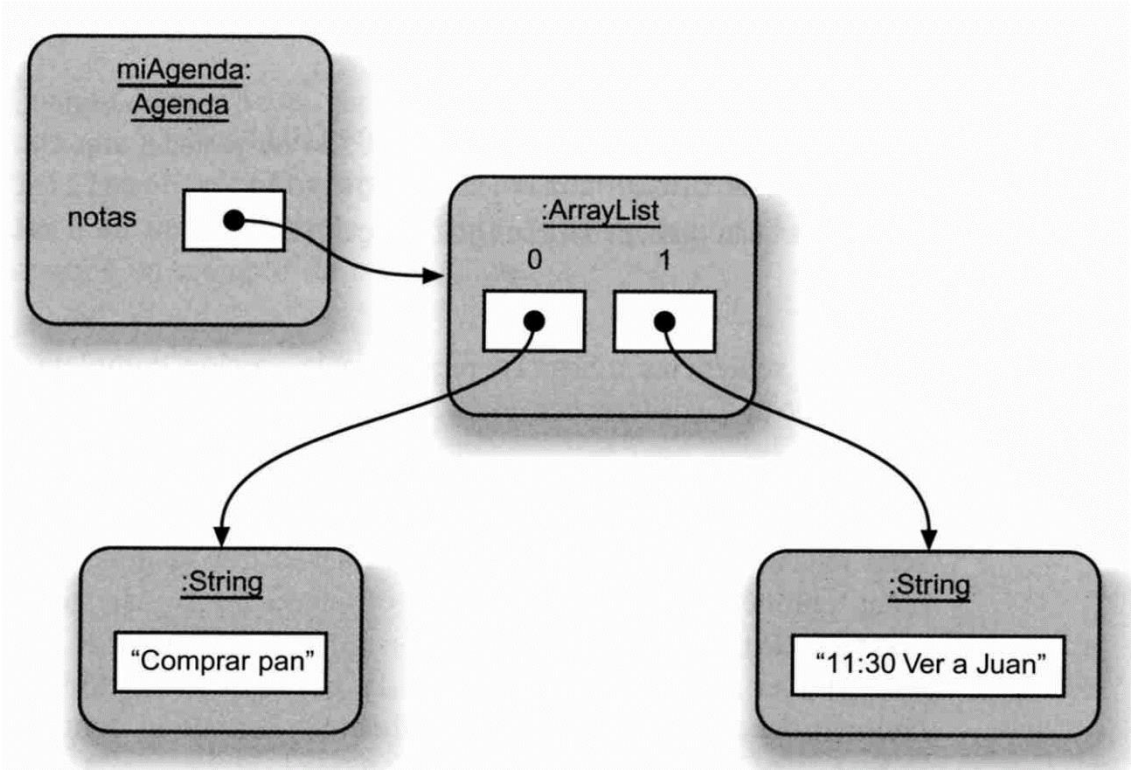
Eliminar un elemento de la “colección”

- La clase ArrayList tiene un método **remove** que toma como parámetro el índice del elemento o el propio elemento a eliminar.

```
public void eliminarNota(int numeroDeNota)
      {
            if(numeroDeNota < 0) {
                  // No es un número de nota válido, no
se hace nada.
            }
            else if(numeroDeNota < numeroDeNotas()) {
                  // Número de nota válido, se la puede
                     borrar.
                  notas.remove(numeroDeNota);
            }
            else {
                  // No es un número válido de nota,
                     entonces no se hace nada.
            }
      }
```

Modificación de los índices

- Una complicación del proceso de eliminación es que se modifican los valores de los índices de los restantes objetos que están almacenados en la colección.



