



CLASES Y OBJETOS

PROGRAMACIÓN ORIENTADA A OBJETOS

JAVA

¿QUÉ ES JAVA?

- ✓ Lenguaje de programación orientada a objetos de proposito general:

Independiente de arquitectura.

Sus programas se ejecutan en cualquier plataforma.



CARACTERÍSTICAS DEL LENGUAJE

- ✓ Modela sistemas en clases y objetos.
- ✓ Requiere Java Development Kit (JDK):
Provee herramientas y librerías para el desarrollo
programas en JAVA
- ✓ Manejo de Hilos (Thread)

IDE DE DESARROLLO

✓ Entornos de desarrollo IDE:

- NetBeans
- Eclipse
- JCreator



NetBeans



JCreator

eSoftner



eclipse

API JAVA

✓ API de JAVA:

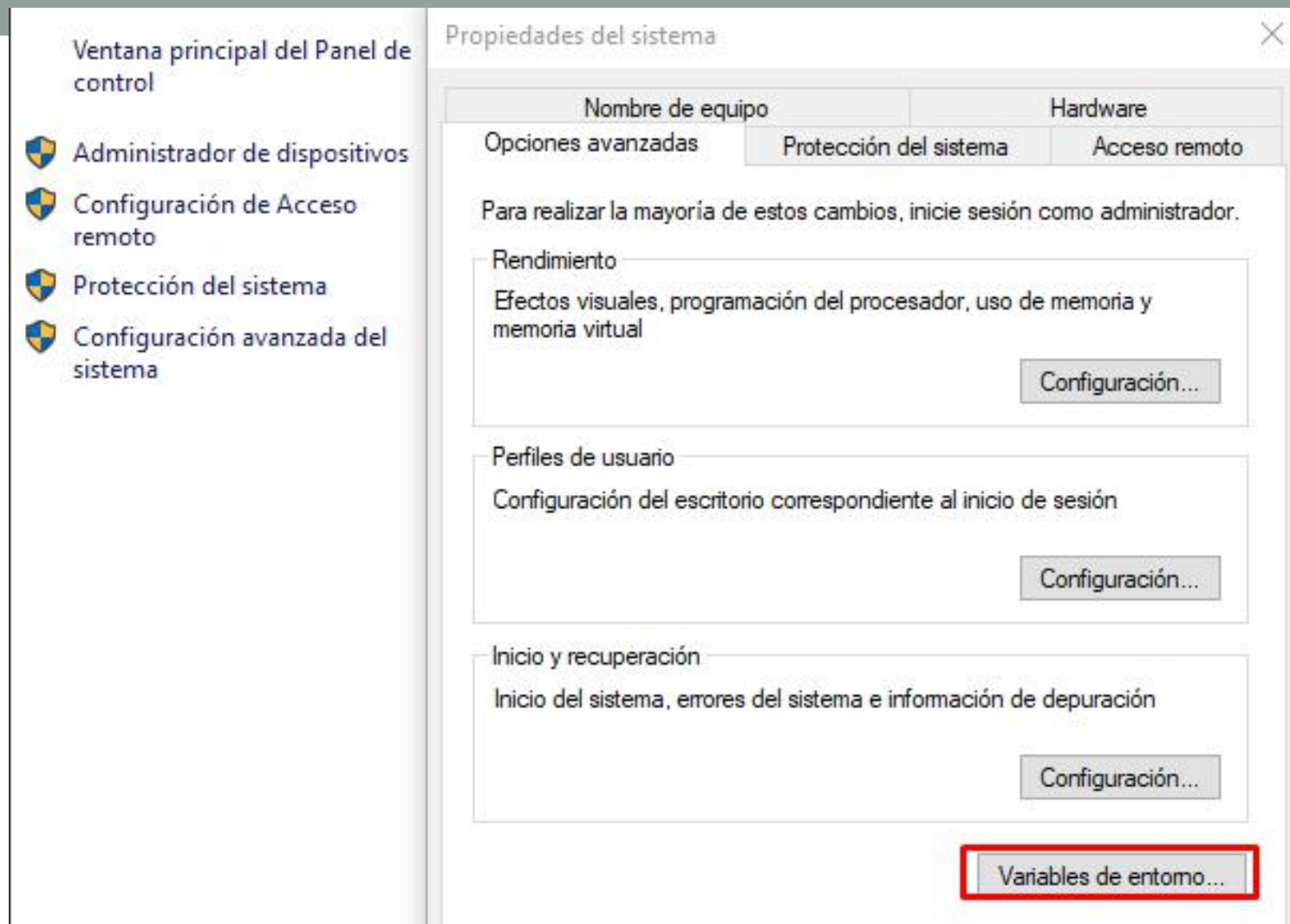
Conjunto de librerías de código Java compilado o clases ya listas para ser usadas por los desarrolladores.

<https://docs.oracle.com/javase/8/docs/api/>

CONFIGURAR VARIABLES DE ENTORNO

- ✓ Inicio → panel de control → sistema.
- ✓ Configuración avanzada del sistema → pestaña opciones avanzadas variables de entorno.

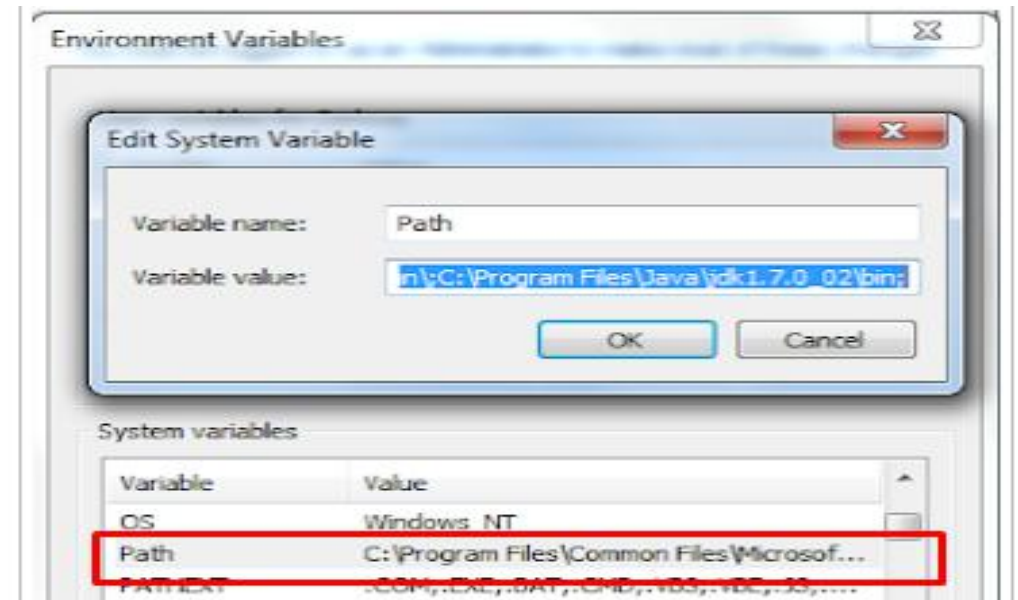
CONFIGURAR VARIABLES DE ENTORNO



CONFIGURAR VARIABLES DE ENTORNO

- ✓ Editaremos el Path, nos vamos hasta el final del texto que ya está escrito y agregamos la ruta del JDK instalado:

