

Tema 3.- Clases

1	Cuestionario sobre Smalltalk	2
2	Ejemplo de clase en Smalltalk: Stack	3
3	Ejemplo de clase en Smalltalk: El juego de las ocho Reinas.....	6
4	Ejemplo de clase en Smalltalk: ContabilidadMacana.....	9
5	Modificadores de Clase en JAVA.....	14
6	Modificadores de variables (campos o datos miembro)	14
6.1	Modificadores de acceso (ámbito o visibilidad)	14
6.2	Otros modificadores.....	14
7	Modificadores de métodos (funciones miembro)	14
7.1	Modificadores de acceso (ámbito o visibilidad)	14
7.2	Otros modificadores.....	14
8	Ejemplo de clase en JAVA: Carta	15
9	Ejemplo de clase en JAVA: POINT	17
10	Ejemplo de clase en JAVA: Cuenta.....	18
11	Clases y objetos en los lenguajes con tipos	20
11.1	Concepto de tipo en los LOO.....	20
11.1.1	Utilidad	20
11.1.2	Regla de compatibilidad de tipos.....	20
11.2	Ligadura mixta y verificación de tipos	22
11.3	Características generales de los LOO con tipos.....	23
11.3.1	Tipos básicos y objetos	23
11.3.2	Features o miembros	23
11.3.3	Mensajes	23
11.3.4	Creación e inicialización de objetos	24
11.3.5	Objetos en el Heap y objetos en el Stack.....	24
12	Polimorfismo en JAVA (1).....	26
13	Polimorfismo en JAVA (2).....	27
14	Polimorfismo en Java (3).....	28
15	Algunas colecciones (antiguas) de JAVA.....	29
16	Ejemplo de Applet	30
17	Cuestionario sobre JAVA	31

1 Cuestionario sobre Smalltalk

1. ¿Qué son los argumentos de un mensaje con palabras clave?
2. ¿Qué tipos de variables se manejan en Smalltalk?
3. ¿En función de qué criterio se hace esta distinción de variables?
4. ¿Qué son las variables de instancia con nombre?
5. ¿Cómo se declaran las variables de instancia con nombre?
6. ¿Qué son las variables de instancia indexadas?
7. ¿Cómo se declaran las variables de instancia indexadas?
8. ¿Cómo se sabe el número de variables de instancia indexadas?
9. ¿Pueden convivir las variables de instancia con nombre y las indexadas?
10. ¿Si una clase tiene variables de instancia indexadas, ¿qué sucede con las subclasses?
11. Ejemplos de clases con variables de instancia indexadas
12. Tipos de variables compartidas
13. ¿Para qué sirven las variables de clase? ¿Cómo se declaran?
14. ¿Qué son las variables de pool?
15. ¿Qué es el pool Smalltalk?
16. ¿Qué es un método?
17. Diferencias y semejanzas entre métodos de clase y de instancia
18. ¿Cuál es el valor devuelto por un método? ¿De qué depende?
19. ¿Qué es la pseudovariable self? ¿Para qué sirve?
20. Ponga dos ejemplos de uso de self
21. ¿Qué es la pseudovariable super? ¿Para qué sirve?
22. ¿Qué es una variable temporal? ¿Cómo se declara? ¿Cuál es su tipo y valor inicial?
23. ¿Qué es la máquina virtual de Smalltalk?
24. ¿Qué es la imagen virtual de Smalltalk?
25. ¿Qué es y cómo se llama a un método primitivo?
26. ¿Qué significa que un argumento es una pseudovariable variable?

2 Ejemplo de clase en Smalltalk: Stack

'From VisualWorksÂ®, Release 3.0 of February 5, 1998 on October 26, 2001 at 2:13:09 pm'!

```
Object subclass: #Stack
  instanceVariableNames: 'elems '
  classVariableNames: "
  poolDictionaries: "
  category: 'HEDES-aux'!
```

```
!Stack methodsFor: 'management'!
```

```
isEmpty
"      Abstract: Yes/No
      Requires:
      Effect: determines if the receiver is empty
      Modifies:
      Answer: true or false
      Author: Maria Jose
      Version: 1
      Date:May 23, 1998"
```

```
^(elems isEmpty).!
```

```
pop
"      Abstract: Yes/No
      Requires:
      Effect: removes the element of the top of the stack
      Modifies: elements
      Answer: the element of the top
      Author: Maria Jose
      Version: 1
      Date:May 23, 1998"
```

```
(elems isEmpty) ifFalse: [^elems removeFirst.].!
```

push: anElement

```
"    Abstract: Yes/No
    Requires:
    Effect: add a new element on the top of the stack
    Modifies: elements
    Answer:
    Author: Maria Jose
    Version: 1
    Date:May 23, 1998"
```

elems addFirst: anElement.!

size

```
"    Abstract: Yes/No
    Requires:
    Effect:
    Modifies:
    Answer: the size of elems
    Author: María José
    Version: 1
    Date:August 4, 1998"
```

^elems size.!!

!Stack methodsFor: 'initialization'!

initialize

```
"    Abstract: Yes/No
    Requires:
    Effect: initialize instance variables
    Modifies: instance variables
    Answer:
    Author: Maria Jose
    Version: 1"
```

elems:= OrderedCollection new.!!

