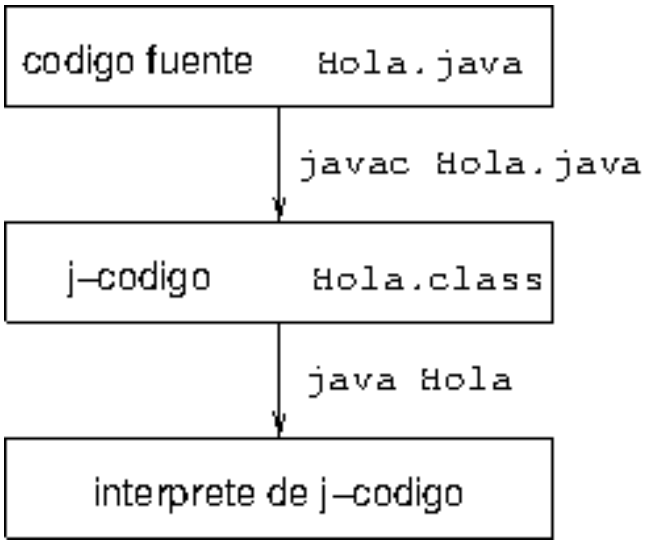
Java

Entorno de Programación Emacs/JDEE/Ant

Luis Fernando Llana Díaz

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

8 de abril de 2013



- Sintaxis inspirada en C++.
- Lenguaje orientado a objetos.
- Código compilado independiente de la máquina j-código.
- Lenguaje *de moda*.
- Compilador e intérpretes gratuito. Libres según la versión.

Entornos de programación

- Consola+editor de textos.
- Editor de texto emacs + JDEE.
- Eclicpse y NetBeans, entornos libres.
- JBuilder entorno propietario.

GNU-emacs (JDEE)

Ventajas

- Código libre.
- Editor potente (coloreado, sangrado automático, compilación integrada).
- Editor programable.

GNU-emacs (JDEE)

Ventajas

- Código libre.
- Editor potente (coloreado, sangrado automático, compilación integrada).
- Editor programable.

Desventajas

- Dificultad para entornos gráficos.
- Entorno desconocido.

Programa "Hola Mundo"

```
/* El primer programa
   como aparece en todos los
   libros */

/**  Esto es un comentario
 *   para ser procesado con 'javadoc'.
 *   Clase "ejecutable" Hola mundo.
 */
public class HolaMundo {
    // Clase úpblica. óSlo puede haber una en cada fichero.
    // áAdems el fichero se debe llamar 'PrimerPrograma.java'
    /**  El émtodo áesttico "main" es el
     *   que comienza a ejecutarse.
     * @author Luis Fernando Llana íDaz
     * @version 1.0
     * @since 5/oct/98
     * @param args son los parametros introducidos en la ílnea de comandos
     *
     */
    public static void main (String[] args){
        System.out.println("Hola");
    }
}
```

Elmentos básicos

- Instrucción de asignación.
- Instrucción condicional (`if`).
- Instrucción iterativa (`while`, `for`, `do-while`).

Tipos Básicos

Tipo	Que contiene	Rango
`byte`	entero de 8 bits	-128 a 128
`short`	entero de 16 bits	-32.768 a 32.767
`int`	entero de 32 bits	-2.147.483.648 a 2.147.483.647
`long`	entero de 64 bits	-2^{63} a $2^{63}-1$
`float`	coma flotante de 32 bits	6 dígitos significativos $(10^{-46}, 10^{38})$
`double`	coma flotante de 64 bits	15 dígitos significativos $(10^{-324}, 10^{308})$
`char`	Carácter Unicode	`'a', 'b', ... 'á', 'ñ',... '0', '1', ...`
`boolean`	valores booleanos	`true` y `false`

Literales

Enteros:

- Decimal: 37
- Hexadecimal: 0x25
- Octal: 045

Caracteres

- `'a', 'b', ... 'á', 'ñ', ..., '0', '1', ...`
- Caracteres "escape": `'\n', '\\', '\'', '\"'`.

Reales

Float: `1.0345F, 1.04E-12f, .0345f, 1.04e-13f`

Double: `1.0345, 1.0345d, 5.6E-120D`

Cadenas de caracteres: `"hola\n"`.

Declaración de variables

- Estilo C++.
- Permite declaración de variables al vuelo.
- Java es sensible a mayúsculas y minúsculas.
- Notación:
  - Nombres de variables y clases largos.
  - Las variables todas en minúsculas, si una variable se compone de varias palabras, la segunda y siguientes palabras empiezan por mayúscula:

    `numero`, `numeroPalabras`
  - Las clases empiezan por mayúsculas y las demás en minúsculas:

    `Lista`, `ListaEnlazada`

Declaración de variables

```
public class Variables {
  public static void main (String[] args)
    {
      int num1;  // óDeclaracin áestndar
      double media=2.5; // óDeclaracin e óinicializacin

      num1=6;
      System.out.println("num1: "+num1);

      int x, num2; // óDeclaracin al vuelo
      x=2; num2=10;
      System.out.println("x: "+x+"\nnum2: "+num2);

      int num3=x*num2;  // óDeclaracin e óinicializacin al vuelo
      System.out.println("num3: "+num3);

      media=num1/num2;
      // media=0
      System.out.println("media: "+media);

      media=(double)num1/num2;
      // media=0.6
      System.out.println("media: "+media);
    }
}
```

Operadores

Operadores aritméticos

- + suma para enteros y reales.
- − resta para enteros y reales.
- ∗ producto para enteros y reales.
- / división entera entre enteros y con decimales para reales.
- % resto de la división entera entre enteros.

Operadores lógicos

Operadores lógicos

- conjunción &&, ¡ojo! no confundir con &.
- disyunción || , ¡ojo! no confundir con |.
- negación !.

Operadores relacionales

- igualdad ==.
- desigualdad !=.
- comparación <, <=, >, >=.

Operadores lógicos

Expresión condicional

```
i%2 ? 1:0
.........
mes==2 && bisiesto ? 29:28
```

Operadores lógicos no estrictos

```
if (x!=0 && 1/x>3)
..................
while (i<=n && v[i]==0)
```

Constantes

```
public static final double CUALQUIERA=3.14159265;                    1
public static final int MAXIMO_NUMERO_ELEMENTOS=5000;                2
```

Cadenas de caracteres

```
public class PrString {

    public static void main(String args[]) {
        String s1 = "Hola";
        String s2 = "Patata";
        String s3 = s1+" "+s2;
        System.out.println(s3.length());
        for (int i = 0; i < s3.length(); i++) {
            String mensaje = "Letra "+i+":"+s3.charAt(i)+":";
            System.out.println(mensaje);
        }

    }
}
```

`String` es una clase *especial* en Java. Sus objetos son inmutables.

Instrucción condicional

```
if (expresión booleana)
  instrucción

if (expresión booleana)
  instrucción
else
  instrucción
```

```
if (x==0)
   System.out.println("x vale 0");
else {
   System.out.print("x vale ");
   System.out.println(x);
}
```

Instrucción condicional

```
public class PruebaIf {
  public static void main(String[] args) {
      int x=0;

       if (x!=0 && 1/x=0);
         System.out.println(1/x);
       else
         System.out.print("x áest ");
         System.out.println("indefinido");
}
```

Instrucciones iterativas

```
while (expresión booleana)                                    1
  instrucción                                                 2
```

```
do                                                            1
  instrucción                                                 2
while (expresión booleana)                                    3
```

```
for (inicialización; expresión booleana; incremento)          1
  instrucción                                                 2
```

Instrucciones iterativas

```
i=0;
while (i<n && v[i]==0) i++;
```

```
1 for (int i=1; i<n; i++){
2   int posMin=i;
    for (int j=i+1; j<n; j++){
      if (v[j]<v[posMin]) posMin=j;
    }
    intercambiar(v,i,posMin);
}
```

Usos "prohibidos" de bucle `for`

Uso de bucle `for` como bucle `while`:

```
for (i=0; i<n && v[i]==0; i++);
```

1

Instrucciones iterativas

```
public class PruebaBucles {
    static void bucleWhile() {
        int i=1;
        boolean para = false;
        System.out.println("Bucle WHILE");
        while (i<=1000 && !para) {
            System.out.println("i: "+i);
            // Sale del bucle cuando se cumple la ócondicin.
            if (i==10) {
                System.out.println("Me voy....");
                para = true;
            }
            i++;
        }
    }

    static void bucleFor() {
        System.out.println("Bucle FOR");
        for (int i=1; i<=10; i++)
            System.out.println("i: "+i);
    }
    // éDespus del bucle la varaible 'i' no áest definida;

    public static void main(String[] args) {
        bucleWhile();
        bucleFor();
    }
}
```

Distinción de casos

```
switch (n%7){
case 0:
  System.out.println("Es lunes");
  break;
case 1:
  System.out.println("Es martes");
  break;
  ...
  ...
case 6:
  System.out.println("Es domingo");
  break;
default:
  System.out.println("Esto es imposible");
}
```

Distinción de casos

```
public class PruebaSwitch {
    static char prueba1(char c) {
        char letra;

        switch (c) {
        case 'a':
            letra='-';
            break;
        case 'e':
            letra='-';
            break;
        case 'i':
            letra='-';
            break;
        case 'o':
            letra='-';
            break;
        case 'u':
            letra='-';
            break;
        default: letra=c;
        }
        return letra;
    }
```

Distinción de casos

```
static void prueba2(int i) {
    switch (i){
    case 1:
        System.out.println("Caso 1: "+i);
    case 2:
        System.out.println("Caso 2:"+i);
    case 3:
        System.out.println("Caso 3:"+i);
    case 4:
        System.out.println("Caso 4:"+i);
    case 5:
        System.out.println("Caso 5:"+i);
    default:
        System.out.println("Caso por defecto");
    }
}
```

Distinción de casos

```
  public static void main(String[] args) {
      char letra=prueba1(args[0].charAt(0));
      System.out.println("letra:"+letra+":");

      System.out.println("llamada con 1");
      prueba2(1);
      System.out.println("llamada con 2");
      prueba2(2);
      System.out.println("llamada con 3");
      prueba2(3);
      System.out.println("llamada con 4");
      prueba2(4);
      System.out.println("llamada con 5");
      prueba2(5);
      System.out.println("llamada con 6");
      prueba2(6);
    }
}
```

Métodos

Java no tiene procedimientos, sólo hay funciones (métodos).

- Existen funciones que devuelven nada (tipo `void`).
- Todos los parámetros son de entrada, pero cuidado con los punteros o referencias.
- Las funciones admiten sobrecarga.

Sobrecarga

```
public class Sobrecarga {
    final static double PI=3.1415926535897932;

    static double max(double a, double b){
        double m=a;
        if (b>a) m=b;
        return m;
    }
    static double max(double a){
        double m=PI;
        if (a>PI) m=a;
        return m;
    }
    static int max(int a, int b){
        int m=a;
        if (b>a) m=b;
        return m;
    }
```

Sobrecarga

```
    public static void main(String [] args){
        double x=Double.parseDouble(args[0]);
        double y=Double.parseDouble(args[1]);
        int a=Integer.parseInt(args[2]);
        int b=Integer.parseInt(args[3]);

        System.out.print("áMximo entre "+x+" y "+y+": ");
        System.out.println(max(x,y));

        System.out.print("áMximo entre "+x+" y "+PI+": ");
        System.out.println(max(x));

        System.out.print("áMximo entre "+a+" y "+b+": ");
        System.out.println(max(a,b));
    }
}
```

Arrays

```
int [] v;  //definicion de arrays
double [][] m; //ódefinicin de matrices
.....................................
int [] v = new int[10]; //ódefinicin y ócreacin de array de 10  componentes
.....................................
int [] v = {1,2,3,4,5}; //ódefinicin y ócreacin de array
.....................................
int [] m = new int[10][10];
.....................................
int [][] m = {{1,2,3},{4,5,6},{7,8,9}};
```

Todos los arrays empiezan a numerarse desde 0;

Arrays

```
import java.util.Random;
public class PrArray {
    private static final int N=10;
    private static void escribe(int [] v) {
        System.out.print("[");
        for (int i = 0;  i<v.length ; i++) {
            if (i>0)
                System.out.print(",");
            System.out.print(v[i]);
        }
        System.out.println("]");
    }
    public static void main(String [] args) {
        int [] v = new int[N];
        genera(v);
        escribe(v);
        ordena(v);
        escribe(v);
        int [] u = v;
        u[0]=10000;
        escribe(u);
    }
```

Arrays

```
    private static void genera(int [] v) {
        Random r = new Random();
        for (int i = 0;  i<v.length ; i++)
            v[i] = r.nextInt() % (3*N);
    }
    private static void ordena(int [] v) {
        for (int i = 0; i < v.length ; i++) {
            int pos = i;
            for (int j = i+1; j<v.length ; j++)
                if (v[j]<v[pos])
                    pos = j;
            int aux = v[i];
            v[i] = v[pos];
            v[pos]=aux;
        }
    }
}
```

Paquetes

Hechos

- Cualquier programa puede tener varios cientos o miles de clases.
- La propia API de Java tiene más de 3000 clases.
- No es difícil escoger el nombre de una clase que ya esté cogido.

Paquetes

Hechos

- Cualquier programa puede tener varios cientos o miles de clases.
- La propia API de Java tiene más de 3000 clases.
- No es difícil escoger el nombre de una clase que ya esté cogido.

Paquete

Es una colección de clases con un mismo objetivo.

Clase soporte.Teclado

```
package soporte;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Teclado {
  ...
  ...
}
```

Esqueleto de proyecto Java

```
.
|-- build.xml
|-- prj.el
`-- src
|   |-- pr1
|   |   `-- Pr.java
|   `-- pr2
|       `-- Pr.java
|-- lib
`-- classes
    |-- pr1
    |   `-- Pr.class
    `-- pr2
        `-- Pr.class
```

- *Objeto* es un ejemplar de una clase.
  - Estado interno, no manipulable directamente.
  - Métodos de manipulación.
- *Programa* colección de clases.
- *Clase* definición de objetos.
  Similar a `RECORD` de PASCAL + métodos de manipulación.

Java no es un lenguaje orientado a objetos puro, tiene tipos primitivos.

Implementación de fechas

Implementación de fecha:

- Representación de una fecha ordinaria (día/mes/año) moderna (desde 01/01/1601).
- Dada una fecha, pasar a la siguiente, anterior, añadir/restar días.
- Comparar fechas.
- ¿Cuántos días hay entre 2 fechas?

Uso de fechas

```
import fecha.Fecha;
public class PrFecha {
    public static void main (String [] args) {
        Fecha f1=new Fecha(30,Fecha.DICIEMBRE,1999);
        Fecha f2=new Fecha(1,Fecha.MARZO,2004);
        while (f1.compareTo(f2)<=0) {
            System.out.println(f1.diaSemana()+":"+f1);
            f1.siguiente();
        }
    }
}
```

Implementación de fechas

```
package fecha;
public class Fecha implements Comparable{
    private int dia;
    private int mes;
    private int anyo;
    public Fecha( int _dia, int _mes, int _anyo ) {
        dia = _dia; mes = _mes; anyo = _anyo;
    }
    public String toString( ) {
        return dia + "/" + mes + "/" + anyo;
    }
    public int compare(Object o)
        ..........
    }
    public void anyadir(int ídas) {
        ..........
    }
    public void siguiente() {
        anyadir(1);
    }
    public void anterior() {
        anyadir(1);
    }
    public int diasHasta(Fecha otra) {
        ......
    }
}
```

Implementación *más real*

- Establecemos una fecha inicial *el día 0*, para simplificar el 01/01/1601 ¿por qué?.
- Representamos cada fecha como el número de días transcurridos desde el día 0.
- Necesitamos métodos de *traducción*

Clase Fecha

```
package fecha;

public class Fecha implements Comparable, Cloneable{
    //Constantes para los meses
    public final static int ENERO=0;
    public final static int FEBRERO=1;
.....................................
    public final static int DICIEMBRE=11;

    //Constantes para los ídas
    public final static int LUNES=0;
    public final static int MARTES=1;
.....................................
    public final static int DOMINGO=6;
.....................................
.....................................
.....................................

    //íDas transcurridos desde del 1 de enero de 1601
    private int diasDesdeInicio;
    public Fecha(int dia, int mes, int anyo) {
        compruebaFecha(dia, mes , anyo);
        diasDesdeInicio=calculaDiasDesdeInicio(dia, mes, anyo);
    }
}
```

Clase Fecha

```
public void anyadirDias(int inc) {
    int d=diasDesdeInicio+inc;
    if (d<0) throw new FechaFueraDeRango("íDa anteror al permitido");
    diasDesdeInicio=d;
}

public void siguiente() {
    anyadirDias(1);
}
public void anterior() {
    anyadirDias(-1);
}

public boolean equals(Object obj) {
    if (! (obj instanceof Fecha)) return false;
    return diasDesdeInicio==((Fecha)obj).diasDesdeInicio;
}

public int compareTo(Object o) {
    if (!(o instanceof Fecha))
        throw new ClassCastException("Se requiere un objeto de clase Fecha");
    return diasDesdeInicio-((Fecha)o).diasDesdeInicio;
}
```

Clase Fecha

```
1  private static int calculaDiasDesdeInicio(int elDia, int elMes, int elAnyo){
2      int dias;
3
4      dias=calulaDiasHastaPrimeroDe(elAnyo);
5      dias=dias+calculaDiasHastaPrimeroDelMes(elMes,elAnyo);
6      dias=dias+elDia-1; // el 1/1/1601 es el dia 0
7      return dias;
8  }
```

Clase Fecha

```
protected static int diasMes(int mes, int anyo) {
    int[] diasMes={31, 28, 31,
                   30, 31, 30,
                   31, 31, 30,
                   31, 30, 31};
    int dias = diasMes[mes] + ( (mes==FEBRERO)?anyoBisiesto(anyo):0 );
    return dias;
}
private static int calculaDiasHastaPrimeroDelMes(int mes, int anyo){
    int dias = 0;
    for (int i = 0; i < mes; i++) {
        dias = dias + diasMes(i,anyo);
    }
    return dias;
}

private static int calulaDiasHastaPrimeroDe(int elAnyo){
    int dias;
    int diff = elAnyo - ANYO_INICIO;
    dias = diff*365; // se anyaden 365 dias por cada ñao
    dias = dias + diff / 4; //ñaadimos 1 por cada úmltiplo de 4
    dias = dias - (diff/100); //se quida 1 por cada úmltimo de 100
    dias = dias + (diff/400); //se ñaade 1 por cada úmltimo de 400

    return dias;
}
```

Clase Fecha

```
    public String toString(){
        FechaTerna f = new FechaTerna(diasDesdeInicio);
        return f.toString();
    }
.....................................
.....................................
class FechaTerna {
    private int dia;
    private int mes;
    private int anyo;
    public FechaTerna(int _dia, int _mes, int _anyo ) {
        dia = _dia; mes = _mes; anyo = _anyo;
    }
    public String toString() {
        return dia+"/"+mes+"/"+anyo;
    }
.....................................
}
```

Clase Fecha

```
private static final int grupo1 = 365;
private static final int grupo4 = 4*365+1;
private static final int grupo100 = 25*grupo4-1;
private static final int grupo400 = 4*grupo100+1;

public FechaTerna(int dias) {
    int quedan = dias;
    int grupos400 = quedan / grupo400;
    quedan = quedan % grupo400;
    int grupos100 = quedan / grupo100;
    quedan = quedan % grupo100;
    int grupos4 = quedan / grupo4;
    quedan = quedan % grupo4;
    int grupos1 = quedan / grupo1;
    quedan = quedan % grupo1;
    int anyo = Fecha.ANYO_INICIO +
               grupos1 + 4*grupos4 +
               100*grupos100 + 400*grupos400;
    int mes=0;
    while (quedan>=Fecha.diasMes(mes,anyo)) {
        quedan = quedan-Fecha.diasMes(mes,anyo);
        mes++;
    }
    this.dia = quedan+1;
    this.mes = mes+1;
    this.anyo = anyo;
}
```

Terminología

clase

Definición de objetos. Todo objeto pertenece a una clase.

```
Fecha f1=new Fecha(30,Fecha.DICIEMBRE,1999);
Fecha f2=new Fecha(1,Fecha.MARZO,2004);
```

```
package fecha;

public class Fecha implements Comparable{
.....................................................
.....................................................
}

class FechaTerna {
.....................................................
.....................................................
}
```

Terminología

constructor

Procedimiento que construye objetos

```
public Fecha(int dia, int mes, int anyo) {                    1
    compruebaFecha(dia, mes , anyo);                          2
    diasDesdeInicio=calculaDiasDesdeInicio(dia, mes, anyo);   3
}                                                             4
```

```
public FechaTerna(int _dia, int _mes, int _anyo ) {           1
    dia = _dia; mes = _mes; anyo = _anyo;                     2
}                                                             3
```

En Java los constructores tienen el mismo nombre que la clase.