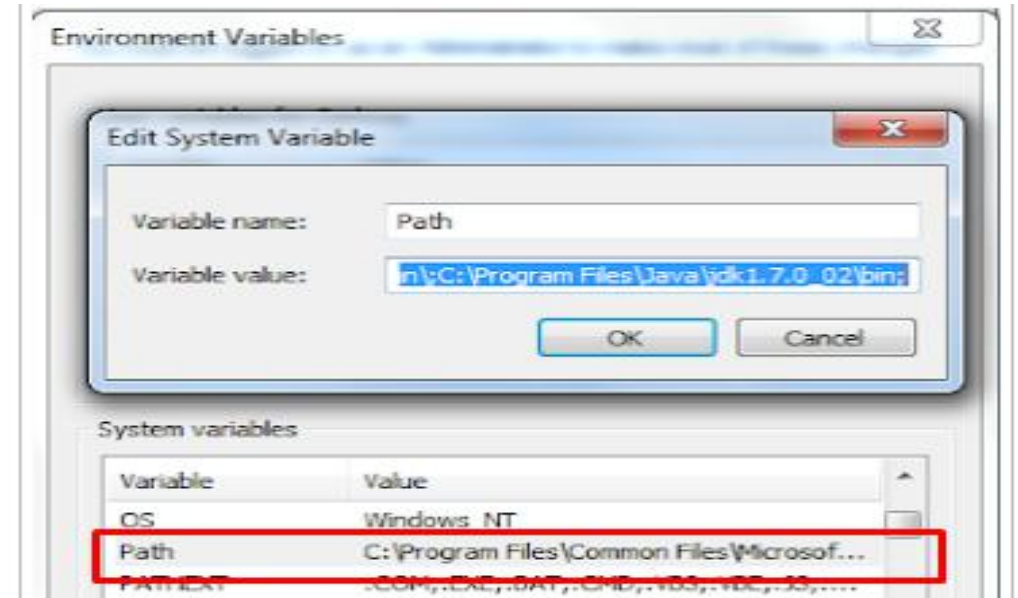
;C:\Program Files\Java\jdk1.8.0_111\bin;



CONFIGURAR VARIABLES DE ENTORNO

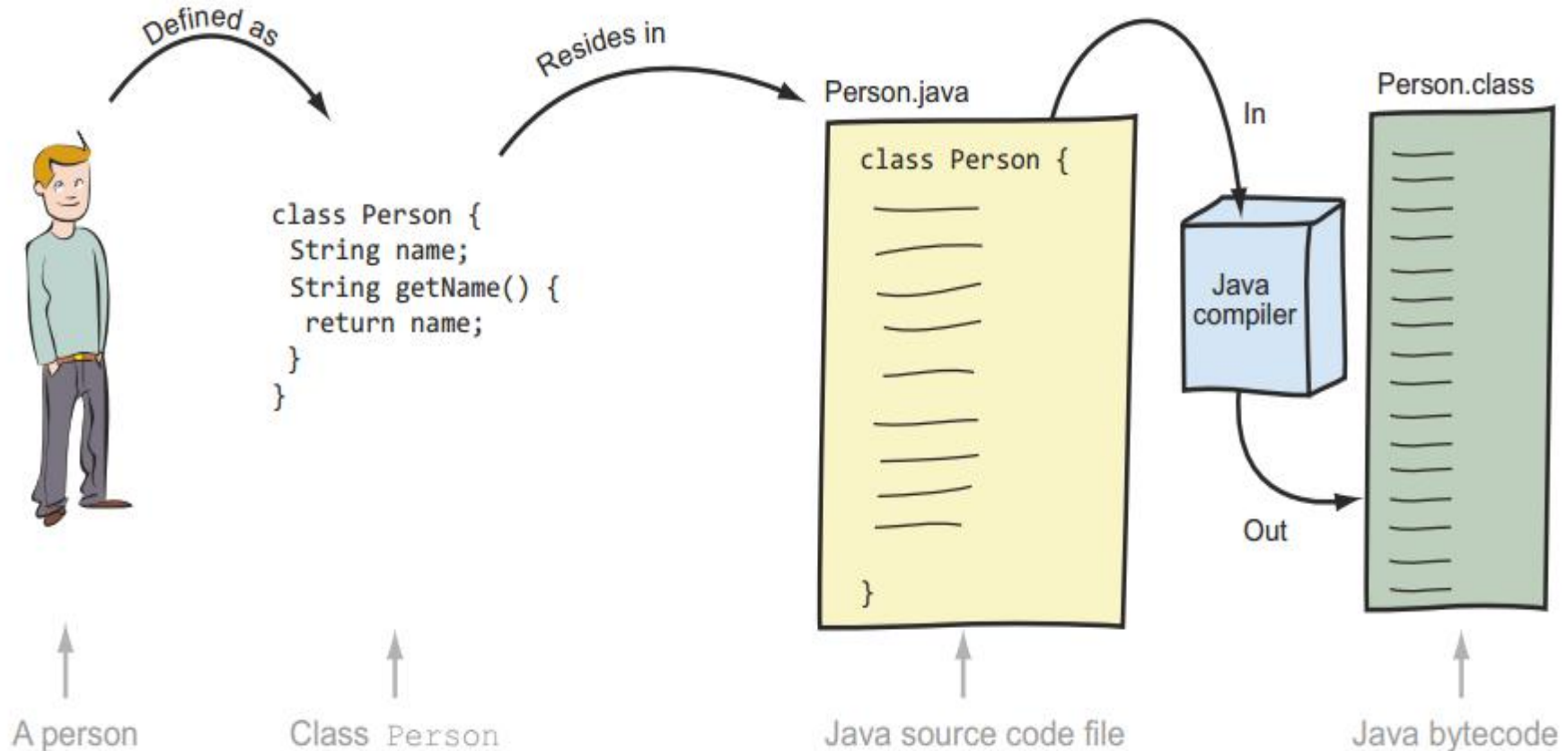
- ✓ Editaremos el Path, nos vamos hasta el final del texto que ya está escrito y agregamos la ruta del JDK instalado:

;C:\Program Files\Java\jdk1.8.0_111\bin;



