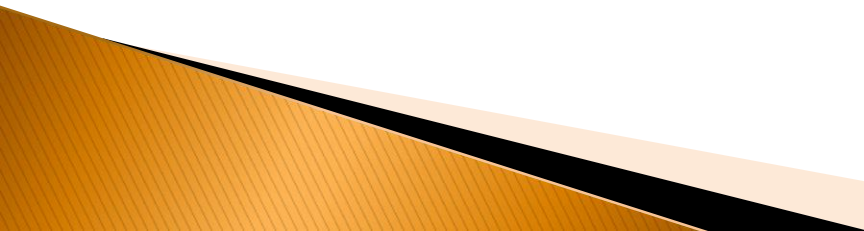


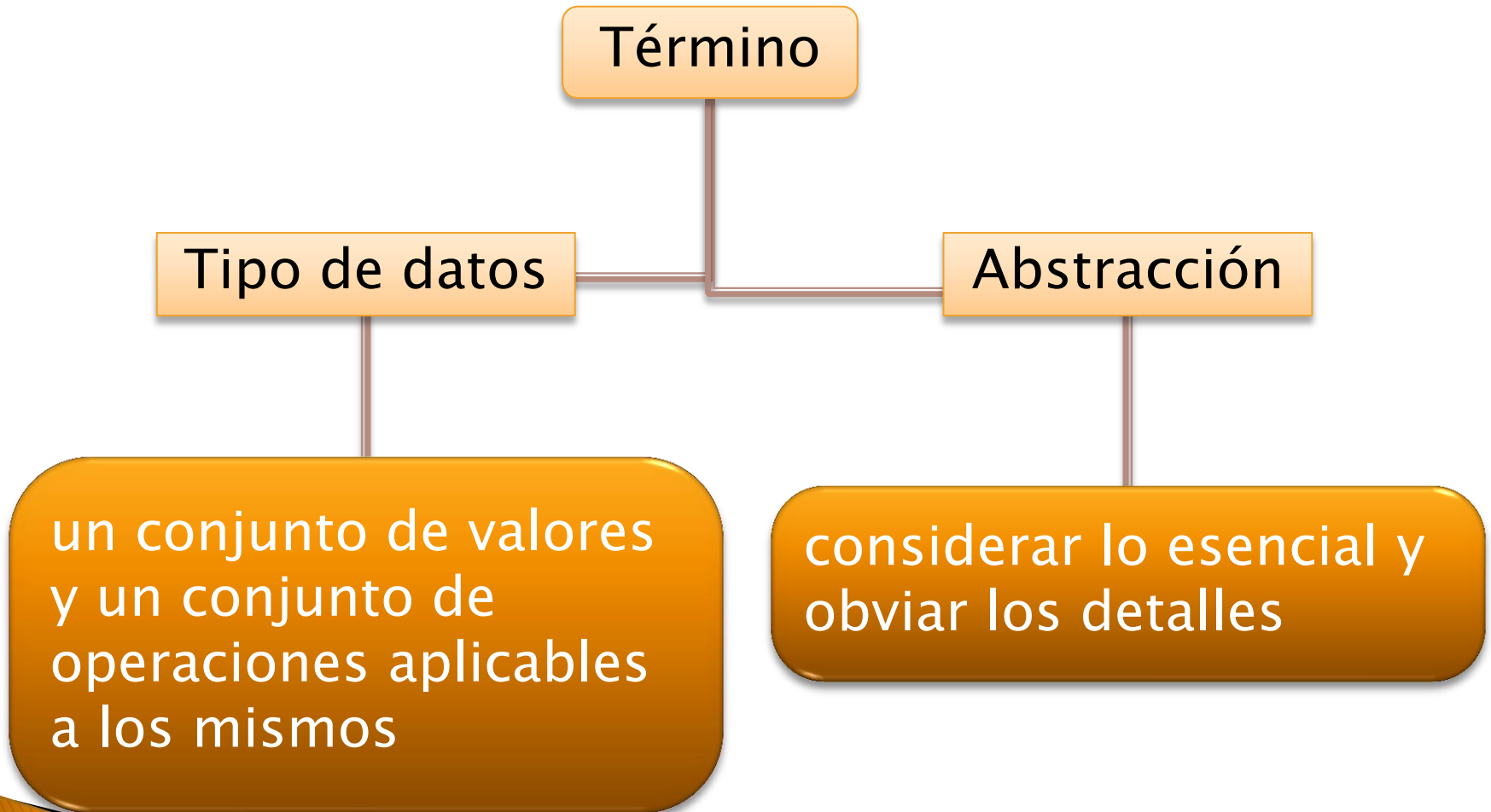
# Clases, objetos y referencias en Java

Programación I  
Grado en Ingeniería Informática  
MDR, JCRdP y JDGD

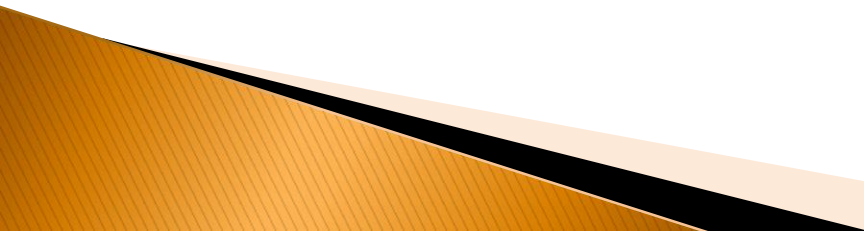
# Contenido

- ▶ Tipos abstractos de Datos (TAD)
  - ▶ Clases, objetos y referencias
  - ▶ Paso de parámetros
  - ▶ Ocultando implementación (modificadores de acceso)
  - ▶ Objeto actual (this)
