

# Agrupamiento de Objetos

Colecciones e iteradores

# Conceptos principales a ser cubiertos

- Colecciones
- Ciclos
- Iteradores
- Arreglos

# Los requerimientos para agrupar objetos

- Muchas aplicaciones usan colecciones de objetos:
  - Organizadores personales.
  - Catálogos de biblioteca.
  - Sistemas de archivo de estudiantes.
- El número de items a almacenar cambia.
  - Items agregados.
  - Items suprimidos.

# Una agenda personal

- Se pueden almacenar notas.
- Se pueden ver notas individuales.
- No hay límites al número de notas.
- Se puede informar sobre el número de notas almacenadas.
- Explore el proyecto *agenda1*.

Ej.: 4.1

# Bibliotecas de clases

- Colecciones de clases útiles.
- No se debe escribir todo desde cero.
- Java denomina a sus bibliotecas, *packages*.
- El agrupamiento de objetos es un requerimiento recurrente.
  - El package `java.util` contiene clases para hacer esto.

```
import java.util.ArrayList;

/**
 * ...
 */
public class Agenda
{
    // Espacio para almacenar un número de notas.
    private ArrayList<String> notas;

    /**
     * Realiza cualquier inicializacion que sea
     * requerida para la agenda
     */
    public Agenda()
    {
        notas = new ArrayList<String>();
    }

    ...
}
```

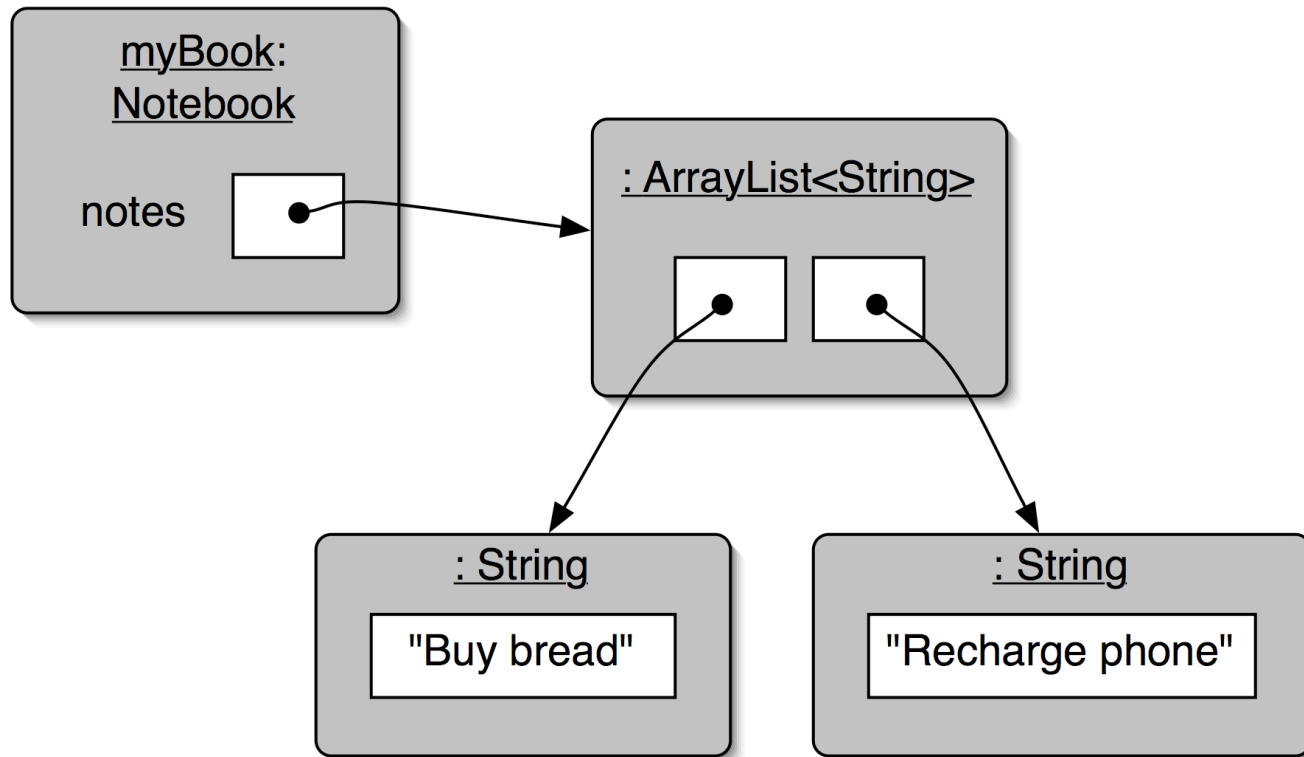
# Colecciones

- Se especifica:
  - el tipo de colección: `ArrayList`
  - el tipo de objetos que contendrá:  
`<String>`
- Se expresa, “`ArrayList de String`”.