Stack class

instanceVariableNames: ""

!Stack class methodsFor: 'creation'!

new

```
"    Abstract: Yes/No
    Requires:
    Effect:
    Modifies: creates a new Stack
    Answer:
```

Author: Maria Jose
Version: 1"

^(super new) initialize!!

3 Ejemplo de clase en Smalltalk: El juego de las ocho Reinas

“Colocar 8 reinas en un tablero sin que se amenacen. Resuelto con recursividad”

'From Smalltalk/X, Version:2.10.5 on 16-jul-1996 at 12:19:05 pm!'

```
Object subclass:#Reina
  instanceVariableNames:'columna fila vecina'
  classVariableNames:"
  poolDictionaries:"
  category:'ST/V classes'
!

!Reina methodsFor:'ST/V methods!'

columna
  ^columna!

asignarColumna: unNumero vecina:unaReina
  "inicia valores de columna y vecina par la nueva reina"
  columna := unNumero.
  vecina := unaReina!

avanzar
  "avanza la reina actual"

  (fila = 8) ifTrue: [(vecina siguiente) ifFalse: [^false].
                   fila := 0].
  fila := fila + 1.
  ^true!

fila
  ^fila!

primera
  "genera la primera solución aceptable"

  vecina primera.
  fila := 1.
  ^self verPosicion!

vecina
  ^vecina!
```

siguiente

"genera la siguiente solución aceptable"

^(self avanzar) and:[self verPosicion]!

verPosicion

"prueba y posiblemente avanza la reina actual"

[vecina verFila: fila columna: columna] whileTrue:
 [(self avanzar) iffFalse: [^false]].

^true!

verFila: unaFila columna: unaColumna

"verifica si esta reina, o cualquier vecina
pueden atacar la posicion dada en el argumento"

| diferenciasColumnas|

diferenciasColumnas := unaColumna - columna.

((fila = unaFila) or: [fila + diferenciasColumnas = unaFila]) or:
 [fila - diferenciasColumnas = unaFila])

ifTrue: [^true].

^vecina verFila: unaFila columna: unaColumna!

resultado

"produce una lista con el resultado actual"

vecina resultado addLast: fila! !

'From Smalltalk/X, Version:2.10.5 on 16-jul-1996 at 12:19:12 pm!'

Object subclass:#ReinaNula
 instanceVariableNames:"
 classVariableNames:"
 poolDictionaries:"
 category:'ST/V classes'

!

```
!ReinaNula methodsFor:'ST/V methods'!
```

```
primera
```

```
^true!
```

```
verFila: unaFila columna: unaColumna
```

```
^false!
```

```
resultado
```

```
^OrderedCollection new!
```

```
siguiente
```

```
^false! !
```

```
“-----”
```

"Código para ejecutar el programa de las ocho reinas"

```
|reina ultimaReina juego|
```

```
ultimaReina:= ReinaNula new.
```

```
juego:=Array new: 8.
```

```
1 to: 8 do: [ :i | reina:= Reina new.
```

```
    reina asignarColumna: i vecina: ultimaReina.
```

```
    juego at:i put: reina.
```

```
    ultimaReina:= reina.].
```

```
reina primera.
```

```
juego inspect.
```

"En juego se almacena una reina para cada columna del tablero. examinando la fila de cada una de estas reinas puede observarse que no se interfieren"