  - ▶ Inicializando objetos (constructores)
  - ▶ Métodos y atributos de clase (static)
  - ▶ Constantes (final)
- 

# Tipo Abstracto de Datos



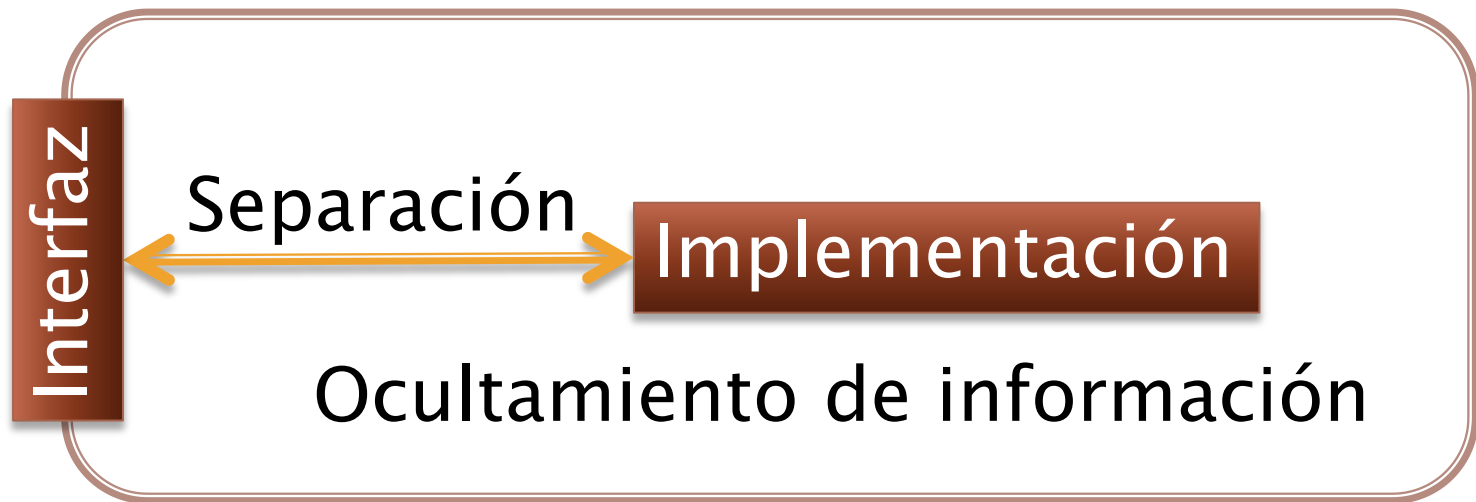
# Tipo Abstracto de Datos

- ▶ Todo tipo de datos sería un tipo abstracto de datos
  - ▶ El término se reserva para tipos no primitivos en cuyo diseño se han tenido en cuenta tres principios:
    - Separación
    - Encapsulamiento
    - Ocultamiento de información
- 

