Escuela de Ingeniería Informática

Unidad 5: Agrupar objetos

Introducción a la Programación

Curso 2015-2016

Conceptos

- **Colecciones**
  - Tamaño fijo
  - Tamaño flexible
- **Bucles**
- **iteradores**

Introducción

- Dar a conocer algunas maneras en las que pueden agruparse los objetos para formar **colecciones**.

- **Colección**. *Es una **estructura de datos** que permite **almacenar** diversos elementos y **acceder** a ellos.*

- **Las colecciones de objetos.** Son **objetos** que pueden **almacenar** un **número arbitrario de objetos**.

Introducción

- Se verán dos modalidades:
  - Colecciones de tamaño flexible ⟹ `ArrayList`
  - Colecciones de tamaño fijo ⟹ `vectores y arrays`

Colecciones de tamaño flexible (I)

- Cuando escribimos programas necesitamos **agrupar los objetos en colecciones**.
  - Las agendas electrónicas guardan notas sobre citas, reuniones, tareas, fechas de cumpleaños, etc.
  - Las bibliotecas registran detalles de los libros y revistas que poseen.
  - Las universidades mantienen registros del expediente académico de los estudiantes.
  - Los talleres mantienen un registro con información de los coches que reparan.

Colecciones de tamaño flexible (II)

- El número de elementos almacenados en la colección varía a lo largo del tiempo.
- ¿Cual sería la solución para almacenar un número arbitrario de objetos?
  - Definir una sola clase con una gran cantidad de atributos individuales.
  - Una solución que nos permita establecer un límite mayor que el número de elementos a almacenar.
  - Una solución que no requiera conocer anticipadamente la cantidad de elementos que queramos almacenar.

Bibliotecas de clases

- Una característica de los LOO es que frecuentemente están acompañados de **bibliotecas de clases**.
- Contienen varios cientos o miles de clases diferentes que sirven de ayuda a los desarrolladores en diferentes proyectos.
- Java cuenta con varias bibliotecas y las denomina paquetes (packages)
  - La clase `ArrayList` está definida en el paquete `java.util` y es una clase *colección*

Ejemplo: Una agenda personal

- Modelar una aplicación que represente una agenda personal con las siguientes características:
  - Permite almacenar notas.
  - El número de notas que se puede almacenar no tienen límite.
  - Mostrará las notas de manera individual.
  - Nos informará sobre la cantidad de notas que tiene actualmente almacenada.

Ejemplo: Una agenda personal

```
import java.util.ArrayList;

public class Agenda
{
  private ArrayList<String> notas;

  public Agenda()
  {
    notas = new ArrayList<String>();
  }
}
```

Sentencia import

Tipo del atributo notas

Se crea un objeto de tipo `ArrayList<String>`

Ejemplo: Una agenda personal

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

- Muestra el modo en que obtenemos el acceso a una clase de una biblioteca de Java mediante la `sentencia import`
- Hace que la clase `ArrayList` del paquete `java.util` esté disponible para nuestra clase.
- Se deben colocar siempre antes de la definición de la clase.
- Se usa de la misma forma que las clases propias.

Ejemplo: Una agenda personal

```
private ArrayList<String> notas;
```

- Cuando se usan colecciones hay que especificar dos tipos, el propio de la colección (`ArrayList`) y el tipo de los elementos que se van a almacenar en la colección (`String`)
- Se lee `ArrayList de String`.

Ejemplo: Una agenda personal

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

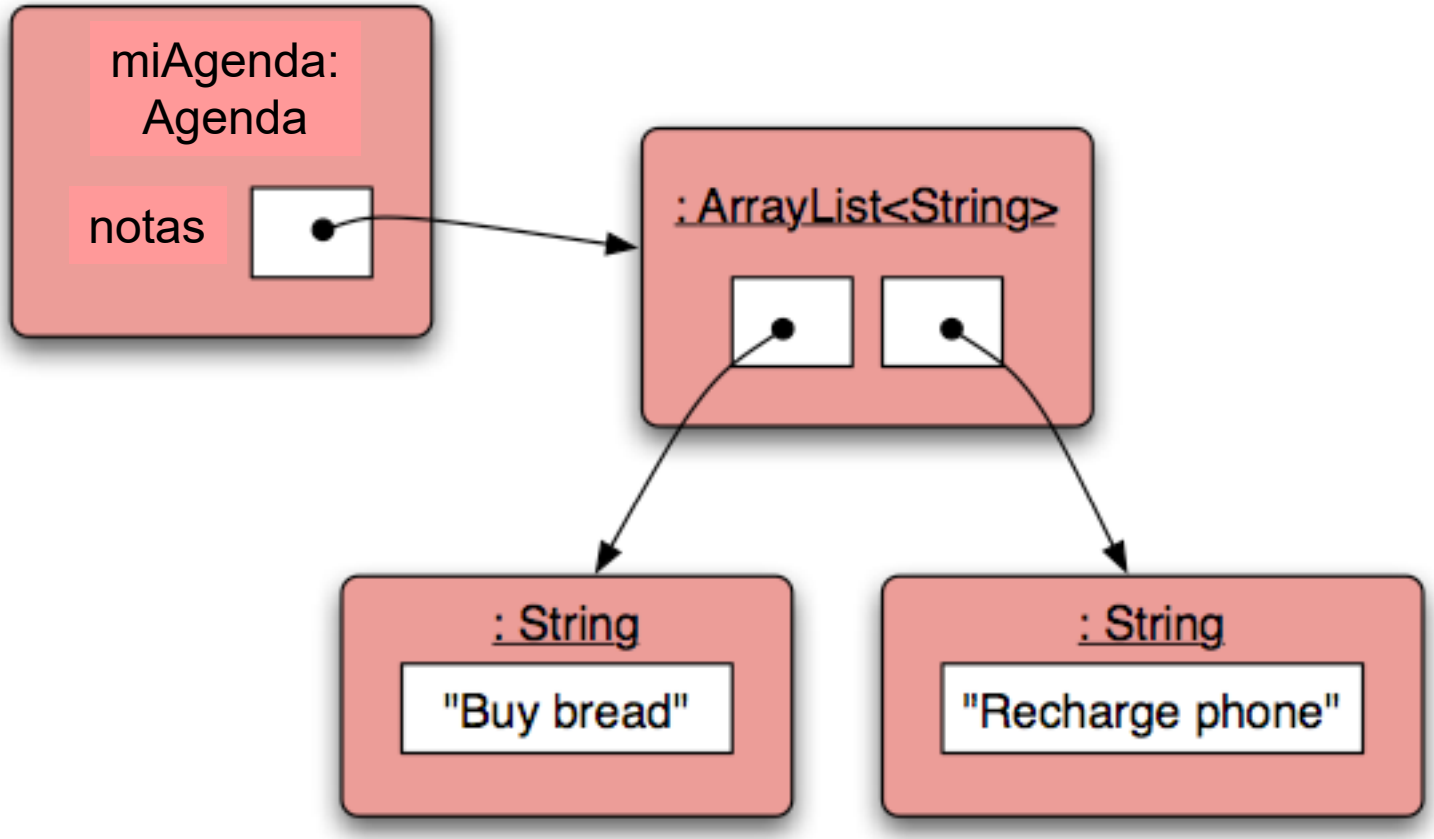
- Se crea un objeto de tipo `ArrayList<String>` y se almacena en la propiedad `notas` la referencia a dicho objeto.
- Se especifica el tipo completo
- La lista de parámetros está vacía.

