

WOMEN WHO CODE - MÉRIDA \\ BEATRIZ SÁNCHEZ

INTRO TO JAVA

CONTENIDO

▶ Intro e Instalación

▶ Sintaxis

▶ Modelos de Objetos



"JAVA ES LA BASE PARA CASI TODO TIPO DE APLICACIONES DE RED Y ES EL ESTÁNDAR MUNDIAL PARA EL DESARROLLO Y DISTRIBUCIÓN DE SOFTWARE EMPRESARIAL, CONTENIDO WEB, JUEGOS Y APLICACIONES MÓVILES. JAVA DISFRUTA DE UN ECOSISTEMA GRANDE Y MADURO CON HERRAMIENTAS FUERTES. JAVA PROPORCIONA PORTABILIDAD DE LAS APLICACIONES Y EL RENDIMIENTO ROBUSTO EN MUCHOS ENTORNOS DE COMPUTACIÓN "

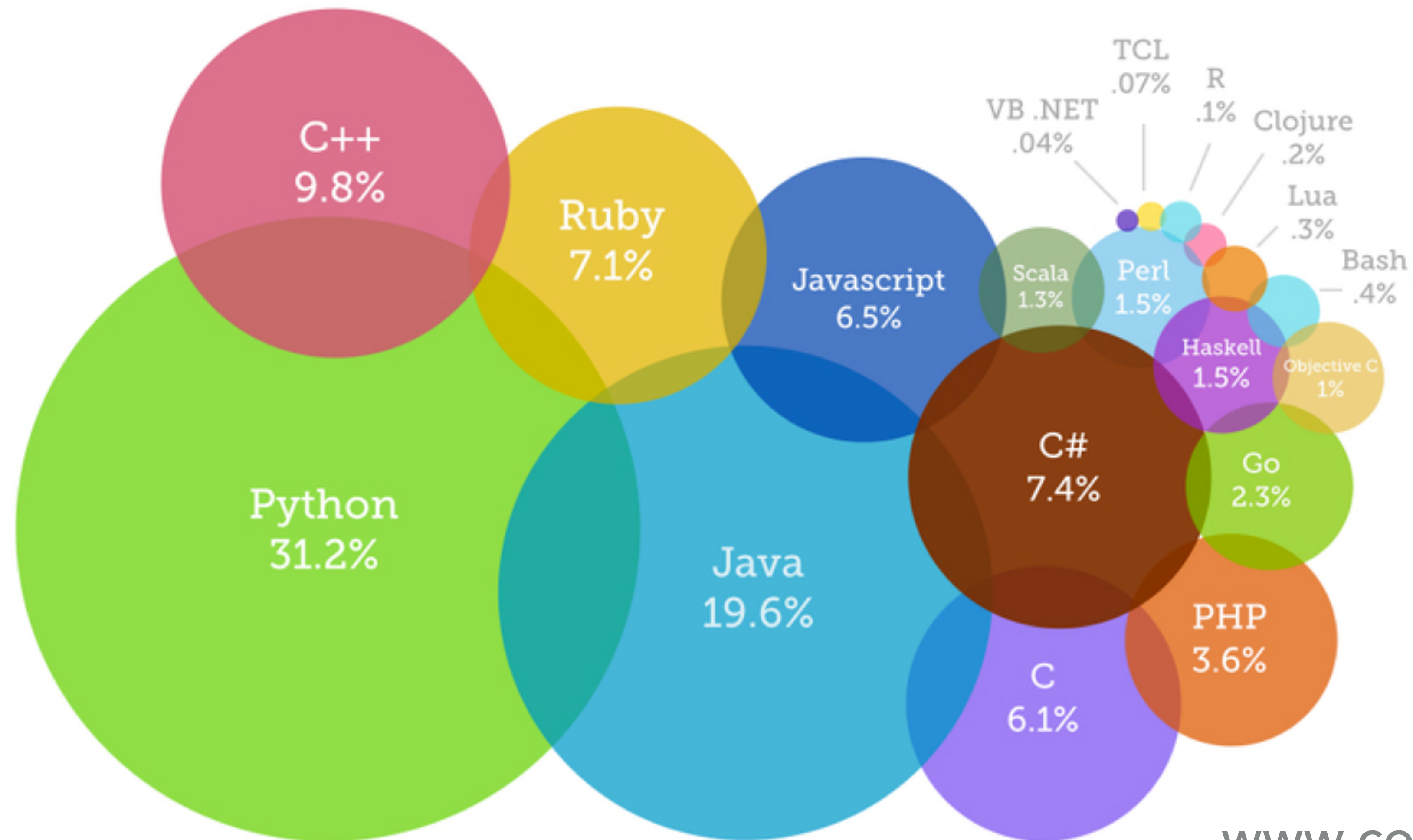
WWW.ORACLE.COM



INTRO TO JAVA 1.1

INTRO E INSTALACIÓN

LENGUAJES DE PROGRAMACION MAS POPULARES 2015



PRINCIPIOS BAJO LOS QUE SE DESARROLLO



James Gosling

- ▶ Debe ser "simple, orientado a objetos y familiar"
- ▶ Debe ser "robusto y seguro"
- ▶ Debe ser "de arquitectura neutral y portable"
- ▶ Debe ejecutarse con "alto rendimiento"
- ▶ Debe ser "interpretado, dinámico y multi-procesos"

INFO GENERAL

```
Class Triangle{  
    ...  
    float surface() {  
        return b*h/2;  
    }  
}
```

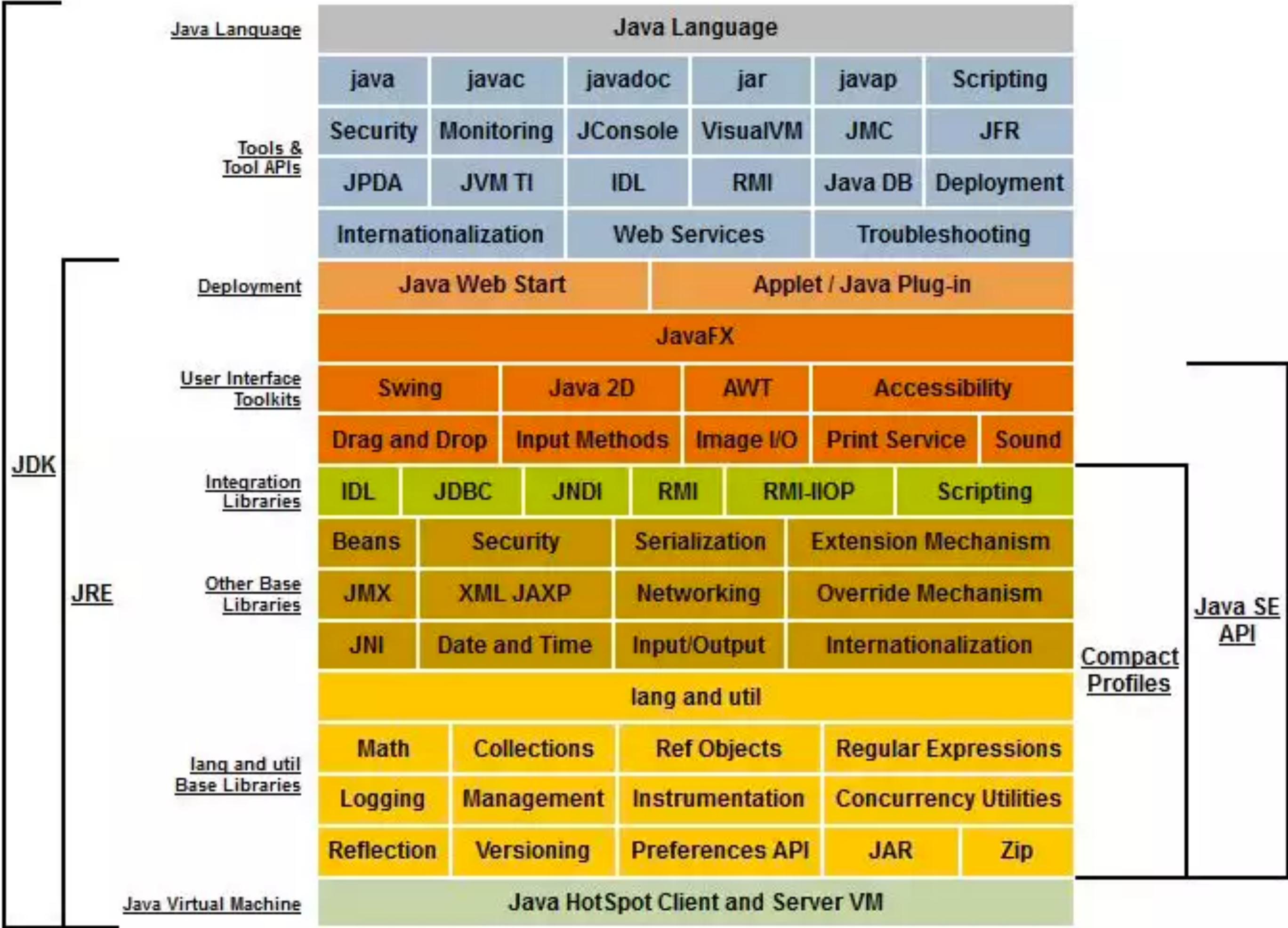
```
LOAD r1, b  
LOAD r2, h  
MUL r1, r2  
DIV r1, #2  
RET
```

```
00000100110001001  
11100011111000010  
00011010101010110
```

- ▶ James Gosling & Sun Microsystems 1991
- ▶ Lenguaje de alto nivel
- ▶ Manejo automático de memoria
- ▶ Multi-paradigma:
 - Orientado a objetos
 - Estructurado
 - Imperativo
 - Genérico
 - Reflexivo
 - Concurrente

API, JRE & JDK

- ▶ API (Application Programmable Interface)
 - J2EE (Java 2 Enterprise Edition)
 - J2SE (Java 2 Standard Edition)
 - J2ME (Java 2 Micro Edition)
- ▶ JRE (Java Runtime Environment)
- ▶ JDK (Java Development Kit)



INSTALACION

1. Descargar [Java SE JDK8](#) para el sistema operativo de la máquina (Aceptar la licencia)
2. Declarar las variables de entorno



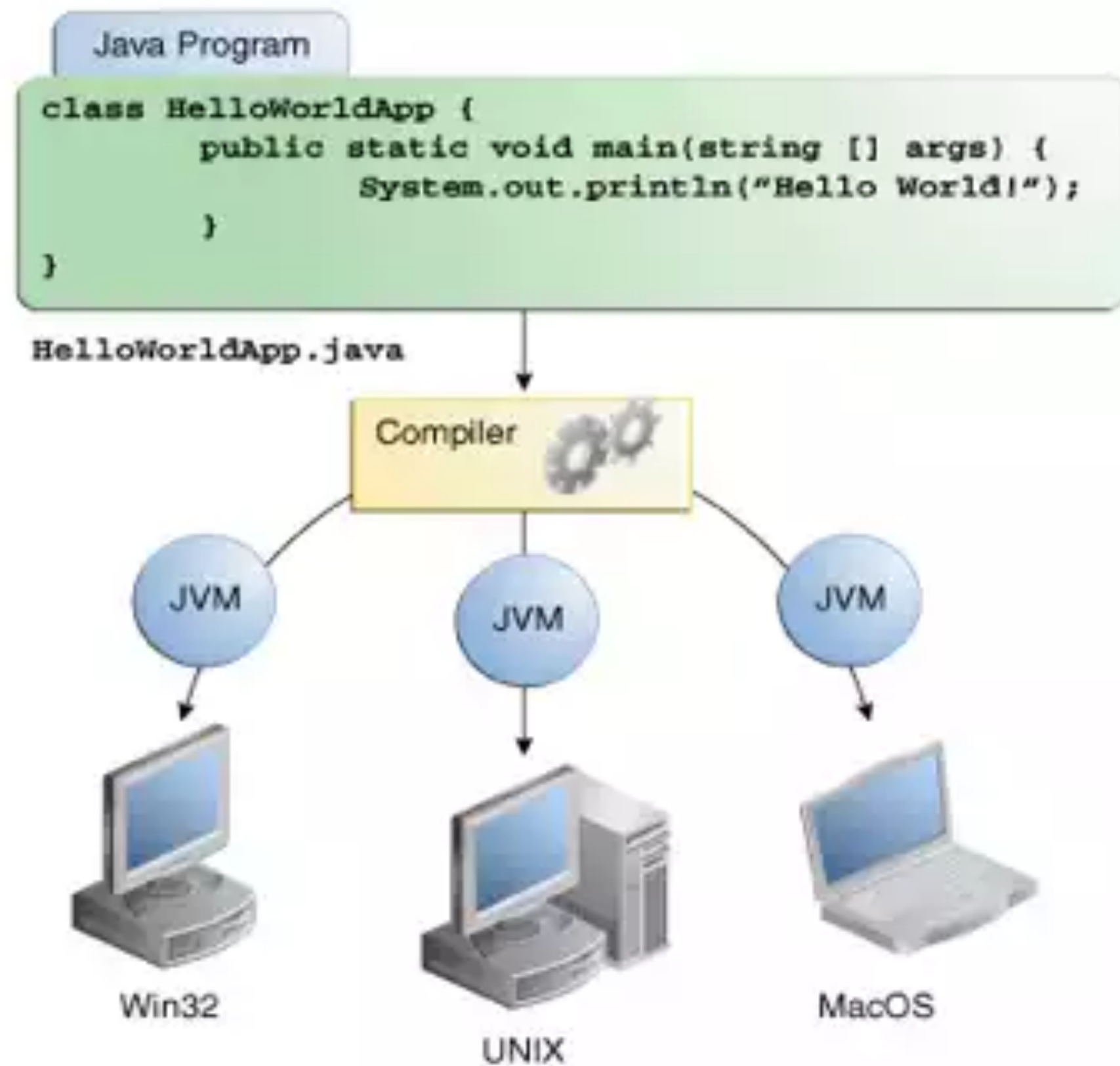
Windows

1. Click derecho en 'Mi PC' y seleccionar 'Propiedades', Click en botón 'Variables de Entorno' en la pestaña 'Avanzado'.
 2. Copiar el 'Path' y agregar al final la ruta hasta el bin del jdk (donde se instaló) ';c:\Program Files\java\jdk\bin'.
3. Escoger un editor de texto. Éste puede ser un IDE (Entorno de Desarrollo Integrado) como [Eclipse](#), Netbeans, IntelliJ.

