Fundamentos de la Programación Orientada a Objetos

Agrupación de objetos. Colecciones e iteradores

Programación Orientada a Objetos
Facultad de Informática

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense Madrid

Basado en el curso Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Conceptos a tratar

- Colecciones
- Bucles
- Iteradores
- Arrays

- Paquetes: liberías de clases

La necesidad de agrupar objetos

- Muchas aplicaciones requieren agrupar objetos
  - Agendas
  - Catálogos de librerías o de productos
  - Sistema de matrícula de la Universidad
  - ...
- 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

Ejemplo: Agenda de notas

- Las notas se pueden almacenar
- Se pueden consultar notas individuales
- No hay límite en el número de notas
- Se puede saber cuántas notas se tienen guardadas
- Proyecto *notebook1*

Librerías de clases

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

Un ejemplo de lista

```
import java.util.ArrayList;

/**
 * ...
 */
public class Notebook
{
    // Storage for an arbitrary number of notes.
    private ArrayList<String> notes;

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

    ...
}
```

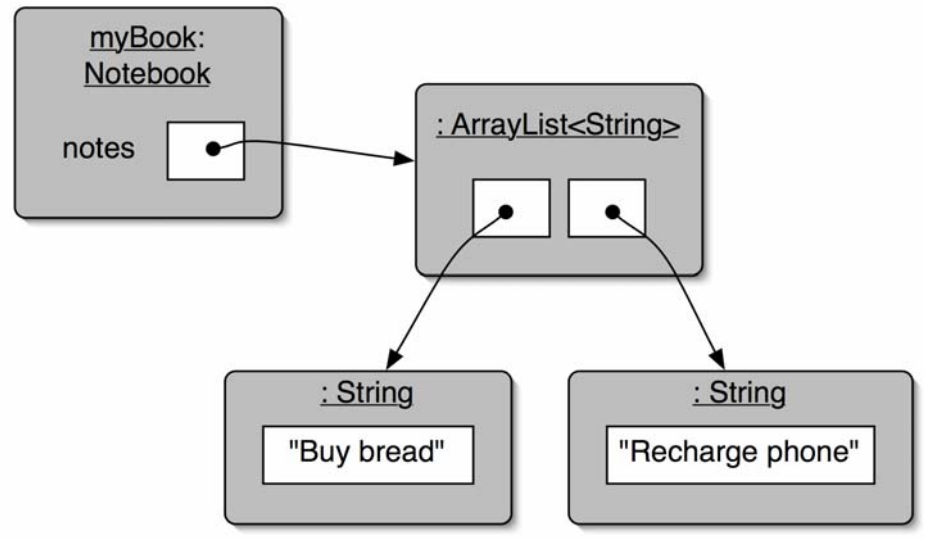
Indica que se va a utilizar la clase ArrayList del paquete java.util

Colecciones

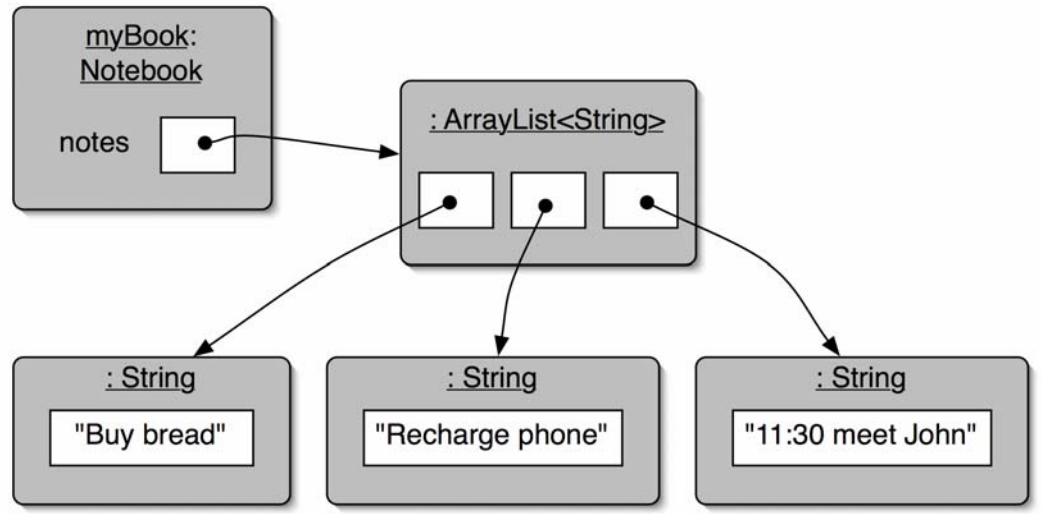
- Se especifica
  - El tipo de la colección: `ArrayList`
  - El tipo de objetos que contendrá: `<String>`

