Fundamentos de Java para la Concurrencia

Tema 2 - Programación Concurrente y de Tiempo Real

Antonio J. Tomeu[1] Manuel Francisco[2]

[1]Departamento de Ingeniería Informática
Universidad de Cádiz

[2]Alumno colaborador de la asignatura
Universidad de Cádiz

PCTR, 2015

Contenido

1. La Plataforma J2SE.
2. Elementos Básicos de Programación.
3. Nociones de Orientación a Objetos.
4. E/S y Manejo Básico de Expresiones.
5. Otros Conceptos Sobre POO: Herencia, Sobrecarga, Interfaces...

Diferencias Principales Entre Java y ANSI C/C++

Lo que Java no soporta/no tiene

- Punteros y aritmética de punteros.
- Necesidad de compilarse en diferentes plataformas.
- Preprocesador, `includes`, o archivos de encabezado.
- Sobrecarga de operadores.
- Necesidad de liberar memoria manualmente.
- Herencia múltiple o clases de base múltiple.
- Campos de bits, `struct`, `union` o `typedef`.
- `inline`, `register`, `friend`, `template`.

Diferencias Principales Entre Java y ANSI C/C++

Lo que Java no soporta/no tiene

- Plantillas.
- Variables de referencia.
- Operador `sizeof`.
- Declaraciones `extern`.
- Superclases `protected` o `private`.
- Conversiones de tipo definidas por el usuario.
- Copia implícita de objetos.
- Notación de asignación explícita.

Diferencias Principales Entre java y ANSI C/C++

Aspectos de Java no presentes en C/C++

- Es interpretado cuando se ejecuta.
- Recolección automática de basura.
- Soporta applets Web.
- Soporta programación GUI independiente de la plataforma.
- Soporta programación controlada por eventos.
- Soporta multiprocesamiento y métodos sincronizados.
- Soporta trabajo en red.

Diferencias Principales Entre java y ANSI C/C++

Aspectos de Java no presentes en C/C++

- Toda la creación de objetos es mediante new.
- Programas y caracteres en Unicode de 16 bits.
- Tipo primitivos `boolean` y `byte`.
- Cadenas y arrays como objetos.
- Variables y parámetros de objeto como referencias.
- Herencia simple.
- Otros.

Diferencias Principales Entre java y ANSI C/C++

Aspectos de Java que trabajan de otro modo que en C/C++

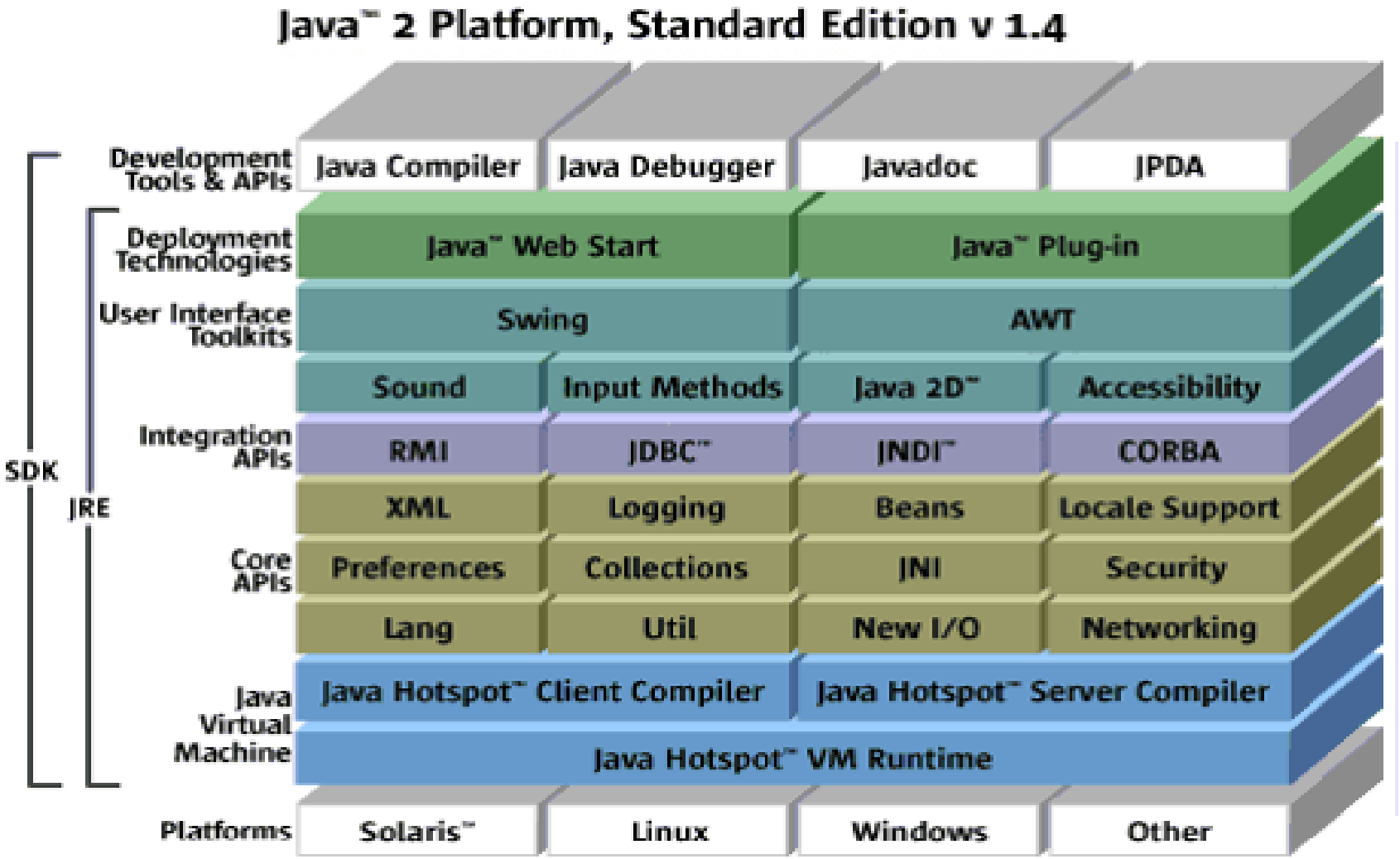
- Java utiliza true-false en lugar de cero-no cero para operaciones booleanas.
- Java no tiene declaracion unsigned. Todos los tipos enteros tienen signo.
- Java soporta los operadores de desplazamiento >> y >>>.
- Java no tiene tipo de devolución predeterminado para métodos.
- El operador % funciona en Java para enteros y punto flotante.

Diferencias Principales Entre java y ANSI C/C++

Aspectos de Java que trabajan de otro modo que en C/C++

- La resolución de una llamada a un método sobrecargado difiere profundamente en Java y en C++.
- Las clases de Java utilizan el método `finalize` en lugar de destructores.
- Las clases de Java se organizan en paquetes con un nombre. Cada paquete puede contener uno o más archivos de código fuente.
- Otros.

La Plataforma J2SE



Programa Hola Mundo

```
1  class Hola_mundo
2  {
3    public static void main (String[] args)
4    {
5      System.out.println("HOLA MUNDO");
6    }
7  }
```

Estructura de un Programa en Java

```
1  class Media //nombre de la clase
2  {
3    public static void main (String[] args) //metodo
4    {
5      int i=11, j=20;
6      double a=(i+j)/2.0;
7      System.out.println("La media es"+a);
8    }
9  }
```

Estructura de una Clase en Java

```
1  class Nombre_de_la_clase
2  {
3    miembros
4  }
```

- Un miembro de la clase puede ser un campo (datos) o un método.
- Un programa de Java siempre incluye el método `main`.
- Una clase pública es declarada anteponiendo a su declaración el modificador `public`.
- Si una clase no es pública, sólo será utilizable dentro de su propio paquete.
- Todas las partes de un programa pueden acceder a aquellos miembros declarados como `public`. En concreto, el método `main` es público para que el intérprete de Java lo pueda invocar.

Métodos I

- Un método es un procedimiento de cálculo definido en una clase.
- Un método contiene instrucciones y variables
- Cada método contiene una cabecera y un cuerpo
- La cabecera especifica los parámetros formales
- Los parámetros son siempre pasados por valor
- Cuando el parámetro es una referencia a objeto, lo que se pasa por valor es la referencia, y por tanto los cambios realizados en el objeto se ven reflejados en el método que realizó la invocación.

