

# Java Fundamentals

---

Juan Carlos Trejo



# 1

---

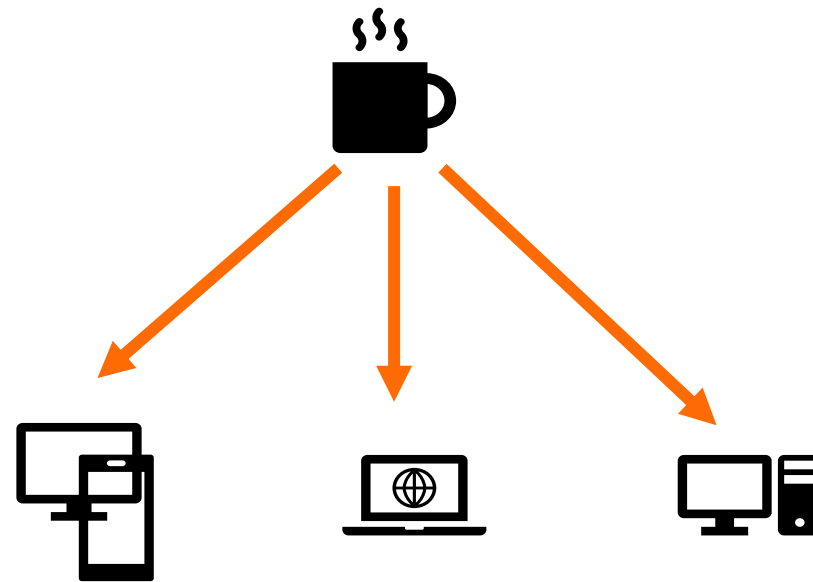
# Módulo 1

Programación en Java

Sun Microsystems patrocinó en 1991 un proyecto interno de investigación denominado Green, el cual desembocó en el desarrollo de un lenguaje basado en C++ al que su creador, James Gosling, llamó Oak

Cuando un grupo de gente de Sun visitó una cafetería local, sugirieron el nombre Java (una variedad de café) y así se quedó.

Sun anunció formalmente a Java en una importante conferencia que tuvo lugar en mayo de 1995. Java generó la atención de la comunidad de negocios debido al fenomenal interés en World Wide Web



## Versiones de Java

- **J2SE o simplemente Java SE:** Java 2 Standard Edition o Java Standard Edition. Orientado al desarrollo de aplicaciones cliente / servidor. No incluye soporte a tecnologías para internet. Es la base para las otras distribuciones Java y es la plataforma que utilizaremos nosotros en este curso por ser la más utilizada.
- **J2EE: Java 2 Enterprise Edition:** Orientado a empresas y a la integración entre sistemas. Incluye soporte a tecnologías para internet. Su base es J2SE.
- **J2ME: Java 2 Micro Edition:** Orientado a pequeños dispositivos móviles (teléfonos, tabletas, etc.).

JDK es el Java Development Kit o, en español, Herramientas de Desarrollo de Java.

Sirve para construir programas usando el lenguaje de programación Java. Trae herramientas útiles como el compilador (javac), el desensamblador de binarios (javap), debugger, entre otras herramientas. Una instalación de JDK ya contiene un JRE dentro de las carpetas.

# Java Runtime Environment

---

JRE es el Java Runtime Environment o, en español, el Entorno de Ejecución de Java. Contiene a la JVM y otras herramientas que permiten la ejecución de las aplicaciones Java.

JRE no posee compiladores ni herramientas para desarrollar las aplicaciones Java, solo posee las herramientas para ejecutarlas.

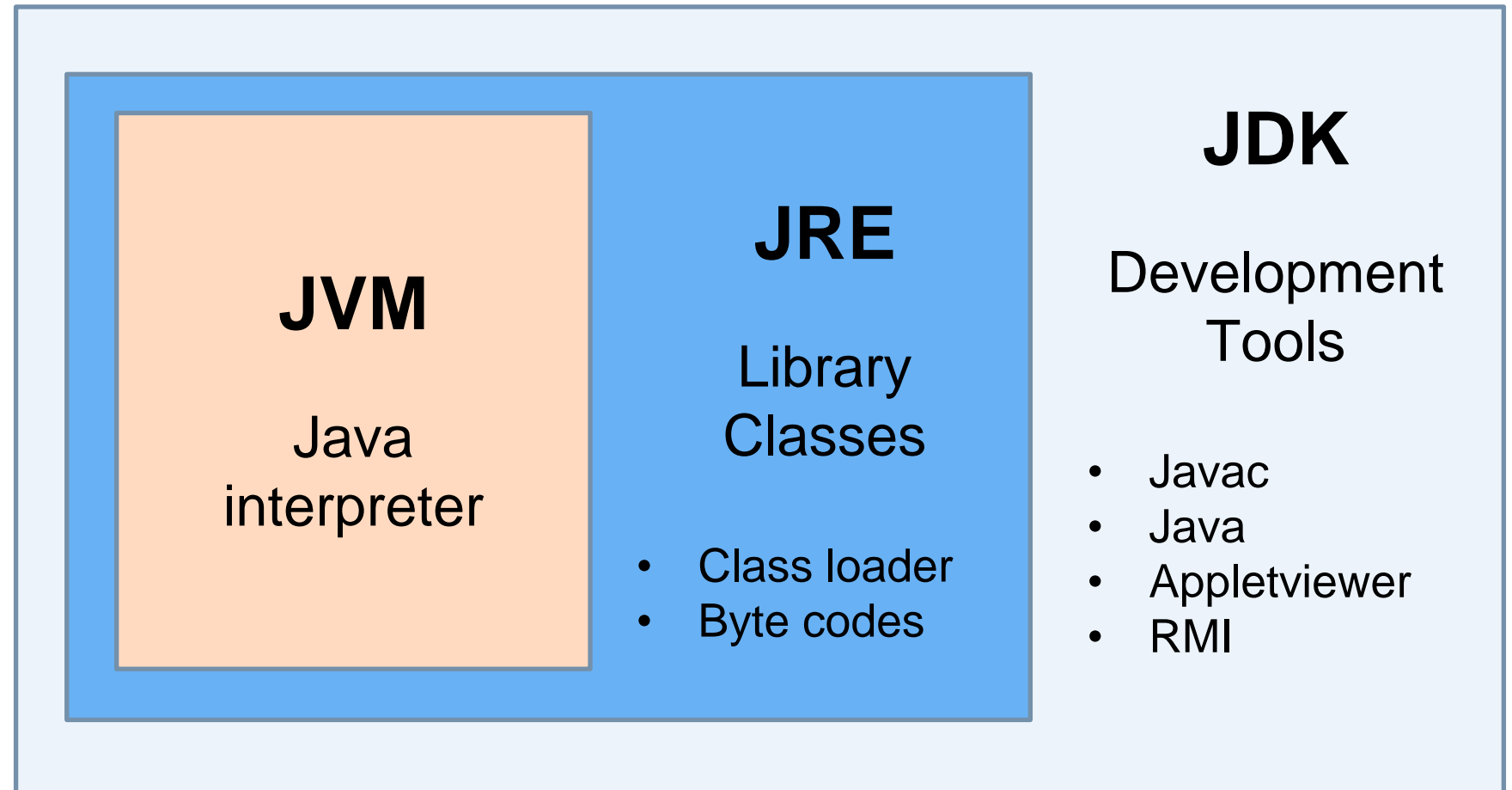
# Java Virtual Machine

---

Java Virtual Machine, o JVM, carga, verifica y ejecuta el código de bytes de Java. Se le conoce como el intérprete o el núcleo del lenguaje de programación Java porque ejecuta la programación Java.



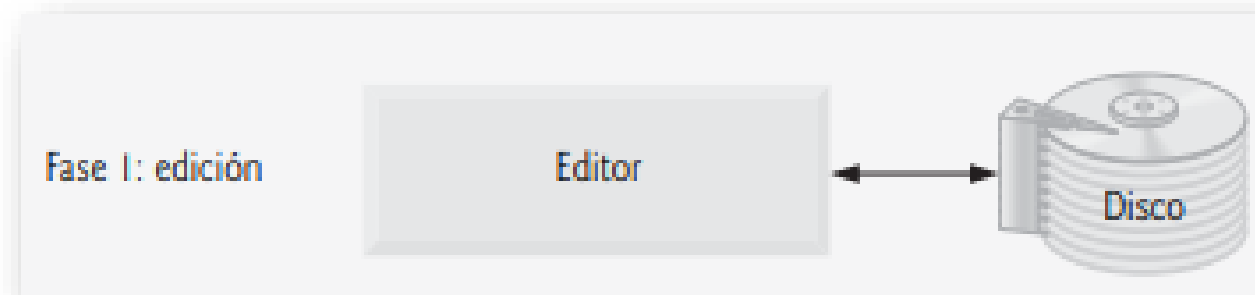
# Java Virtual Machine



# Fases de compilación

## Edición

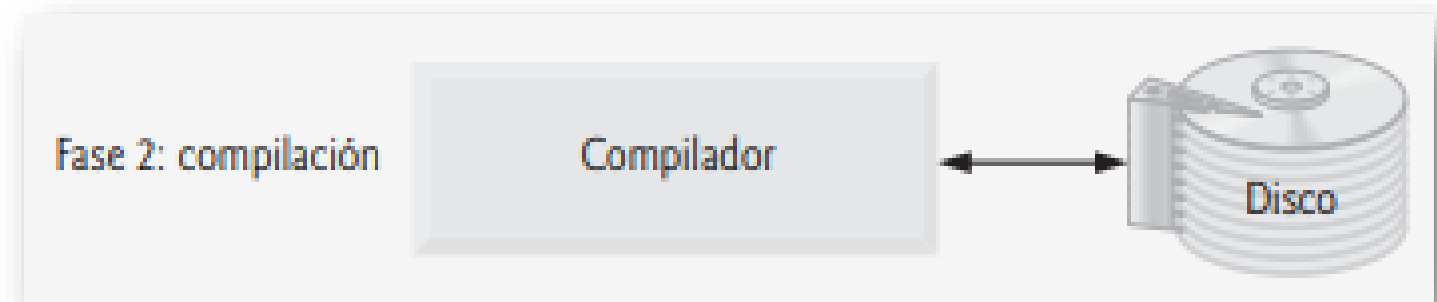
El programa Java se crea y edita en un editor de textos y se almacena como un archivo con extensión .java