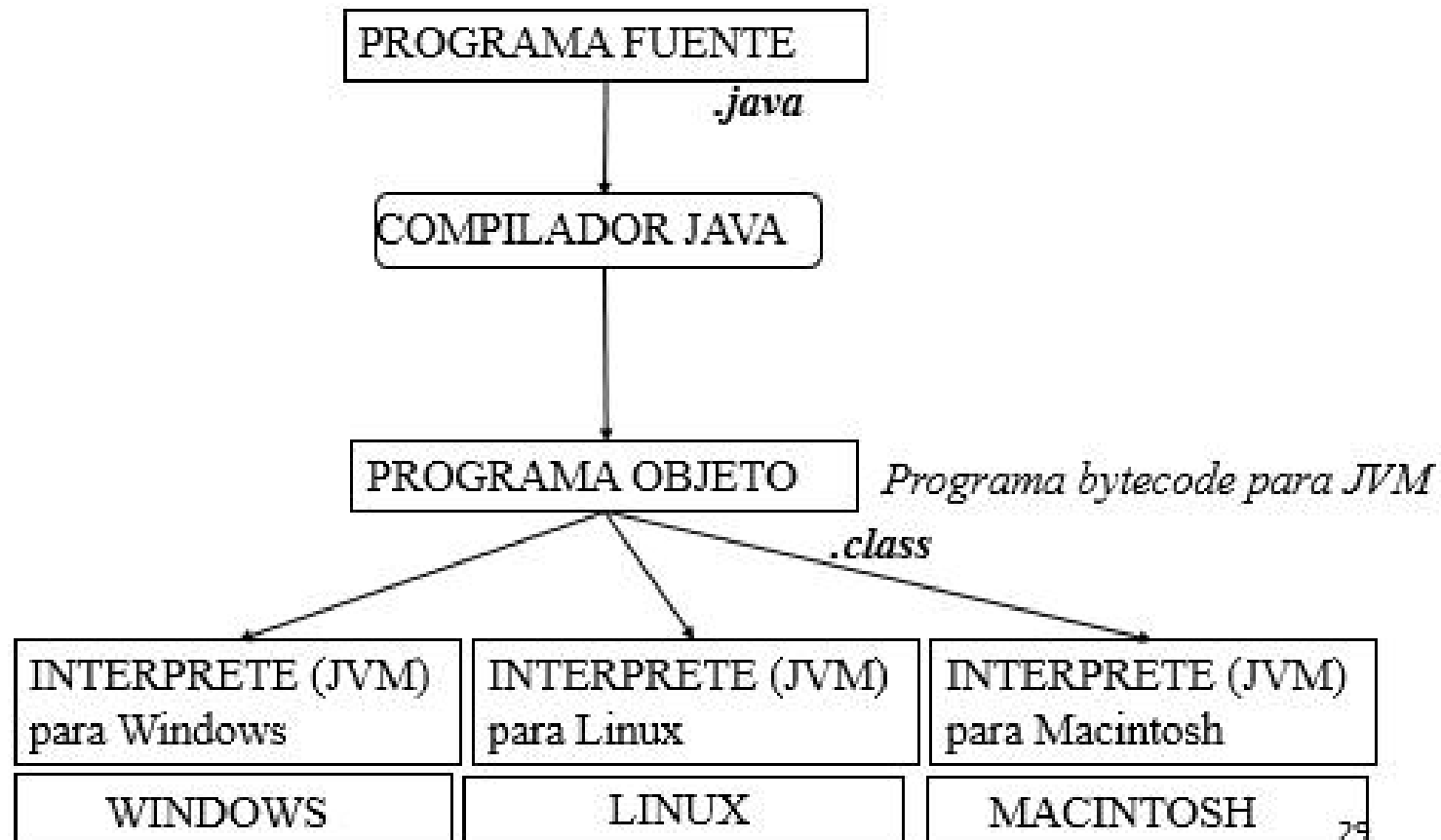
ESTRUCTURA CÓDIGO JAVA

✓ Abstracción



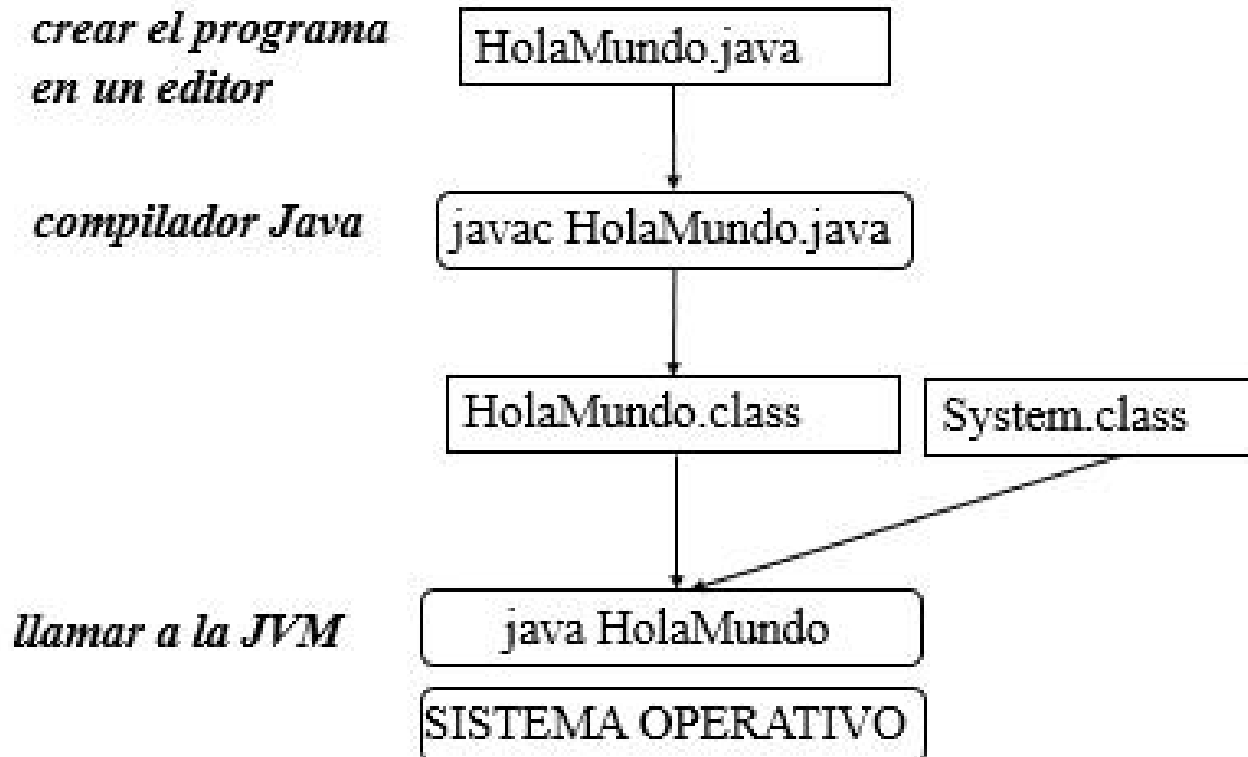
ESTRUCTURA CÓDIGO JAVA

✓ Máquina Virtual Java (JVM)



ESTRUCTURA CÓDIGO JAVA

✓ Ejecución programa Java



ESTRUCTURA CÓDIGO JAVA

- ✓ Código fuente: `.java`
- ✓ El compilador de java traduce el código fuente en código de byte: `.class` (Comando `javac`).
- ✓ Un intérprete específico interpreta el código byte para ejecutar el programa JVM-Java Virtual Machine. (Comando `java`).