- Esto se lee como: "*ArrayList de String*"

Estructuras de los objetos con colecciones



Adición de otro nodo



Características de la colección

- Incrementa su capacidad como sea necesario
- Se puede saber el tamaño con el método `size()`
- Guarda los objetos en orden
- Los detalles de cómo ocurre todo esto están ocultos
  - ¿Realmente importan?
  - ¿El hecho de no conocerlos nos impide usar la colección?

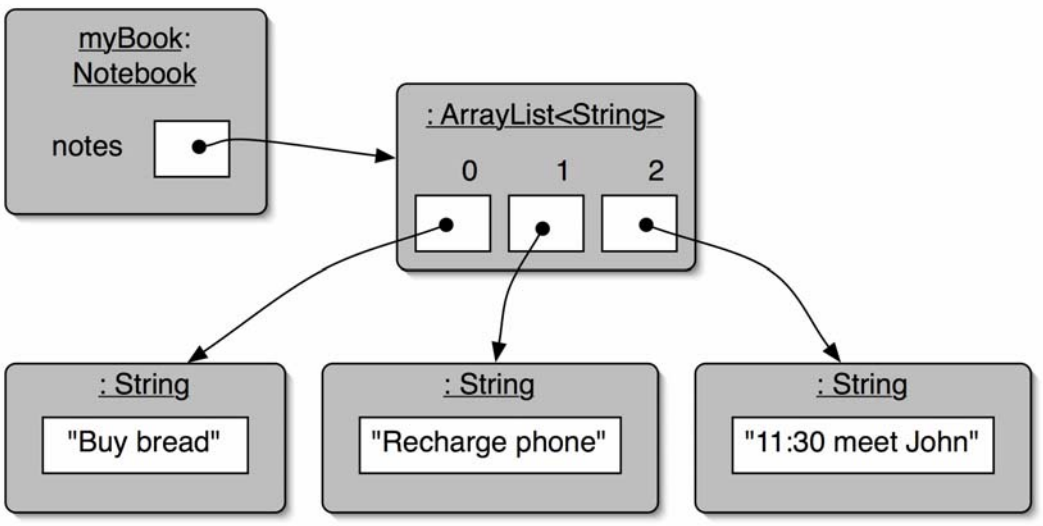
Uso de la colección

```
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*)

Numeración de los elementos



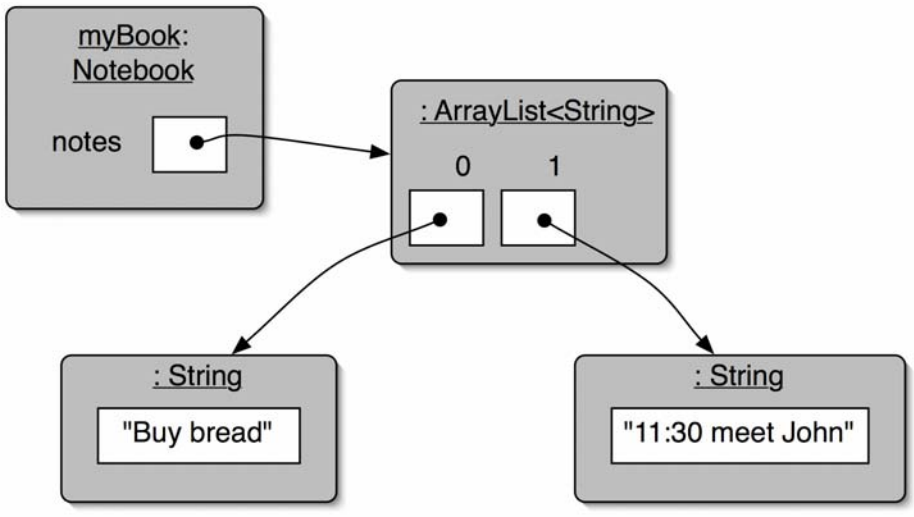
Recuperación de un objeto