UNIX

1. PATH debe apuntar donde los bin fueron instalados.
2. Si usa bash como consola, agregar al final del .bashrc
'export PATH=/path/to/java:\$PATH'

COMPILACION



Java maneja una compilación en dos pasos:

1. Compilación del código Java a bytecode

javac MiPrograma.java

2. La máquina virtual (JVM) convierte el bytecode a código máquina

a) Compilación Just-in-Time (JIT)

b) Interpretación del bytecode

java MiPrograma

HELLO WORLD!

HelloWorld.java

(Mismo nombre que la clase)

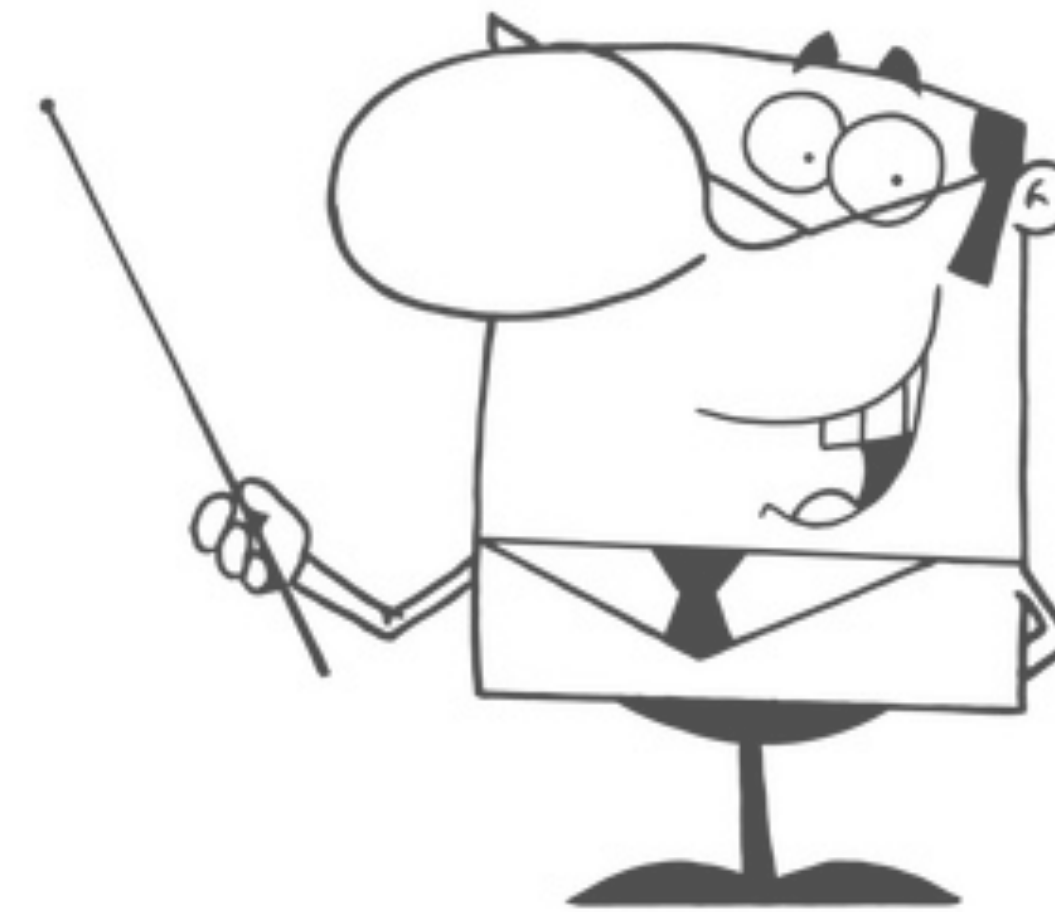
```
public class HelloWorld {  
    public static void main(String []args) {  
        System.out.println("Hello World");  
    }  
}
```

Terminal

```
>> javac HelloWorld.java  
>> java HelloWorld
```

INTRO TO JAVA 1.2

SINTAXIS



COMENTARIOS

```
public class HelloWorld {  
  
    /* Este es mi primer programa  
     * Escribirá "Hello World" en la consola  
     * Ejemplo de comentario multilinea  
     */  
  
    public static void main(String []args){  
        // Ejemplo de comentario de una sola línea  
        /* Que igual se puede hacer de esta forma */  
        System.out.println("Hello World");  
    }  
}
```

- ▶ Java ignora:
 - Comentarios
 - Lineas en blanco

TIPOS DE DATOS

- ▶ Primitivos
- ▶ Objeto/Referencia
 - El valor por defecto es null
 - Se crean utilizando constructores y sirven para acceder a objetos
 - Se caracterizan por: Clases, Métodos, Objetos , Instancias, Abstracción, Herencia , Polimorfismo, Encapsulación, Pase de mensajes

TIPOS DE DATOS PRIMITIVOS

TYPE	SIZE	RANGE
int	4 bytes	−2,147,483,648 to 2,147,483,647 (just over 2 billion)
short	2 bytes	−32,768 to 32,767
long	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
byte	1 byte	−128 to 127
float	4 bytes	approximately ±3.40282347E+38F (6–7 significant decimal digits)
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)
char	2 bytes	\u0000 to \uFFFF
boolean		true or false

OPERADORES

▶ ARITMÉTICOS

- (+) suma
- (−) resta)
- (*) multiplicación)
- (/) división
- (%) módulo
- (++) (−−) incremento / decremento en uno

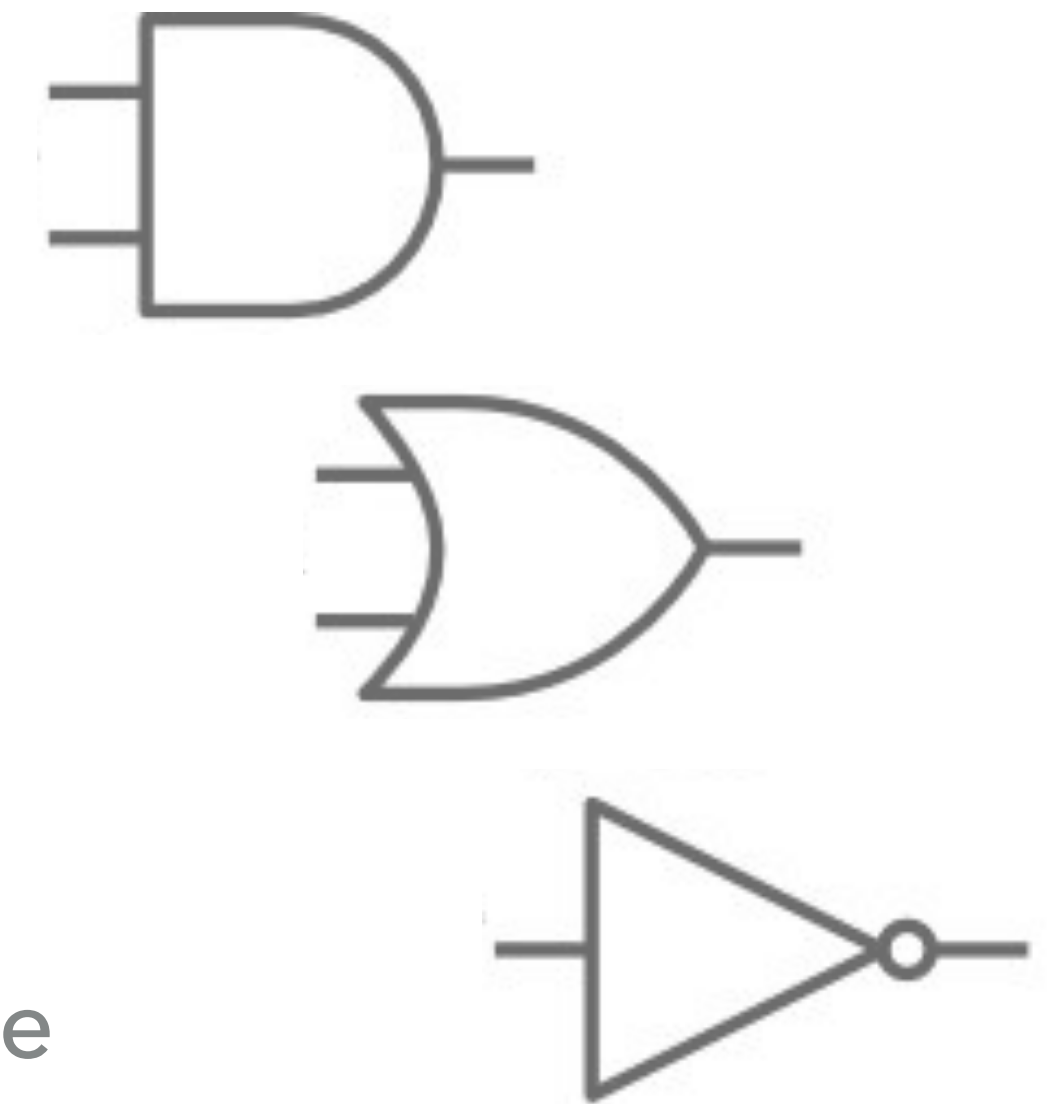


▶ CONDICIONAL (? :)

- `variable x = (expresión) ? valor_si_true : valor_si_false`

▶ LÓGICOS

- (&&) AND
- (||) OR
- (!) NOT



▶ ASIGNACIÓN

- (=) asignación simple
- (+=) (−=) (*=) (/=) (%=)
- Ejemplo : `A += B` \rightarrow `A = A+B`

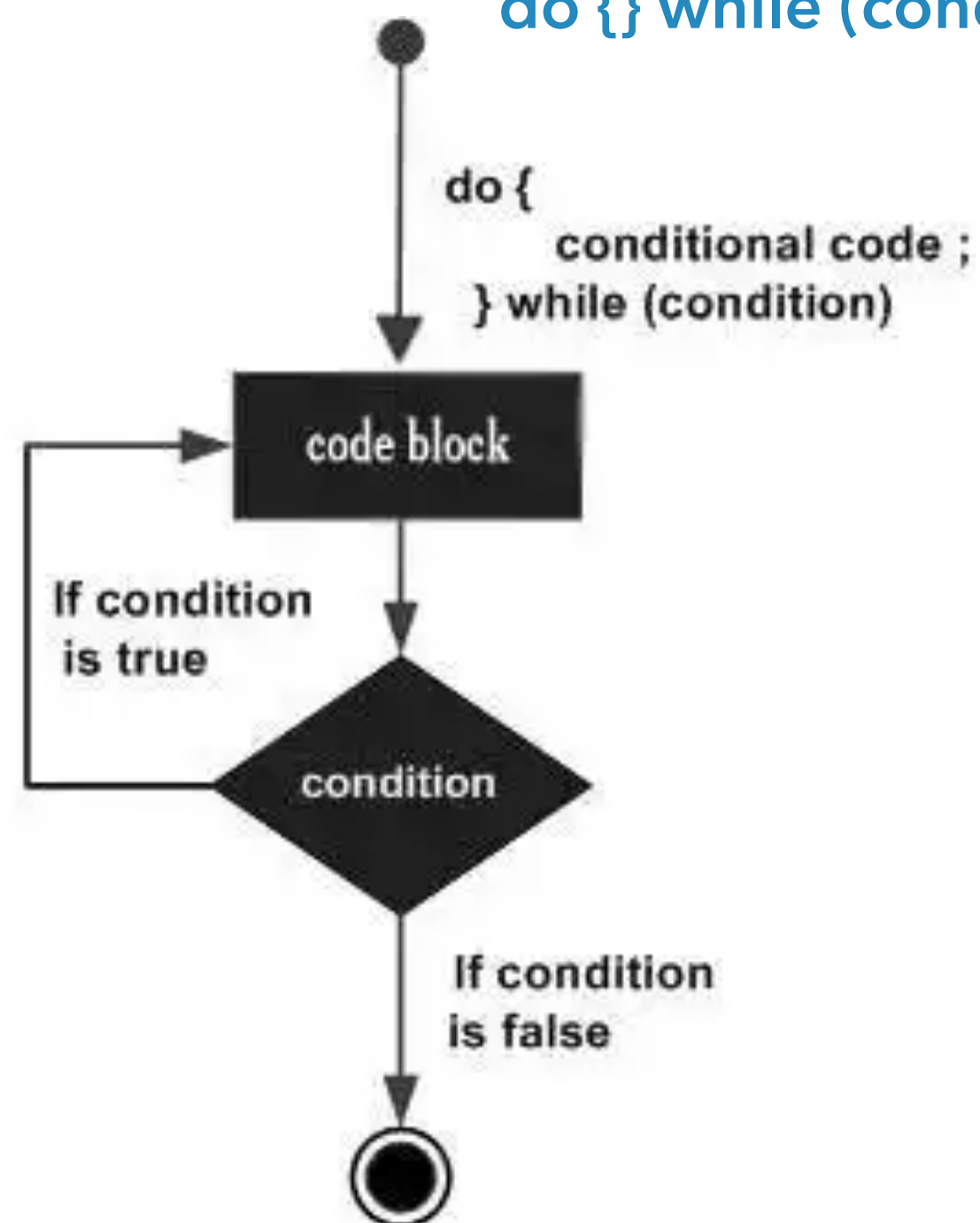
▶ RELACIONALES

- (==) (!=) Igualdad / desigualdad
- (<) (<=) (>=) (>)

