

Objetos

Luis Fernando Llana Díaz

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

15 de abril de 2007

- *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();
        }
    }
}
```

1
2
3
4
5
6
7
8
9
10
11

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 días) {
        .....
    }
    public void siguiente() {
        anyadir(1);
    }
    public void anterior() {
        anyadir(1);
    }
    public int diasHasta(Fecha otra) {
        .....
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

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 días
    public final static int LUNES=0;
    public final static int MARTES=1;
    .....
    public final static int DOMINGO=6;
    .....
    .....
    .....

    //Días 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);
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Clase Fecha

```
public void anyadirDias(int inc) {
    int d=diasDesdeInicio+inc;
    if (d<0) throw new FechaFueraDeRango("Día anterior 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;
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

Clase Fecha

```
private static int calculaDiasDesdeInicio(int elDia, int elMes, int elAnyo){
    int dias;

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

1
2
3
4
5
6
7
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 año
    dias = dias + diff / 4; //añadimos 1 por cada múltiplo de 4
    dias = dias - (diff/100); //se quida 1 por cada múltimo de 100
    dias = dias + (diff/400); //se añade 1 por cada múltimo de 400

    return dias;
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

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;
    }
    .....
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

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;
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

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);
```

1
2

```
package fecha;  
  
public class Fecha implements Comparable{  
.....  
.....  
}  
  
class FechaTerna {  
.....  
.....  
}
```

