# Clase (I)



- Concepto
  - Una clase es una agrupación de datos y funciones que operan sobre esos datos. A los datos se les llama **variables miembro o atributos** y a las funciones, **funciones miembro o métodos**.

- Sintaxis de declaración de una clase

**[Modificadores]** **class** **Identificador** **[Superclase]** **[Interfaces]** **CuerpoClase**

- Modificadores
  - los modificadores que puede tener una clase son: **abstract**, **final** o **public**.
  - **abstract**
    - una clase declarada **abstract** es una clase de la que no se pueden crear objetos.
    - una clase declarada **abstract** puede tener métodos declarados como **abstract** (aquellos que no se definen) o no. Pero si tiene algún método **abstract** es obligatorio que la clase sea declarada **abstract**.

# Clase (II)



- **final**
  - una clase declarada **final** no puede tener clases derivadas.
  - Se produce un error de compilación si una clase se declara **abstract** y **final**.
- **public**
  - una clase declarada **public** es accesible desde su paquete y desde otros. Si se omite este modificador, la clase solamente es accesible desde el paquete donde está definida.

- Identificador
  - Es el nombre de la clase. Debe de ser un identificador válido en Java.

# Clase (III)



- Superclase
  - Opcional, especifica la superclase inmediata.
  - Sintaxis: **extends IdentificadorSuperclase**
  - Si se omite en la declaración de la clase, entonces se considera que la superclase inmediata es **Object**. (Así se consigue que todas las clases deriven de manera inmediata o no de la clase **Object**, formándose así una jerarquía de clases).

- Interfaces
  - Opcional, especifica las interfaces implementadas por la clase.
  - Sintaxis: **implements ListaInterfaces**
  - La lista de interfaces se compone de identificadores de interfaces separados por comas.

- Cuerpo de la Clase
  - Formado por una lista de declaraciones de variables miembro y métodos, encerradas entre paréntesis. La lista de declaraciones puede estar vacía.

# Clase (IV)

- **Ejemplo:** declaración de la clase *Complejo*.

```
public class Complejo
{
   private double re , im;
   public Complejo(double r, double i)
   {
      re = r ;
      im = i ;
   }
   public void setRe(double r) { re = r ;}
   public void setIm(double i) { im = i ;}
   public double getRe() { return re;}
   public double getIm() { return im;}
   public double modulo() { return Math.sqrt(re * re + im * im);}
   public static Complejo conjugado(Complejo c)
   {
      Complejo elConjugado = new Complejo(c.getRe(), -c.getIm());
      return elConjugado;
   }
}
```

Declaración de las variable miembro

Constructor

Declaración de los métodos o funciones miembro

# Variables miembro (I)



- Propiedades
  - Definen el **estado o estructura de los objetos** de una clase.
  - Dentro de una clase cada variable miembro debe tener un **nombre único**.
  - Pueden ser **tipos primitivos** (**int**, **boolean**, etc) o **referencias** a objetos de otra clase.
  - Se **puede iniciar** un atributo a un valor concreto, pero no se hace habitualmente porque las tareas de iniciación de los atributos se delegan a los **constructores**.
  - **Cada objeto** que se crea de una clase tiene **su propia copia** de las variables miembro. A estas variables miembro se las denomina **variables miembro de instancia** frente a las **variables miembro de clase** de las cuales existe **una única copia** que comparten todos los objetos de la clase.
  - Los **métodos** acceden a las variables miembro por **su nombre directamente**, o usando la palabra reservada **this** y el operador punto.

**[this.]NombreAtributo**

# Variables miembro (II)



- Sintaxis de declaración de variables miembro

**[modificadores] TipoVariable Identificador**

- Modificadores
  - Modificadores de acceso
    - **public**: una variable miembro declarada **public** está accesible para cualquier otra clase o subclase.
    - **private**: una variable miembro declarada **private** está accesible solamente por los métodos de la propia clase.
    - **protected**: una variable miembro declarada **protected** está accesible para las clases del mismo paquete y las subclases (independientemente del paquete al que pertenezcan).
    - Si se omite el modificador de acceso, la variable tendrá el **acceso predeterminado/por defecto**, que establece que puede ser accedida por cualquier clase perteneciente al mismo paquete. No corresponde a ninguno de los anteriores

# Variables miembro (III)



- Modificador de instancia/clase
  - **static**: Una variable que no se declara **static** es una **variable de instancia**, y existe una copia distinta para cada objeto de la clase. Una variable que se declara **static** se llama **variable de clase** y solo existe una copia que es compartida por todos los objetos de la clase. (Ver apartado "Variables miembro de clase").
- Modificadores final, trasient y volatile
  - **final**: Una variable declarada **final** no puede cambiar su valor durante la ejecución del programa, es decir, se comporta como una constante
  - **transient**: indica que la variable no es persistente.
  - **volatile**: indica que la variable puede ser utilizada por distintos hilos sincronizados.