CICLOS

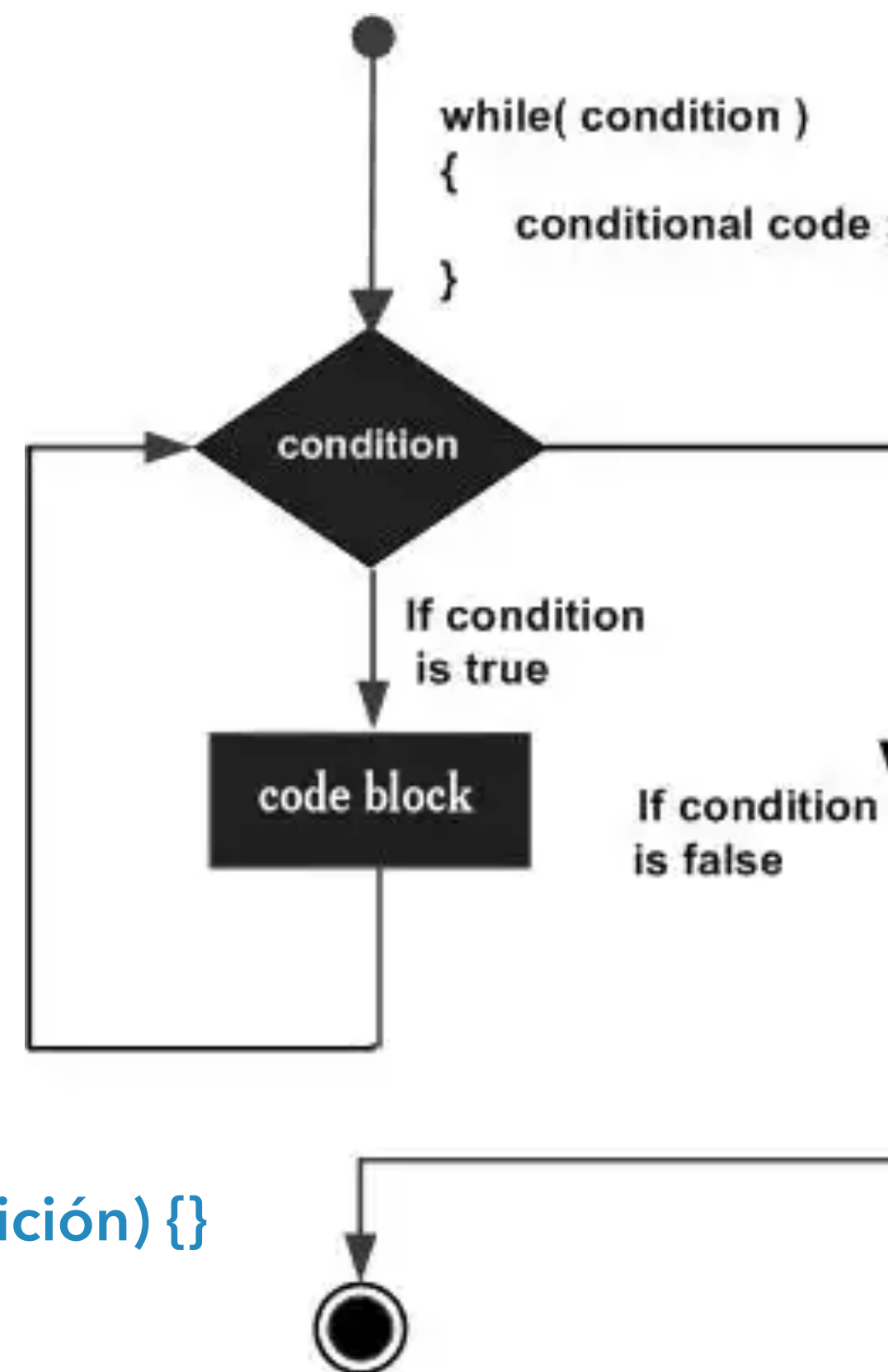
- ▶ Para continuar con un ciclo : **continue;**
- ▶ Para salir de un ciclo : **break;**

do {} while (condición)



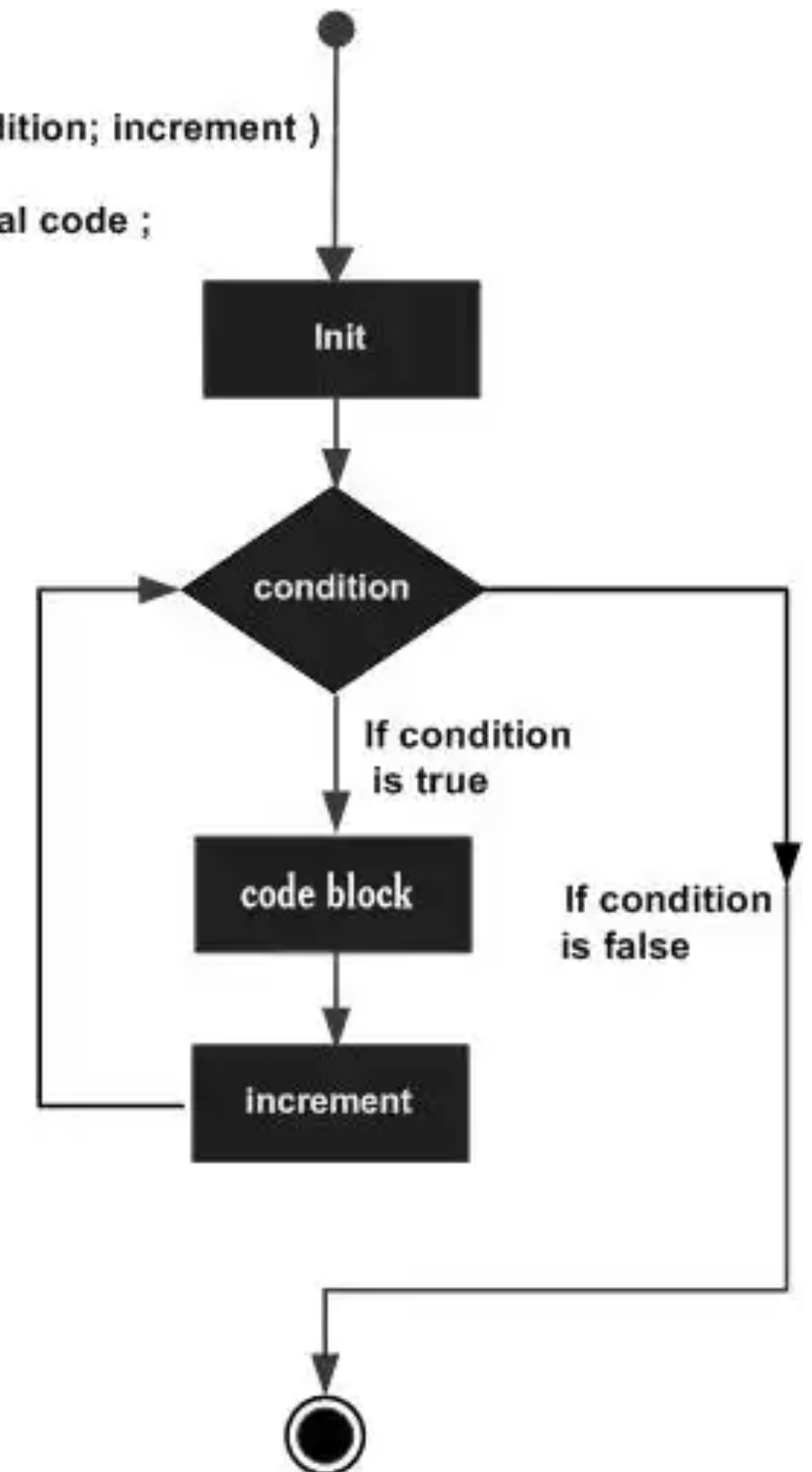
while(condición)
{
 conditional code ;
}

while (condición) {}



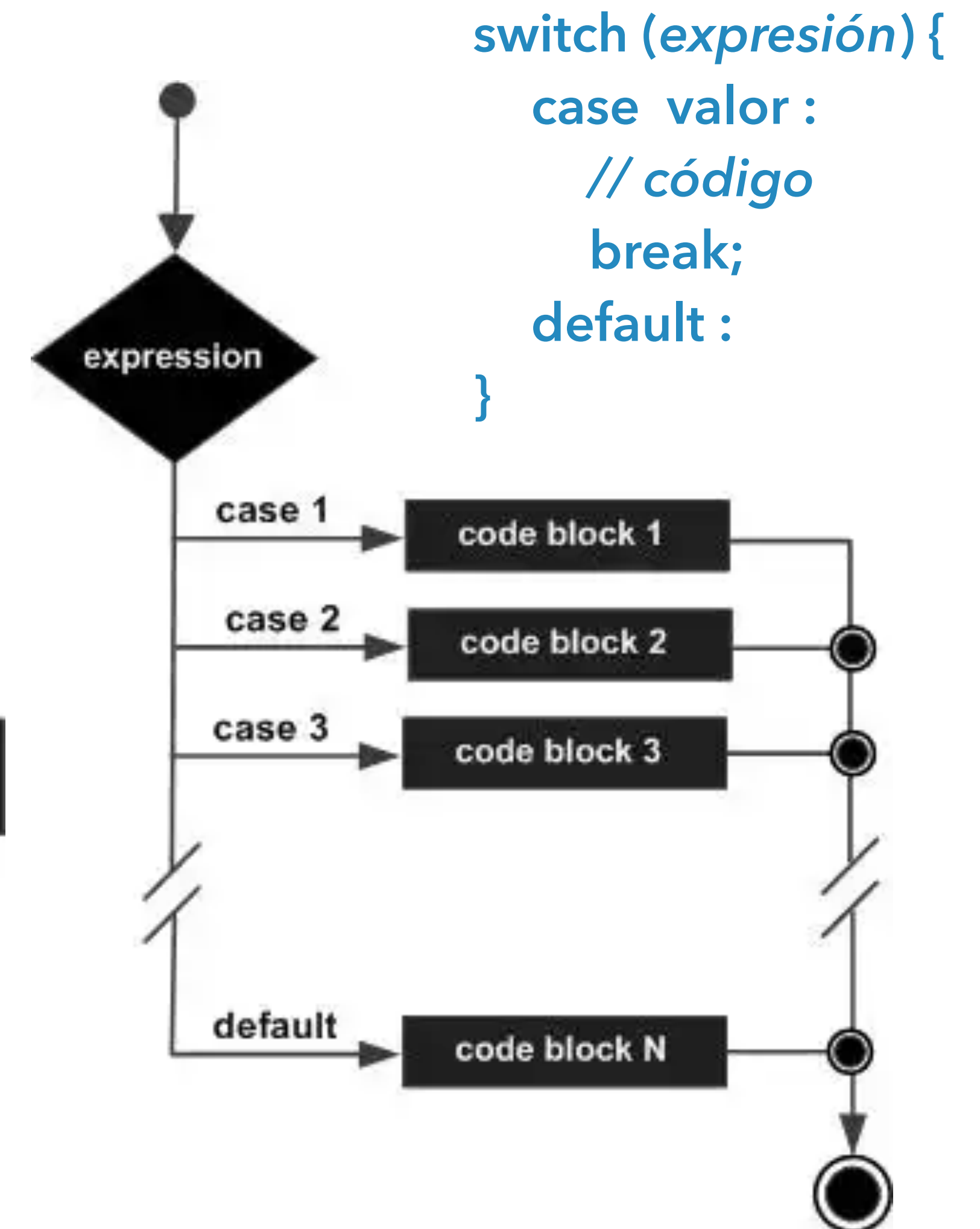
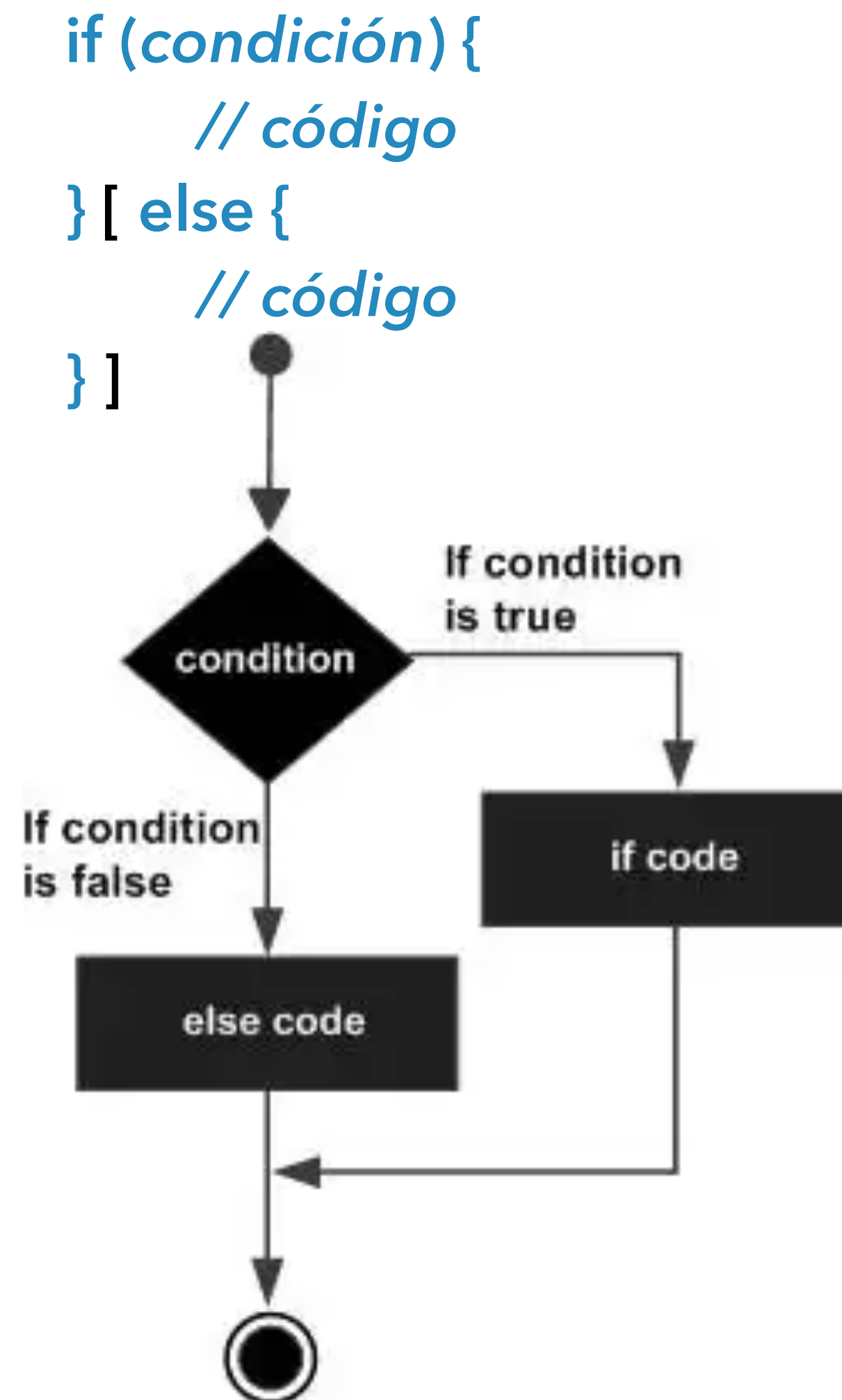
for (inicial ; condición ; incremento) {}
for (valor : valores) {}