```
public void showNote(int noteNumber)
{
    if(noteNumber < 0) {
        // This is not a valid note number.
    }
    else if(noteNumber < numberOfNotes()) {
        System.out.println(notes.get(noteNumber));
    }
    else {
        // This is not a valid note number.
    }
}
```

Comprobación de la validez de los índices

Recupera e imprime la nota

La recuperación puede afectar a la numeración



Clases genéricas

- Las colecciones están implementadas como tipos *genéricos* o *parametrizados*

- La clase `ArrayList` implementa la funcionalidad de la lista
  - **`add`**, **`get`**, **`size`**, etc.
- El tipo de parámetro indica de qué queremos que sea la lista:
  - **`ArrayList<Person>`**
  - **`ArrayList<TicketMachine>`**
  - etc.

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

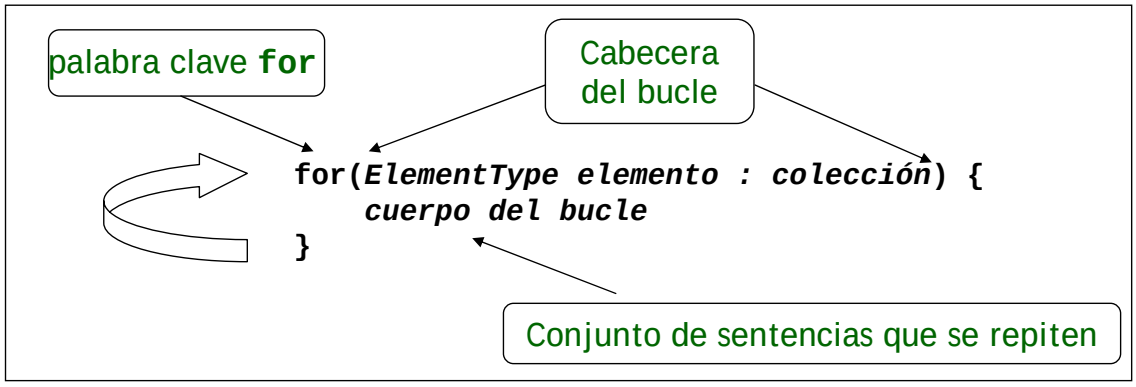
Iteración

- 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, …

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

Bucle for-each en Java



Para cada *elemento* **de la** *colección*, **hacer las cosas del cuerpo del bucle**

Ejemplo

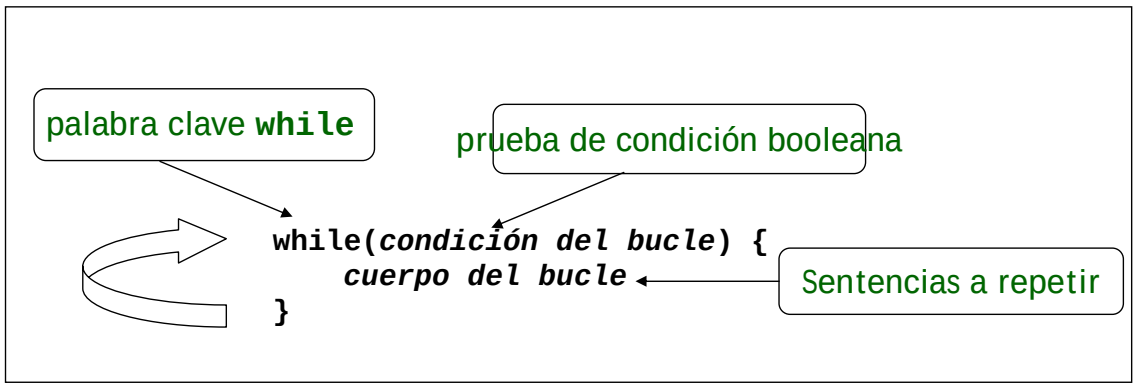
```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    for(String note : notes) {
        System.out.println(note);
    }
}
```

para cada *note* **en** *notes*, **imprimir la** *note*

Bucle while

- El bucle for-each repite el cuerpo del bucle para cada objeto de una colección
- A veces es necesario una variación mayor
  - Se puede utilizar entonces una condición booleana para decidir si seguir o no con el bucle
  - Esto se puede hacer con un bucle while

Bucle while



Mientras se quiera continuar, hacer las cosas del cuerpo del bucle

El mismo ejemplo, pero con while