# Variables miembro (IV)



- Variables miembro de clase
  - Son variables de las que **sólo existe una copia** que es **compartida** por **todos los objetos de la clase**.
  - Se definen agregando a la definición de la variable la palabra reservada **static**.
  - Una variable miembro de clase **existe y se puede utilizar aunque no haya objetos** de la clase.
  - Para **acceder** a ellas:
    - Desde **dentro** de la clase: de la misma manera que con una variable miembro de instancia.
    - Desde **fuera** de la clase:
      - Nombre de la clase junto con la notación punto.
      - Se puede utilizar un nombre de objeto pero se suele usar el de la clase.
  - Inicialización:
    - Siempre antes que cualquier objeto de la clase.
    - Si no se les asigna explícitamente un valor:
      - Para los tipos primitivos: valores por defecto
      - Se usa **null** si es una referencia.

# Variables miembro (V)



- **Ejemplo:** la clase *Rectangulo* tiene una variable miembro de clase, *numRectangulos* para almacenar el número de objetos creados. La variable *numRectangulos* se incrementa cada vez que se crea un objeto de la clase (en el constructor).
  Desde fuera de la clase, se comprueba el valor de la variable *numRectangulos* antes de crear el primer objeto de la clase, y después de crear algunos de ellos.

```
public class Rectangulo{
 public static int numRectangulos;
 private double ladoA, ladoB;

 public Rectangulo (double lA, double lB)
     {
         ladoA = lA;
         ladoB = lB;
         numRectangulos++;
     }
}
```

Declaración de la variable de clase (static)

Declaración de las variables privadas que van a almacenar los dos lados del rectángulo

Incremento de la variable de clase *numRectangulos* cada vez que se crea un objeto de la clase

# Variables miembro (VI)



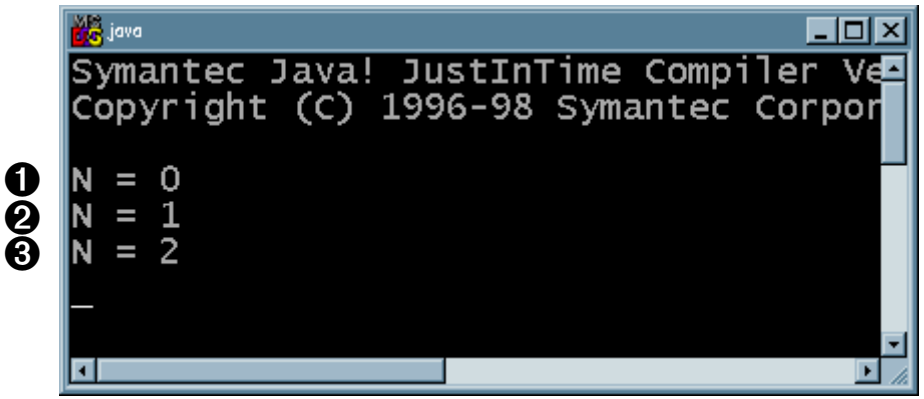
```
public class MainClass
{
    public static void main (String[]  args)
    {
❶   System.out.println("N = " + Rectangulo.numRectangulos);

        Rectangulo rec1 = new Rectangulo(3.0,3.0);
❷   System.out.println("N = " + Rectangulo.numRectangulos);

        Rectangulo rec2 = new Rectangulo(5.0,5.0);
❸   System.out.println("N = " + rec2.numRectangulos);
     }
}
```

Clase de pruebas

Se puede consultar la variable de clase aunque aún no se haya creado ningún objeto de la clase

Acceso a la variable de clase con el identificador de la clase

Acceso a la variable de clase a través del objrto *rec2*



# La referencia **this**



- La referencia **this** representa dentro de la definición de un método de instancia (no de clase) al propio objeto. Por lo tanto, el tipo de referencia es el de la clase que contiene al método en el que aparece la palabra reservada **this**.
- Cada vez que un método hace referencia a las variables de instancia de su clase o a otros métodos de la misma, aparece implícitamente delante de la referencia la expresión "**this**."
- Distintos usos de **this**:
  - Para referenciar las variables o métodos de instancia con la notación punto.
  - Para invocar a un constructor desde otro constructor de la misma clase (Ver apartado "Constructores")
  - Cuando un objeto necesita pasarse a sí mismo como argumento de un método de otro objeto.