Pregunta: Dada la siguiente clase Prueba:

```
public class Prueba {
	public static void main(String[] args) {
		ArrayList < Integer > valores = new ArrayList < Integer > ();
		valores.add(4);
		valores.add(5);
		valores.set(1, 6);
		valores.remove(0);
		for(Integer v : valores) {
			System.out.print(v);
		}
	}

}
```

Al ejecutar el código obtendremos:

a. Un error en tiempo de ejecución
b. Se mostrará 4
c. Se mostrará 5
d. Se mostrará 6

Resumen

- Las colecciones permiten guardar un número arbitrario de objetos
  - Que se pueden añadir y quitar
  - Cada elemento tiene un índice
  - Los valores de los índices pueden cambiar al eliminar o añadir elementos
  - Los principales métodos de ArrayList son `add`, `get`, `remove` y `size`
- Las librerías de clases suelen contener clases de colección bien probadas
- Las librerías de clases en Java se llaman *packages*
- La clase *ArrayList* del paquete *java.util* es un ejemplo de tipo genérico o parametrizado

Iteración

- Habitualmente queremos repetir varias veces un conjunto de acciones
  - Por ejemplo, imprimir las notas de la agenda una a una
- La mayoría de los lenguajes de programación incluyen sentencias de bucle para hacer esto:
  - while, repeat, for, …
- Los bucles permiten controlar cuántas veces se repite un conjunto de acciones
  - En el caso de las colecciones es habitual repetir acciones para cada objeto de la colección particular

4.9.1 El ciclo “for-each” iteración definida

- Realiza el ciclo una vez por cada elemento de la colección
- Adecuado cuando hay que tratar a todos los elementos de la colección

Un **ciclo** puede usarse para ejecutar repetidamente un bloque de sentencias sin tener que escribirlas varias veces.

```
for (TipoDelElemento elemento : colección) {
    cuerpo del ciclo
}
```

```
Para cada elemento en la colección hacer:  {
    cuerpo del ciclo
}
```

```
/**
* Imprime todas las notas de la agenda
*/
public void imprimirNotas()
{
    for(String nota : notas) {
        System.out.println(nota);
    }
}
```

Define la variable de ciclo.
El tipo debe ser el mismo que el declarado en la colección

```java
/**
 * Show a list of all the files in the collection.
 */
public void listAllFiles()
{
    for(String filename : files) {
        System.out.println(filename);
    }
}
```

Proceso selectivo

```java
/**
 * List the names of files matching the given search string.
 * @param searchString The string to match.
 */
public void listMatching(String searchString)
{
    for(String filename : files) {
        if(filename.contains(searchString)) {
            // A match.
            System.out.println(filename);
        }
    }
}
```

```java
// Declaro un ArrayList
ArrayList<String> pelis = new ArrayList<String>();


// Estructura for each
for (String nombre: pelis)
     System.out.println(nombre);


for (String nombre:pelis){
   if (nombre.equals("Casablanca")){
   pelis.remove("Casablanca");
   }
}
```

El código no funciona y lanza una excepción.

```
Exception in thread "main" java.util.ConcurrentModificationException
   at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:901)
   at java.util.ArrayList$Itr.next(ArrayList.java:851)
   at Cine.main(Cine.java:83)
```

4.10.1 El bucle“while” iteración indefinida

```
while (condición del ciclo) {
    cuerpo del ciclo
}
```

Comparación con “for-each”

```
int indice = 0;
while(indice < notas.size()) {
   System.out.println(notas.get(indice));
   indice ++;
}
```