4 Ejemplo de clase en Smalltalk: ContabilidadMacana

'From VisualWorks(R), Release 2.5 of September 26, 1995 on January 14, 2002 at 1:09:38 pm'!

```
Object subclass: #ContabilidadMacana
  instanceVariableNames: 'debe haber saldo '
  classVariableNames: 'TotalSaldos '
  poolDictionaries: ''
  category: 'PdoEjemplos'!
```

```
!ContabilidadMacana methodsFor: 'consulta'!
```

```
consultaAdeudo: unString
```

```
" Abstract: Yes/No
```

```
Requires:
```

```
Effect: Consulta el total de adeudos en el capítulo unString
```

```
Modifies:
```

```
Answer:
```

```
Author: JosÃ© Parets
```

```
Version: 1.0
```

```
Date:January 14, 2002"
```

```
^ debe at: unString ifAbsent: ['No existe el capítulo de adeudos: ', unString].!
```

```
consultaIngreso: unString
```

```
" Abstract: Yes/No
```

```
Requires:
```

```
Effect: Consulta el total de adeudos en el capítulo unString
```

```
Modifies:
```

```
Answer:
```

```
Author: JosÃ© Parets
```

```
Version: 1.0
```

```
Date:January 14, 2002"
```

```
^ haber at: unString ifAbsent: ['No existe el capítulo de haberes: ', unString].!
```

```
saldo
```

```
" Abstract: Yes/No
```

```
Requires:
```

```
Effect: Consulta el saldo.
```

```
Modifies:
```

```
Answer:
```

```
Author: JosÃ© Parets
```

```
Version: 1.0
```

```
Date:January 14, 2002"
```

```
^ saldo! !
!ContabilidadMacana methodsFor: 'adeudos!'

adeudaEnCapitulo: unString laCantidad: unInteger
"    Abstract: Yes/No
    Requires:
    Effect: Adeuda en el capÃ-tulo la cantidad unInteger
    Modifies:
    Answer:
    Author: JosÃ© Parets
    Version: 1.0
    Date:January 14, 2002"

    debe at: unString ifAbsent: [debe at: unString put: 0].
    debe at: unString put: (debe at: unString) - unInteger.
    saldo := saldo - unInteger.
    TotalSaldos := TotalSaldos - unInteger.! !
```

```
!ContabilidadMacana methodsFor: 'ingresos'!
```

```
ingresaEnCapitulo: unString laCantidad: unInteger
"    Abstract: Yes/No
    Requires:
    Effect: Ingresa en el capÃ-tulo la cantidad unInteger
    Modifies:
    Answer:
    Author: JosÃ© Parets
    Version: 1.0
    Date:January 14, 2002"

    haber at: unString ifAbsent: [haber at: unString put: 0].
    haber at: unString put: (haber at: unString) + unInteger.
    saldo := saldo + unInteger.
    TotalSaldos := TotalSaldos + unInteger.! !
```

```
!ContabilidadMacana methodsFor: 'initialization'!
```

```
initialize
"    Abstract: Yes/No
    Requires:
    Effect: inicializa el receptor
    Modifies:
    Answer:
    Author:
    Version:
    Date:January 14, 2002"
    saldo := 0.
    haber := Dictionary new.
```

```

    debe := Dictionary new.!
initialize: anInteger
"    Abstract: Yes/No
    Requires:
    Effect: inicializa el receptor con saldo a anInteger
    Modifies:
    Answer:
    Author:
    Version:
    Date:January 14, 2002"
    saldo := anInteger.
    haber := Dictionary new.
    debe := Dictionary new.
    TotalSaldos := TotalSaldos + anInteger.! !
"-----"!

```

```

ContabilidadMacana class
    instanceVariableNames: "!

```

```

!ContabilidadMacana class methodsFor: 'creation'!

```

```

new
"    Abstract: No
    Requires:
    Effect: crea un objeto inicializado
    Modifies:
    Answer:
    Author: JosÃ© Parets
    Version: 1.0
    Date:January 14, 2002"

    ^super new initialize.!

```

```

new:anInteger
"    Abstract: No
    Requires:
    Effect: crea un objeto inicializado con saldo inicial anInteger
    Modifies:
    Answer:
    Author: JosÃ© Parets
    Version: 1.0
    Date:January 14, 2002"

    ^super new initialize: anInteger.! !

```

```

!ContabilidadMacana class methodsFor: 'ejemplos'!

```

ejemplo1

```
" Abstract: No
  Requires:
  Effect: ejemplo de uso de la clase con una sola contabilidad
  Modifies:
  Answer:
  Author:
  Version:
  Date:January 14, 2002"
```

```
| conta1|
ContabilidadMacana iniciaTotalSaldos.
conta1 := ContabilidadMacana new:100.
conta1 ingresaEnCapitulo: 'Bebidas' laCantidad: 1000.
conta1 ingresaEnCapitulo: 'Bebidas' laCantidad: 2000.
conta1 consultaAdeudo: 'Bebidas'.
conta1 consultaIngreso: 'Bebidas'.
conta1 adeudaEnCapitulo: 'Cubatas' laCantidad: 5000.
conta1 consultaAdeudo: 'Cubatas'.
conta1 saldo.
ContabilidadMacana totalSaldos.!
```

ejemplo2

```
" Abstract: No
  Requires:
  Effect: ejemplo de uso de la clase con dos contabilidades
  Modifies:
  Answer:
  Author:
  Version:
  Date:January 14, 2002"
```

```
| conta1 conta2|
ContabilidadMacana iniciaTotalSaldos.
conta1 := ContabilidadMacana new:100.
conta2 := ContabilidadMacana new:200.
conta1 ingresaEnCapitulo: 'Juegos' laCantidad: 1000.
conta1 ingresaEnCapitulo: 'Juegos' laCantidad: 2000.
conta2 ingresaEnCapitulo: 'Trabajo' laCantidad: 2000.
conta1 consultaAdeudo: 'Juegos'.
conta1 consultaIngreso: 'Juegos'.
conta1 adeudaEnCapitulo: 'Cubatas' laCantidad: 5000.
conta1 saldo.
ContabilidadMacana totalSaldos.!!
```

!ContabilidadMacana class methodsFor: 'variables de clase'!

iniciaTotalSaldos

```
" Abstract: No
  Requires:
  Effect: inicializa el total de saldos de todas las contabilidades.
  Modifies:
  Answer:
  Author: JosÃ© Parets
  Version: 1.0
  Date:January 14, 2002"
```

```
TotalSaldos := 0.!
```

totalSaldos

```
" Abstract: No
  Requires:
  Effect:
  Modifies:
  Answer: devuelve el total de saldos de todas las contabilidades
  Author: JosÃ© Parets
  Version: 1.0
  Date:January 14, 2002"
```

```
^ TotalSaldos.!!
```