# Tipo Abstracto de Datos

- ▶ Dos aspectos de la realización de un TAD
  - Externo (interfaz)
  - Interno (implementación)

Encapsulamiento



# Identificación de objetos

- ▶ La descripción de la solución puede ser un texto informal
- ▶ En esa descripción en lenguaje natural, aparecen nombres
- ▶ Los nombre son los candidatos a ser objetos
- ▶ Los verbos serán las acciones (métodos) que pueden realizar (ejecutar) los objetos
- ▶ Los adjetivos pueden ser estados (atributos) de los objetos que califica
- ▶ Los complementos pueden ser parámetros o resultados de los métodos

# Clases

- ▶ Se diseñan a partir de la identificación de los objetos
- ▶ Descripción de las características y comportamiento de un conjunto de objetos
- ▶ Es la definición de los atributos y funciones de los objetos
- ▶ Todo objeto pertenece a una clase
- ▶ La relación: Clase=>Objeto  
similar a: Tipo=>Variable

# Objetos

- ▶ **Visión externa:** estado y funciones (operaciones) que puede realizar
- ▶ **Visión interna:** atributos (datos y objetos internos) que mantienen su estado y funciones
- ▶ Terminología de las **funciones** que realiza un objeto: *servicios, métodos o mensajes*
- ▶ Los objetos se crean explícitamente con **new**
- ▶ Para hacer uso de los servicios que ofrece un objeto se usa el siguiente formato:  
`nombreReferencia.nombreMetodo(parámetros)`