Ej.: 4.2

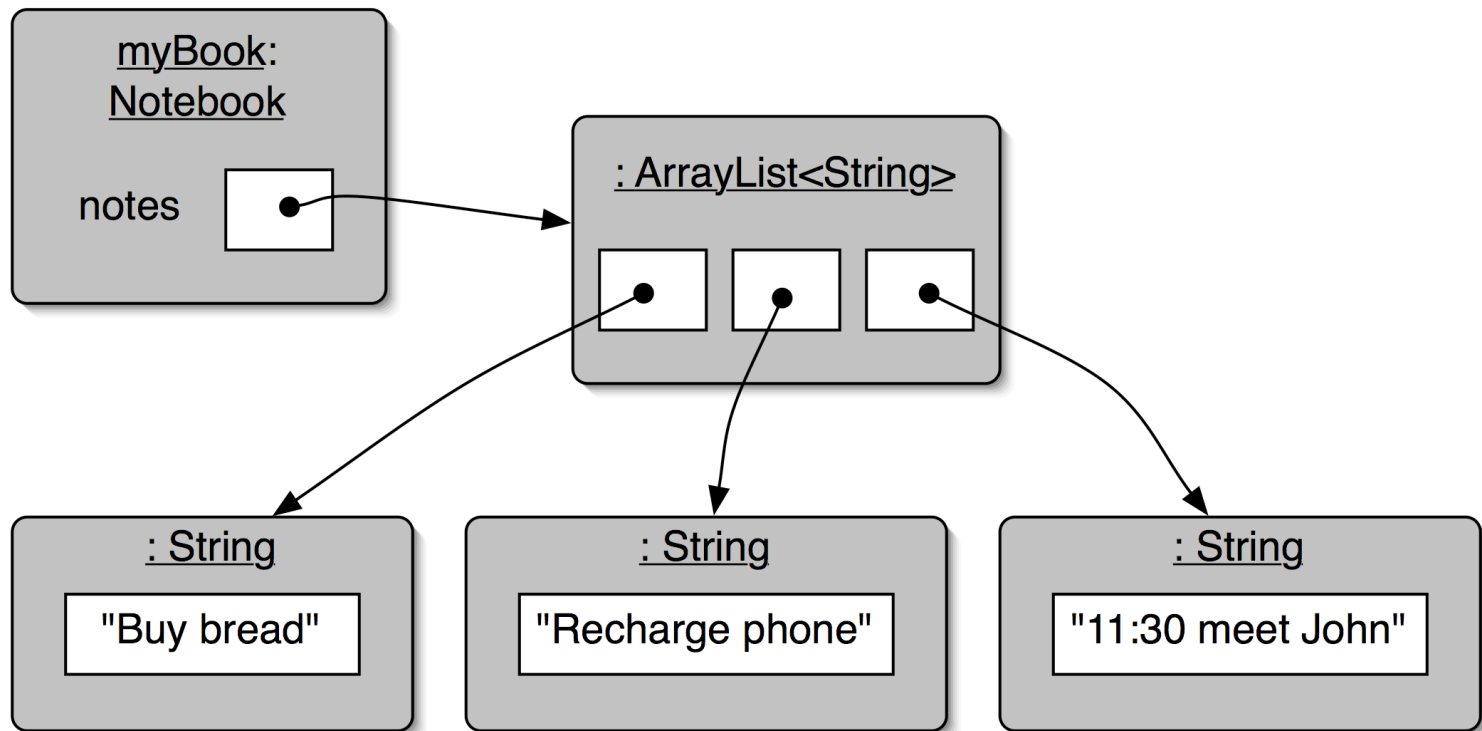


# Objetos de estructuras con colecciones





# Agregando una tercera nota



# Características de una colección

- Incrementa su capacidad si es necesario.
- Mantiene un contador privado (`size()` método\_de\_acceso).
- Mantiene los objetos en orden.
- Los detalles de como se realiza esto estan ocultos.
  - ¿Esto importa? ¿No saberlo nos previene de usarlo?