Ejemplo: Una agenda personal

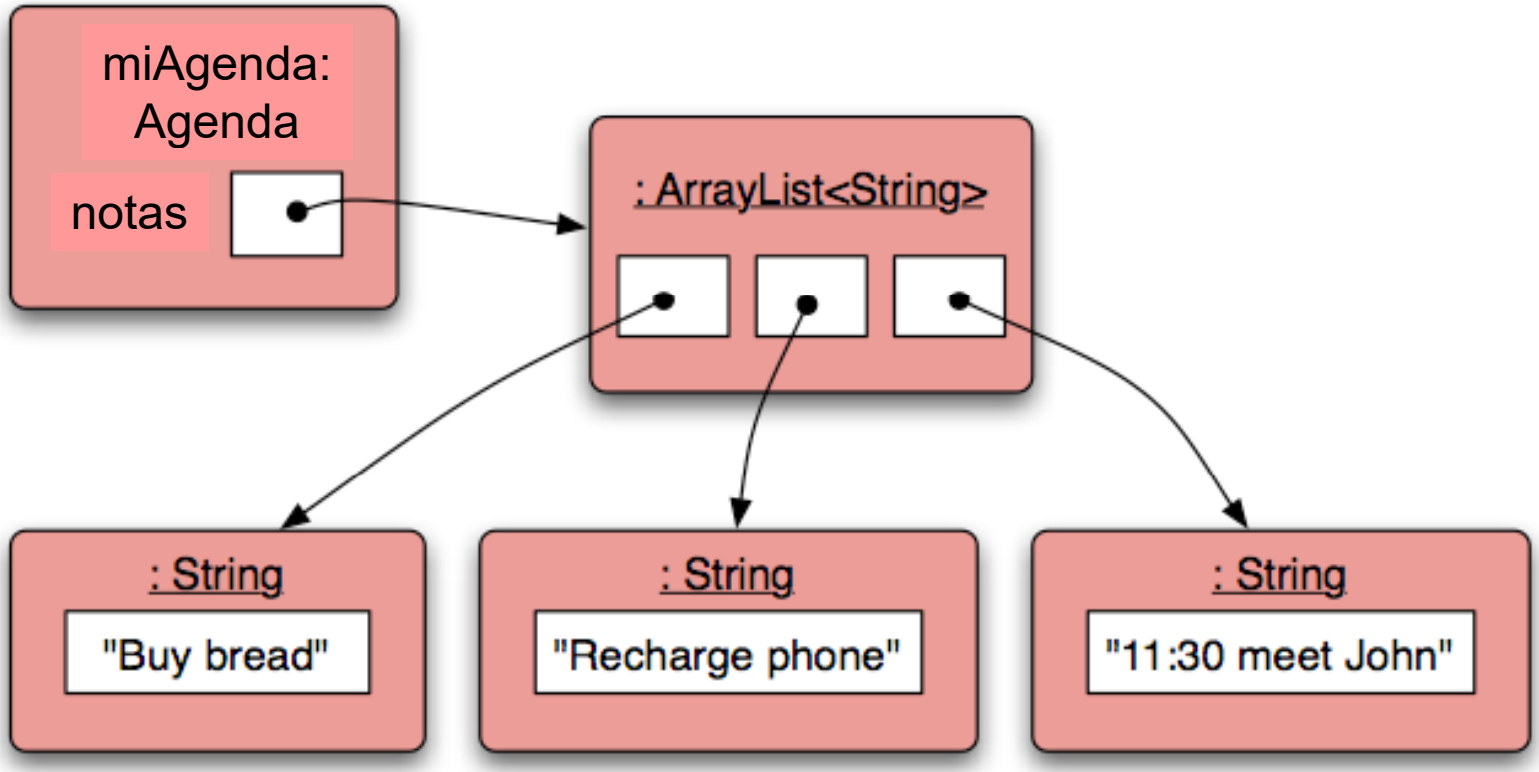
- Las clases similares a `ArrayList` que se parametrizan con un segundo tipo, se denominan **clases genéricas**.
- La clase `ArrayList` contiene muchos métodos pero en este ejemplo solo se usan algunos: `add`, `size` y `get`

```
// Almacena una nota nueva en la agenda
public void guardarNota (String nota)
{
    notas.add(nota);
}
```

Estructuras de objetos con colecciones



Añadiendo una tercera nota



Características de la clase ArrayList

- Es capaz de aumentar su capacidad interna a medida que se va necesitando.
- Mantiene un contador privado del número de elementos que tiene actualmente almacenados. Su método: `public int size()` devuelve el número de objetos que contiene actualmente.
- Mantiene el orden de los elementos que se añaden para que se puedan recuperar en el mismo orden posteriormente.
- Los detalles de cómo se realiza todo quedan ocultos.

Usando la colección

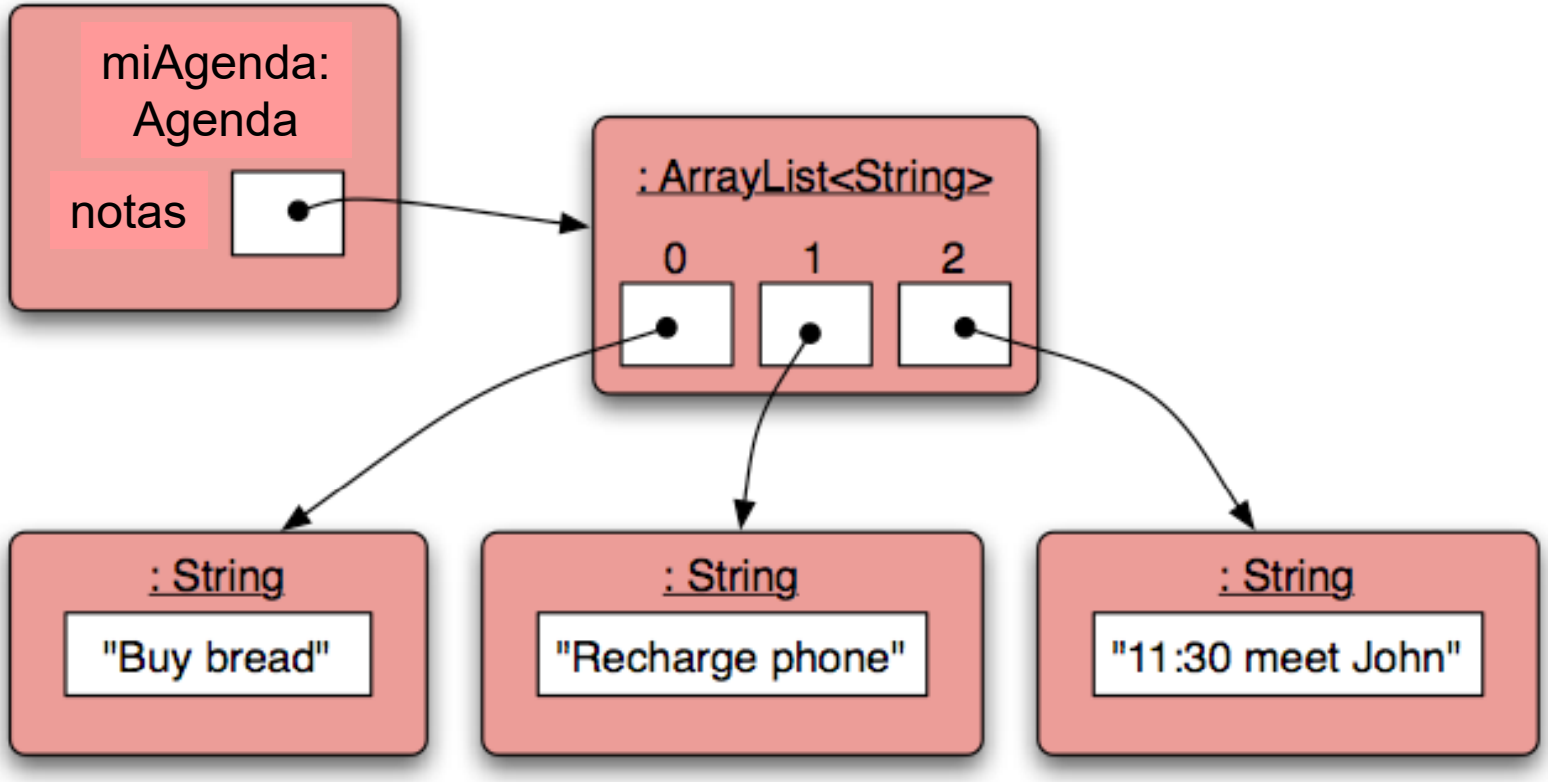
```
// Almacena una nueva nota en la agenda
public void guardarNota(String nota)
{
    notas.add(nota);
}
```

Añade una nueva nota

```
// Devuelve el número de notas que tiene la agenda
public int numeroDeNotas()
{
   return notas.size();
}
```

Devuelve el número de notas (*delegation*)

Numeración dentro de las colecciones



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 es conocida más comúnmente como su **índice**.
  - El primer elemento que se añade tiene índice 0
  - El segundo elemento que se añade tiene índice 1
  - ...