5 Modificadores de Clase en JAVA

public: acceso desde todas las clases (por defecto sólo es accesible a las clases que estén dentro de su paquete *packaje*)

abstract: la clase no puede instanciarse y debe tener un método abstracto

final: no se puede heredar de la clase

6 Modificadores de variables (campos o datos miembro)

6.1 Modificadores de acceso (ámbito o visibilidad)

public: accesible desde todas las clases

protected: sólo es accesible por la propia clase y sus subclases

package: es el acceso por defecto. Accesible a las clases que hay dentro del paquete

private: accesible sólo para los métodos de la propia clase (no por las subclases)

6.2 Otros modificadores

static: indica que una variable es de clase, no de instancia. Podrá accederse a ella desde los métodos de la clase. También se podrá acceder desde el exterior (NombreClase.NombreVariableStatic) si su ámbito es *public* o *package*.

final: para variables con valor constante. Su valor no puede cambiar después de ser inicializada. Deben inicializarse antes de ser utilizadas, en caso contrario el compilador señala un error.

7 Modificadores de métodos (funciones miembro)

7.1 Modificadores de acceso (ámbito o visibilidad)

public: accesible desde todas las clases

protected: sólo es accesible por la propia clase y sus subclases

package: es el acceso por defecto. Accesible a las clases que hay dentro del paquete

private: accesible sólo para otros métodos de la propia clase (no por las subclases)

7.2 Otros modificadores

static: indica que una método es de clase, no de instancia. Podrá ser usado desde los métodos de instancia la clase. También se podrá usar desde el exterior (NombreClase.NombreMetodoStatic) si su ámbito es *public* o *package*.

final: el método no puede ser sobrescrito (reimplementado) en las subclases.

abstract: deben ser sobrescritos (reimplementados) por las subclases. Un clase con métodos *abstract* debe ser *abstract*.

8 Ejemplo de clase en JAVA: Carta

```
import java.io.*;

public class Carta {

    // VARIABLES DE INSTANCIA

    private String palo;
    private Integer numero;

    // MÉTODO DE INICIALIZACIÓN. CONSTRUCTOR

    Carta( String pal, Integer num) {
        palo = pal;
        numero = num;
    }

    // MÉTODO QUE MUESTRA EL PALO. CONSULTOR

    public String palo() {
        return palo;
    }

    // MÉTODO QUE MUESTRA EL NÚMERO. CONSULTOR

    public Integer numero() {
        return numero;
    }

    // MÉTODO QUE CAMBIA EL VALOR DEL PALO Y NUMERO DE UNA
    CARTA.

    private void ponerPaloNumero(String pal, Integer num) {
        palo= pal;
    }
}
```

```

        numero= num;
    }
}

```

// MÉTODO MAIN.

```
public static void main(String args[]) {
```

// Contiene sentencias para probar sus métodos

```
String p= new String("oros");;
```

```
Integer n= new Integer (7);;
```

```
String palillo;
```

```
Integer numerillo;
```

```
System.out.println("Ahora voy a crear una carta que es el 7 de oros");
```

```
Carta cartilla = new Carta(p,n);
```

```
System.out.println("Ahora voy a mostrar las variables de instancia");
```

```
System.out.println(cartilla.palo());
```

```
System.out.println(cartilla.numero());
```

```
System.out.println("Ahora voy a modificar las variables de instancia");
```

```
palillo= new String("bastos");
```

```
numerillo= new Integer(10);
```

```
cartilla.ponerPaloNumero(palillo, numerillo);
```

```
System.out.println("Los nuevos valores son:");
```

```
System.out.println(cartilla.palo());
```

```
System.out.println(cartilla.numero());
```

9 Ejemplo de clase en JAVA: POINT

```
class POINT {

    private int x,y;

    void scale (int factor){
        x= factor * x;
        y= factor * y;}

    void translate (int a, int b) {
        x=x+a;
        y=y+b; }

    int distance (POINT other){
        return (int)Math.sqrt ( (x-other.x)^2 + (y-other.y)^2 ) }

    POINT (int a, int b) {

        x= a;
        y= b; }
}
```

Uso de esta clase:

```
public static void main (String args[]) {

    POINT q = new POINT( 10,150);
    q.scale (50);
    System.out.println("La coordenada x del punto q es: " + q.x)
    POINT p = new POINT(10,200);
    System.out.println("La distancia a otro punto es de: " +
        q.distance(p));
}
```

10 Ejemplo de clase en JAVA: Cuenta

```

import java.util.*;

class Cuenta {

    private int disponible;
    private Hashtable ingresos, gastos;

    Cuenta(){
        disponible = 0;
        this.inicializar(); }

    Cuenta( int cantidad) {
        disponible = cantidad;
        this.inicializar(); }

    private void inicializar(){
        ingresos = new Hashtable();
        gastos = new Hashtable();}

    public int disponible() {
        return(disponible);}

    public void gastarEn( int cantidad, String motivo){
        Integer g= new Integer(this.totalGastadoEn(motivo)+cantidad);
        gastos.put(motivo, g);
        disponible = disponible - cantidad;}

    public void ingresarDe( int cantidad, String fuente){
        Integer i= new Integer(this.totalIngresadoDe(fuente)+cantidad);
        ingresos.put(fuente,i);
        //vale: put(objeto,objeto). No vale: put(objeto,int)
        disponible = disponible + cantidad;}

    public int totalGastadoEn( String motivo){
        if (gastos.containsKey(motivo))
            return(((Integer)gastos.get(motivo)).intValue());
            //conversión de Integer a int
        else return(0);}
}