# Usando la colección

```
public class Agenda
{
    private ArrayList<String> notas;
    ...

    public void guardarNota(String nota)
    {
        notas.add(nota);
    }

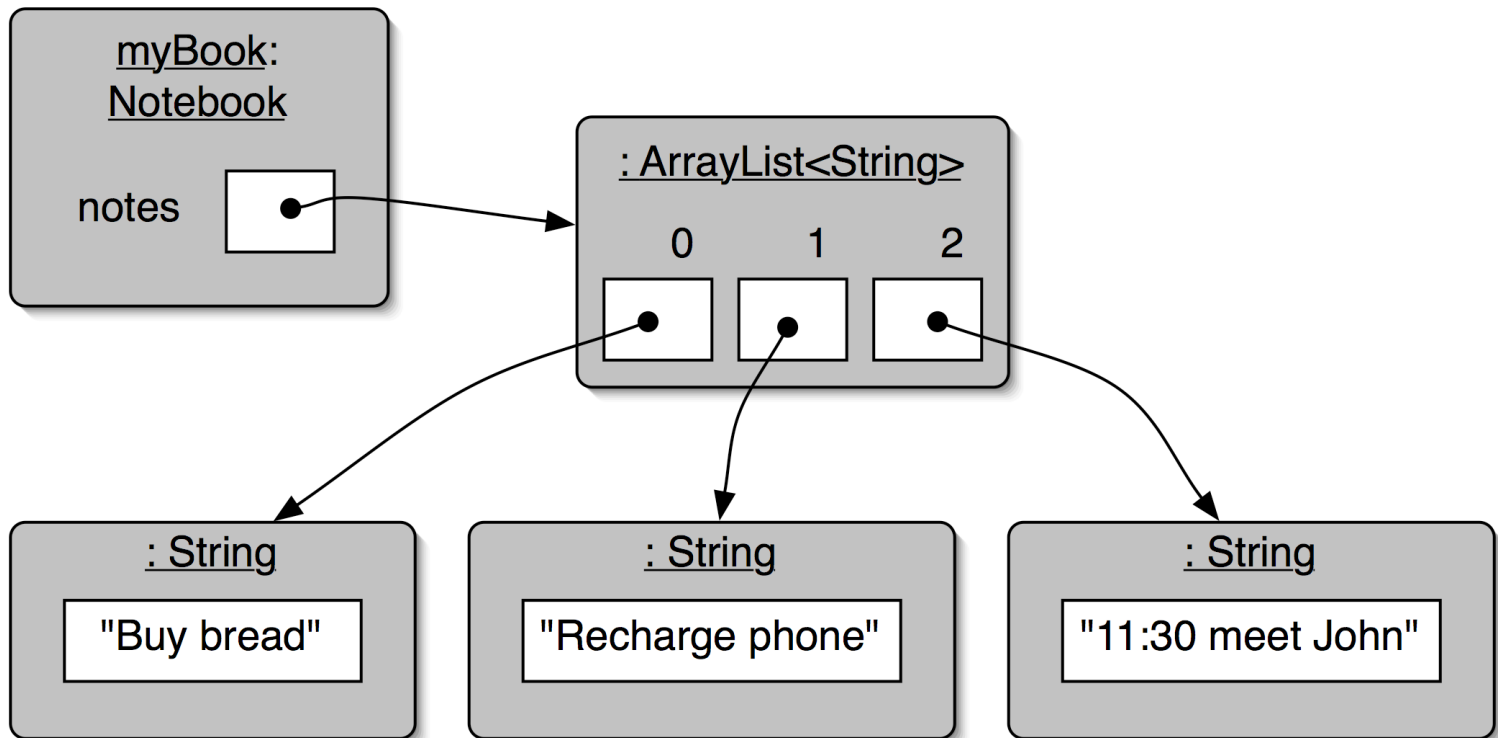
    public int numeroDeNotas()
    {
        return notas.size();
    }

    ...
}
```

Agrega una nota nueva

Retorna el número de notas  
(delegacion)