1
2
3
4
5
6
7
8
9
10
11

Terminología

constructor

Procedimiento que construye objetos

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

1
2
3
4

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

1
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;  
private int mes;  
private int anyo;
```

1

2

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("Día anterior al permitido");  
    diasDesdeInicio=d;  
}  
public void siguiente() {  
    anyadirDias(1);  
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

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.

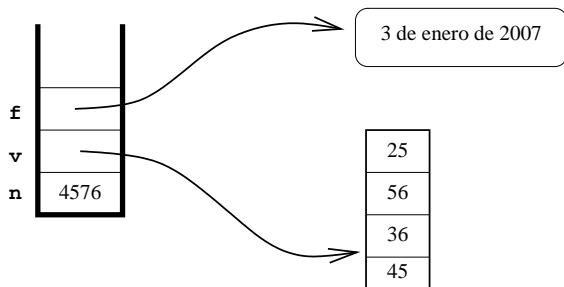
private sólo desde la misma clase.

Referencias

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

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

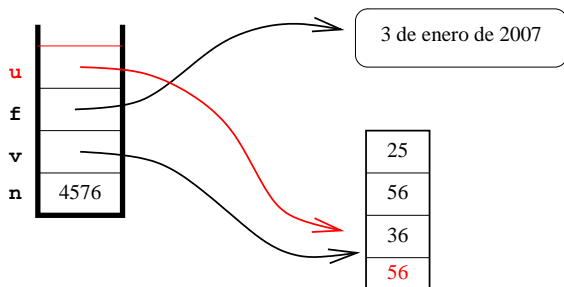
1
2
3



Referencias, efecto lateral

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

1
2

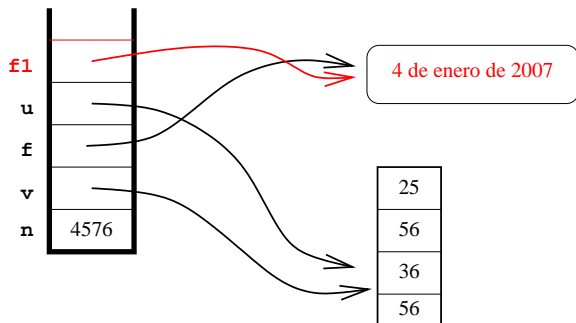


Referencias, efecto lateral

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

1

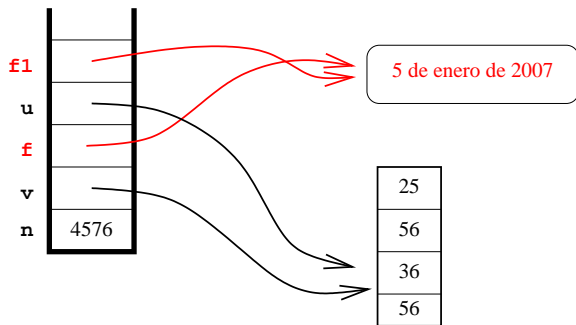
2



Referencias, objetos como parámetros

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

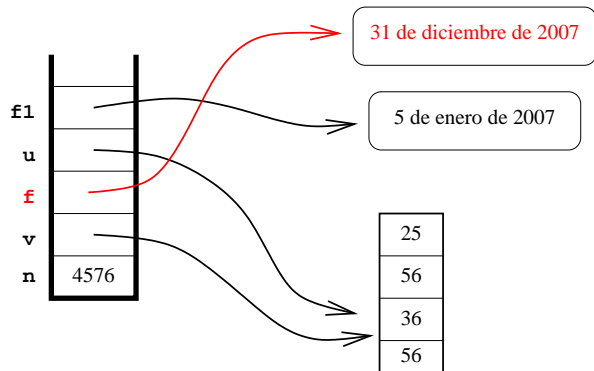
1
2
3
4
5



Referencias, cambiar de referencia

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

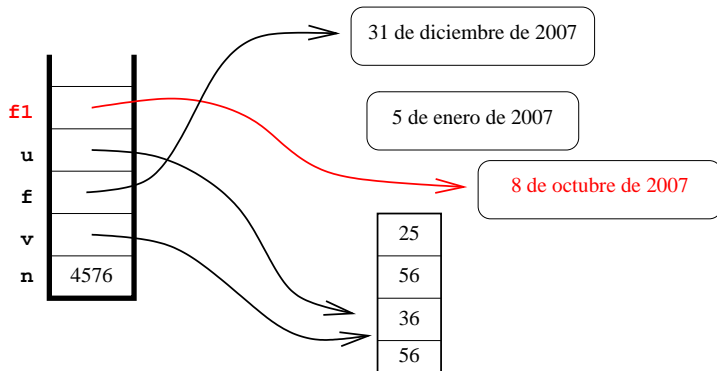
1



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 ser 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 añaden las partes que no se pueden cambiar

Variables constantes.

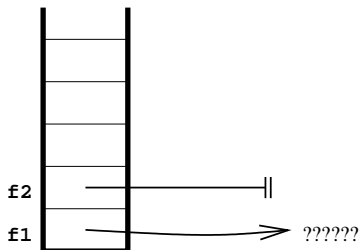
Métodos que no se pueden reescribir en la *herencia*.

Referencia null

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

1

2

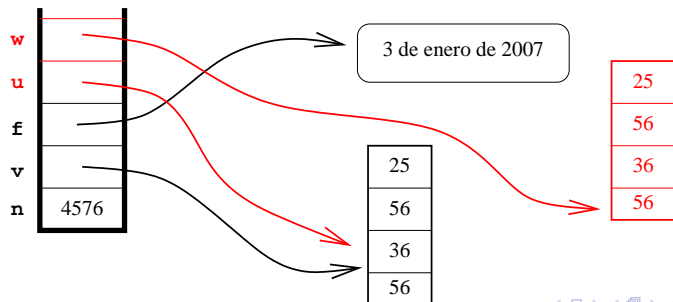


Igualdad de referencias

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

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

1
2
3
4
5
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 objetos iguales.
```

1
2
3
4
5
6
7
8
9
10
11
12
13

Igualdad de referencias

```
boolean b;  
b = f==f1; // b es true;  
b = f==f2; // b es false;  
b = f1==f2; // b es false;
```

1
2
3
4

```
boolean b;  
b = f.equals(f1); // b es true;  
b = f.equals(f2); // b es true;  
b = f1.equals(f2); // b es true;  
b = f1.equals(f); // b es true;  
b = f2.equals(f1); // b es true;  
b = f2.equals(f); // b es true;
```

1
2
3
4
5
6
7

Igualdad de referencias

```
f.siguiete();  
b = f==f1; // b es true;  
b = f==f2; // b es false;  
b = f1==f2; // b es false;
```

1
2
3
4

```
boolean b;  
b = f.equals(f1); // b es true;  
b = f.equals(f2); // b es false;  
b = f1.equals(f2); // b es false;  
b = f1.equals(f); // b es true;  
b = f2.equals(f1); // b es false;  
b = f2.equals(f); // b es false;
```

1
2
3
4
5
6
7