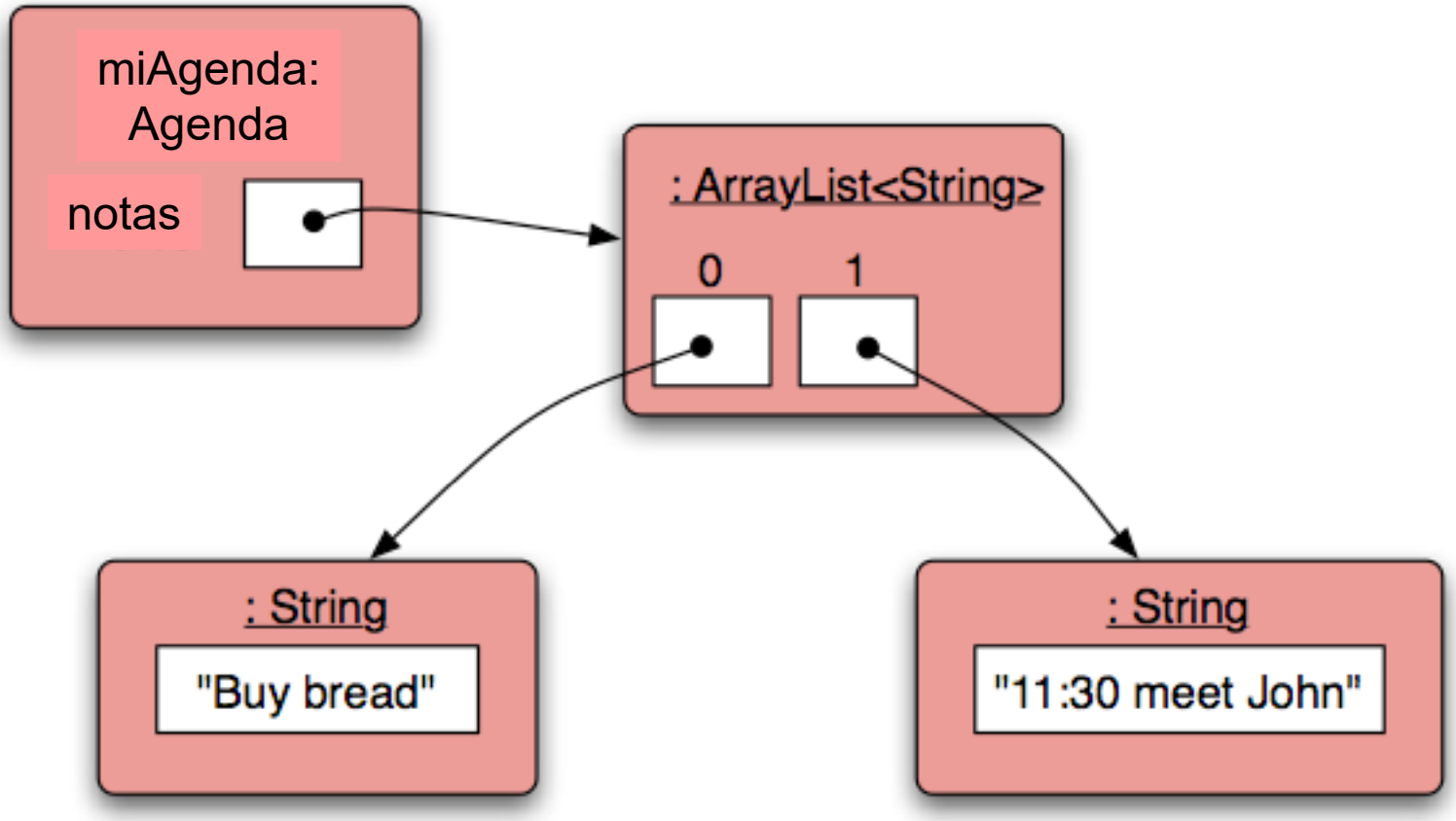
Usando la colección

Comprobación de índice

```
// Muestra una nota en pantalla
public void mostrarNota(int numeroDeNota)
{
    if(numeroDeNota < 0) {
        // No es un número de nota válido
    }
    else if(numeroDeNota < numerodeNotas()) {
     System.out.println(notas.get(numeroDeNota));
    }
    else {
        // No es un número de nota válido
    }
}
```

Recuperar y visualizar la nota

Eliminar un elemento de una colección



Eliminar un elemento de una colección

- La clase `ArrayList` tiene un método borrar: `public void remove(int index)` que toma como parámetro el índice de la nota que será eliminada.
- Una complicación del proceso de eliminación es que se modifican los valores de los índices del resto de las notas almacenadas en la colección.
  - Si se elimina una nota con índice 0, la colección desplaza los siguientes elementos una posición a la izquierda para rellenar el hueco. Sus índices disminuyen un aposición.
- También es posible añadir elementos en cualquier posición no solo al final de la colección. Los elementos deben incrementar una posición

Usando la colección

Comprobación de índice

```
// Elimina una nota de la agenda por su índice
public void eliminarNota(int numeroDeNota)
{
    if(numeroDeNota < 0) {
        // No es un número de nota válido
    }
    else if(numeroDeNota < numerodeNotas()) {
      notas.remove(numeroDeNota);
    }
    else {
        // No es un número de nota válido
    }
}
```

Elimina la nota de esa posición

Ejercicios

- Escriba una llamada al método correspondiente de la clase `ArrayList` para eliminar el tercer objeto almacenado en una colección de nombre **notas**.
- Suponga que un objeto está almacenado en una colección bajo el índice 6. ¿Cuál será su índice inmediatamente después de que se eliminen los objetos de las posiciones 0 y 9 por este orden?

Clases genéricas (I)

```
ArrayList<String>
```

- La clase que estamos usando se denomina `ArrayList` y se debe especificar un segundo tipo como parámetro cuando se usa para declarar campos u otras variables.
- Las clases que requieren este tipo de parámetro, se denominan **clases genéricas**.
  - No definen un tipo único sino potencialmente muchos tipos (`ArrayList de Strings`, `ArrayList de Personas`, `ArrayList de Aviones`,…).

Clases genéricas (II)

```
private ArrayList<Persona> delegaciónEII;
private ArrayList<MaquinaDeBoletos> maquinasEstación;
```

- `ArrayList<Persona>` y `ArrayList<MaquinaDeBoletos>` son dos tipos diferentes.
- Los atributos no pueden ser asignados uno a otro, aun cuando sus tipos deriven de la misma clase.
- **Ejercicio**. Escriba la declaración de una propiedad privada de nombre `biblioteca` que pueda contener un `ArrayList`. Los elementos del `ArrayList` son de tipo `Libro`.

Revisión (I)

- Las colecciones permiten almacenar un número arbitrario de elementos.
- Las bibliotecas de clases contienen colecciones de clases probadas.
- Las bibliotecas de clases en java se denominan *packages*.
- Hemos usado la clase `ArrayList` del package `java.util`.

Revisión (II)

- Se pueden añadir y eliminar elementos de una colección.
- Cada elemento tienen un índice.
- Los valores de los índices pueden cambiar si los elementos son eliminados (o se añaden nuevos elementos).
- Los principales métodos de la clase `ArrayList` son: `add`, `get`, `remove` y `size`.
- `ArrayList` es una clase parametrizada, define un tipo genérico.

Algunos errores populares

El siguiente código tiene un error

```
/**
 * Visualiza cuantas notas están almacenadas en la
 * agenda
 */
public void showStatus()
{
    if(notas.size() == 0);
      System.out.println("La agenda está vacía");
    else
    {
        System.out.print("la agenda tiene:");
        System.out.println(notas.size() + " notas");
    }
}
```

Versión correcta

```
/**
 * Visualiza cuantas notas están almacenadas en la
 * agenda
 */
public void showStatus()
{
    if(notas.size() == 0)
        System.out.println("La agenda está vacía");
    else
    {
        System.out.print("la agenda tiene:");
        System.out.println(notas.size() + " notas");
    }
}
```

Algunos errores populares

El siguiente código tiene un error

```
/**
 * Visualiza cuantas notas están almacenadas en la
 * agenda
 */
public void showStatus()
{
    if(notas.isEmpty() = true)
      System.out.println("La agenda está vacía");
    else
    {
      System.out.print("la agenda tiene:");
      System.out.println(notas.size() + " notas");
    }
}
```

Versión correcta

```
/**
 * Visualiza cuantas notas están almacenadas en la
 * agenda
 */
public void showStatus()
{                                    // if (notas.isEmpty())
  if(notas.isEmpty() == true)
     System.out.println("La agenda está vacía");
  else
   {
      System.out.print("la agenda tiene:");
      System.out.println(notas.size() + " notas");
   }
}
```

Algunos errores populares

El siguiente código tiene errores

```
/**
 * Añade una nueva nota a la agenda. Si la agenda
 * está llena, trata los elementos y crea una nueva.
 */
public void añadirNota(String nota)
{
    if(notas.size == 100)

      // tratar los elementos
      // crea una nueva agenda
      notas = new ArrayList<String>();

    notas.add(nota);
}
```

Versión correcta

```
/**
 * Añade una nueva nota a la agenda. Si la agenda
 * está llena, trata los elementos y crea una nueva.
 */
public void añadirNota(String nota)
{
    if(notas.size() == 100)
    {
      // tratar los elementos
      // crea una nueva agenda
      notas = new ArrayList<String>();
    }
    notas.add(nota);
}
```

Procesar una colección completa

- Como se pueden añadir y eliminar elementos, a veces es necesario listar todos los elementos con sus índices actuales.

- **Ejemplo**. ¿Como debería ser la signatura del método `ListarTodasLasNotas`? ¿Debe tener algún parámetro?

- ¿Qué harían las siguientes sentencias?

