Programación orientada a objetos

Capítulo 4

Agrupar objetos

- Separación de código	
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

Biblioteca (packages)de clases

```
import java.util.ArrayList;
/**
 * A class to maintain an arbitrarily long list of notes.
 * Notes are numbered for external reference by a
human user.
 * In this version, note numbers start at 0.
 *
 * @author David J. Barnes and Michael Kolling.
 * @version 2006.03.30
 */
public class Notebook
{
    // Storage for an arbitrary number of notes.
    private ArrayList<String> notes;

    /**
     * Perform any initialization that is required for the
     * notebook.
     */
    public Notebook()
    {
        notes = new ArrayList<String>();
    }

    /**
     * @return The number of notes currently in the
notebook.
     */
    public int numberOfNotes()
    {
        return notes.size();
    }

    /**
     * Show a note.
     * @param noteNumber The number of the note to
be shown.
     */
    public void showNote(int noteNumber)
    {
        if(noteNumber < 0) {
            // This is not a valid note number, so do nothing.
        }
        else if(noteNumber < numberOfNotes()) {
            // This is a valid note number, so we can print it.
            System.out.println(notes.get(noteNumber));
        }
        else {
            // This is not a valid note number, so do nothing.
        }
    }
}
```

Estructura de objetos con “colecciones”

Existen por lo menos tres características importantes de la clase `ArrayList` que debería observar:

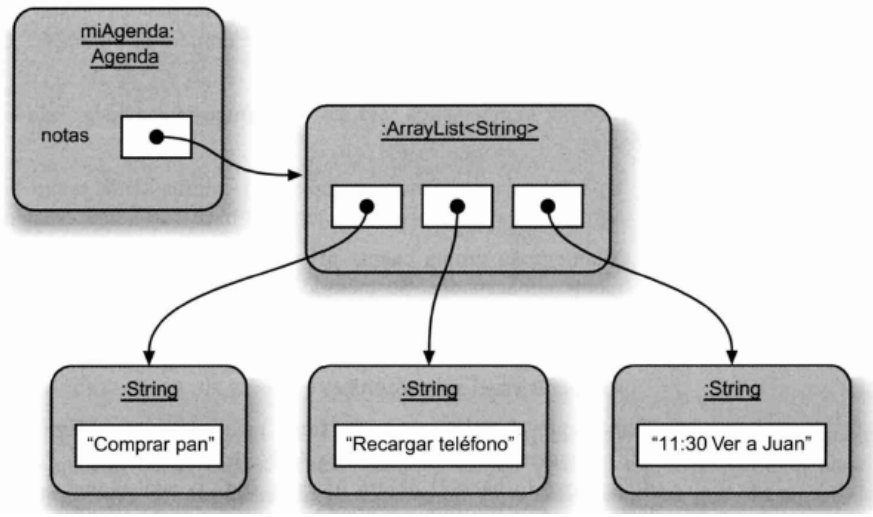
- Es capaz de aumentar su capacidad interna tanto como se requiera: cuando se agregan más elementos, simplemente hace suficiente espacio para ellos.
- Mantiene su propia cuenta privada de la cantidad de elementos que tiene actualmente almacenados. Su método `size` devuelve el número de objetos que contiene actualmente.
- Mantiene el orden de los elementos que se agregan, por lo que más tarde se pueden recuperar en el mismo orden.

Clases genéricas

No definen un único tipo

```
private ArrayList<Persona> miembros;
private ArrayList<MaquinaDeBoletos> misMaquinas;
```

Define el tipo



Eliminar un elemento de la “colección”

```
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.
            }
      }
```

El ciclo “for-each”

- Realiza el ciclo una vez por cada elemento 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

El ciclo “while”

```
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);
      }
}
```

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
```

Clase “Iterator”

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

- 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`:

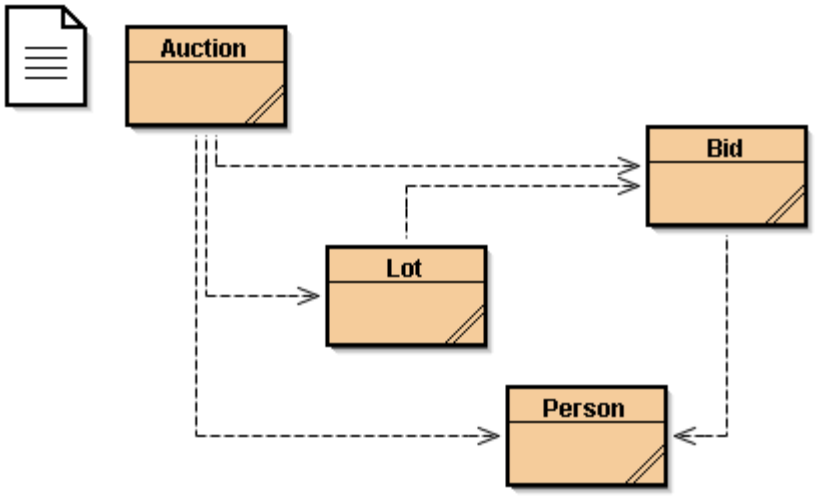
```
Iterator<TipoDelElemento> it = miColeccion.iterator();
while (it.hasNext()) {
  Invocar it.next() para obtener el siguiente elemento
  Hacer algo con dicho elemento
}
```

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