for(init; condition; increment)
{
 conditional code ;
}



TOMA DE DECISIONES

- ▶ if / else
 - puede ser anidado
- ▶ switch case
 - la expresión puede ser de tipo:
 - int
 - String
 - enum



ARREGLOS

- ▶ Primer índice es 0

- ▶ Declaración

```
tipoDato[] arreglo;
```

- ▶ Creación

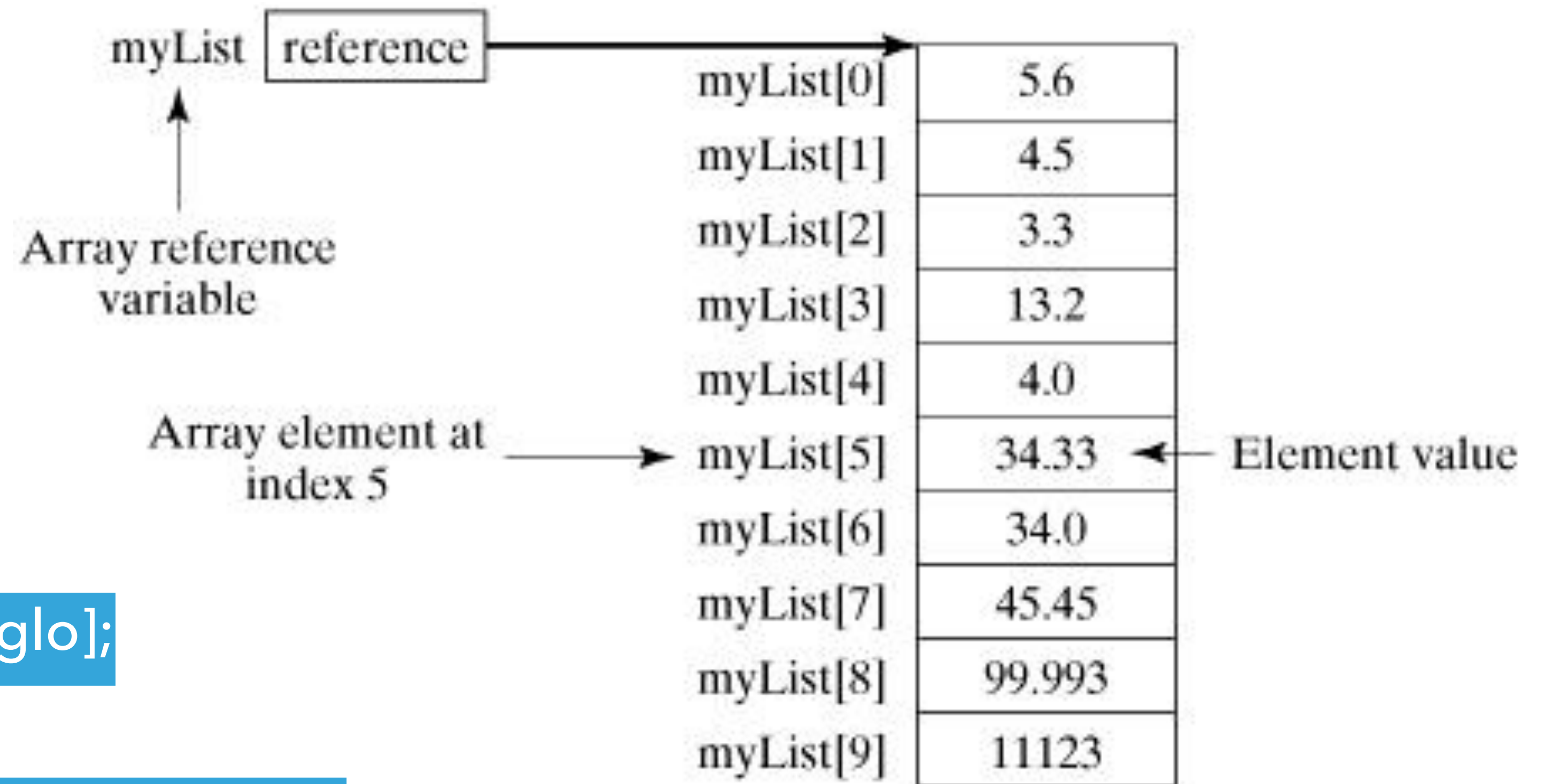
```
arreglo = new tipoDato[tallaArreglo];
```

```
arreglo = new double[10];
```

```
tipoDato[] arreglo = {valor0, valor1, ..., valork};
```

```
int[] miArreglo = {1,2,4,10};
```

```
int[][] arregloMatricial = {{1,2},{3,4},{5,6}};
```



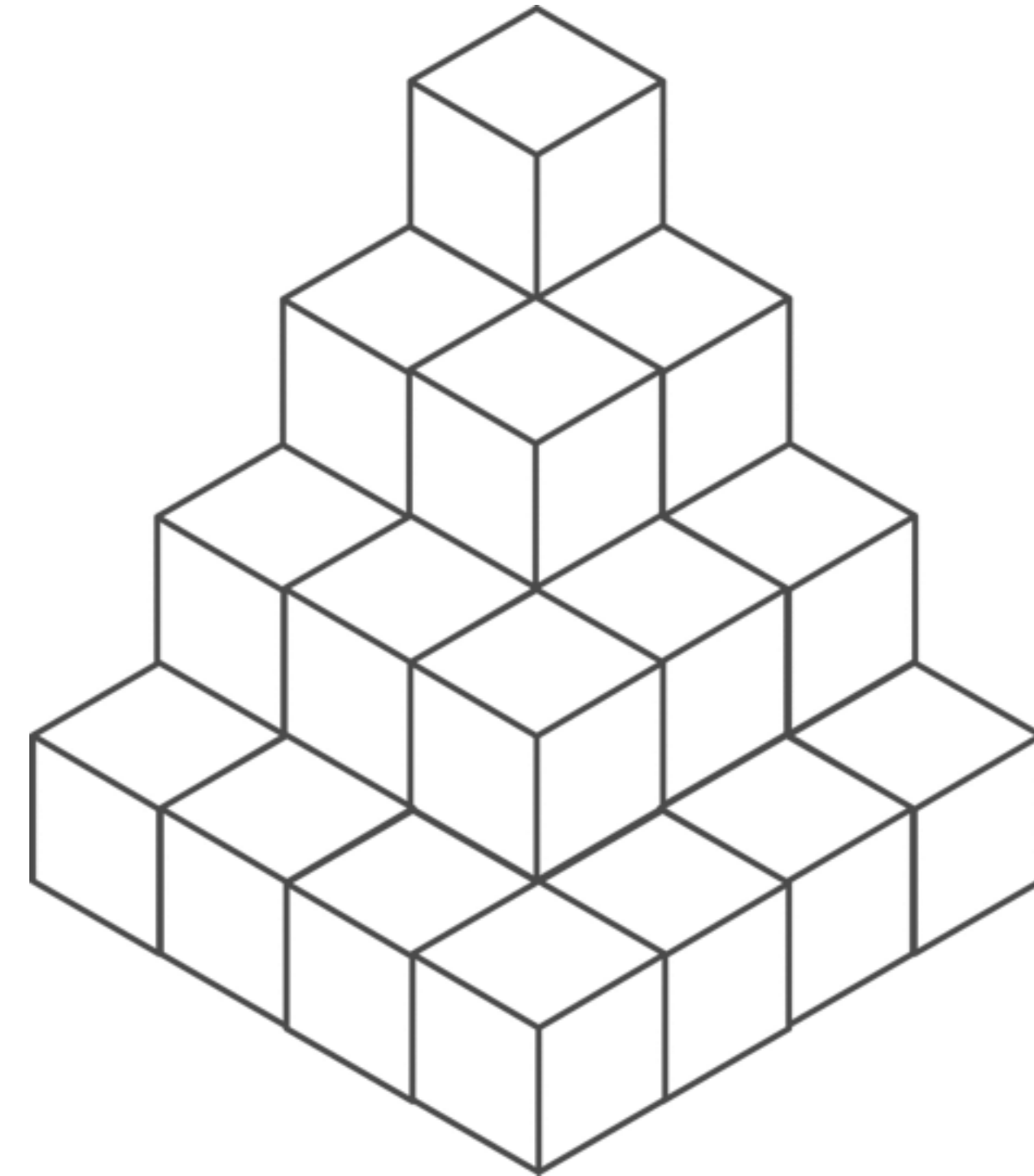
PALABRAS RESERVADAS

- ▶ No pueden ser usadas como nombres de constantes, variables o cualquier otro tipo de identificador

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

EJERCICIOS DE SINTAXIS

1. Declara dos variables x e y de tipo int, dos variables n y m de tipo double y asigna a cada una un valor. A continuación realiza y muestra por pantalla una serie de operaciones entre ellas.
2. Declare una variable c de tipo entero y asígnale un valor. A continuación muestra un mensaje indicando si el valor de c es positivo o negativo, si es par o impar, si es múltiplo de 5, si es múltiplo de 10 y si es mayor o menor que 100. Consideraremos el 0 como positivo. Utiliza los operadores condicionales if () {} y posteriormente el (?:), este último dentro del `println`.
3. Programa Java que muestre los números del 1 al 100 utilizando las instrucciones while, do..while, for
4. Calcular el promedio de una serie de 10 números dentro de un array



INTRO TO JAVA 1.3

OBJETOS

OBJETOS

Clase : PERSONA

Características :

- ▶ Nombre
- ▶ Fecha de nacimiento
- ▶ Sexo

Acciones :

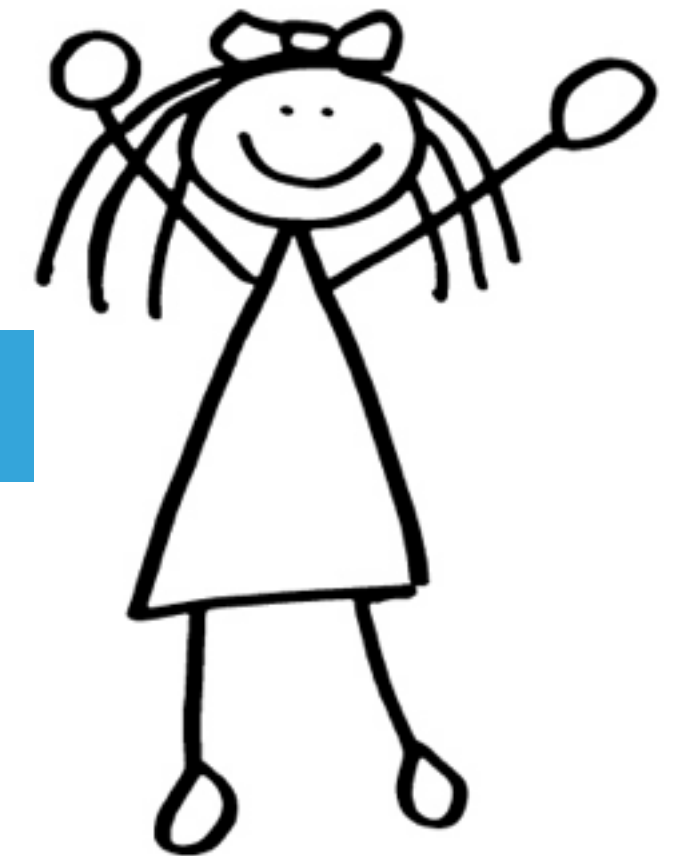
- ▶ Asignar nombre
- ▶ Preguntar nombre
- ▶ Saludar