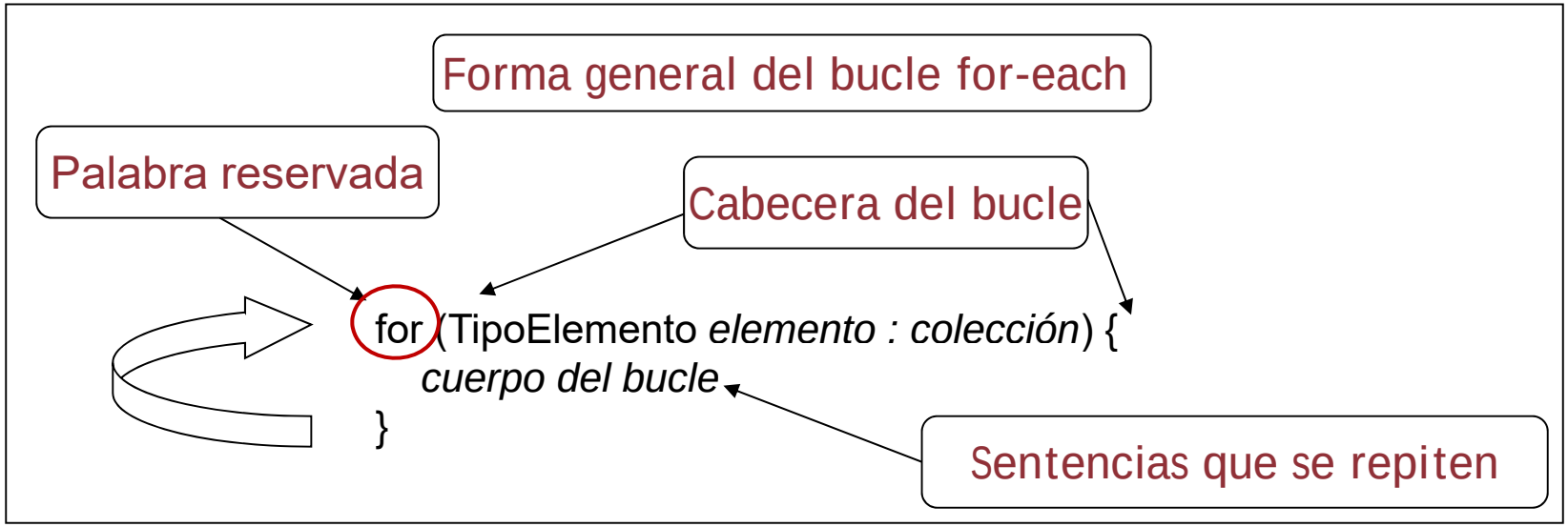
```
System.out.println(notas.get(0));
System.out.println(notas.get(1));
System.out.println(notas.get(2));
...
```

- ¿Cuantas sentencias se necesitarían para listar todas las notas?

El bucle **for-each** *(loop statement)*

- Un ciclo o **bucle** puede usarse para ejecutar repetidamente un bloque de sentencias sin tener que escribirlas varias veces.
- Cuando se debe usar un bucle
  - Si se necesita repetir algunas acciones una y otra vez.
  - Si es necesario controlar cuantas veces es necesario repetir las acciones.
  - Con colecciones, a veces es necesario repetir acciones para cada uno de los objetos de una colección en particular.

Bucle for-each en pseudocódigo



Pseudocódigo de las acciones de un bucle for-each

Para cada elemento **en la** colección, ejecutar las sentencias del cuerpo del bucle

Ejemplo de uso - bucle for-each

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

Variable local. Se usa para contener los elementos de la lista

Para cada *nota* **en la** colección *notas*, imprimir *nota*

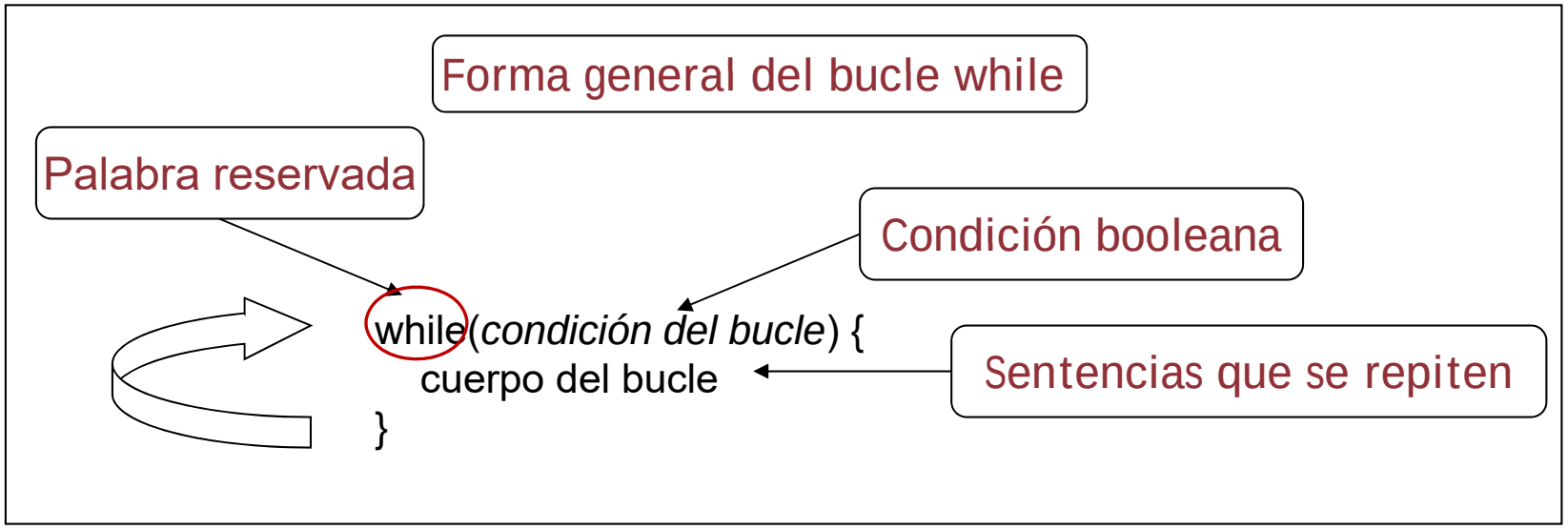
Cada elemento de la colección es asignado a la variable `nota` y para cada una de estas asignaciones, el cuerpo del bucle se ejecuta una sola vez.

Ejercicio

- Defina la clase `Teléfono` y use la biblioteca `ArrayList` para almacenar números de teléfono.
- Implemente el método `añadirContacto` para añadir un nuevo contacto (solo su teléfono)
- Implemente los métodos `mostrarTeléfono` y `eliminarTeléfono` dada una posición. Se debe visualizar un mensaje de error si la posición no es válida.
- Implemente el método `imprimirContactos` para mostrar los teléfonos de los contactos.

Bucle **while** en pseudocódigo

- Es similar al bucle **for-each** aunque más flexible



Pseudocódigo de las acciones de un bucle while

Mientras sea **verdadera la condición** del bucle, ejecutar las sentencias del cuerpo del bucle

Ejemplo de uso - bucle while

```
/**
 * Lista todas las notas de la agenda.
 */
public void listarNotas()
{
    int index = 0;
    while(index < notas.size()) {
        System.out.println(notas.get(index));
        index++;
    }
}
```

Variable local. Se usa para recorrer las posiciones de la colección

Incrementa *index* una unidad

Mientras el valor de *index* sea menor que el tamaño de la colección, imprimir la nota, y luego incrementar *index*

Ejercicio

- Modifique el método `imprimirContactos` que muestra los teléfonos de los contactos y sustituya el bucle for-each por el bucle while.
- Modifique el método `imprimirContactos` para que muestre solo los teléfonos que están almacenados en una posición par.

for-each versus **while**

- **for-each**:
  - Es fácil de escribir
  - Siempre se garantiza que el bucle para, no hay bucles infinitos
- **while**:
  - Permite que la colección se procese parcialmente
  - Este bucle no se usa solo en colecciones
  - Es necesario ser cuidadosos y no escribir bucles infinitos

Ejemplo

```
/**
 * Lista todas las notas de la agenda.
 */
public void listarNotas()
{
    int index = 0;
    while(index < notas.size()) {
        System.out.println(notas.get(index));
    }
}
```

¡ Bucle infinito !

Ejercicio

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

¿ Qué hace el siguiente bloque ?