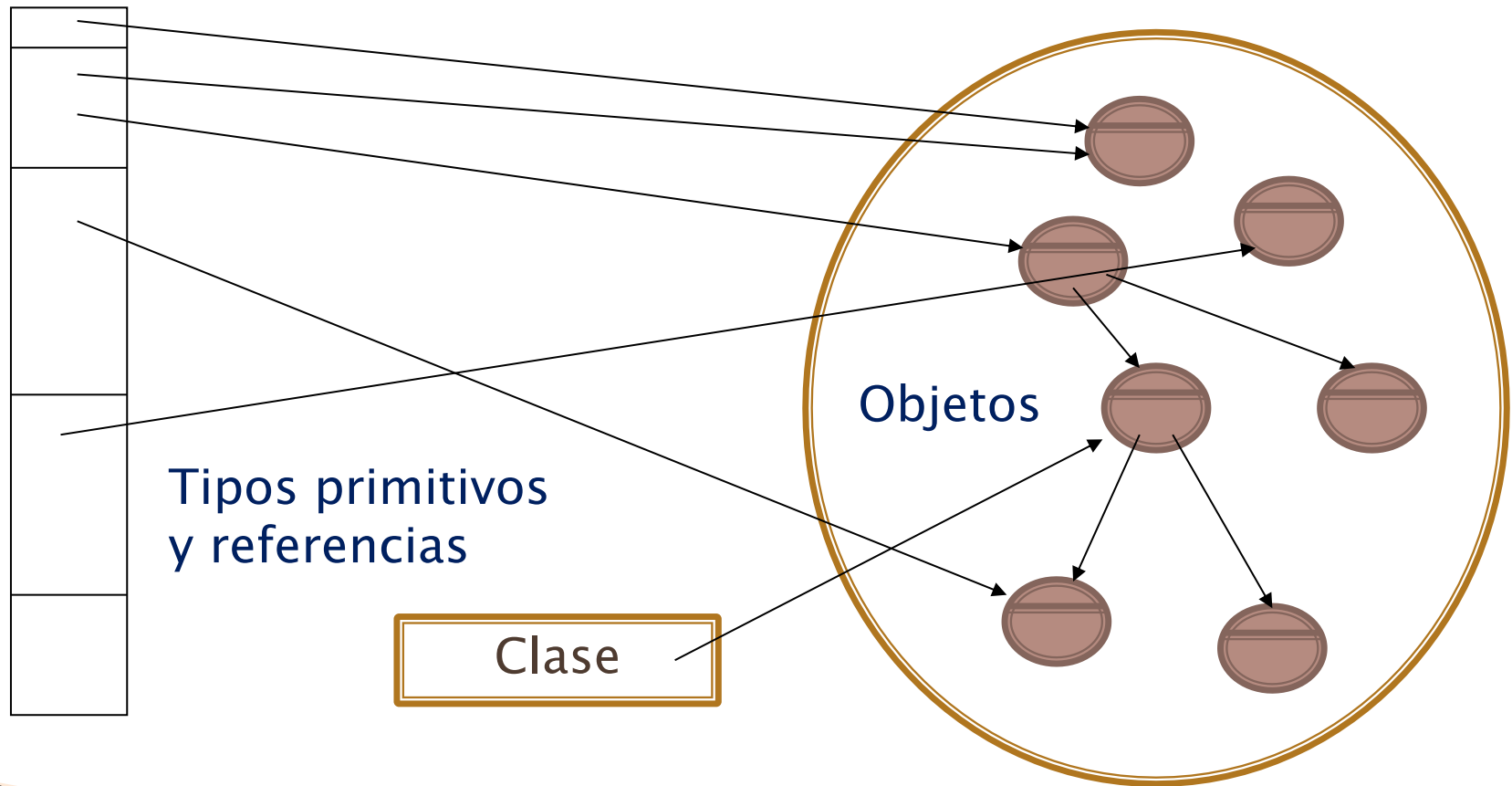
# Referencias

- ▶ Permiten acceder a los objetos (los objetos no son accesibles per sé)
- ▶ Son de un tipo
- ▶ Se almacenan como tipos primitivos
- ▶ Se declaran con:  
NombreClase nombreReferencia
- ▶ Permite las operaciones "=", "==", "!="
- ▶ Las operaciones anteriores **NO asignan** objetos o **comparan** objetos sólo referencias
- ▶ Para copiar un objeto se debe usar **clone()** y para compararlos **equals()**

# Representación de la memoria

Pila de ejecución

Memoria dinámica



# Nombres de identificadores

- ▶ Elija como identificadores nombres completos y no abreviaturas
- ▶ Los nombres de identificadores pueden estar formados por varias palabras
- ▶ Como regla general la palabra o palabras que forman un identificadores están en minúscula excepto el primer carácter de cada palabra que será mayúscula
- ▶ El primer carácter de los identificadores que no sean clases debe escribirse en minúscula
- ▶ Las constantes (`final`) se pueden escribir en mayúsculas separando cada palabra por "\_"

# Formato de Clase

```
[mod acceso] class NombreClase {  
    [mod acceso] atributo1;  
    [mod acceso] atributo2;  
    ...  
    [mod acceso] función miembro 1  
    [mod acceso] función miembro 2  
    ...  
}
```

# Acciones / Servicios / Métodos / Funciones / Mensajes

- ▶ Determinan lo que puede hacer un objeto
- ▶ Cada uno se identifica unívocamente mediante el nombre y la lista de parámetros
- ▶ **Lista de parámetros:** indica los tipos y el número de datos que es necesario pasar al método
- ▶ **Tipo de retorno:** tipo del valor que se obtiene al invocar al método

# Sobrecarga de métodos

- ▶ Pueden existir varios métodos con el mismo nombre
- ▶ Se distinguen por medio de la lista de parámetros
  - Parámetros de distinto tipo
  - Distinto número de parámetros
- ▶ No es posible distinguir dos métodos por el tipo del valor de retorno
  - En muchas ocasiones se descarta el valor de retorno