# Números índices



# Recuperando un objeto

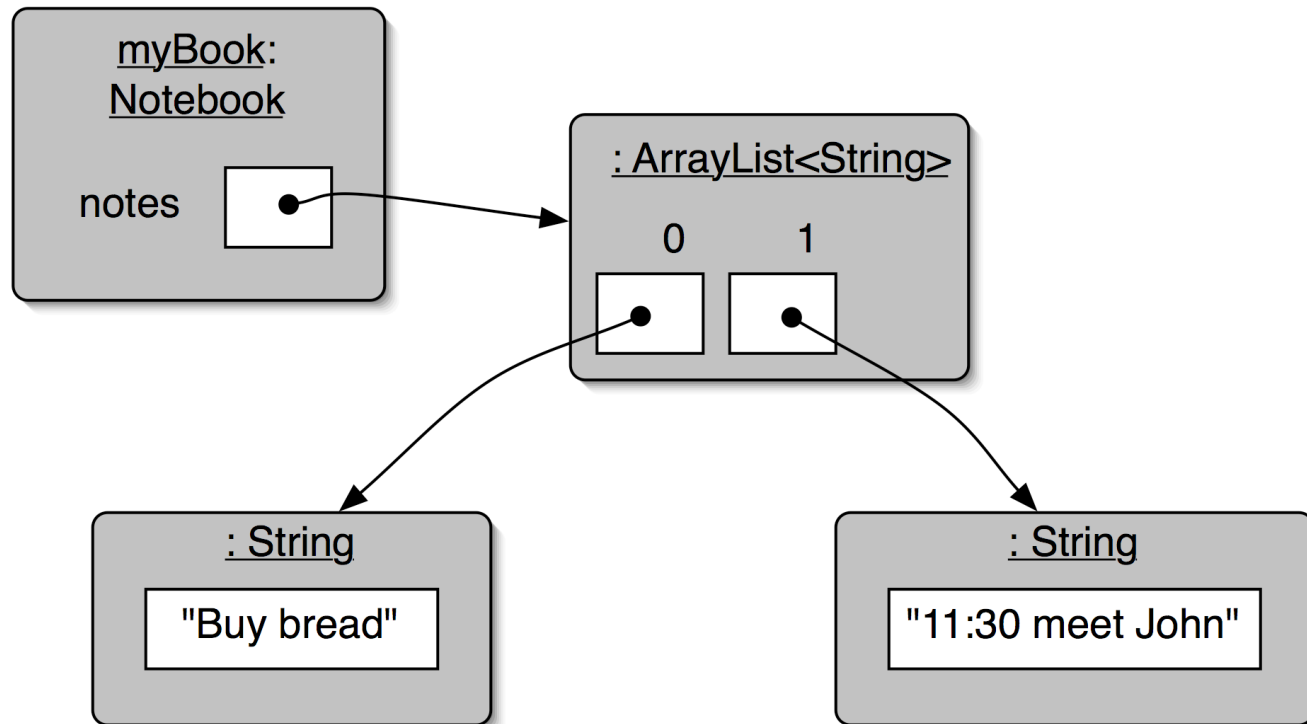
Verifica validez del índice

```
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.
    }
}
```

Recupera e imprime la nota

Ej.: 4.3; 4.4; 4.5; 4.6

# La remoción puede afectar la numeración



Ej.: 4.10; 4.11

# Clases genéricas

- Las colecciones se conocen como tipos *parametrizados o genéricos*.
- `ArrayList` implementa la funcionalidad de una lista:
  - **add**, **get**, **size**, etc.
- El parámetro de tipo indica si se quiere una lista de:
  - `ArrayList<Persona>`
  - `ArrayList<Boleto>`
  - etc.



# Repaso

- Las colecciones permiten que se almacene un número arbitrario de objetos.
- Las bibliotecas de clases usualmente contienen colecciones de clases probadas y verificadas.
- Las bibliotecas de clases de Java se llaman *packages*.
- Se ha usado la clase `ArrayList` del *package* `java.util`.

# Repaso

- Los items pueden ser agregados o removidos.
- Cada item tiene un índice.
- Los valores de los índices pueden cambiar si los items son removidos (o agregados con posterioridad).
- Los principales métodos de `ArrayList` son *add*, *get*, *remove* y *size*.
- `ArrayList` es un tipo parametrizado o genérico.