Métodos II

- El cuerpo incluye una secuencia de declaraciones e instrucciones encerradas entre `{}`.
- Los métodos pueden ser de clase (lo habitual) o de instancia.
- Un método `static` es común a todas las instancias de la clase.
- La estructura general de un método es:

```
1  Tipo_devuelto Nombre (tipo_1 arg_1, tipo_2 arg_2,...)
2  {
3      declaraciones e instrucciones
4  }
```

Instrucciones

- El cuerpo de un método contiene instrucciones
- Las instrucciones de java pueden ser simples y compuestas
- Una instrucción simple termina en punto y coma: ;
- Una instrucción compleja comprende a varias simples encerradas entre llaves; puede emplearse en cualquier lugar donde se emplee una simple. Una instrucción compuesta también es llamada bloque
- En general, debe recordarse que:
  1. Una declaración siempre termina en ;
  2. Una instrucción simple siempre termina en ;
  3. No hay ; después de una instrucción compuesta
  4. No hay ; después de una definición de clase;

La Instruccion while I

- Se define la clase Factorial que contiene dos métodos: factorial y main.

```
1  public static int factorial (int n)
2  {
3      int acum=1;
4      while (n>1)
5      {
6          acum=acum*n;
7          n=n-1;
8      }
9      return acum;
10 }
```

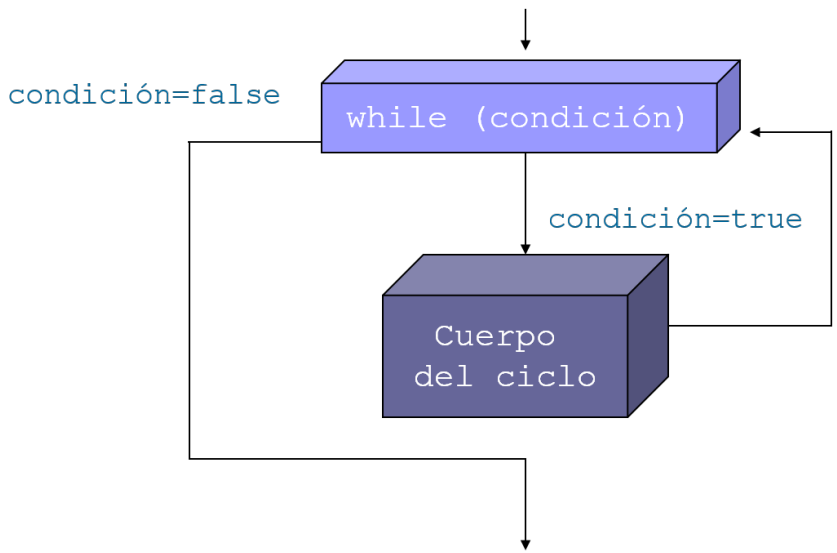
La Instruccion while II

- En general, la estructura sintáctica de la instrucción es:

```
1  while (condicion)
2  {
3      bloque de sentencias
4  }
```

La Instruccion while III

- La instrucción `while` es controlada por una condición ($n > 1$) que repite el cuerpo de la misma mientras esta no es `false`.
- La instrucción `while` prueba el valor de la condición. Si es `true`, ejecuta el cuerpo. Si no lo es, termina el ciclo y pasa el control a la siguiente instrucción. La variable que controla el ciclo es llamada variable de control.



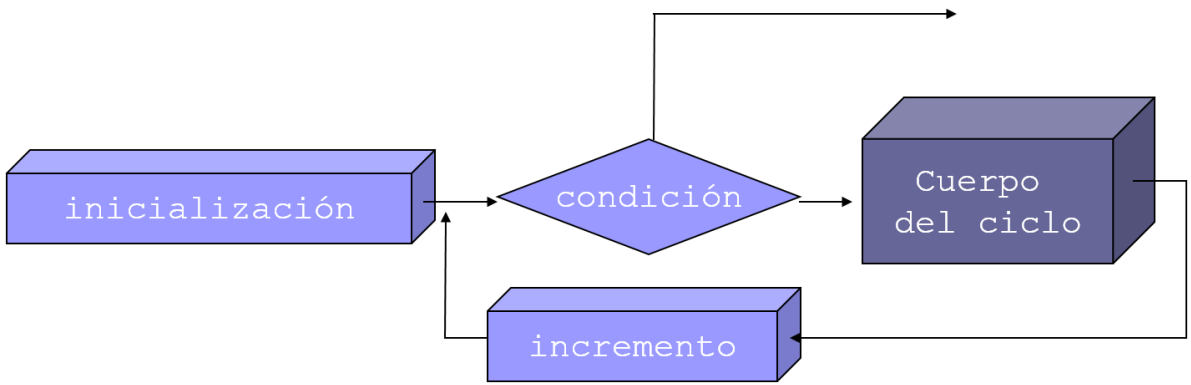
Código 1: codigos_t2/Factorial.java

```java
//Factorial.java
class Factorial //Enumera el factorial de n para n=0,1,...,12
{
  public static int factorial(int n) //metodo factorial
  {
    int acum = 1;
    while (n > 1) {
      acum = acum * n;
      n = n - 1;
    }
    return acum;
  }
  public static void main(String[] args) //metodo main
  {
    int n = 1;
    while (n < 13) {
      System.out.print(n + " != ");
      System.out.println(factorial(n));
      n = n + 1;
    }
  }
}
```

La Instrucción for I

- Es una forma especializada de `while`, donde el cuerpo del ciclo se ejecuta un número fijo de veces, siempre que la condición de control sea `true`.
- Si la condición es `false` al principio, el cuerpo no se ejecuta.
- Su forma general es la siguiente:

```
1  for (inic; condicion de control; incremento)
2  {
3    cuerpo del ciclo
4  }
```



La Instrucción for II

```
1  static int potencia (int x, int y)
2  {
3      int iter, acum=1;
4      for(iter=1; iter<=y; iter=iter+1)
5      acum=acum*x;
6      return acum;
7  }
```

```
1  for (;;){cuerpo}     //es un ciclo infinito
2  for (control) {}     //cuerpo de ciclo vacio
3  for (control);       //cuerpo de ciclo con instruccion nula
```

- El incremento puede ser representado (como en C).

```
1  iter++
2  ++iter
3  iter--
4  --iter
```

Código 2: codigos_t2/Fibonacci.java

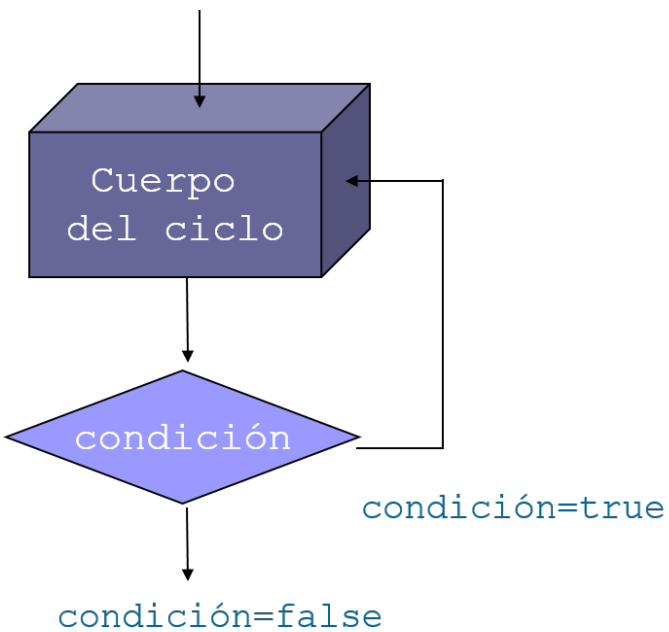
```
1  class Fibonacci
2  {
3    public static int Fibonac (int n)
4    {
5      int s1, s2, s3, i;
6      if(n==0) return 0;
7      if(n==1) return 1;
8      s1=0;
9      s2=1;
10     s3=0;
11
12     for(i=1;i<=n;i++) //Ejemplo de bucle for con bloque de
           sentencias
13     {
14       s3=s1+s2;
15       s1=s2;
16       s2=s3;
17     }
18
19     return s3;
20   }
21
22
```

```
23      public static void main (String[] args)
24      {
25        int n;
26        n=0;
27        while (n<25) //Ejemplo de bucle while con llamada a metodo
28        {
29          System.out.println("el termino "+n+" es "+Fibonac(n));
30          n++;
31        }
32      }
33  }
```

La Instrucción do-while I

- Es similar a la instrucción `while`, pero garantiza que el cuerpo del bucle se ejecuta al menos una vez (con `while` podía no ejecutarse nunca).
- La forma general de la instrucción es:

```
1  do {cuerpo del bucle} while (condicion);
```




```
1  int n=1;
2
3  do
4  {
5       System.out.println(n);
6       n++;
7  } while (n<10);
```

La Instrucción if I

- Es una instrucción de bifurcación condicional, y por tanto permite modificar la dirección del flujo de ejecución de una lista de instrucciones.
- Su forma simple es la siguiente:

```
1  if (condicion)
2      instruccion uno;
3  else
4      instruccion dos;
```