# Paso de parámetros/retorno

- ▶ El paso de parámetros y retorno es por valor. Se pasan valores, no variables
- ▶ Si no retornan datos el tipo de retorno se declara **void**
- ▶ La sentencia **return** indica el valor devuelto y produce su devolución inmediata
- ▶ Llamada a un método:  
Nombre de la referencia al objeto, seguido de un punto y del nombre del método:  
`nombreReferencia.nombreMetodo(parámetros)`

# Número variable de parámetros

- ▶ Para pasar un número variable de parámetros se emplea la característica *varargs* de Java
- ▶ En realidad es una forma abreviada de pasar un array
- ▶ Para establecer esta forma de paso de parámetros se escribe: el nombre del tipo, después una elipsis (tres puntos ...), un espacio y el nombre del parámetro
- ▶ El resultado es poder llamar al método con una lista de elementos separados por comas que se convierte en un array, o directamente un array



# Ejemplo de número variable de parámetros

```
public class Ejemplo {  
    public static int suma(int ... vec){  
        int s = 0;  
        for(int v : vec){  
            s += v;  
        }  
        return s;  
    }  
    ...  
    public static void main(String[] args){  
        int[] lista = {1, 2, 3};  
        System.out.println(suma(lista));  
        System.out.println(suma(4, 5, 7, -3));  
    }  
}
```

# Clasificación de los métodos

- ▶ **Constructores:** Se encargan de inicializar un objeto para poder comenzar a usarlo correctamente
- ▶ **Accesores:** Permiten acceder al estado de un objeto (información interna) sin transformarlo
- ▶ **Transformadores:** Alteran el estado de un objeto
- ▶ **Productores:** Partiendo de los parámetros y el estado del objeto producen un resultado sin transformar el objeto

# Ejemplo de clase

```
public class Cilindro {  
    private double altura, radio;  
    public void inicializar (double h, double r) {  
        altura = h;  
        radio = r;  
    }  
    public double calculaVolumen () {  
        double volumen;  
        volumen = altura*radio*radio*Math.PI;  
        return volumen;  
    }  
}
```

# Ejemplo de clase

Cilindro
-altura:double -radio:double
+inicializar():void +calculaVolumen():double

# Ejemplo de uso de la clase Cilindro

```
import java.util.Scanner;

public class UsaCilindro {
    public static void main(String[] args) {
        double alt, rad;
        Scanner teclado = new Scanner(System.in);
        System.out.println("Introduce la altura: ");
        alt = teclado.nextDouble();
        System.out.println("Introduce el radio: ");
        rad = teclado.nextDouble();
        Cilindro c = new Cilindro();
        c.inicializar(alt, rad);
        System.out.println("El volumen del cilindro es: "
                           + c.calculaVolumen());
    }
}
```

# Ocultando implementación

- ▶ Para ocultar los detalles de implementación de una clase se usan **modificadores de acceso**
- ▶ Se aplican a cada atributo y método
- ▶ Se sitúan antes de la definición del elemento
- ▶ Determinan quién puede acceder al elemento
- ▶ Existen **cuatro modificadores** (niveles)
- ▶ Hay que aplicar siempre el modificador más restrictivo posible
- ▶ Los métodos de una clase tienen total acceso a sus elementos y objetos

# Modificadores de acceso a atributos y métodos (1 / 2)

- ▶ **public**: disponible a todo el mundo. Se debe usar para establecer la interfaz. No debe usarse para atributos a menos que sean **final** (constantes)
- ▶ *Privadas de paquete*: es el tipo por defecto, si no se establece un modificador se aplica éste. Las clases del mismo paquete y subpaquetes y clases derivadas tienen acceso.