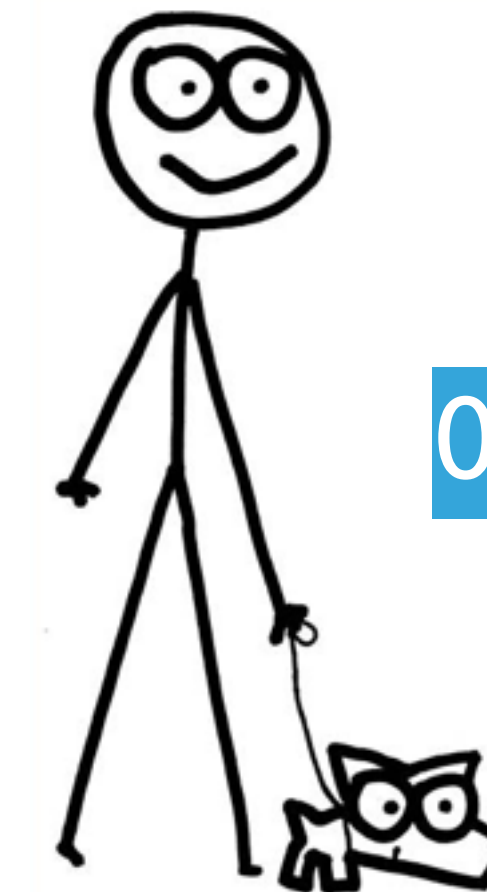
Daniel
31 mayo 2009
Hombre



Margarita
15 enero 2006
Mujer



Jorge
05 febrero 1990
Hombre



Rosana
20 abril 2000
Mujer



CLASES

- ▶ Clase

```
public class Persona {}
```

- ▶ Atributo / variable de instancia

```
private String nombre;
```

- ▶ Metodo

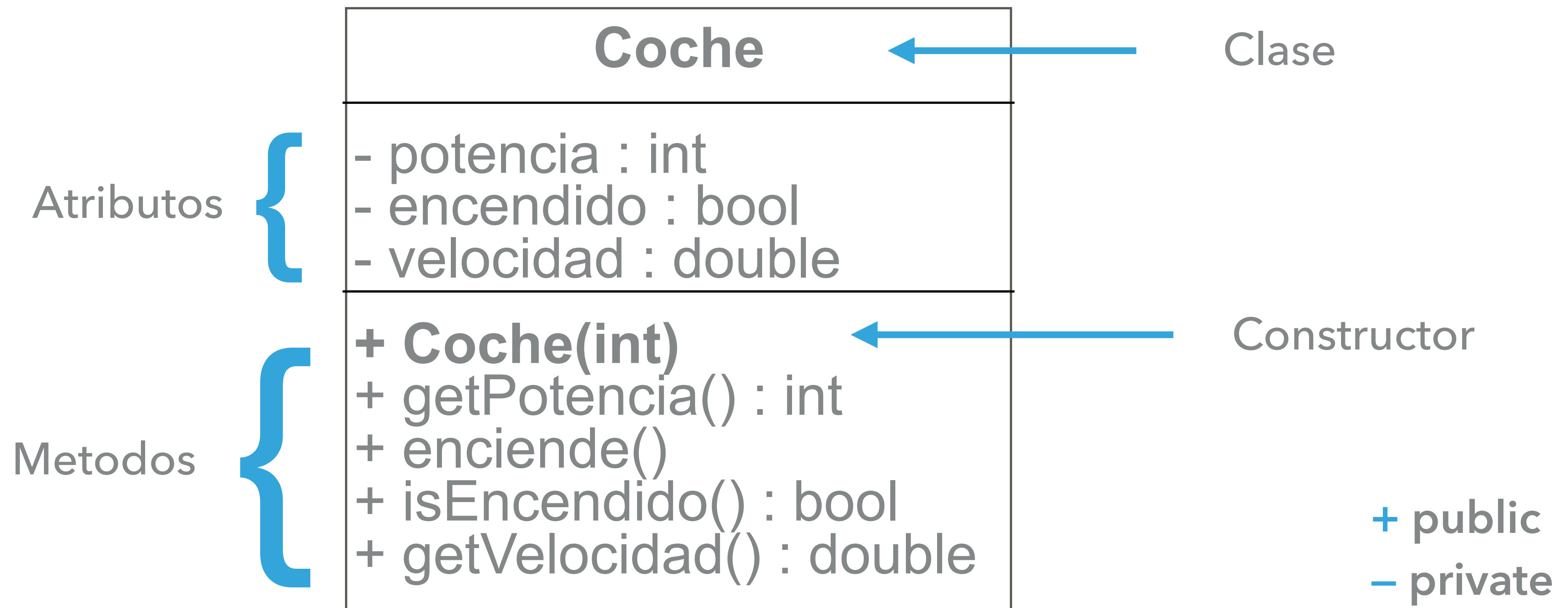
```
public String getNombre() {}
```

- ▶ Constructor

```
public Persona() {}
```

```
public class Persona {  
    // atributos  
    private String nombre;  
    private int edad;  
  
    // métodos  
    public String getNombre() {  
        return nombre;  
    }  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
    public void saludar() {  
        //lógica para saludar  
    }  
  
    // constructor  
    public Persona(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

DIAGRAMAS DE CLASE UML



MODIFICADORES

▶ ACCESO

- private (clase)
- sin modificador (paquete)
- protected (paquete y subclasses)
- public (todos)

▶ OTROS

- final
- abstract
- static
- synchronised / volatile

```
public class Persona {}  
private boolean bandera;  
static final double SEMANAS = 9.5;  
protected static final int ANCHO = 42;  
public static void main(String[] arguments) {}
```

TIPOS DE VARIABLES : LOCALES

- ▶ Declaradas en métodos, constructores o bloques
- ▶ Creadas al momento de entrar y destruidas al salir del método, constructor o bloque
- ▶ No usan modificadores de acceso
- ▶ Visibles únicamente donde son declaradas
- ▶ Debe asignárseles un valor inicial antes de su primer uso.

```
public class Test{  
    public void edadPerro() {  
        int edad = 0;  
        edad = edad + 7;  
        System.out.println(edad + " años");  
    }  
    public static void main(String args[]){  
        Test test = new Test();  
        test.pupAge();  
    }  
}
```

TIPOS DE VARIABLES : DE INSTANCIA

- ▶ Declaradas en una clase pero fuera de un método, constructor o bloque
- ▶ Se crean al momento de crear un objeto ('new') y se destruyen junto con el objeto
- ▶ Pueden asociárseles modificadores de acceso
- ▶ Si son públicas pueden invocarse así:
ReferenciaObjeto.nombreVariable

```
public class Empleado{
    // Visible en sub clases
    public String nombre;
    // Visible unicamente en la clase
    private double salario;
    // Asignación en constructor
    public Empleado (String nombre){
        this.nombre = nombre;
    }
    // Asignación en método
    public void setSalario(double sal){
        salario = sal;
    }
    public void printEmp(){
        System.out.println("nombre : " + nombre );
        System.out.println("salario :" + salario);
    }
}

public static void main(String args[]){
    Empleado empUno = new Empleado("Ana");
    empUno.setSalario(1000);
    System.out.println(empUno.nombre);
    empUno.printEmp();
}
```

TIPOS DE VARIABLES : DE CLASE / ESTATICAS

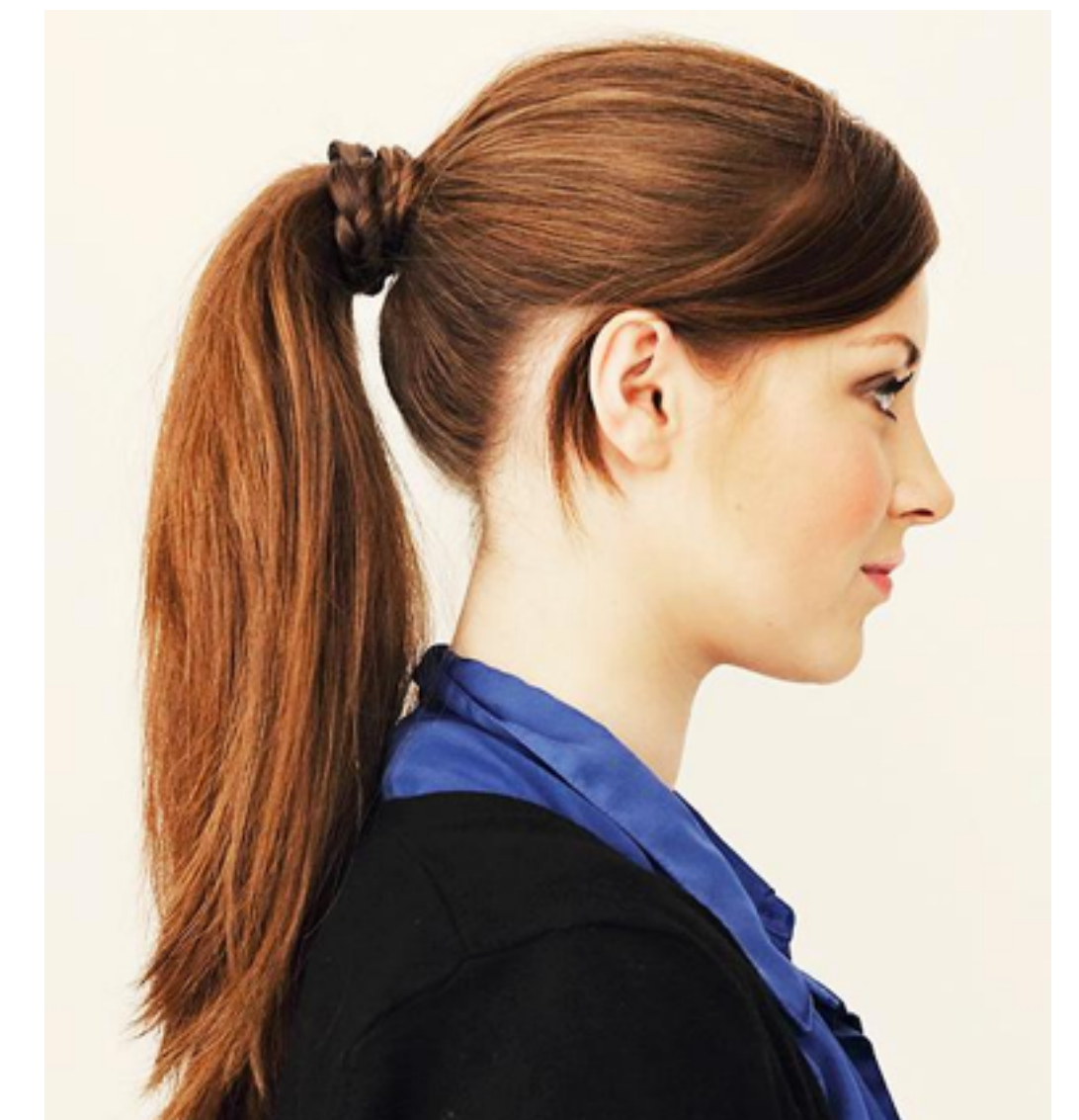
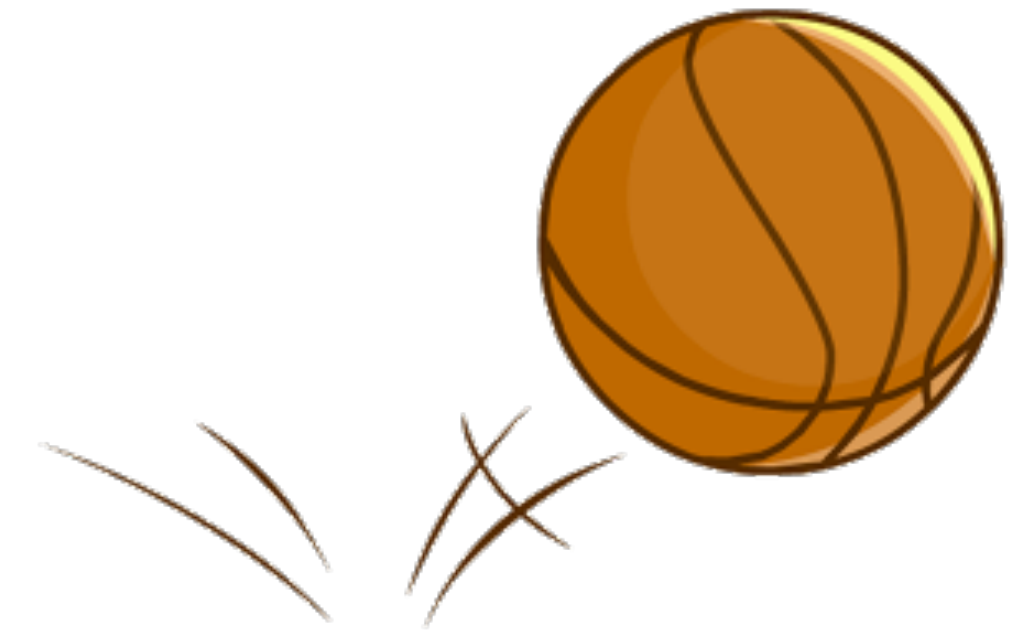
```
public class Empleado{  
    private static double salario;  
    public static final String  
        DEPARTAMENTO = "Development ";  
}  
  
public static void main(String args[]){  
    salario = 1000;  
    System.out.println(Empleado.DEPARTAMENTO  
        +": "+Empleado.salario);  
}
```

- ▶ Se declaran con la palabra **static** en una clase pero fuera de un método, constructor o bloque
- ▶ Sólo habrá una copia por clase sin importar el número de objetos
- ▶ Se crean cuando inicia el programa y se destruyen cuando finaliza
- ▶ Si la variable es **public static final** deben ir en Mayusculas **NOMBRE_VARIABLE**
- ▶ Pueden ser llamadas mediante :
NombreClase.nombreVariable

SOBRECARGA DE METODOS

- Uso del mismo nombre de método para ejecutar acciones con argumentos diferentes