Terminología

atributo

Variable de un objeto

```
private int diasDesdeInicio;                                    1
```

```
private int dia;                                                1
private int mes;                                                2
private int anyo;                                               3
```

Normalmente los atributos son *privados* al objeto: desde fuera del objeto no deben ser accesibles, se debe acceder a ellos a través de *métodos de acceso*.

Terminología

métodos

```
while (f1.compareTo(f2)<=0) {
    System.out.println(f1.diaSemana()+":"+f1);
    f1.siguiente();
}
.....................................
public int compareTo(Object o) {
    if (!(o instanceof Fecha))
        throw new ClassCastException("Se requiere un objeto de clase Fecha");
    return diasDesdeInicio-((Fecha)o).diasDesdeInicio;
}
.....................................
public void anyadirDias(int inc) {
    int d=diasDesdeInicio+inc;
    if (d<0) throw new FechaFueraDeRango("íDa anteror al permitido");
    diasDesdeInicio=d;
}
public void siguiente() {
    anyadirDias(1);
}
```

Restricción de permisos

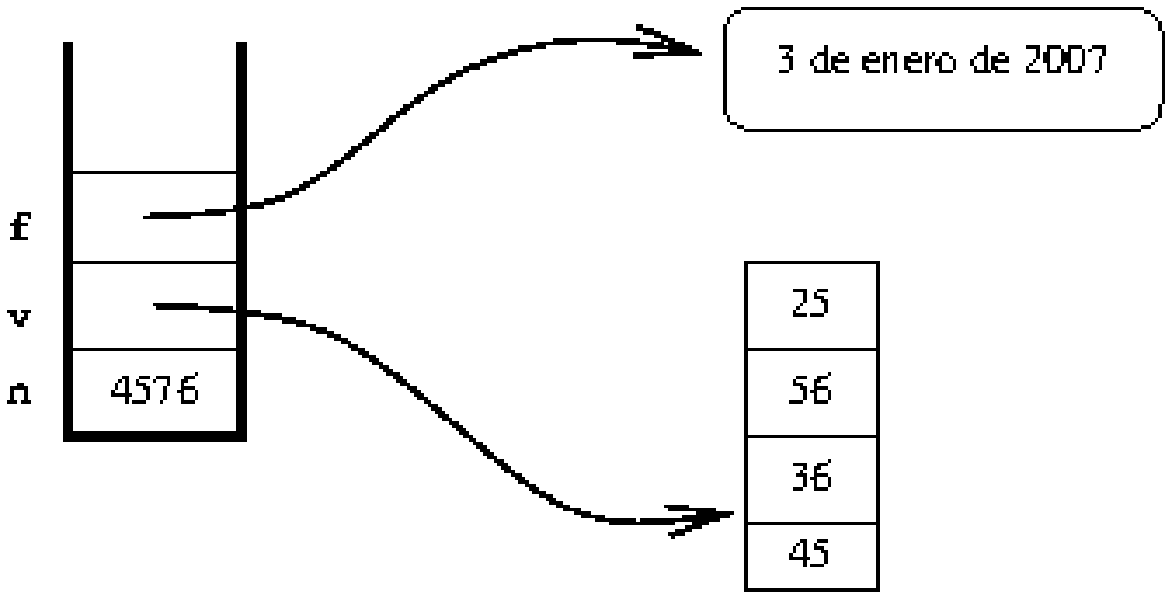
Los objetos pertenecen a clases, las clases se agrupan en paquetes.
Conviene restringir el acceso a los elementos del objeto

public	desde cualquier clase (independientemente del paquete).
protected	desde clases que están el mismo paquete.
private	sólo desde la misma clase.

Referencias

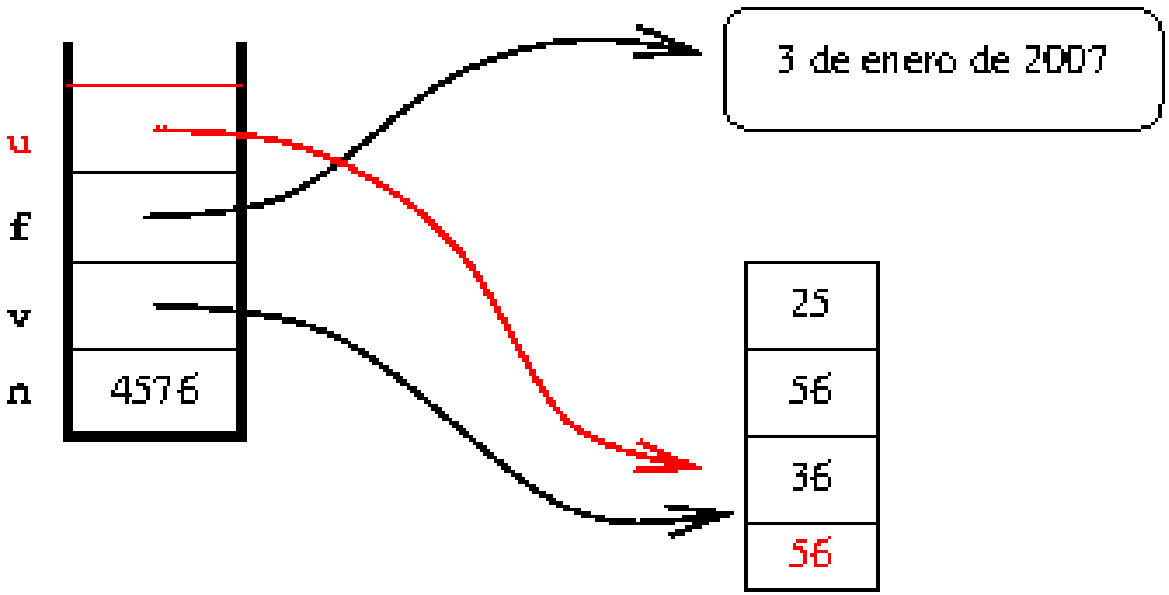
Tanto las variables de objetos como los arrays son *referencias*

```
int n=4576;                                                    1
int [] v={45,36,56,25};                                        2
Fecha f = new Fecha(3,Fecha.DICIEMBRE,2007);                   3
```



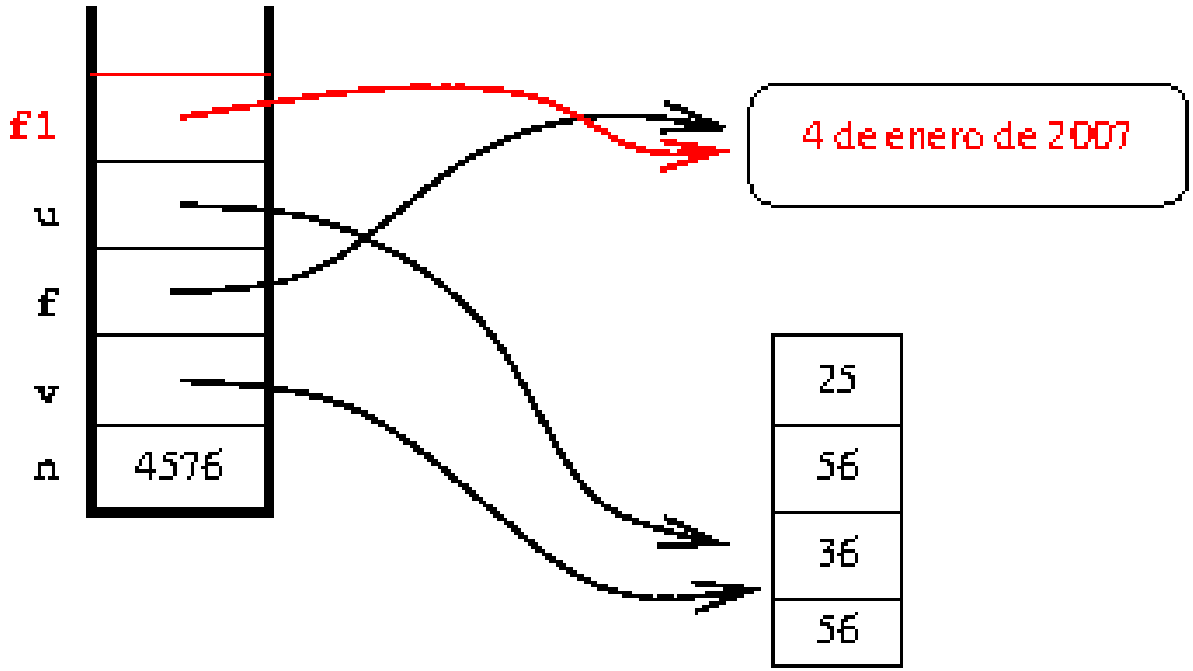
Referencias, efecto lateral

```
int [] u=v;                    1
u[0]=56; // v[0]=56            2
```



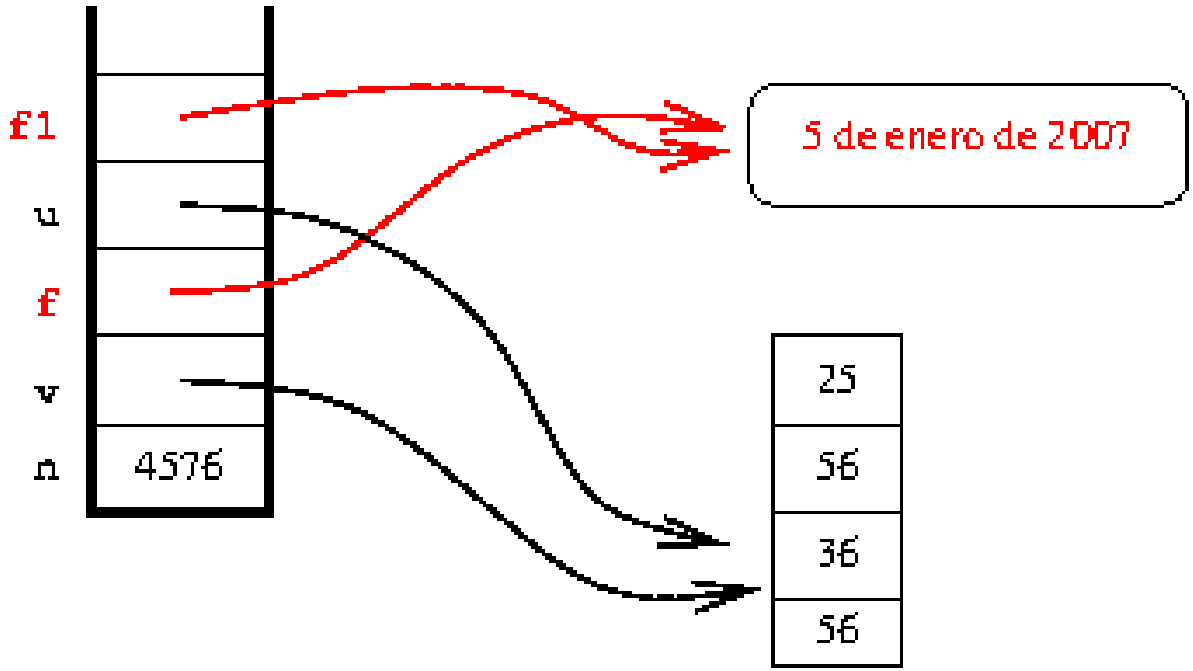
Referencias, efecto lateral

```
Fecha f1 = f;                    1
f1.siguiente();                  2
```



Referencias, objeros como parámetros

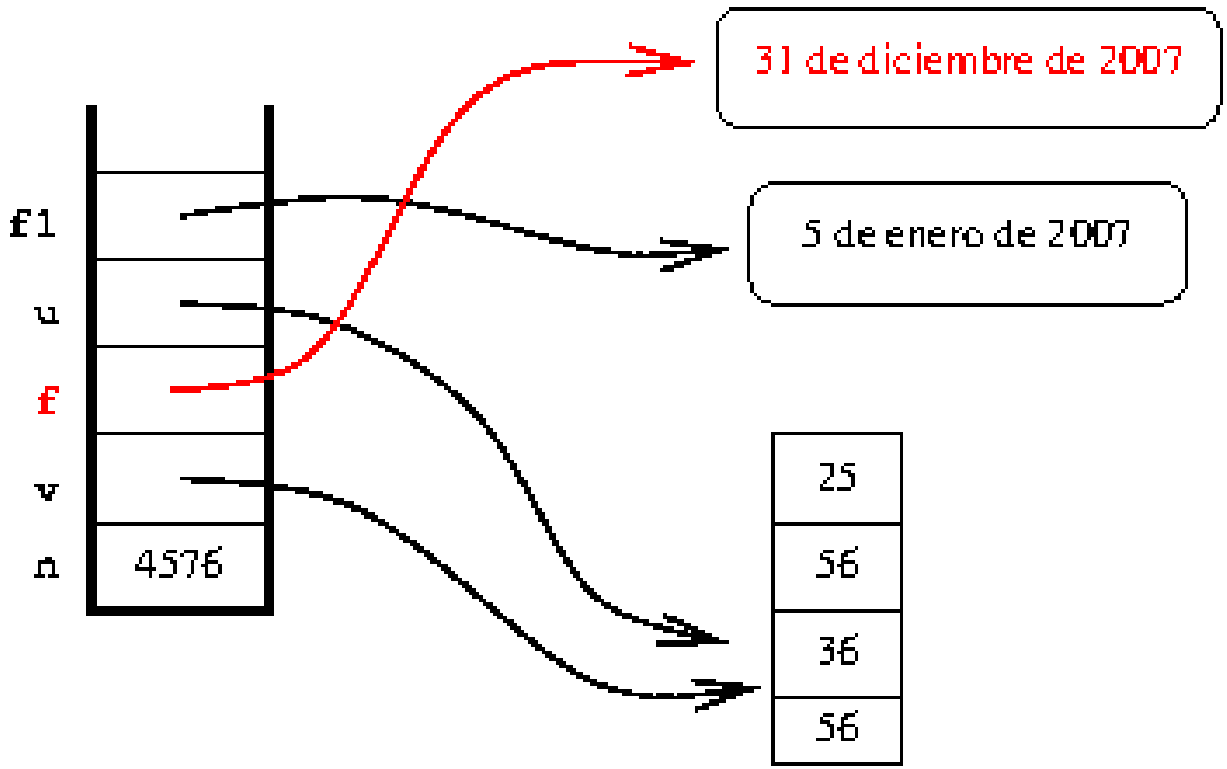
```
void avanza(Fecha fecha) {
  fecha.siguiente();
}

avanza(f);
```



Referencias, cambiar de referencia

```
f = new Fecha(31,Fecha.DICIEMBRE,2007);
```

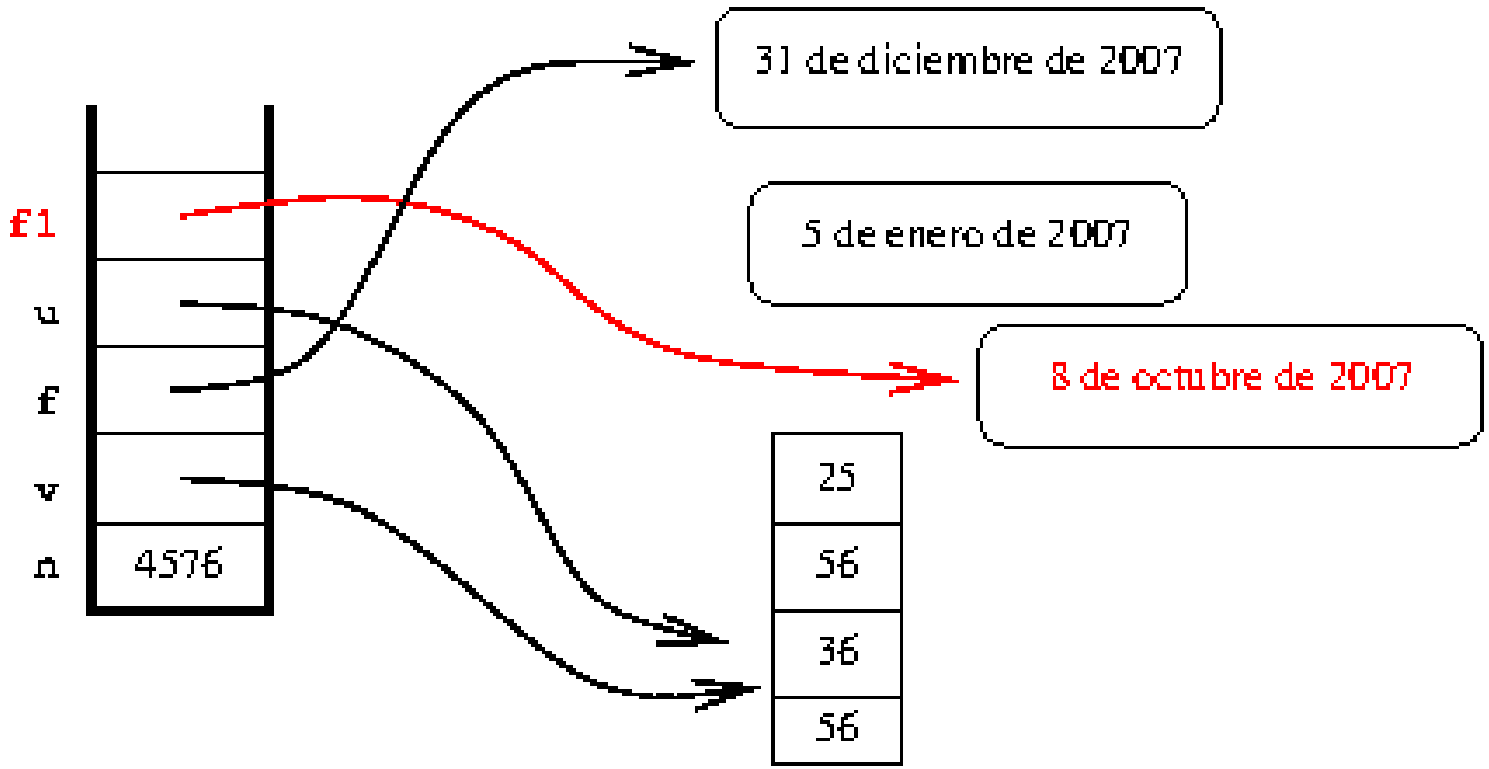
1



Referencias, generación de basura

```
f1 = new Fecha(8,Fecha.OCTUBRE,2007);
```

1



Referencias

El contenido de las variables de objetos y arrays son *referencias* a los objetos y arrays.

Referencias

El contenido de las variables de objetos y arrays son *referencias* a los objetos y arrays.

- Si dos variables hacen referencia al mismo objeto las modificaciones tienen *efectos laterales*

Referencias

El contenido de las variables de objetos y arrays son *referencias* a los objetos y arrays.

- Si dos variables hacen referencia al mismo objeto las modificaciones tienen *efectos laterales*
- Los parámetros de los procedimientos son *por valor*: Las referencias no pueden cambiar, pero el contenido de los objetos sí.

Referencias

El contenido de las variables de objetos y arrays son *referencias* a los objetos y arrays.

- Si dos variables hacen referencia al mismo objeto las modificaciones tienen *efectos laterales*
- Los parámetros de los procedimientos son *por valor*: Las referencias no pueden cambiar, pero el contenido de los objetos sí.
- Cuando un objeto no tiene referencias es considerado *basura*, que debe se recogida por el *recolector de basura* (*garbage collector*). En Java existe un recolector de basura automático.