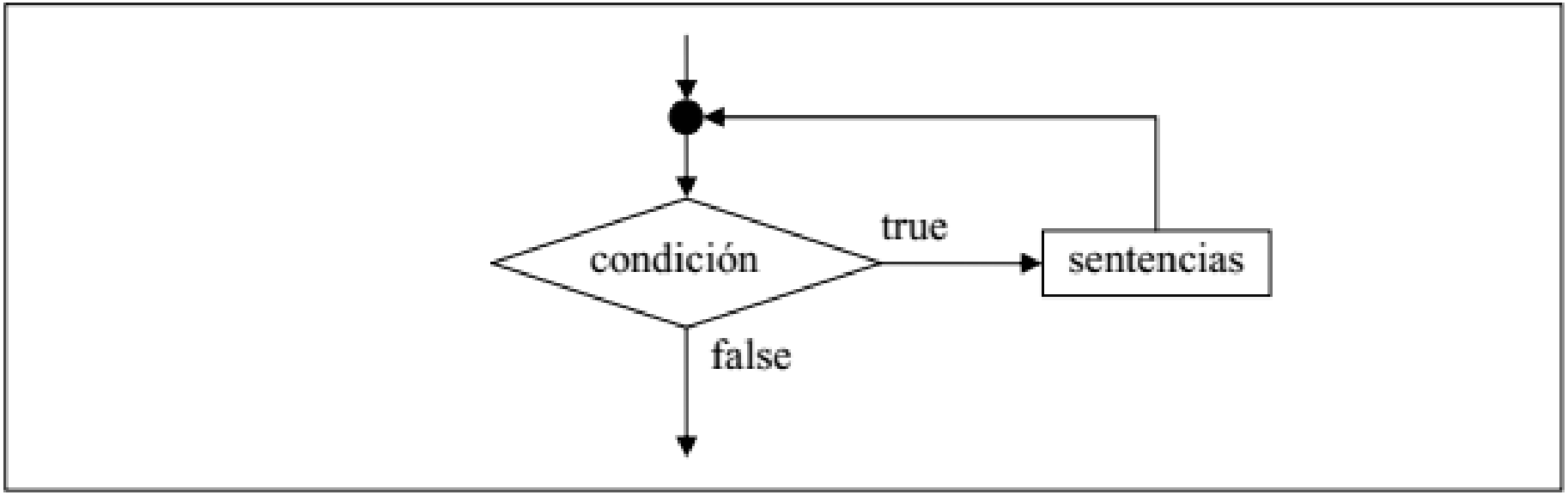
```
public void imprimirNotas()
{
      for(String nota : notas) {
            System.out.println(nota);
      }
}
```

Este *ciclo while* es equivalente al *ciclo for-each* que hemos discutido en la sección anterior. Son relevantes algunas observaciones:

- En este ejemplo, el *ciclo while* resulta un poco más complicado. Tenemos que declarar fuera del ciclo una variable para el índice e iniciarlo por nuestros propios medios en 0 para acceder al primer elemento de la lista.
- Los elementos de la lista no son extraídos automáticamente de la colección y asignados a una variable. En cambio, tenemos que hacer esto nosotros mismos usando el método `get` del `ArrayList`. También tenemos que llevar nuestra propia cuenta (índice) para recordar la posición en que estábamos.
- Debemos recordar incrementar la variable contadora (índice) por nuestros propios medios.

La sintaxis de esta sentencia es:

```
while (condición) {
    sentencias
}
```



Comparación

- for-each:
  - Más fácil de escribir
  - Más seguro: hay garantía de que acabará
- while:
  - No es necesario procesar toda la colección
  - Se puede utilizar para otras cosas que no sean colecciones
  - Pero hay que tener más cuidado: podría darse un bucle infinito

```
while (true) {
// esto no acaba nunca
}
```

```
// Print all even numbers from 0 to 30.
int index = 0;
while(index <= 30) {
    System.out.println(index);
    index = index + 2;
}
```

4.10.3 Búsqueda en una colección

Ejemplos

```
int numero = 0;
while (numero <= 30) {
      System.out.println(numero);
      numero = numero + 2;
}
```

```
int indice = 0;
boolean encontrado = false;
while (indice < notas.size() && !encontrado) {
      String nota = notas.get(indice);
      if (nota.contains(cadABuscar)) {
          encontrado = true;
      }
      else {
         indice++;
      }
}
```

```
System.out.println (n1 == n2);          //false
System.out.println (n1.equals(n2));     //true
```

Un **iterador** es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.