```

```
public int totalIngresadoDe( String fuente){
    if (ingresos.containsKey(fuente))
        return(((Integer)ingresos.get(fuente)).intValue());
    else return(0);}

```

```
public static void main (String args[]){

```

```

    Cuenta cuen = new Cuenta();
    cuen.ingresarDe(3000, "Rentas");
    cuen.ingresarDe(2000, "Rentas");
    cuen.gastarEn(1000, "Comida");
    System.out.println("Gastos de comida " +
        cuen.totalGastadoEn("Comida"));
    System.out.println("Ingresos de rentas " +
        cuen.totalIngresadoDe("Rentas"));
    System.out.println("Disponibile " + cuen.disponible);}

```

```
}

```

11 Clases y objetos en los lenguajes con tipos

11.1 Concepto de tipo en los LOO

Los lenguajes orientados a objetos con tipos son lenguajes en los que convive la declaración de clases con la verificación fuerte de tipos. A nivel práctico ello supone que las variables se declaran estáticamente asociándoles un tipo, que restringe los posibles objetos a que la variable referenciará en tiempo de ejecución. Las funciones también tendrán un tipo de retorno.

El sentido de las declaraciones de tipo es el mismo que en los lenguajes tradicionales con extensiones justificadas por la herencia y la ligadura dinámica.

11.1.1 Utilidad

Como en el caso de los lenguajes procedimentales tradicionales la utilidad de los tipos reside en :

1. Posibilidad de verificar estáticamente las asignaciones de variables
2. Verificación de la correspondencia entre los parámetros formales y actuales de una llamada a procedimiento o función.

Todo ello redundará en un aumento de la fiabilidad de los programas y en la detección de errores en tiempo de compilación.

11.1.2 Regla de compatibilidad de tipos

La regla usual en la asignación de tipos es la que caracteriza la verificación fuerte de tipos, a saber:

Regla 1.- Dos variables X e Y son asignables si el tipo declarado para ambas variables es el mismo. Un parámetro formal X es sustituible por un parámetro formal Y si su tipo declarado es el mismo.

En general, esto obliga a que el tipo de X e Y tengan el mismo nombre:

```
TYPE uno = ARRAY [1..10] of INTEGER
      dos = ARRAY [1..10] of INTEGER
```

```
VAR X: uno
     Y: dos
```

En este ejemplo X e Y son incompatibles. Se trata de una compatibilidad de tipos por NOMBRE y no por estructura. Si esta misma regla se aplicará en los LOO nos encontraríamos con situaciones muy curiosas (utilizaremos una sintaxis independiente del lenguaje):

```

CLASS Uno
  Integer A ()
    {write ('Soy Uno -- A')}
  Integer B ()
    {write ('Soy Uno -- B')}
  Integer C ()
  {
    self A().
    self B() }
    
```

```

CLASS Dos: inherit from Uno
  integer A()
    { write ('Soy Dos - A') }
  integer B()
    { write ('Soy Dos -B') }
  integer D()
    {...}
    
```

Observando el ejemplo veamos lo que sucedería si la regla 1 fuera la aplicada:

```

main {
  Uno X; Dos Y
  ...
  X := Y // No se pueden asignar dado que no se cumple la regla 1}
    
```

Dado que X e Y no se pueden asignar sería imposible que una variable tuviera un tipo dinámico diferente del tipo estático, con lo que el polimorfismo no tendría ninguna utilidad.

Para permitir el polimorfismo se amplía la regla anterior siendo válida, en los LOO con tipos la siguiente:

Regla 2.- Dada una variable X declarada del tipo de la clase A y una variable Y del tipo de la clase B, Y puede ser asignada a X si B es subclase de A. Lo mismo sucede si X es un parámetro formal e Y es un parámetro actual.

En el ejemplo anterior $X := Y$ es una asignación válida. La pregunta inmediata es porqué $Y := X$ no lo es. La respuesta también es inmediata: si esto fuera posible podría darse la siguiente situación:

```
Y := X
Y.D
```

El compilador permitiría la asignación y, puesto que Y tiene como tipo la clase Dos, se supondría que Y.D es un mensaje válido. Sin embargo, como en tiempo de ejecución el objeto al que haría referencia Y es de la clase Uno, este método no se podría ejecutar.