```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int index = 0;
    while(index < notes.size()) {
        System.out.println(notes.get(index));
        index++;
    }
}
```

Incrementa *index* en 1

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

Comparativa *for-each* vs. *while*

- *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
}
```

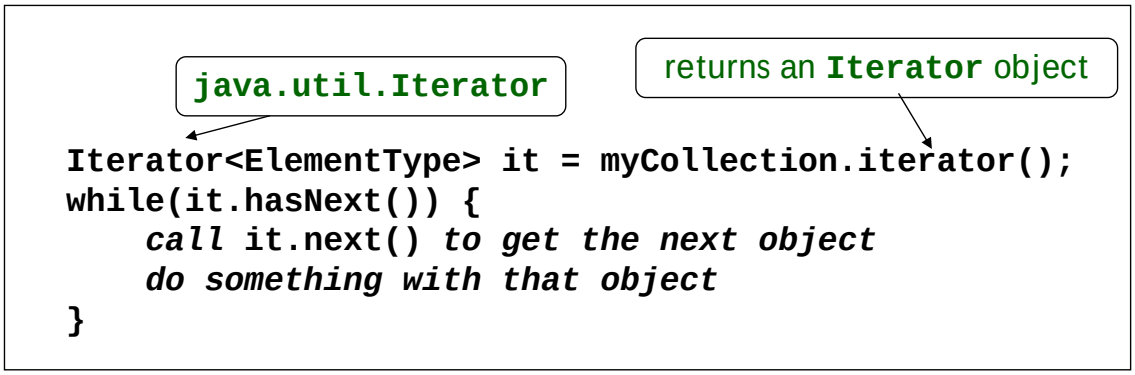
While sin una colección

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


Buscando en una colección

```
int index = 0;
boolean found = false;
while(index < notes.size() && !found) {
    String note = notes.get(index);
    if(note.contains(searchString)) {
        // We don't need to keep looking.
        found = true;
    }
    else {
        index++;
    }
}
// Either we found it, or we searched the whole
// collection.
```

Usando un objeto Iterator (patrón iterador)




```
Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}
```

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


Index vs. Iterator

- Varias formas de iterar en una colección:
  - Bucle for-each
    - Para procesar todos los elementos uno a uno
  - Bucle while
    - Si se quiere parar a mitad de camino
    - Para repeticiones que no se hacen sobre una colección
  - Objeto Iterator
    - Si se quiere parar a mitad de camino
    - A menudo en las colecciones donde el acceso indexado no es muy eficiente o imposible
- La iteración es un *patrón de diseño* muy importante

El proyecto *auction* (subasta)

- Otro ejemplo del uso de colecciones e iteraciones
- Y además veremos otro concepto: el valor `null`
  - Permite indicar 'no object'
  - Se puede probar si una variable de referencia a objeto tiene el valor *null* (esto es, que no hace referencia a ningún objeto)

- Para jugar con este ejemplo:
  - Inspeccionad el código de las clases
  - Cread algún objeto Auction
  - Cread objetos Person
  - Cread lotes para subasta sobre el objeto Auction
  - Realizad pujas sobre los lotes
  - Listad los lotes

Resumen

- Las sentencias de bucle permiten repetir un bloque de instrucciones
  - El bucle for-each permite la iteración sobre una colección entera
  - El bucle while permite la repetición controlada por una expresión booleana
  - Todas las clases de colección suelen ofrecer objetos iteradores especiales que permiten el acceso secuencial a la colección

Colecciones de tamaño fijo: Arrays

- A veces el tamaño máximo de una colección puede estar predeterminado
  - Para ello los lenguajes de programación suelen ofrecer un tipo de colección de tamaño fijo:

array

- Los arrays Java pueden guardar referencias a objetos o valores de tipos primitivos
  - Tienen una sintaxis especial

Arrays en Java

- Colección de variables, todas del mismo tipo
  - Los elementos pueden ser tipos primitivos u objetos
- Aunque son objetos, su declaración, creación e inicialización es diferente que para otros objetos