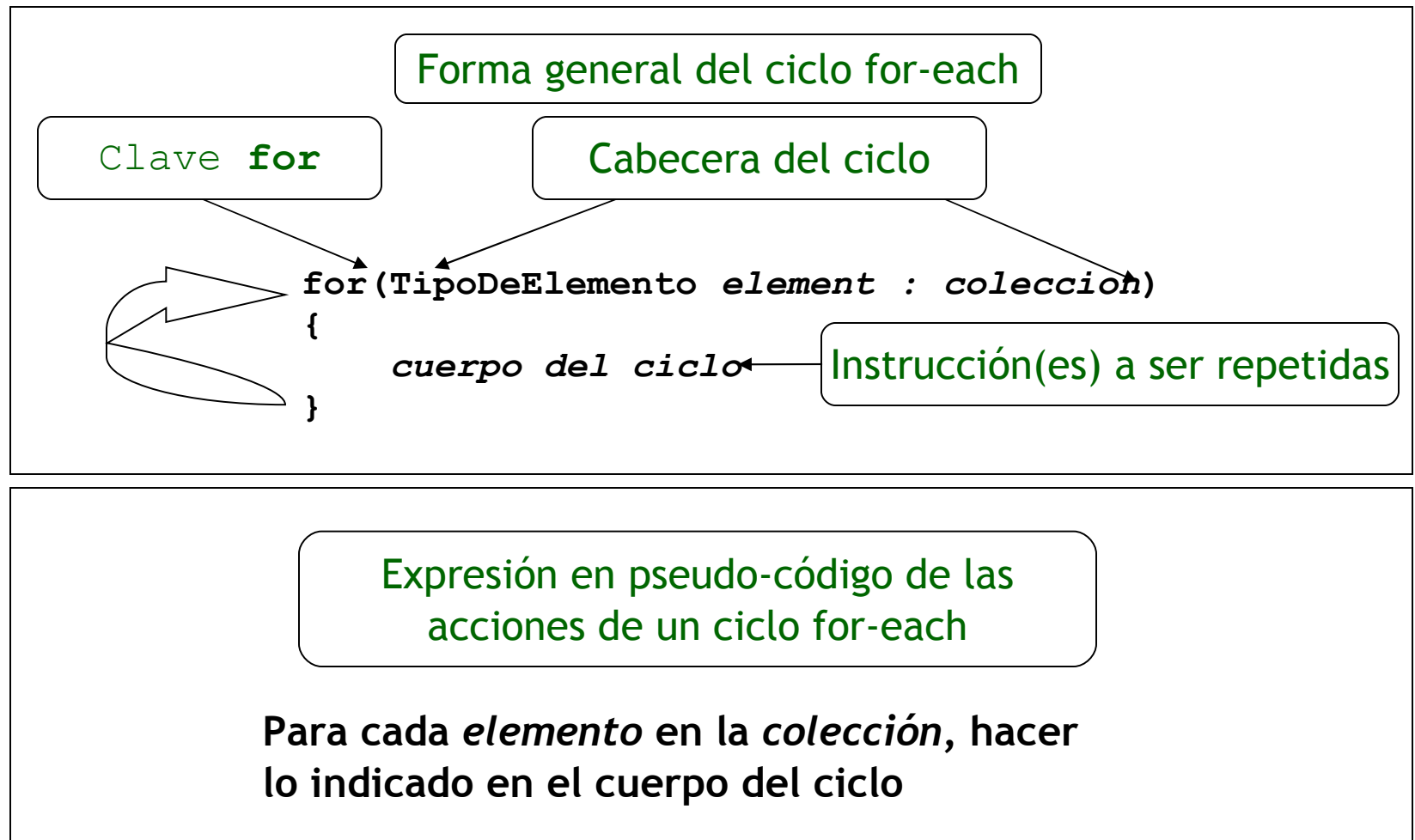
# Iteración

- Frecuentemente queremos realizar algunas acciones un número arbitrario de veces.
  - v.g., imprimir todas las notas de la agenda.  
¿Cuántas hay?
- La mayoría de los lenguajes de programación incluyen *instrucciones de repetición* para hacer esto posible
- Java tiene varias clases de instrucciones de repetición.
  - Se inicia su estudio con el ciclo *for-each*.

# Conceptos de la iteración

- Frecuentemente queremos repetir una acción una y otra vez.
- Los ciclos proveen la forma de controlar cuantas veces se repiten esas acciones.
- Con las colecciones, frecuentemente queremos repetir cosas una vez para cada objeto en particular.