```
void bota (String color);  
void bota (int veces);  
void cola (String colorCabello);  
void cola (int numPersonas, int duracion);
```



ENCAPSULACION

```
public class Usuario{  
    private String password  
    private String username  
  
    public Usuario(String user, String pass){  
        this.username = user  
        setPassword(pass)  
    }  
  
    private void setPassword(String pass) {  
        this.password = encriptar(pass)  
    }  
}
```

Atributos privados y Métodos públicos (getters & setters)

Beneficios:

- ▶ Solo lectura / escritura
- ▶ La clase tiene control total de lo que se almacena en los atributos
- ▶ El usuario no sabe la forma en la que la clase almacena la información

EJERCICIOS

Crea dos clases (Persona y Punto) siguiendo las siguientes estructuras UML.
En todo caso no olvides crear sus getters, setters, toString y constructor.

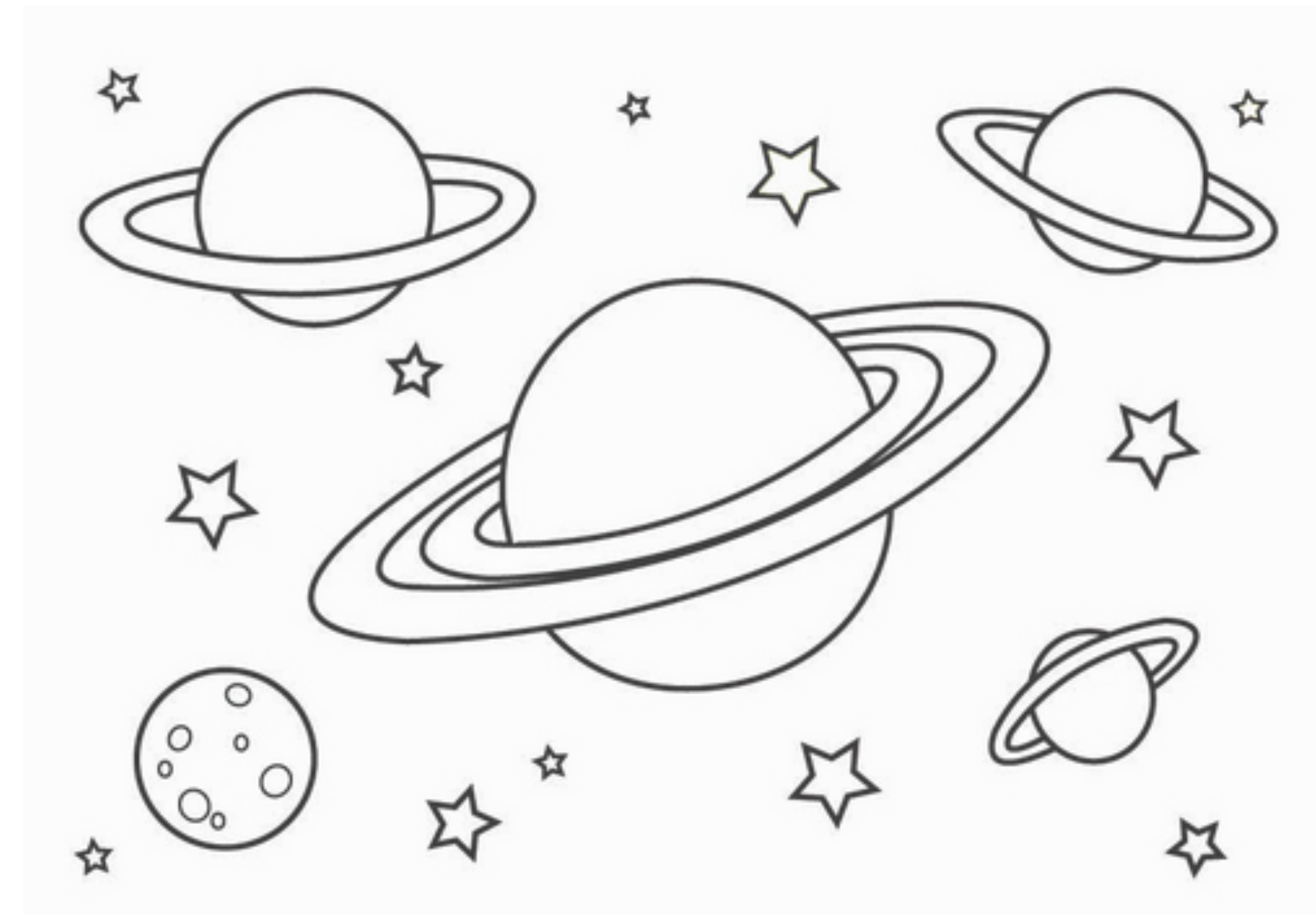
Persona
- nombre : String
- edad : int
+ Persona(String nombre)
+ getNombre() : String
+ setEdad(int edad)
+ getEdad() : int
+ toString() : String

Punto
-x: int
-y: int
+ Point(int x, int y)
+ distancia(Point point) : double
+ distancia() : double
+ trasladar(int dx, int dy) : Point

ENUM

```
public enum Planeta {  
    MERCURIO(3.303+23, 2.4397e6),  
    TIERRA(5.976e+24, 6.37814e6),  
    MARTE(6.421e+23, 3.3972e6),  
    JUPITER(1.9e+27, 7.1492e7);  
  
    private final double masa;  
    private final double radio;  
  
    Planeta(double masa, double radio) {  
        this.masa = masa;  
        this.radio = radio;  
    }  
    public double getMasa() {  
        return this.masa;  
    }  
    private double calculaPesoSuperficie() {  
        ...  
    }  
}
```

- ▶ Conjunto de constantes
- ▶ Pueden tener un constructor personalizado
- ▶ Pueden tener métodos especiales



COMPOSICION

- ▶ Asociación : Relación entre dos clases

Conductor **ASOCIADO** Auto

- ▶ Agregación : ciclo de vida independiente. Relación unidireccional

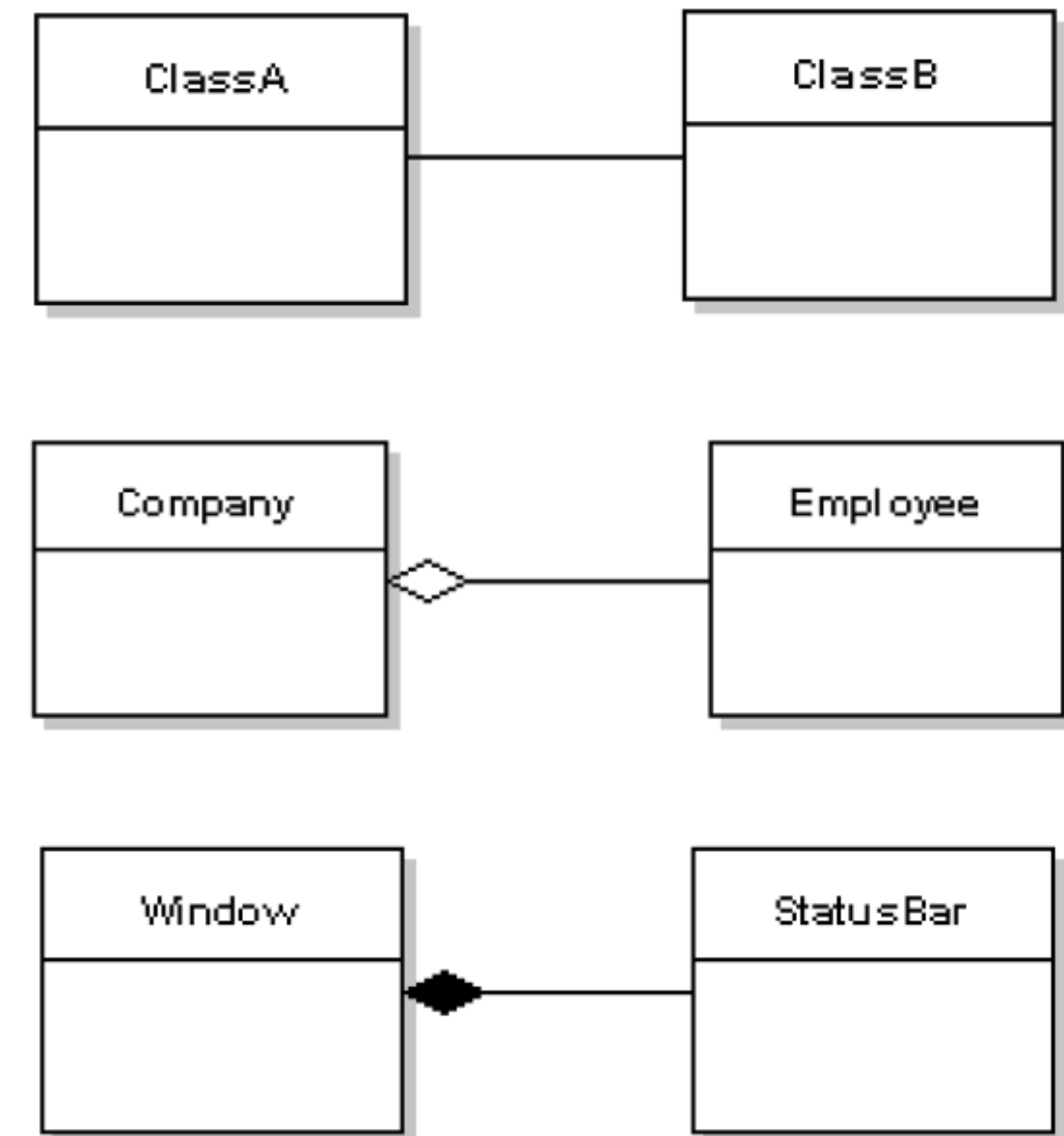
Estudiante **TIENE** Dirección

```
Estudiante(Direccion dir){ this.direccion = dir}
```

- ▶ Composición : ciclo de vida identico

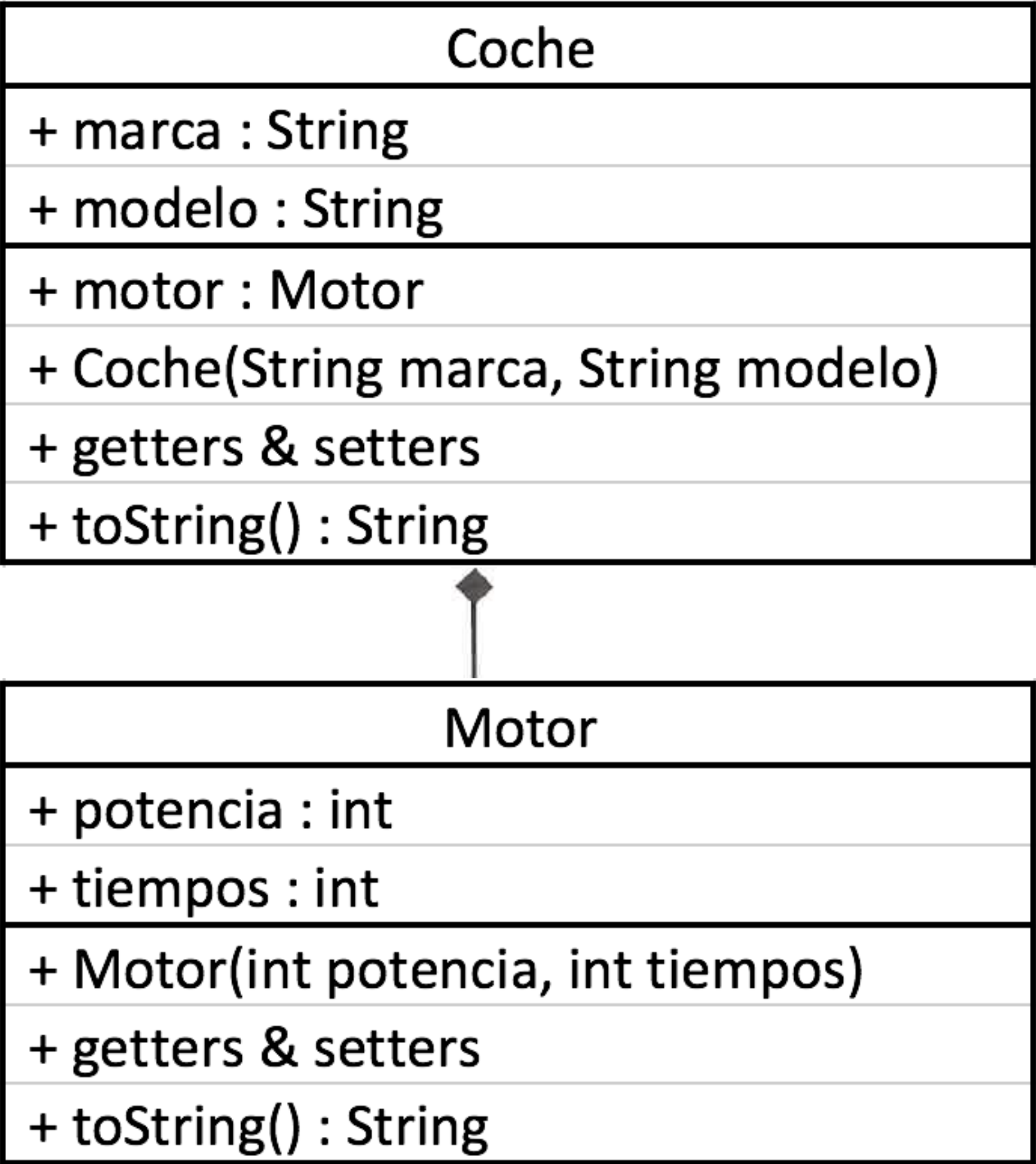
Humano **NECESITA** Corazón