Calificativo `static`

Algo estático *pertenece* a la clase. Es común a todos los objetos de la clase.

- Variables comunes a todos los objetos.
- Métodos que no hacen referencia a atributos dinámicos (no estáticos).

Permiten mejor aprovechamiento de los recursos.

Calificativo `final`

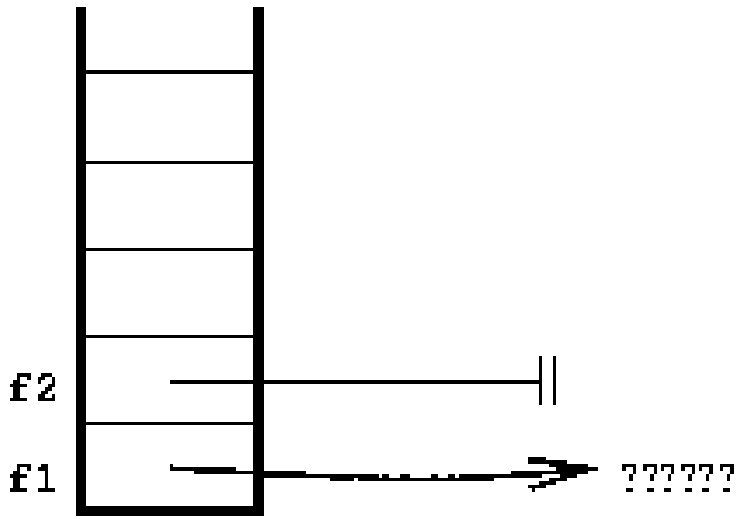
Se añade las partes que no se pueden cambiar

Variables constantes.

Métodos que no se pueden reescribir en la *herencia*.

Referencia null

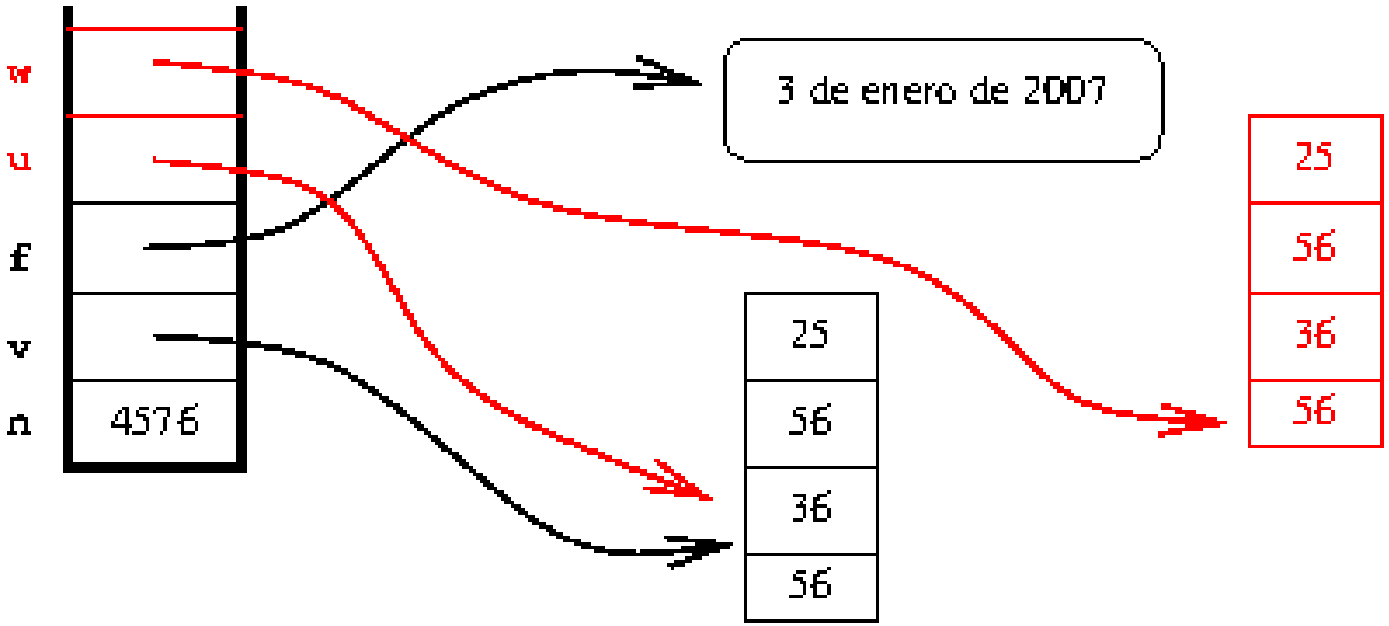
```
Fecha f1; //Referencia no definida                    1
Fecha f2=null; //Referencia a objeto nulo.            2
```



Igualdad de referencias

Dos referencias son iguales *si y sólo si* apuntan al mismo objeto

```
boolean b;                                          1
int [] u = v;                                       2
b = u==v; // b es true                              3
int [] w = new int[4];                              4
w[0]=56; w[1]=36; w[2]=56; w[3]=25;                 5
b = u==w; // b es false                             6
```



Igualdad de referencias

```
public class Fecha {
   ...........
    public Fecha clone() {
        Fecha fecha = new Fecha();
        fecha.diasDesdeInicio = this.diasDesdeInicio;
        return fecha;
    }
     ..........
}
............
Fecha f = new Fecha(1,ENERO,1956);
Fecha f1 = f; // referencias iguales;
Fecha f2 = f.clone(); // referencias distintas a objeros iguales.
```

Igualdad de referencias

```
boolean b;                          1
b = f==f1; // b es true;            2
b = f==f2; // b es false;           3
b = f1==f2; // b es false;          4
```

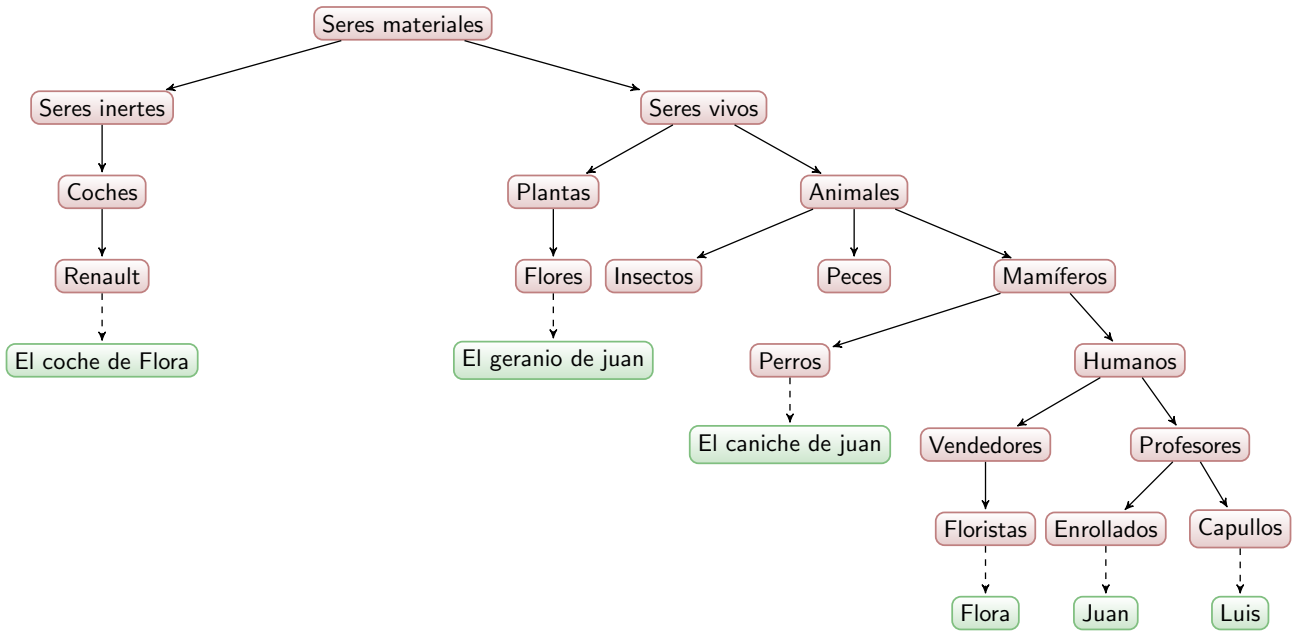
```
boolean b;                          1
b = f.equals(f1); // b es true;     2
b = f.equals(f2); // b es true;     3
b = f1.equals(f2); // b es true;    4
b = f1.equals(f); // b es true;     5
b = f2.equals(f1); // b es true;    6
b = f2.equals(f); // b es true;     7
```

Igualdad de referencias

```
f.siguiente();                                  1
b = f==f1; // b es true;                        2
b = f==f2; // b es false;                       3
b = f1==f2; // b es false;                      4
```

```
boolean b;                                      1
b = f.equals(f1); // b es true;                 2
b = f.equals(f2); // b es false;                3
b = f1.equals(f2); // b es false;               4
b = f1.equals(f); // b es true;                 5
b = f2.equals(f1); // b es false;               6
b = f2.equals(f); // b es false;                7
```

Herencia: relación *es un* I



Herencia: relación *es un* II

Recorrido de la jerarquía

Según se recorre la jerarquía los elementos que están por debajo heredan características de los elementos superiores:

- Todos los objetos tienen masa y volúmen.
- Todos los seres vivos nacen, crecen, etc. . . .
- Todos los mamíferos maman cuando son pequeños, etc. . . .
- Todos los perros ladran.

Herencia: relación *es un* III

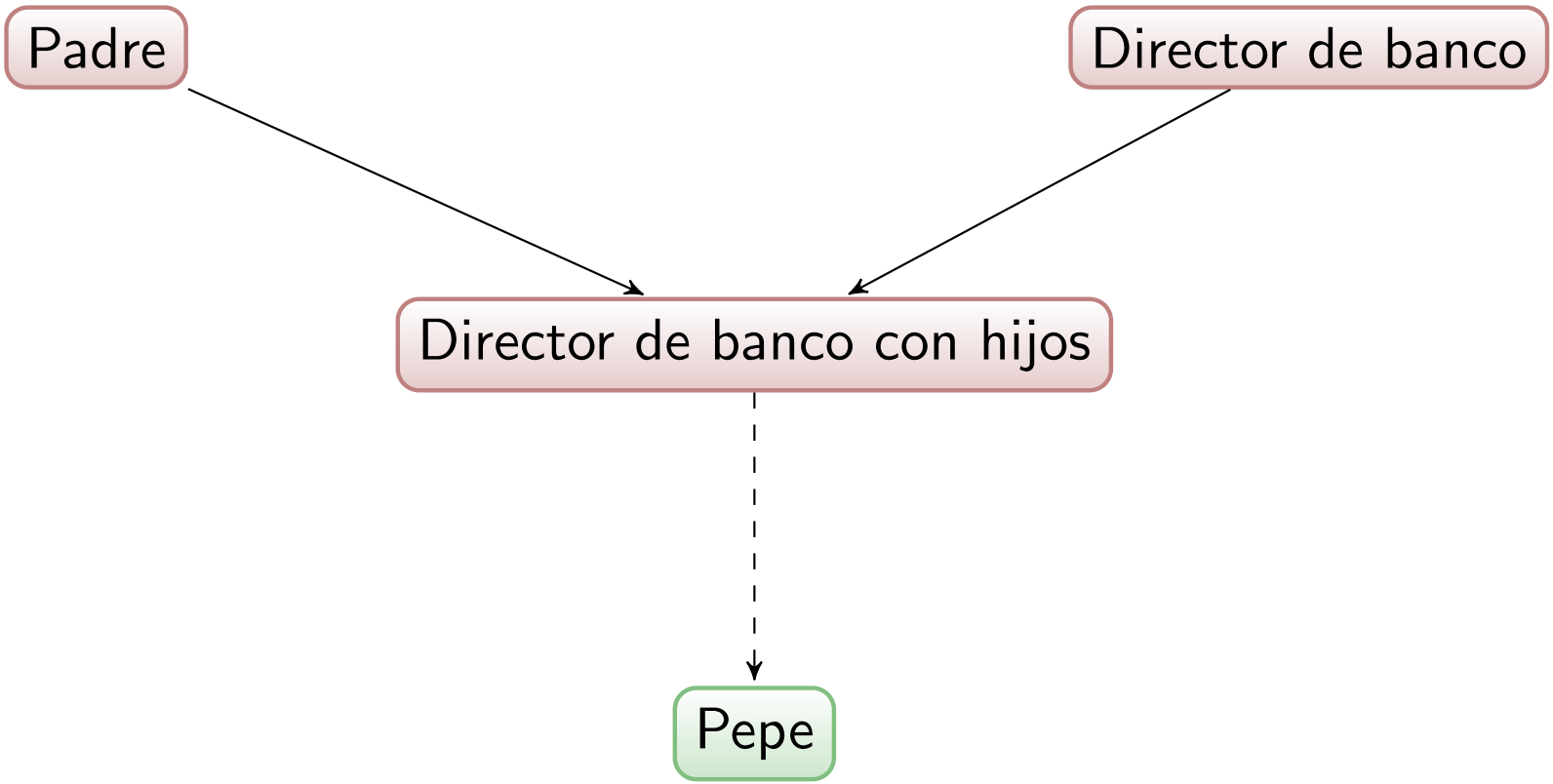
Por tanto el perro de juan

- tienen masa y volúmen.
- nace, crece, etc. . . .
- mamó cuando fue pequeño, etc. . . .
- ladra.

Características heredadas

- Existen características que no cambian (peso y volúmen).
- Caracteríticas que se añaden.
- Características que se modifican, se refinan o cambian.

Herencia múltiple



Herencia en Java

- Interfaces.
- Clases abstractas.
- Extensión de clases.
- Implementación de interfaces.

Clase `Object`

```
public class Patata {
    .......
    .......
    .......
}
```

```
public class Patata extends Object {
    .......
    .......
    .......
}
```

Clase `Object`

La clase `Object` es la superclase de todas las clases en Java.
Dispone, entre otros, de los métodos:

- `boolean equals(Object obj)`
- `String toString()`

Estos métodos se deben sobreescribir en las subclases.

Clase Fecha

```
public boolean equals(Object obj) {
    if (! (obj instanceof Fecha)) return false;
    return diasDesdeInicio==((Fecha)obj).diasDesdeInicio;
}
```

```
public String toString(){
    FechaTerna f = new FechaTerna(diasDesdeInicio);
    return f.toString();
}
```

Extensión de clases

Extensión de clase

```
package fecha;
public class FechaFueraDeRango extends RuntimeException {
  public FechaFueraDeRango(String s) {
      super(s);
    }
}
```

Extensión de interfaz

```
public class Fecha implements Comparable{
  .........
  .........
  .........
}
```

Clases, clases abstracas e interfaces

Clase normal: todalmente definida, puede haber objetos de esa clase.

Clase abstracta: parcialmente definida, puede tener algún atributo, métodos totalmente definidos y métodos no definidos.

Interfaz: sólo define algún método, pero sin implementar.

Clase Abstracta

```
public abstract class Patata{
  private int atributo;
  public abstract modifica(int x);
}
```

Interfaz

El interfaz `Comparable` del API de Java *debe ser* de la forma

```
package java.lang;
public interface Comparable {
  public int compareTo(Object o);
}
```

Herencia múltiple

```
public class Patata {
  public int x;
  public void incrementa(){
    x=x+1;
  }
}
```

```
public class Tomate {
  public int x;
  public void incrementa(){
    x=x+10;
  }
}
```

```
public class Lechuga extends Patata, Tomate {
  public int y;
}
......
......
Lechuga lechuga = new Lechuga();
lechuga.incrementa(); // incrementa x = x+1 ó x = x+10 ?
```

Herencia múltiple en Java

En Java existe versión restringida de herencia

- Se puede extender sólo 1 clase.
- Se puende implementar más de 1 interfaz.

```
public class Trazador extends Frame                                    1
                      implements WindowListener,ObjetoComparable {     2
```

Objeto super

Invocación a la superclase

```
public class Estudiante{
  private String nombre;
  public Estudiante(String nombre) {
    this.nombre = nombre;
  }
  public void estudia(double tiempo){....}
}
```

```
public class EstudiantePerezoso extends Estudiante{
   public EstudiantePerezoso(String nombre) {
     super(nombre);
   }
   public void trabaja(double tiempo){
     super.estudia(tiempo/3);
     descansa(tiempo/3);
     super.estudia(tiempo/3);
   }
}
```

Objeto this

```
public class Fecha {
    ..................
    public Fecha(int dia, int mes, int anyo) {
        compruebaFecha(dia, mes , anyo);
        diasDesdeInicio=calculaDiasDesdeInicio(dia, mes, anyo);
    }

    public Fecha() {
        this(1,1,1998);
    }
    ..................
}
```

```
 class FechaTerna {
   private int dia;
   private int mes;
   private int anyo;
   ....................
   public FechaTerna(int dias) {
        this.dia = quedan+1;
        this.mes = mes+1;
        this.anyo = anyo;
    }
    ........
}
```

Figuras planas

```
package geometria.figurasPlanas;                                    1
public interface FiguraPlana{                                       2
    public boolean intersecaCon(FiguraPlana otra);                  3
    public double distancia(Punto p);                               4
}                                                                   5
```

```
package geometria.figurasPlanas;                                    1
public abstract class Superficie implements FiguraPlana{            2
    public abstract boolean estaDentro(Punto P);                    3
    public abstract double superficie();                            4
}                                                                   5
```

Puntos

```
public class Punto implements FiguraPlana{
    public Punto(double x, double y){
        posX=x; posY=y;
    }
    public double distancia(Punto otro){
        Vector v=new Vector(this,otro);
        return v.modulo();
    }
    public boolean equals(Object o){
        if (o instanceof Punto) {
            Punto otro = (Punto)o;
            return (this.posX==otro.posX) && (this.posY==otro.posY);
        }
        else return false;
    }
    public boolean intersecaCon(FiguraPlana otra){
        if (otra instanceof Punto) {
            return this.equals(otra);
        }
        else {
            return otra.intersecaCon(this);
        }
    }
}
```

Método equals

Definido en Object, todos los objetos son subclase suya: todos tienen definido el método equals.

```
public boolean equals(Object o){
    if (o instanceof Punto) {
        Punto otro = (Punto)o;
        return (this.posX==otro.posX) && (this.posY==otro.posY);
    }
    else return false;
}
```

Interfaces Comparable y Cloneable

```
public class Fecha implements Comparable, Cloneable{                    1
  ..................................                                     2
    public int compareTo(Object o) {                                     3
        if (!(o instanceof Fecha))                                       4
            throw new ClassCastException("Se requiere un objeto de clase fecha.Fecha."+  5
                                         "He recibido algo de clase "+o.getClass().getNa  6
        return diasDesdeInicio-((Fecha)o).diasDesdeInicio;               7
    }                                                                    8
    public Fecha clone() {                                               9
        Fecha fecha = new Fecha();                                       10
        fecha.diasDesdeInicio = this.diasDesdeInicio;                    11
        return fecha;                                                    12
    }                                                                    13
  ......................                                                 14
}                                                                        15
```

Interfaz Comparable

```
package java.lang;
public interface Comparable<T> {
  public int compareTo(T o);
}
```

Se usa para:

- ordenación.
- colecciones ordenadas.

```
public class Fecha implements Comparable<Fecha>, Cloneable{
    ............
    ............
    public int compareTo(Fecha o) {
        return diasDesdeInicio-o.diasDesdeInicio;
    }
}
```

Interfaz `Comparable`

- *Autoriza* el uso del método `clone` de la clase object.
- Copia los atributos, no hace un `clone` de los atributos.
- Si se quiere hacer público hace falta reescribirlo.

Estático y dinámico

Estático

Conocido en tiempo de compilación.

Dinámico

Lo que se conoce en tiempo de ejecución.

Tipo estático

```
public class Fecha {
  ....................
    public boolean equals(Object obj) {
      ..............
      ..............
    }
  ....................
}
```

¿Qué sabemos de obj?

El objeto al que hace referencia obj pertenece a un subclase de Object.