COMPILAR CÓDIGO FUENTE

 Ejemplo.java: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
/* Primer Ejemplo Hola Mundo */
```

```
public class Ejemplo {  
    public static void main(String[ ] arg) {  
        System.out.println("Hola Java");  
    }  
}
```

COMPILAR CÓDIGO FUENTE

```
C:\ejercicios>
C:\ejercicios>cd C:\ejercicios
C:\ejercicios>javac Ejemplo.java
C:\ejercicios>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: A4E9-4D30

Directorio de C:\ejercicios

30/08/2018  10:09 p. m.      <DIR>          .
30/08/2018  10:09 p. m.      <DIR>          ..
30/08/2018  10:09 p. m.      417 Ejemplo.class
30/08/2018  09:36 p. m.      158 Ejemplo.java
                2 archivos          575 bytes
                2 dirs  130.894.209.024 bytes libres

C:\ejercicios>
```

COMPILAR CÓDIGO FUENTE

```
C:\ejercicios>  
C:\ejercicios>java Ejemplo  
Hola Java  
  
C:\ejercicios>
```


API DE JAVA

| | | | | | | | | | | | |
|-------------------|------------------------------|----------------------|-----------------------------------|-------------|-----------------------|---------------|-----------------------|---------------------|---------------|-----------------|--|
| Java Language | | Java Language | | | | | | | | | |
| Tools & Tool APIs | | java | javac | javadoc | apt | jar | javap | JPDA | JConsole | Java VisualVM | |
| | | Security | Int'l | RMI | IDL | Deploy | Monitoring | Troubleshoot | Scripting | JVM TI | |
| JDK | RIAs | Java Web Start | | | | | Applet / Java Plug-in | | | | |
| | User Interface Toolkits | AWT | | | | Swing | | | Java 2D | | |
| | | Accessibility | | Drag n Drop | | Input Methods | | Image I/O | Print Service | Sound | |
| | Integration Libraries | IDL | JDBC | JNDI | RMI | RMI-IIOP | | Scripting | | | |
| | Other Base Libraries | Beans | Intl Support | | Input/Output | | JMX | JNI | | Math | |
| | | Networking | Override Mechanism | | Security | | Serialization | Extension Mechanism | | XML JAXP | |
| | lang and util Base Libraries | lang and util | | Collections | Concurrency Utilities | | JAR | Logging | Management | | |
| | | Preferences API | | Ref Objects | Reflection | | Regular Expressions | Versioning | Zip | Instrumentation | |
| | | Java Virtual Machine | Java Hotspot Client and Server VM | | | | | | | | |

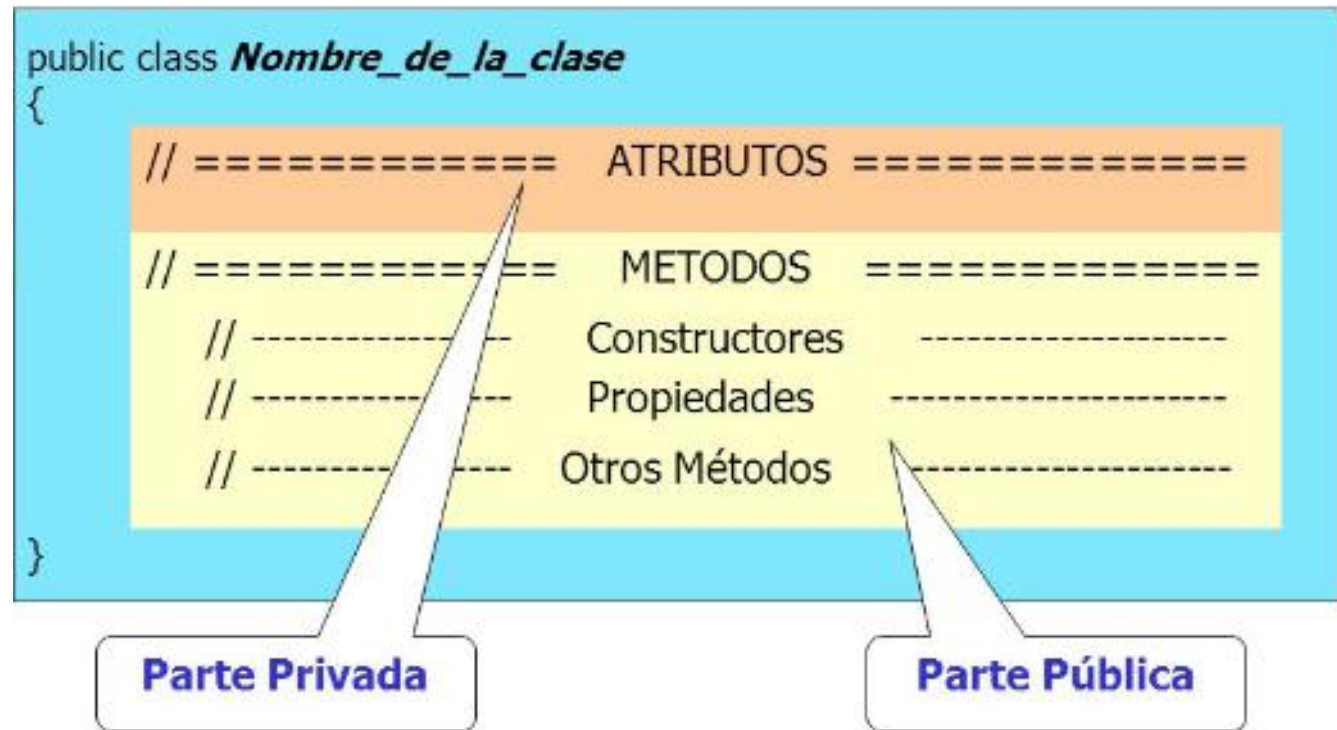
Java SE API

PAQUETES JAVA

- ✓ **java.lang**: para funciones del lenguaje
- ✓ **java.util**: para utilidades adicionales
- ✓ **java.io**: para manejo de ficheros
- ✓ **java.awt**: para diseño gráfico e interfaz de usuario
- ✓ **java.awt.event**: para gestionar eventos
- ✓ **javax.swing**: nuevo diseño de GUI
- ✓ **java.net**: para comunicaciones

IMPLEMENTACIÓN DE UNA CLASE

- ✓ Las clases son abstracciones que representan a un objeto que tienen el mismo comportamiento y características.



ESTRUCTURA DE UNA CLASE

```
1  package <package_name>;  
2  
3  import <other_packages>;  
4  
5  public class ClassName {  
6      |  
7      <variables o campos>;  
8  
9      <constructor (s)>;  
10  
11     <other methods>;  
12 }
```

1

2

3

4

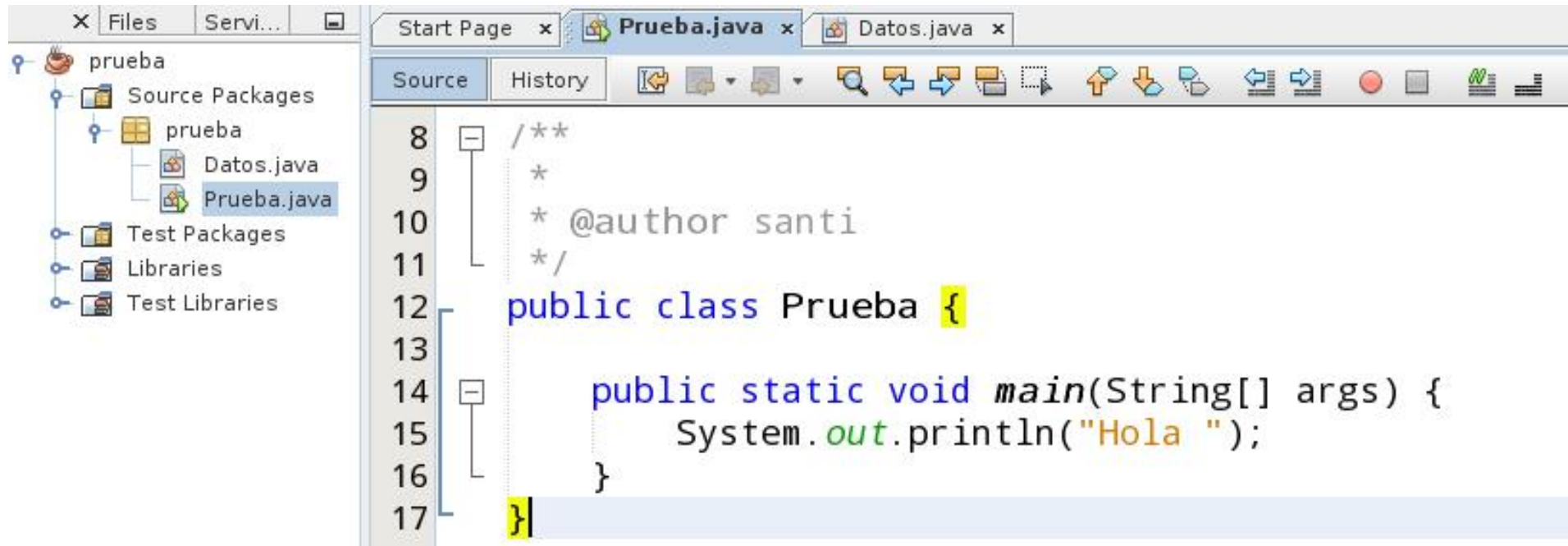
5

6

- ✓ Java **NO** soporta variables globales o funciones
- ✓ Todo hace parte de una clase

ESTRUCTURA DE UNA CLASE

- ✓ Una clase por lo general es de tipo **public**
- ✓ Un proyecto con multiples clases siempre tiene una clase principal:



The screenshot shows an IDE window with a project named 'prueba'. The left sidebar displays the project structure: 'Source Packages' containing 'prueba' (which includes 'Datos.java' and 'Prueba.java'), 'Test Packages', 'Libraries', and 'Test Libraries'. The main editor area shows the 'Prueba.java' file with the following code:

```
8  /**
9   *
10  * @author santi
11  */
12  public class Prueba {
13
14      public static void main(String[] args) {
15          System.out.println("Hola ");
16      }
17  }
```

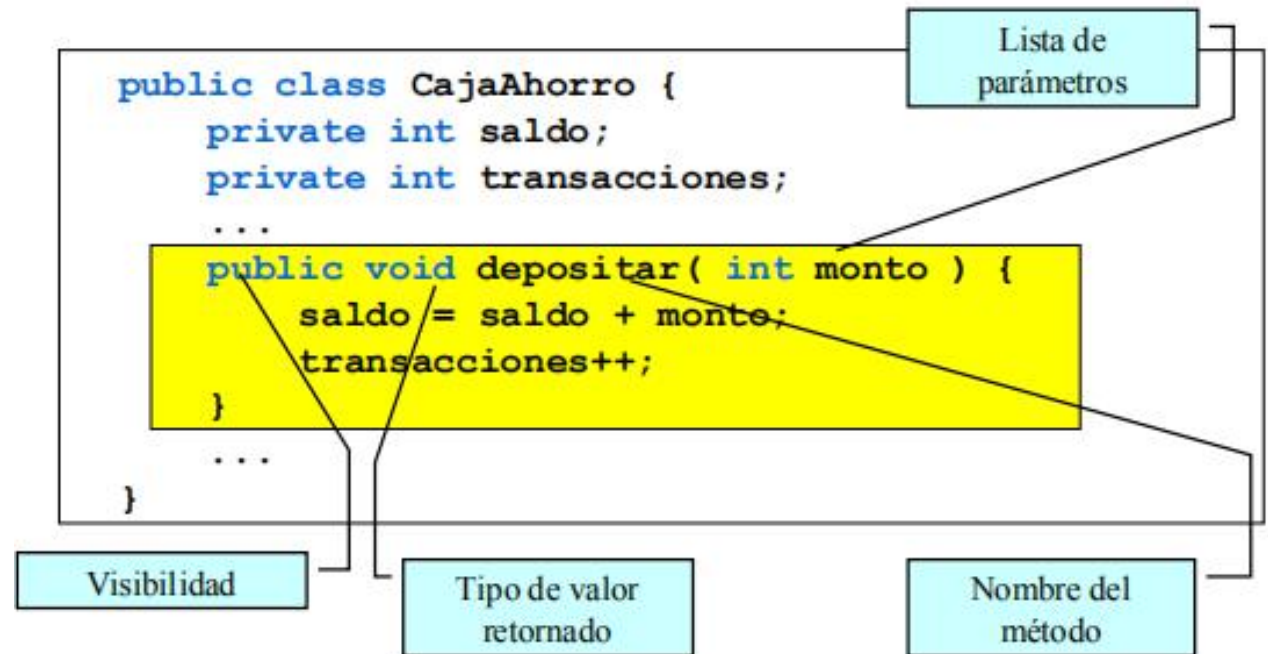
ESTRUCTURA DE UNA CLASE

Una clase define:

- ❖ **Atributos:** Determinan estado de una clase (variables de instancia de la clase)
- ❖ **Métodos:** Procesos que modifican el estado de un objeto. Los métodos pueden ser de tipo **proceso** o **retorno**
- ✓ El nombre de la clase empieza por **Mayúsculas**
- ✓ Atributos y métodos en minúsculas

MÉTODOS

- ❖ Conjunto de instrucciones que permiten a un objeto **reliazar una tarea** específica.
- ❖ Los métodos pueden o no retornar un valor.
- ❖ Modificador de **visibilidad** determina quien puede **utilizar** el método.



MÉTODOS

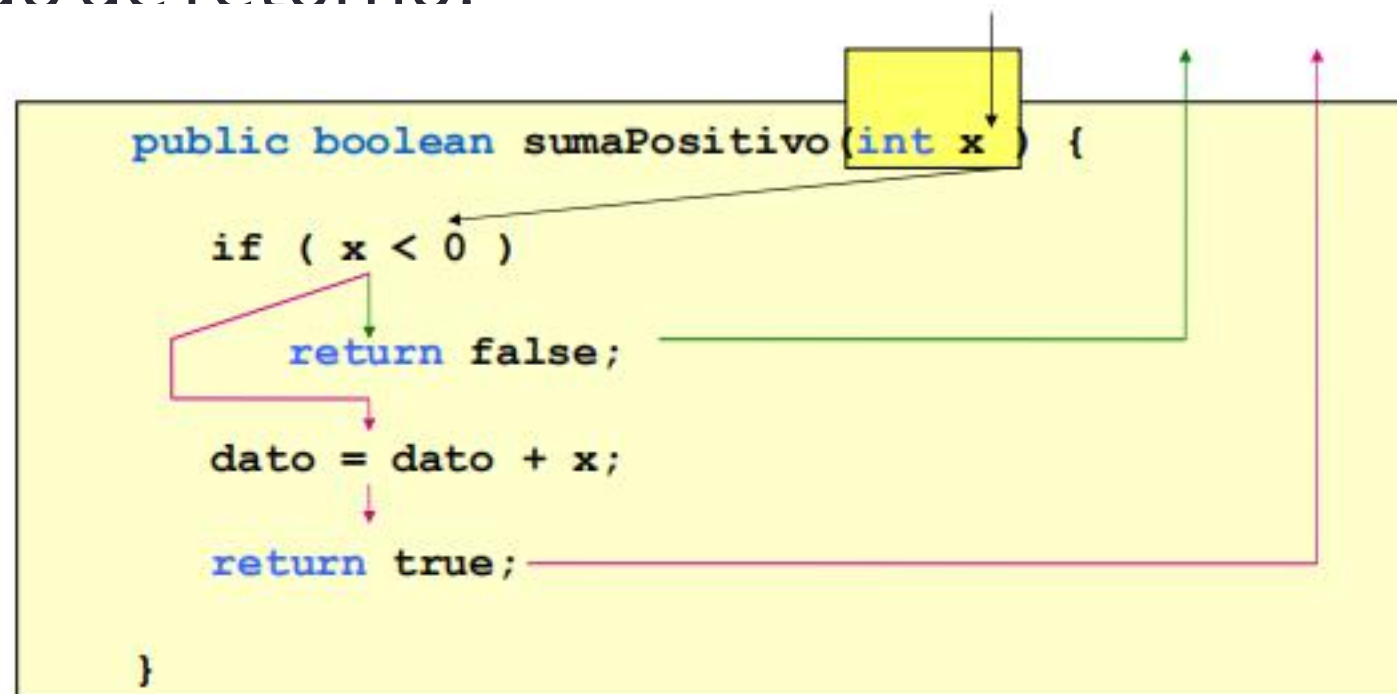
Tipos de valor de retorno de un método:

- ❖ **void**: No retornan un valor.
- ❖ **primitivo**: `int`, `double`, `boolean`, `char`, `long`
- ❖ **Clase**: `String`, `Cuenta`, `Biblioteca`, `Integer`
- ❖ **Arreglos primitivos o clase**: `String []`, `int []`, `Cuenta[]`

MÉTODOS

- Todos métodos que no son **void** deben terminar en **return**.

Ejemplo método de retorno.



MÉTODOS

- Un método **void** puede incluir la sentencia **return sin valor**, para interrumpir la ejecución del método.