## Fases de compilación

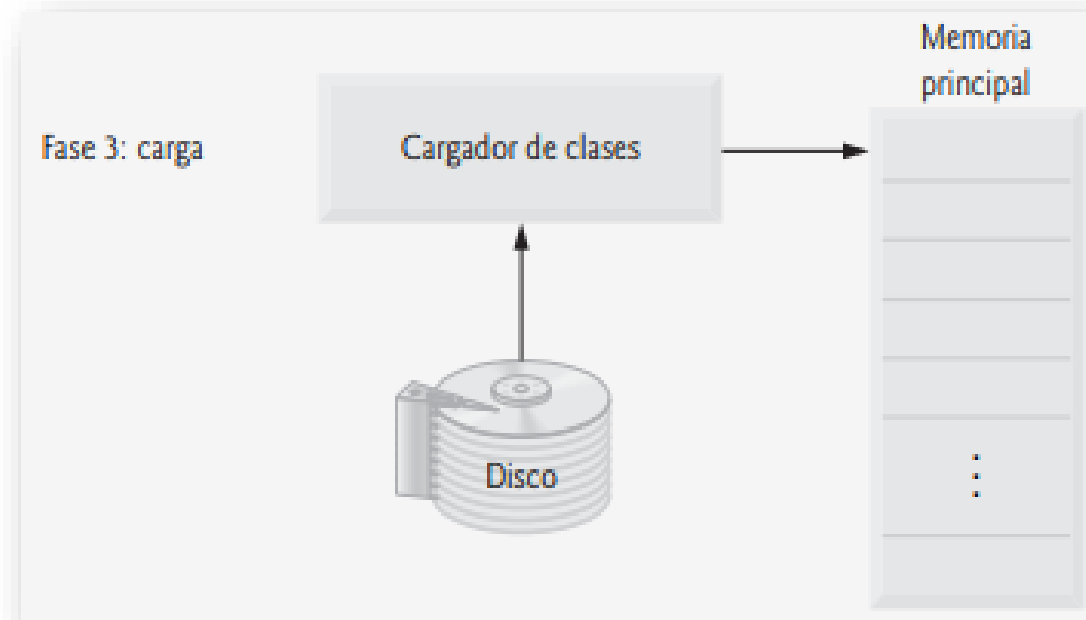
### Compilación

El compilador crea los códigos bytes y los almacena en un archivo con extensión .class



## Fases de compilación

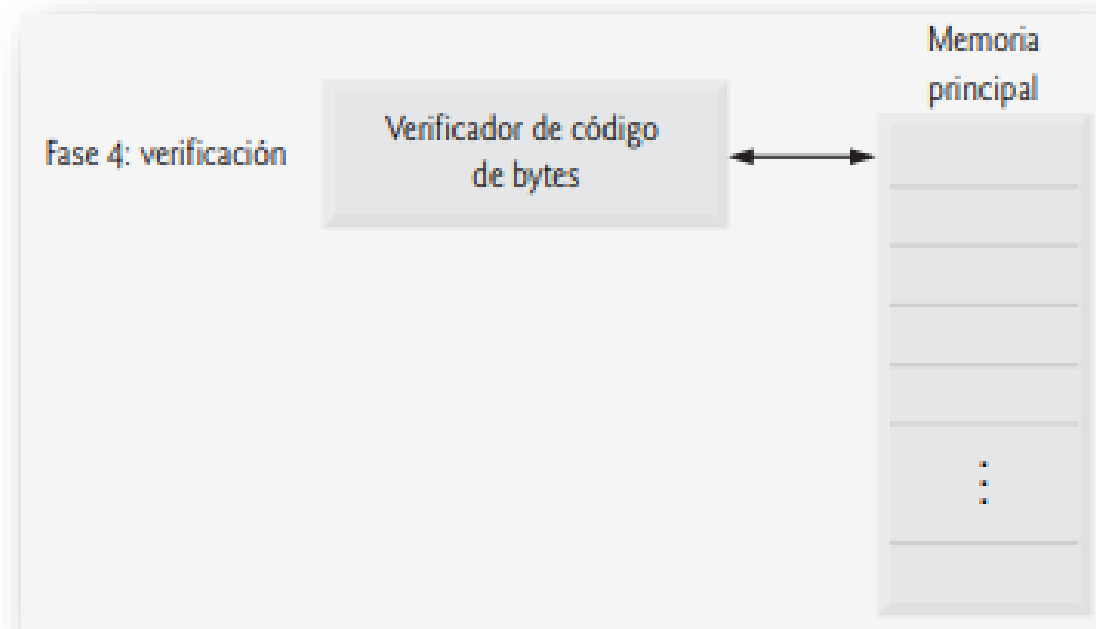
### Carga



El cargador de clases lee los códigos bytes almacenados en un archivo .class y los coloca en memoria

## Fases de compilación

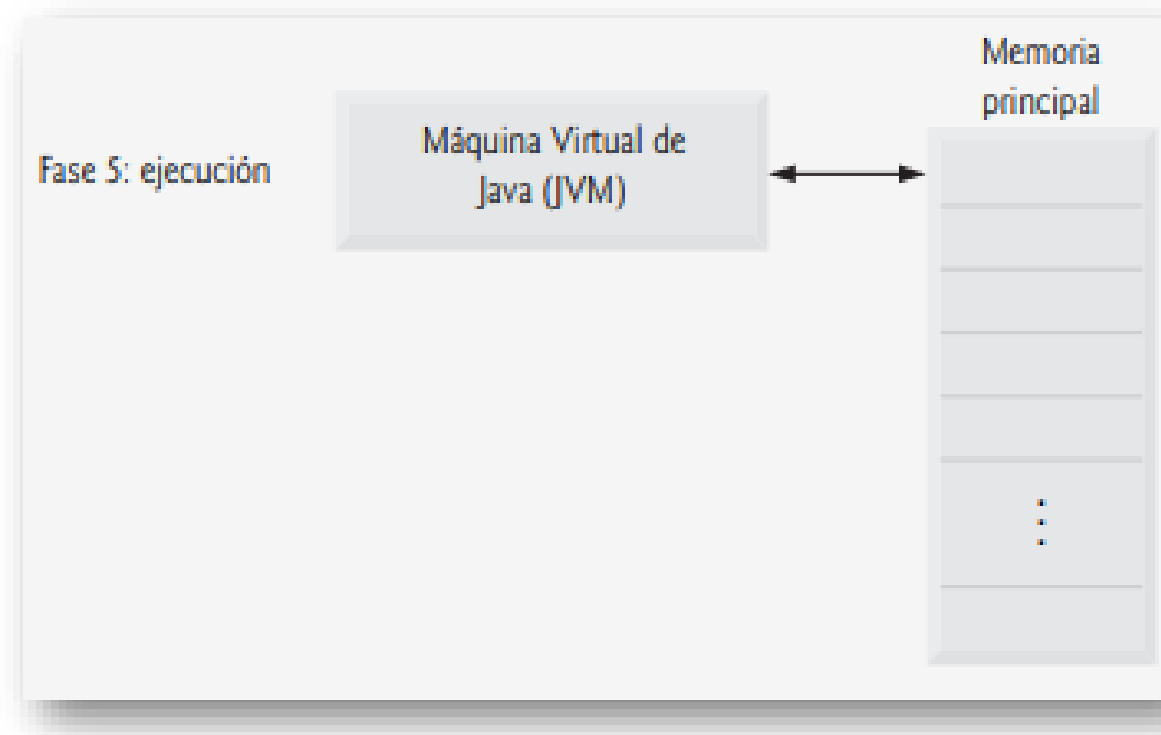
### Verificación



El verificador de códigos bytes confirma que todos estos sean validos y no violen las restricciones de seguridad de Java.

## Fases de compilación

### Ejecución



Para ejecutar el programa, la JVM lee los códigos bytes y los traduce "Just in Time" (JIT), es decir, los traduce a un lenguaje que la computadora host los pueda entender.

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 1

**Bibliografía**

---

# 2

---

## Módulo 2

Creación de una clase



## **Java JDK 17+**

[Java Archive Downloads - Java SE 17 \(oracle.com\)](#)

## **Eclipse IDE 2022-06**

[2022-06 R | Eclipse Packages](#)

## **JAVA\_HOME**

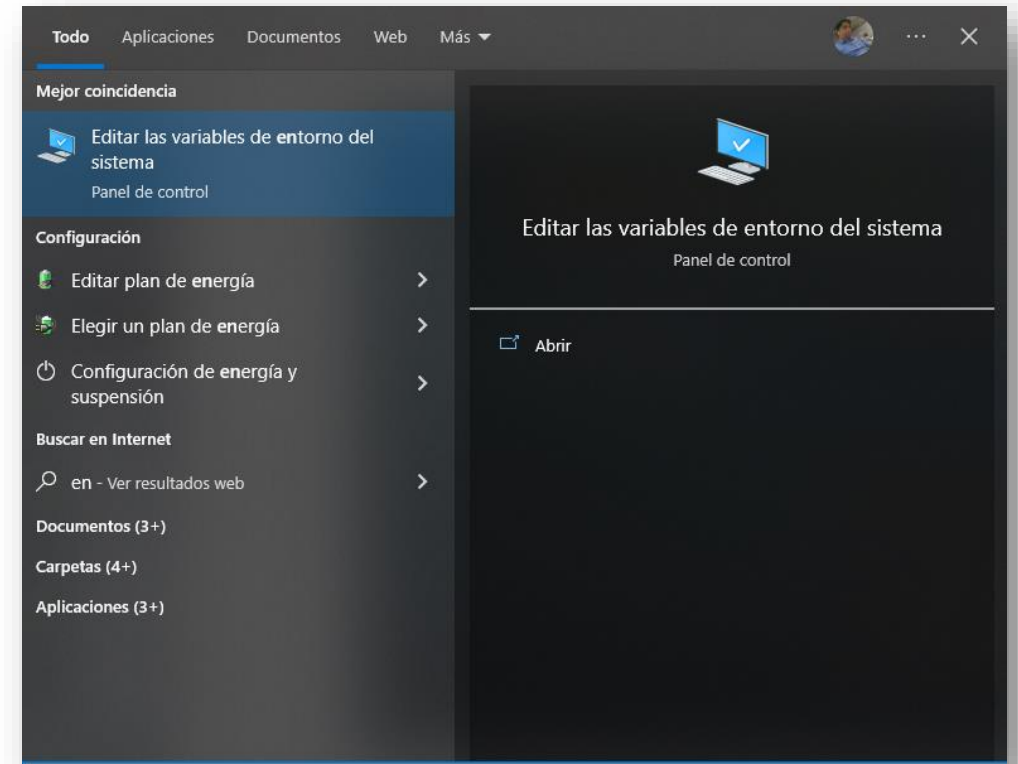
Es una variable de entorno del sistema que informa al sistema operativo sobre la ruta donde se encuentra instalado Java

## **PATH**

Es una variable de sistema que contiene un conjunto de rutas (directorios) donde residen los ficheros ejecutables (programas, scripts...) que el usuario ejecuta mediante una orden simple en la línea de comandos.