La Instrucción if II

- La instrucción prueba la condición; si es `true`, ejecuta la instrucción uno mientras que en caso contrario, ejecuta la instrucción dos.
- Naturalmente, ambas instrucciones pueden ser reemplazadas por bloques de instrucciones entre llaves, y la parte `else` puede omitirse si es necesario.

```
1  int compara (int a, int b)
2  {
3      if (a>b)
4          return(1);
5      else
6          if (a<b)
7              return(-1)
8          else
9              return (0);
10 }
```

Código 3: codigos_t2/Potencia.java

```
1  //Clase para calcular la potencia de un entero.
2  class Potencia {
3    public static int potencia(int x, int y) {
4      int i;
5      int p = 1;
6
7      if (y == 0) return 1;
8      else for (i = 1; i <= y; i++) {
9        p = p * x;
10     }
11     return p;
12   }
13
14   public static void main(String[] args) {
15     int Base = 2;
16     int Exponente = 4;
17
18     System.out.println(Base + " elevado a " + Exponente + " =
                          " + potencia(Base, Exponente));
19   }
20 }
```

Tipos de datos

- Java no utiliza el código ASCII de 7 bits como la mayoría de lenguajes.
- En su lugar, utiliza Unicode, estándar internacional que representa la información mediante 16 bits, lo que permite representar caracteres en prácticamente todos los idiomas importantes.
- Si se da como entrada a Java un fuente en ASCII, este lo traduce previamente.

Tipo primitivo	Descripción de los objetos del tipo
`boolean`	`true` o `false`
`char`	Carácter Unicode de 16 bits
`byte`	Entero con signo de 8 bits
`short`	Entero con signo de 16 bits
`int`	Entero con signo de 32 bits
`long`	Entero con signo de 64 bits
`float`	Coma flotante de 32 bits
`double`	Coma flotante de 64 bits

Operadores Lógicos y Relacionales

Operador Lógico	Significado
&&	AND lógico
||	OR lógico
!	NOT lógico
∧	XOR lógico

Operador Relacional	Significado
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual que
! =	Distinto de

Operador Aritmético	Significado
+	Suma
−	Resta
*	Producto
/	Cociente
%	Resto

- Precedencia habitual.
- Forman expresiones.
- Paréntesis para precedencia.

Expresiones I

- Los distintos operadores se combinan para formar expresiones relacionales (evalúan a datos boolean), aritméticas (evalúan a datos de tipos numéricos) o lógicas (evalúan también a boolean).
- Son expresiones relacionales las siguientes:

```
1  a==b; 25>=326; 0!=1.
```

- Son expresiones aritméticas las siguientes:

```
1  a+b/c, a/(3.4+b)-3*c, i/4, a%b, x*x*x, ++i, j++, --i, j--
```

- Son expresiones lógicas las siguientes:

```
1  a==b && a!=b, a==true && b==false, a>1 || a<-1
2  a|| !b && c, str1==str2
```

Expresiones II

- Como vemos, es habitual ligar variar expresiones relaciones mediante operadores lógicos para formar una expresión lógica compleja.
- Cualquier expresión lógica o relacional puede servir como condición de control de estructuras de bifurcación o iteración.

Instrucciones break-continue

- La instrucción break suele utilizarse para salir de una iteración de forma incondicional.
- Al ejecutarse break, el control se transfiere de inmediato a la primera instrucción a continuación del bloque o la iteración actual. El siguiente código muestra un ejemplo.

```
1  static boolean monotono (int [] a)
2  {
3    int n = a.length, i:
4    for (i = 0; i < n-1; i++)
5      if (a[i+1]<a[i]) break;
6    if (i==n-1) return (true); //secuencia monotona
7    for (i = 0; i < n-1; i++)
8      if (a[i+1]>a[i]) return (false); //secuencia no monotona
9  }
```

- La instrucción continue aborta la iteración actual e inicia la siguiente iteración.
- Dentro de un lazo while o do-while, esto significa transferir el control de inmediato a la parte de prueba de la condición.

Opción Múltiple

- La instrucción switch proporciona una manera muy cómoda de elegir una de entre un conjunto de opciones predefinidas. Su sintaxis es la siguiente:

```
1       switch (expresion de tipo int)
2       {
3         case valor_1: instrucciones
4         case valor_2: instrucciones
5         ..........................
6         case valor_n: instrucciones
7         default: instrucciones
8       }
```

- La instrucción evalúa la expresión de cambio y compara su valor con todas las etiquetas case. El control se transfiere a aquella etiqueta cuyo valor coincida con el de la expresión o a default si no coincide ninguna.
- Todas las etiquetas case deben ser distintas.

Entrada/Salida de Carácter

- Todo programa Java tiene asociados tres flujos estándar de E/S.
- La entrada estándar lee el teclado y se representa mediante el objeto de E/S System.in.
- La llamada `System.in.read()` devuelve un byte (entre 0 y 255) como un tipo de dato int o -1.
- La salida estándar escribe en la pantalla. Se representa con el objeto `System.out`.
- El método `System.out.write(int c)` da salida al byte más bajo de c a la pantalla.
- El método `System.out.println(str)` da salida a la cadena str más una terminación de línea dependiente de la plataforma.
- El método `System.out.print(str)` actúa igual sin terminación de línea.
- El flujo de error estándar es igual al de salida.

Código 4: codigos_t2/Minuscula.java

```
1  import java.io.*; //importa clases necesarias del paquete
       java.io
2  class Minuscula {
3    public static void main(String[] args)
4    throws IOException //indica que el metodo puede dar error E/S
5    {
6      int i;
7      while ((i = System. in .read()) != -1) {
8        i = Character.toLowerCase((char) i);
9        System.out.write(i);
10     }
11   }
12 }
```

Entrada con la Clase Scanner

- Es una alternativa cómoda para hacer entrada de datos.
- Decora con una instancia de la clase Scanner al objeto System.in
- Proporciona un API de uso inmediato de lectura de tipos básicos

Tipo	Método
byte	nextByte();
short	nextShort();
int	nextInt();
long	nextLong();
float	nextFloat();
double	nextDouble();
boolean	nextBoolean();

La Unidad de Biblioteca: Paquetes I

- Un paquete es una colección de clases que deseamos utilizar
- Para indicar que se van a utilizar se utiliza la claúsula

```
1  import nombre del paquete;
```

- Si se desea importar una única clase de un paquete concreto

```
1  import nombre del paquete.nombre de la clase;
```

- El concepto de paquete proporciona una gestión del espacio de nombres
- El programador puede definir sus propias bibliotecas de clases mediante el uso de package nombre_paquete;

La Unidad de Biblioteca: Paquetes II

```
1  package mipaquete;
2  public class miclase1{...} //clases del paquete
3  public class miclase2{...}
4
5
6  import mipaquete; //si se desean usar las clase de mipaquete
7  ...
8  miclase1 dato = new miclase1(); //uso de mi clase1...
```

Generando Documentación: javadoc

- Java intercala la documentación del programa dentro del mismo.
- Facilita de este forma el mantenimiento de la documentación.
- La herramienta para generar la documentación es `javadoc`.
- Su salida es un archivo `html` que documenta el código.
- Las etiquetas más útiles en la documentación son:
  - `@version`
  - `@author`
  - `@param`
  - `@return`
  - `@throws`
  - `@deprecated`
- Si queremos referenciar a otras clases se emplea `@see`.
  - `@see nombre_clase` genera un enlace a la documentación de la clase indicada.

Código 5: codigos_t2/Fecha.java

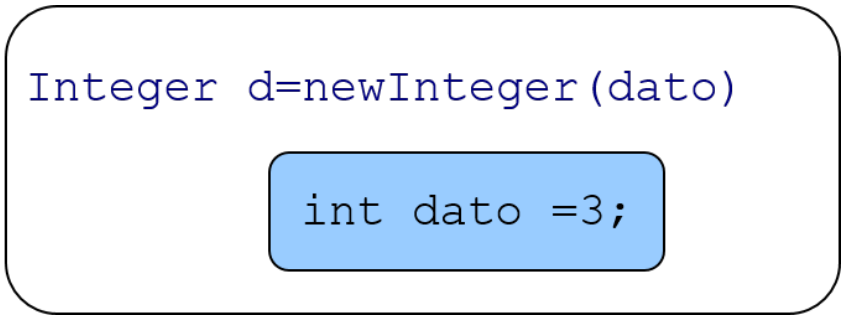
```java
import java.util.*;

/**
 * Ejemplo de documentacion de un codigo
 * @author Antonio Tomeu
 * @version 1.0
 */
public class Fecha {

  /**
   * Como es habitual, el metodo main puede recibir como
   * argumento un array de cadenas de caracteres
   * @param args array de cadenas de caracteres
   * @return no retorna nada
   * @exception no se generaran excepciones
   */
  public static void main(String[] args) {
    System.out.println("Hola, hoy es: ");
    System.out.println(new Date());
  }
}
```

Clases de Envoltura

- Permiten considerar campos de tipos base del lenguaje como objetos.
- Clases Integer, Byte, etc.



