

Tecnología de la Programación

Igualdad, copias, tipos de datos

(Tema 3 de los apuntes)

Simon Pickin,
Alberto Díaz, Puri Arenas, Yolanda García

Igualdad entre objetos

- Se puede comprobar la igualdad entre dos objetos con :
 - el operador “==”
 - compara las referencias, es decir, las direcciones de memoria,
 - Devuelve cierto si los dos elementos contienen la misma dirección de memoria
 - es decir, apuntan al mismo objeto
 - el método `equals` que hereda toda clase de la clase `Object`
 - por defecto, es equivalente a “==”
 - pero se puede, y muchas veces se debe, sobrescribir (es decir, redefinir)

Ejemplo1a: igualdad entre objetos

```
public class A {  
    private int x; private int y;  
    public A(int x, int y){ this.x = x; this.y = y; }  
}  
  
public class Main  
    public static void main(String[] args) {  
        A a1 = new A(2,2);  
        A a2 = new A(2,2);  
        if (a1==a2) System.out.println("Cierto");  
        else System.out.println("Falso");  
    }  
}
```

- ¿Qué imprime este programa en la salida estándar?

Ejemplo1a: igualdad entre objetos

```
public class A {  
    private int x; private int y;  
    public A(int x, int y){ this.x = x; this.y = y; }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        A a1 = new A(2,2);  
        A a2 = a1;  
        if (a1==a2) System.out.println("Cierto");  
        else System.out.println("Falso");  
    }  
}
```

- ¿Qué imprime este programa en la salida estándar?

Ejemplo1a: igualdad entre objetos

```
public class A {
    private int x; private int y;
    public A(int x, int y){ this.x = x; this.y = y; }
}

public class Main {
    public static void main(String[] args) {
        A a1 = new A(2,2);
        A a2 = new A(2,2);
        if (a1.equals(a2)) System.out.println("Cierto");
        else System.out.println("Falso");
    }
}
```

•¿Qué imprime este programa en la salida estándar?

Ejemplo1b: igualdad entre objetos

```
public class A {
    private int x; private int y;
    public A(int x, int y){ this.x = x; this.y = y; }
    public boolean equals(A a){
        return (this.x == a.x && this.y == a.y);
    }
}

public class Main {
    public static void main(String[] args) {
        A a1 = new A(2,2);
        A a2 = new A(2,2);
        if (a1.equals(a2)) System.out.println("Cierto");
        else System.out.println("Falso");
    }
}
```

•¿Qué imprime este programa en la salida estándar?

Ejemplo1b: igualdad entre objetos

```
public class A {
    private int x; private int y;
    public A(int x, int y){ this.x = x; this.y = y; }
    public boolean equals(A x){
        return (this.x == a.x && this.y == a.y);
    }
}

public class Main {
    public static void main(String[] args) {
        A a1 = new A(2,2);
        A a2 = a1;
        if (a1.equals(a2)) System.out.println("Cierto");
        else System.out.println("Falso");
    }
}
```

•¿Qué imprime este programa en la salida estándar?

• §3 - 7

Ejemplo 2a: igualdad entre objetos

```
public class B
    private int z;

    public B(int z){
        this.z=z;
    }
}

public class A {
    private int x;
    private B b;

    public A(int x, B b){
        this.x = x;
        this.b = b;
    }

    public boolean equals(A a){
        return (this.x==a.x
                && this.b == a.b);
    }
}
```

• §3 - 8

Ejemplo 2a: igualdad entre objetos

```
public class Main {  
  
    public static void main(String[] args) {  
        B b1 = new B(2);  
        B b2 = new B(2);  
        A a1 = new A(3,b1);  
        A a2 = new A(3,b2);  
        if (a1.equals(a2)) System.out.println("Cierto");  
        else System.out.println("Falso");  
    }  
  
}
```

- ¿Qué imprime este programa en la salida estándar?

Ejemplo 2b: igualdad entre objetos

```
public class B  
  
    private int z;  
  
    public B(int z){  
        this.z=z;  
    }  
  
    public boolean equals(B b){  
        return (this.z == b.z);  
    }  
  
}
```

```
public class A {  
  
    private int x;  
    private B b;  
  
    public A(int x, B b){  
        this.x = x;  
        this.b = b;  
    }  
  
    public boolean equals(A a){  
        return (this.x==a.x  
                && this.b.equals(a.b));  
    }  
  
}
```

Ejemplo 2b: igualdad entre objetos

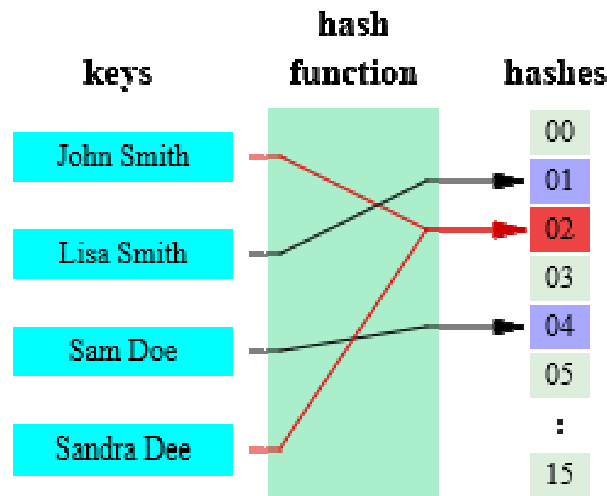
```
public class Main {  
  
    public static void main(String[] args) {  
        B b1 = new B(2);  
        B b2 = new B(2);  
        A a1 = new A(3,b1);  
        A a2 = new A(3,b2);  
        if (a1.equals(a2)) System.out.println("Cierto");  
        else System.out.println("Falso");  
    }  
  
}
```

- Qué imprime este programa en la salida estándar?

El método `equals()` y el método `hashCode()`

- Propiedades que debería preservar el método sobrescrito `equals()`
 - $a.equals(b) \Leftrightarrow b.equals(a)$ (simetría)
 - $a.equals(a)$ (reflexividad)
 - $a.equals(b) \text{ and } b.equals(c) \Rightarrow a.equals(c)$ (transitividad)
- Y el sentido común dice que:
 - $a == b \Rightarrow a.equals(b)$ (coherencia1)
- Cada clase hereda el método `hashCode()` de la clase `Object`
 - Es utilizada por colecciones tales como `Hashtable` y `HashMap`
- Siempre debería ser cierto que:
 - $a.equals(b) \Rightarrow a.hashCode() == b.hashCode()$ (coherencia2)
- Por tanto, una clase que sobrescribe `equals()`
 - también debe sobrescribir `hashCode()`

Ejemplo 3, *hash table*



Una función hash que mapea entre nombres y enteros de 0 a 15.
Hay una collision entre las claves "John Smith" y "Sandra Dee".