4.12 El tipo "Iterator"

- Es una clase de tipo genérico, no define un tipo único
- Hay que indicarle el tipo
- Está definida en el paquete *java.util*; hay que importarla

```
import java.util.ArrayList;
import java.util.Iterator;
```

Un `Iterator` provee dos métodos para recorrer una colección: `hasNext` y `next`. A continuación describimos en pseudocódigo la manera en que usamos generalmente un `Iterator`:

```
Iterator<TipoDelElemento> it = miColeccion.iterator();
while (it.hasNext()) {
   Invocar it.next() para obtener el siguiente elemento
   Hacer algo con dicho elemento
}
```

It.hasNext():
comprueba si hay mas elementos
It.next()
Obtiene el siguiente elemento

Métodos de Iterator

- hasNext()
  - Para comprobar que siguen quedando elementos en el iterador.
- next()
  - Para que nos dé el siguiente elemento del iterador.
- remove()
  - Para eliminar un elemento del iterador.

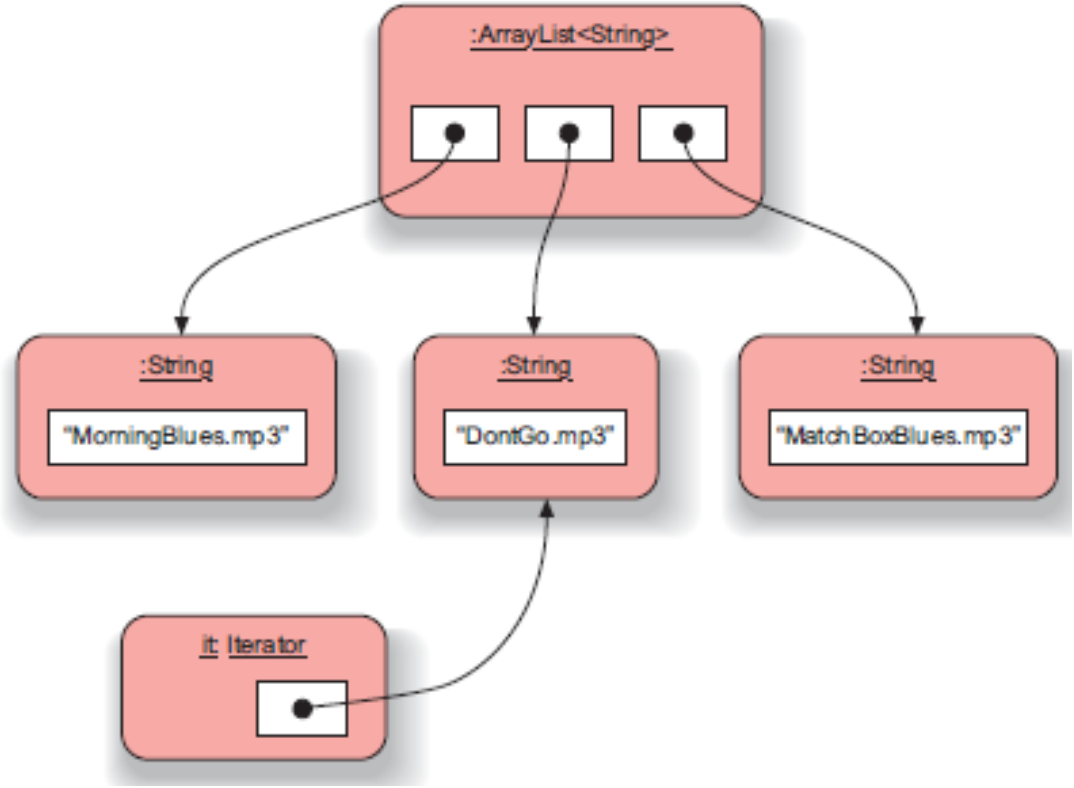
```
ArrayList<String> pelis = new ArrayList<String>();

    //Agrego cuatro peliculas
    pelis.add ("Casablanca");
    pelis.add ("39 Escalones");
    pelis.add ("La reina de Africa");
    pelis.add ("El Padrino");


// Usamos iterator para recorrer el ArrayList
Iterator<String> it = pelis.iterator();
  while (it.hasNext()){
      String elemento = it.next();
      System.out.println(elemento);
```