Clases y Programación Orientada a Objetos I

- La construcción clase (`class`) de Java soporta abstracción y encapsulación.
- Un objeto contiene datos y operaciones que se ocultan al exterior.
- El comportamiento externo de un objeto se conoce a través de un contrato de interfaz entre el objeto y sus clientes.
- Una clase describe la estructura de un objeto y sirve de base para construirlo.
- Un objeto concreto es una instancia de una clase.
- Toda clase tiene un nombre, y define a los miembros que pertenecen a ella; estos pueden ser campo-satributos (datos) o métodos (funciones).

Clases y Programación Orientada a Objetos II

- Una vez declarada una clase, su nombre es un nuevo tipo de dato y se utiliza para declarar variables de ese tipo o crear objetos nuevos.
- La estructura general de una clase es:

```
1  class Nombre de la clase
2  {
3      cuerpo de la clase
4  }
```

- El cuerpo define a los campos y a los métodos.
- Estos pueden ser públicos (accesibles por cualquier código con acceso a la clase), privados (accesibles sólo por métodos de la clase) o de paquete.

Clases y Programación Orientada a Objetos III

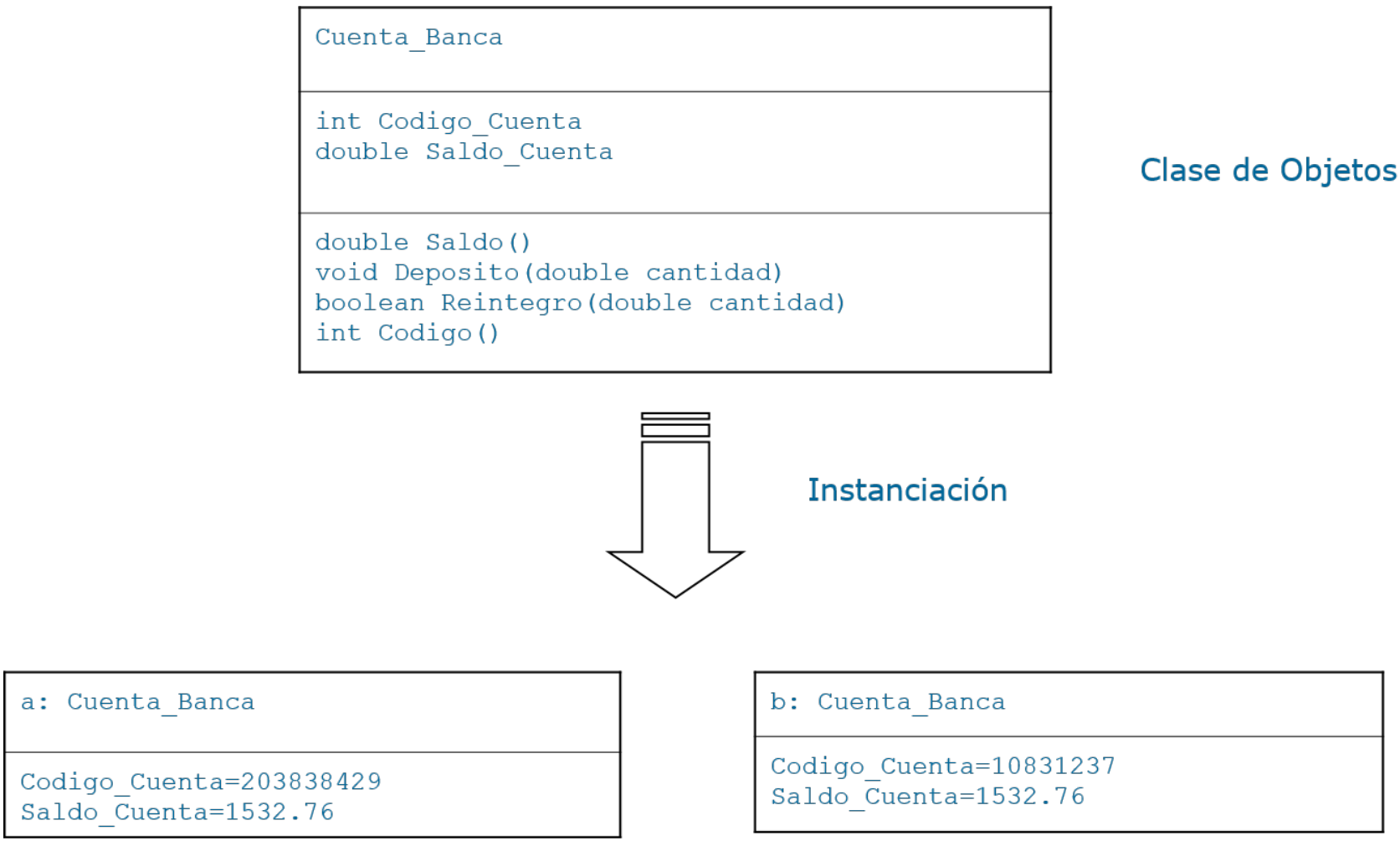


Figura: Objetos individuales (instancias de clase)

Tipos de Métodos en una Clase

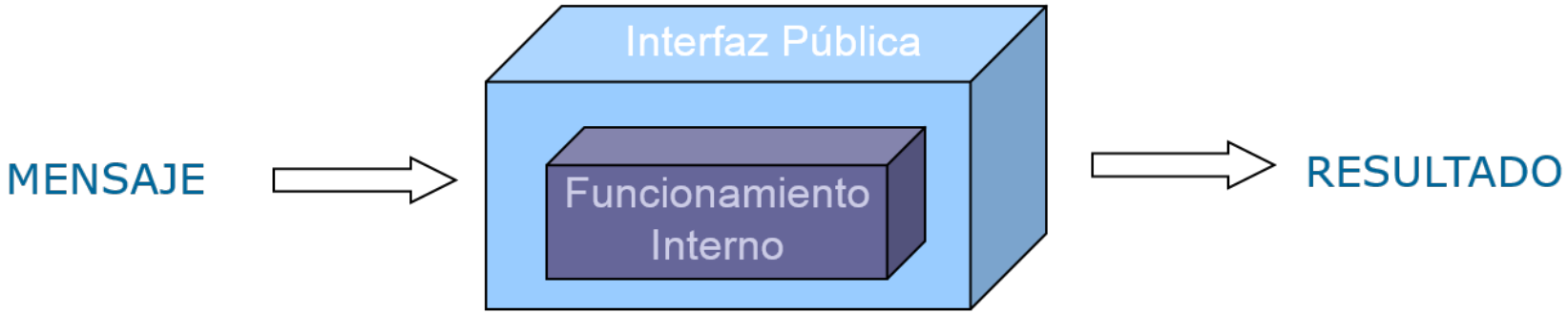
- Constructores. Permiten crear instancias de una clase.
- Observadores. Permiten consultar la información encapsulada.
- Modificadores. Permiten cambiar la información encapsulada.
- Destructores. Destruyen objetos innecesarios y liberan recursos. En Java no son necesarios, al incorporar el lenguaje la recolección automática de basura.