# Pseudo-código del ciclo for-each



# Un ejemplo en Java

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

Para cada *nota* en *notas*, imprime la *nota*  
*Ver proyecto agenda2*

Ej.: 4.2; 4.13; 4.14; 4.15

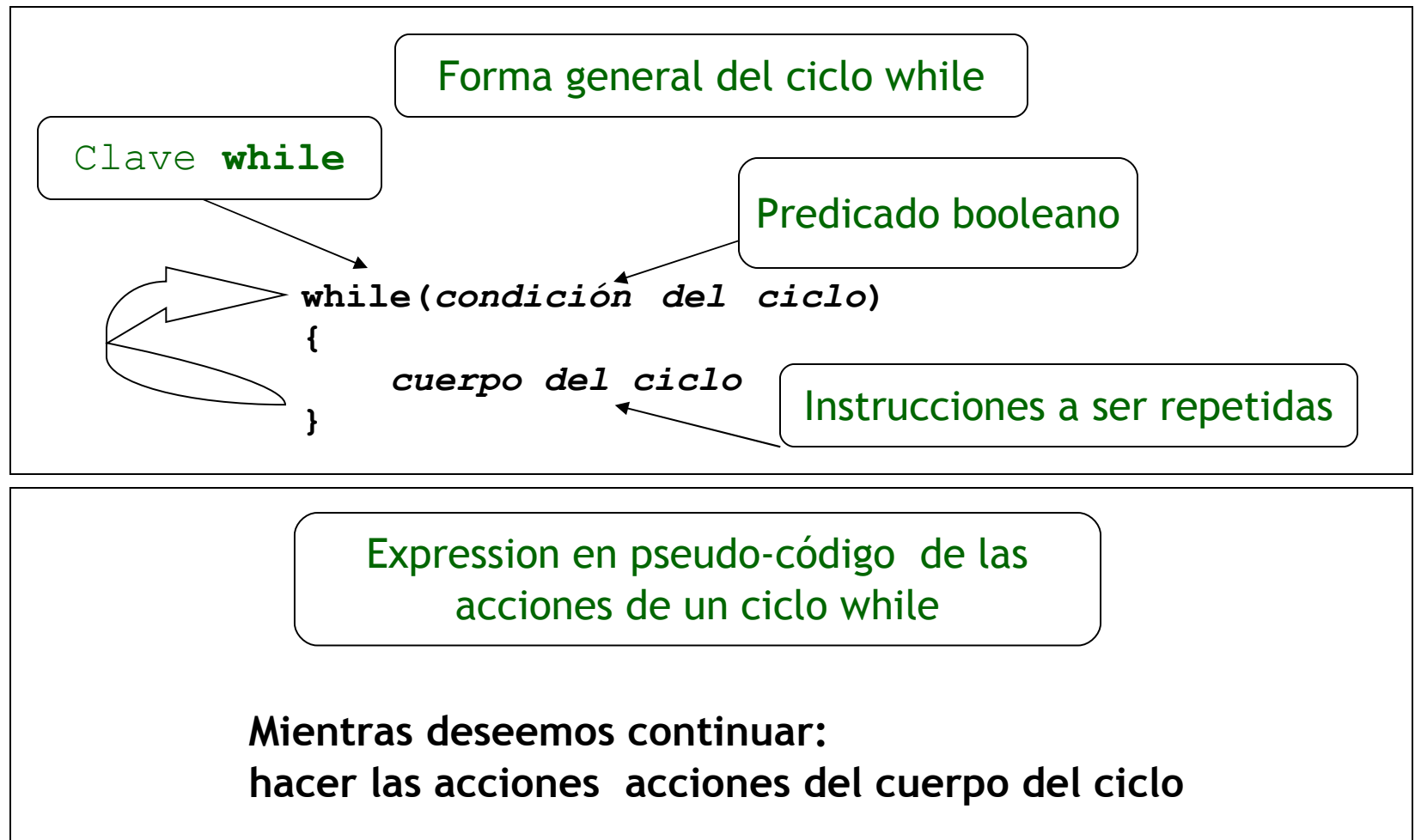


# El ciclo while

- Un ciclo *for-each* repite el cuerpo del ciclo para cada objeto en la colección.
- Algunas veces se requiere más variación que ésta.
- Se puede usar una condición booleana para decidir si se sigue iterando o no.
- Un ciclo *while* provee este control.