Tipo dinámico

```
public class Fecha {
  ..................
    public boolean equals(Object obj) {
      ............
      ............
    }
  ..................
      Fecha f1 = new Fecha(31,JULIO,2006);
      Fecha f2 = new Fecha(31,JULIO,2005);
      boolean b = f1.equals(f2);
  ........
}
```

¿Qué sabemos de obj?

f2 es de clase Fecha. Cuando se hace la llamada obj va a ser una referencia a un objeto de clase Fecha.

Calificativo `static`

Los métodos y atributos marcados como `static`

- son inherentes a la clase,
- todos los objetos los comparten ,
- se pueden invocar directamente desde la clase.

```
public class Fecha {
  ..................
  ..................
  public final static int ENERO=0;
  ................
}
........
    Fecha f =  new Fecha(1,Fecha.ENERO,2005);
```

Si un atributo es estático

- Se comparte por todos los objetos de la clase

Calificativo `static`

```
public class Fecha {
  ..................
  ..................
      private static int calculaDiasDesdeInicio(int elDia, int elMes, int elAnyo){
        int dias;

        dias=calulaDiasHastaPrimeroDe(elAnyo);
        dias=dias+calculaDiasHastaPrimeroDelMes(elMes,elAnyo);
        dias=dias+elDia-1; // el 1/1/1601 es el dia 0
        return dias;
    }

  ..................

}
```

Si un método es estático

- Se gana en eficiencia (tiempo/memoria)
- No puede hacer referencia a atributos no estáticos.

Permisos de acceso

Los métodos y atributos de una clase pueden ser

`public`: accesibles desde cualquier clase.

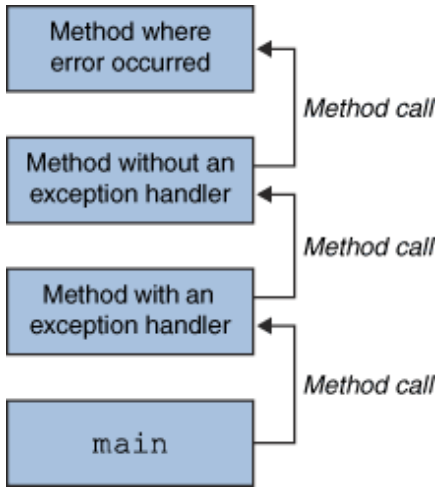
`protected`: accesibles desde clases que estén en el mismo paquete o desde subclases.

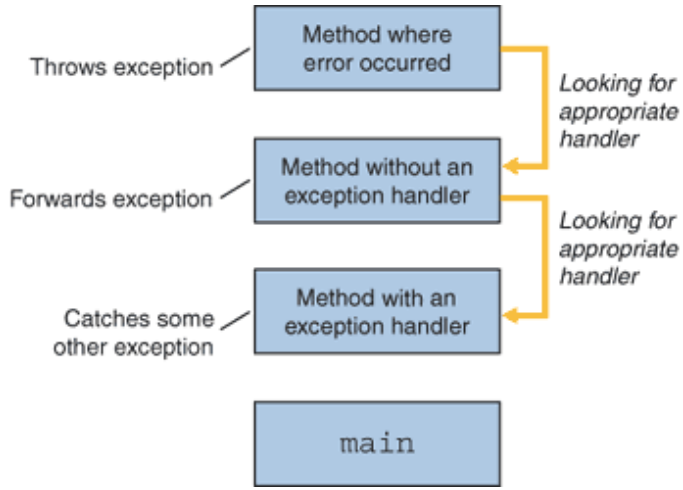
`private` sólo es accesible desde la propia clase.

¿Qué es una excepción? I

Excepción

Es un evento que ocurre durante la ejecución de un programa e interrumpe su ejecución normal





¿Qué es una excepción? II

Cuando se produce una excepción se puede

- Capturarla y arreglar la situación
- Transmitir el error al procedimiento llamante

Capturar/Transmitir una excepción

```
public static int leeInt() throws IOException  {
    try  {
        return Integer.parseInt(teclado.readLine());
    }
    catch (NumberFormatException e) {
        System.out.println("úNmero incorrecto, ádmelo otra vez.");
        return leeInt();
    }
}
```

- El método `readLine` puede lanzar `java.io.IOException`
- El método `parseInt` puede lanzar `java.lang.NumberFormatException`

Instrucción `try/catch/finally`

```
 try {
    codigo
} catch (Excepcion_1 e_1) {
    codigo_1
} catch (Excepcion_2 e_2) {
    codigo_2
}
..........
} finally {

}
```

- Intenta ejecutar el *código*.
- Si se produce una excepción de clase $Excepcion_k$ se ejecuta el código $codigo_k$
- El código en `finally` se ejecuta al final en cualquier caso.

Instrucción try/catch/finally

```
public void writeList() {
    PrintWriter out = null;
    try {
        System.out.println("Entering try statement");
        out = new PrintWriter(
                            new FileWriter("OutFile.txt"));
            for (int i = 0; i < SIZE; i++)
                out.println("Value at: " + i + " = "
                                + vector.elementAt(i));
    } catch (ArrayIndexOutOfBoundsException e) {
         System.err.println("Caught "
                        + "ArrayIndexOutOfBoundsException: "
                        +   e.getMessage());
    } catch (IOException e) {
         System.err.println("Caught IOException: "
                                +  e.getMessage());
    } finally {
         if (out != null) {
             System.out.println("Closing PrintWriter");
             out.close();

         }
         else {
             System.out.println("PrintWriter not open");
         }
     }
}
```

Declarar/lanzar excepciones

Requisito captura o declarar

Si una instrucción en un método puede lanzar una excepción se debe

- Capturar con instrucción `catch`
- El método debe declarar (`throws`) que la puede lanzar.

Excepciones normales deben cumplir el requisito

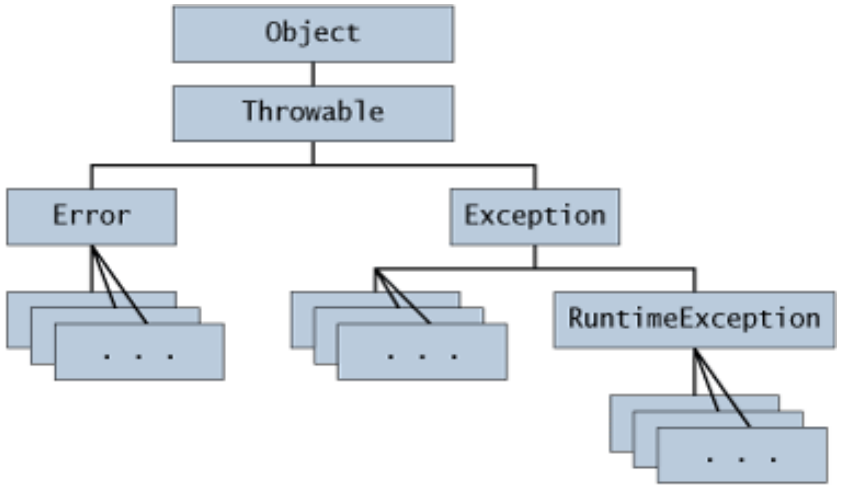
Errores errores externos (fallos de hardaware). No necesitan cumplir el requisito.

Excepciones *Runtime* errores internos que se debe a fallos, usos incorrectos de una API. No necesitan cumplir el requisito.

Lanzar excepciones

```
public void anyadirDias(int inc) {
    int d=diasDesdeInicio+inc;
    if (d<0) throw new FechaFueraDeRango("íDa anteror al permitido");
    diasDesdeInicio=d;
}
```

Escribiendo excepciones




```
package fecha;
public class FechaFueraDeRango extends RuntimeException {
  public FechaFueraDeRango(String s) {
      super(s);
    }
}
```

¿Para qué?

```
private static List leePuntos(String fichero)
    throws FileNotFoundException, IOException{

    BufferedReader lector = new BufferedReader(new FileReader(fichero));
    List listaPuntos=new LinkedList();
    String datos=lector.readLine();
    while (datos!=null){
        Punto p=new Punto(datos);
        listaPuntos.add(p);
        datos=lector.readLine();
    }
    return listaPuntos;
}
```

Evitar conversiones de tipo

```
private static void analizaListaPuntos(List c){
    Iterator itr=c.iterator();
    int i=1;
    while(itr.hasNext()){
      Punto p = (Punto)itr.next();
      ..........................
    }
}
```

Evitar conversiones de ejecución

```
Lista lista = leePuntos(fichero);                                    1
lista.add(new Recta(new Punto(0,0), new Punto(1,0)));                2
analizaPuntos(lista);                                                3
```

Definiendo clases genéricas

```
public interface List <E>{
    void add(E x);
    Iterator<E> iterator();
}

public interface Iterator<E>{
    E next();
    boolean hasNext();
}
```

Subclases

```
List<Punto> ls = new ArrayList<Punto>();                    1
List<FiguraPlana> lo = ls; // Es admisible?                 2
```

```
lo.add(new Recta(A,B)); // ísera legal.                     1
```

Comodines

```
1 void muestra(Collection<Object> c) {
2   for (Object e : c) {
3       System.out.println(e);
4   }
```

¿Puedo pasarle algo de tipo `Lista<FiguraPlana>`?

```
1 void muestra(Collection<?> c) {
2   for (Object e : c) {
3       System.out.println(e);
4   }
```

Es una colección de tipo desconocido, pero al menos deben ser objetos.

Comodines Acotados

```
private static void muestra(List<? extends FiguraPlana> c){
    Iterator itr=c.iterator();
    int i=1;
    while(itr.hasNext()){
        System.out.format("Elemento %d "+itr.next(),i);
        System.out.println();
        i++;
    }
}
```

```
    List<FiguraPlana> listaFiguras=leeFiguras(nombreFiguras);
    List<Punto> listaPuntos=leePuntos(nombrePuntos);
    System.out.println("Puntos:");
    muestra(listaPuntos);
    System.out.println("Figuras:");
    muestra(listaFiguras);
```

Métodos genéricos

```
1  void insterta(Collection<?> c, FiguraPlana [] figuras) {
2    for (FiguraPlana f : figuras) {
3      c.add(f); //Ilegal, no se el tipo base de figura.
4    }
5  }
```

```
1  void insterta(Collection<? extends FiguraPlana> c, FiguraPlana [] figuras) {
2    for (FiguraPlana f : figuras) {
3      c.add(f);  //Ilegal, no se el tipo base de figura.
4    }
5  }
```

```
private static <Fig extends FiguraPlana> void insterta(Collection<Fig> c,
                                                       Fig [] figuras) {
  for (Fig f : figuras) {
    c.add(f);
  }
}
```

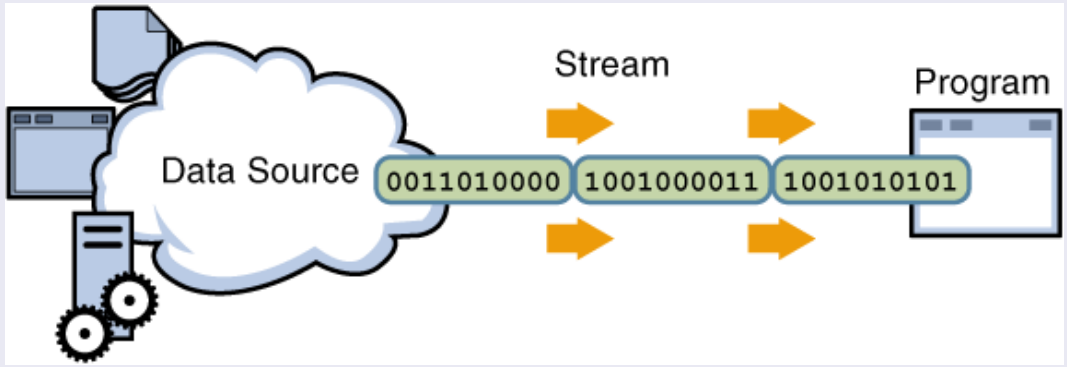
```
interface Sink<T> {
    flush(T t);
}
public static <T> T writeAll(Collection<T> coll, Sink<T> snk) {
    T last;
    for (T t : coll) {
        last = t;
        snk.flush(last);
    }
    return last;
}

.....................

Sink<Object> s;
Collection<String> cs;
String str = writeAll(cs, s); // Ilegal.
```

```
public static <T> T writeAll(Collection<? extends T>, Sink<T>) {
...
}
...
String str = writeAll(cs, s); // Devuelve algo de clase Object
```

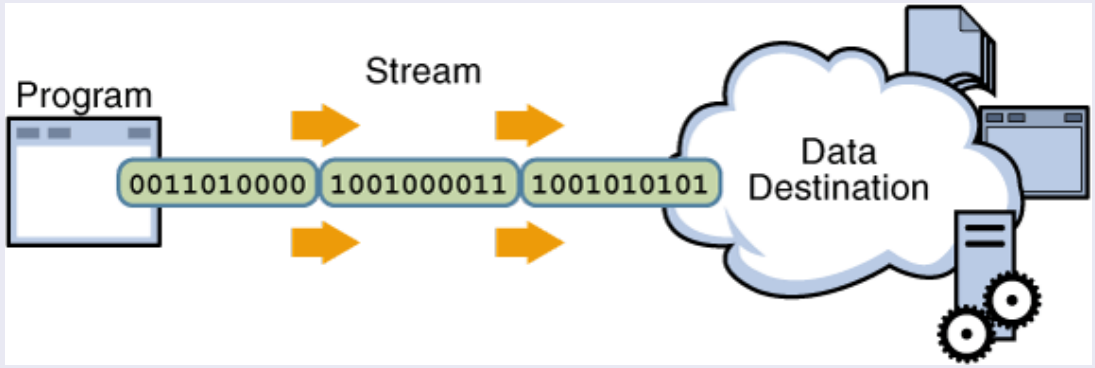
```
public static <T> T writeAll(Collection<T>, Sink<? super T>) {
...
}
...
String str = writeAll(cs, s); // Devuelve algo de clase Object
```

Flujos de entrada/salida

Stream de entrada



Stream de entrada



Byte Streams

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopiaBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream(args[0]);
            out = new FileOutputStream(args[1]);
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }

        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
                }
        }
    }
}
```

Llamada desde la línea de comandos

```
luis@casa:~/docencia/Java06-07$ java CopiaBytes quijote.txt quijote2.txt
```

```
public static void main(String[] args) throws IOException {
    ..............
}

args.length() ⇝ 2
args[0] ⇝ "quijote.txt"
args[1] ⇝ "quijote2.txt"
```

Streams caracteres

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopiaCaracteres {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader(args[0]);
            outputStream = new FileWriter(args[1]);

            int c = inputStream.read();
            while (c!= -1) {
                outputStream.write(c);
                c = inputStream.read();
            }
        } finally {
            if (inputStream != null) {
                    inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

Streams Líneas

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;

public class CopiaLineas {
    public static void main(String[] args) throws IOException {
        BufferedReader inputStream = null;
        PrintWriter outputStream = null;
        try {
            inputStream = new BufferedReader(new FileReader(args[0]));
            outputStream = new PrintWriter(new FileWriter(args[1]));

            String l = inputStream.readLine();
            while ( l!= null ) {
                outputStream.println(l);
                l = inputStream.readLine();
            }
        } finally {
            if (inputStream != null)
                inputStream.close();
            if (outputStream != null)
                outputStream.close();
        }
    }
}
```

Buffering

```
inputStream =
    new BufferedReader(new FileReader(args[0]));
outputStream =
    new BufferedWriter(new FileWriter(args[1]));
.........................
.........................
outputStream.flush(); //vaciamos el buffer de escritura
```

Leyendo cosas que no son caracteres

- Métodos `Integer.parseInt(java.lang.String)`, `Double.parseDouble(java.lang.String)`
- Clase `java.util.Scanner`

clase Scanner

```
import java.io.*;
import java.util.Scanner;

public class Escaner {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        try {
            s = new Scanner(new BufferedReader(new FileReader(args[0])));

            while (s.hasNext()) {
                System.out.println(s.next());
            }
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

clase Scanner

```
import java.io.*;
import java.util.Scanner;

public class SumaScan {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        try {
            s = new Scanner(new BufferedReader(new FileReader(args[0])));
            double suma =0;
            while (s.hasNext()) {
                if (s.hasNextDouble()) {
                    suma += s.nextDouble();
                } else {
                    String d = s.next();
                    suma += Double.parseDouble(d);
                    System.out.println("Error:"+d);
                }
            }
            System.out.format("Suma: %s\n",suma);
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

clase Scanner

Fichero numeros.txt

```
1.994.986,34
2.000.000,00
3.000.000,00
3000000
10.11
```

```
~/Java$ export CLASSPATH=./classes/
~/Java$ /opt/java/jdk1.5/bin/java SumaScan numeros.txt
Error:10.11
Suma: 9994996.45
```

Construyendo puntos

```
public class Punto implements FiguraPlana{
    private double posX,posY;
    public Punto(String s){
        Scanner scan = new Scanner(s);
        scan.useLocale(Locale.US);
        posX=scan.nextDouble();
        posY=scan.nextDouble();
    }
    ..............
    ..............
}
```

Construyendo puntos

```
public class Punto implements FiguraPlana{
    private double posX,posY;
    public Punto (String s){
        Scanner scan = new Scanner(s);
        posX=Double.parseDouble(scan.next());
        posY=Double.parseDouble(scan.next());
    }
    ................
    ................
}
```

Fichero de figuras

Fichero `figuras.txt`

```
circulo
5 5 2
segmento
3 4 8 4
recta
3 5 3 6
recta
0 1 1 1
circulo
0 2 0.5
recta
0 2 4 5
paralelogramo
0 4 2 2 -2 2
paralelogramo
0 0 2 2 -2 2
paralelogramo
2 2 0 4 0 0
paralelogramo
0 0 2 2 4 0
punto
2 3
```

Leyendo puntos

Fichero `puntos.txt`

```
4 5
7 3
1 1
0.5 1
1 0.5
7 4
0 2
```

Leyendo

```
public class PruebaGeometriaI{
    private static void leeFiguras(String fichero) {..}

    private static void leePuntos(String fichero) {..}

    public static void main(String args[]) throws IOException,FileNotFoundException{
        String nombrePuntos=args[0];
        String nombreFiguras=args[1];
        System.out.println("Puntos:");
        leePuntos(nombrePuntos);
        System.out.println("Figuras:");
        leeFiguras(nombreFiguras);
    }
}
```

Leyendo puntos

```
private static void leePuntos(String fichero)
    throws FileNotFoundException, IOException{

    BufferedReader lector = new BufferedReader(new FileReader(fichero));
    String datos=lector.readLine();
    while (datos!=null){
        Punto p=new Punto(datos);
        System.out.println(p);
        datos=lector.readLine();
    }
}
```

Leyendo figuras

```
private static FiguraPlana construyeFigura(String tipoFigura, String datos){..}1
                                                                                2
private static void leeFiguras(String fichero)                                  3
    throws FileNotFoundException, IOException{                                  4
                                                                                5
    BufferedReader lector = new BufferedReader(new FileReader(fichero));        6
    String tipoFigura=lector.readLine();                                        7
    int i=1;                                                                    8
    while (tipoFigura!=null){                                                   9
        String datos=lector.readLine();                                         10
        FiguraPlana figura=construyeFigura(tipoFigura,datos);                   11
        System.out.println(figura);                                             12
        tipoFigura=lector.readLine();                                           13
    }                                                                           14
}                                                                               15
```

Construyendo figuras

```
private static FiguraPlana construyeFigura(String tipoFigura, String datos){
    FiguraPlana figura;
    if (tipoFigura.equals("punto")) figura=new Punto(datos);
    else if (tipoFigura.equals("recta"))  figura=new Recta(datos);
    else if (tipoFigura.equals("segmento"))  figura=new Segmento(datos);
    else if (tipoFigura.equals("paralelogramo"))  figura=new Paralelogramo(datos);
    else if (tipoFigura.equals("circulo"))  figura=new Circulo(datos);
    else throw new RuntimeException("Figura no implementada:"+tipoFigura+":");
    return figura;
}
```