Ejemplo. Buscar en una colección

```
int index = 0;
String palabra = "cita";
boolean encontrado = false;
while(index < notas.size() && !encontrado) {
    String nota = notas.get(index);
    if(nota.contains(palabra)) {
        encontrado = true;
    }
    else {
        index++;
    }
}
// El bucle para cuando encuentra una nota que
// contiene la palabra buscada o cuando se acaba de
// recorrer la colección.
```

Iteradores

- Es una forma especial de recorrer una colección.
- Un iterador es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.
- El método `iterator` de la clase `ArrayList` devuelve un objeto de la clase `Iterator`
- La clase `Iterator` está definida en el paquete `java.util`

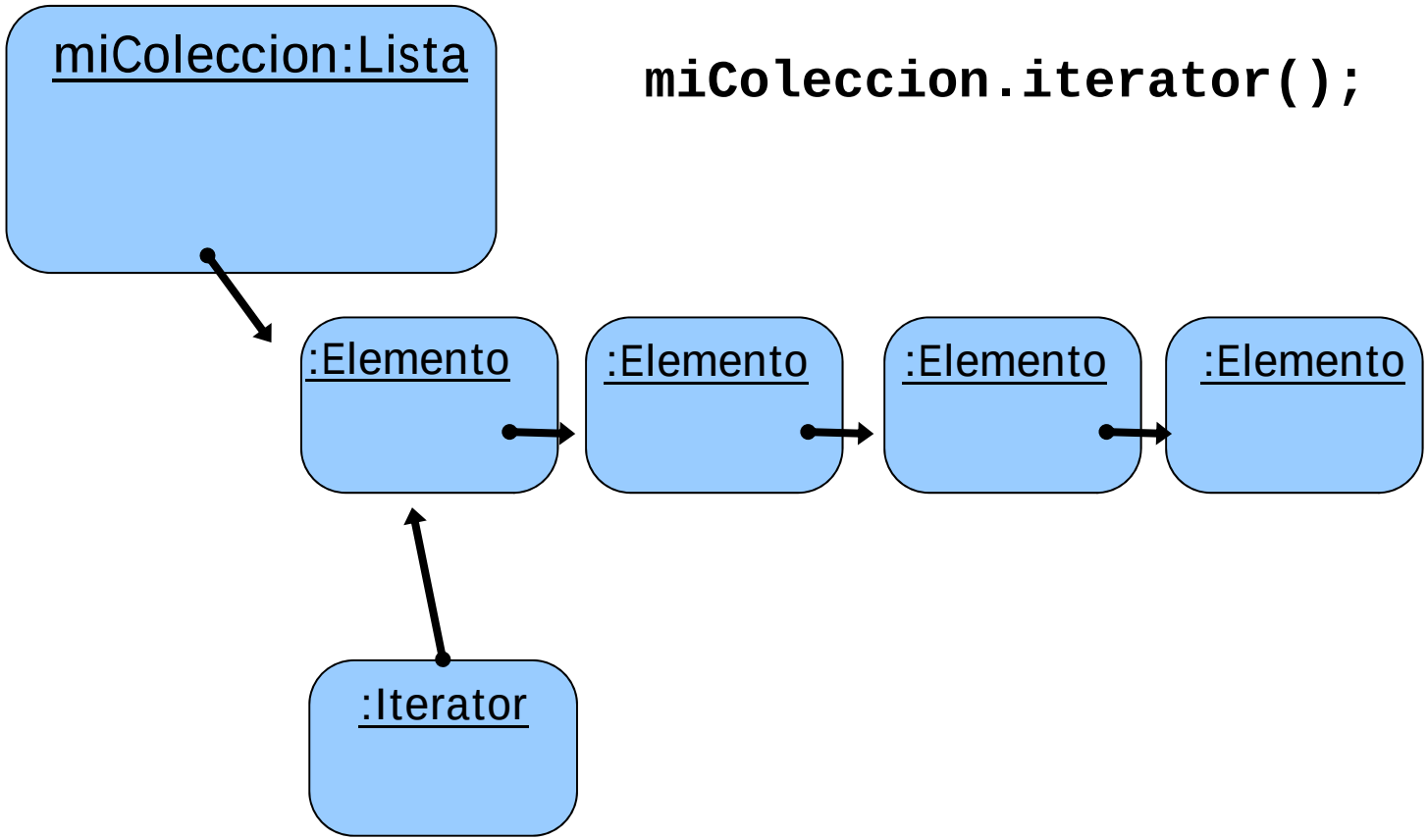
Usando un objeto Iterator

java.util.Iterator

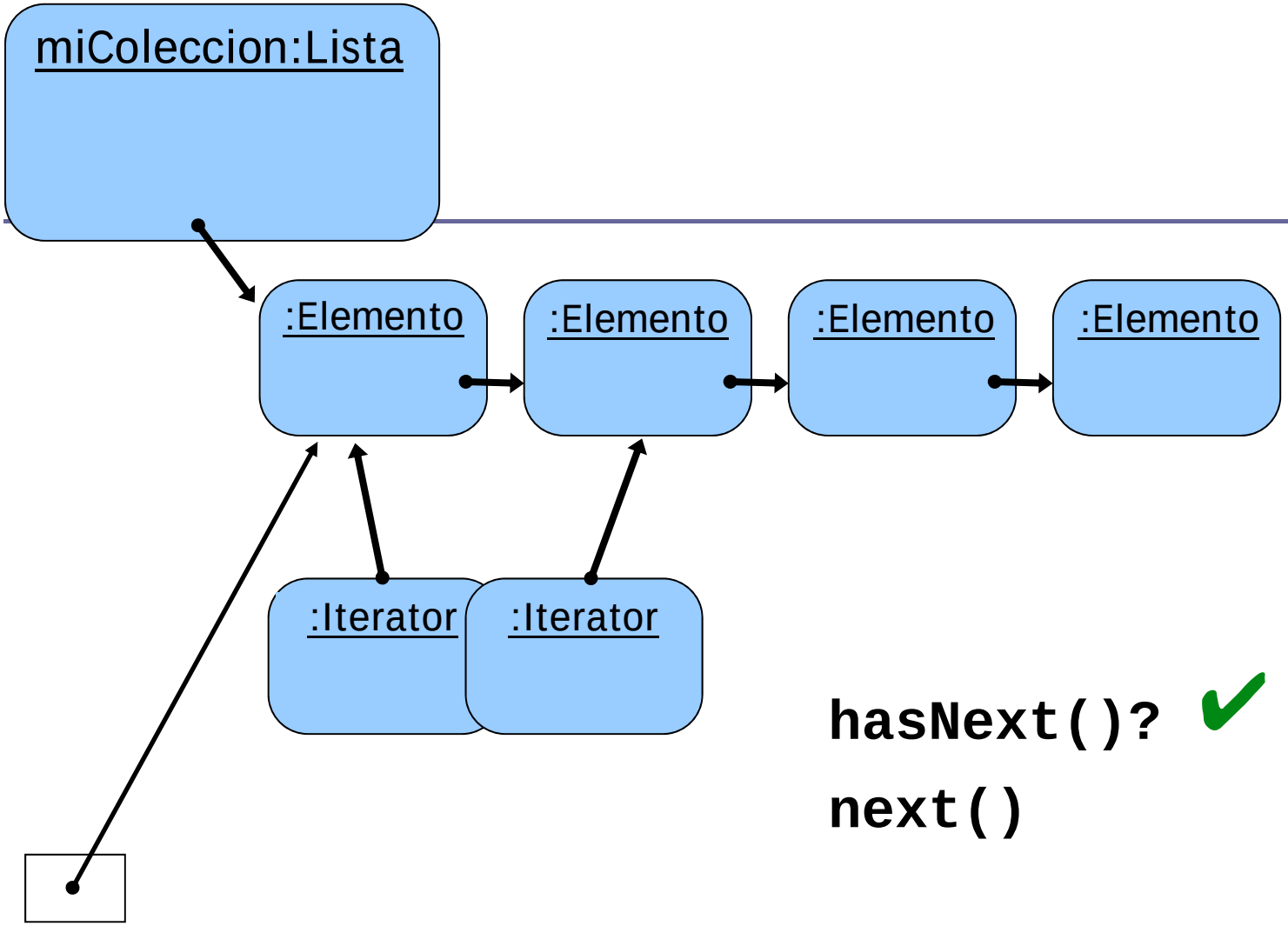
Devuelve un objeto Iterator

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

Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
   // Llamar it.next() para obtener el siguiente
   // elemento (objeto de la colección)
   // Hacer algo con dicho objeto
}
```

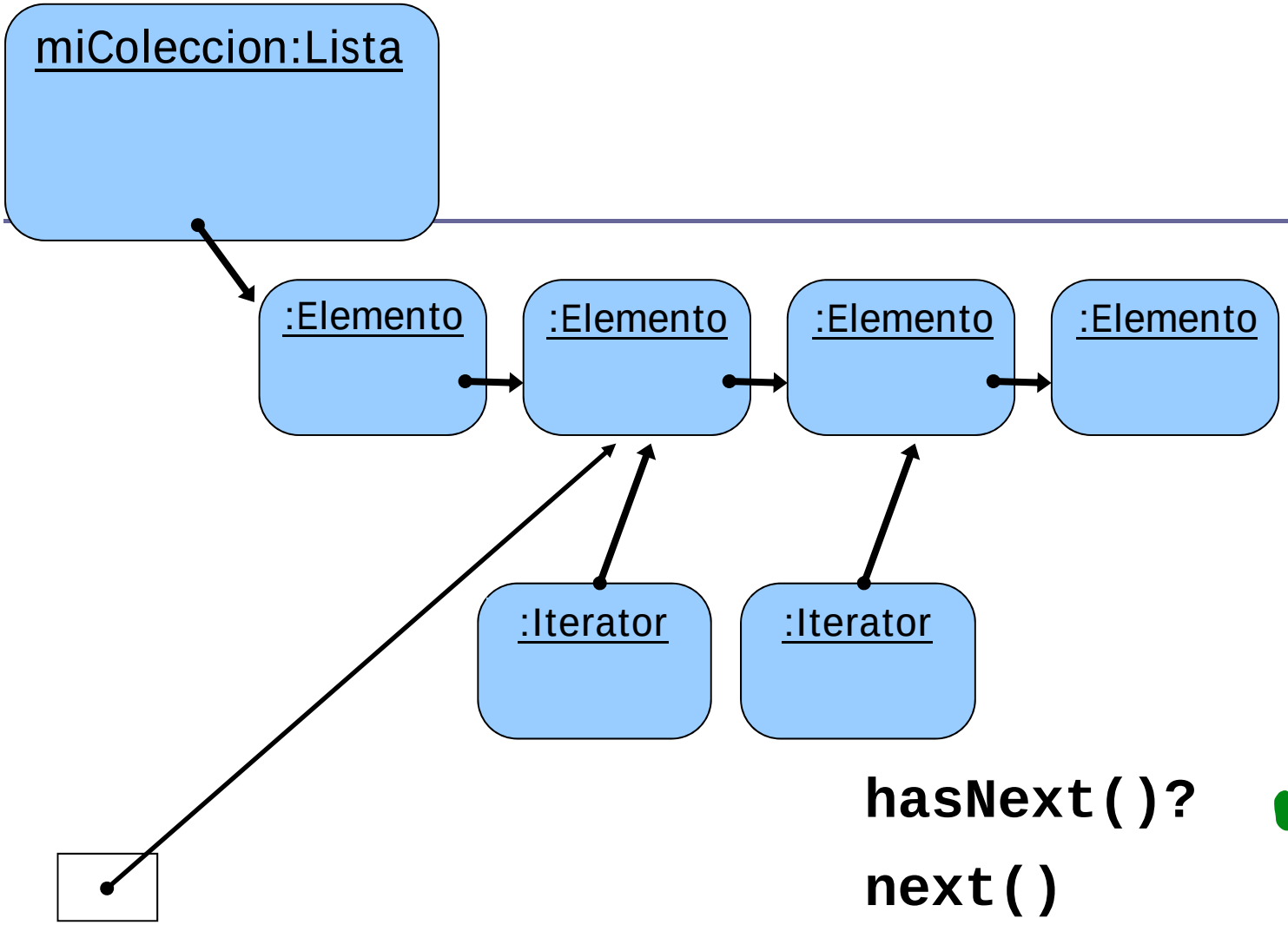
Iteradores



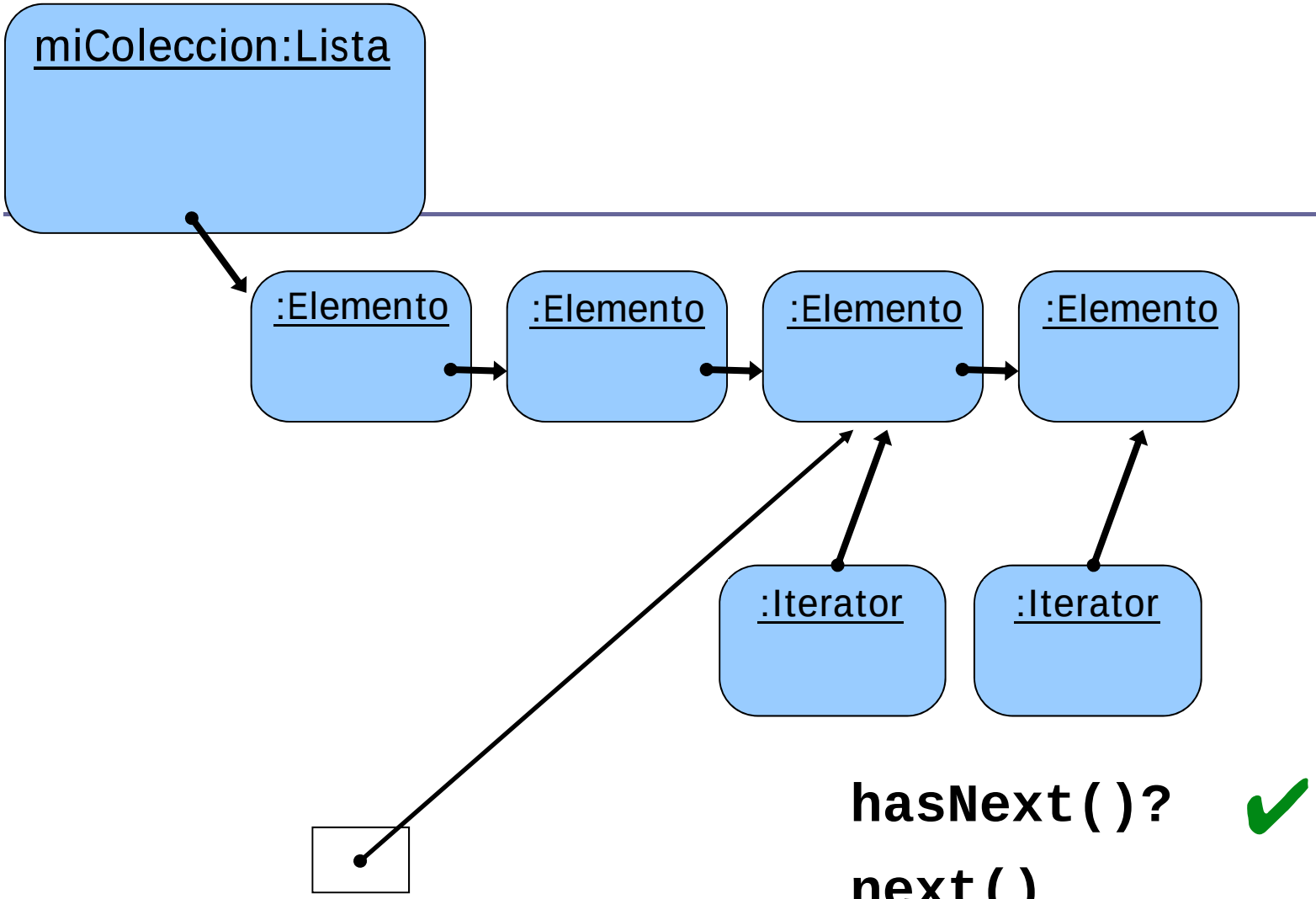