Source: Jorge Stolfi - Own work, Public Domain,
<https://commons.wikimedia.org/w/index.php?curid=6601264>

Copiar objetos con el método Java `clone()`

- La asignación entre objetos es una asignación de referencias.
- Para crear una copia de un objeto, existe el método Java `clone()`
 - Un método de la clase `Object` (que se puede sobrescribir)

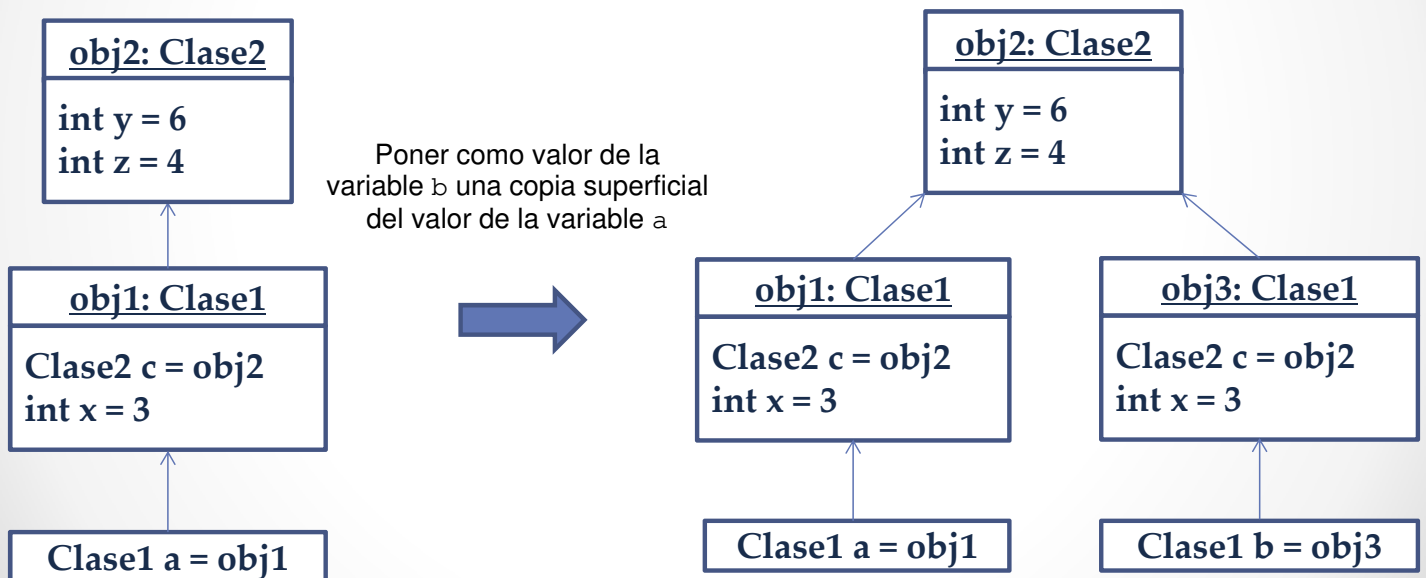
```
Fecha f1, f2 = new Fecha(25, 12, 17);  
f1 = f2.clone();    // Copia del objeto
```

```
if (f1 == f2)        // Falso (no apuntan al mismo objeto)  
    System.out.println("El mismo");
```

```
if (f1.equals(f2))   // Cierto (si hemos sobrescrito equals)  
    System.out.println("Iguales");
```

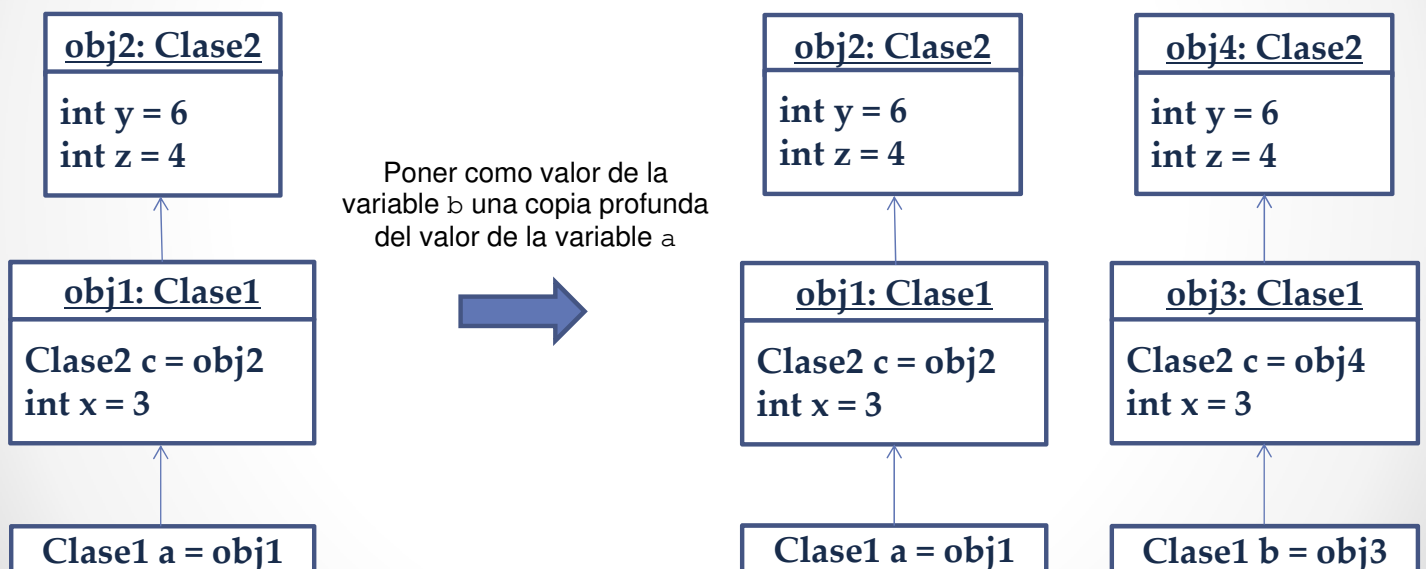
Ejemplo 4: copia superficial (*shallow copy*)

En pseudo-UML:



Ejemplo 4: copia profunda (*deep copy*)

En pseudo-UML:



Copias superficiales & profundas y el método `clone()`

- El método `Object.clone()` implementa una copia superficial
- Si se requiere una copia profunda, la documentación Java recomienda sobrescribir el método protegido `clone()` con un método público
 - Además, la clase en el que está el método sobrescrito debe implementar la interfaz `Cloneable`
- Sin embargo, Josh Bloch (“Effective Java”) y otros expertos avisan que implementar la copia profunda con `clone()` tiene muchos problemas
- En su lugar, recomiendan otras técnicas tal como el uso de un “constructor de copia”
 - Un constructor que toma un objeto de la misma clase como argumento

Tipos primitivos de Java

<u>Tipo</u>	<u>Tamaño</u>	<u>Valor mínimo</u>
<code>byte</code>	8 bits	-128
<code>short</code>	16 bits	-32768
<code>int</code>	32 bits	-2147483648
<code>long</code>	64 bits	$< -9 \times 10^{18}$
<code>float</code>	32 bits	+/- 3.4×10^{38} con 7 dígitos significativos
<code>double</code>	64 bits	+/- 1.7×10^{308} con 15 dígitos significativos
<code>boolean</code>	1 bit	true false
<code>char</code>	16 bits	Codificación UNICODE (interno: UTF-16)

Promoción automática (*widening*) de tipos en Java

- Un tipo A es de mayor rango que un tipo B
 - si A es un superconjunto de B
- Los valores de tipos primitivos siempre se pueden asignar a variables cuyo tipo es un tipo primitivo de mayor rango
 - `double > float > long > int > short > byte`
- Expresiones con operadores aritméticos (+, -, *, /, %)
 - Al operar con `byte` y `short`, estos se convierten implícitamente a `int`
 - Cuando los tipos de los operandos no coinciden
 - el operando de menor rango se convierte implícitamente al tipo de mayor rango
 - El resultado de la operación es del tipo de mayor rango

Tipos de Java: arrays

- Declaración de un array

```
int A[];           // A es un array de enteros
A[1] = 3;          // error, el array no ha sido creado
int A[10];         // error, el tamaño no va aquí
int[] A;           // A es un array de enteros
int A, B[];        // A es un entero, B un array de enteros
int[] A,B;         // A y B son arrays de enteros
int [][] C;        // C es un array de enteros de 2 dimensiones
```

- Creación de un array

```
A[] = new int[10]; // crear A como array de 10 enteros
```

- Declaración + creación de un array

```
int A[] = new int[10];
```

Tipos de Java: arrays

- Acceso a los elementos de un array (suponiendo array de 10 int)
`A[0], A[1], ..., A[9] // correcto`
`A[10] // error, índice fuera de rango (excepción)`
- Modificación de un array (asignando un valor a un elemento)
`A[3] = 8;`
- Un array es un “pseudo-objeto”
 - Se manipula a los arrays como si fueran objetos (tipo de referencia)
 - pero no existe una definición de clase correspondiente
- Tamaño de un array: `length`
 - Un “pseudo-atributo” (`final`), no un “pseudo-método”:
`A.length // correcto`
`A.length() // error`
 - Nótese que no da el número de elementos ocupados del array

• §3 - 21

Tipos de Java: arrays

- Inicialización de un array
`for (int i = 0; i < A.length; i++)`
`A[i] = 2*i;`
 - Uso del bucle for-each (desde Java 5) con un array (*)
`for (int elem : A)`
`System.out.println(elem); // imprime el contenido`
 - Declaration + creation + initialisation
`int A[] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};`
 - Copia (superficial) de un array
`B = A.clone(); // desde Java 5, no hace falta usar un cast`
- * Se puede usar "for-each " para cualquier clase que implementa la interfaz `Iterable`
`// Añadir 1 a cada elemento del "iterable" intlist`
`for(int integer : intlist){ integer++; }`

• §3 - 22

Ejemplo: arrays son “pseudo-objetos”

- La asignación del valor de una variable array a otra variable array
 - hace que ambas variables apunten al mismo array
 - por lo que el cambio en uno es un cambio en el otro:

```
int [] v1 = new int [10];
int [] v2;
v2 = v1;
v1[0] = 7;
System.out.println(v2[0]); // ¿qué valor se imprime? ¿0 o 7?
```

Arrays multidimensionales

- Una array bidimensional
 - es un array de referencias a arrays unidimensionales
- Se crea un array bidimensional como sigue:

```
int[][] m;
m = new int [3][];
m[0] = new int [2];
m[1] = new int [2];
m[2] = new int [2];

int[][] m = new int [3] [2]; // es equivalente a lo anterior
```

Manipulación de arrays de objetos

- Un array de objetos puede no estar lleno
 - es decir, no tener todos los elementos inicializados
- Antes de acceder a un objeto de un tal array
 - hay que estar seguro de que en la posición a la que se quiere acceder efectivamente hay una referencia a un objeto.
- Es estos casos, se suele comprobarlo, por ejemplo

```
for (int i = 0; i < banco.length; i++)  
    if (banco[i] != null) { // comprueba que hay un objeto  
        System.out.println(banco[i])  
    }
```

Tipos de datos de Java: enumerados

- El tipo `enum` se introdujo en Java 5
- Consta de un conjunto de valores constantes

```
public enum Dia {LUNES, MARTES, MIÉRCOLES, JUEVES, VIERNES};  
Dia dia = Dia.MARTES;  
System.out.println(dia);
```
- Todos los enums extienden implícitamente a la clase `Enum`
 - Un enum es equivalente a la declaración de una clase
 - cuyos atributos se declaran `public`, `static`, y `final`
 - cuyos nombres son los valores de la enumeración
 - salvo que lo puede chequear el verificador de tipos
- El método estático `values()` permite utilizar la construcción “for-each”

```
for (Dia d : Dia.values()) { System.out.println( p ) }
```

Tipos de datos de Java: cadenas de caracteres

- En Java `String` es una clase del paquete `java.lang`

- Declaración de una cadena

```
String t; // declaración
String v = new String("Hola"); // declaración e inicialización
String s = "Hola, "; // declaración e inicialización
String r = "esto es un string"; // declaración e inicialización
```

- Mostrar una cadena

```
System.out.println(s);
System.out.println(t); // error: t no tiene valor
```

Tipos de datos de Java: cadenas de caracteres

- Asignación de un valor cadena a una variable de tipo `String`

```
t = "Noam Chomsky es al autor vivo más citado";
```

- Concatenación de cadenas con el operador '+'

- El resultado es de tipo `String`

- Ejemplos

```
s + r // "Hola, esto es un string"
t = s + ", ¿qué tal?"; // "Hola, que tal?"
```