Código 6: codigos_t2/Cuenta_Banca.java

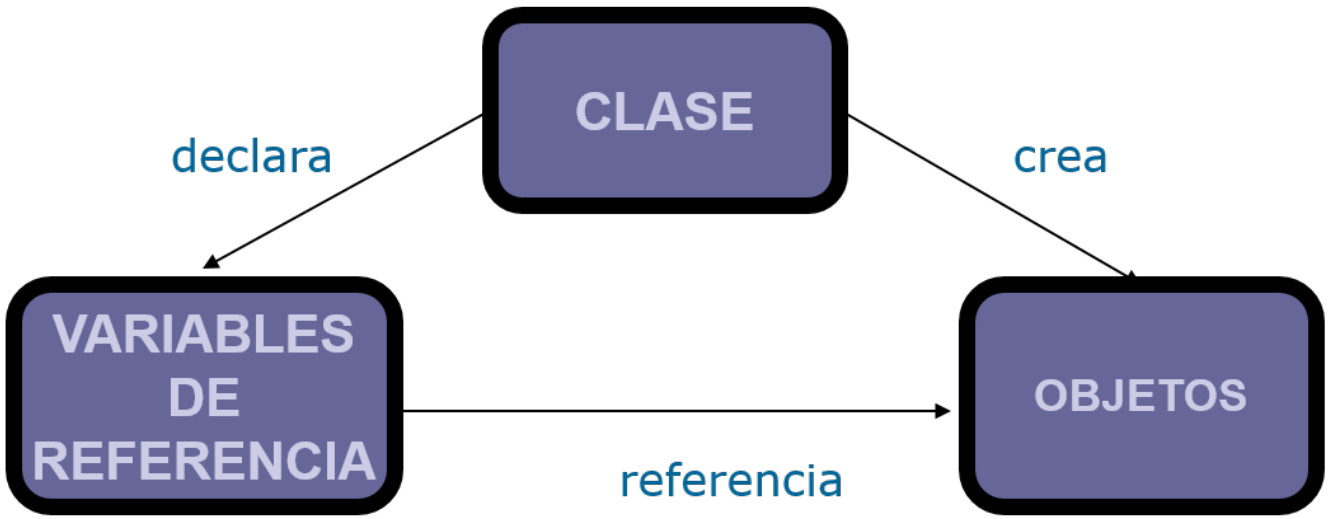
```
1  public class Cuenta_Banca //Nombre de la clase
2  {
3    private int Codigo_Cuenta; //Codigo Cuenta Cliente
4    private double Saldo_Cuenta; //Saldo Actual
5
6    public Cuenta_Banca(int id, double disponible) //constructor
7    {
8      Codigo_Cuenta = id;
9      Saldo_Cuenta = disponible;
10   }
11
12   public double Saldo() //observador
13   {
14     return (Saldo_Cuenta);
15   }
16
17   public void Deposito(double Cantidad) //modificador
18   {
19     if (Saldo_Cuenta > 0)
20       Saldo_Cuenta = Saldo_Cuenta + Cantidad;
21   }
22
23
```

```
24      public boolean Reintegro(double Cantidad) //modificador
25      {
26        if (Cantidad <= 0) || (Cantidad > Saldo_Cuenta)
27          return (false)
28        else
29        {
30          Saldo_Cuenta = Saldo_Cuenta - Cantidad;
31          return (true);
32        }
33      }
34
35      public int Codigo() //observador
36      {
37        return (Codigo_Cuenta);
38      }
39  }
```

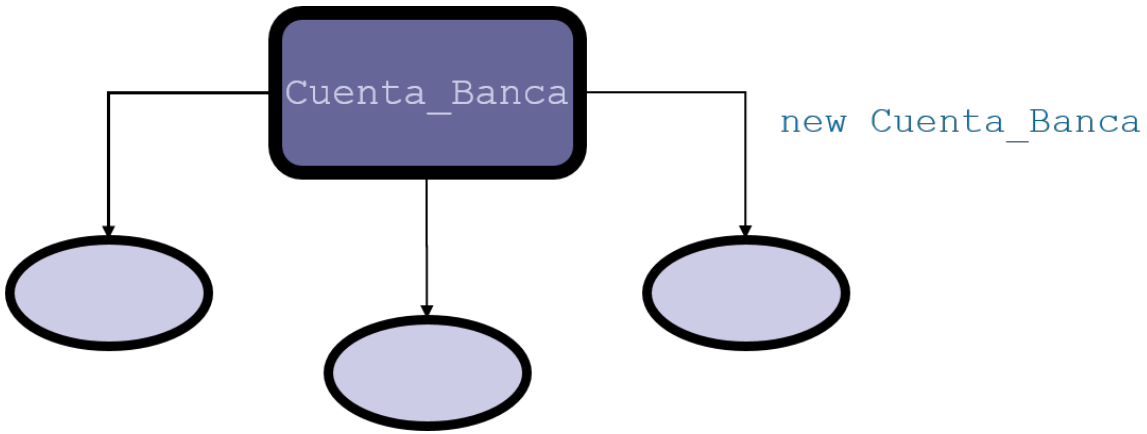
Esquema de la Estructura de un Objeto



Esquema de la Estructura de un Objeto



Creación de Objetos de una Clase I



- Una vez declarada una clase pueden crearse objetos de la misma
- El nombre de la clase actúa como un tipo de dato, y los objetos se crean y se referencian con variables del tipo.
- Tales variables son de referencia, y por tanto contienen la dirección de memoria de un objeto de la clase o el valor `null` para una referencia no válida.

Creación de Objetos de una Clase II

- El operador new asigna espacio dinámicamente y se utiliza para crear objetos.
- En Java, todos los objetos son creados en tiempo de ejecución mediante new.
- En cada clase hay un método especial, que recibe el mismo nombre de la clase. Es el constructor, y sirve para inicializar los campos del objeto creado.

Creación de Objetos de una Clase III

- Las siguientes sentencias declaran variables del tipo Cuenta_Banca.

```
1  Cuenta_Banca Antonio;
2  Cuenta_Banca Juan = new Cuenta_Banca (32723586, 100000);
```

- La primera declaración no inicializa la cuenta de Antonio, y la referencia por tanto sería null. La segunda declara el objeto Juan, lo crea mediante el operador new, y lo inicia mediante la llamada al constructor Cuenta_Banca (32723586, 100000).
- Si ahora deseamos iniciar el objeto Antonio, bastará con incluir la siguiente sentencia en nuestro programa

```
1  Antonio = new Cuenta_Banca (45072569, 25000);
```

Creación de Objetos de una Clase IV

- El propósito único de un constructor es realizar inicializaciones cuando se crea un objeto.
- En consecuencia, un constructor nunca retorna un tipo de dato.
- El resultado de utilizar el constructor es una referencia (dirección de memoria) al objeto creado, que se guarda en una variable del mismo tipo. A partir de entonces, el objeto es referenciado a través de dicha variable.

Creación de Objetos de una Clase V

- Es posible tener varias versiones del constructor, siempre que se distingan por tener diferentes listas de argumentos.
- Los constructores pueden ser públicos, privados, etc., lo cual permite otorgar diferentes privilegiso de creación de objetos.

```
1       public Cuenta_Banca(int id) //constructor simplificado
2       {
3         Codigo_Cuenta=id;
4       }
5
6       public Cuenta_Banca() //constructor  nulo
7       {}
```

Protección de Miembros de la Clase

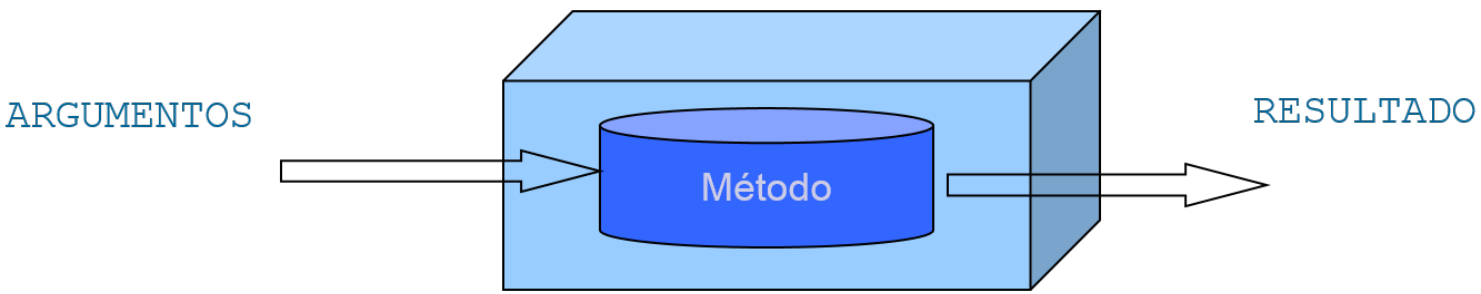
- En java el principio de ocultación se concreta en diferentes niveles de visibilidad de los miembros de una clase
- En particular podemos tener:
  - Miembros públicos: no tienen ninguna protección especial
  - Miembros privados: inaccesibles desde el exterior de la clase
  - Miembros protegidos: como los anteriores, pero permitiendo el acceso desde clases descendientes (herederas).
  - Miembros de paquete: accesibles desde el resto de clases del paquete.

Control de Acceso a Miembros de la Clase

Modificador	Clase	Paquete	Subclase	El mundo
public	S	S	S	S
protected	S	S	S	N
sin modificador	S	S	N	N
private	S	N	N	N

Llamadas a Objetos de una Clase I

- Una vez que un objeto ha sido creado, podemos comunicarnos con él enviándole mensajes, a través de la interfaz que su conjunto de métodos nos define.
- Un mensaje será por tanto una llamada a un métododel objeto, junto con los parámetros que en su caso sean necesarios.



Llamadas a Objetos de una Clase II

- La notación general de acceso a un campo o método de un objeto ya creado es

```
1  objeto.campo; objeto.metodo (lista de parametros);
```

- Desde el exterior, sólo son accesibles aquellos campos o métodos declarados public.
- Desde el interior son accesible todos los campos o métodos.