# Configuración de las variables del entorno – JAVA\_HOME

1. Abrir la edición de variables de entorno

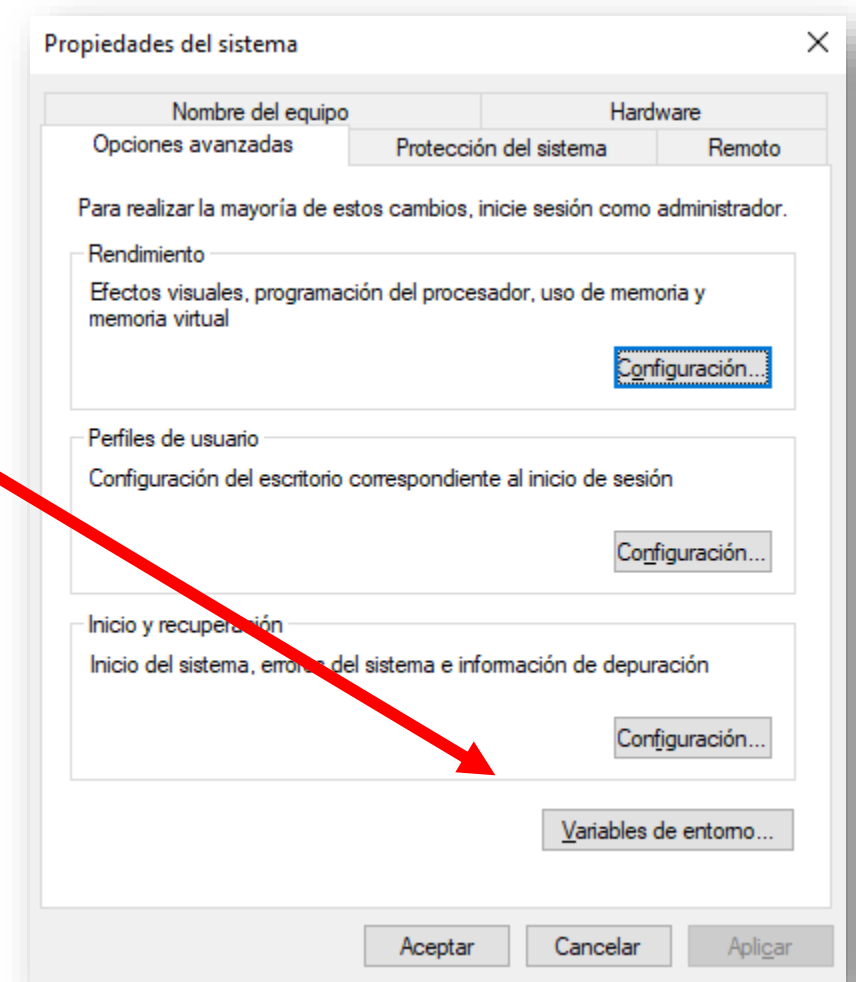


## Ejercicio 1

## Configuración de las variables del entorno – JAVA\_HOME

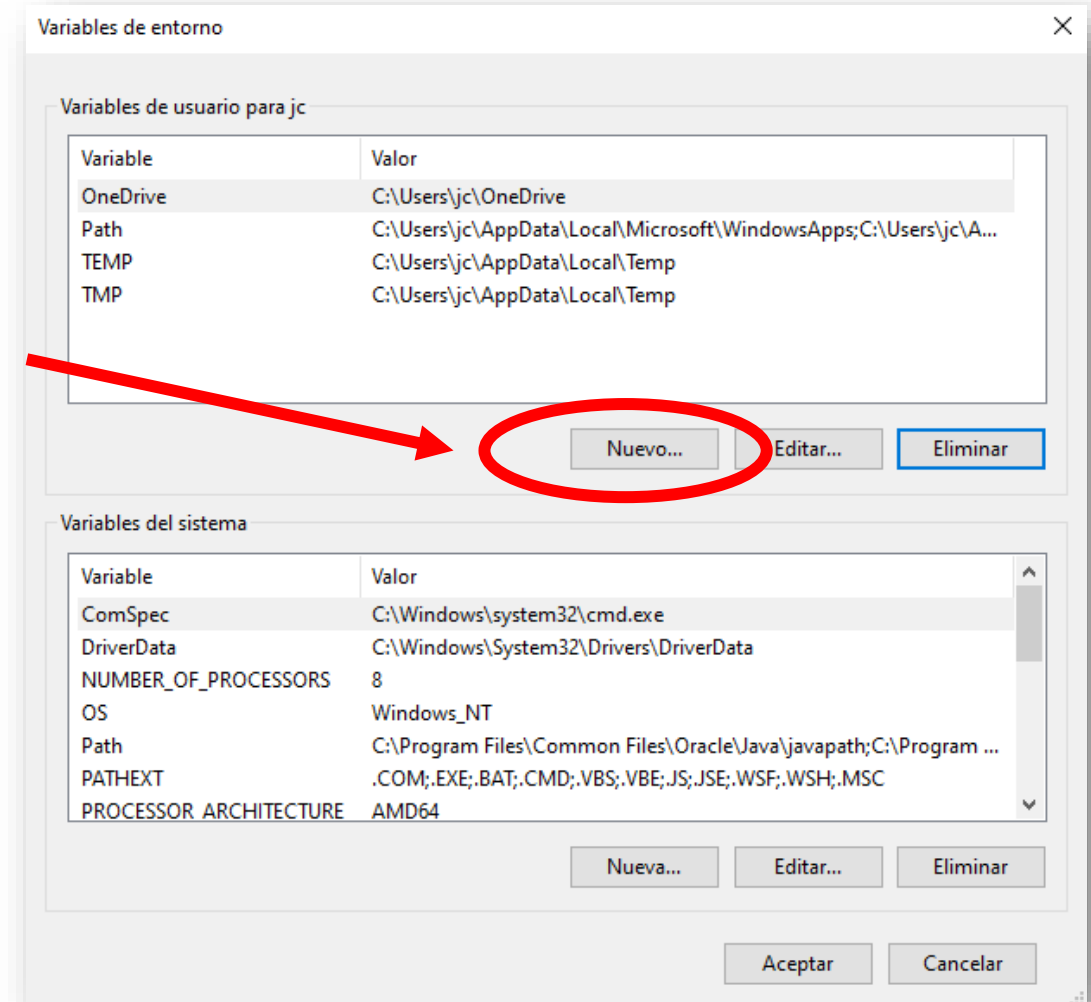
2. Seleccionar la opción de Variables de entorno.

### Ejercicio 1



## Configuración de las variables del entorno – JAVA\_HOME

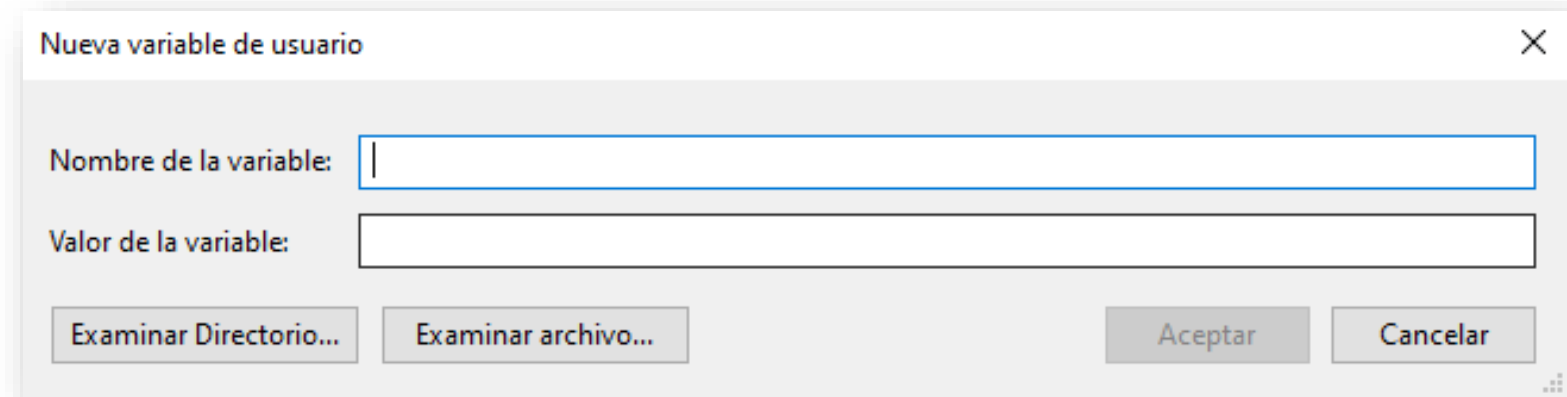
3. Seleccionar la opción de Nueva variable de entorno



## Configuración de las variables del entorno – JAVA\_HOME

4. Seleccionar la opción de Nueva variable de entorno

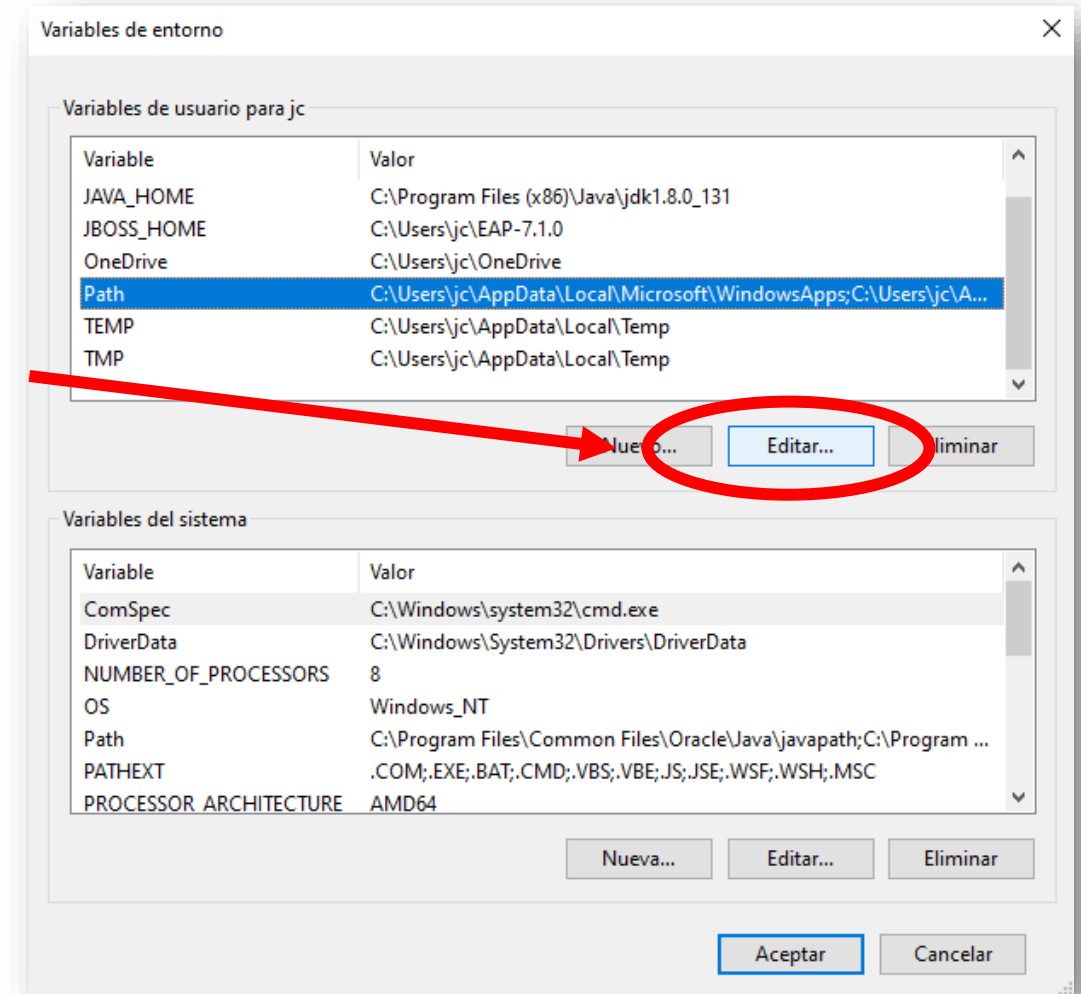
- **Nombre de la variable:** JAVA\_HOME
- **Valor de la variable:** Ruta a la instalación de Java JDK 8



The image shows a Windows dialog box titled "Nueva variable de usuario" (New User Variable). It has a close button (X) in the top right corner. The dialog contains two text input fields: "Nombre de la variable:" (Variable name) and "Valor de la variable:" (Variable value). Below these fields are four buttons: "Examinar Directorio..." (Browse Directory...), "Examinar archivo..." (Browse File...), "Aceptar" (OK), and "Cancelar" (Cancel). The "Aceptar" and "Cancelar" buttons are disabled (grayed out).