Comparación índice/iterator

```
/**
   * Listar todas las notas de la agenda.
   */
public void listarTodasLasNotas()
{
     Iterator<String> it = notas.iterator();
     while (it.hasNext()) {
        System.out.println(it.next());
     }
}
```

```
int indice = 0;
while(indice < notas.size()) {
   System.out.println(notas.get(indice));
   indice ++;
}
```

```java
/**
 * List all the tracks.
 */
public void listAllTracks()
{
    Iterator<Track> it = tracks.iterator();
    while(it.hasNext()) {
        Track t = it.next();
        System.out.println(t.getDetails());
    }
}
```

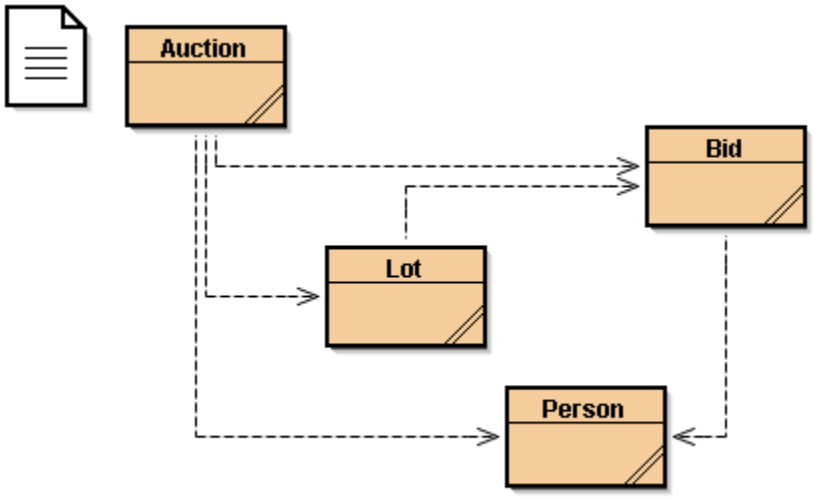


4.12 Eliminación de elementos

- No podemos eliminar un elemento de la colección en un bucle “for-each”.
- Tenemos que utilizar “Iterator”

```
Iterator<Track> it = tracks.iterator();
while(it.hasNext()) {
    Track t = it.next();
    String artist = t.getArtist();
    if(artist.equals(artistToRemove)) {
        it.remove();
    }
}
```

Ejemplo: subasta



4.14.2 La palabra reservada “null”

Se usa la palabra reservada `null` de Java para significar que «no hay objeto» cuando una variable objeto no está haciendo referencia realmente a un objeto en particular. Un campo que no haya sido inicializado explícitamente contendrá el valor por defecto `null`.

```
/**
   * Attempt to bid for this lot. A successful bid
   * must have a value higher than any existing bid.
   * @param bid A new bid.
   * @return true if successful, false otherwise
   */
  public boolean bidFor(Bid bid)
  {
     if((highestBid == null) ||
           (bid.getValue() > highestBid.getValue())) {
        // This bid is the best so far.
        highestBid = bid;
        return true;
     }
     else {
        return false;
     }
  }
```

4.14.5 Objetos “anónimos”

```
/** clase auction (subasta)
    * Enter a new lot into the auction.
    * @param description A description of the lot.
    */
   public void enterLot(String description)
   {
       lots.add(new Lot(nextLotNumber, description));
       nextLotNumber++;
}
```

Aquí estamos haciendo dos cosas:

- Creamos un nuevo objeto `Lote` y
- Pasamos este nuevo objeto al método `add` de `ArrayList`.

Podríamos haber escrito lo mismo en dos líneas de código para producir el mismo efecto pero en pasos separados y más explícitos:

```
Lote nuevoLote = new Lote(numeroDeLoteSiguiente, descripcion);
lotes.add(nuevoLote);
```

Ambas versiones son equivalentes, pero si la variable `nuevoLote` no se usa más dentro del método, la primera versión evita declarar una variable que tenga un uso tan limitado. En efecto, creamos un objeto anónimo, un objeto sin nombre, pasándolo directamente al método que lo utiliza.