```
miColeccion.iterator();
```



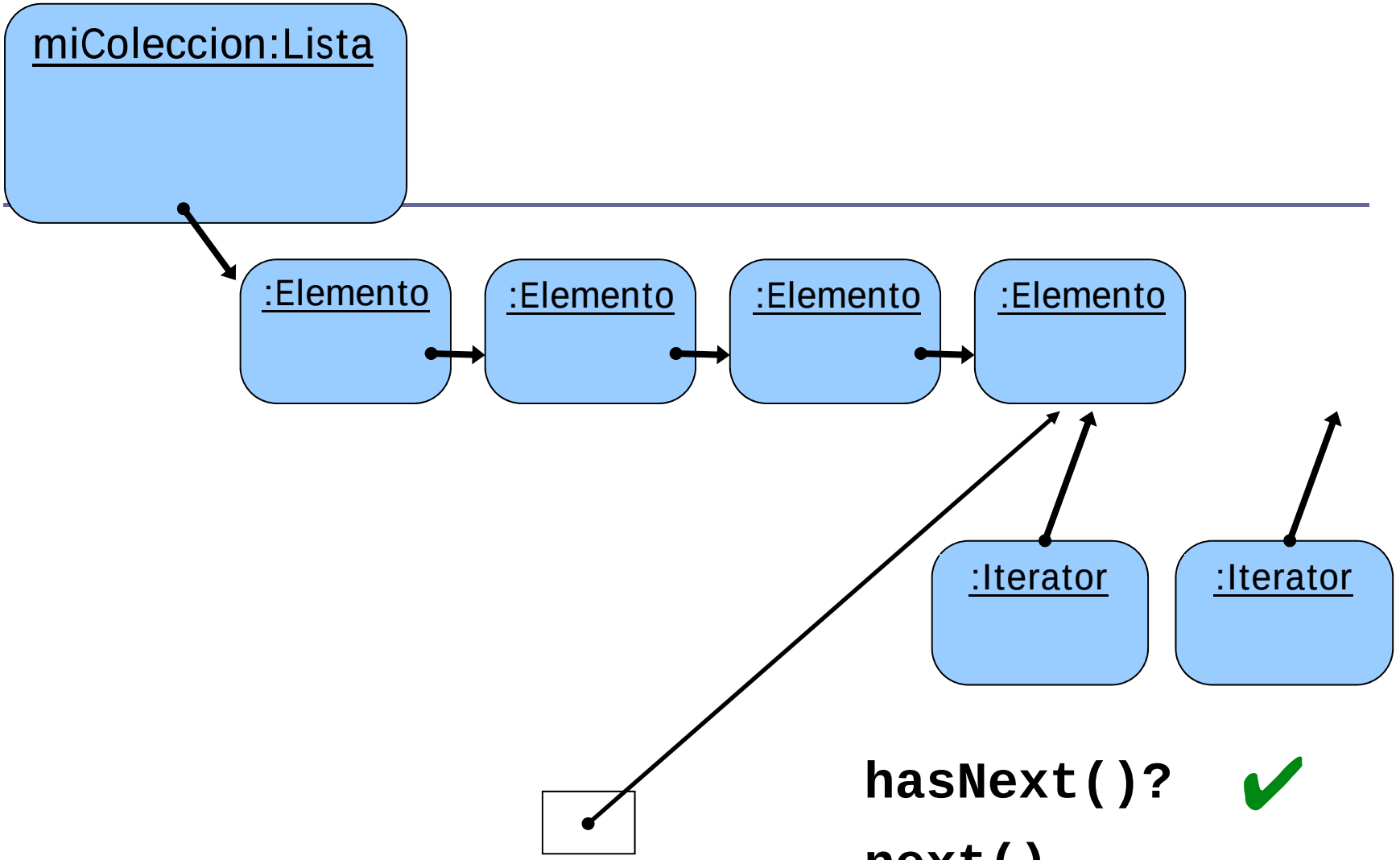

```
Elemento e = iterator.next();
```



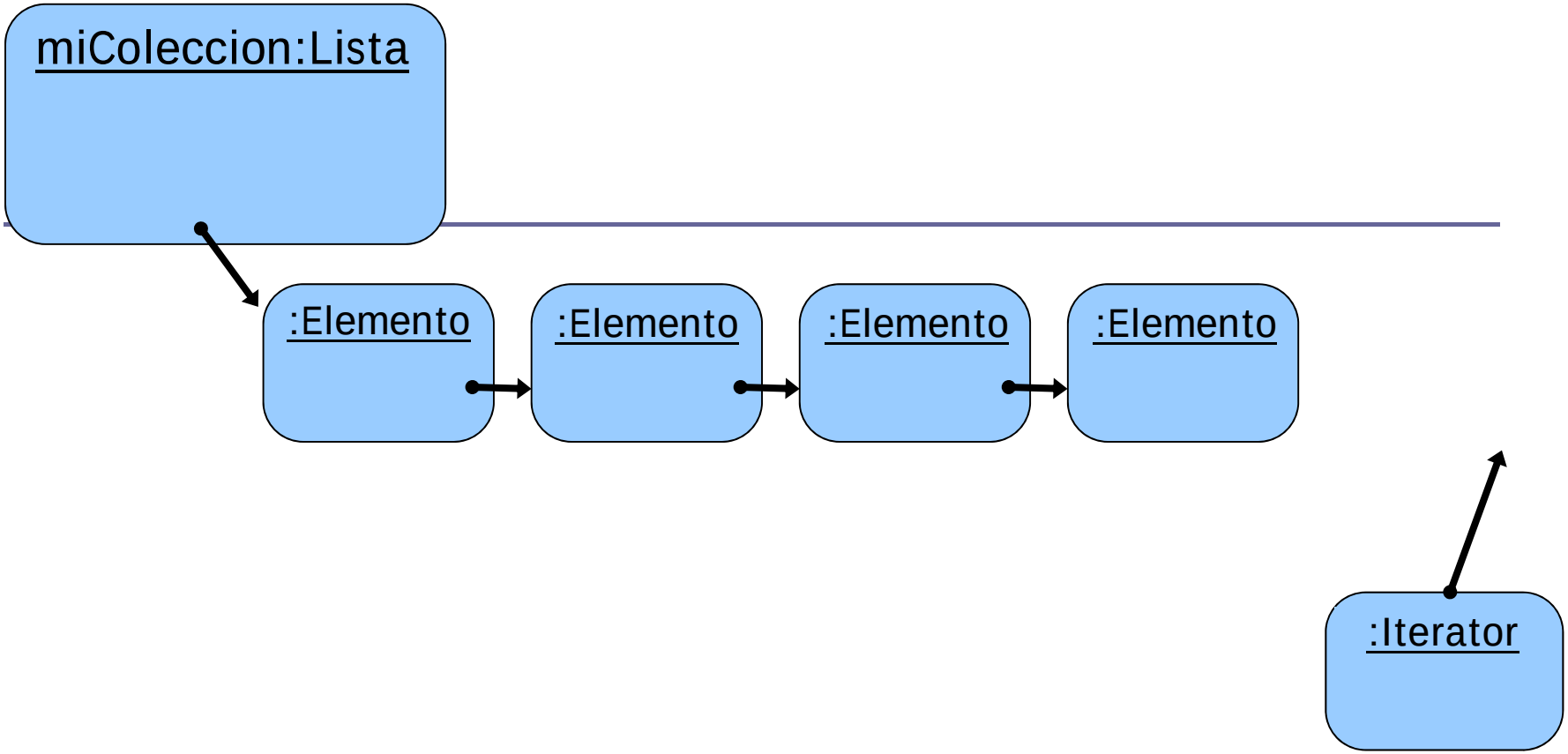

```
Elemento e = iterator.next();
```




```
Elemento e = iterator.next();
```




```
Elemento e = iterator.next();
```



`hasNext()?` ✗

Usando un objeto Iterator

java.util.Iterator

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

Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
// Llamar it.next() para obtener el siguiente objeto
// Hacer algo con dicho objeto
}
```

Devuelve un objeto Iterator

```
    public void listarNotas()
    {
        Iterator<String> it = notas.iterator();
        while(it.hasNext()) {
            System.out.println(it.next());
        }
    }
```

Formas de recorrer una colección

- **for-each**
  - Usar si queremos procesar todos los elementos.
- **while**
  - Usar si queremos parar de procesar antes de llegar al final o procesar solo ciertos elementos.
  - Usar para repetir la ejecución de un bloque de sentencias. Sin usar colección.
- **Iterator object**
  - Usar si queremos parar de procesar antes de llegar al final o procesar solo ciertos elementos.
  - A menudo se usa con colecciones donde el acceso indexado no es muy eficiente o es imposible.

- Un iterador está disponible para todas las colecciones de las clases de las bibliotecas de Java. Iteración es un patrón importante de programación

Revisión

- El **bucle while** permite que un bloque de sentencias sea ejecutado varias veces hasta que la condición sea falsa.
- El **bucle for-each** procesa todos los elementos de una colección.
- El **bucle while** permite que la repetición en una colección, sea controlada por una expresión booleana.
- **Todas las colecciones** de clases proporcionan objetos `Iterator` especiales que permiten el acceso secuencial a todos los elementos de la colección.

Objetos anónimos (I)

Clase con dos atributos nombre y teléfono

```
private ArrayList<Contacto> contactos;

    public Telefono()
    {
       contactos = new ArrayList<Contacto>();
    }
```

Para añadir un nuevo contacto se puede hacer de dos formas:

```
Contactos.add (new Contacto("pepe", "12121212"));
```

```
Contacto nuevoContacto = new Contacto(("pepe", "12121212"));

contactos.add(nuevoContacto);
```

Objetos anónimos (II)

```
Contactos.add (new Contacto("pepe", "12121212"));
```

Se hacen dos cosas:

- Se crea un nuevo objeto `Contacto`
- Se pasa ese nuevo objeto al método add de `ArrayList`

```
Contacto nuevoContacto = new Contacto(("pepe", "12121212"));

contactos.add(nuevoContacto);
```

Si la variable `nuevoContacto` no se usa más dentro del método, la primera versión evita declarar una variable que tenga un uso tan limitado. **Es mejor crear un objeto anónimo**.

Colecciones de tamaño fijo

- Algunas veces, el tamaño máximo de una colección puede ser pre-determinado.
- Los lenguajes de programación ofrecen usualmente una colección especial de tamaño fijo denominada array (o arreglo).
- Los arrays usan una sintaxis especial
- Ventajas:
  - El acceso a los **elementos de un array** es generalmente más eficiente que el acceso a los elementos de una colección de tamaño flexible.
  - Los **arrays** son capaces de almacenar objetos o valores primitivos. Las **colecciones de tamaño flexible** solo pueden almacenar objetos.

Analizador de un archivo de registro - log

- Los servidores web, mantienen archivos de registro (fichero log) de los accesos de los clientes a las páginas web que almacenan.
- Las tareas que puede realizar un webmaster son:
  - Determinar cuales son las páginas más populares.
  - Los periodos de mayor acceso durante un día, una semana o un mes.
  - Cuantos datos se transmiten a los clientes.
  - Si se rompieron enlaces a las páginas del servidor.