## Configuración de las variables del entorno – PATH

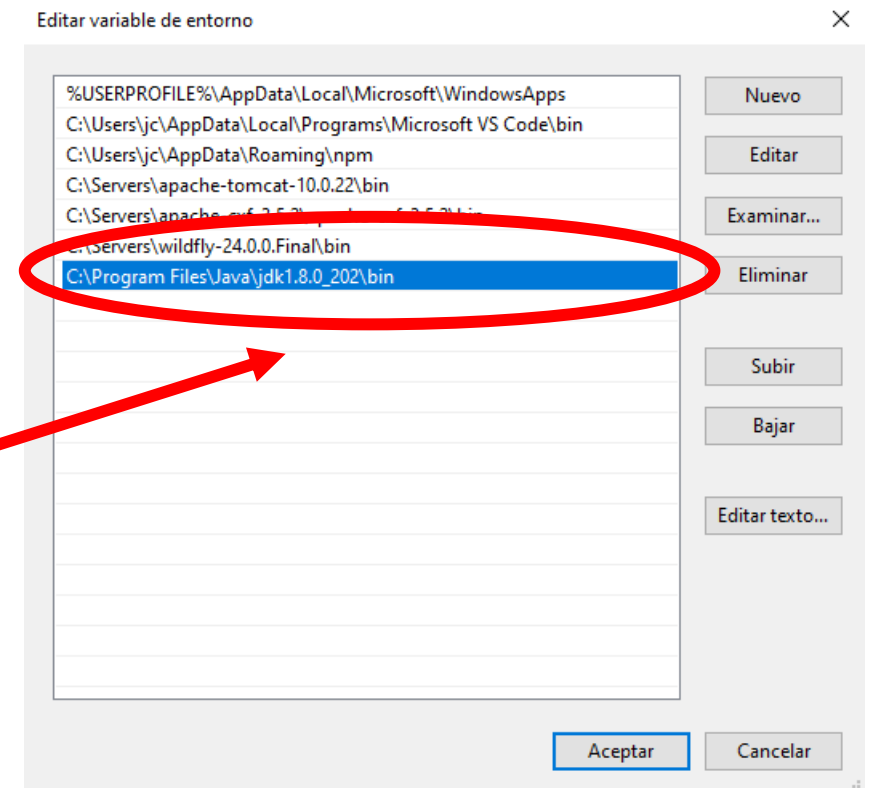
1. Seleccionar la variable Path de la lista de variables de entorno y dar clic en el botón de Editar.



## Configuración de las variables del entorno – PATH

### Ejercicio 1

2. Agregar a la variable Path la ruta hacia la carpeta bin de la instalación del JDK





# Creación del primer programa

## 1. Creación de un primer programa en Java

```
public class Bienvenida {  
  
    public static void main(String[] args) {  
  
        System.out.println("Bievenido a Java");  
    }  
}
```

## 2. Compilación manual de una clase en Java

```
$_ javac Bienvenida.java
```

## 3. Ejecución del método main.

```
$_ java Bienvenida
```

Un archivo **JAR** (por sus siglas en inglés, en inglés **J**ava **A**Rchive) es un tipo de archivo que permite ejecutar aplicaciones y herramientas escritas en el lenguaje Java.

Las siglas están deliberadamente escogidas para que coincidan con la palabra inglesa "jar" (tarro). Los archivos JAR están comprimidos con el formato ZIP y cambiada su extensión a .jar.



En Java un archivo manifest es un archivo específico contenido en un archivo Jar. Se usa para definir datos relativos a la extensión y al paquete.

Si se pretende usar el archivo jar como ejecutable, el archivo de manifest debe especificar la clase principal de la aplicación.



## Creación de un archivo JAR

1. Crear y ejecutar un archivo jar

```
$_ jar cvfe [Nombre_Archivo.jar] MANIFEST.MF [file1..fileN]
```

```
$_ java -jar [Nombre_Archivo.jar]
```

2. Ver contenido de un archivo JAR.

```
$_ jar tvf [Nombre_Archivo.jar]
```

3. Descomprimir un archivo JAR.

```
$_ jar xvf [Nombre_Archivo.jar]
```

# 3

---

## Módulo 3

Datos en el carro

## Secuencias de escape

Secuencia	Descripción
\n	Nueva línea. Coloca el cursor de la pantalla al inicio de la siguiente línea.
\t	Tabulador horizontal. Desplaza el cursor de la pantalla hasta la siguiente posición de tabulación.
\r	Retorno de carro. Coloca el cursor de la pantalla al inicio de la línea actual; no avanza a la siguiente línea.
\\	Barra diagonal inversa. Se usa para imprimir un carácter de barra diagonal inversa.
\"	Doble comilla. Se usa para imprimir un carácter de doble comilla.

## Operadores aritmeticos

Operación	Operador aritmético	Expresión en Java
Suma	+	variable1 + variable2
Resta	-	variable1 - variable2
Multiplicación	*	variable1 * variable2
División	/	variable1 / variable2
Residuo	%	variable1 % variable2

## Operadores relacionales

Operador estándar	Instrucción de java	Expresión en Java
=	x == y	x igual a y
!=	x != y	x es diferente de y
>	x > y	x es mayor a y
<	x < y	x es menor a y
<=	x <= y	x es menor o igual a y
>=	x >= y	x es mayor o igual a y



## Banderas de formateo

Operador de formateo	
%c	Carácter
%d	Decimal
%i	Entero
%s	Cadena de texto
%f	Punto flotante

## Tipos de variables

Nombre	Tipo	Ocupa (bytes)	Rango	Wrapper
byte	Entero	1	-128 a 127	Byte
short	Entero	2	-32768 a 32767	Short
int	Entero	4	2,000,000,000	Integer
long	Entero	8	Muy grande	Long
float	Decimal simple	4	Muy grande	Float
double	Decimal double	8	Muy grande	Double
char	Carácter simple	2	---	Character
String	Cadena de texto	---	---	---
boolean	F/V	1	---	Boolean

## Desarrollar los siguientes programas

1. Escriba una aplicación que reciba del usuario un número compuesto por cinco dígitos, que separe ese número en sus dígitos individuales y los imprima, cada uno separado de los demás por tres espacios. Por ejemplo, si el usuario escribe el número 42339, el programa debe imprimir:

```
4  2  3  3  9
```

Suponga que el usuario escribe el número correcto de dígitos. ¿Qué ocurre cuando ejecuta el programa y escribe un número con más de cinco dígitos? ¿Qué ocurre cuando ejecuta el programa y escribe un número con menos de cinco dígitos?

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 1

**Bibliografía**

---

# 4

---

## Módulo 4

Descripción de objetos y  
clases

Se refiere A una teoría O conjunto de teorías que sirve de modelo A seguir para resolver problemas O situaciones determinadas que se planteen.

**Paradigma**

---

## **Paradigma de programación**

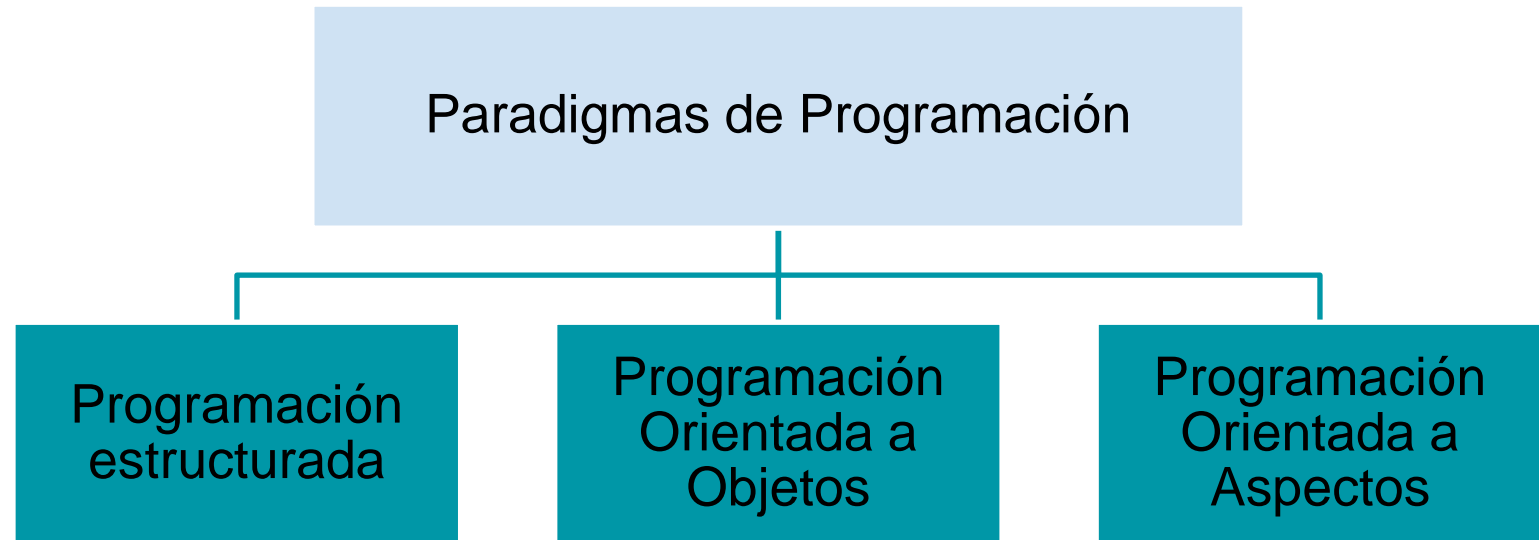
---

Un paradigma de programación es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales.

El comportamiento del programa es llevado a cabo por objetos, entidades que representan elementos del problema a resolver y tienen atributos y comportamiento.