- Tamaño de una cadena

```
s.length() // 6
```

El valor devuelto por el método `length()` es de tipo `int`

Tipos de datos de Java: cadenas de caracteres

- Extracción de un carácter

```
s[0]           // error: una cadena no es un array de char
s.charAt(0)    // 'H'
s.charAt(4)    // ','
```

El valor devuelto por el método `charAt()` es de tipo `char`

- Extracción de subsecuencias

```
s.substring(2,5)    // "la,": intervalo [2,5) has 5-2 chars
```

El valor devuelto por el método `substring()` es de tipo `String`

- Comparación de cadenas

```
s.equals("Hola, ") // true
s == "Hola, "      // false, compara referencias
s.compareTo(r)     // < 0, == 0 ó > 0
```

• §3 - 29

Tipos de datos de Java: cadenas de caracteres

- Conversión de tipos: si uno de los operandos de '+' es de tipo `String`

- o el otro se convierte implícitamente a `String` (vía método `toString`)
- o Ejemplos (ambas resultan en una llamada a `toString()` de `Position`):

```
Position pos = new Position(3,4);
System.out.println("pos = " + pos);
String t = "El valor de pos es " + pos
```

- Operaciones prohibidas

```
s[0] = 'h';           // error: las cadenas no son arrays
s.charAt(0) = 'h';    // error: las cadenas son inmutables
s = s+1;              // no hay aritmética de referencias; denota s + "1"
```

- Argumento del método `main` es un array de `String` (llamado `args`)

- o Se utiliza para pasar los argumentos de la línea de comandos al programa
- o Primer elemento, `arg[0]`, contiene primer argumento, no nombre de programa

• §3 - 30

Ejemplo 5: manipulación de la clase String

```
public class MisUtilidadesCadenas{

    public static String invertir (String entrada) {
        String salida = "";
        for (int i=0; i < entrada.length(); i++)
            salida = entrada.charAt(i) + salida;
        return salida;
    }
    // Clase String de la biblioteca estándar contiene varios métodos replace()
    public static String reemplazar(String cad, String subIn, String subOut){
        int startSubIn = cad.indexOf(subIn);
        if (startSubIn != -1)
            cad = cad.substring(0, startSubIn)
                + subOut
                + cad.substring(startSubIn + subIn.length());
        return cad;
    }
}
```

● §3 - 31

Ejemplo 5: manipulación de la clase String

```
public class StringyThings{

    public static void main(String args[]){
        String s1 = "rats live on no evil star";
        String out1 = "It's equality Jim but not as we know it."
        String out2 = "It's equality; not a lot of people know that."

        String s2 = MyStringUtils.invert(s1);
        if (s1 == s2) System.out.println(out1);
        if (s1.equals(s2)) System.out.println(out2);

        String s3 = s1;
        s1 = MyStringUtils.replace(s1, "on no", "on - madam - no");
        if (s1 == s3) System.out.println(out1);
        if s1.equals(s3)) System.out.println(out2);
    } // ¿qué es lo que se imprime en la entrada estándar?
}
```

● §3 - 32

El método `toString()`

- Todas las clases heredan el siguiente método de la clase `Object`:

```
public String toString()
```

Y (casi) todas las clases deberían sobre-escribirlo.
- Construye una representación textual del objeto propietario
 - Eso es, el objeto referenciado por `this`
- El método `toString()` de una clase se invoca por el sistema
 - al pasar como parámetro un objeto de esta clase al método `print` o `println` de la clase `PrintStream`
 - al enviar un objeto de esta clase a la salida estándar o salida de error en cualquier otro caso
 - P.ej. para imprimir su estado en un “punto de parada” (*breakpoint*) durante la depuración del programa
 - al concatenar un objeto de esta clase con una cadena de caracteres

El método `toString()`

- Si no se sobreescribe devuelve la cadena que consta de
 - el nombre de la clase
 - seguido de una arroba
 - seguido por el valor devuelto por el método `hashCode()` invocado sobre este mismo objeto
- Esta funcionalidad no suele ser la deseada
 - Se suele sobreescribir este método para poder construir la representación textual más adecuada
 - Ejemplo: la clase `GamePrinter` de la primera práctica

La clase `StringBuilder`

- Un problema con `String`
 - Los objetos de la clase `String` son inmutables
 - Uso excesivo de memoria al construir una cadena dentro de un bucle
- `Java.lang.StringBuilder` proporciona un string mutable
 - Tiene los siguientes métodos:
 - `insert`, `append`, `setCharAt`, ...
 - `append` y `insert` están sobrecargados para aceptar datos de cualquier tipo
 - Para devolver el `String` creado hay que llamar al método `toString()`
- Ejemplo

```
StringBuilder holamundoBuilder = new StringBuilder();
holamundoBuilder.append("Hola, c-po");
holamundoBuilder.insert(8, 3);
String holamundo = holamundoBuilder.toString();
```

Tipos valor y tipos referencia

- Existen dos categorías de tipos de datos en Java:
 - Los tipos valor:
 - Las variables de estos tipos almacenan directamente los datos (tipados).
 - Los únicos tipos valor en Java son los tipos primitivos: `boolean`, `int`, etc.
 - Los tipos referencia:
 - Las variables de estos tipos almacenan referencias (tipadas) a los datos
 - eso es, almacenan la localización de los datos, es decir, la dirección de memoria donde está almacenado el principio de los datos
 - Los tipos referencia en Java son los arrays y las clases
 - La referencia nula se denota `null`
 - Al declarar una variable de tipo-referencia, su valor se inicializa a `null`
 - Un intento de acceder a los datos referenciados por una variable cuyo valor es `null` produce una excepción en tiempo de ejecución ("null pointer exception")

Tipos valor y tipos referencia

- Variables de tipo valor
 - *En la declaración:* se reserva espacio en memoria para el dato
 - *En la asignación:* se copia el dato al espacio reservado
- Variables de tipo de referencia
 - *En la declaración:* se reserva espacio en memoria para una referencia
 - inicialmente su valor es `null`
 - *En la asignación:* se copia la referencia a los datos al espacio reservado
- Operador `new` para crear un nuevo elemento de tipo referencia
 1. Se reserva espacio para el objeto o el array
 2. En el caso de un objeto, se ejecuta el constructor
 3. Se devuelve la referencia al objeto o array creado
 4. Se asigna la referencia devuelta a una variable de tipo referencia