```
Humano(){ this.corazon = new Corazon() }
```

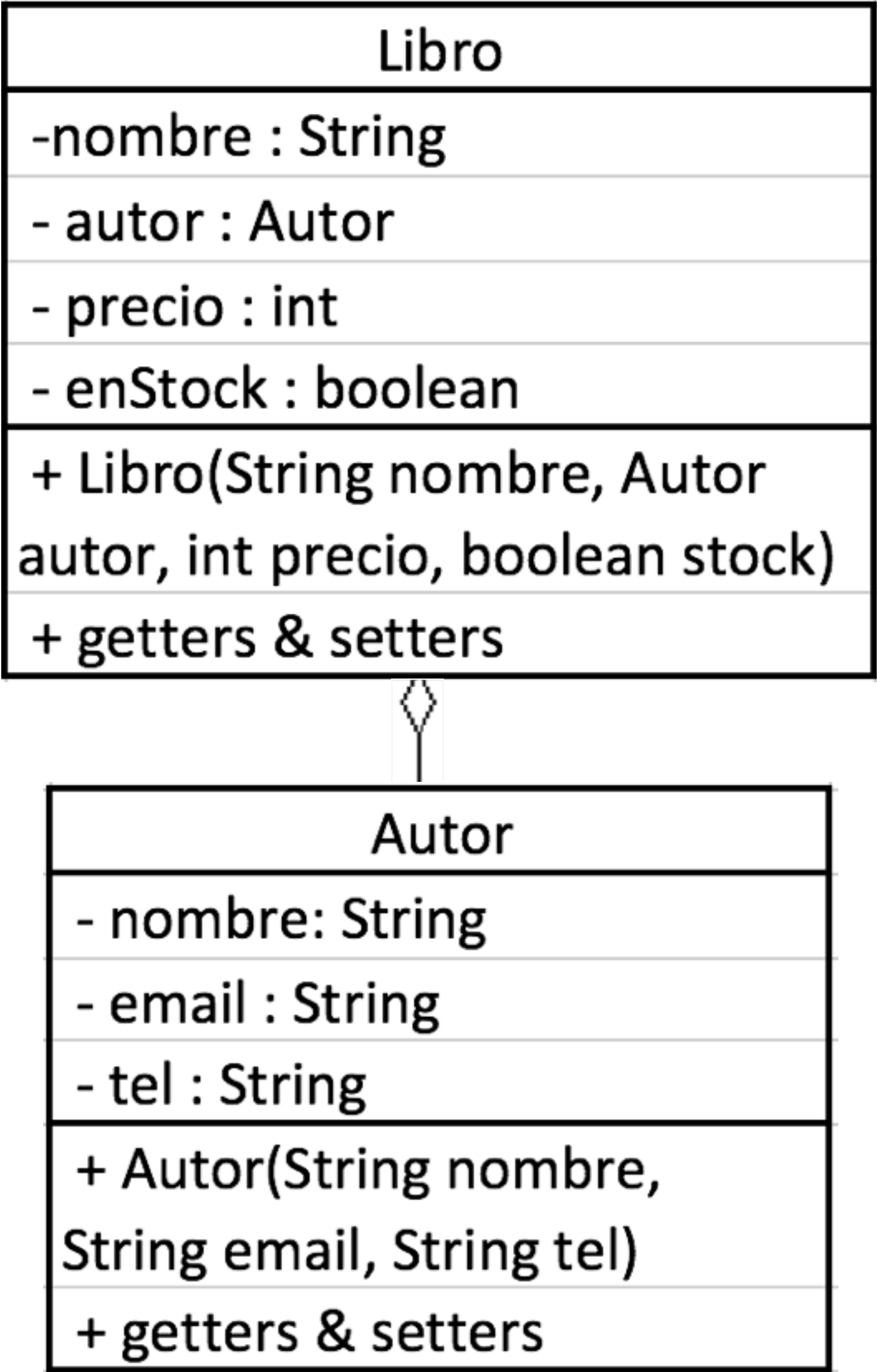


EJERCICIOS

Coche compuesto de un motor al momento de instancias un objeto



Agregación de Autor a Libro



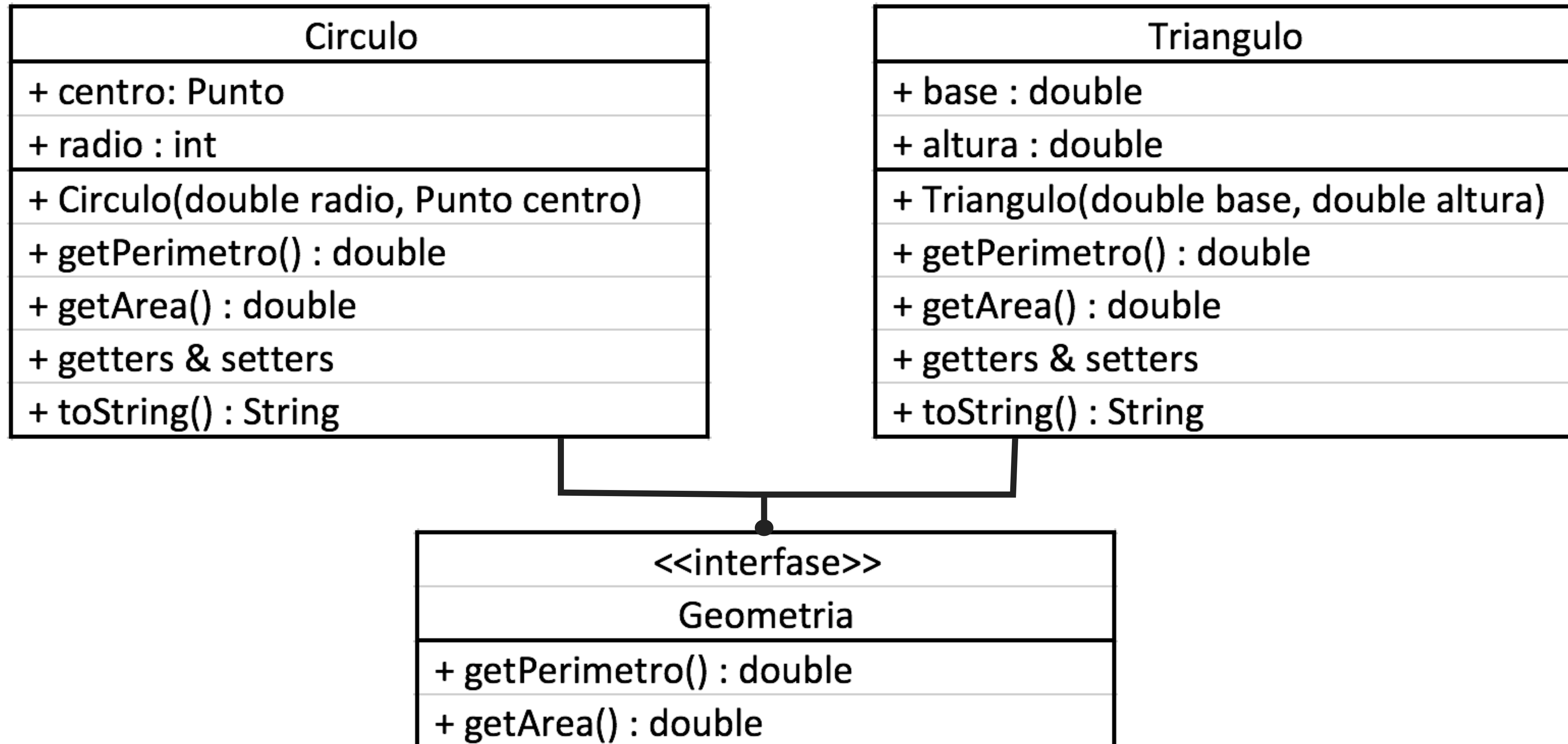
INTERFACES

- ▶ No se pueden instanciar
- ▶ No contienen constructores
- ▶ Todos sus métodos son abstractos (sin abstract)
- ▶ Puede contener variables de tipo static final
- ▶ Es implementada por una clase
- ▶ Puede extender multiples interfaces
- ▶ En UML <<Interface>>
- ▶ Una clase puede implementar más de una interfaz

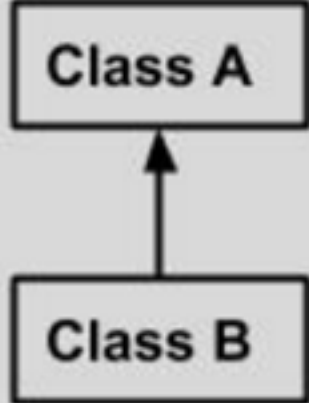
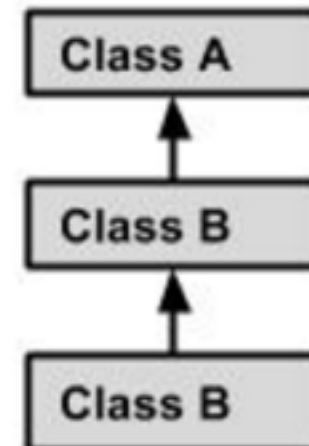
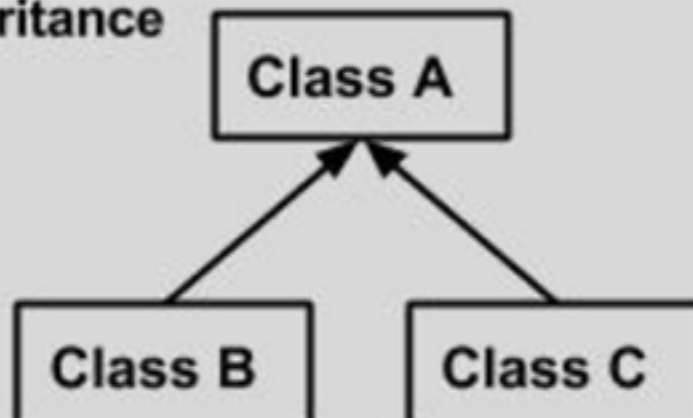
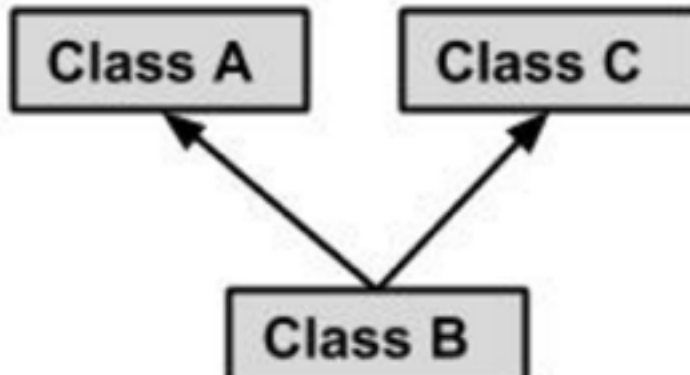
```
public class Basketball implements Deportes,  
Evento {}
```

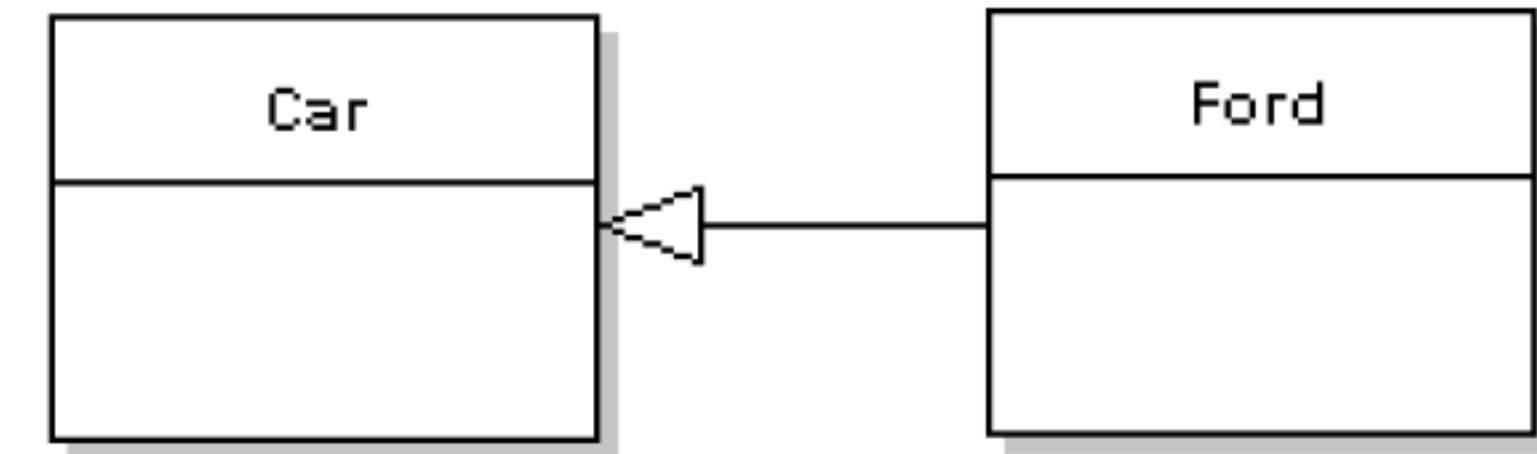
```
//Archivo: Deportes.java  
public interface Deportes  
{  
    public void setEquipoCasa(String nombre);  
    public void setEquipoVisitante(String nombre);  
}  
//Archivo: Football.java  
public interface Football implements Deportes  
{  
    public void equipoCasaAnoto(int puntos);  
    public void equipoVisitanteAnoto(int puntos);  
    public void terminaMedioTiempo(int tiempo);  
}  
//Archivo: Basketball.java  
public interface Basketball implements Deportes  
{  
    public void equipoCasaEncesto(int puntos);  
    public void equipoVisitanteEncesto(int puntos);  
    public void terminaCuarto(int cuarto);  
}
```

EJERCICIOS



HERENCIA

Single Inheritance  <pre> graph BT B[Class B] --> A[Class A] </pre>	<pre> public class A { } public class B extends A { } </pre>
Multi Level Inheritance  <pre> graph BT B2[Class B] --> B1[Class B] B1 --> A[Class A] </pre>	<pre> public class A { } public class B extends A { } public class C extends B { } </pre>
Hierarchical Inheritance  <pre> graph BT B[Class B] --> A[Class A] C[Class C] --> A </pre>	<pre> public class A { } public class B extends A { } public class C extends A { } </pre>
Multiple Inheritance  <pre> graph BT B[Class B] --> A[Class A] B --> C[Class C] </pre>	<pre> public class A { } public class B { } public class C extends A,B { } </pre> <p>// Java does not support mutiple Inheritance</p>



- ▶ Proceso por el cual una clase adquiere las propiedades (methods y atributos) de otra
- ▶ La super clase es extendida por la sub clase
`public class Sub_class extends Super_class`
- ▶ La palabra **super** :
 - distingue la super de la sub clase
`this.method();`
`super.method();`
 - invoca a la super clase
`super.variable`
`super()`

OVERRIDING (SOLAPAR METODOS)

Al heredar de una super clase se pueden solapar o reescribir sus métodos en la clase hija

- ▶ Los argumentos, return-type y nombre del método deben ser iguales
- ▶ El nivel de acceso no puede ser más restrictivo que el de la super clase.
- ▶ Dentro del mismo paquete se pueden sobre escribir los métodos que no son `private` o `final` en la super clase.
- ▶ Fuera del paquete sólo los `public` or `protected`.
- ▶ No aplica a constructores.

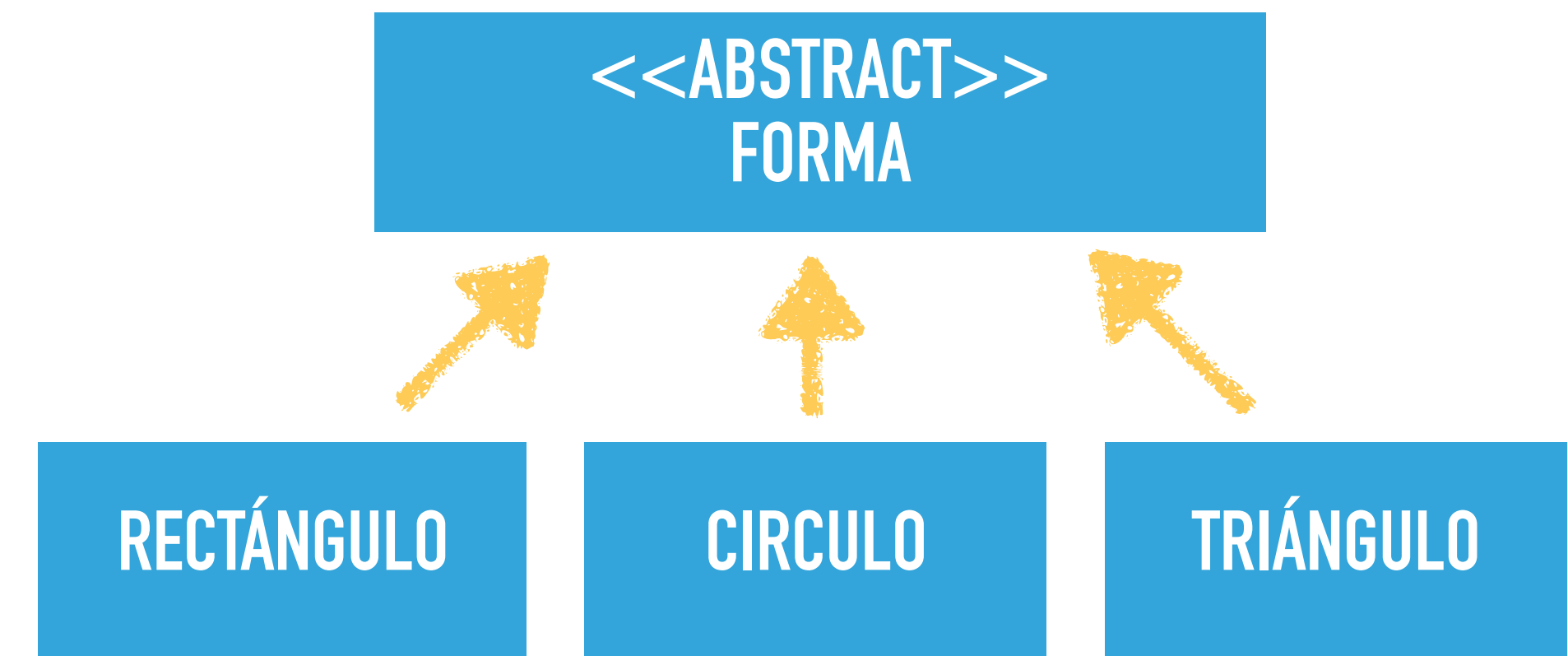
```
class Animal{
    public void mover(){
        System.out.println(
            "Los animales pueden moverse"
        );
    }
}