# Paradigmas



## Lenguajes que usan este tipo de programación:

- Fortran
- C
- Basic

## Características:

- Grandes cantidades de código
- Más difícil de mantener
- Poco reutilizable
- Código espagueti

# Lenguajes orientados a objetos

- Java
- C++
- .Net
- Python

## Ventajas:

- Modularización: código dividido en trozos
- Herencia: código reutilizable
- Encapsulamiento: ocultamiento de los estados de un objeto
- Polimorfismo: muchas formas que tienen los objetos para responder a un evento.

Consiste en trasladar la naturaleza de los objetos de la vida real al código de programación.

## Los objetos tienen:

- Estado
- Comportamiento
- Propiedades

## Ejemplo: objeto coche

- ¿Su estado? Detenido, avanzando, estacionado
- ¿Sus propiedades? Color, peso, tamaño, forma etc.
- ¿Su comportamiento? Arrancar, frenar, girar, acelerar

Modelo donde se establecen las características comunes de un grupo de objetos.



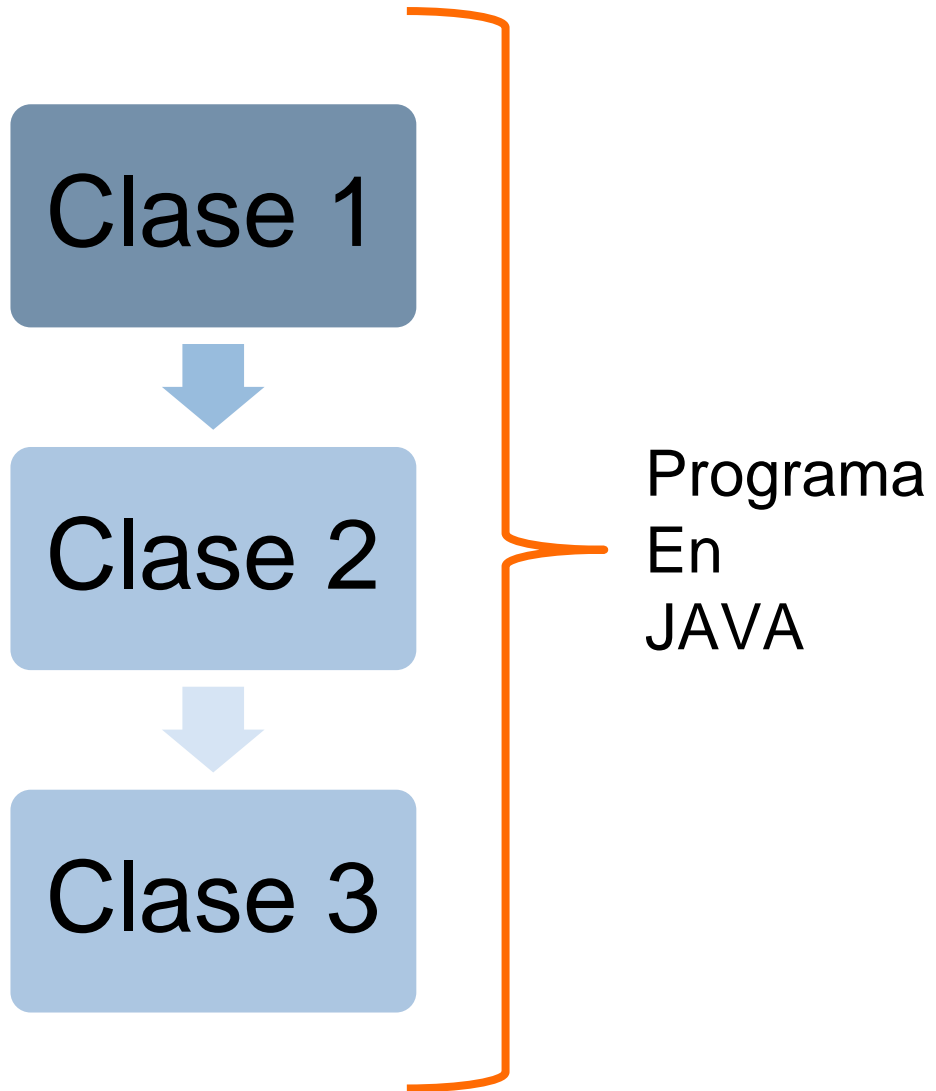
Es un ejemplar o un objeto que se ha creado a partir de la clase o molde

**OBJETO**



**Instancia**

# Modularización



## Atributos de Clase

Los atributos, también llamados datos o variables miembro son porciones de información que un objeto posee o conoce de sí mismo.

```
public class Carro {  
  
    private int numeroRuedas;  
    private String color;  
    private String modelo;  
    private String linea;  
}
```



## Métodos de clase o instancia

Un método es una abstracción de una operación que puede hacer o realizarse con un objeto. Una clase puede declarar cualquier número de métodos que lleven a cabo operaciones de lo más variado con los objetos. Los métodos de instancia operan sobre las variables de instancia de los objetos pero también tienen acceso a las variables de clase.

```
public class Carro {  
  
    private int numeroRuedas;  
  
    public void cambiarNumeroRuedas() {  
  
    }  
}
```

## Métodos set y get

Los métodos get y set, son métodos que se asocian a un atributo de la clase y que se usan en las clases para mostrar (get) o modificar (set) el valor de un atributo. El nombre del método siempre será get o set y a continuación el nombre del atributo.

```
public class Carro {  
  
    private int numeroRuedas;  
  
    public int getNumeroRuedas() {  
        return numeroRuedas;  
    }  
  
    public void setNumeroRuedas(int numeroRuedas) {  
        this.numeroRuedas = numeroRuedas;  
    }  
  
}
```

## Obtener el estado de un objeto

En muchas ocasiones necesitamos obtener el estado actual de un objeto, para lo cual podemos crear un método personalizado o podemos sobre escribir el método toString()

```
public class Carro {  
  
    private int numeroRuedas;  
  
    @Override  
    public String toString() {  
        return "Carro{" + "numeroRuedas=" + numeroRuedas + '}';  
    }  
}
```

Modelar una solución orientada a objetos para un Estudiante en un diagrama UML.

Implementar la clase Estudiante diseñada en el ejercicio anterior. La clase debe incluir:

- Métodos setter y getter
- Método toString()
- Constructor
- Implementar un método que calcule la calificación de un estudiante

Implementar un programa en Java que permita:

- Crear un objeto Estudiante a partir de los datos introducidos por el usuario.
- Calcular la calificación del estudiante.
- Obtener la calificación del estudiante.
- Obtener el estado actual del estudiante.

Cree una clase llamada Fecha, que incluya tres piezas de información como variables de instancia:

- Un mes (tipo int),
- Un día (tipo int)
- Un año (tipo int)

La clase debe tener:

- Proporcionar un métodos set y get para cada variable de instancia.
- Proporcione un método mostrarFecha, que muestre el mes, día y año, separados por barras diagonales (/).
- Escriba una aplicación de prueba que demuestre las capacidades de la clase Fecha

## Ejercicio 4

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 8

**Bibliografía**

---



# 5

---

## Módulo 5

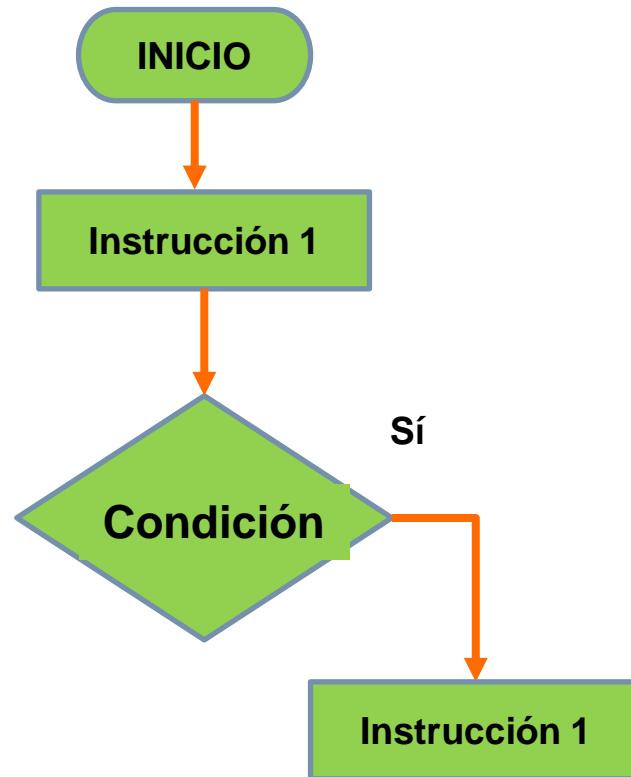
Estructuras condicionales  
y de control

Java tiene tres tipos de estructuras de selección:

- If
- If - else
- Switch - Case

# Estructuras condicionales

IF



## Código Java

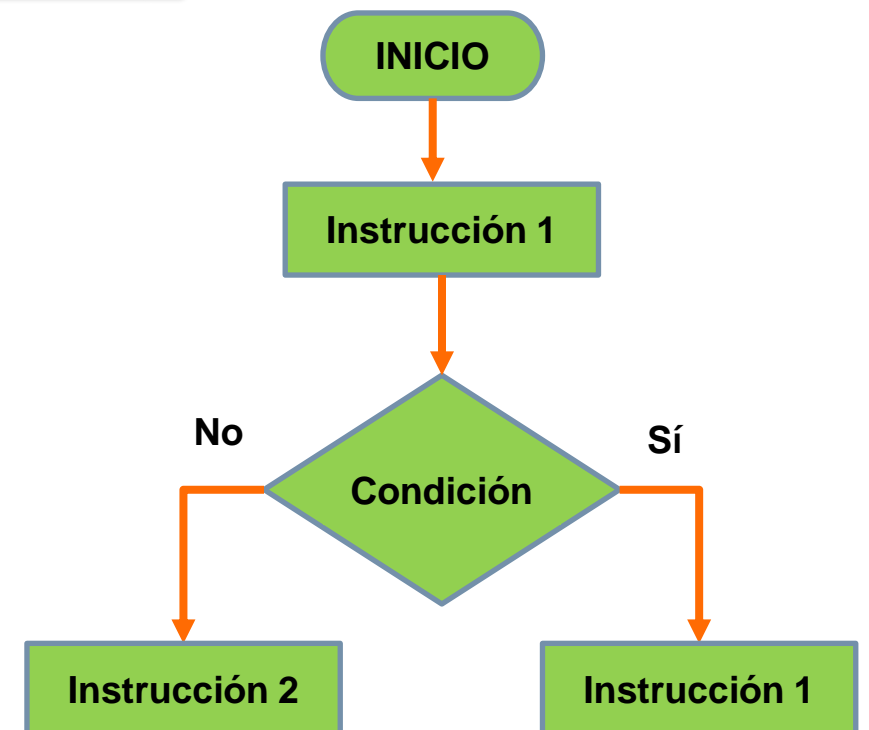
```
if (calificacion >= 60) {  
    System.out.print("Aprobado");  
}
```