Llamadas a Objetos de una Clase III

Código 7: codigos_t2/Usa_Cuenta.java

```
1  //Fichero Usa_Cuenta.java
2  public class Usa_Cuenta {
3    public static void main(String[] args) {
4      Cuenta_Banca Antonio;
5      Cuenta_Banca Juan = new Cuenta_Banca(32756821, 100000);
6
7      Juan.Deposito(25000);
8      //Se muestra el nuevo saldo de la cuenta de Juan;
9      System.out.println("El saldo de Juan es:" + Juan.Saldo());
10
11     //Se crea la cuenta de Antonio;
12     Antonio = new Cuenta_Banca(45073256, 25000);
13
14     //Se muestra el codigo de la cuenta de Antonio;
15     System.out.println("El codigo de Antonio es:" +
           Antonio.Codigo());
16
17     //Antonio hace un reintegro por valor de 2000 y se muestra
           el saldo;
18     Antonio.Reintegro(2000);
```

Llamadas a Objetos de una Clase IV

```
19        System.out.println("Nuevo saldo de Antonio: " +
              Antonio.Saldo());
20    }
21  }
```

Ejercicios I

- Descargue el fichero `Usa_Cuenta_Banca.java`.
- Compile y ejecute.
- Cree dos nuevas instancias de la clase `Cuenta_Banca`.
- Haga un ingreso en la primera y un reintegro en la segunda
- Muestre en el terminal toda la información posible de ambas instancias usando los observadores que la clase `Cuenta_Banca` proporciona.
- Descargue, compile y ejecute la clase `GradosProteccion.java` y `UsaGradosProteccion.java`. Analice qué sucede.

Ejercicios II

- Desarrolle una clase que abstraiga el concepto de alumno universitario. Llámela `Alumno.java`.
- Escriba ahora un programa que haga uso de la clase anterior. Llámelo `Usa_Alumno.java`.
- Desarrolle una clase que abstraiga el concepto de semáforo de tráfico. Inicialmente estará rojo. Llámela `Semaforo.java`.
- Escriba ahora un programa que haga uso de la clase anterior. Comenzará escribiendo el color, luego lo cambiará a ámbar, y volverá imprimir el color. Llámelo `Usa_Semaforo.java`.
- Desarrolle una clase que abstraiga un Punto del Plano. Llámela `Punto.java`
- Escriba ahora un programa que haga uso de la clase anterior. Llámelo `Usa_Punto.java`.

Objetos de Array I

- En Java, un array es un objeto que contiene varias ubicaciones contiguas de memoria, que almacenan datos del mismo tipo.
- Como cualquier objeto, un array es creado con new, y sus posiciones son indexadas desde cero hasta la longitud total menos uno.
- Las siguientes, son declaraciones válidas de array:

```
1       int [] b = new int[10];
```

- Se crea un array de 10 enteros referenciado por b, y accesibles por b[0],...,b[9]

```
1       int [] arr;
```

Objetos de Array II

- Se declara arr como un array, pero el objeto no existe al no haber sido creado. Para ello, asignándole memoria, podemos hacer

```
1       arr = new int[30];
```

- El método length nos dice siempre la longitud de un array.
- Por tanto arr.length retorna 30 y b.length retorna 10.

Objetos de Array III

Código 8: codigos_t2/Usa_Cuenta_2.java

```
1  //Fichero Usa_Cuenta_2.java
2  //Declara un array de cuentas bancarias;
3  public class Usa_Cuenta_2 {
4    private static int Num_Clientes = 30;
5    static void Muestra_Cuentas(Cuenta_Banca[] Cuentas) {
6      int i;
7      System.out.println("Numero de Cuenta" + "   " +
           "Disponible");
8      for (i = 0; i <= Cuentas.length; i++) {
9        System.out.println(Cuentas[i].Codigo() + "       " +
             Cuentas[i].Saldo());
10     }
11   }
12
13   public static void main(String[] args) {
14     //Se crea un array con la lista de clientes;
15     //Cada celda del array es un objeto de la clase
           Cuenta_Banca;
16     Cuenta_Banca[] Clientes = new Cuenta_Banca[Num_Clientes];
17     //Se inicializan algunos objetos del array;
18     Clientes[0] = new Cuenta_Banca(45072253, 10000);
```

Objetos de Array IV

```
19        Clientes[1] = new Cuenta_Banca(32074263, 1000);
20        Clientes[2] = new Cuenta_Banca(75395412, 25394);
21        //Se muestra el estado de las cuentas;
22        Muestra_Cuentas(Clientes);
23      }
24    }
```

Ejercicios

- Desarrolle, haciendo uso de arrays, una clase que modele el concepto de matriz de números reales. Llámela `Matriz.java`.
- Escriba ahora un programa que haga uso de la clase anterior. Llámelo `Usa_Matriz.java`.
- Desarrolle, haciendo uso de arrays, una clase que represente el concepto de polinomio. Llámela `Polinomio.java`.
- Escriba ahora un programa que haga uso de la clase anterior. Llámelo `Usa_Polinomio.java`.

Objetos de Cadena de Caracteres I

- Una cadena es una secuencia finita de caracteres.
- Java proporciona la clase `String` para manejarlas. Un objeto de la clase `String` referencia a una cadena.
- La siguiente declaración inicializa mensaje con la cadena indicada.

```
1       String mensaje = "En un lugar de la Mancha...";
```

- Cualquier objeto de la clase `String` tiene asociado un método `length` que devuelve al ser invocado la longitud del objeto de cadena. En nuestro caso, `mensaje.length()` devolvería 27.
- Igualmente, cada carácter de la cadena es accesible individualmente mediante la llamada al método `charAt(posición)`. Por ejemplo, `mensaje.charAt(0) == "E"`.

Objetos de Cadena de Caracteres II

- Es posible encadenar objetos de la clase `String` utilizando el operador +.
- La siguiente instrucción nos produciría como salida “Don Quijote”.

```
1       System.out.println("Don" + " " + "Quijote");
```

Llamadas a Métodos y Paso de Argumentos I

- Un método siempre está encapsulado en una clase. Su definición especifica el número y tipo de los argumentos requeridos (parámetros formales).
- Dentro de una clase, un método no se identifica sólo por su nombre, sino también por la lista de parámetros formales.
- En consecuencia, es posible tener dentro de una misma clase diferentes métodos con diferentes argumentos, pero con el mismo nombre. Esta característica se conoce con el nombre de sobrecarga de métodos (esto incluye a los constructores).
- Un ejemplo de método sobrecargado es `System.out.println`, que como hemos visto permite enviar a la salida estándar texto, enteros, etc.

Llamadas a Métodos y Paso de Argumentos II

- Al examinar los tipos de los parámetros utilizados en la llamada, el compilador de Java decide a cuál de las múltiples versiones del método sobrecargado debe invocar
- En Java los argumentos de tipos primitivos se pasan siempre por valor, mientras que los objetos (incluyendo los arrays) se pasan siempre por referencia. Esto no necesita ser indicado por el programador, sino que es el propio compilador, examinando cada argumento, quien decide como debe pasarse. Esto no es estrictamente cierto: al pasar un objeto, lo que se pasa es la referencia al mismo por valor, teniendo esto el mismo efecto que un paso por referencia.

Ejemplo de Sobrecarga de Métodos

Código 9: codigos_t2/Sobrecarga.java

```
1  public class Sobrecarga //Ejemplo de sobrecarga de metodos
2  {
3    public static int Suma(int x, int y) {
4      return x + y;
5    }
6
7    public static float Suma(float x, float y) {
8      return x + y;
9    }
10
11   public static double Suma(double x, double y) {
12     return x + y;
13   }
14
15   public static void main(String[] args) {
16     System.out.println(Suma(2, 3));
17     System.out.println(Suma(2.0, 3.0));
18     System.out.println(Suma(2.0, 3.0));
19   }
20 }
```

Argumentos de la Línea de Comandos I

- El método main es especial en una clase de Java, puesto que marca el inicio de la ejecución por parte del intérprete del lenguaje.
- Un método main también recibe argumentos, especificados en la línea de comandos cuando se ejecuta un programa Java desde el *shell* del sistema.
- Los argumentos de la línea de comandos se pasan a main como cadenas.
- La declaración habitual del método `main` en cualquier clase es de la forma

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

- donde `args` es un array de cadenas cada una de las cuales es un argumento que se deberá pasar a `main` desde la línea de comandos.

Argumentos de la Línea de Comandos II

- Si se desean pasar argumentos de otros tipos, primero deben pasarse como cadenas dentro de `args`, y luego deben ser convertidos a los tipos deseados.
- Si la línea de comando es `java prueba arg1 arg2`, entonces el método `main` de la clase prueba recibe dos argumentos, situados en `args[0]` y `args[1]`.
- Naturalmente, se tendrá para este caso que `args.length()` vale 2.