• §3 - 37

Diferencias, valores de tipos primitivos y objetos

Declaración

Reservar espacio en memoria para un valor

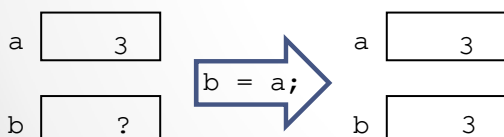
Inicialización

Poner el valor inicial en el espacio reservado

Asignación

Copiar el valor

```
int a = 3;  
int b;  
b = a;
```

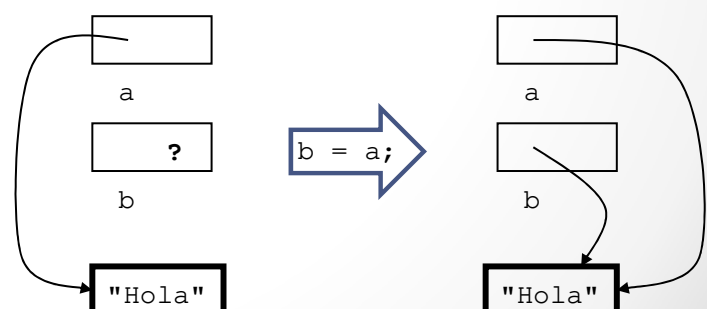


Reservar espacio en memoria para una referencia de objeto

Crear un objeto y poner su referencia en el espacio reservado

Copiar la referencia

```
String a = new String("Hola");  
String b;  
b = a;
```



• §3 - 38

Paso de un tipo-referencia como argumento

- En Java, los argumentos se pasan siempre por valor
- Si se pasa un tipo referencia a un método, en el cuerpo del método
 - se asigna una copia de la referencia pasada al parámetro correspondiente
 - cualquier *modificación del valor del parámetro*
 - eso es, un cambio en la referencia almacenada en el parámetro,
 - no puede afectar a ninguna variable de tipo-referencia externa al método
 - cualquier *modificación de los datos referenciados por el valor del parámetro*
 - afecta a todas aquellas variables de tipo-referencia externas al método
 - que referencian estos mismos datos
- Es decir, un método con parámetros de tipo referencia
 - puede hacer cambios al estado del programa que sobreviven a la ejecución del método
- De este modo, el paso de un tipo-referencia por valor
 - parece el paso por referencia de C y C++ (parámetros de entrada/salida)

• §3 - 39

•

Ejemplo 6: paso de un tipo-valor

```
/* Este método NO funciona. Se trabaja sobre copias, por
   lo que los cambios introducidos en el cuerpo del
   método no sobreviven a la ejecución del método; cuando
   la ejecución termina se pierden */
void swap(int a, int b){
    int aux = a;
    a = b;
    b = aux;
}
```

• §3 - 40

•

Ejemplo 7: paso de un tipo referencia

```
// Método de la clase Ejemplo
public void cambiarContenido(Fecha f) {
    // Cambiar el estado del objeto 'Fecha' referenciado por la
    // variable 'f' (incrementando el valor del atributo dia)
    f.avanzaUnDia();
}
Ejemplo obj = new Ejemplo();
Fecha unaFecha = new Fecha(25, 12, 2017);
// La referencia contenida en 'unaFecha' se copiará a 'f'
obj.cambiarContenido(unaFecha);

/* Después de la ejecución de 'cambiarContenido', la ejecución de
'unaFecha.getDia()' devolvería 26.
El método 'cambiarContenido' cambió el estado del objeto de la
clase 'Fecha' referenciada por 'f' (dentro del alcance del
método) y por tanto también el del objeto de la clase 'Fecha'
referenciada por 'unaFecha', ya que es el mismo objeto. */
```

● §3 - 41

Ejemplo 7: paso de un tipo referencia

```
// Método de la clase Ejemplo
public void cambiarRef(Fecha f) {
    // Cambiar la referencia contenido en la variable 'f'
    f = new Fecha(1, 1, 2000);
}
Ejemplo obj = new Ejemplo();
Fecha unaFecha = new Fecha(12, 10, 1492);
// La referencia contenida en 'unaFecha' se copiará a 'f'
obj.cambiarRef(unaFecha);

/* Después de la ejecución de 'cambiarRef', la ejecución de
'unaFecha.getDia()' devolvería 12.
El método 'cambiarRef' cambió la referencia contenida en 'f'
pero no cambió el estado del objeto 'Fecha' referenciado por
'f' al empezar la ejecución del método. Por tanto, la
ejecución del método 'cambiarRef' deja sin cambios al estado
del objeto 'Fecha' referenciada por 'unaFecha'. */
```

● §3 - 42

Clases envoltura (*wrapper classes*)

- Para cada tipo primitivo, existe una clase envoltura en `java.lang`
 - Un objeto de la clase `Byte` puede usarse para envolver un valor de tipo `byte`
 - Un objeto de la clase `Short` puede usarse para envolver un valor de tipo `short`
 - Un objeto de la clase `Integer` puede usarse para envolver un valor de tipo `int`
 - Un objeto de la clase `Long` puede usarse para envolver un valor de tipo `long`
 - Un objeto de la clase `Float` puede usarse para envolver un valor de tipo `float`
 - Un objeto de la clase `Double` puede usarse para envolver un valor de tipo `double`
 - Un objeto de la clase `Character` puede usarse para envolver un valor de tipo `char`
 - Un objeto de la clase `Boolean` puede usarse para envolver un valor de tipo `boolean`

Clases envoltura (*wrapper classes*)

- ¿Para qué sirven las clases envoltura?
 - Permiten tratar valores de tipos primitivos como objetos
- Los objetos de las clases envoltura son inmutables
 - Es decir, el valor que contiene un tal objeto no puede cambiarse
 - ⇒ Operaciones tales como adición y substracción crean un nuevo objeto
 - ⇒ No se pueden usar para pasar tipos primitivos como parámetros de E/S
- ¿Qué interés tienen?
 - Para poder usar el valor especial `null` con tipos primitivos
 - Para poder usar tipos primitivos como valores de “parámetros de tipo”
 - En particular, en colecciones tales como `ArrayList<T>`
 - Declarar un tipo como `ArrayList<int>` es un error
 - En Java, tipos que contienen parámetros de tipo se llaman *tipos genéricos*
 - Para poder usar tipos primitivos en el contexto del polimorfismo de subclase

Boxing y unboxing

- *(Auto)boxing*
 - Conversión (automática) del tipo primitivo al tipo envoltura correspondiente
 - Implica la creación de un objeto
 - que se realiza de forma transparente al programador
- *Unboxing*
 - Conversión automática del tipo envoltura al tipo primitivo correspondiente