```
public void sumaPositivo(int x ) {  
    if ( x < 0 )  
        return;  
    dato = dato + x;  
}
```

The diagram illustrates the execution flow of the `sumaPositivo` method. A black arrow points to the parameter `x` in the method signature. A blue arrow originates from the `return;` statement and points to the right, indicating the exit path from the method. A red arrow originates from the `if` statement, loops around the `return` statement, and points to the `dato = dato + x;` statement, indicating the flow when the condition is not met. Another red arrow points from the `dato = dato + x;` statement to the right, indicating the flow after the method body completes.

PARÁMETROS

- Son variables que **reciben** valores que requiere el método para cumplir su función.
- **No corresponde** a variables de instancia.
- No todos los métodos requieren parámetros.
- Forma de declarar parámetros:

```
( tipo1 var1, tipo2 var2, ..., tipoN varN )
```

PARÁMETROS

```
public class Reloj {  
    int hora, minuto;  
    boolean activado;  
    ...  
    public void setHora (int h, int s , boolean a){  
        hora = h;  
        minuto = s;  
        activado = a;  
    }  
    ...  
}
```

Parámetros formales

```
public class Aplicación {  
    ...  
    Reloj citizen = new Reloj();  
    int y = 8;  
    citizen.sethora( y, 20, true);  
    ...  
}
```

Parámetros actuales

PARÁMETROS

```
public class Tiempo {  
    private int minutos;  
  
    ...  
    public int sumaTiempo(int hora, int min) {  
        int aux;  
        aux= hora * 60 + min;  
        minutos = minutos + aux;  
    }  
    ...  
}
```

Parámetros

Variable de
instancia

Variable local



PROYECTO MÉTODOS

ESTRUCTURA DE UNA CLASE

✓ Método principal:

```
public static void main(String[] args) {  
    System.out.println("Hola ");  
}
```

TIPOS DE DATOS

VARIABLES DE TIPOS PRIMITIVOS.

| Nombre | Tipo | Tamaño | Valor por defecto | Forma de inicializar | Rango |
|---------|---------------|---------|-------------------|----------------------|--|
| Boolean | Lógico | 1 bit | False | Boolean a=true | True-false |
| Char | Carácter | 16 bits | Null | Char a='Z' | Unicode |
| Byte | Numero entero | 8 bits | 0 | Byte a =0 | -128 a 127 |
| Short | Numero entero | 16 bits | 0 | Short a =12 | -32.768 a 32.767 |
| Int | Numero entero | 32 bit | 0 | Int a= 1250 | -2.147.483.648 a 2.147.483.649 |
| Long | Numero entero | 64 bits | 0 | Long a= 125000 | -9*10 ¹⁸ a 9*10 ¹⁸ |
| Float | Numero real | 32 bits | 0 | Float a =3.1 | -3,4*10 ³⁸ a 3,4*10 ³⁸ |
| Double | Numero real | 64 bits | 0 | Double a = 125.2333 | -1,79*10 ³⁰⁸ a 1.79*10 ³⁰⁸ |

TIPOS DE DATOS

No primitivos (Objetos):

- ❖ Tipos de datos que tienen métodos propios para modificar su estado.
- ❖ También permiten almacenar un dato NULL

Ejemplo:

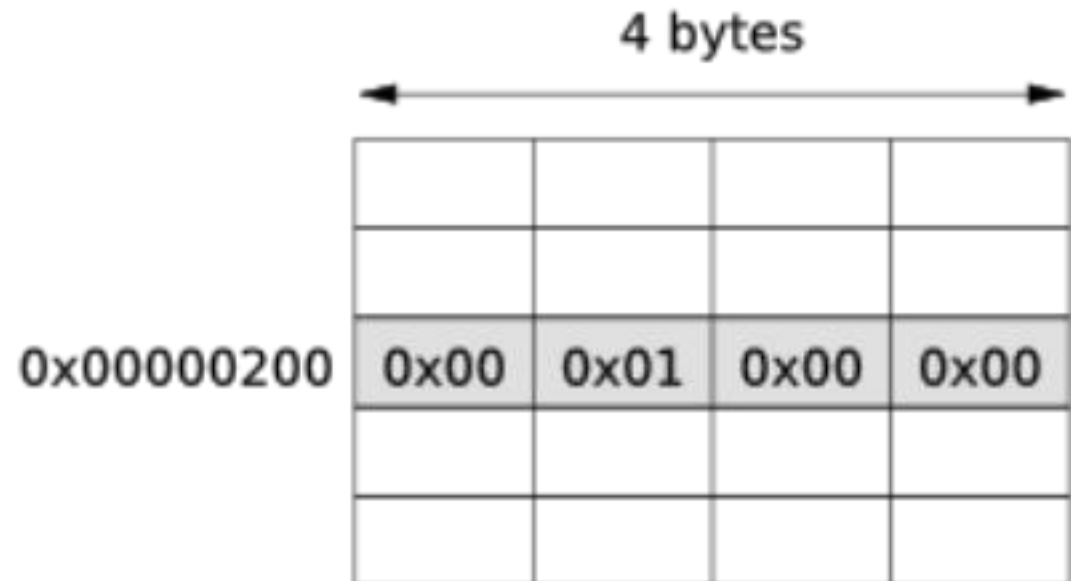
Integer: Enteros

String : Almacenar cadena de caracteres

VARIABLES

- ✓ Espacio de memoria del equipo para almacenar un valor que puede cambiar de estado durante la ejecución del programa.
- ✓ Almacena datos temporalmente utilizados por el programa.

Memoría RAM ----->



CONSTANTES

- ✓ Espacio de memoria del equipo para almacenar un valor que no va cambiar de estado durante la ejecución del programa.

Ejemplo:

`final <tipo _ dato> <nombre _ variable>`