# Pseudo-código del ciclo *mientras*



# Un ejemplo en Java

```
/**
 * Listar todas las notas en la agenda.
 */
public void listaNotas()
{
    int indice = 0;
    while(indice < notas.size()) {
        System.out.println(notas.get(indice));
        indice++;
    }
}
```



Incrementa el *indice* en 1

Mientras el valor del *índice* sea menor que el tamaño de la colección, imprime la siguiente nota, e incrementa el *índice*

Ej.: 4.6; 4.17; 4.18; 4.19; 4.20; 4.21; 4.22

# for-each vs while

- for-each:
  - Más fácil de escribir.
  - Más seguro: tiene garantizada la parada.
- while:
  - No se *tiene* que procesar toda la colección.
  - Ni siquiera es necesario tener que usarlo con una colección.
  - Tener cuidado: podría ser un *ciclo infinito* y no tener parada.

# While sin una colección

```
// Imprime todos los números pares
// de 0 a 30.
int indice = 0;
while(indice <= 30) {
    System.out.println(indice);
    indice = indice + 2;
}
```

# Buscando en una colección

```
int indice = 0;
boolean encontrado = false;
while(indice < notas.size() && !encontrado) {
    String nota = notas.get(indice);
    if(nota.contains(secadenaDeBusqueda)) {
        // No necesitamos seguir buscando.
        encontrado = true;
    }
    else {
        indice++;
    }
}
// O se lo encuentra, o se busca la colección
// completa
```

# Usando un objeto iterador

`java.util.Iterator`

retorna un objeto `Iterator`

```
Iterator<TipoDeElemeno> it = miCollection.iterator();  
while(it.hasNext()) {  
    llamar a it.next() para obtener el próximo objeto  
    hacer algo con ese objeto  
}
```

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

# Indices versus Iteradores

- Formas de iterar sobre una colección:
  - Ciclo for-each.
    - Se si se quiere procesar todo elemento.
  - Ciclo while.
    - Se usa si se quisiera detener en una parte del ciclo.
    - Se usa si la repetición no involucra una colección.
  - Objeto iterator.
    - Se usa si se quisiera detener en una parte del ciclo.
    - Usado a menudo con colecciones donde el acceso indexado no es muy eficiente, o imposible.
- La iteration es un *patrón* de programación importante.

Ej.: 4.23; 4.24; 4.25



# El proyecto *subasta*

- El proyecto *subasta* provee una ilustración posterior de las colecciones y la iteración.
- Un punto adicional a seguir más allá: el valor `null`.
  - Usado para indicar, 'ningún objeto'.
  - Se puede verificar si una variable de objeto sostiene la variable `null`.

Ej.: 4.26; 4.27; 4.28; 4.29; 4.30; 4.31

# Repaso

- Las instrucciones de repetición permiten que una instrucción bloque de instrucciones sea repetida.
- El ciclo *for-each* permite la iteración sobre una colección completa.
- El ciclo *while* permite que la repetición sea controlada por una expresión booleana.
- Todas las clases colección proveen objetos especiales `Iterator` para el acceso secuencial a una colección completa.

Ej.: 4.32; 4.33; 4.34; 4.35; 4.36; 4.37; 4.38

# Colecciones de tamaño fijo

- Algunas veces el tamaño máximo de una colección puede ser pre-determinado.
- Los lenguajes de programación usualmente ofrecen un tipo de colección especial de tamaño fijo: un *arreglo*.
- Los arreglos en Java pueden almacenar objetos o valores de tipo primitivo.
- Los arreglos usan una sintaxis especial.

# El proyecto *analizador-weblog*

- Los servidores de Web registran los detalles de cada acceso.
- Que soportan las tareas de los webmaster.
  - Las páginas más populares.
  - Los períodos más ocupados.
  - Cuántos datos están siendo enviados.
  - Referencias quebradas.
- Analizan los accesos por hora.