# Modificadores de acceso a atributos y métodos (2/2)

- ▶ **protected**: sólo los miembros de la propia clase y clases derivadas pueden modificar y leer los atributos y ejecutar los métodos. Se establece cuando se desea que sólo clases derivadas tengan acceso
- ▶ **private**: sólo miembros de la propia clase pueden modificar y leer los atributos y ejecutar los métodos. Es la recomendable, siempre que sea posible



# Modificadores de acceso a atributos y métodos

Modificador/ quién puede acceder	Propia clase	Propio paquete	Subclases otro paquete	Todos
<b>public</b>	Sí	Sí	Sí	Sí
<i>sin modificador</i>	Sí	Sí	No	No
<b>protected</b>	Sí	Sí	Sí	No
<b>private</b>	Sí	No	No	No

# Objeto actual (this)

- ▶ Los métodos al ser llamados actúan sobre un objeto (independientemente de los parámetros)
- ▶ Dentro de la clase lo llamamos el **objeto actual**
- ▶ Dentro de un método, **this** referencia el objeto actual
- ▶ **this** nos permite acceder a los atributos del objeto actual y llamar a sus métodos
- ▶ Se puede omitir en la mayoría de los casos pero es **importante que se tenga presente**

# Usos de la referencia `this`

- ▶ Si se puede omitir ¿para qué sirve?
  - En sentencias `return` para devolver una referencia al propio objeto
  - Cuando un atributo queda oculto por una variable o parámetro con el mismo nombre (**error común**)
    - Se puede resolver cambiando el nombre de la variable
- ▶ Todo método dispone de esta referencia, exceptuando los de tipo **`static`**

# Ejemplo de uso de la referencia this

```
public class Hoja {  
    private int i=0;  
    public Hoja incrementar () {  
        this.i++;  
        return this;  
    }  
    public void print () {  
        System.out.println("i= "+ this.i);  
    }  
}
```

# Ejemplo de uso de la clase Hoja

```
public class UsaHoja {  
    public static void main(String args[]) {  
        Hoja x = new Hoja();  
        x.incrementar().incrementar().print();  
        // this es igual a x en todos los métodos  
    }  
}
```

# Estado de los objetos

- ▶ Los objetos durante su existencia tienen un estado
- ▶ El estado de un objeto lo determina los valores de sus atributos
- ▶ Pueden existir combinaciones de valores que lleven a un estado no válido o incorrecto
- ▶ Por ejemplo: si un objeto que representa un conjunto tiene un atributo entero para representar su cardinalidad, y éste tiene un valor negativo entonces el objeto no se encontraría en un estado válido

# Inicializando objetos

- ▶ Para garantizar que todo objeto antes de usarse tenga un estado válido, debemos inicializarlo al crearlo
- ▶ Los **objetos** que no cambian de estado se denominan **inmutables**. Ejemplo: String
- ▶ Aunque los atributos tienen **valores por defecto**: 0 para byte, short, int, long, float, double, '\u0000' para char, **false** para boolean y **null** para todas las referencias

# Inicializando objetos

- ▶ En Java podemos darle un valor inicial a sus atributos con: atributo = expresión
- ▶ Ejemplo:  
`private int length = 0;`
- ▶ Al crear un objeto con `new` cada uno de sus atributos se inicializará evaluando la expresión correspondiente y tomando el valor resultante



# Inicializando con constructores

- ▶ Para inicializar los atributos (el estado del objeto) teniendo en cuenta parámetros y ejecutando código se usan los constructores
- ▶ Son métodos que no devuelven nada (**ni void**)
- ▶ Tienen el mismo nombre que la clase
- ▶ Pueden tener parámetros
- ▶ Pueden sobrecargarse
- ▶ Se puede invocar a un constructor desde otro con  
`this(lista de parámetros)`

# Inicializando con constructores

- ▶ Si no tiene parámetros se le denomina **constructor por defecto**
- ▶ Se invoca automáticamente cuando se crea un objeto de la clase
- ▶ Si existen, es obligatorio llamarlos al hacer el **new** (la inicialización está garantizada)
- ▶ Si no está definido explícitamente se toma por defecto vacío de sentencias

# Ejemplo de clase con constructor

```
public class Cilindro {  
    double altura, radio;  
    public Cilindro(double h, double r) {  
        this.altura = h;  
        this.radio = r;  
    }  
    public double calculaVolumen() {  
        double volumen;  
        volumen = altura*radio*radio*Math.PI;  
        return volumen;  
    }  
}
```