Invocación desde la línea de comandos

Variable de entorno CLASSPATH

Debe apuntar al directorio raíz donde estén los ficheros .class

```
~/Java$ ls
build.xml  figuras.txt  prj.el          quijote3.txt  src
classes    lib          puntos.txt      quijote4.txt
CVS        numeros.txt  quijote2.txt    quijote.txt
~/Java$ echo $CLASSPATH

~/Java$ export CLASSPATH=./classes
~Java$ echo $CLASSPATH
./classes/
```

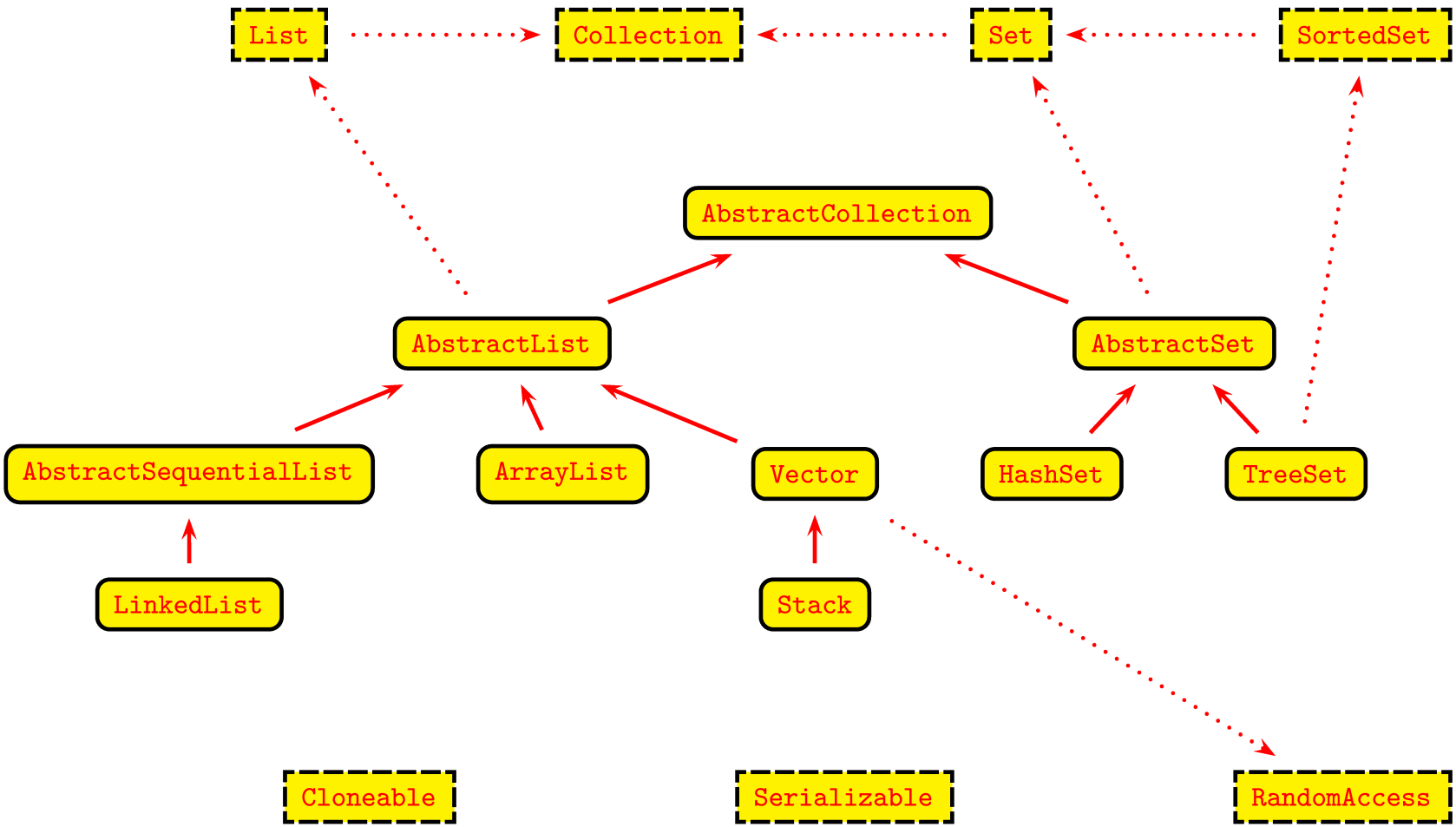
Invocación desde la línea de comandos

Variable de entorno JAVA_HOME

Debe apuntar al directorio donde está la instalación de Java.

```
~/Java$ ls ls -ald /opt/java/jdk1.5                                                    1
lrwxrwxrwx 1 root root 11 2006-07-18 07:53 /opt/java/jdk1.5 -> jdk1.5.0_07             2
~/Java$ echo $JAVA_HOME                                                                3
/opt/java/jdk                                                                          4
~/Java$ export JAVA_HOME=/opt/jdk/jdk1.5                                               5
~/Java$ echo $JAVA_HOME                                                                6
/opt/jdk/jdk1.5                                                                        7
~/Java$ $JAVA_HOME/bin/java PruebaGeometriaI puntos.txt figuras.txt                    8
Puntos:                                                                                9
Punto: (4.0,5.0)                                                                       10
..............                                                                         11
Punto: (0.0,2.0)                                                                       12
Figuras:í                                                                              13
Crculo:Punto: (5.0,5.0):2.0*                                                           14
Segmento: Punto: (3.0,4.0)-Punto: (8.0,4.0)                                            15
Recta: 1.0*X  +  0.0*Y = 3.0                                                           16
................................í                                                     17
Paraleppedo*Punto: (2.0,2.0):Punto: (0.0,4.0):Punto: (-2.0,2.0):Punto: (0.0,0.0)*í     18
Paraleppedo*Punto: (0.0,0.0):Punto: (2.0,2.0):Punto: (6.0,2.0):Punto: (4.0,0.0)*       19
Punto: (2.0,3.0)                                                                       20
```

Jerarquía de clases de colecciones



Colecciones

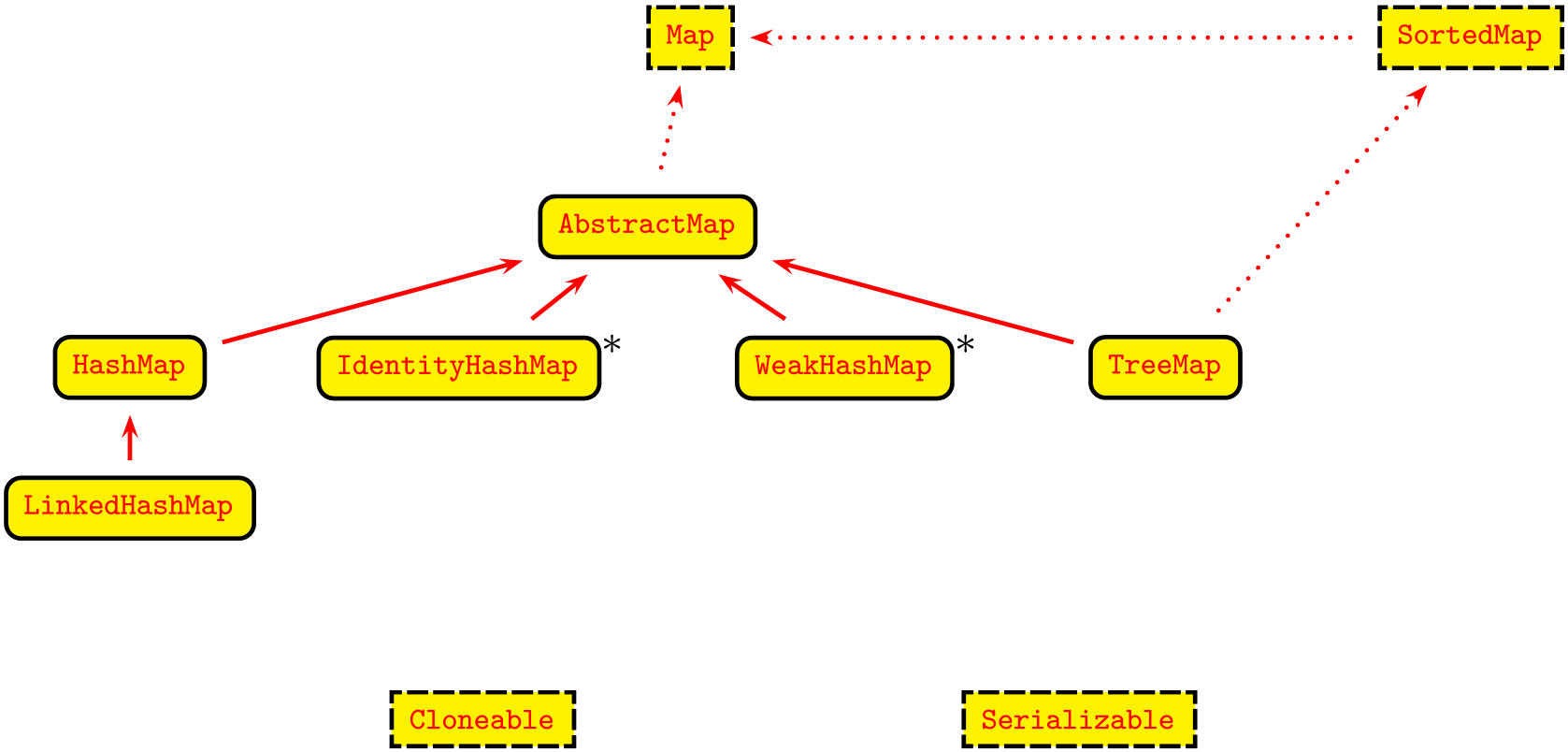
`Collection` Grupo de objetos.

`Set` Concepto matemático. El orden no importa, no hay repeticiones. `SortedSet`.

`List` El orden es importante, admite repeticiones.

`Map` Concepto de función matemática. `SortedMap`

Jerarquía de clases: Map



Interfaz Collection

```
public interface Collection<E> extends Iterable<E> {
    ..............................
    boolean add(E element);
    Iterator<E> iterator();
    ........................
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Interface Iterador

```
public Interface Iterator {
  public boolean hasNext();
  public Object next();
  void  remove();
}
```

Uso de listas

```
private static List leePuntos(String fichero)
    throws FileNotFoundException, IOException{

    BufferedReader lector = new BufferedReader(new FileReader(fichero));
    List listaPuntos=new LinkedList();
    String datos=lector.readLine();
    while (datos!=null){
        Punto p=new Punto(datos);
        listaPuntos.add(p);
        datos=lector.readLine();
    }
    return listaPuntos;
}
```

```
private static void analizaListaPuntos(List c){
    Iterator itr=c.iterator();
    int i=1;
    while(itr.hasNext()){
      Punto p = (Punto)itr.next();
      ..........................
    }
}
```

Errores en tiempo de ejecución

¿Qué pasaría si añadiera un objeto que no sea de clase Punto?

```
Lista lista = leePuntos(fichero);                          1
lista.add(new Recta(new Punto(0,0), new Punto(1,0)));      2
analizaPuntos(lista);                                      3
```

La línea

```
Punto p = (Punto)itr.next();                               1
```

lanzaría un `java.lang.ClassCastException`.

Clases genéricas

```
public Interface List<E> {
  ......
  public Iterator<E> iterator()
  .....
}
```

```
public Interface Iterator<E> {
  public boolean hasNext();
  public E next();
  void  remove();
}
```

Uso de listas con genéricos

```
private static List<Punto> leePuntos(String fichero)
    throws FileNotFoundException, IOException{

    BufferedReader lector = new BufferedReader(new FileReader(fichero));
    List<Punto> listaPuntos=new LinkedList<Punto>();
    String datos=lector.readLine();
    while (datos!=null){
        Punto p=new Punto(datos);
        listaPuntos.add(p);
        datos=lector.readLine();
    }
    return listaPuntos;
}
```

```
private static void analizaListaPuntos(List<Punto> c){
    Iterator<Punto> itr=c.iterator();
    int i=1;
    while(itr.hasNext()){
      Punto p = itr.next();
      ..........................
    }
}
```

Errores en tiempo de compilación

¿Qué pasaría si añadiera un objeto que no sea de clase Punto?

```
Lista<Punto> lista = leePuntos(fichero);                    1
lista.add(new Recta(new Punto(0,0), new Punto(1,0)));       2
```

No compila, la lista es de puntos y puedo añadir rectas.

Interfaz Collection

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);
    boolean remove(Object element);
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    void clear();

    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Los métodos `contains` y `remove` comparan los objetos según el método `equals`.

Interfaz `Set`

- No añade métodos.
- No puede haber elementos repetidos.

Subclases

`HashSet`	La mejor en eficiencia, los iteradores no garantizan el orden. Método `hashCode` en la clase `Object`.
`TreeSet`	Implementación con árboles roji-negros. Los iteradores recorren el conjunto según el orden establecido por sus elementos
`LinkedHashSet`	Otra implementación de las tablas hash.

Método `equals`

Definido en la clase Object. Hay que sobreescribirlo si se va a usar una clase como elemento de una colección.

- Debe ser una relación de orden: `o1!=null`, `o2!=null` y `o3!=null`
  - Reflexivo `o1.equals(o1)`
  - Simétrico `o1.equals(e2)==o2.equals(e1)`
  - Si `o1.equals(o2)` y `o2.equals(o3)` entonces `o1.equals(o3)`
- Si `o!=null` entonces `o.equals(null)==false`.
- Debe ser consistente: distintas invocaciones a lo largo de un programa debe dar el mismo resultado si no cambian el valor de los objetos.

Método `hashCode`

Definido en la clase Object. Hay que sobreescribirlo si se va a usar una clase como elemento de una colección tipo *hash*.

- Debe ser consistente.
- Si `o1.equals(o2)`, entonces `o1.hashCode()=o2.hashCode`.
- No debe ser necesariamente inyectiva, dos objetos distintos pueden tener el mismo valor.
- Debería distribuir uniformemente los objetos.

Método CompareTo

```
public class Fecha implements Comparable<Fecha>, Cloneable {
  ................
  public int compareTo(Fecha f) {
    ....
  }
  ..........
}
```

Implementa relación de orden

$$x.\texttt{compareTo} \begin{cases} < 0 & \text{si } x < y \\ = 0 & \text{si } x = y \\ > 0 & \text{si } x > y \end{cases}$$

Debe ser implementada para tener colecciones ordenadas.

Método CompareTo

- `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))`.
- Debe ser reflexiva antisimétrica y transitiva, para objetos no nulos
  - si `x.compareTo(x)==0`.
  - si `x.compareTo(y)<=0` y `y.compareTo(x)<=0` entonces `x.compareTo(z)==0` y `z.compareTo(x)==0`.
  - si `x.compareTo(y)<=0` y `y.compareTo(z)<=0` entonces `x.compareTo(z)<=0`.
- Debe ser consistente con el resto de los objetos. Si `x.compareTo(y)==0` entonces para todo `z` entonces `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`.
- es conveniente que sea consistente con equals: `x.equals(y)` sii `x.compareTo(y)==0`

PrColeccion

```
public class PrColeccion {
    public static void lee(String nombre, Collection<String> conjunto)
        throws IOException {
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader(nombre));
            String linea = reader.readLine();
            while (linea!=null) {
                conjunto.add(linea);
                linea = reader.readLine();
            }
        } finally {
            if (reader!=null) {
                reader.close();
            }
        }
    }
    public static void muestra(Collection<String> conjunto) {
        for (String s : conjunto) {
            System.out.println(s);
        }

    }
}
```

PrConjunto

```
public class PrConjunto {
    public static void main(String [] args) throws Exception{
        String tipo = args[1];
        Set<String> conjunto = null;
        if (tipo.equals("hash")) {
            conjunto = new HashSet<String>();
        } else if (tipo.equals("tree")) {
//          Collator c = Collator.getInstance(new Locale("es","ES"));
//          conjunto = new TreeSet<String>(c);
            conjunto = new TreeSet<String>();
        } else {
            throw new RuntimeException("Clase no implementada: "+tipo);
        }

        PrColeccion.lee(args[0],conjunto);
        PrColeccion.muestra(conjunto);
    }
}
```

Lista de nombres

Fichero nombres.txt

```
Luis
Almudena
JavierÁ
lvaro
Isabel
Marisa
Natalia
Albertoé
Andrs
Andres
Andros
Luis
Almudena
JavierÁ
lvaro
Isabel
Marisa
Natalia
Albertoé
Andrs
Andres
Andros
Luis
Almudena
JavierÁ
lvaro
```

Ejecución de conjuntos

```
~/Java$ $JAVA_HOME/bin/java PrConjunto nombres.txt tree
Alberto
Almudena
Andres
Andrés
Andrs
Isabel
Javier
Luis
Marisa
NataliaÁ
lvaro
```

Iteradores

Un iterador es un objeto que sirve para recorrer una estructura

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
}
```

```
    for (Iterator<String> it = coleccion.iterator(); it.hasNext(); )
      String s = s.next();
      System.out.println();
    }
```

```
    for (String s : coleccion) {
      System.out.println(s);
    }
```

Listas

El orden de los objetos es importante, puede haber repeticiones de objetos.

```
public interface List<E> extends Collection<E> {
    // Acceso posicional
    E get(int index);
    E set(int index, E element);     //optional
    boolean add(E element);          //optional
    void add(int index, E element); //optional
    E remove(int index);             //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional

    // úBsquedas
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteradores de listas
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Sublistas
    List<E> subList(int from, int to);
}
```

Iteradores de listas

```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void remove(); //optional
    void set(E e); //optional
    void add(E e); //optional
}
```

Implementaciones de listas

`ArrayList`	Generalmente más eficiente.
`LinkedList`	Eficiente cuando no se hacen accesos posicionales.

Maps

```
public interface Map<K,V> {

  // Operaciones ábsicas
  V put(K key, V value);
  V get(Object key);
  V remove(Object key);
  boolean containsKey(Object key);
  boolean containsValue(Object value);
  int size();
  boolean isEmpty();

  void putAll(Map<? extends K, ? extends V> m);
  void clear();

  // Vistas
  public Set<K> keySet();
  public Collection<V> values();
  public Set<Map.Entry<K,V>> entrySet();

  // Interface for entrySet elements
  public interface Entry {
      K getKey();
      V getValue();
      V setValue(V value);
  }
}
```

PrMap

```
public class PrMap {                                                    1
    private static void lee(String nombre,                              2
                            Map<String,Integer> map) throws IOException {3
        BufferedReader reader = null;                                   4
        try {                                                           5
            reader = new BufferedReader(new FileReader(nombre));        6
            String linea = reader.readLine();                           7
            while (linea!=null) {                                       8
                Integer i = map.get(linea);                             9
                if (i==null) {                                          10
                    i = new Integer(1);                                 11
                } else {                                                12
                    i = new Integer(i.intValue()+1);                    13
                }                                                       14
                map.put(linea,i);                                       15
                linea = reader.readLine();                              16
            }                                                           17
        } finally {                                                     18
            if (reader!=null) {                                         19
                reader.close();                                         20
            }                                                           21
        }                                                               22
    }                                                                   23
```

PrMap

```
    private static void muestra(Map<String,Integer> map) {
        for (String s: map.keySet()) {
            System.out.println(s+":"+map.get(s));
        }
    }
    public static void main(String [] args) throws Exception{
        String tipo = args[1];
        Map<String,Integer> map = null;
        if (tipo.equals("hash")) {
            map = new HashMap<String,Integer>();
        } else if (tipo.equals("tree")) {
            map = new TreeMap<String,Integer>(Collator.getInstance());
        } else {
            throw new RuntimeException("Clase no implementada: "+tipo);
        }

        lee(args[0],map);
        muestra(map);
    }
}
```

Implementaciones

`HashMap` Basadas en tablas hash.

`TreeMap` Basadas en árboles, cuando las claves se pueden ordenar.

Clases interesantes I

Paquete `java.util`

`Collections` Clase con métodos estáticos para manejar colecciones.

`Date` Para manejar instantes en el tiempo (UTC).

`GregorianCalendar` Para manejar fechas.

`Locale` Para ayudar a la *localización* de las aplicaciones.

`Random` Generación de números aleatorios.