# Estructuras condicionales

IF - ELSE

## Código Java

```
if (calificacion >= 60) {  
    System.out.print("Aprobado");  
} else {  
    System.out.print("Reprobado");  
}
```

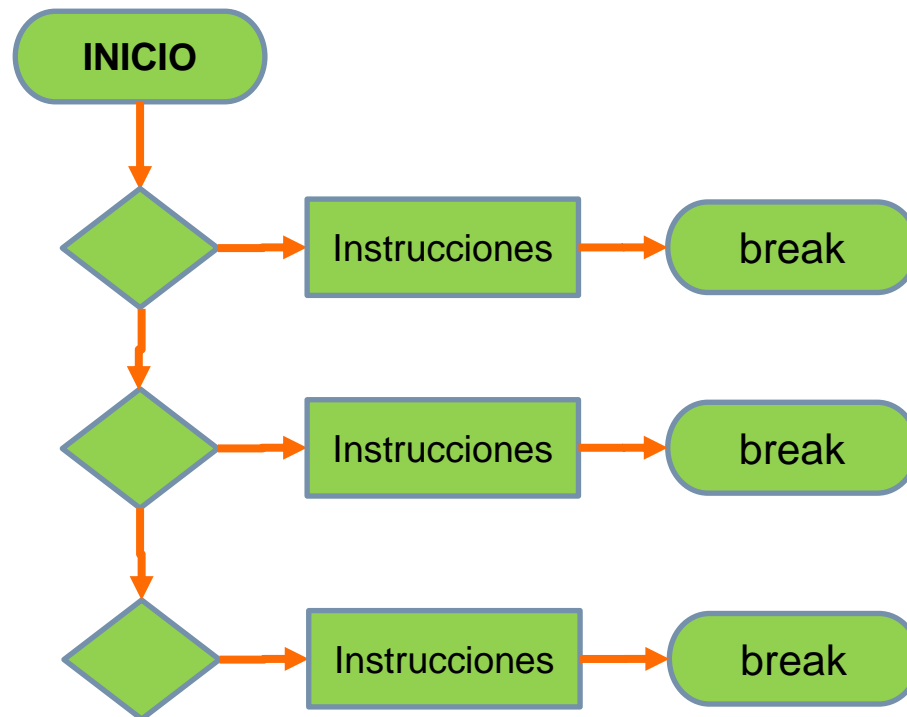


## Estructuras condicionales

### SWITCH - CASE

### Código Java

```
switch (calificacion) {  
    case 60:  
        System.out.print("Suficiente");  
        break;  
    case 80:  
        System.out.print("Bien hecho");  
        break;  
}
```

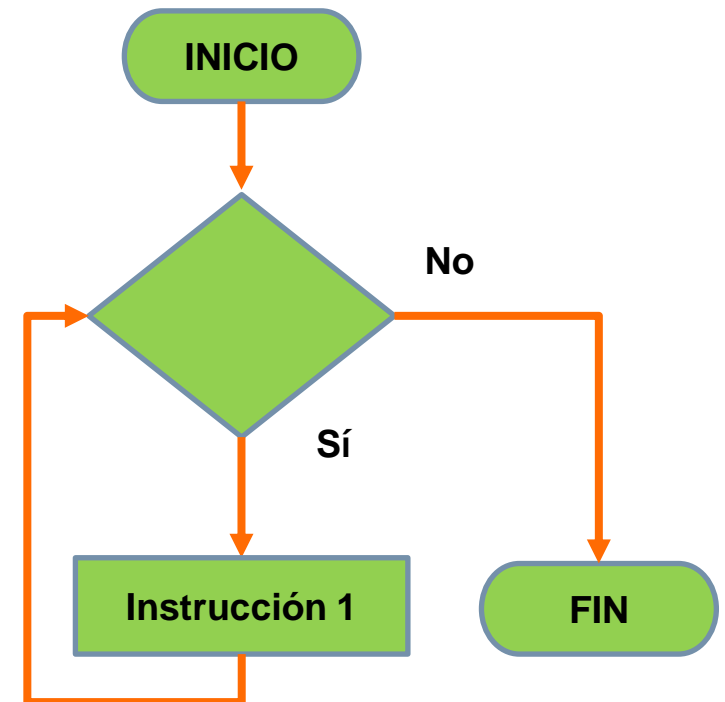


Java tiene tres tipo de estructuras de selección:

- While
- Do - While
- For

## Código Java

```
while (calificacion > 60) {  
    System.println("Aprobado");  
}
```

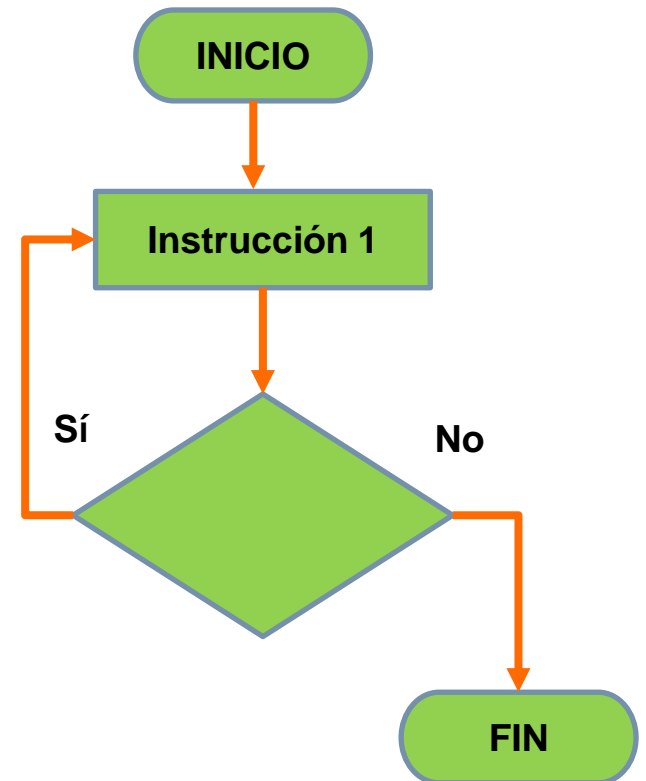


# Estructuras de control

## DO - WHILE

### Código Java

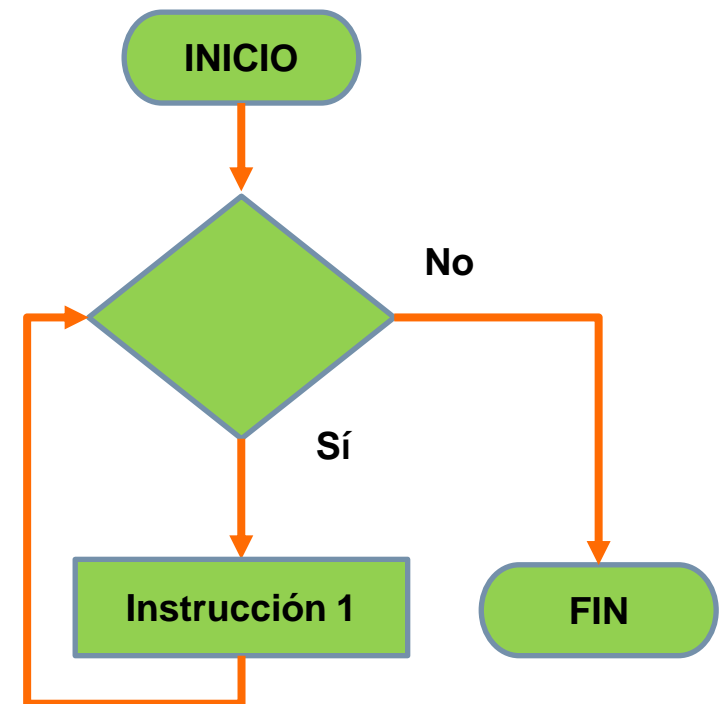
```
do {  
    System.println("Aprobado");  
} (calificacion > 60);
```





### Código Java

```
for (i=0; i < 10; i++) {  
    i++;  
}
```



## Ejercicios

1. Un palíndromo es una secuencia de caracteres que se lee igual al derecho y al revés. Por ejemplo, cada uno de los siguientes enteros de cinco dígitos es un palíndromo: 12321, 55555, 45554 y 11611. Escriba una aplicación que lea un entero de cinco dígitos y determine si es un palíndromo. Si el número no es de cinco dígitos, el programa debe mostrar un mensaje de error y permitir al usuario que introduzca un nuevo valor. (15 min)
2. Los factoriales se utilizan frecuentemente en los problemas de probabilidad. El factorial de un entero positivo  $n$  (se escribe como  $n!$ ) es igual al producto de los enteros positivos del 1 a  $n$ . Escriba una aplicación que evalúe los factoriales de los enteros del 1 al 5. Muestre los resultados en formato tabular. ¿Qué dificultad podría impedir que usted calculara el factorial de 20? (15 min)

3. Implementar un programa que permita crear diez objetos Estudiante e imprimir el promedio de la clase. (30 min)
- Implementar un programa que permita crear objetos Estudiante hasta que el usuario teclee un numero negativo e imprima el promedio de la clase.
  - Contar el número de resultados de estudiantes.
  - Mostrar un resumen de los resultados de los estudiantes, indicando el número de estudiantes que aprobaron y el número de estudiantes que reprobaron.
  - Si más de ocho estudiantes aprobaron el examen, imprimir el mensaje “Aumentar la colegiatura”.

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 4, 5

**Bibliografía**

---

# 6

---

## Módulo 6

Métodos

Las funciones permiten automatizar tareas que requerimos con frecuencia y que además se pueden generalizar por medio de parámetros o argumentos.

**Modificador de  
acceso**      **Tipo de  
retorno**      **Nombre de  
método**

```
public String toString() {  
    return "Carro{" + "numeroRuedas=" + numeroRuedas + "}";  
}
```

**Retorno**

### Métodos de instancia

Los métodos son subrutinas que manipulan los datos definidos por la clase y, en muchos casos, brindan acceso a esos datos.

```
public class Carro {  
  
    private int numeroRuedas;  
  
    public void cambiarNumeroRuedas() {  
  
    }  
}
```

### Métodos estáticos

Un método estático en Java es un método que pertenece a la clase y no al objeto.

.

```
public class Bienvenida {  
  
    public static void main(String[] args) {  
  
        System.out.println(Bienvenida.saludar());  
    }  
  
    public static String saludar() {  
        return "Hola desde un metodo Estatico";  
    }  
}
```



## Sobrecarga de métodos

Pueden declararse métodos con el mismo nombre en la misma clase, siempre y cuando tengan distintos conjuntos de parámetros (determinados en base al número, tipos y orden de los parámetros). A esto se le conoce como **sobrecarga de métodos**.