- Declaración
  - El tipo de variable que tiene un array se especifica al declararlo, añadiendo un par de corchetes ([])

```
int A[];      // A es un array de enteros
int [] A;     // igual que antes
int A[10];    // ERROR: no se especifica el tamaño en la declaración
String S[];   // un array de cadenas
String S,T[]; // S es una cadena y T un array de cadenas
String[] S,T; // Ambos, S y T, son arrays de cadenas
```

Arrays en Java

- Creación e inicialización de arrays
  - El tamaño de un array se especifica al crearlo con el operador new

```
int A[] = new int[10];  // array de 10 enteros:
                        // A[0]..A[9]
String[] S;
S = new String[10];
```

  - Y también puede inicializarse

```
for ( int i=0; i<10; i++ )
      S[i]= new String(i);
```

  - Y ambas cosas a la vez:

```
int[] arrayDeEnteros = {1,2,3,4,5};
```


Arrays en Java

- Array de arrays

```
int[][] A;
A = new int[2];
for (int i = 0; i < A.length; i++)
    A[i] = new int[3];

// también se puede hacer así:
A = new int[2][3];
```

- Se puede hacer un array de arrays de diferentes tamaños

Arrays en Java

- Acceso a los elementos de un array
  - El primer elemento de un array tiene el índice 0
  - Se comprueba automáticamente los límites del array
    - Si se intenta acceder fuera de los límites del array (entre 0 y length-1), se produce la excepción IndexOutOfBoundsException

- Tamaño de un array: miembro *length*

```
A.length          // correcto
A.length()        // ERROR: no se usan paréntesis

for (int i = 0; i < A.length; i++)   A[i]=i;
```


El proyecto *weblog-analyzer*

- Un servidor web registra los detalles de cada acceso
- Ayuda al webmaster a realizar sus tareas:
  - Páginas más populares
  - Periodos de mayor ocupación
  - Tráfico de datos
  - Referencias rotas
- Los accesos se analizan cada hora

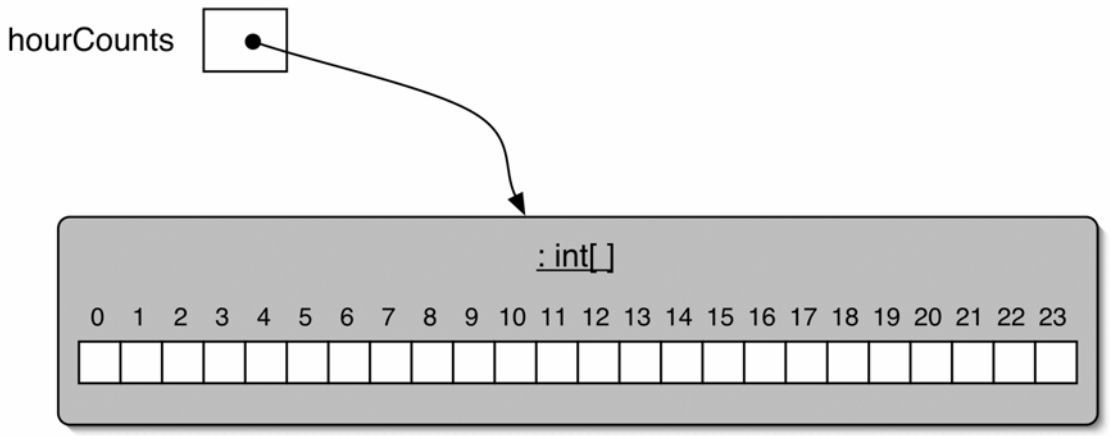
Creación de un array

```
public class LogAnalyzer
{
    private int[] hourCounts;
    private LogfileReader reader;

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

Declaración de variable array

Creación de objeto array

El array hourCounts



Uso del array

- Para acceder a un elemento del array se utilizan los corchetes:

```
hourCounts[...]
```

- Los elementos del array se utilizan como variables normales:
  - Para asignarles un valor:
    - `hourCounts[hour] = ...;`
  - En una expresión:
    - `adjusted = hourCounts[hour] – 3;`
    - `hourCounts[hour]++;`

El bucle for

- Hay dos variaciones del bucle for: *for-each* y *for*.
- El bucle for se utiliza normalmente para iterar un número fijo de veces
- Generalmente se utiliza con una variable que se incrementa (o decrementa) una cantidad fija en cada iteración

Bucle for