11.2 Ligadura mixta y verificación de tipos

En el ejemplo 1 hemos supuesto la existencia de ligadura dinámica, es decir, que la resolución del mensaje a un objeto dependerá de la clase a la que pertenezca. Si consideramos la regla 2 y el ejemplo 1:

```
X := Y.
X.C
```

Cuando se ejecute este código, el objeto al que hace referencia X es un objeto de la clase Dos. Ello implica que la resolución del mensaje C supone la ejecución de los métodos A y B de la clase Dos. En este caso el resultado de X.C es

```
'Soy Dos - A'
'Soy Dos -B'
```

En algunos lenguajes (C++, Pascal) la ligadura es, por defecto, estática. Ello implica que el mismo código anterior se comporta de forma distinta: aunque X hace referencia a un objeto de la clase Dos, el método C supone la ejecución de A y B de la clase Uno, es decir:

```
'Soy Uno - A'
'Soy Uno -B'
```

Por tanto, y aunque la regla de verificación de tipos es la misma, el comportamiento final del código dependerá del tipo de ligadura utilizado.

En estos mismos lenguajes (C++, Pascal) para que la ligadura sea dinámica es necesario que el programador lo defina explícitamente para cada método utilizando el término VIRTUAL.

Esto supone que es el programador el que decide el tipo de ligadura que se debe utilizar. En principio, parece que esto da mayor flexibilidad a la programación, sin embargo, el efecto es el contrario. Cuando el programador diseña una clase, en general, no sabe qué subclases va a tener ni cómo va a redefinir los métodos en estas subclases. Si observamos el ejemplo nos daremos cuenta de que, con ligadura estática, la redefinición de los métodos A y B no tiene ningún efecto sobre el método C, es decir, A y B de la clase Dos sólo se utilizarán cuando las variables que reciben el mensaje sean de esta clase.

```
Y.A
Y.B
```

En definitiva, la regla 2 de asignación de tipos tiene toda su potencia cuando se utiliza polimorfismo dinámico, es decir, cuando la ligadura es dinámica. Si la ligadura fuera siempre estática nos bastaría con la regla 1, puesto que aunque es posible la asignación el efecto de la ejecución es siempre el mismo y no depende del tipo dinámico de la variable.

11.3 Características generales de los LOO con tipos

Las siguientes características son comunes a los lenguajes con tipos más utilizados: C++, Pascal, Eiffel y JAVA. Sin embargo, no todas ellas obedecen a la presencia de verificación de tipos. Sólo 3.1 y 3.4 tienen relación directa con la presencia de tipos. 3.2 y 3.3 coinciden en ellos, pero obedecen a decisiones de diseño del lenguaje no relacionadas con los tipos.

11.3.1 Tipos básicos y objetos

Todos los lenguajes con tipos diferencian entre tipos básicos y clases. Los tipos básicos son una serie de tipos predefinidos para cada lenguaje (enteros, reales, caracteres,...) y el resto de estructuras de datos se definen como clases. Cada clase podrá ser utilizada como un tipo, es decir, cada variable se define de una clase que se constituye en su tipo.

Cada lenguaje incluye una serie de tipos básicos distintos. Por ejemplo:

Eiffel: sólo incluye Integer, Boolean, Character, Real. No pueden definirse nuevos tipos que no sean clases.

JAVA: incluye los tipos básicos boolean, char, double, int, long, short. Al igual que en Eiffel no se pueden definir nuevos tipos que no sean clases.

C++, Pascal: incluyen los tipos básicos de estos lenguajes así como los tipos estructurados (struc, array, record). Se pueden construir nuevos tipos estructurados QUE NO SEAN CLASES, lo que implica que se puede trabajar con una filosofía no orientada a objetos.

11.3.2 Features o miembros

En general se denominan de esta forma a las variables de instancia y los métodos de una clase. Las variables de instancia, los parámetros y las funciones se declaran de uno de los tipos permitidos en el lenguaje, tanto tipos básicos como clases.

Eiffel, Pascal: los métodos pueden ser funciones (devuelven un valor) o procedimientos.

C+, JAVA: sólo utilizan funciones

11.3.3 Mensajes

La sintaxis es prácticamente la misma en todos ellos:

NombreVariable.nombreMetodo [(parámetros)]

En C++ la sintaxis es distinta para objetos en el Heap y variables en el stack (consultar los manuales del lenguaje)

11.3.4 Creación e inicialización de objetos

En todos ellos las variables que referenciarán objetos se declaran de forma estática con un tipo:

Eiffel: `b: Point`
 C++, JAVA: `Point b`
 PASCAL: `VAR b: Point`

El significado de estas declaraciones dependerá de que se trate de objetos en el Heap o en el Stack y, por tanto, la creación de objetos se realizará de forma estática o dinámica.

Eiffel, JAVA: todas las declaraciones serán referencias a objetos en el HEAP, por tanto la creación de objetos se realizará en tiempo de ejecución con una operación específica del lenguaje:

create en el caso de Eiffel, *new* en JAVA.

C++, Pascal: las declaraciones de variables para objetos en el heap o el stack serán diferentes:

STACK:	<code>b: Point</code>	<code>Pascal</code>
	<code>Point b</code>	<code>C++</code>

Los objetos en el stack son creados por el compilador.

HEAP:	<code>b: ^Point</code>	<code>Pascal</code>
	<code>Point * b</code>	<code>C++</code>

Los objetos en el heap se crean en tiempo de ejecución con NEW.

11.3.5 Objetos en el Heap y objetos en el Stack

Se estudiaron las diferencias en la lección 1.2. En el punto anterior se han comentado las diferencias en la creación de objetos.