Argumentos de la Línea de Comandos III

Código 10: codigos_t2/Factorial_n.java

```
1  //Ejemplo de clase que toma argumentos desde la linea de
       comandos
2  //Uso:   java Factorial_n n siendo n un numero mayor que cero
3  public class Factorial_n {
4    //metodo interno de la clase; calcula el factorial
         recursivamente
5    static int factorial(int n) {
6      if (n == 1) return 1;
7      else return n * factorial(n - 1);
8    }
9    public static void main(String[] args) {
10     int Auxiliar;
11     if (args.length != 1) {
12       System.err.println("Uso java Factorial_n numero_entero");
13       System.exit(-1);
14     }
15     Auxiliar = Integer.valueOf(args[0]).intValue();
16     System.out.println("Resultado es: " + factorial(Auxiliar));
17   }
18 }
```

Manejo Básico de Errores. Excepciones. I

- En ocasiones es necesario desplegar mensajes de error. En concreto, un método `main` debería verificar que los argumentos recibidos desde la línea de comandos son correctos.
- Para ello, se utiliza el objeto `System.err` que envía el mensaje de error a la pantalla.
- Si debido al error se desea terminar el método, se usa la llamada `System.exit(estado)` donde estado es un número.
- Se define una excepción como un error producido en tiempo de ejecución. En Java, tales errores se manejan con mecanismos que capturan la excepción y la tratan.
- Los diferentes errores se representan con varios objetos de excepción.

Manejo Básico de Errores. Excepciones. II

- Para capturar y tratar excepciones se utiliza el bloque de prueba, de estructura:

```
1       try {
2         codigo que podria generar la excepcion;
3       } catch (tipo_excepcion_1 e) {
4         instrucciones que tratan la excepcion 1
5
6       } catch (tipo_excepcion_2 e) {
7         instrucciones que tratan la excepcion 2;
8
9       } finally {
10        instrucciones a ejecutar siempre tras tratar (o no)
              la excepcion;
11      }
```

- Las cláusulas de captura hacen coincidir secuencialmente las excepciones causadas por cualquier instrucción de la parte try con los tipos de excepción, tratándolas de acuerdo con las instrucciones correspondientes.

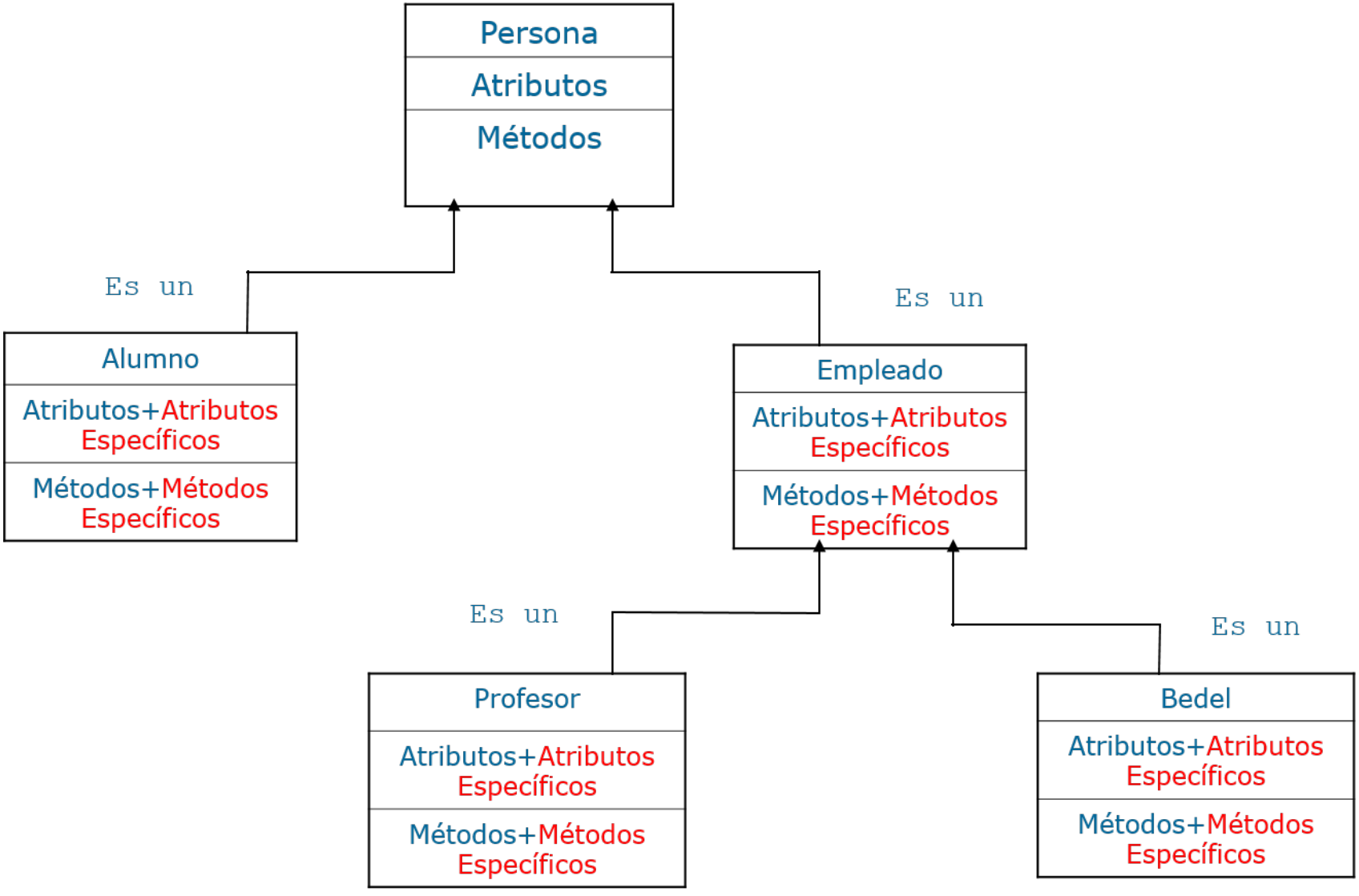
Reutilización de Clases. Modelo de Herencia en Java I

- Java soporta herencia simple pero no múltiple. Lo hace mediante el mecanismo de extensión de clase.
- A partir de él, una subclase se extiende (hereda) a partir de una única superclase todos los campos y métodos de esta, pudiendo además añadir los suyos propios.
- Esto permite la reutilización de código ya existente.
- La definición general de una clase extendida mediante herencia adopta la siguiente estructura

```
1       class Nombre_Clase extends Nombre_SuperClase
2       {
3         cuerpo de la clase extendida
4       }
```

Reutilización de Clases. Modelo de Herencia en Java II

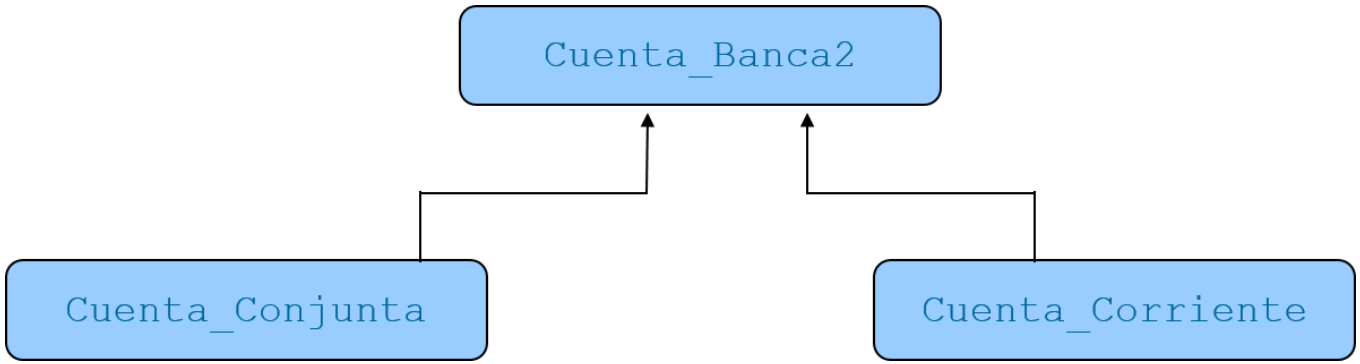
- Una clase extendida hereda todos los miembros de la superclase y agrega otros nuevos de su propiedad.
- Presentamos el uso de este modelo de herencia extendiendo la clase que modela a una cuenta bancaria a una subclase que modela una cuenta conjunta.
- Una clase etiquetada `final` no puede extenderse.