# Métodos (I)



- Concepto
  - Los métodos (también llamados funciones miembro) son funciones definidas dentro de las clases.
  - Un método se aplica a un objeto de la clase mediante el operador punto y dicho objeto es el **argumento implícito** del método. Además también puede tener **argumentos explícitos**.

- Sintaxis de declaración de métodos

**[modificadores] tipoRetorno Identificador( [parametros] ) [excepciones] cuerpoMetodo**

- Modificadores
  - Modificadores de acceso
    - **public**: accesible para cualquier otra clase o subclase.
    - **private**: accesible solamente para la propia clase.
    - **protected**: accesible para clases del mismo paquete y para subclases.

# Métodos (II)



- Modificador de instancia/clase
  - **static**: Un método que no se declara **static** es un **método de instancia**, y sólo se puede invocar sobre un objeto de la clase. Un método que se declara **static** se llama **método de clase** y no es necesario invocarlo sobre objetos de la clase. (Ver apartado "Métodos de clase").
- Modificadores abstract, final, native y synchronized
  - **abstract**: Un método declarado **abstract** no proporciona implementación, se sustituye el cuerpo del método por un punto y coma. Sólo se pueden declarar métodos abstractos dentro de clases que también sean abstractas, si no, se genera un error de compilación. Un método privado no puede ser declarado abstracto ya que al ser privado no es visible por las subclases, entonces no se puede sobrescribir, y por lo tanto nunca se podría utilizar.
  - **final**: Un método declarado final no puede ser sobrescrito.
  - **native**: Un método declarado **native** no incluye implementación (se sustituye el cuerpo del método por un punto y coma). El código deberá estar en una librería dinámica (DLL). Este es el mecanismo para poder utilizar desde Java funciones implementadas en otros lenguajes.
  - **synchronized**: este concepto se estudiará en el tema de hilos.

# Métodos (III)



- Tipo de retorno
  - Un método puede devolver un tipo primitivo, una referencia a un objeto, o no devolver nada, en cuyo caso se especifica con la palabra reservada **void**.
  - Si un método devuelve una matriz, debe incorporarse un par de corchetes a continuación del tipo de datos de la matriz.
  - **Ejemplos:**
    - Un método que no devuelve nada

      ```
      void mostrarMensaje(String msg)
      ```

    - Un método que devuelve un tipo primitivo

      ```
      int factorial (int n)
      ```

    - Un método que devuelve una referencia a un objeto

      ```
      Fecha diaActual ()
      ```

    - Un método que devuelve una matriz (1D) de enteros

      ```
      int[] generarMatrizEntera1D (int n)
      ```

    - Un método que devuelve una matriz (2D) de booleanos

      ```
      boolean[] [] generarMatrizBooleana2D(int n)
      ```

# Métodos (IV)



- Identificador
  - El nombre del método debe de ser un identificador válido en Java.

- Parámetros
  - Los parámetros de un método se especifican mediante una lista de identificadores con sus tipos, separados por comas. Si el método no tiene parámetros, sólo aparece un par de paréntesis a continuación del nombre del método.

- Cláusula excepciones
  - La palabra reservada **throws** permite a un método declarar la lista de excepciones que puede lanzar. Esta lista consiste en los nombres de las clases de excepción separadas por comas. (Este tema se estudiará en detalle en el capítulo de excepciones).
  - **Ejemplo:**

```
void mostrarMensaje(String msg) throws IOException
```

# Métodos (V)



- Cuerpo del método
  - El cuerpo del método es un bloque (colección de sentencias simples incluidas entre llaves) que tiene el siguiente aspecto:

```
{
  declaraciones de variables locales
  sentencias
  [return [(] exprexion [)] ]
}
```

  - La sentencia **return** puede o no ser la última, y puede aparecer más de una vez en el cuerpo del método. Si el método no devuelve nada, se puede omitir, o especificar simplemente **return**.

# Métodos (VI)



- Métodos de clase
  - Métodos que no actúan sobre un objeto concreto y por lo tanto, no tienen argumento implícito.
  - Se definen agregando a la definición la palabra reservada **static**.
  - Para llamarlos se utiliza el nombre de la clase (de la misma manera que se hace con las variables miembro de clase).
  - Los métodos de clase solo pueden hacer referencia a variables y métodos que también sean de clase.
  - Cualquier método **static** es implícitamente **final**.
  - Un método de clase no se puede declarar **abstract**.
  - **Ejemplo:** en la clase **System** todos los métodos son de clase
    - `public static void exit(int status)` termina la ejecución de la máquina virtual Java.
    - `public static void gc()` ejecuta el recolector de basura.