Clases interesantes II

Paquete `java.lang`

Clases envoltorio `Integer`, `Double`, etc Clases de objetos *inmutables* para los tipos básicos.

`StringBuffer` Cadedas de caracteres de objetos *no inmutables*.

Expresiones regulares I

Comprobación de cadenas de caracteres

- Un número de factura es correcto si es de la forma 2345/07.
- Las facturas no se borran, si se borra una la marcamos *poniendo una R* delante.
- A veces interesa construir una factura a partir de unas existentes. Si tengo los números 0967/06, 0124/07, 0345/07, construimos la auxiliar AUX-0967/06-0124/07-0345/07.

Hacer una función que compruebe si una factura es correcta

Expresiones regulares II

Sustituciones genéricas

Tenemos un fichero de texto en el que hay muchas fechas con formato mm/dd/aaaa. Las queremos cambiar al formato aaaa/mm/dd.

Programa de prueba

```
private static void prueba(String patron,
                           String s) {
  Pattern p = Pattern.compile(patron);
  Matcher m = p.matcher(s);

  System.out.format("\nPatron:%s:\nCadena:%s:\n",patron,s);

  boolean enc = false;
  while (m.find()) {
      enc=true;
      System.out.format("Encontrado %s desde la óposicin %d "+
                        "hasta la óposicin %d\n",
                        m.group(),m.start(),m.end());
      for (int i=0; i<=m.groupCount(); i++) {
        System.out.format("Grupo %d:%s: desde la óposicin %d "+
                          "hasta la óposicin %d\n",
                          i,m.group(i),m.start(i),m.end(i));
      }
  }
  if (!enc) {
      System.out.println("No encontrado");
  }
}
```

Cadena de caracteres

Una cadena de caracteres casa consigo misma

```
Patron:foo:
Cadena:foofoofoo:
Encontrado foo desde la óposicin 0 hasta la óposicin 3
Grupo 0:foo: desde la óposicin 0 hasta la óposicin 3
Encontrado foo desde la óposicin 3 hasta la óposicin 6
Grupo 0:foo: desde la óposicin 3 hasta la óposicin 6
Encontrado foo desde la óposicin 6 hasta la óposicin 9
Grupo 0:foo: desde la óposicin 6 hasta la óposicin 9
```

Caracter simple

El carácter . casa con cualquier caracter. Es un *meta-carácter* (no significan ellos mismos)

```
Patron:gato.:
Cadena:gatos:
Encontrado gatos desde la óposicin 0 hasta la óposicin 5
Grupo 0:gatos: desde la óposicin 0 hasta la óposicin 5
```

Los meta-caracteres son: `([{\^-$|]})?*+.` Si queremos hacer casar un meta-carácter hay que precederlo con \.

```
prueba("gato\\.","gato.");

Patron:gato\.:
Cadena:gato.:
Encontrado gato. desde la óposicin 0 hasta la óposicin 5
Grupo 0:gato.: desde la óposicin 0 hasta la óposicin 5
```

Caracteres

`[abc]`	casa con `a`, `b` y `c`.
`[a-m]`	casa con las letras desde la `a` hasta la `m`
`[a-mA-M]`	casa con las letras desde la `a` hasta la `m` *o* desde la `A` hasta la `M`
`[^abc]`	Casa on cualquier letra excepto con `a`, `b` y `c`.
`[a-z&&[^be]`	casa con las letras desde la `a` hasta la `z` excepto con `b` y `e`
`[a-z&&[^b-e]]`	casa con las letras desde la `a` hasta la `z` excepto desde la `b` hasta la `e`

Caracteres

```
Patron:[aeg]:                                                    1
Cadena:abcdefghijklmnopqrstuvxyz:                                2
Encontrado a desde la óposicin 0 hasta la óposicin 1             3
Grupo 0:a: desde la óposicin 0 hasta la óposicin 1               4
Encontrado e desde la óposicin 4 hasta la óposicin 5             5
Grupo 0:e: desde la óposicin 4 hasta la óposicin 5               6
Encontrado g desde la óposicin 6 hasta la óposicin 7             7
Grupo 0:g: desde la óposicin 6 hasta la óposicin 7               8
```

Caracteres

```
Patron:[^c-v]:
Cadena:abcdefghijklmnopqrstuvxyz:
Encontrado a desde la óposicin 0 hasta la óposicin 1
Grupo 0:a: desde la óposicin 0 hasta la óposicin 1
Encontrado b desde la óposicin 1 hasta la óposicin 2
Grupo 0:b: desde la óposicin 1 hasta la óposicin 2
Encontrado x desde la óposicin 22 hasta la óposicin 23
Grupo 0:x: desde la óposicin 22 hasta la óposicin 23
Encontrado y desde la óposicin 23 hasta la óposicin 24
Grupo 0:y: desde la óposicin 23 hasta la óposicin 24
Encontrado z desde la óposicin 24 hasta la óposicin 25
Grupo 0:z: desde la óposicin 24 hasta la óposicin 25
```

Caracteres

```
Patron:[a-z&&[^b-u]]:
Cadena:abcdefghijklmnopqrstuvxyz:
Encontrado a desde la óposicin 0 hasta la óposicin 1
Grupo 0:a: desde la óposicin 0 hasta la óposicin 1
Encontrado v desde la óposicin 21 hasta la óposicin 22
Grupo 0:v: desde la óposicin 21 hasta la óposicin 22
Encontrado x desde la óposicin 22 hasta la óposicin 23
Grupo 0:x: desde la óposicin 22 hasta la óposicin 23
Encontrado y desde la óposicin 23 hasta la óposicin 24
Grupo 0:y: desde la óposicin 23 hasta la óposicin 24
Encontrado z desde la óposicin 24 hasta la óposicin 25
Grupo 0:z: desde la óposicin 24 hasta la óposicin 25
```

Caracteres predefinidos

`.`	casa con cualquier carácter, puede o no casar con final línea.
`\d`	`[0-9]`
`\D`	`[^0-9]`
`\s`	espacio en blanco `[ \t\n\x0B\f\r]`
`\S`	`[^ \t\n\x0B\f\r]`
`\w`	`[a-zA-Z_0-9]`
`\W`	`[^a-zA-Z_0-9]`

Caracteres predefinidos

```
prueba("\\d","aaa bbb 1 df 3 f");

Patron:\d:
Cadena:aaa bbb 1 df 3 f:
Encontrado 1 desde la óposicin 8 hasta la óposicin 9
Grupo 0:1: desde la óposicin 8 hasta la óposicin 9
Encontrado 3 desde la óposicin 13 hasta la óposicin 14
Grupo 0:3: desde la óposicin 13 hasta la óposicin 14
```

Repeticiones

Voraz	No voraz	Posesivo	
`X?`	`X??`	`X?+`	0 ó 1 aparición
`X*`	`X*?`	`X*+`	0, 1 ó más
`X+`	`X+?`	`X++`	1 ó más
`X{n}`	`X{n}?`	`X{n}+`	`n` apariciones
`X{n,}`	`X{n,}?`	`X{n,}+`	al menos `n`
`X{n,m}`	`X{n,m}?`	`X{n,m}+`	entre `n` y `m`

```
Patron:.*foo:
Cadena:xfooxxxxxxfoo:
Encontrado xfooxxxxxxfoo desde la óposicin 0 hasta la óposicin 13

Patron:.*?foo:
Cadena:xfooxxxxxxfoo:
Encontrado xfoo desde la óposicin 0 hasta la óposicin 4
Encontrado xxxxxxfoo desde la óposicin 4 hasta la óposicin 13

Patron:.*+foo:
Cadena:xfooxxxxxxfoo:
No encontrado
```

Agrupaciones

(X)	Agrupación con captura
\n	*n*-ésimo de captura
(?:X)	Agrupación sin capturar

```
Patron:(?:foo){3}:
Cadena:foofoofoofoofoofoo:
Encontrado foofoofoo desde la óposicin 0 hasta la óposicin 9
Grupo 0:foofoofoo: desde la óposicin 0 hasta la óposicin 9
Encontrado foofoofoo desde la óposicin 9 hasta la óposicin 18
Grupo 0:foofoofoo: desde la óposicin 9 hasta la óposicin 18

Patron:(foo){3}\1:
Cadena:foofoofoofoofoofoo:
Encontrado foofoofoofoo desde la óposicin 0 hasta la óposicin 12
Grupo 0:foofoofoofoo: desde la óposicin 0 hasta la óposicin 12
Grupo 1:foo: desde la óposicin 6 hasta la óposicin 9
```

Agrupaciones

```
prueba("(\\d{1,2})/(\\d{1,2})/(\\d{4})","6/30/1950");

Patron:(\d{1,2})/(\d{1,2})/(\d{4}):
Cadena:6/30/1950:
Encontrado 6/30/1950 desde la óposicin 0 hasta la óposicin 9
Grupo 0:6/30/1950: desde la óposicin 0 hasta la óposicin 9
Grupo 1:6: desde la óposicin 0 hasta la óposicin 1
Grupo 2:30: desde la óposicin 2 hasta la óposicin 4
Grupo 3:1950: desde la óposicin 5 hasta la óposicin 9
```

Facturas

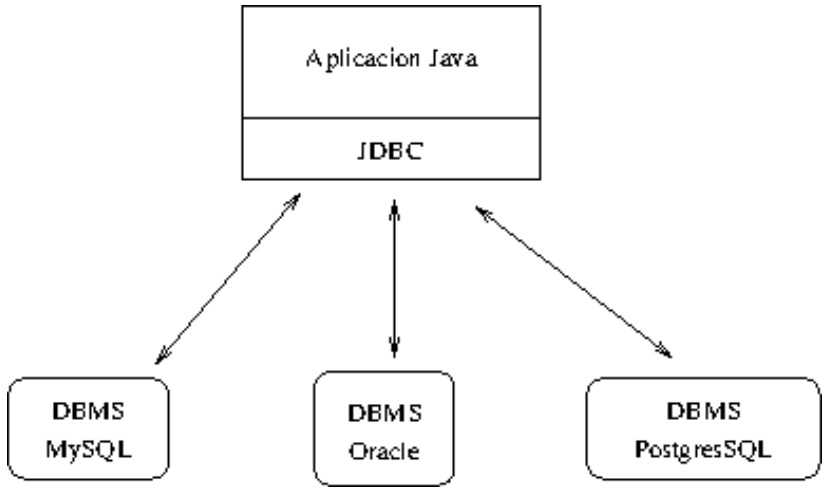
```
public class PrFactura {
    public static boolean esNumeroCorrecto(String numFactura) {
        Pattern p = Pattern.compile("(AUX-(\\d{4}/\\d{2}-)*)?R?\\d{4}/\\d{2}");
        Matcher m = p.matcher(numFactura);
        return m.matches();
    }
    public static final void main(final String[] args) {
        String num="0001/04";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
        num="00001/04";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
        num="00001-04";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
        num="0001/2004";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
        num="0001/04";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
        num="R0001/04";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
        num="AUX-0001/04-0001/04";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
        num="AUX-0001/04-0001/04-0001/04-0001/04-0001/04";
        System.out.println(":"+num+":"+esNumeroCorrecto(num));
    }
}
```

Fechas

```
public class PrFecha {
    private static String convierteFecha(String fecha) {
        String s1 = fecha.replaceAll("(\\d{1,2})/(\\d{1,2})/(\\d{4})",
                                     "$3/$1/$2");
        return s1;
    }
    public static void main(String [] args) {
        String fecha = "6/30/1950";
        System.out.println(convierteFecha(fecha));
    }
}
```

JDBC

- Conexión con una fuentes de datos.
- Realizar peticiones y actualizaciones.
- Manejar los resultados de las consultas



Drivers JDBC

Es necesario un *driver* dependiente del proveedor de la Base de datos.

$$\text{MySQL} \quad \rightsquigarrow \quad \text{MySQL java connector}$$

Creando la base de datos

Fichero crea.sql

```
create database valores;
use valores;

create table valores (
    id int unsigned auto_increment primary key,
    nombre varchar(255),
    ibex_id varchar(100),
    url varchar(255),
    tipo enum("acciones","fondos")
);

create table datos (
       id int unsigned auto_increment primary key,
       empresa int unsigned,
       fecha date,
       valor double
);

grant all on valores.* to luis identified by 'patata';
```

```
~/sql$ mysql -u root < crea.sql
```

Interfaz `java.sql.Connection`

- Establece conexión con la base de datos. No tiene constructor.
- Es un interfaz (está implementado en el *driver*).
- Constuye instrucciones SQL *precompiladas* (método `prepareStatement`).

Realizando conexión con la base de datos

```
    package valores;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Conexion {
    protected static Connection getMysqlConnection(String url, String db,
                                                   String user, String passwd)
        throws ClassNotFoundException,
               InstantiationException,
               SQLException,
               IllegalAccessException {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        return DriverManager.getConnection("jdbc:mysql://"+url+"/"+db+
                                           "?user="+user+"&password="+passwd);
    }
    protected static Connection getMysqlConnection()
        throws ClassNotFoundException,
               InstantiationException,
               SQLException,
               IllegalAccessException {
        return getMysqlConnection("localhost", "valores", "luis", "patata");
    }
    protected static String getFich(String fich) {
        java.net.URL url = ClassLoader.getSystemResource(fich);
        return url.getPath();
    }
}
```

Ficheros de datos

Fichero de valores: valores.csv

```
BSCH!BSCH!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.htm    1
Endesa!ENDESA!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.htm2
Repsol-YPF!REPSOL YPF!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1
Acciona!ACCIONA!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.4tm
Inditex!INDITEX!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.5tmé
Telfonica!TELEFONICA!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.
Banco Popular!BA.POPULAR!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind
Ebro-Puleva!EBRO PULEVA!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/igbm_3_
Mittal-Steel!ARCELOR MIT.!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/igbm_
Indra!INDRA A!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/igbm_6_1.htm10
EADS!EADS!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/igbm_2_1.htm    11
Unipapel!UNIPAPEL!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/igbm_3_1htm
Madrid Bolsa!MADRID BOLSA!fondos!http://www.bolsamadrid.es/esp/mercados/fondos/htmfondos
Plus Madrid!PLUSMADRID!fondos!http://www.bolsamadrid.es/esp/mercados/fondos/htmfondos/00
FonCaixa 65!FONCAIXA 65 BOLSA INDICE ESPA&#241;A!fondos!http://www.bolsamadrid.es/esp/me
```

Cargar datos

```
package valores;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
public class LeeValores {
    private static void insertaDatos(Connection con, BufferedReader in)
        throws IOException, SQLException {
          ....................
    }
    public static void main (String [] args) throws Exception {
        Connection con = null;
        BufferedReader in = null;
        try {
            con = Conexion.getMysqlConnection();
            in = new BufferedReader(new FileReader(Conexion.getFich("sql/valores.csv")))
            insertaDatos(con,in);
        } finally {
            if (con!=null) con.close();
            if (in!=null) in.close();
        }
    }
}
```

¿Dónde está el fichero sql/valores.csv?

```
.
|-- |-- sql
|    |-- borra.sql
|    |-- crea.sql
|    |-- datos.csv
|    |-- usuarios.csv
|    `-- valores.csv
|-- src
|    |-- valores
|    |    |-- CVS
|    |    |    |-- Entries
|    |    |    |-- Repository
|    |    |    `-- Root
|    |    |-- Conexion.java
|    |    |-- ConsigueDatos.java
|    |    |-- LeeDatos.java
|    |    `-- LeeValores.java
|-- classes
|    `-- valores
|         |-- Conexion.class
|         |-- ConsigueDatos.class
|         |-- LeeDatos.class
|         `-- LeeEmpresas.class
```

Accediendo sql/valores.csv

```
~/Java$ export CLASSPATH=.:./classes
~/Java$ $JAVA_HOME/bin/java valores.LeeValores sql/valores.csv
```

Cargar datos

```
private static final String sqlIns =
    "insert into valores (nombre,ibex_id,url,tipo) "+
    "values ( ? , ?  , ? , ?)";
private final static int NOMBRE=1;
private final static int IBEX_ID=2;
private final static int URL_BOLSA=3;
private final static int TIPO=4;

private static void insertaDatos(Connection con, BufferedReader in)
    throws IOException, SQLException {
    String linea = in.readLine();
    PreparedStatement pstmt = con.prepareStatement(sqlIns);
    while (linea!=null) {
        Scanner scan = new Scanner(linea);
        scan.useDelimiter("|");
        String nombre = scan.next();
        String ibexId  = scan.next();
        String tipo = scan.next();
        String url = scan.next();
        pstmt.setString(NOMBRE,nombre);
        pstmt.setString(IBEX_ID,ibex_id);
        pstmt.setString(TIPO,tipo);
        pstmt.setString(URL_BOLSA,url);
        int n = pstmt.executeUpdate();
        linea=in.readLine();
    }
}
```

Ficheros de datos

Fichero de valores: valores.csv

```
Fecha!BSCH!ENDESA!REPSOL-YPF!ACCIONA!INDITEX!óTelefnica!Banco Popular!Ebro-Puleva!Mittal
28/03/02!9.6!17.05!14.45!!!!!!!!!!12.35!17.33!5.49
01/04/02!9.6!17.05!14.45!!!!!!!!!!12.35!17.33!5.49
!!!!!!!!!!!!!!!!
27/08/04!8!15.25!16.98!50.15!19!11.83!!!!!!!11.81!17.03!4.64
30/08/04!7.99!15.29!17.03!50.05!19.08!11.78!!!!!!!11.89!17.1!4.66
12/12/06!14.01!35.76!27.71!137.95!39.59!16.06!13.7!18.62!31.5!18.35!!!21.08!25.22!87
!!!!!!!!!!!!!!!!
19/12/06!13.98!35.15!26.96!138.2!40.78!16.11!13.7!19.17!31.25!18.15!24.20!!21.51!259
20/12/06!14.08!35.16!26.92!139.9!40.95!16.28!13.68!19.28!31.86!18.42!25.45!!21.4!251
21/12/06!14.06!35.43!26.6!140.9!40.61!16.15!13.65!19.25!31.86!18.46!25.79!!21.2!25.
!!!!!!!!!!!!!!!!
27/12/06!14.16!35.72!26.5!140!41.01!16.2!13.69!19.59!31.95!19.06!26.07!21.34!21.16!
28/12/06!14.12!35.55!26.41!141.05!40.89!16.16!13.68!19.12!32.26!18.67!26.00!21.40!21
29/12/06!14.14!35.83!26.2!141.1!40.81!16.12!13.73!19.2!32!18.61!26.00!21.50!21.23!25
01/01/07!14.14!35.83!26.2!141.1!40.81!16.12!13.73!19.2!32!18.61!26.00!21.50!21.18!25
02/01/07!14.49!35.57!26.63!143.05!40.79!16.39!13.97!19.46!32.87!18.88!26.29!21.85!21
```

Cargar datos

```
private static void insertaDatos(Connection con, BufferedReader in)
    throws IOException, SQLException, ParseException {
    int [] id_valores = valores(con);
    String linea = in.readLine();
    linea = in.readLine();
    String sql = "insert into datos (fecha,valor,precio) values ( ?, ? , ? )";
    PreparedStatement pstmt = con.prepareStatement(sql);
    while (linea!=null) {
        Scanner scan = new Scanner(linea);
        scan.useDelimiter("!");
        String strFecha=scan.next();
        if (!strFecha.equals("")) {
            DateFormat df = new SimpleDateFormat("dd/MM/yy");
            Date fecha = df.parse(strFecha);
            System.out.println(df.format(fecha));
            for (int i = 0; i < id_valores.length ; i++) {
                String strValor = scan.next();
                if (!strValor.equals("")) {
                    double precio = Double.parseDouble(strValor);
                    pstmt.setDate(1,new java.sql.Date(fecha.getTime()));
                    pstmt.setInt(2,id_valores[i]);
                    pstmt.setDouble(3,precio);
                    int n = pstmt.executeUpdate();
                }

            }
        }
        linea = in.readLine();
```