class Perro extends Animal{
    public void mover(){
        super.move();
        System.out.println(
            "Los perros pueden correr"
        );
    }
}
```


ABSTRACCION

- ▶ Se declaran

```
public abstract class Empleado {}
```
- ▶ Pueden o no contener métodos abstractos (sin contenido)
- ▶ No pueden ser instanciadas
- ▶ Las clases no abstractas las heredan (extends) e implementan sus métodos abstractos
- ▶ En UML <<abstract>>

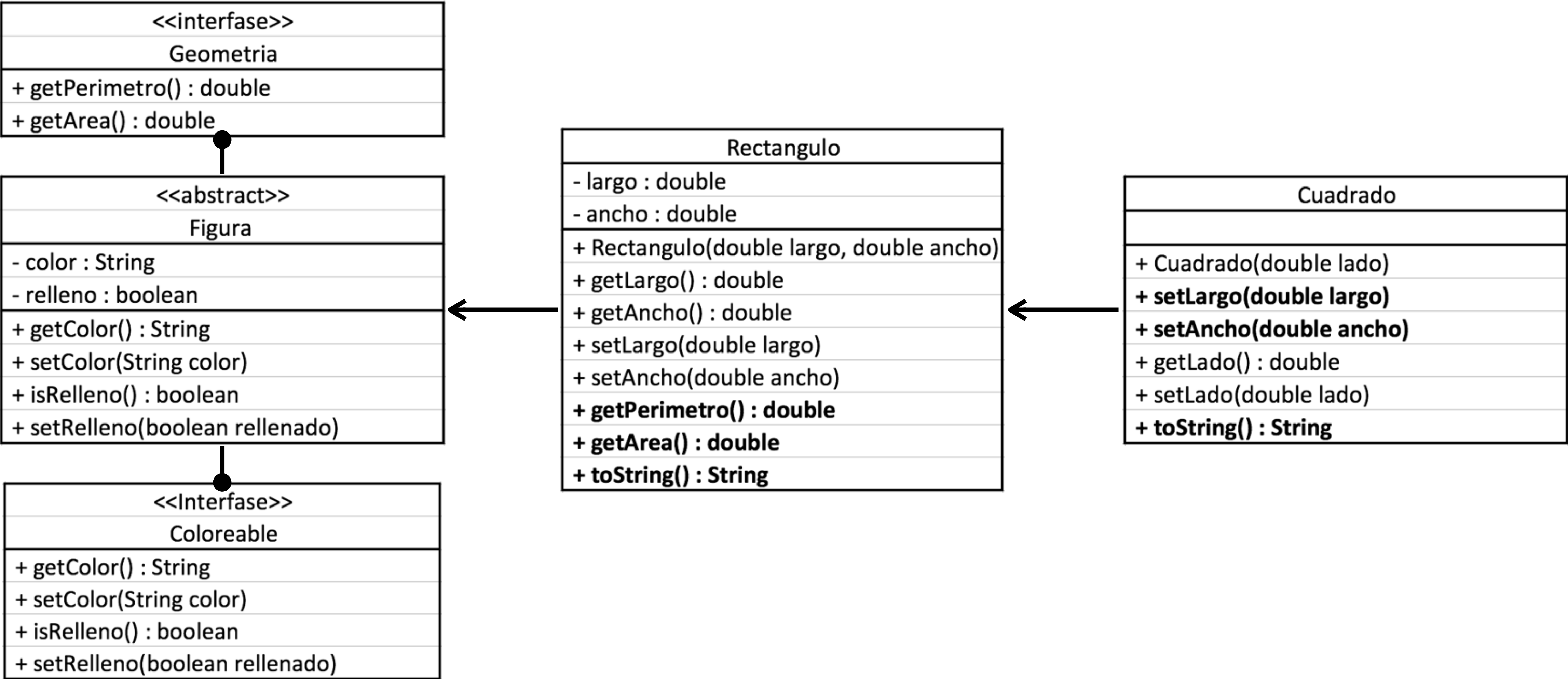


POLIMORFISMO

- ▶ La super clase puede usarse como referencia para referirse a la sub clase del objeto

```
public interface Vegetariano{}  
public class Animal{}  
public class Venado extends Animal implements Vegetariano{}  
  
// Venado es venado  
// Venado es animal  
// Venado es vegetariano  
  
Venado venado = new Venado();  
Animal animal = venado;  
Vegetariano vegetariano = venado;  
Object objeto = venado;
```

EJERCICIOS



PAQUETES

► Beneficios

- Reuso
- Fácil acceso
- Conflicto de nombres

► Pueden ser importados

```
import java.sql.*
```

► ... o definidos por el usuario

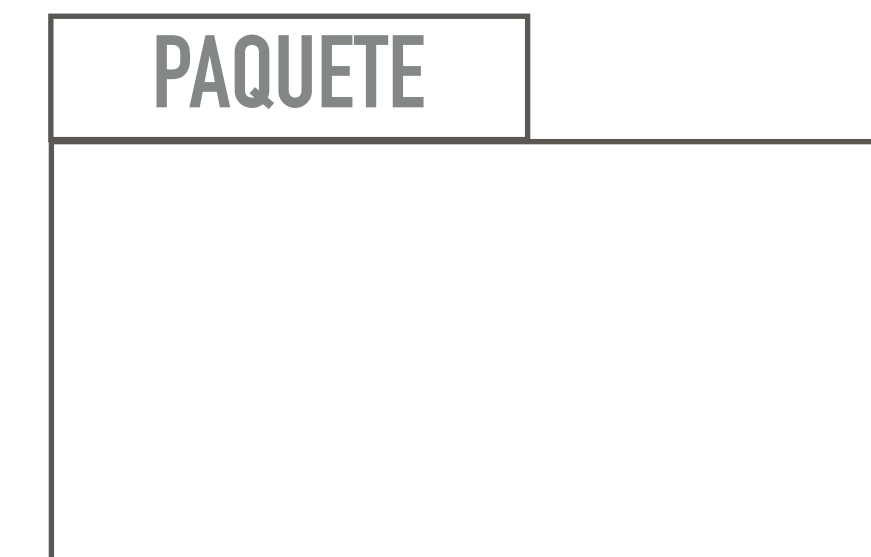
```
package tools; //primera linea de código
```

► Compilación

```
javac -d directorioDestino nombreArchivo.java
```

► Observaciones

- Declaración de un paquete máximo por archivo
- Es la primera línea en archivo
- Nombre `com.empresa.nombrePaquete`
- Directorio `com/empresa/nombrePaquete/`



APLICACIONES ESPECIALES

Java Servlet

- ▶ Extiende la funcionalidad de un servidor web
- ▶ Recibe peticiones y responde adecuadamente

Java Applet

- ▶ Programas embebidos en otras aplicaciones (e.g.: en una pagina web)
- ▶ Se ejecuta en una JVM dentro del navegador

Java Application

- ▶ Programas que corren en el escritorio

Java Server Pages (JSP)

- ▶ Documentos de texto que describen el comportamiento del servidor ante peticiones específicas
- ▶ Son interpretados por un JSP Server y a veces compilados a Servlet.
- ▶ El JSP server monitorea los cambios y recompila si es necesario

ENLACES EXTERNOS

- ▶ <http://github.com/codificadas/Javaficadas>
- ▶ [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
- ▶ <http://www.tutorialspoint.com/java/index.htm>
- ▶ <http://www.oracle.com/>
- ▶ <https://dzone.com/storage/assets/3980-rc112-010d-uml.pdf>
- ▶ <https://dzone.com/storage/assets/487673-rc024-corejava.pdf>
- ▶ <http://www.tutorialspoint.com/javaexamples/index.htm>