- **Ejemplo.**

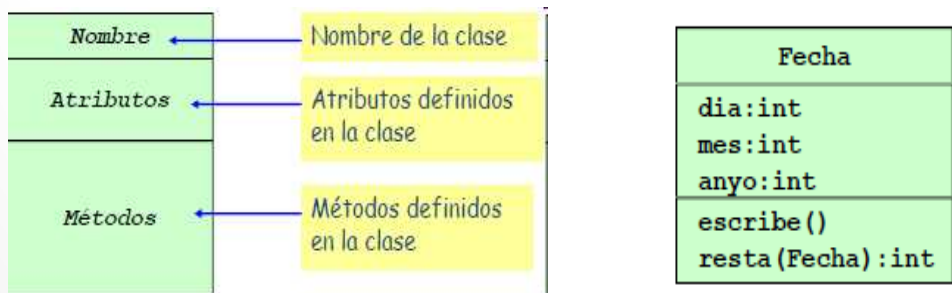
```
Integer a;  
int i = 0;  
a = 3;           // Boxing: a = new Integer(3);  
a = i;           // Boxing: a = new Integer(i);  
i = a;           // Unboxing: i = a.intValue();
```

Diseño OO: representación gráfica de clases

- Utilizaremos diagramas de clase UML para representar gráficamente
 - los elementos de los programas orientados a objetos
 - y las relaciones entre ellos
- A medida que vayamos estudiando conceptos / mecanismos de la POO
 - iremos viendo cómo representarlos gráficamente en UML
- Hay dos tipos de herramienta UML
 - Herramientas de modelado UML
 - Producen y almacenan modelos UML
 - Sintaxis abstracta (un modelo se representa como una instancia del meta-modelo UML)
 - Un diagrama UML es una vista gráfica de un modelo UML almacenado
 - La parte del modelo que es visible en cada vista (diagrama) suele ser configurable
 - Herramientas de dibujo UML
 - Producen y almacenan diagramas UML

Diagrama de clases UML: clase

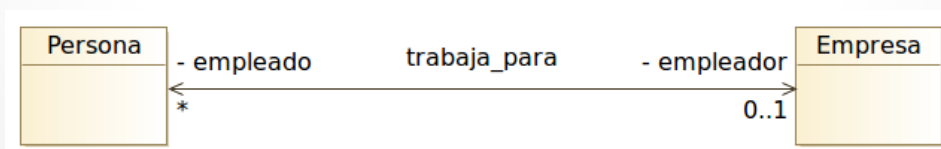
- Una caja denota una clase:
 - Tiene tres secciones: una para el nombre de la clase, otra para sus atributos y otra para sus métodos.
 - UML tiene su propia sintaxis para especificar atributos y métodos
 - Las secciones de atributos y de métodos pueden dejarse en blanco o incluso ser omitidas, dependiendo del nivel de detalle requerido.



• §3 - 47

Diagrama de clases UML: asociación

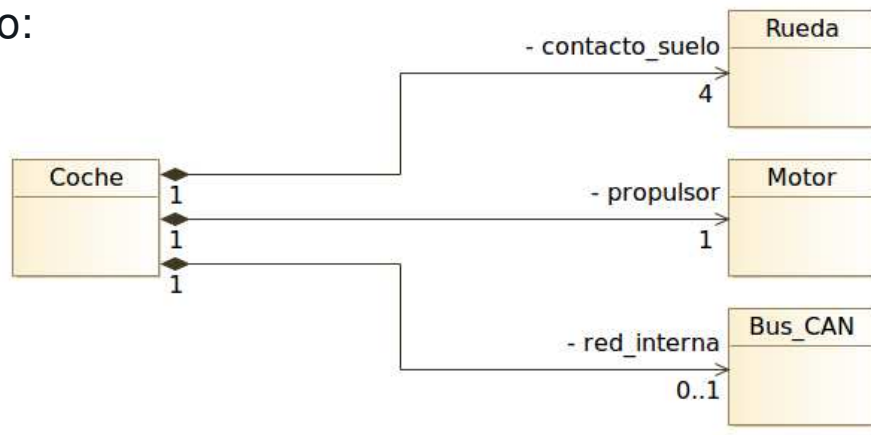
- Una línea básica entre dos clases denota una *asociación* entre ellas
 - La asociación puede, o no, tener nombre
- Una flecha denota que la asociación es *navegable* en esta dirección
 - Sin flecha en ninguno de los lados \equiv una flecha en ambos lados
- En cada lado de una asociación se puede poner:
 - La *multiplicidad*: la cardinalidad de la participación de la clase en la asociación
 - El *rol* que desempeña un objeto de la clase en la asociación
 - Corresponde a un atributo de la clase al otro lado de la asociación
- Ejemplo
 - Cada objeto de la clase `Persona` trabaja_para 0 o 1 objetos de la clase `Empresa`
 - Cada objeto `Empresa` tiene 0 o muchos objetos `Persona` que trabaja_para el



• §3 - 48

Diagrama de clases UML: composición

- Una línea con un rombo relleno en un lado denota una composición
 - Eso es, un objeto compuesto por otros objetos
 - Se entiende que si se borra un compuesto, se borra sus componentes
 - El rombo se coloca del lado de la clase compuesta
 - Se puede incluir información de multiplicidad
- La composición es un caso especial de una relación “parte-todo”
- Ejemplo:



• §3 - 49

Universidad Complutense de Madrid
Facultad de Informática
Grado en Ingeniería Informática, Grupo B

Tecnología de la programación

Apéndice

Implementar una lista de objetos con un array

- Se definirá una clase lista implementada con un array
 - El array de objetos se almacenará en un atributo de la clase
- No todas las posiciones del array estarán ocupadas por objetos
 - Debemos saber cuántas posiciones están ocupadas (y cuáles).
- Por ello, mantenemos todos los objetos en posiciones contiguas,
 - a partir de la primera
 - Lo único que necesitamos saber es cuántos hay
 - por lo que utilizaremos un contador de objetos (como atributo)
- El contador indicará la primera posición disponible
 - Se inicializará a cero
 - La lista estará llena cuando el contador llegue a su valor máximo
 - Podría redimensionarse cuando hace falta pero aquí no lo hacemos

Implementar una lista de objetos con un array: métodos

- Diseño de los métodos de la clase lista
 - `length()` devolverá el número de elementos en la lista
 - `llena()` devolverá `true` si no caben más objetos y `false` si no
 - `vacía()` devolverá `true` si no hay objetos y `false` si no
 - `insertar()` insertará un objeto al final de la lista
 - `borrar()` borrará un objeto de una posición dada de la lista
 - `get()` devolverá el objeto en una posición dada de la lista
 - `toString()` devolverá una representación textual de la lista
- Ocultación de los detalles de la implementación
 - Podríamos querer esconder de los clientes de la clase el hecho de que la lista está implementado por un array (cuya numeración empieza en 0), en particular, para que puedan usar una numeración que empieza en 1 para identificar la posición en los métodos `borrar()` y `get()`. Sin embargo, aquí no lo hacemos.