# Métodos (VII)



- Sobrecarga de métodos (*overloading*)
  - Java permite la sobrecarga de métodos, es decir, la definición de métodos de la misma clase que tienen el mismo identificador pero que difieren en el número y/o tipo de los argumentos. (No se considera sobrecarga cuando los métodos solo difieren en el valor de retorno).
  - **Ejemplo:** la clase **PrintStream** del paquete **java.io** implementa muchas formas del método **println**:
    - public void println (int x)
    - public void println (boolean x)
    - public void println ()
  - Para seleccionar el método que se va a invocar, Java aplica las siguientes reglas:
    - si existe un método cuyos parámetros formales se ajustan exactamente a los parámetros actulaes, se invoca ese método.
    - si no ocurre el caso anterior, se intenta promover los parámetros actuales al tipo inmediatamente superior (**int** a **long**, **float** a **double**, etc) y se busca un método cuyos parámetros se ajusten a los nuevos tipos.
    - Si no se puede aplicar ninguna de las reglas anteriores, se produce un error.

# Métodos (VIII)



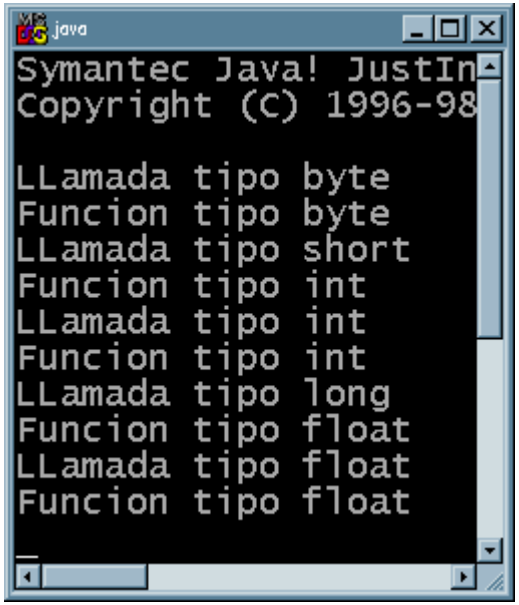
- **Ejemplo:** la clase *A* tiene sobrecargado el método *f*. Esta función tiene un único parámetro, y muestra un mensaje indicando el tipo de dicho parámetro.
  Desde fuera de la clase, se observa la aplicación de las reglas que definen cuál de los métodos se ejecuta dependiendo del tipo del parámetro de la llamada. En algunos casos el tipo del parámetro actual coincide con el tipo del parámetro formal, y en otros casos se promueve el tipo del parámetro actual.

```
public class A
{
   public void f ( byte a) { System.out.println("Funcion tipo byte");};
   public void f ( int a) { System.out.println("Funcion tipo int");};
   public void f ( float a) { System.out.println("Funcion tipo float");};
}
```

# Métodos (IX)



```
public class MainClass
{
    A a =  new A();
    byte unByte = 1;
    short unShort = 1;
    int unInt = 1;
    long unLong = 1;
    float unFloat = 1;
    System.out.println("LLamada tipo byte");
    a.f(unByte);
    System.out.println("LLamada tipo short");
    a.f(unShort);
    System.out.println("LLamada tipo int");
    a.f(unInt);
    System.out.println("LLamada tipo long");
    a.f(unLong);
    System.out.println("LLamada tipo float");
    a.f(unFloat);
}
```




❶ Hay un método *f* para **byte** ❶

❷ Promoción de **short** a **int** ❷

❸ Hay un método *f* para **int** ❸

❹ Promoción de **long** a **float** ❹

❺ Hay un método *f* para **float** ❺

# Métodos (X)



- Paso de parámetros a los métodos
  - *objetos:* los parámetros que son objetos, son siempre referencias, y por lo tanto, cualquier modificación que se haga a esos objetos dentro del método afecta al objeto original.
    Las matrices también son tipos referenciados, y por lo tanto también se pasan por referencia.
  - *Tipos primitivos:* los parámetros que son tipos primitivos se pasan siempre por valor, es decir, se pasa una copia, y como consecuencia cualquier modificación que se haga dentro del método no afecta a la variable original.
    La forma de modificar el valor de un parámetro de un tipo primitivo desde dentro de un método, es incluirlo como variable miembro de una clase, y pasar al método como parámetro un objeto de dicha clase.
    También se puede utilizar una matriz de una posición para almacenar el tipo primitivo, y pasar la matriz como parámetro del método.