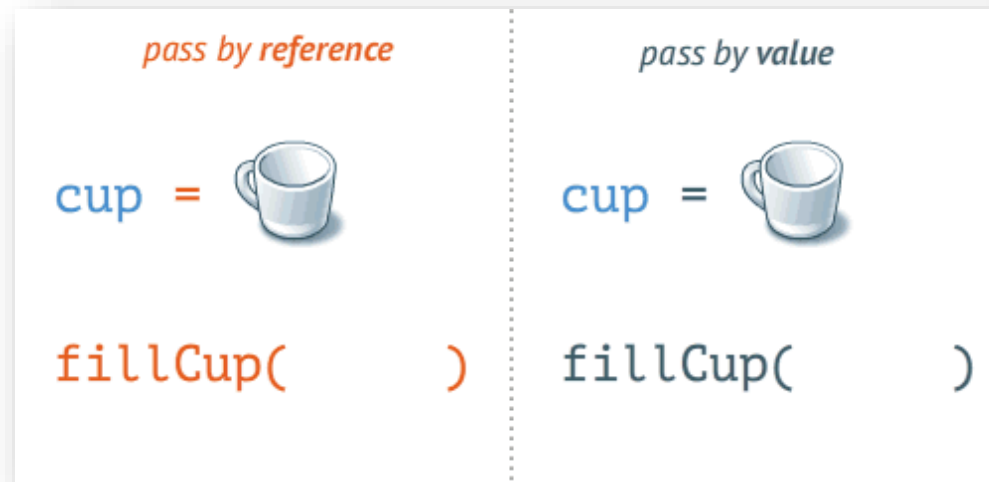
```
public static String saludar(String mensaje) {  
    return mensaje;  
}  
  
public static String saludar(Boolean bandera) {  
    return (bandera) ? "Verdadero" : "Falso";  
}
```

### Paso por valor

El paso por valor significa que al método en la variable del argumento le llega una copia del valor en el caso de un tipo primitivo de datos o una copia del puntero a la dirección de memoria del objeto.

### Paso por referencia

En el paso por referencia el argumento contiene un puntero con la dirección de memoria de la variable.

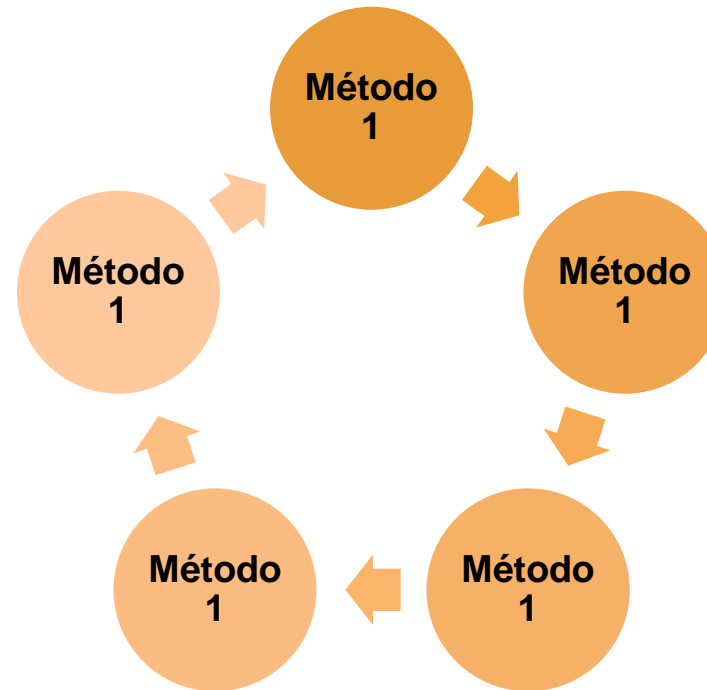


### De acuerdo al siguiente programa...

- ¿Qué pasa en el programa?
- ¿Se cambia el valor de la variable dentro del método?
- ¿Por qué sí o por que no?

```
public class PasoPorValor {  
  
    public static void saludar(String mensaje) {  
        mensaje = "Saludos desde el metodo saludar";  
        System.out.println(mensaje);  
    }  
  
    public static void main(String[] args) {  
  
        String mensajeOriginal = "Saludos desde main";  
        saludar(mensajeOriginal);  
        System.out.println(mensajeOriginal);  
    }  
}
```

En Java los métodos pueden llamarse a sí mismos. Si dentro de un método existe la llamada a sí mismo decimos que el método es recursivo. Cuando un método se llama a sí mismo, se asigna espacio en la pila para las nuevas variables locales y parámetros.



## Recursividad

De acuerdo al siguiente programa...

- ¿Qué pasa en el programa?
- ¿Implica algún riesgo la ejecución?
- ¿Por qué sí o por que no?

```
public class Recursividad {  
  
    void imprimir(int x) {  
        System.out.println(x);  
        imprimir(x-1);  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re = new Recursividad();  
        re.imprimir(5);  
    }  
}
```

De acuerdo al siguiente programa...

- ¿Qué pasa en el programa?
- ¿Implica algún riesgo la ejecución?
- ¿Por qué sí o por que no?

```
public class Recursividad {  
  
    void imprimir(int x) {  
        System.out.println(x);  
        imprimir(x-1);  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re = new Recursividad();  
        re.imprimir(5);  
    }  
}
```

Escriba una aplicación que juegue a “adivina el número” de la siguiente manera: su programa elige el número a adivinar, seleccionando un entero aleatorio en el rango de 1 a 1000. La aplicación muestra el indicador Adivine un número entre 1 y 1000. El jugador escribe su primer intento.

Si la respuesta del jugador es incorrecta, su programa debe mostrar el mensaje Demasiado alto. Intente de nuevo. o Demasiado bajo. Intente de nuevo., para ayudar a que el jugador “se acerque” a la respuesta correcta. El programa debe pedir al usuario que escriba su siguiente intento. Cuando el usuario escriba la respuesta correcta, muestre el mensaje Felicidades. Adivino el numero! y permita que el usuario elija si desea jugar otra vez. El programa debe mostrar también el número de intentos hechos al adivinar el numero. (30 min)

## Ejercicio 1

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 6

**Bibliografía**

---



# 7

---

## Módulo 7

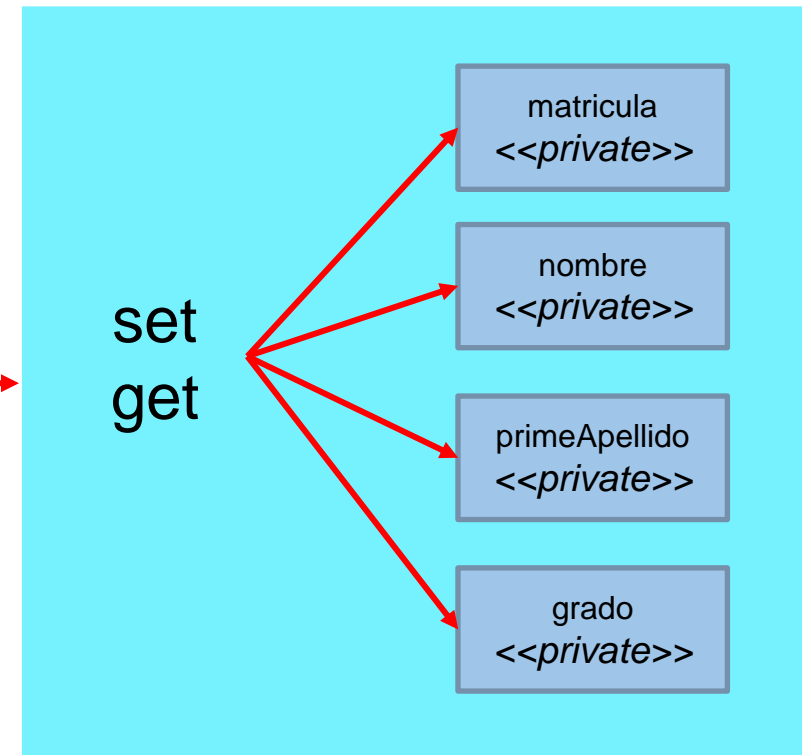
Encapsulamiento

El término encapsulamiento en Java, consiste en ocultar atributos de un objeto de manera que solo se pueda cambiar mediante operaciones definidas en ese objeto.

## Objetos



## Estudiante



Los modificadores de acceso en Java ayudan a restringir el alcance de una clase, constructor, variable, método o miembro de datos. Hay cuatro tipos de modificadores de acceso disponibles en Java:

Modificador/Acceso	Clase	Paquete	Subclase
public	Sí	Sí	Sí
protected	Sí	Sí	Sí
default	Sí	Sí	No
private	Sí	No	No

## Modificadores de acceso

Los modificadores de acceso en Java ayudan a restringir el alcance de una clase, constructor, variable, método o miembro de datos. Hay cuatro tipos de modificadores de acceso disponibles en Java:

Modificador	Descripción
public	El modificador de acceso público tiene el alcance más amplio entre todos los demás modificadores de acceso. Las clases, métodos o miembros de datos que se declaran como públicos son accesibles desde cualquier lugar del programa.
protected	Los métodos o miembros de datos declarados como protected son accesibles dentro del mismo paquete o sub-clases en paquetes diferentes.
default	Los miembros de datos, clase o métodos que no se declaran utilizando ningún modificador de acceso, es decir, que tengan un modificador de acceso predeterminado, solo son accesibles dentro del mismo paquete.
private	Los métodos o los miembros de datos declarados como privados solo son accesibles dentro de la clase en la que se declaran.

# 8

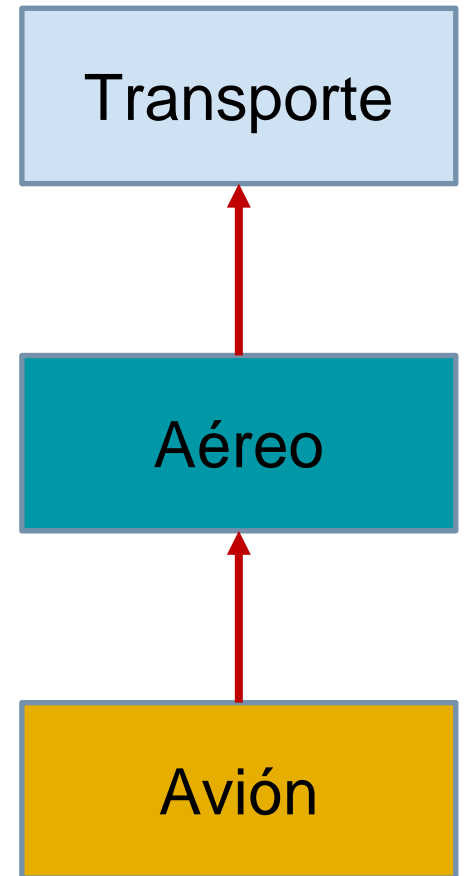
---

## Módulo 8

Herencia

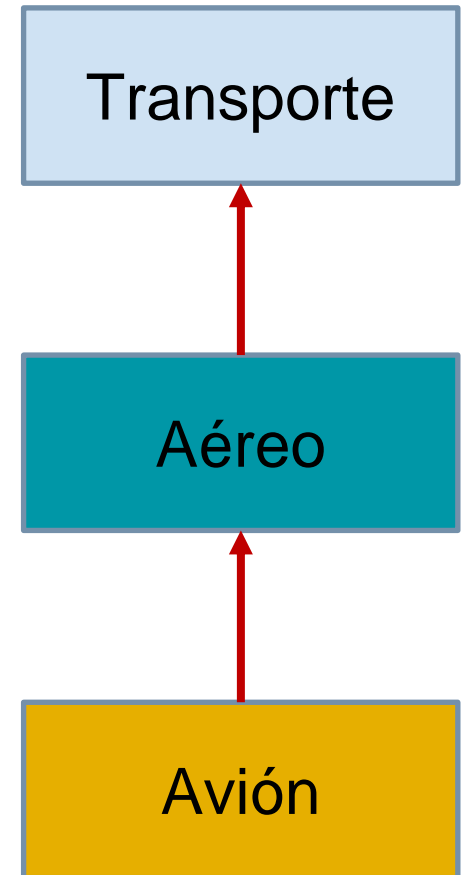
## Herencia

Es el mecanismo en Java por el cual una clase permite heredar las características (atributos y métodos) de otra clase. Aprenda más a continuación.