En la página siguiente tenéis una tabla comparativa de la forma de creación e inicialización de objetos en distintos lenguajes de programación, tanto tradicionales como orientados a objetos. Lo importante no es aprendérsela, sino entender sus contenidos y observar cómo los lenguajes más recientes tienden a simplificar y unificar estos mecanismos mediante creación e inicialización dinámicas.

Lenguajes	CREACIÓN		INICIALIZACION	
	ESTATICA (compilador-stack)	DINAMICA (ejecución-heap)	ESTATICA (compilador)	DINAMICA (ejecución)
Fortran	Declaración	-----	En la declaración	Asignación
Pascal	Declaración <i>var i:integer;</i> <i>ptr: ^integer</i>	New <i>new (ptr)</i>	-----	Asignación (stack y heap)
C	Declaración <i>int i;</i> <i>int* ptr</i>	Malloc <i>ptr = malloc()</i>	-----	Asignación (stack y heap)
CLU	Declar. punteros vacíos con tipo <i>x: int</i> <i>a,b: array[int]</i>	Oper. de creación para cada TAD <i>a:= array[int] new()</i>	-----	Asignación incluida en creación
C++	Declaración <i>int i;</i> <i>TClass a</i> <i>TClass* b</i>	New (sobrecargable) <i>new b</i>	Constructor (automático) TClass a (args)	Constructor (automático) <i>new b (args)</i>
TPascal OO	Declaración <i>int i;</i> <i>TClass a</i> <i>TClass* b</i>	New (no sobrecargable) <i>b:= new()</i>	-----	Constructor stack - manual heap - auto
Smalltalk	-----	Método de clase	-----	Mét. de instancia
JAVA	Declaración <i>int a</i> <i>Integer a;</i>	New (no sobrecargable) <i>a = new Integer (7)</i>	-----	Asignación tipos básicos. Constructor (automático, new)
Eiffel	Declar. punteros vacíos con tipo <i>a: integer</i> <i>b: array[integer]</i>	Create (redefinible) <i>b.create()</i>	-----	Asignación tipos básicos. Asignación en create

12 Polimorfismo en JAVA (1)

```

class Persona{
    private String nombre;
    protected String cancion;

    public void canta(){System.out.println("la la la");}

} //End Persona

class Ladron extends Persona{
    private int ganancias;

    public void canta(){System.out.println("Yo no he sido");}
    public void roba(int n){ ganancias:= ganancias + n; }

} //End Ladron

class Juez extends Persona {
    public void canta() {System.out.println("Yo no robo -mucho-");}

} //End Juez

public class Poli5 {

// ejemplo de polimorfismo de mensaje canta.
// la variable fulanito es polimórfica

public static void main(String args[])
    {
        Persona fulanito = new Persona();
        Ladron ladroncito = new Ladron();
        Juez juececito = new Juez();
        fulanito.canta();    // canta una persona
        fulanito = ladroncito;
        fulanito.canta();    // canta un ladrón
        fulanito.roba();    //error de compilación: fulanito es Persona. ¿Soluciones?
        fulanito = juececito;
        fulanito.canta ();    // canta un juez
    }
} // End Poli5

```

13 Polimorfismo en JAVA (2)

Discutir lo que sucede en las líneas señaladas con /**

```

class Transporte {
    llevarCosas() {...}
}
class Coche extends Transporte {
    correr() {...}
}
class Barco extends Transporte {
    navegar() {...}
}
class Pesquero extends Barco {
    pescar() {...}
    navegar() {...}
}

class PruebaLigadura {

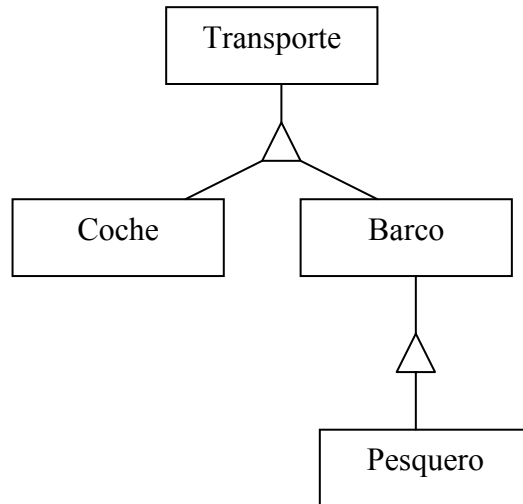
    public static void main ( String args[]) {

        Ccoche c1 = new Coche();
        c1.llevarCosas();
        c1.correr();
        Coche c2;
        c2=c1;

        Barco b = new Barco();
        b.llevarCosas();
        b.navegar();
        b.correr();    /**
        b=c1;          /**

        Pesquero p;
        p= new Pesquero();
        p.navegar();
        p.pescar();
        p.llevarCosas();
        p=b;          /**
        b=p;
        b.navegar();
        b.pescar(); /**

        Vector v= new Vector();
        v.addElement(c1);
        v.addElement(p);
        (v.elementAt(1).navegar()); /**
        (v.elementAt(1).llevarCosas()); /**
    }
}
    
```