Persona
Atributos
Métodos
Es un
Es un
Alumno
Atributos+Atributos Específicos
Métodos+Métodos Específicos
Empleado
Atributos+Atributos Específicos
Métodos+Métodos Específicos
Es un
Es un
Profesor
Atributos+Atributos Específicos
Métodos+Métodos Específicos
Bedel
Atributos+Atributos Específicos
Métodos+Métodos Específicos

Código 11: codigos_t2/Cuenta_Banca2.java

```
1  //Fichero Cuenta_Banca2.java
2  public class Cuenta_Banca2 //Nombre de la clase
3  {
4    public Cuenta_Banca2() {} //constructor nulo
5    public Cuenta_Banca2(int id, double disponible, String
         propietario) {
6      Codigo_Cuenta = id;
7      Saldo_Cuenta = disponible;
8      this.Nombre = Titular;
9    }
10   public double Saldo() {
11     return (Saldo_Cuenta);
12   }
13
14   public void Deposito(double Cantidad) {
15     if (Saldo_Cuenta > 0) Saldo_Cuenta = Saldo_Cuenta +
           Cantidad;
16   }
17
18   public boolean Reintegro(double Cantidad) {
19     if (Cantidad <= 0) || (Cantidad > Saldo_Cuenta) return
           (false)
20     else {
```

```
21          Saldo_Cuenta = Saldo_Cuenta - Cantidad;
22          return (true);
23        }
24      }
25      public int Codigo() {
26        return (Codigo_Cuenta);
27      }
28      protected int Codigo_Cuenta; //Codigo Cuenta Cliente
29      protected String Titular; //Titular de la Cuenta
30      private double Saldo_Cuenta; //Saldo Actual
31  }
```



- Cuando se llama a un método de una clase extendida se comienza buscando en el ámbito de la propia clase. Si no se encuentra, se examina la superclase, ascendiendo hacia arriba en la jerarquía.
- Los objetos extendidos de la clase extendida suelen tener dos partes: un objeto de super clase (que contiene a los miembros heredados) y una parte anexa.
- Un objeto protected solo puede ser accedido por código de paquete y por subclases que lo tengan como miembro heredado.

Código 12: codigos_t2/Cuenta_Conjunta.java

```
1  class Cuenta_Conjunta extends Cuenta_Banca2 //Herencia de la
       clase Cuenta_Banca2
2  {
3    public Cuenta_Conjunta() {} //constructor nulo
4    public Cuenta_Conjunta(int n, double b, String prop1, String
       prop2) {
5      super(n, b, prop1); //llama al constructor de la superclase
6      Titular2 = prop2;
7    }
8    private String Titular2; //Otro titular de una cuenta conjunta
9  }
```

Código 13: codigos_t2/Usa_Cuenta_Conjunta.java

```
1  public class Usa_Cuenta_Conjunta //Ejemplo de uso de la clase
2  {
3    public static void main(String[] args) {
4      Cuenta_Conjunta AntonioyJuan;
5
6      AntonioyJuan = new Cuenta_Conjunta(1234, 25000, "Antonio
           Lopez", "Juan Perez");
7      AntonioyJuan.Deposito(25000);
8      System.out.println(AntonioyJuan.Saldo());
9    }
10 }
```

Código 14: codigos_t2/Cuenta_Corriente.java

```
1  class Cuenta_Corriente extends Cuenta_Banca2 {
2    public Cuenta_Corriente() {} //constructor nulo
3    public Cuenta_Corriente(int n, double b, String prop) {
4      super(n, b, prop);
5      gratuita = (b >= Saldo_Minimo);
6    }
7
8    public static void Cambiar_Saldo_Minimo(float m) {
9      Saldo_Minimo = m;
10   }
11
12   public static void Cambiar_Comision(float f) {
13     Comision = f;
14   }
15
16   public boolean Reintegro(double Cantidad) {
17     boolean Resultado = super.Reintegro(Cantidad);
18     if (Resultado && (Saldo() < Saldo_Minimo))
19     gratuita = false;
20     return (Resultado);
21   }
22   public boolean Cobro_Comision() //cargo mensual, si procede
23   {
24     boolean Resultado;
```

```
25        if (!gratuita) {
26          if (Resultado = Reintegro(Comision)) gratuita = (Saldo()
                >= Saldo_Minimo);
27          return (Resultado);
28        }
29        return (true);
30      }
31
32      private static float Saldo_Minimo = 2500;
33      private static float Comision = 18;
34      private boolean gratuita;
35    }
```

Ejercicios

- Considere las entidades persona, alumno, empleado, profesor y bedel. Escriba clases, utilizando herencia y aprovechando las relaciones "es un" que modelen a las entidades.
- Escriba un código `Usa_clases`, que emplee lo anterior.
- Considere las entidades coche, avión, barco, vehículo, avioneta, fragata, camión. Escriba clases, utilizando herencia y provechando las relaciones "es un" que modelen a las entidades.
- Escriba un código `Usa_clases_2`, que emplee lo anterior.

Sobreescritura I

- Es una modificación parcial del comportamiento de la superclase, para adaptarlo a un nuevo comportamiento en la subclase.
- Sintácticamente, se vuelve a escribir el método que se quiere modificar en el cuerpo de la subclase, dándole el comportamiento que se desea, y atendiendo a las siguientes reglas:
  - Prevención de conflictos con métodos estáticos.
  - La signatura del método original y el sobreescrito deben ser iguales.
  - El método sobreescrito pueden tener claúsula `throws`.
  - Pero sólo podrá lanzar excepciones declaradas en el método original.
  - Métodos declarados `final` no son susceptibles de sobreescritura.

Sobreescritura II

Código 15: codigos_t2/Superclase.java

```
1  public class Superclase {
2    private int i;
3    public Superclase(int dato) {
4      i = dato;
5    }
6
7    public void accion() {
8      for (int cont = 1; cont < i; cont++) {
9        System.out.println("Superclase escribiendo...");
10     }
11   }
12
13   public void hola() {
14     System.out.println("hola");
15   }
16 }
```

Sobreescritura III

Código 16: codigos_t2/Subclase.java

```
1  import java.util.*;
2  public class Subclase extends Superclase //hereda de superclase
3  {
4    public Subclase() {
5      super(3);
6    }
7
8    //sobreescribe el metodo accion, con otro compartamiento
9    public void accion(double a, double b) {
10     System.out.println("La hipotenusa es " +
           Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2)));
11   }
12 }
```

Sobreescritura IV

Código 17: codigos_t2/UsaSuperySub.java

```
1  public class UsaSuperySub {
2    public static void main(String[] args) {
3      Superclase A = new Superclase(10);
4      Subclase B = new Subclase();
5      A.accion(); //superclase usa metodo original
6      B.accion(3, 4); //subclase usa metodo sobreescrito
7      A.hola();
8      B.hola(); //y esto, que significa?
9    }
10 }
```

Ejercicios

- La clase `Object` incorpora el método `toString`, para proporcionar una representación de cadena de los objetos. Inspeccione el API de la jerarquía de clases y familiarícese con él.
- Sobreescriba ahora el método en el ámbito de la clase `Cuenta_Conjunta` de forma que permita desplegar objetos de esta clase en pantalla en un formato legible.
- Escriba una clase.

Interfaces I

- Se pueden entender como "clases" sin implementación definida.
- Se componen de un conjunto de signaturas de métodos.
- Palabra reservada `interface`.
- Por defecto, sus métodos son `public`.
- Es implementada por una clase, que describe un procesamiento concreto para los métodos de interfaz.
- Diferentes clases-diferentes implementaciones de la misma interfaz.
- Una clase puede implementar varias interfaces ("herencia múltiple").

Interfaces II

Código 18: codigos_t2/Ejemplo_interface.java

```
1  public interface Ejemplo_interface {
2    int operacion_binaria(int a, int b);
3    int operacion_monaria(int a);
4  }
5
6  public class Imp_Interface implements Ejemplo_interface {
7    public int operacion_binaria(int a, int b) {
8      return a + b;
9    }
10
11   public int operacion_monaria(int a) {
12     return -a;
13   }
14 }
```

Interfaces III

Código 19: codigos_t2/Imp_Interface_2.java

```
1  public class Imp_Interface_2 implements Ejemplo_interface {
2    public int operacion_binaria(int a, int b) {
3      return a * b;
4    }
5
6    public int operacion_monaria(int a) {
7      if (a != 0) return 1 / a;
8      else return 0;
9    }
10 }
```

Interfaces IV

Código 20: codigos_t2/Imp_Interface_3.java

```
1  public class Imp_Interface_3 implements Ejemplo_interface {
2    public int operacion_binaria(int a, int b) {
3      System.out.println("Hola, como estas...");
4      return 25;
5    }
6
7    public int operacion_monaria(int a) {
8      return a * a * a;
9    }
10 }
```

Bibliografía


Arnold, K.
El lenguaje de programación Java
Prentice Hall, 2002
[Capítulo 20]


Eckel, B.
Piensa en Java
Prentice Hall, 2002