Cargar datos

```
private static String [] valores = {
    "BSCH","ENDESA","REPSOL YPF",
    "ACCIONA","INDITEX","TELEFONICA",
    "BA.POPULAR","EBRO PULEVA","ARCELOR MIT.",
    "INDRA A","EADS","UNIPAPEL","MADRID BOLSA",
    "PLUSMADRID","FONCAIXA 65 BOLSA INDICE ESPA&#241;A"
};
private static int [] valores(Connection con) throws SQLException {
    int [] ids = new int[valores.length];
    String sql = "select id from valores where ibex_id like ?";
    PreparedStatement pstmt = con.prepareStatement(sql);
    for (int i = 0; i < ids.length; i++) {
        pstmt.setString(1,valores[i]);
        ResultSet rs = pstmt.executeQuery();
        rs.next();
        ids[i]=rs.getInt(1);
    }
    return ids;
}
```

Realizar actualizaciones

Instrucciones: INSERT, UPDATE, DELETE

```
public static void pr2(Connection con) throws SQLException {
    String sql = "update datos set precio=precio+1";
    PreparedStatement pstmt = con.prepareStatement(sql);
    int n = pstmt.executeUpdate();
    System.out.println(n+" filas actualizadas");
    SQLWarning warning = pstmt.getWarnings();
    if (warning!=null) {
        System.out.println("AVISOS...");
        while (warning!=null) {
            System.out.println("Message: " + warning.getMessage());
            warning = warning.getNextWarning();
        }
    } else {
        System.out.println("No hay avisos");
    }
}
```

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

```
ResultSet rs = pstmt.executeQuery();
```

rs	id	nombre	tipo

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

```
rs.next();
```

	id	nombre	tipo
rs	1	BSCH	acciones

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

```
rs.next();
```

	id	nombre	tipo
	1	BSCH	acciones
rs	2	Endesa	acciones

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

```
rs.next();
```

	id	nombre	tipo
	1	BSCH	acciones
	2	Endesa	acciones
rs	3	Repsol-YPF	acciones

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

```
rs.next();
```

	id	nombre	tipo
	1	BSCH	acciones
	2	Endesa	acciones
	3	Repsol-YPF	acciones
rs	4	Acciona	acciones

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

```
rs.next();
```

id	nombre	tipo
1	BSCH	acciones
2	Endesa	acciones
3	Repsol-YPF	acciones
4	Acciona	acciones
rs 5	Inditex	acciones

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

```
rs.next();
```

	id	nombre	tipo
	1	BSCH	acciones
	2	Endesa	acciones
	3	Repsol-YPF	acciones
	4	Acciona	acciones
	5	Inditex	acciones
rs	6	Telefónica	acciones

Realizar Consultas

Instrucciones Select

```
select id,nombre,tipo from valores
```

1

id	nombre	tipo
1	BSCH	acciones
2	Endesa	acciones
3	Repsol-YPF	acciones
4	Acciona	acciones
5	Inditex	acciones
6	Telefónica	acciones

Ejemplos de consultas I

```
1  public static void pr1(Connection con) throws SQLException {
2      String sql = "select * from valores";
3      PreparedStatement pstmt = con.prepareStatement(sql);
4      ResultSet rs = pstmt.executeQuery();
5      while (rs.next()) {
6          int id = rs.getInt("id");
7          String nombre = rs.getString("nombre");
8          System.out.println(id+":"+nombre);
9      }
10 }
```

Ejemplos de consultas II

```
public static void pr3(Connection con) throws SQLException {
    String sql = "select valores.nombre, datos.fecha, datos.precio "+
        " from datos left join valores on valores.id=datos.valor "+
        " where datos.fecha>='2007-03-01'"+
        " order by valores.tipo, valores.nombre, datos.fecha";
    PreparedStatement pstmt = con.prepareStatement(sql);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        String nombre = rs.getString("valores.nombre");
        Date fecha = rs.getDate("datos.fecha");
        double precio = rs.getDouble("datos.precio");
        DateFormat df = new SimpleDateFormat("dd 'de' MMMM 'de' yyy");
        NumberFormat nf = new DecimalFormat("#,000.00'euro'");
        System.out.println(df.format(fecha)+":"+nombre+":"+nf.format(precio));
    }
}
```

Ejemplos de consultas III

```
public static int getID(String ibex_id, Connection con) throws SQLException {
    String sql = "select id from valores where ibex_id like '"+ibex_id+"'";
    PreparedStatement pstmt = con.prepareStatement(sql);
    ResultSet rs = pstmt.executeQuery();
    if (rs.next()) {
        return rs.getInt(1);
    } else {
        return -1;
    }
}
```

Consiguiendo datos desde www.bolsamadrid.es

- Conseguir el fichero HTML.
- Analizar el fichero.
- Añadir datos en la base de datos.

Conseguir el fichero HTML

Fichero de valores: valores.csv

```
BSCH!BSCH!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.htm
Endesa!ENDESA!acciones!http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.htm
```

```
    private static String cargaPagina(java.net.URL url)
        throws IOException{
        InputStream in = url.openStream();
        StringBuffer sb = new StringBuffer();
        int c = in.read();
        while (c!=-1) {
            sb.append((char)c);
            c = in.read();
        }
        String res = sb.toString();
        return res;
    }
```

Analizar el fichero HTML

Fichero de valores:

http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.h

```
.....................
....<IMG SRC="/images/arr-up9.gif" BORDER=0> BSCH</A></TD><TD>13,50</TD>.....
.....................
....<IMG SRC="/images/arr-dw9.gif" BORDER=0> ENDESA</A></TD><TD>40,29</TD>...
```

```
    private static double getValor(String ibex_id, String datos, String tipo)
        throws ParseException {
        String regExp=null;
        if (tipo.equals("acciones")) {
            regExp=ibex_id+" *</A></TD><TD>([0-9,]+)</TD>";
        } else if (tipo.equals("fondos")) {
            regExp=ibex_id+" *</a></TD>.*?<TD *[^>]*>([0-9,]+)</TD>";
        } else {
            throw new RuntimeException("Tipo \'"+tipo+"\' desconocido");
        }
        Pattern p = Pattern.compile(regExp);
        Matcher m = p.matcher(datos);
        if (m.find()) {
            // Double.parseDouble no vale, no analiza 13,50
            NumberFormat nf = NumberFormat.getInstance();
            return nf.parse(m.group(1)).doubleValue();
        } else {
            throw new RuntimeException("Acciones \'"+ibex_id+"\' no encontrada");
```

Insertar los datos en la Base de datos

```
private static final String sqlIns =
       "insert into datos (fecha,valor,precio) values( ? , ? , ?)";
private static final int FECHA=1;
private static final int VALOR=2;
private static final int PRECIO=3;

private static void insertaDatos(java.sql.Date fecha,Connection con)
    throws SQLException, IOException, ParseException {
    HashMap<String,String> paginas = new HashMap<String,String>();
    String sql = "select * from valores";
    PreparedStatement pstmtValores = con.prepareStatement(sql);
    ResultSet rs = pstmtValores.executeQuery();
    PreparedStatement pstmtIns = con.prepareStatement(sqlIns);
    while (rs.next()) {
        insertaDatos(pstmtIns,
                     fecha,
                     rs,paginas);
    }
}
```

Insertar los datos en la Base de datos

```
private static void insertaDatos(PreparedStatement pstmtIns,
                                 java.sql.Date fecha,
                                 ResultSet rs,
                                 HashMap<String,String> paginas)
    throws SQLException,IOException, ParseException {
    String url = rs.getString("url");
    String datos = paginas.get(url);
    if (datos==null) {
        datos = cargaPagina(paginas,url));
        paginas.put(url,datos);
    }
    String tipo = rs.getString("tipo");
    String ibex_id = rs.getString("ibex_id");
    double valor = getValor(ibex_id,datos,tipo);
    int id = rs.getInt("id");
    String nombre = rs.getString("nombre");
    pstmtIns.setDate(FECHA,fecha);
    pstmtIns.setInt(VALOR,id);
    pstmtIns.setDouble(PRECIO,valor);
    int n = pstmtIns.executeUpdate();
}
```

Consiguiendo datos automáticamente

```
#!/bin/sh
# $Id: transparencias.tex,v 1.8 2007-05-10 11:37:22 luis Exp $

JAVA_HOME=/usr/lib/jvm/default-java
CLASSPATH=/home/casa/ahorros2/classes
$JAVA_HOME/bin/java -cp $CLASSPATH valores.ConsigueDatos $*
```

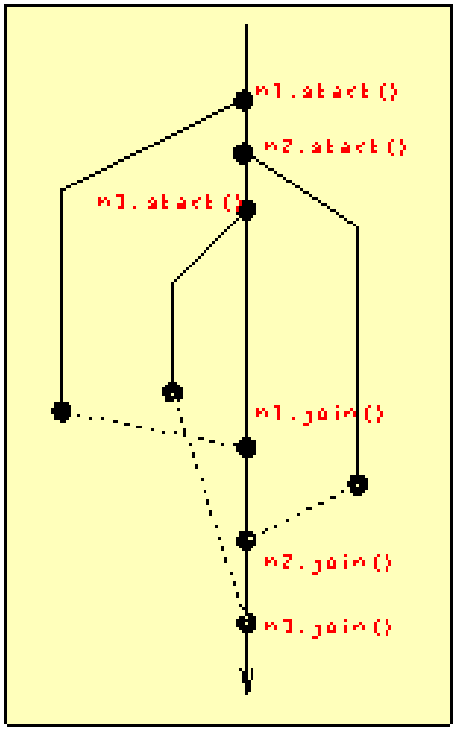
```
$ crontab -l

SHELL=/bin/bash
MAILTO=luis
# Directorios donde buscar programas
PATH=/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/home/luis/bin

45 23 * * 1,2,3,4,5     /home/casa/ahorros2/scripts/consigueDatos.sh > /dev/null 2>&1
```

Ejemplos de *programación concurrente*

- En un sistema operativo, diversos programas compiten por los recursos del sistema: memoria, dispositivos.
- Bases de datos.
- Aplicaciones Web.

Hebras, hilos

En un programa concurrente puede haber *varios hilos de computación*.



Sincronización de Objetos

- Puede haber varias hebras ejecutando *simultáneamente* métodos de objetos
- Es necesario *sincronizar* los accesos al objeto.

Threads

Extendiendo la clase java.lang.Thread.

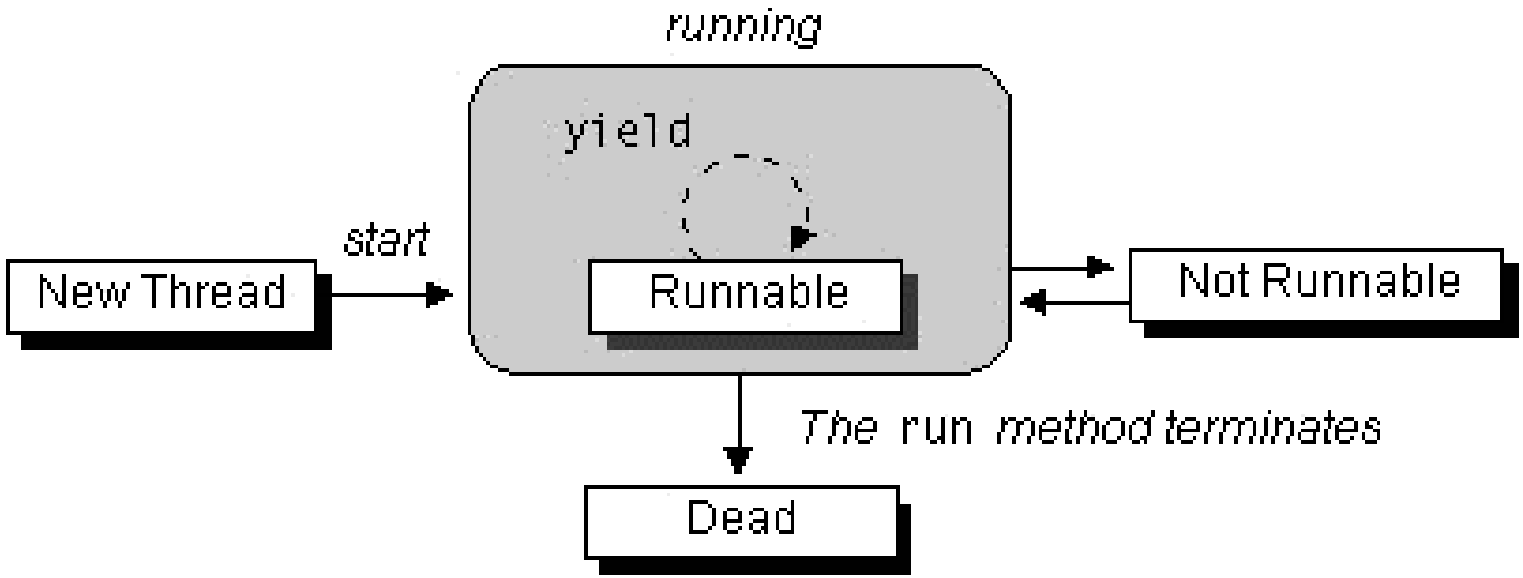
```
public class PrThread extends Thread{
    public PrThread(String s) {
        super(s);
    }
    public final void run() {
        boolean sigue=true;
        for (int i=0; i<100 && sigue; i++) {
            try {
                System.out.println(getName()+":"+i);
                sleep(20);
            } catch (InterruptedException e) {
                System.out.println(getName()+" interrumpida");
                sigue=false;
            }
        }
    }
    public static final void main(final String[] args){
        Thread p = new PrThread("mia");
        p.start();
    }
}
```

Threads

Implementado el interfaz `java.lang.Runnable`.

```
public class PrRunnable implements Runnable {                              1
    public final void run() {                                              2
        Thread hebra = Thread.currentThread();                             3
        boolean sigue=true;                                                4
        for (int i=0; i<100 && sigue; i++) {                               5
            try {                                                          6
                System.out.println(hebra.getName()+":"+i);                 7
                hebra.sleep(20);                                           8
            } catch (InterruptedException e) {                             9
                System.out.println(hebra.getName()+" interrumpida");       10
                sigue=false;                                               11
            }                                                              12
        }                                                                  13
    }                                                                      14
    public static final void main(final String[] args) {                  15
        Thread p = new Thread(new PrRunnable(),"mia");                     16
        p.start();                                                         17
    }                                                                      18
}                                                                          19
```

Ciclo de vida de una hebra



Tras crear una hebra y se ejecuta el método `start`, la hebra puede estár:

1. En ejecución.
2. Suspendida: ha ejecutado `sleep`, `join`, `wait`.

Parar una hebra

Usar el método `interrupt`

```
public static void ex1() throws InterruptedException{
    Thread h = new PrThread("1");
    h.start();
    Thread.sleep(100);
    h.interrupt();
    System.out.println(h.isInterrupted());
}
```

Métodos *Deprecated*: `stop`, `suspend`, `resume`.

Una hebra para cunado está *Not Runnnable*, ha ejecutado `sleep`, `join`, `wait`.

Pueden lanzar `InterruptedException`, la hebra debería parar.

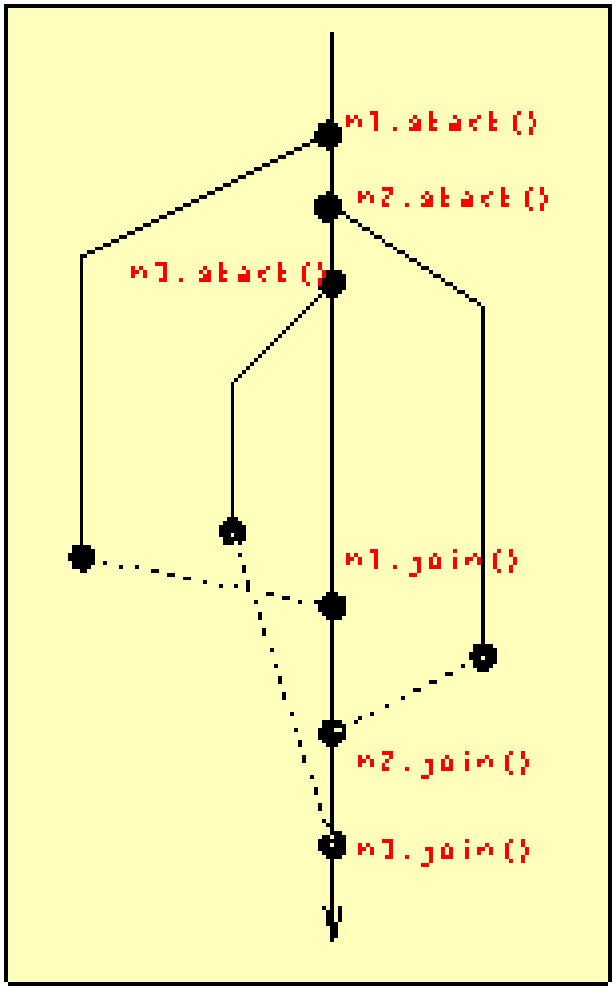
Esperamos a que una hebra acabe

Método: join

```
public static void ex2() throws InterruptedException {
    Thread h1 = new PrThread("1");
    Thread h2 = new PrThread("2");
    Thread h3 = new PrThread("3");
    h1.start();
    h2.start();
    h3.start();
    h1.join();
    h2.join();
    h3.join();
}
```

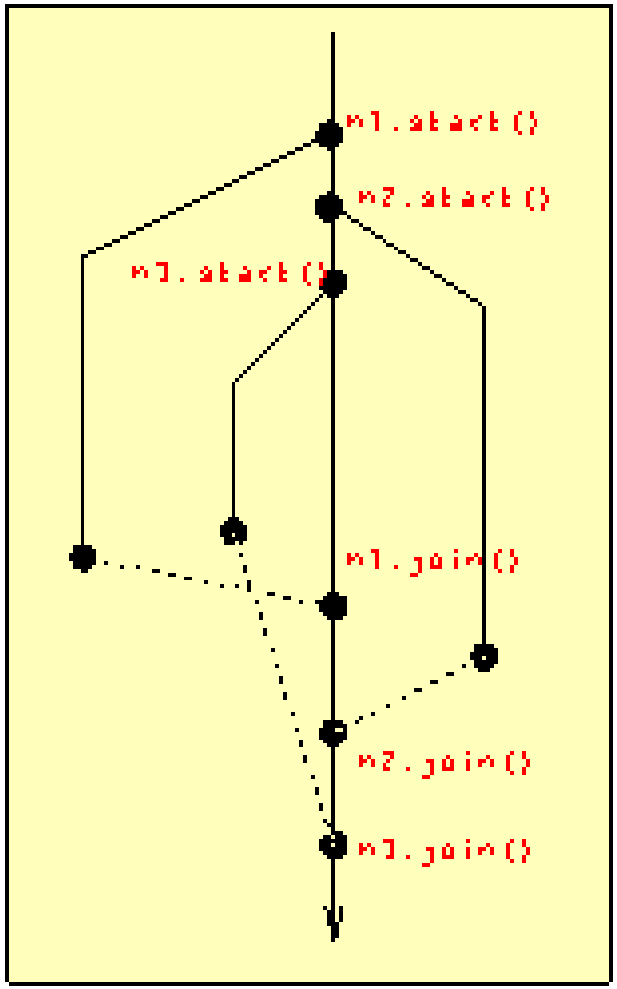
Esperamos a que una hebra acabe

Método: `join`



Esperamos a que una hebra acabe

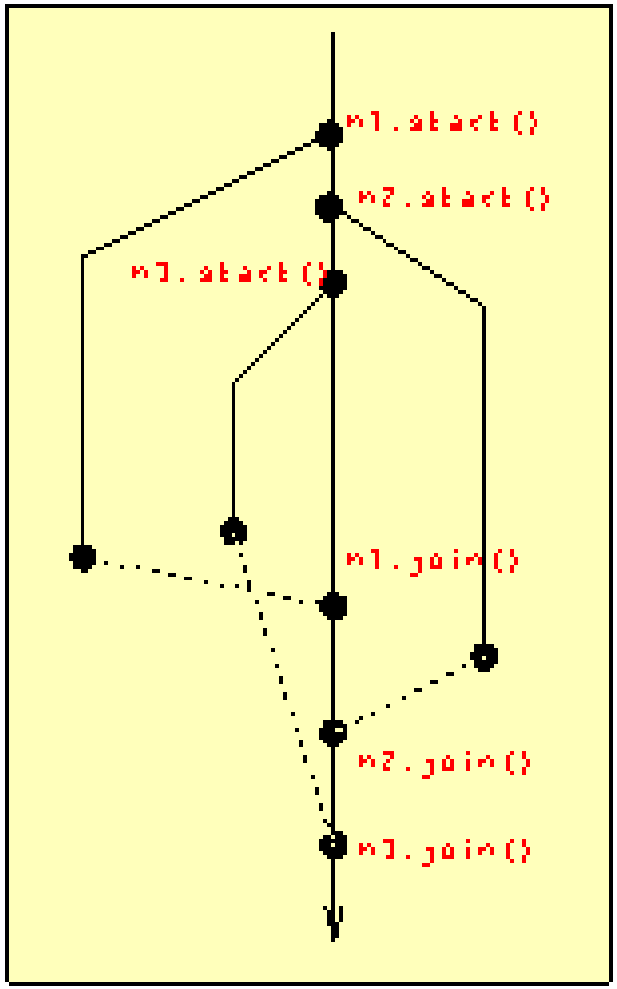
Método: `join`



1. `h1.start()`: `h1` se ejecuta.