```
/**
 * 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 ++;
}
```

Ejemplo: subasta



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;
     }
  }
```

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:

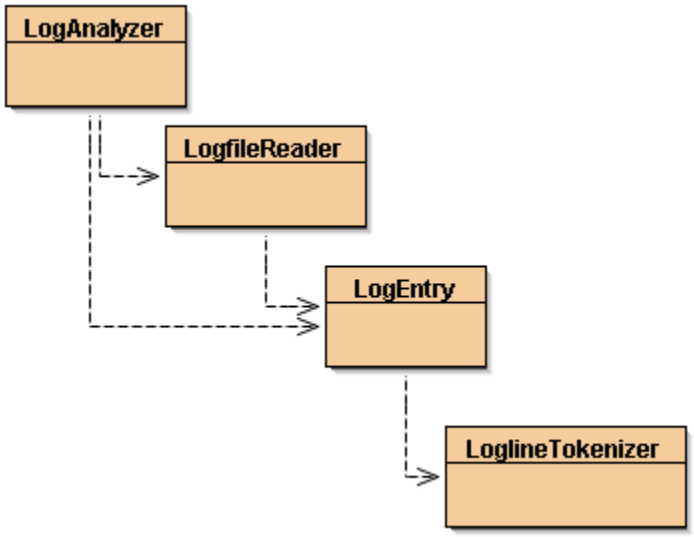
- Creamos un nuevo objeto `Lote` y
- Pasamos este nuevo objeto al método `add` de `ArrayList`.

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.

El sistema de subasta



Colecciones de tamaño fijo: “arreglos” o “arrays”

- El acceso a los elementos de un arreglo es generalmente más eficiente que el acceso a los elementos de una colección de tamaño flexible.
- Los arreglos son capaces de almacenar objetos o valores de tipos primitivos. Las colecciones de tamaño flexible sólo pueden almacenar objetos[3].

Declaración de variables arreglos

```
private int[ ] contadoresPorHora;
```

La característica distintiva de la declaración de una variable de tipo arreglo es un par de corchetes que forman parte del nombre del tipo: `int[ ]`. Este detalle indica que la variable `contadoresPorHora` es de tipo *arreglo de enteros*. Decimos que `int` es el tipo base de este arreglo en particular. Es importante distinguir entre una declaración de una variable arreglo y una declaración simple ya que son bastante similares:

```
int hora;
int[ ] contadoresPorHora;
```

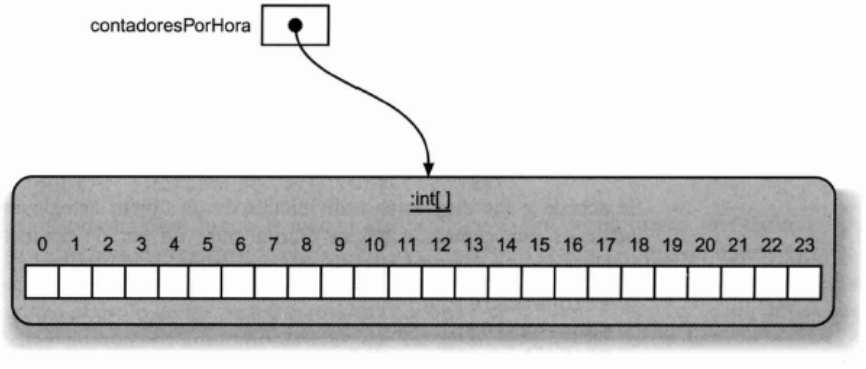
Creación de objetos “arreglo”

La forma general de la construcción de un objeto arreglo es:

```
new tipo[expresión-entera]
```

La elección del tipo especifica de qué tipo serán todos los elementos que se almacenarán en el arreglo. La expresión entera especifica el tamaño del arreglo, es decir, un número fijo de elementos a almacenar.

```
int[ ] contadoresPorHora;
contadoresPorHora = new int[24];
```



En un solo paso

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

Usar objetos “arreglos”

Se accede a los elementos individuales de un objeto arreglo mediante un *índice*. Un índice es una expresión entera escrita entre un par de corchetes a continuación del nombre de una variable arreglo. Por ejemplo:

```
etiquetas[6];
maquinas[0];
gente[x + 10 - y];
```

Los valores válidos para una expresión que funciona como índice depende de la longitud del arreglo en el que se usarán. Los índices de los arreglos siempre comienzan por cero y van hasta el valor uno menos que la longitud del arreglo. Por lo que los índices válidos para el arreglo `contadoresPorHora` son desde 0 hasta 23, inclusive.

```
etiqueta[5] = "Salir";
double mitad = lecturas[0] / 2;
System.out.println(gente[3].getNombre());
maquinas[0] = new MaquinaDeBoletos(500);
```

El ciclo “for”

Un *ciclo for* tiene la siguiente forma general:

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

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

Comparación con “while” y “for-each

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

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

¿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).

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.