4.14.6 Encadenamiento de llamadas a métodos

- Modos alternativos

```
Bid bid = lot.getHighestBid();
Person bidder = bid.getBidder();
String name = bidder.getName();
System.out.println(name);
```

```
System.out.println(lot.getHighestBid().getBidder().getName());
```

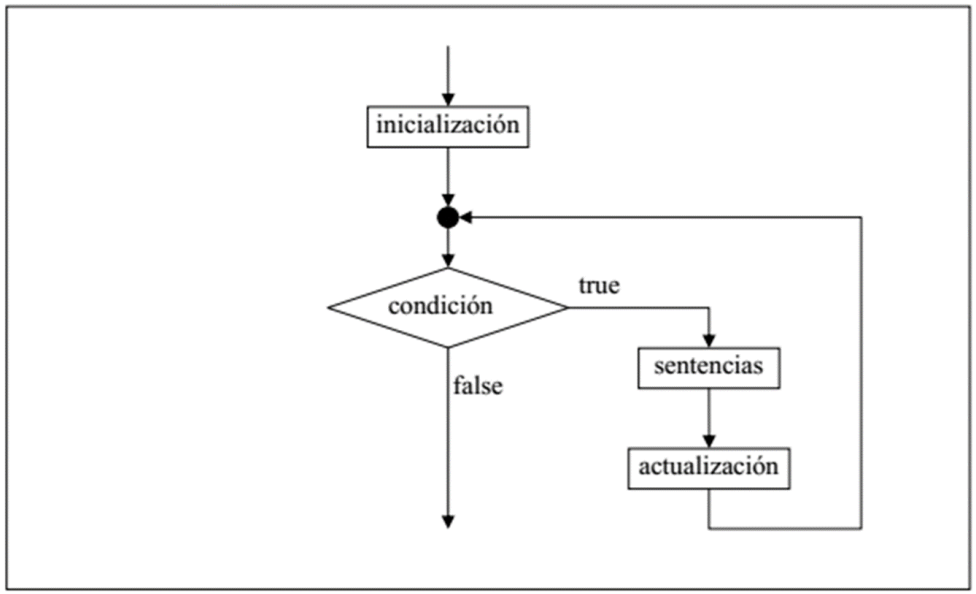
```
lot.getHighestBid().getBidder().getName()
```

4.16.6 El ciclo “for”

Un *ciclo for* tiene la siguiente forma general:

```
for (inicialización; condición; acción modificadora) {
   setencias a repetir
}
```

- La inicialización se realiza sólo una vez, antes de la primera iteración
- La condición se comprueba cada vez antes de entrar al bucle. Si es cierta, se entra. Sino, se termina.
- La actualización se realiza siempre al terminar de ejecutar la iteración, antes de volver a comprobar la condición
- Los bucles **for** son una alternativa a los bucles **while** cuando el número de repeticiones es conocido



Variable local al bucle

for loop version

```
for(int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

while loop version

```
int hour = 0;
while(hour < hourCounts.length) {
    System.out.println(hour + ": " + hourCounts[hour]);
    hour++;
}
```

Número de Línea	Código
4	`class Examen {`
5	`private int i;`
6	`public void ejemplo () {`
7	`for (int i=0; i<5; i++)`
8	`System.out.print (this.i++);`
9	`}`
10	`}`

a. Imprime 00000.
b. Imprime 01234.
c. Imprime infinitos ceros.
d. Se producirá un error en tiempo de compilación por no estar inicializada la propiedad i.

Comparación de “for” con “while” y “for-each”

```
for(int hora = 0; hora < contadoresPorHora.length; hora++)
{
                    System.out.println(hora + ": " +
contadoresPorHora[hora]);
   }
```

```
int hora = 0;
while (hora < contadoresPorHora.length) {
  System.out.println(hora + ": " + contadoresPorHora[hora]);
  hora++
}
```

```
for(int valor : contadoresPorHora) {
   System.out.println(":    " + valor(;
}
```

4.16.8 El bucle for y los iteradores

Para eliminar elementos de una colección

```
for(Iterator<Track> it = tracks.iterator(); it.hasNext(); ) {
    Track t = it.next();
    if(t.getArtist().equals(artist)) {
        it.remove();
    }
}
```