Esperamos a que una hebra acabe

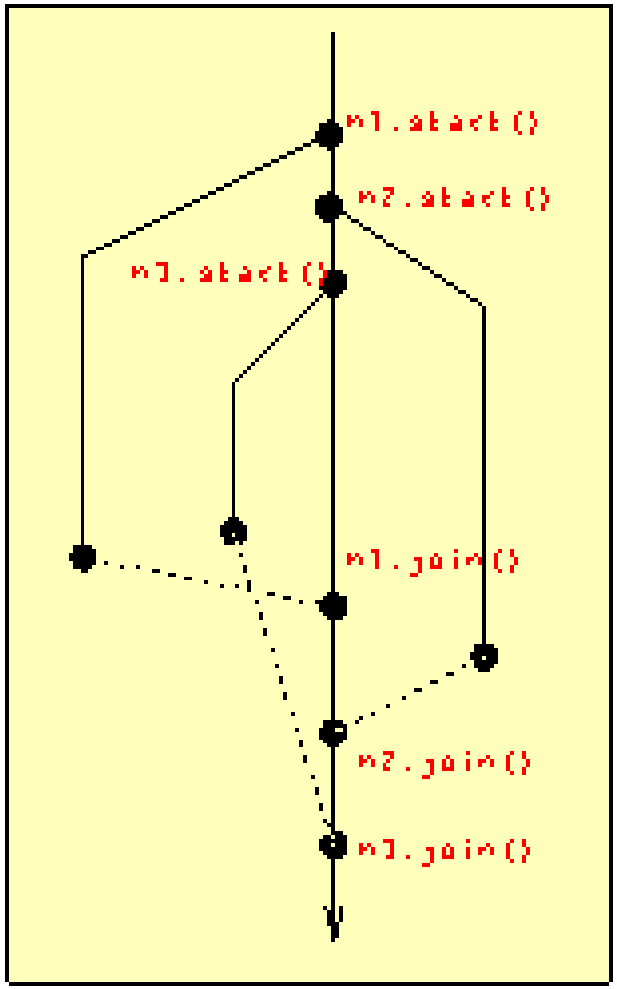
Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.

Esperamos a que una hebra acabe

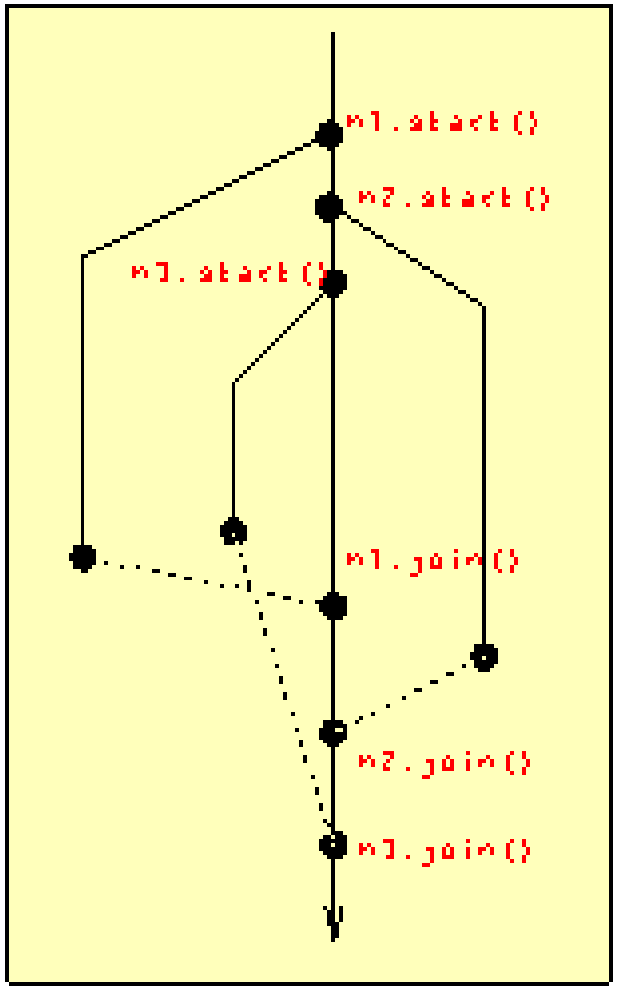
Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.
3. `h3.start()`: `h3` se ejecuta.

Esperamos a que una hebra acabe

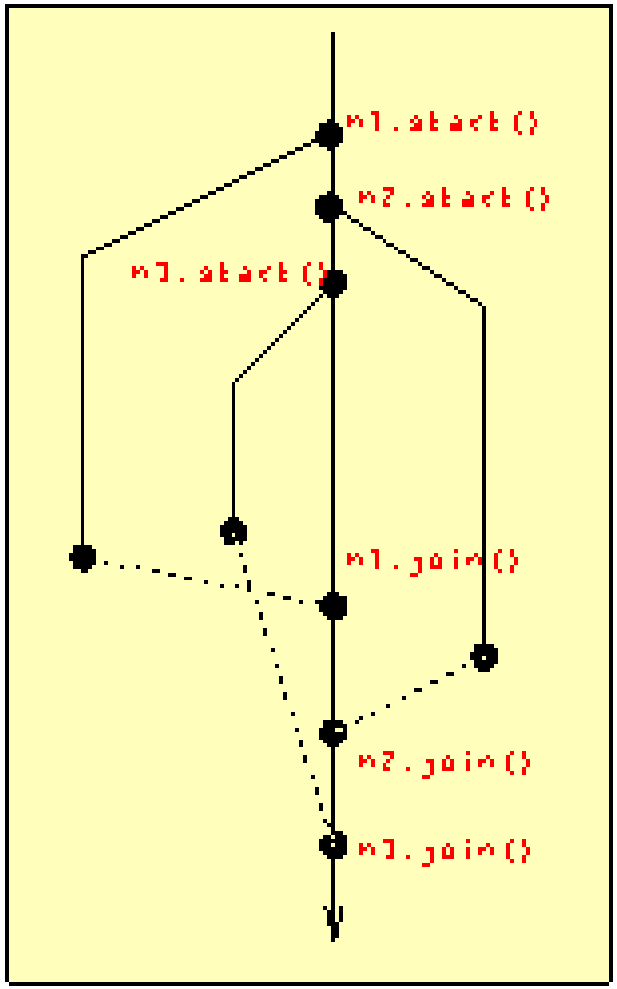
Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.
3. `h3.start()`: `h3` se ejecuta.
4. `h1.join()`: esperamos a que `h1` pare.

Esperamos a que una hebra acabe

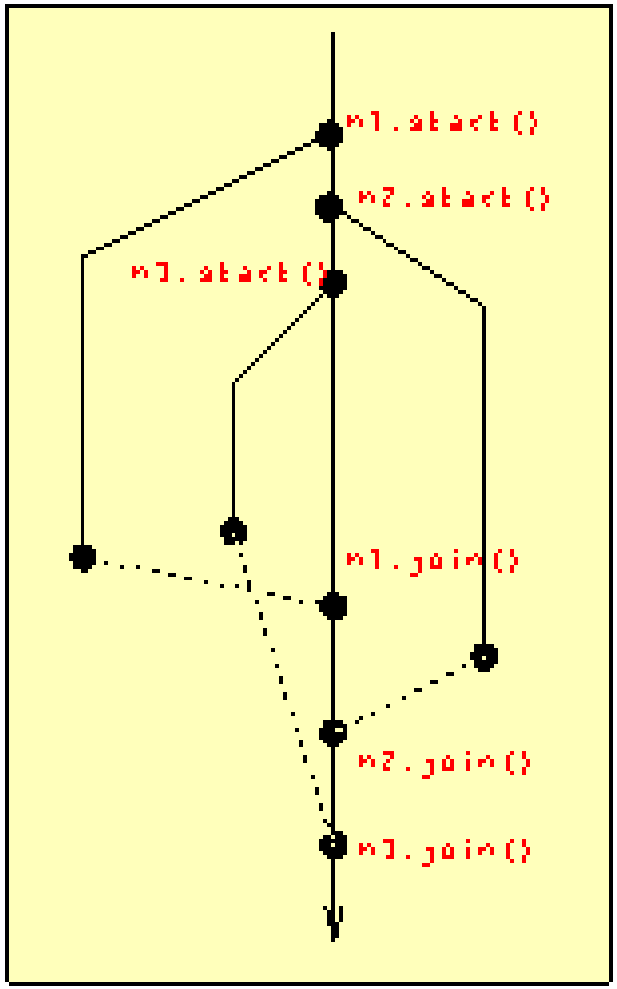
Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.
3. `h3.start()`: `h3` se ejecuta.
4. `h1.join()`: esperamos a que `h1` pare.
5. `h2.join()`: esperamos a que `h2` pare.

Esperamos a que una hebra acabe

Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.
3. `h3.start()`: `h3` se ejecuta.
4. `h1.join()`: esperamos a que `h1` pare.
5. `h2.join()`: esperamos a que `h2` pare.
6. `h3.join()`: esperamos a que `h3` pare (no hace falta).

Cerrojos de objetos

- Cada objeto en Java tiene un cerrojo

```
synchronized (obj) {
   /*  */
    }
```

  - Sólo pueda haber una hebra *propietaria* del cerrojo.
  - Sólo una hebra propietaria del cerrojo puede ejecutar un código `synchronized`.

- El cerrojo puede abarcar a todo un método

```
type method (...) {
  synchronized(this) {
     /*   */
   }
```

```
synchronized type method (...) {
     /*  */
}
```

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

- La hebra queda *suspendida* hasta que alguien la *despierte*.
- Se libera para que otra hebra pueda adquirirlo.
- Cuando es liberarla debe adquirir de nuevo el cerrojo para seguir la ejecución.

`wait(tiempo)` Igual, pero se queda dormida un `tiempo` máximo.

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

- La hebra queda *suspendida* hasta que alguien la *despierte*.
- Se libera para que otra hebra pueda adquirirlo.
- Cuando es liberarla debe adquirir de nuevo el cerrojo para seguir la ejecución.

`wait(tiempo)` Igual, pero se queda dormida un `tiempo` máximo.

`notify()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `notify()` del objeto.

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

- La hebra queda *suspendida* hasta que alguien la *despierte*.
- Se libera para que otra hebra pueda adquirirlo.
- Cuando es liberarla debe adquirir de nuevo el cerrojo para seguir la ejecución.

`wait(tiempo)` Igual, pero se queda dormida un `tiempo` máximo.

`notify()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `notify()` del objeto.

- Despierta *una* hebra suspendida en el objeto.
- *No* libera el cerrojo del objeto.

`notifyAll` Igual, pero despierta a todas.

Regiones críticas

El acceso a las regiones críticas debe ser *exclusivo*

```
public void run() {
    boolean para = false;
    while (!para) {
        try {
            ciclo();
        } catch (InterruptedException e) {
            para = true;
        }
    }
}
private void ciclo() throws InterruptedException {
    monitor.entrar();
    InterruptedException salir=null;
    try {
        regCritica();
    } catch (InterruptedException e) {
        salir=e;
    } finally {
        monitor.salir();
        if (salir!=null) {
            throw salir;
        }
        int t = random.nextInt(tiempoDentro);
        sleep(t);
    }
}
```

Monitor regiones críticas

```
public class Monitor {
    private int caben; //caben>=0
    public Monitor() {
        caben=1;
    }
    public synchronized void salir() {
        caben++;
        notify();
    }

    public synchronized void entrar() throws InterruptedException{
        while (caben==0) wait();
        caben--;
    }
}
```

Colecciones sincronizadas

En la clase `java.util.Collections` encontramos los siguientes métodos estáticos:

```
public static <T> Collection<T> synchronizedCollection(Collection<T> c)
```
1

```
public static <T> Set<T> synchronizedSet(Set<T> s)
```
1

```
public static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s)
```
1

```
public static <T> List<T> synchronizedList(List<T> list)
```
1

```
public static <K,V> Map<K,V> synchronizedMap(Map<K,V> m)
```
1

```
public static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m)
```
1

Lectores/Escritores I

```
public void run()  {
  try {
    monitor.permisoLeer();
    lee();
  } finally {
    monitor.finLeer();
  }
}
```

```
public void run()  {
  try {
    monitor.permisoEscribir();
    escribe();
  } finally {
    monitor.finEscribir();
  }
}
```

Lectores/Escritores II

```
package simulacion.lectoresEscritores;
public class Monitor {
    private int escrEsperando;  //únm escritores esperando
    private int numLect; // únm lectores leyendo
    private int numEscr; // únm escritores escribiendo
// escrEsperando>=numEscr, 0<=numEscr<=1 numLect>=0, nunLect>0 ---> numEscr=0
    public Monitor() {
        escrEsperando=0; numLect=0; numEscr=0;
    }
    public synchronized void permisoLeer() throws InterruptedException {
        while (numEscr>0 || escrEsperando>0) wait();
        numLect++;
    }
    public synchronized void finLeer() {
        numLect--;
        notifyAll();
    }
    public synchronized void permisoEscribir() throws InterruptedException {
        escrEsperando++;
        while (numLect>0) wait();
        numEscr++;
    }
    public synchronized void finEscribir() {
        escrEsperando--;
        numEscr--;
    }
}
```

Lectores/Escritores III

```
package simulacion.lectoresEscritores;
import soporte.Aleatorio;
public class GeneradorLectores extends Thread {
    private double tMedioLlegada;
    private int tLeyendo;
    private int tMaxSimulacion;
    private Aleatorio aleatorio;
    private Monitor monitor;
    public GeneradorLectores (double tllegada, int tleyendo,  int tmax, int semilla, Mon
        tMedioLlegada = tllegada; tLeyendo = tleyendo;  tMaxSimulacion = tmax;
        aleatorio = new Aleatorio(semilla);
        monitor = m;
    }
    public void run() {
        System.out.println("Empezando la ógeneracin de lectores");
        try {
            while (true) {
                int sigCoche = aleatorio.poisson(tMedioLlegada);
                this.sleep(sigCoche*1000);
                Lector lector = new Lector(Simulador.sigUsuario(), tLeyendo, monitor);
                System.out.println("Comenzando "+lector);
                lector.start();
            }
        } catch ( InterruptedException e ) {}
        System.out.println("óSimulacin lectores finalizada");
    }
}
```

Lectores/Escritores IV

```
package simulacion.lectoresEscritores;
class Simulador {
    private static int  numUsuario = -1;
    private static long horaInicio;
    public synchronized static int sigUsuario() {
        numUsuario++;
        return numUsuario;
    }
    public Simulador(double tLlegadaLect, double tLlegadaEscr,
                     int tLeyendo, int tEscribiendo, int tMax)
        throws InterruptedException {
        Monitor monitor = new Monitor();
        GeneradorLectores generaLectores =
            new GeneradorLectores(tLlegadaLect, tLeyendo, tMax, 1111, monitor);
        GeneradorEscritores generaEscritores =
            new GeneradorEscritores(tLlegadaEscr, tEscribiendo, tMax, 3333, monitor);
        generaLectores.start();
        generaEscritores.start();
        horaInicio = System.currentTimeMillis();
        Thread.sleep(tMax*1000);
        generaLectores.interrupt();
        generaEscritores.interrupt();
        generaLectores.join();
        generaEscritores.join();
    }
    public static void main (String[] args) throws Exception {
        Simulador s = new Simulador(2, 8, 1, 2, 30);
    }
```

Filósofos I

Filósofos II

```
while (true) {
  piensa();
  mesa.pideTenedores();
  come();
  mesa.cedeTenedores();
}
```

$$comiendo[i] \begin{cases} true & \text{filósofo } i \text{ está comiendo} \\ false & \text{filósofo } i \text{ NO está comiendo} \end{cases}$$

$$INV \equiv comiendo[i] \rightarrow \neg comiendo[i \oplus 1] \wedge \neg comiendo[i \ominus 1]$$

Filósofos III

```
public class Mesa {
    private int numFilosofos;
    private boolean [] comiendo;
    public Mesa(int n) {
        numFilosofos = n;;
        comiendo = new boolean[numFilosofos];
        for (int i = 0; i < numFilosofos; i++) { comiendo[i] = false; }
    }
    /*  permisoComer(int i) y cedePermiso(int i) */
}
```

Filósofos IV

```
public synchronized void permisoComer(int i) throws InterruptedException{
  while (comiendo[ant(i)] || comiendo[sig(i)]) { wait(); }
  comiendo[i]=true;
}

public synchronized void cedePermiso(int i) {
  comiendo[i]=false;
  notifyAll();
}

public int sig(int i) { return (i+1) % numFilosofos; }

public int ant(int i) { return  ( i-1+numFilosofos ) % numFilosofos; }
```

Filósofos V

```
public class Filosofo extends Thread {
    private Random random;
    private int tiempoComiendo, tiempoPensando;
    private Mesa mesa;
    private int id;
    public Filosofo (Random r,
                     int tc, int tp,
                     Mesa m,
                     int i) {
        random = r;
        tiempoPensando = tp;
        tiempoComiendo = tc;
        mesa = m;
        id = i;
    }
    public void run() {
        boolean para = false;
        while (!para) {
            try {
                ciclo();
            } catch (InterruptedException e) {
                para = true;
            }
        }
    }
```

Filósofos VI

```
    private void ciclo() throws InterruptedException {
        piensa(id,tiempoPensando);
        mesa.permisoComer(id);
        try {
            come(id,tiempoComiendo);
        } catch (InterruptedException e) {
            mesa.cedePermiso(id);
            throw e;
        }
        mesa.cedePermiso(id);
    }
    private void espera(int tiempo) throws InterruptedException {
        int t = random.nextInt(tiempo);
        sleep(t);
    }
    private  void piensa(int id, int tiempo) throws InterruptedException {
        System.out.println("El ófilsofo "+id+" empieza a pensar");
        espera(tiempo);
        System.out.println("El ófilsofo "+id+" acaba de pensar");
    }
    private void come(int id, int tiempo)  throws InterruptedException {
        System.out.println("El ófilsofo "+id+" empieza a comer"+":"+mesa);
        espera(tiempo);
        System.out.println("El ófilsofo "+id+" acaba de comer"+":"+mesa);
    }
}
```

Filósofos VII

```
public static void main (String[] args) throws InterruptedException {
    int  numFilosofos = 5;
    int tiempoPensando = 1000;
    int tiempoComiendo  = 2000;
    int tiempoParada = 10000;
    Random r = new Random();
    Mesa mesa = new Mesa(numFilosofos);
    Filosofo [] filosofo = new Filosofo[numFilosofos];
    for (int i = 0; i < numFilosofos ; i++) {
        filosofo[i] = new Filosofo(r,tiempoComiendo, tiempoPensando,
                                   mesa, i);
        filosofo[i].start();
    }

    for (int i = 0; i < numFilosofos ; i++) {
        Thread.sleep(tiempoParada);
        System.out.println("Parando ófilsofo "+i+".....");
        filosofo[i].interrupt();
    }
    for (int i = 0; i < numFilosofos ; i++) {
        filosofo[i].join();
    }
}
```