```
final double num = 23.4;
```

OPERADORES

- ✓ Aritméticos: $+$, $-$, $*$, $/$
- ✓ Operadores Lógicos, relaciones y booleanos:

$<$, $>$, $!=$, $==$

$\&\&$ --> AND lógico

$\|\$ --> OR lógico

| OPERADOR | NOMBRE | EJEMPLO | DEVUELVE VERDADERO CUANDO... |
|----------|--------|------------------------|--|
| $\&\&$ | y | $(7 > 2) \&\& (2 < 4)$ | las dos condiciones son verdaderas |
| $\ \$ | o | $(7 > 2) \ \ (2 < 4)$ | al menos una de las condiciones es verdadera |
| $!$ | no | $!(7 > 2)$ | la condición es falsa |

ENTRADA Y SALIDA DE DATOS POR CONSOLA

- ✓ Importar librería Scanner
- ✓ Instanciar objeto clase Scanner

```
Scanner in = new Scanner(System.in);  
int numero = in.nextInt();
```

Nos permite capturar datos por teclado

- ❖ Cadena de caracteres

```
Scanner in = new Scanner(System.in);  
String nombre = in.nextLine();
```

PROYECTO 1

- ✓ Adivinar un número

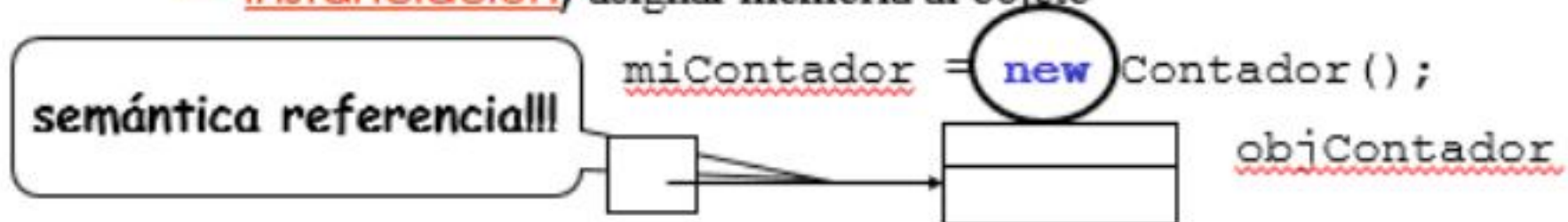
INSTANCIA OBJETO

- Un objeto es una instancia de una clase.
- La creación de un objeto se realiza en tres pasos

- Declaración, proporcionar un nombre al objeto

Contador miContador; (null)

- Instanciación, asignar memoria al objeto



- Inicialización, opcionalmente se pueden proporcionar valores iniciales a las variables de instancia del objeto en la declaración o mediante **CONSTRUCTORES**.

INSTANCIA OBJETO

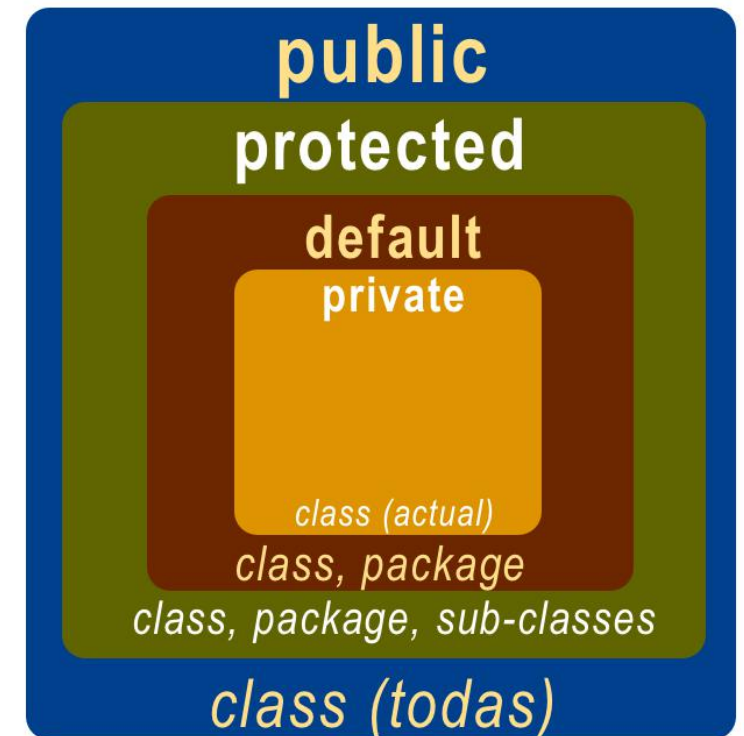


VISIBILIDAD - MODIFICADOR DE ACCESO

✓ Encapsulamiento:

Permite controlar el acceso a los datos que conforman un objeto o instancia.

Un modificador permite dar un nivel de seguridad retringiendo el acceso a atributos, métodos y constructores.



VISIBILIDAD - MODIFICADOR DE ACCESO

✓ Privado

Los atributos y métodos declarados con el modificador **private** solo son accesibles dentro de la clase.

✓ Default

Cuando no se especifica ningún modificador de acceso para una clase, método o atributo.

VISIBILIDAD - MODIFICADOR DE ACCESO

✓ **Protegido**

Los métodos o atributos declarados con el modificar **protected** son accesibles dentro del mismo paquete o sub-clases en paquetes diferentes.

✓ **Público**

Las clases, métodos o atributos miembros de datos que se declaran con el modificador **public** son accesibles desde cualquier lugar del programa.

PROYECTO 2

✓ Clase curso

ENCAPSULAMIENTO

- ✓ Los objetos encapsulan sus operaciones y estado.
- ❖ Los **métodos** definen el comportamiento del objeto.
- ❖ El estado del objeto está definido por los **atributos**.

Elementos necesarios para usar el objeto son visibles (public): **métodos**.

ENCAPSULAMIENTO

Toda clase debe permitir la comunicación entre las partes privadas y públicas:

Ejemplo: Para conducir no es necesario conocer los mecanismos de funcionamiento del carro : pedales, volante y palanca de cambio que son las **partes internas** (atributos) que comunica con las **partes externas** del carro (métodos)

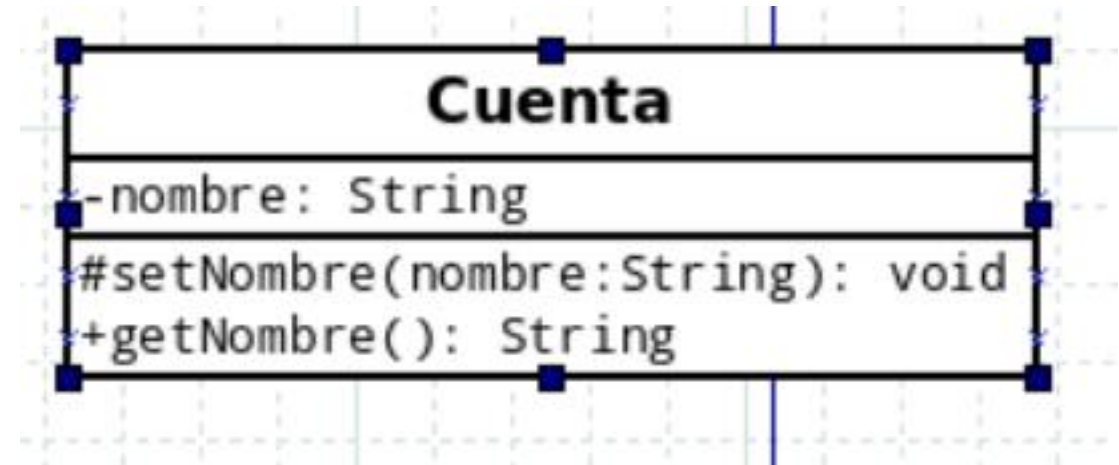
PROYECTO 3

✓ Clase cuenta

Diagrama UML

➤ Diagrama de clases:

Esquema estructura de un sistema que **modela** sus clases, atributos, operaciones y relaciones entre objetos.



ENTRADA DE DATOS - JOPTIONPANE

Ventana emergentes para ingresar datos y mostrar mensajes.

- **showInputDialog**: Capturar datos de teclado
- **showMessageDialog**: Mostrar mensajes de diálogo

MIEMBROS ESTÁTICOS DE UNA CLASE

Se define con **static**

- Miembros de clase, le **pertecene** a la clase y no al objeto.
- Todo cambio que se haga en ese atributo se cambia para todos los atributos de los objetos.
- Estos atributos **no requieren** ser instanciado para ser utilizados . Ejemplo: `Integer.parseInt();`

MIEMBROS ESTÁTICOS DE UNA CLASE