# Ejemplo de uso del constructor

```
import java.util.Scanner;

public class UsaCilindro {
    public static void main(String[] args) {
        double alt, rad;
        Scanner teclado=new Scanner(System.in);
        System.out.println("Introduce la altura: ");
        alt = teclado.nextDouble();
        System.out.println("Introduce el radio: ");
        rad = teclado.nextDouble();
        Cilindro c = new Cilindro(alt, rad);
        System.out.println("El volumen del cilindro es: "
                           + c.calculaVolumen());
    }
}
```

# Métodos de clase (sin objeto)

- ▶ A veces se necesita crear métodos (funciones o procedimientos) no asociados a objetos
- ▶ Se denominan **métodos de clase**
- ▶ Ejemplos: `main`, `Math.sin`, `Math.cos`, etc.
- ▶ No se ejecutan actuando sobre un objeto, pero sí disponen del resto de características de los métodos aplicables a objetos
- ▶ Se establecen con el modificador **static**
- ▶ Pueden acceder a atributos **static**
- ▶ Se invocan con `NombreClase.nombreMétodo()`

# Atributos de clase

- ▶ Cuando es necesario disponer de atributos asociados a la clase y no a cada objeto (únicos)
  - Existe uno solo para toda la clase y objetos de ésta
  - Existen aunque no existan objetos
- ▶ Se crean aplicando el modificador **static** a un atributo
- ▶ El acceso es controlable como el resto de los atributos
- ▶ Se inicializan una sola vez con "=".

# Facilitando el uso de métodos y atributos de clase

- ▶ Se puede abreviar el uso de `NombreDeClase.atributo` o `NombreDeClase.método` pudiendo eliminar el nombre de la clase
- ▶ Para ello se usa  
`import static NombreDeClase.elemento`  
o  
`import static NombreDeClase.*;`
- ▶ A partir de este momento se puede usar el atributo o método correspondiente sin el nombre de la clase

# Ejemplo de uso de modificadores

```
public class Persona {  
    private static int maxId = 0;  
    private String nombre;  
    private long teléfono;  
    private int id;  
    private static int dameIdÚnico () {  
        return maxId++;  
    }  
    public Persona(String r, long i) {  
        nombre = r;  
        teléfono = i;  
        id = dameIdÚnico();  
    }  
    public int dameId() { return id;}  
    public String dameNombre() { return nombre;}  
}
```



# Constantes en Java (final)

- ▶ Es posible disponer de atributos, variables o parámetros constantes
- ▶ Se definen empleando el modificador **final**
- ▶ Más que constantes son variables que una vez inicializadas no se pueden modificar
- ▶ El valor al que se inicializan es resultado de evaluar una expresión durante la ejecución
- ▶ Puede tomar un valor distinto en cada inicialización.
- ▶ Permiten aumentar la fiabilidad del código evitando posibles errores

# Ejemplo de uso de final

```
public class Persona {  
    private static int maxId = 0;  
    private final String nombre;  
    private final long teléfono;  
    private final int id;  
    private static int dameIdÚnico () {  
        return maxId++;  
    }  
    public Persona(String r, long i) {  
        nombre = r;  
        teléfono = i;  
        id = dameIdÚnico();  
    }  
    public int dameId() { return id;}  
    public String dameNombre() { return nombre;}  
}
```

# Uso de la clase Persona

```
import java.util.Scanner;

public class UsaPersona {
    public static void main(String[] args) {
        String nombre;
        long tfno;
        Scanner teclado = new Scanner(System.in);
        System.out.println("Introduzca el nombre: ");
        nombre = teclado.nextLine();
        System.out.println("Introduzca el número de teléfono: ");
        tfno = teclado.nextLong();
        Persona una = new Persona(nombre, tfno);
        System.out.println("El identificador asignado a "
                           + una.dameNombre() + " es: "
                           + una.dameId());
    }
}
```

# Referencias bibliográficas

- ▶ The Java Tutorials: Language Basics  
<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>
- ▶ **Primitive Data Types**  
<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>
- ▶ ArgoUML <http://argouml.tigris.org>