```
// Imprime múltiplos de 3 que sean menores que 40
for(int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}
```

Pregunta: Según el texto de la bibliografía básica de la asignatura, indique cuál de las siguientes afirmaciones es correcta:

a. El lenguaje Java tiene tres variantes del ciclo for : for-each, for y for-do.
b. Un ciclo while es similar en su estructura y propósito que el ciclo for-each.
c. El tipo de la variable de ciclo no tiene porqué ser el mismo que el tipo del elemento declarado para la colección que estamos recorriendo con un ciclo.
d. Un índice es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.

Pregunta: Según el texto de la bibliografía básica de la asignatura, indique cuál de las siguientes afirmaciones es correcta:

a. Las colecciones de objetos son objetos que pueden almacenar un número predeterminado e invariable de otros objetos.
b. Un iterador es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.
c. Un ciclo consiste en la escritura repetida de un bloque de sentencias.
d. Un arreglo (array) es un tipo especial de colección que puede almacenar un número variable de elementos.

Pregunta: Según el texto de la bibliografía básica de la asignatura, indique cuál de las siguientes afirmaciones es correcta en relación a la clase Vector de Java:

a. Es Final
b. Implementa java.util.List
c. Es serializable
d. Dispone de un solo constructor

¿Qué ciclo debo usar? Hemos hablado sobre tres ciclos diferentes: el *ciclo for*, el *ciclo for-each* y el *ciclo while*. Como habrá visto, en muchas situaciones el programador debe seleccionar el uso de alguno de estos ciclos para resolver una tarea. Generalmente, un ciclo puede ser rescrito mediante otro ciclo. De modo que, ¿cómo puede hacer para decidir qué ciclo usar en una situación? Ofrecemos algunas líneas guías:

- Si necesita recorrer todos los elementos de una colección, el *ciclo for-each* es, casi siempre, el ciclo más elegante para usar. Es claro y conciso (pero no provee una variable contadora de ciclo).
- Si tiene un ciclo que no está relacionado con una colección (pero lleva a cabo un conjunto de acciones repetidamente), el *ciclo for-each* no resulta útil. En este caso, puede elegir entre el *ciclo for* y el *ciclo while*. El *ciclo for-each* es sólo para colecciones.
- El *ciclo for* es bueno si conoce anticipadamente la cantidad de repeticiones necesarias (es decir, cuántas vueltas tiene que dar el ciclo). Esta información puede estar dada por una variable, pero no puede modificarse durante la ejecución del ciclo. Este ciclo también resulta muy bueno cuando necesita usar explícitamente una variable contadora.
- El *ciclo while* será el preferido si, al comienzo del ciclo, no se conoce la cantidad de iteraciones que se deben realizar. El fin del ciclo puede determinarse previamente mediante alguna condición (por ejemplo, lee una línea de un archivo (repetidamente) hasta que alcanza el fin del archivo).

```
while (expresión_booleana)
    instrucción

do
    instrucción
while (expresión_booleana)

for (instruccion_inicialización;
    expresión_booleana;
    instrucción_incremento)
    instrucción

for(ElementType elemento : colección) {
    cuerpo del bucle
}
```

`break` permite salir del ciclo

`continue` salta a la siguiente iteración

Términos introducidos en este capítulo

colección, arreglo, iterador, *ciclo for-each*, *ciclo while*, *ciclo for*, índice, sentencia import, biblioteca, paquete, objeto anónimo

Resumen de conceptos

- **colecciones** Las colecciones de objetos son objetos que pueden almacenar un número arbitrario de otros objetos.
- **ciclo** Un ciclo se usa para ejecutar un bloque de sentencias repetidamente sin tener que escribirlas varias veces.
- **iterador** Un iterador es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.
- **null** Se usa la palabra reservada de Java `null` para indicar que «no hay objeto» cuando una variable objeto no está haciendo referencia a un objeto en particular. Un campo que no ha sido inicializado explícitamente contendrá por defecto el valor `null`.
- **arreglo** Un arreglo es un tipo especial de colección que puede almacenar un número fijo de elementos.