```
public class Estatico {  
    private static String frase = "Primera frase";  
  
    public static void main(String[] args){  
        Estatico ob1 = new Estatico();  
        Estatico ob2 = new Estatico();  
  
        ob2.frase = "Segunda frase";  
  
        System.out.println(ob1.frase);  
        System.out.println(ob2.frase);  
    }  
}
```

MIEMBROS ESTÁTICOS DE UNA CLASE

Ventana emergentes para ingresar datos y mostrar mensajes.

- **showInputDialog**: Capturar datos de teclado
- **showMessageDialog**: Mostrar mensajes de diálogo

CONSTRUCTOR

- ✓ Método utilizado para **inicializar** las variables de instancia de la clase.
- ✓ Asignación de memoria al objeto.
- ✓ No especifica **tipo**, ni **retorno de valores**.
- ✓ Es de **tipo** público
- ✓ Puede recibir valores por parámetros.
- ✓ Contienen las instrucciones que se ejecutan al momento de **crear una instancia** de una clase (Crear objeto).

```
class Coche(){  
    int numeroDeRuedas;  
  
    Coche(){  
        numeroDeRuedas = 4;  
    }  
}  
  
//uso  
new Coche();
```

CONSTRUCTOR

- ✓ Toda clase debe tener un constructor para realizar la instancia de objetos.
- ✓ Se definen con el mismo **nombre** de la clase.

Ejemplo: Contructor con parámetros y normal.

```
public class Reloj {  
    private int horas, minutos, segundos;  
  
    public Reloj(int hh, int mm, int ss) {  
        horas = hh;  
        minutos = mm;  
        segundos = ss;  
    }  
    ...  
}
```

```
public class CajaAhorro {  
    private int saldo;  
    private int transacciones;  
  
    public CajaAhorro() {  
        saldo = 0;  
        transacciones = 0;  
    }  
    ...  
}
```

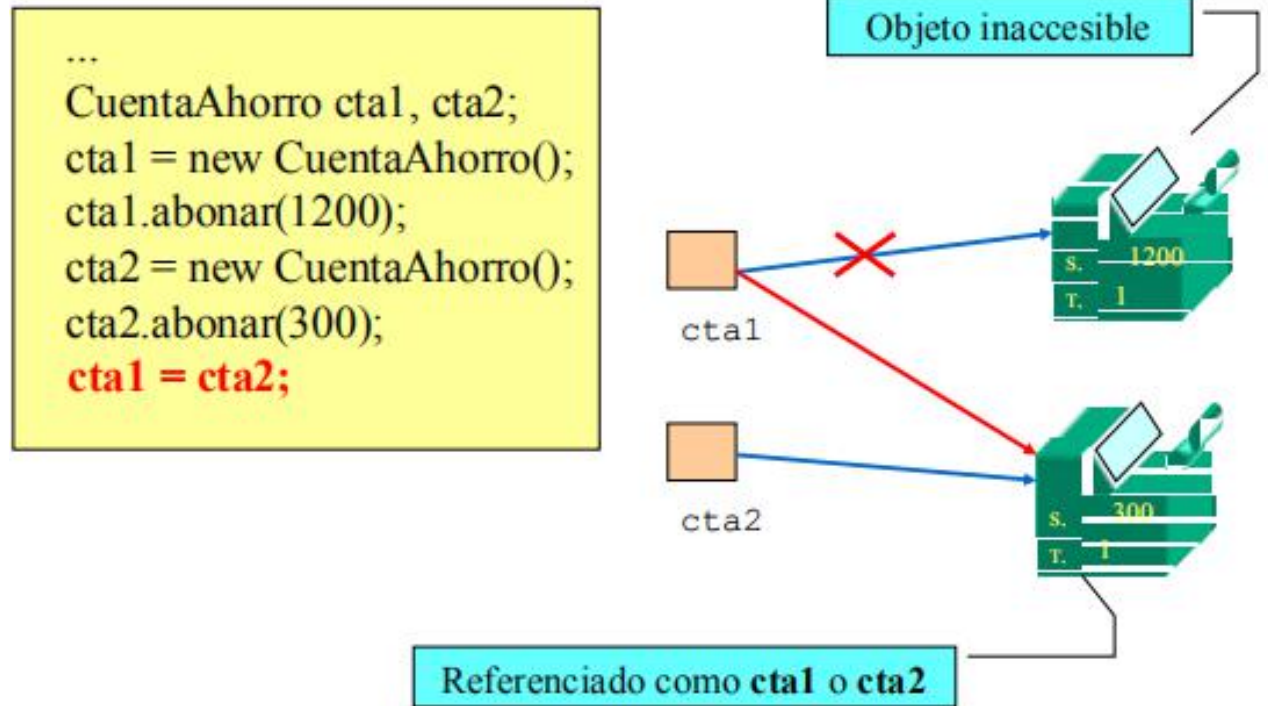
SOBRECARGA DE MÉTODOS

- ✓ Creación de varios métodos con el mismo **nombre** pero con **diferente** lista de parámetros.
- ✓ La definición de parámetros **define** que método ejecutar.
- ✓ También existe la **sobrecarga** de constructores.

```
public class Punto {  
    int x,y;  
  
    public Punto(){  
    }  
    public Punto(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public void imprimePunto() {  
        System.out.println("Punto [x=" + x + ", y=" + y + "]");  
    }  
}
```

REFERENCIA DE OBJETOS

- ✓ Cuando un objeto deja de ser referenciado se vuelve **inaccesible**.
- ✓ El recolector automático de basura de java lo **destruye** y **libera** memoria



REFERENCIA DE OBJETOS

- ✓ Gracias a la comparación de direcciones se puede realizar comparaciones de objetos.

```
CuentaAhorro cuenta1, cuenta2, cuenta3;  
cuenta1 = new CuentaAhorro();  
cuenta2 = new CuentaAhorro();  
cuenta1.abonar(1000);  
cuenta2.abonar(500);  
cuenta3 = cuenta2;  
Cuenta3.girar(50);
```