14 Polimorfismo en Java (3)

```

import java.io.*;

/*POLIMORFISMO: SE DESCONOCE LA CLASE DEL OBJETO RECEPTOR EN TIEMPO
DE COMPILACION, POR TANTO, NO SE SABE QUE METODO SE LIGARÁ AL OBJETO
Y CUÁL SE EJECUTARA*/

class Persona{

    void canta(){System.out.println("la la la");}
}

class Ladron extends Persona{

    void canta(){System.out.println("Yo no he sido");}
}

public class Prue4{

    public static void main(String args[])
        throws java.io.IOException
    {

        Persona uno; /* declaración de variable */

        System.out.println("Introduzca 1 para el ladron y 2 en otro caso");

        int dato = System.in.read();

        if ((char)dato=='1')

            uno = new Ladron();
        else
            uno = new Persona();

        System.out.println("El objeto creado es de la clase "+ (uno.getClass().getName()));

        uno.canta(); /*El compilador comprueba que el método canta() esté definido en la
        clase Persona (ver "declaración estática" de la variable uno. Hasta la
        ejecución no se sabe qué método se ligará a este mensaje*/
    }

}
}

```

15 Algunas colecciones (antiguas) de JAVA

Las colecciones que aquí se muestran están en desuso y han sido sustituidas por otras. Se muestran a modo de ejemplo para ver su utilización. En un tema posterior se estudiarán las colecciones presentes en JAVA 2. Para usarlas hay que incluir al principio del programa:
import java.util.;*

- Vector

```

Vector vehiculos = new Vector();
Coche c = new Coche(111);
Moto m = new Moto(222);
vehiculos.addElement(c);
vehiculos.addElement(m);
vehiculos.size();
vehiculos.removeElement(m);
vehiculos.elementAt(1);
System.out.println("Matricula: " + ((Vehiculo)vehiculos.elementAt(1)).matricula() );

/* Se supone que existe una clase Vehículo con un método matricula() que devuelve el
valor de la matricula para un Vehículo. Coche y Moto son subclases de Vehículo.*/

/*Vehículo también podría ser una interface que implementan Coche y Moto */

/*En la última instrucción es necesario un "casting" para indicar al compilador que
debe buscar el método matrícula() en la clase/interface Vehículo, y no en la clase
Object como haría por defecto */

```

- Stack

```

Stack pila = new Stack();
pila.push(c);
pila.push("dos");
if (pila.empty()!)
    (String)pila.pop();

```

- HashTable (equivalente a Dictionary en Smalltalk)

```

Hashtable sellos = new Hashtable();
Sello sello35 = new Sello(35);
Sello sello100 = new Sello(100);
sellos.put(35, sello35);
sellos.put(100, sello100);
sellos.contains(sello35);
sellos.get(35);

```

16 Ejemplo de Applet

Son programas que se ejecutan dentro de una página web.

A modo de ilustración, se muestra un ejemplo de un applet java y de la página que lo utiliza.

```
// applet que muestra un mensaje, unas figuras y gestiona botones

import java.awt.*;
import java.applet.Applet;

public class AppletComp extends Applet{

    String msg;

    public void init() { //Colocacion de botones
        Button si= new Button("SI");
        Button no= new Button("NO");
        add(si);
        add(no);
    }

    public boolean action(Event evtObj, Object arg){ //Gestion de eventos
        if (evtObj.target instanceof Button) {
            if (arg.equals("SI"))
                msg="Ha pulsado SI";
            else msg="Ha pulsado NO";
            repaint();
            return true;
        }
        return false;
    }

    public void paint( Graphics g){ //colocacion de otros elementos
        g.drawString("Este applet hace varias cosas",20,200);
        g.fillRect(200,150,230,200);
        g.drawOval(190,50,90,80);
        g.drawString(msg,60,100);
    }
}
```

El fuente del fichero *html* que lo invoca es:

```
<HTML>
<HEAD>
<TITLE> Este applet escribe texto, dibuja figuras y contiene botones
</TITLE>
</HEAD>
<BODY>
<APPLET CODE ="AppletComp.class" WIDTH=400 HEIGHT=300>
</APPLET>
</BODY>
</HTML>
```

17 Cuestionario sobre JAVA

1. Diferencias y semejanzas entre el tipo int y la clase Integer.
2. ¿Porqué cree que se mantienen los tipos básicos y las clases en el lenguaje?
3. ¿Para qué sirven las palabras public, protected y private, ... en la definición de una clase o método?
4. ¿Cómo se declaran métodos de clase?
5. ¿Desde dónde se puede cambiar el valor de una variable de clase?
6. ¿Cómo se distinguen las variables de clase de las variables de instancia?
7. El constructor de una clase en Java realmente puede dar valores a las variables de instancia de un objeto, luego ¿es un método de clase o de instancia?
8. Además de los tipos primitivos que ofrece el lenguaje, ¿se pueden crear nuevos tipos de datos en Java? ¿Cómo?
9. ¿Se pueden añadir métodos a las clases ya existentes en Java? ¿Por qué?
10. ¿Y variables?
11. ¿Y en Smalltalk?