Ej.: 4.39

# Creando un objeto arreglo

```
public class AnalizadorLog
{
    private int[] contadoresPorHeras;
    private LectorDeArchivoLog lector;

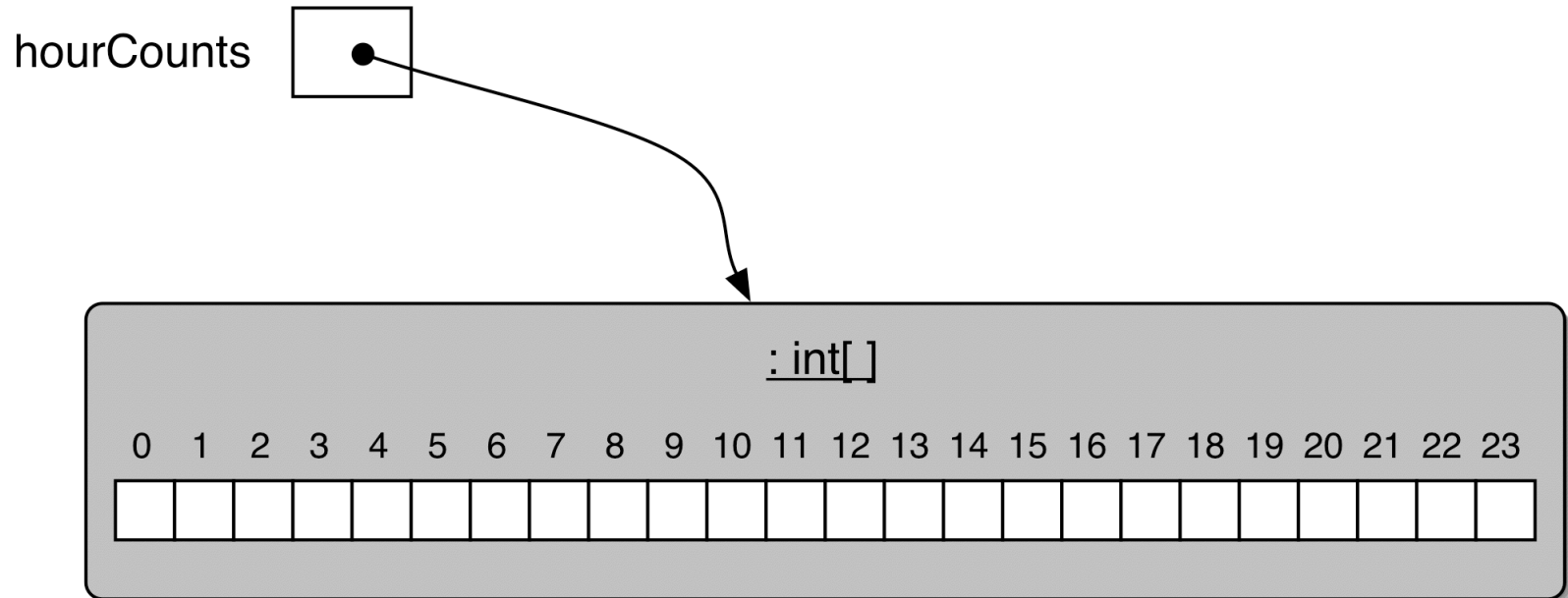
    public AnalizadorLog()
    {
        contadoresPorHera = new int[24];
        lector = new LectorDeArchivoLog();
    }
    ...
}
```

Declaración de variable arreglo

Creación de objeto arreglo

Ej.: 4.40; 4.41; 4.42; 4.43

# El arreglo contadoresPorHora



Ej.: 4.44; 4.45; 4.46



# Usando un arreglo

- Se usa la notación de corchetes para acceder a un elemento de un arreglo:
  - `contadoresPorHora[...]`
- Los elementos se usan como variables ordinarias.
  - A la izquierda de una asignación:
    - `contadoresPorHora[hora] = ...;`
  - En una expresión:
    - `adjustado = contadoresPorHora[hora] - 3;`
    - `contadoresPorHora[hora]++;`



# El ciclo *for*

- Hay dos variantes del ciclo *for*, *foreach* y *for*.
- El ciclo *for* es usado a menudo para iterar un número fijo de veces.
- A menudo es usado con una variable que cambia una cantidad fija en cada iteración.

# Pseudo-código del ciclo *for*

## Forma general de un ciclo *for*

```
for(inicialización; condición; acción post-cuerpo) {  
    instrucciones a ser repetidas  
}
```

## Forma equivalente en un ciclo *while*

```
inicialización;  
while(condición) {  
    instrucciones a ser repetidas  
    acción post-cuerpo  
}
```

# Un ejemplo en Java

## Version ciclo for

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

## Versión ciclo while

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

# El ciclo for con un paso más grande

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

Ej.: 4.47; 4.48; 4.49; 4.50; 4.51; 4.52; 4.53; 4.54;  
4.55; 4.56; 4.57; 4.58; 4.59; 4.60; 4.61; 4.62;  
4.63; 4.64;

# Repaso

- Los arreglos son apropiados donde se requiere una colección de tamaño fijo.
- Los arreglos usan una sintaxis especial.
- Los ciclos loops ofrecen una alternativa a los ciclos while cuando el número de repeticiones es conocido.
- Se usan ciclos for cuando se requiere una variable indexada.