```
for(inicialización; condición; acción posterior ) {
    sentencias a repetir
}
```

Equivalencia con while

```
inicialización;
while(condición) {
    sentencias a repetir
    acción posterior
}
```


Ejemplo de bucle for

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++;
}
```

Bucle for con un paso mayor

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


Resumen

- Los arrays son apropiados cuando se trabaja con colecciones de tamaño fijo
  - Tienen una sintaxis especial y están soportados de manera nativa por el lenguaje de programación
- Los bucles for son una alternativa a los bucles while cuando el número de repeticiones es conocido

Resumen: Instrucciones de control de bucles

- ```
  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

Más sobre paquetes en Java

- Cada clase o interfaz en Java está dentro de un paquete
- El paquete al que pertenecen se declara al principio del fichero en el que se especifique la clase o interfaz:

```
package nombre;// declara todo lo que haya en el fichero
                // como parte del paquete "nombre"
```

- Si no se declara un paquete específico entonces se considera que pertenece a un paquete por defecto (*default)* que no tiene ningún nombre
  - El paquete por defecto sólo se suele utilizar en aplicaciones pequeñas o temporales
  - Se recomienda acostumbrarse a definir paquetes para todas las aplicaciones

Paquetes

- Los paquetes:
  - Definen contextos de denominación
    - Para evitar colisión de nombres
  - Sirven para definir bibliotecas de clases e interfaces
    - Reutilización: no volver a inventar la rueda
  - Permiten organizar el código de una gran aplicación
    - Las clases e interfaces relacionadas se declaran en el mismo paquete
    - Ayuda a encontrar dónde están clases e interfaces
  - Permite restringir el acceso a clases y operaciones de un paquete
    - Mayor seguridad del código

Paquetes

- Los paquetes pueden anidarse
  - Define una jerarquía:
    paquete.subpaquete.subpaquete.clase

- Operador de resolución de ámbito: .
  - Paquetes dentro de paquetes
  - Clases dentro de paquetes
  - Métodos dentro de clases
  - Variables dentro de métodos y clases

    - Ejemplo: java.lang.System.out

      Es la variable *out*,
      de la clase *System*,
      del paquete *lang*,
      del paquete *java*

Paquetes

- Convención para el nombrado de paquetes (para conseguir nombres exclusivos):

dominio.empresa.departamento.proyecto

Autor del paquete
(nombre de dominio Internet al revés)

Ejemplo: **`es.ucm.sip.jpavon.cursoJava`**

Paquetes

- Un paquete determina un subdirectorio del disco
  - En el ejemplo anterior:
    `es/ucm/sip/jpavon/cursoJava/`
- Variable de entorno CLASSPATH
  - Indica en qué directorios o ficheros zip empezar a buscar paquetes
  - Se inicializa así:
    - En Unix (por ejemplo en csh):
      `set classspath=(/usr/jdk1.1/lib/classes.zip; ...)`
    - En Windows, en Propiedades del Sistema-> Opciones avanzadas-> Variables de entorno:
      `CLASSPATH=C:\JAVA\JDK1.1\lib\classes.zip;.;`

Paquetes

- Utilización de nombres (públicos) de un 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);
  }
}
```


Paquetes

- *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
```

El compilador busca en classes/java/applet/*

Paquetes

- Paquetes estándar de Java
  - java.lang // clases e interfaces básicas (se importa por defecto)
  - java.applet // clase Applet e interfaces para interacción con navegador
  - java.awt // Abstract Windowing Toolkit
  - java.io // E/S
  - java.net // clases para comunicación a través de protocolos Internet
  - java.rmi // Programación distribuida
  - java.security // Seguridad en Java
  - java.util // Utilidades
  - Javax.swing // componentes gráficos para Java

- Documentados en línea (Java API):

  http://DirectorioInstalacionJava/docs/api/index.html

  - Se puede descargar desde http://java.sun.com/j2se/

    como un fichero zip que se suele descomprimir en el directorio donde se haya instalado el JDK

Paquetes

- Ejemplo

```
package ucm.sip.juan.curso.java;

import java.util.Date;

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

La clase ImprimeFecha está declarada dentro del paquete ucm.sip.juan.curso.java

La clase System está dentro del paquete java.lang (que se importa por defecto)
Se podría poner *java.lang.System.out.println*