Implementar una lista de objetos con un array: errores

Tratamiento de errores (en ausencia de un mecanismo de excepciones)

- Si un método no tiene que devolver resultado, se puede devolver un `boolean` que indica el éxito o el fallo de la operación
 - `insertar()` devuelve `false` si la lista ya está llena
 - Lista que puede redimensionarse: tipo de retorno `void` (nunca está llena)
 - `borrar()` devuelve `false` si intenta borrar el contenido de una posición donde no hay objeto (incluye el caso de la lista vacía).
- Si un método tiene que devolver un objeto, se puede indicar que la operación ha fallado devolviendo el valor especial `null`
 - `get()` devuelve `null` si se intenta recuperar el contenido de una posición donde no hay objeto (incluye el caso de la lista vacía).

Ejemplo: implementar una lista de piezas con un array

```
public class ListaPiezas {
    private final static int MAX = 100;
    private Pieza[] piezas;
    // Indica el número de elementos
    // Coincide con la primera posición libre del array
    private int contador = 0;
    // Constructor sin argumentos crea el array
    public ListaPiezas() {
        piezas = new Pieza[MAX];
    }
    // si lista se puede redimensionar, llena es privado
    public boolean llena() { return contador == MAX; }
    public boolean vacia() { return contador == 0; }
    public int length() { return contador; }
```

Ejemplo: implementar una lista de piezas con un array

```
public boolean insertar(Pieza p) {
    // lista redimensionable: redimensionarse si llena
    if llena() return false;
    piezas[contador] = p;
    contador++;
    return true;
}

public boolean borrar(int pos){
    if((pos < 1) || (pos > contador)) return false;
    for(i=pos, i < contador-1, i++)
        piezas[i] = piezas[i+1];
    contador--;
    return true;
}
```

Ejemplo: implementar una lista de piezas con un array

```
public Pieza get(int pos) {
    if((pos < 1) || (pos > contador-1)) return null;
    return piezas[pos];
}

public String toString() {
    // %n en format: salto de línea independiente de plataforma
    String cad = String.format("Elementos de la lista:%n%n");
    for(int i = 0; i < contador; i++)
        // llamada implícita a toString() de Pieza
        cad += String.format("%s%n", piezas[i]);
    return cad;
}
}
```

Ejemplo: uso de lista de piezas implementada con array

```
public static void main(String args[]) {  
    // suponer que la clase Pieza está definida  
    Pieza p1 = new Pieza("Broca dim 6mm", 21);  
    Pieza p2 = new Pieza("Bateria", 165);  
    Pieza p3 = new Pieza("Correa ventilador", 63.98);  
    ListaPiezas miLista = new ListaPiezas();  
    miLista.insertar(p1);  
    miLista.insertar(p2);  
    miLista.insertar(p3);  
    System.out.println(miLista); // llamada a toString  
    Pieza p = miLista.recuperar(0);  
    p.setPrecio(4.1); // suponer: Pieza tiene método setPrecio  
    miLista.insertar(new Pieza("Bombilla 10W", 9.80));  
    miLista.delete(p3);  
    System.out.println(miLista);  
}
```

• §3A - 57

Implementar una lista genérica de objetos con un array

- ¿No podemos definir una clase lista para crear listas de lo que sea?
 - Es decir, tener una sola clase genérica en vez de una clase específica para cada tipo de contenido
- Todavía no queremos usar las colecciones de la biblioteca estándar
 - Utilizan los tipos genéricos de Java que no estudiaremos hasta enero
- ¿Cómo podemos hacerlo sin usar los tipos genéricos de Java?
 - Usar el mecanismo que utilizaban los programadores Java antes de Java 5
 - Usar *polimorfismo de subtipo* y el hecho de que los arrays de Java son *covariantes*.
 - Se explicarán estos términos en el siguiente capítulo
 - Es decir, declarar un array de tipo `Object []`
 - Preguntas al respecto de este mecanismo
 - ¿Cómo usar nuestra clase de lista genérica para definir, p.ej. una lista de `int`?
 - ¿Cómo usar un método que devuelve un objeto de la clase `Object` (p.ej. `get`)?

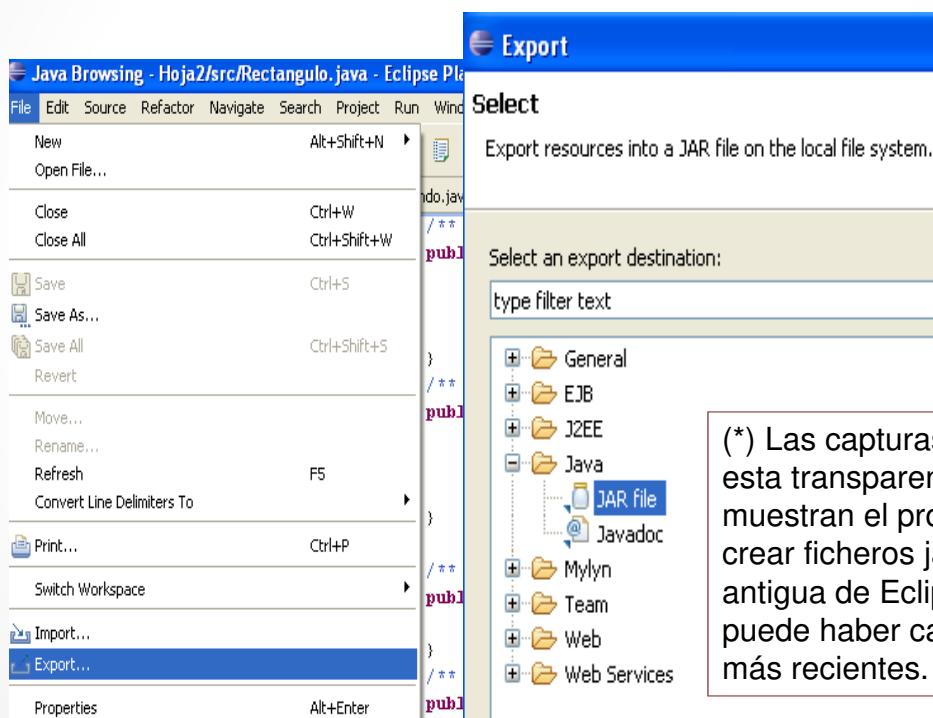
• §3A - 58

Ficheros *java archive* (.jar)