Creando un objeto array

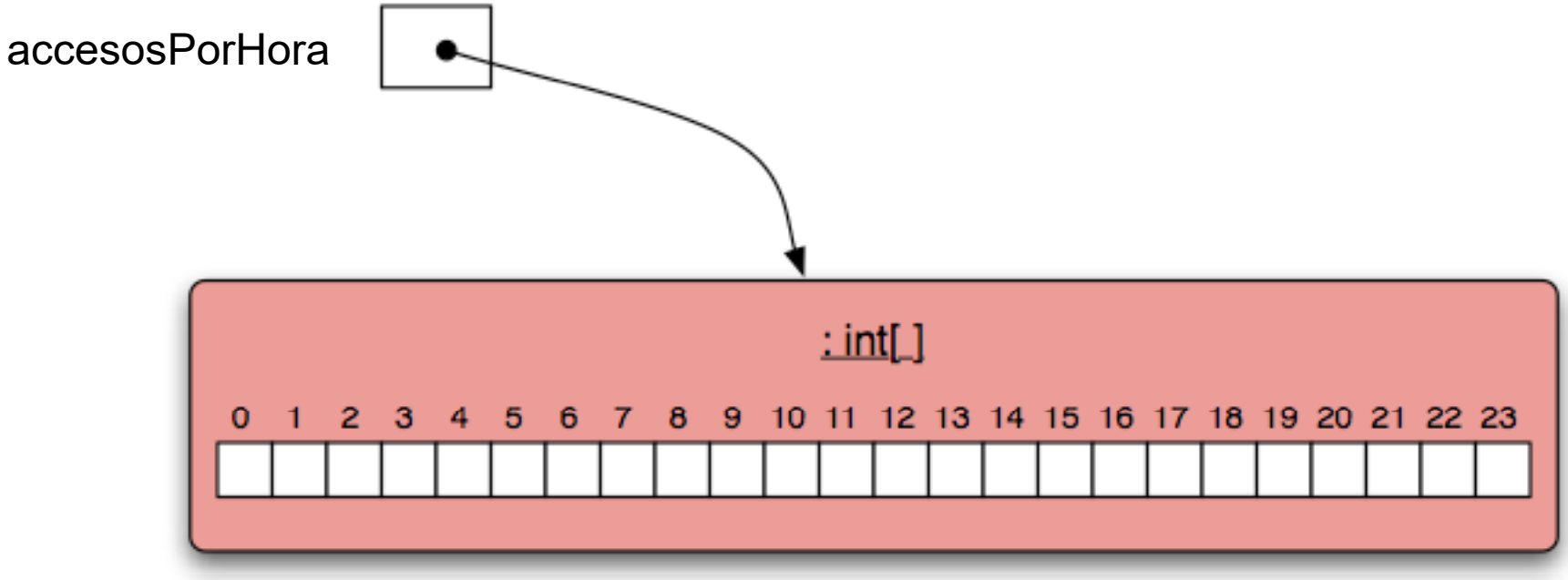
- Ejemplo del proyecto analizador-weblog.
- Cada línea del fichero log (weblog.txt) registra la fecha y hora del acceso.

```
public class AnalizadorLog
{
    private int[] accesosPorHora;
     // Almacena cantidad de accesos por hora

    private LogfileReader reader;

    public AnalizadorLog()
    {
        accesosPorHora = new int[24];
        reader = new LogfileReader();
    }
    ...
}
```

Declaración de la variable Array

Creación del objeto Array

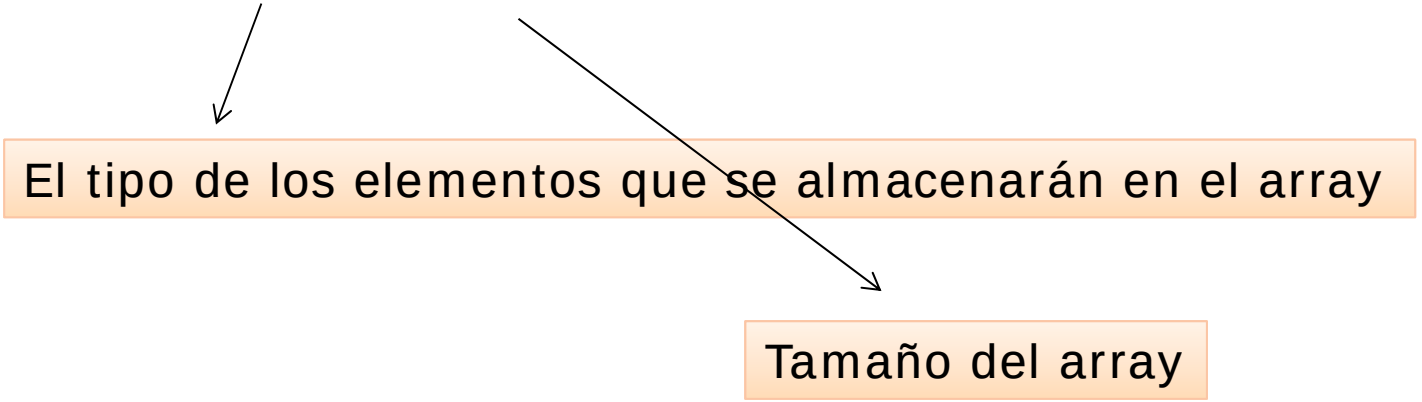
Creando un objeto array



Es un **array de enteros**

Construcción de un objeto array

```
new tipo[expresión-entera]
```



Ejemplo

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

Ejercicios

- Escriba una declaración de una variable array de nombre **`gente`** que podría usarse para referenciar un array de objetos `Persona`.
- Escriba una declaración de una variable array **`vacante`** que haga referencia a un array de `valores lógicos`.

Ejercicios

- Que significan las siguientes declaraciones:

```
double[] lecturas;
String[] urls;
```

- ¿Cuantos objetos String se crean mediante la siguiente declaración?

```
String[] etiquetas = new String[20];
```

- Detectar y corregir el error de la siguiente declaración

```
double[] precios = new double(50);
```

Usando un array

- Los corchetes se usan para acceder a un elemento del array

  `accesosPorHora[…]`

- Se accede a los elementos mediante un **índice**
- Los elementos de un array son usados como variables ordinarias.
  - A la izquierda de una sentencia de asignación:

    ```
    accesosPorHora[hora] = ...;
    ```

  - En una expresión:

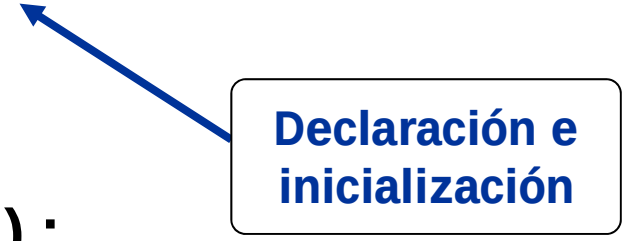
    ```
    adjuste = accesosPorHora[hora] – 3;
    accesosPorHora[hora]++;
    ```

Uso estándar de un array

```
private int[] accesosPorHora;
private String[] nombres;

...

accesosPorHora = new int[24];

...

acceosPorHora[i] = 0;
accesosPorHora[i]++;
System.out.println(accesosPorHora[i]);
```

Arrays literales

```
private int[] numeros = { 3, 15, 4, 5 };
```




```
System.out.println(numeros[i]);
```

Longitud de un Array

```
private int[] numeros = { 3, 15, 4, 5 };
```

```
int n = numeros.length;
```

- Nota: 'length' no es un método es un campo. Lo tienen todos los arrays y su valor es el tamaño del array

El bucle for

- Hay dos variantes del bucle for: ***for-each*** y ***for***.
- El **bucle for**
  - Se usa a menudo para repetir una sentencia o un bloque de sentencias un número fijo (exacto) de veces.
  - Se usa con una variable que cambia una cantidad fija en cada iteración.

El bucle **for** en seudocódigo

Forma general de un bucle for

```
for(inicialización; condición; acción modificadora) {
    sentencias que se repiten en cada iteración
}
```

Equivalente con el bucle while

```
initializacion;
while(condición) {
   sentencias que se repiten en cada iteración
   condición modificadora
}
```

Ejemplo en Java

Versión bucle for

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

Versión bucle while

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

Ejercicio

- Dado un array de números, escribir un método que permita visualizar todos los número del array usando el bucle for.

```
int[] numeros = { 4, 1, 22, 9, 14, 3, 9};
```

Ejercicio

- Que hace el siguiente bucle for.

```
for(int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}
```

Revisión

- Los arrays son apropiados cuando se necesita usar una colección de tamaño fijo.
- Los arrays usan una sintaxis especial.
- Los bucles for ofrecen una alternativa a los bucles while cuando el número de repeticiones es conocida.
- Los bucles for son usados cuando se necesita una variable índice.