- Los Ficheros con extensión .jar son ficheros comprimidos (con zip) de *bytecode* de clases (ficheros .class)
 - También pueden incluir metadatos tales como
 - ficheros asociados, p.ej. imágenes
 - información de configuración o de seguridad,
 - un *classpath* (lista de caminos a otros ficheros .jar)
 - la localización de la clase punto de entrada, es decir, que contiene el `main`
 - Se puede extraer el contenido con cualquier software `unzip`
 - o con el comando `jar`
- Constituye una manera eficiente de desplegar un conjunto de clases y los recursos asociados.
- Si alguna clase tiene `main`, el .jar se puede crear como ejecutable.
 - Para ejecutarlo, se necesita el JRE (incluido en el JDK)
 - Se puede poner/actualizar la información sobre el punto de entrada del programa con la opción `e` del comando `jar`

Crear un fichero *java archive* (.jar)

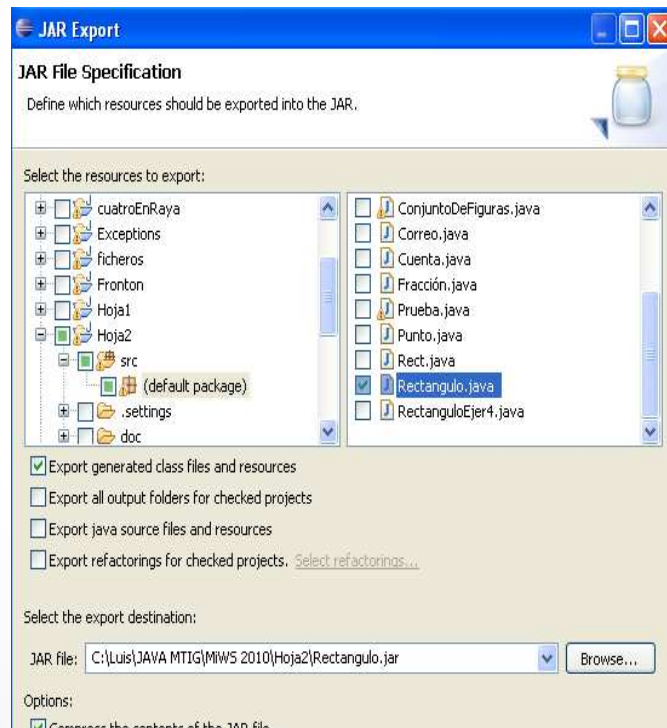
- Para crear un fichero .jar ejecutable en Eclipse (*)



(*) Las capturas de pantalla de esta transparencia y las siguientes muestran el procedimiento para crear ficheros jar con una versión antigua de Eclipse. El procedimiento puede haber cambiado en versiones más recientes.

Crear un fichero *java archive* (.jar)

- Especificar las clases contenidas y el directorio destino

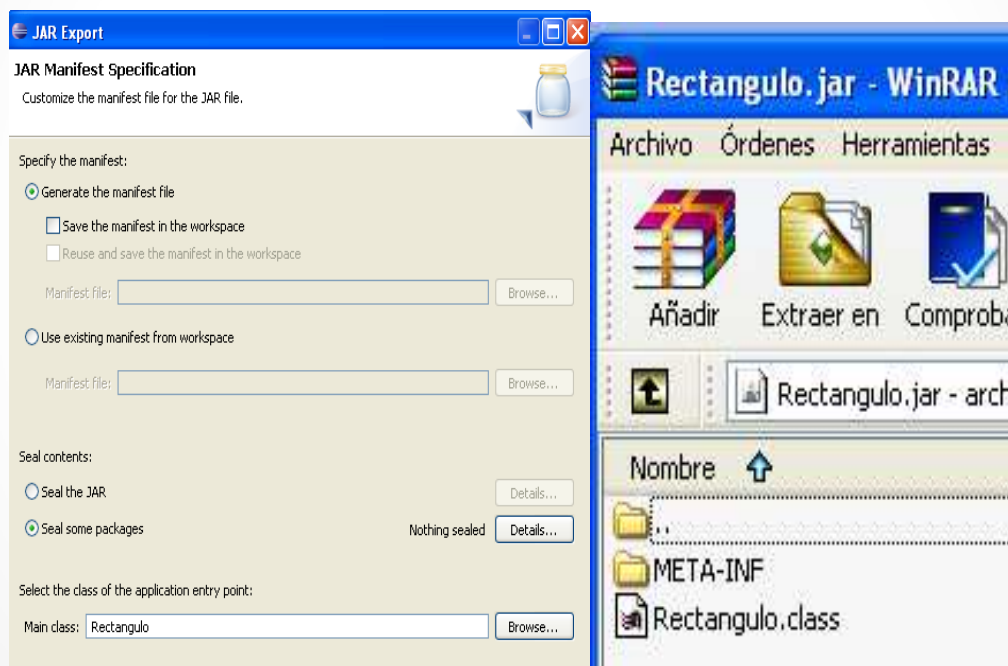


• §3A - 61

• 61

Crear un fichero *java archive* (.jar) ejecutable

- Especificar la clase que contiene el `main`

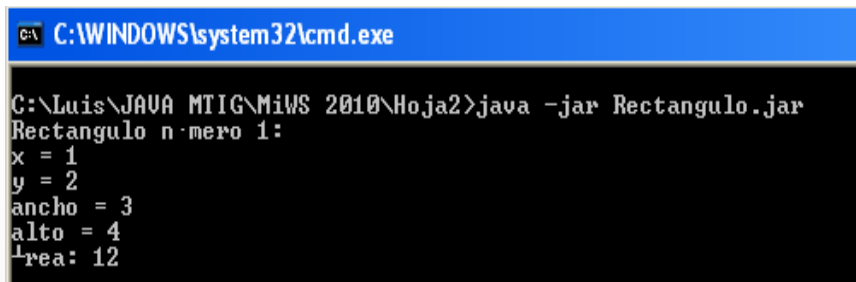


• §3A - 62

• 62

Ejecutar un fichero *java archive* (.jar) ejecutable

- Se ejecuta un fichero .jar con la opción -jar del comando java
- También se puede configurar el SO para ejecutarlo con un doble click



```
C:\WINDOWS\system32\cmd.exe
C:\Luis\JAVA MTIG\MiWS 2010\Hoja2>java -jar Rectangulo.jar
Rectangulo n-mero 1:
x = 1
y = 2
ancho = 3
alto = 4
Area: 12
```