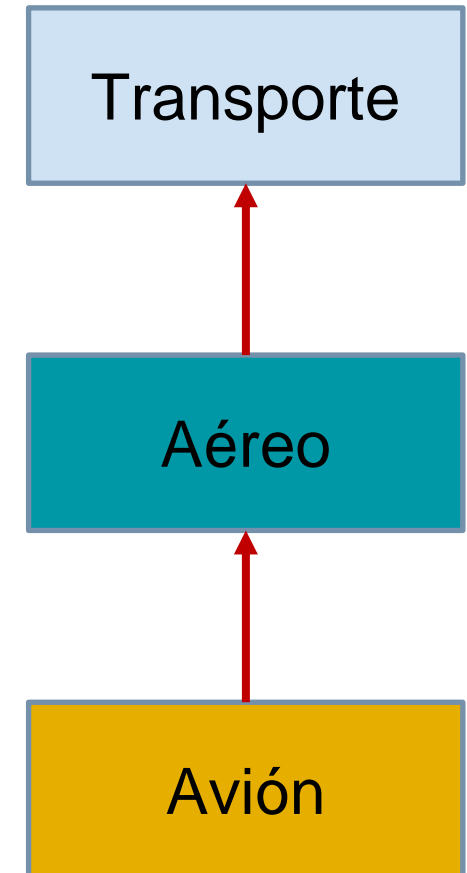
## Superclase

La clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal).



### Subclase

La clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos además de los campos y métodos de la superclase.





# Herencia

```
public class Transporte {  
}
```

```
public class Aereo extends Transporte {  
}
```

```
public class Avion extends Aereo {  
}
```

Transporte

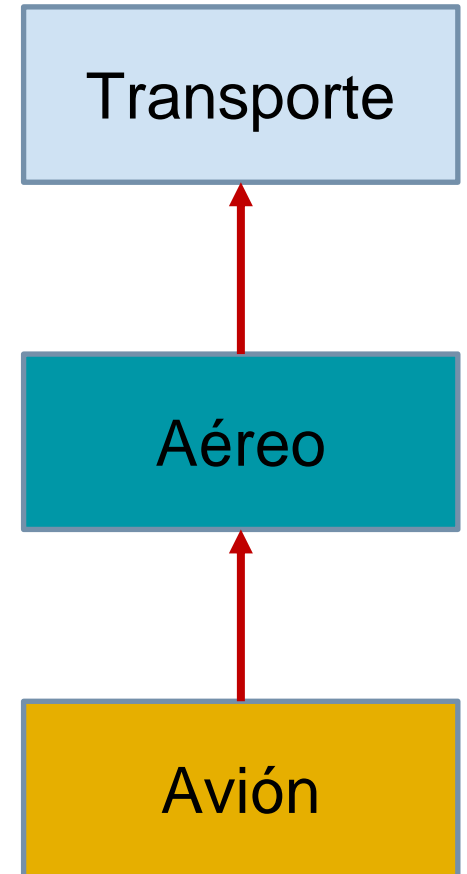
Aéreo

Avión



### La clase Object

Es la clase raíz de todo el árbol de la jerarquía de clases Java, y proporciona un cierto número de métodos de utilidad general que pueden utilizar todos los objetos.



## Sobre escritura de metodos

---

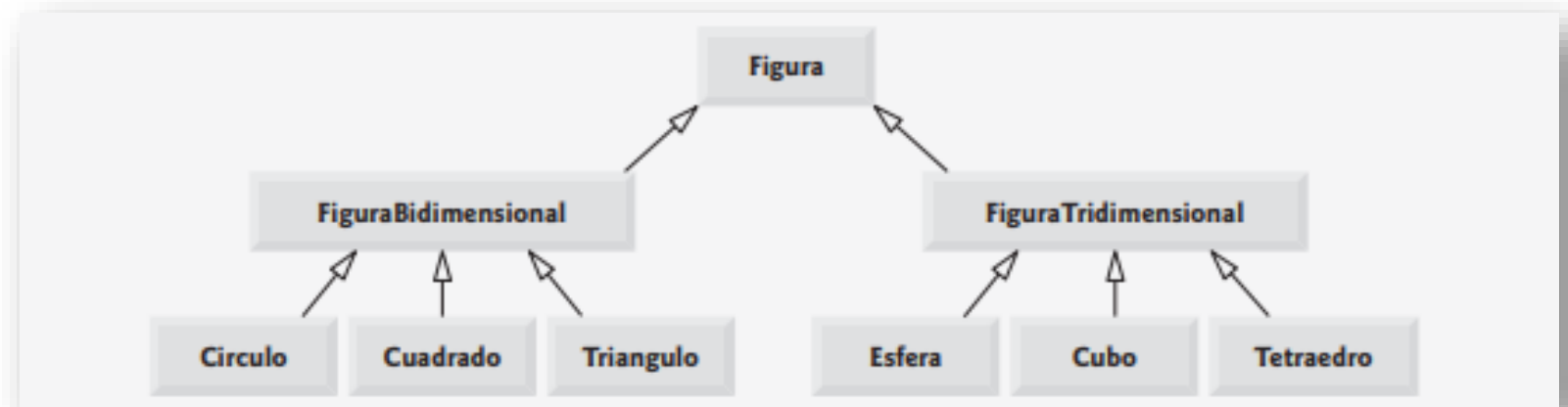
La sobreescritura de métodos en Java es un concepto fundamental de la programación orientada a objetos que permite a una clase proporcionar una implementación específica de un método que ya está definido en una de sus clases ancestrales (clase padre o superclase).

La sobreescritura de métodos se utiliza para modificar o extender el comportamiento de un método heredado sin cambiar su firma.

## Sobre escritura de metodos

equals	Este método compara la igualdad entre dos objetos; devuelve true si son iguales y false en caso contrario. Cuando debe compararse la igualdad entre objetos de una clase en particular, la clase debe sobrescribir el método equals para comparar el contenido de los dos objetos.
getClass	Todo objeto en Java conoce su tipo en tiempo de ejecución. El método getClass devuelve un objeto de la clase Class el cual contiene información acerca del tipo del objeto, como el nombre de su clase.
toString	Este método devuelve una representación String de un objeto. La implementación predeterminada de este método devuelve el nombre del paquete y el nombre de la clase del objeto, seguidos por una representación hexadecimal del valor devuelto por el método hashCode del objeto.

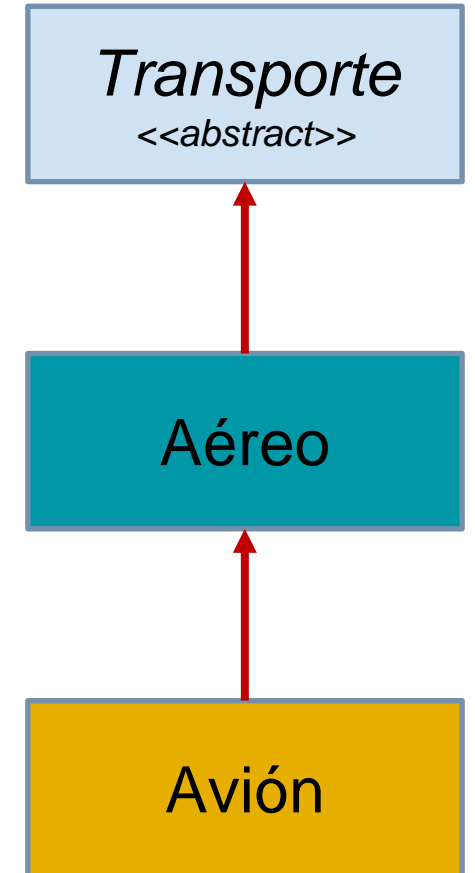
Implementar el siguiente diagrama de clases con Herencia



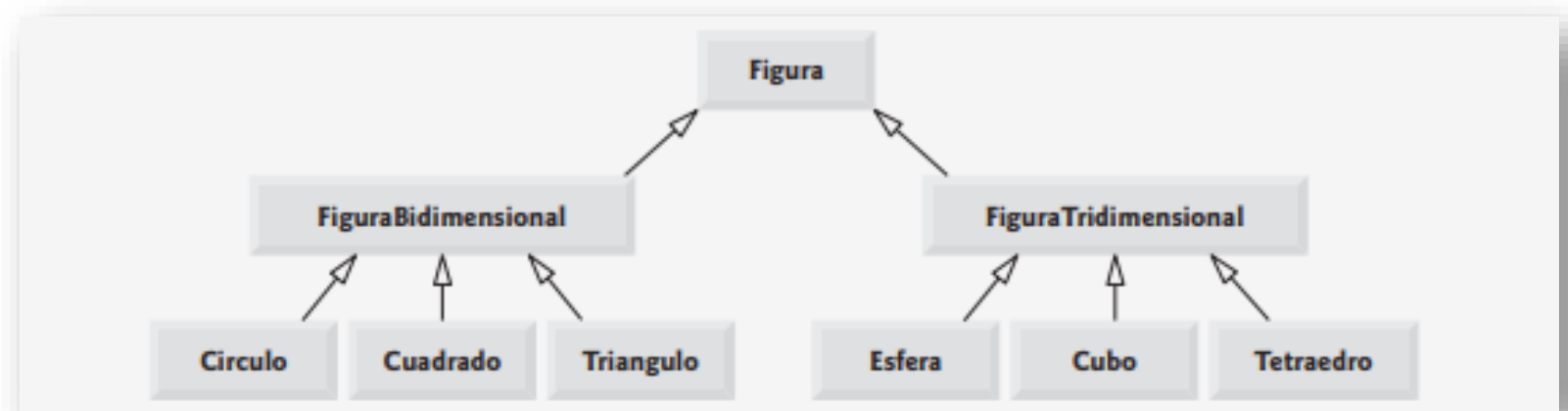
### Clase Abstracta

Una clase abstracta es aquella en la cual su nivel de abstracción es mínimo, dejando que las clases hijas se ocupen de los detalles.

Una clase abstracta puede contener métodos, atributos y constructores.



Modificar la clase *Figura* como abstracta



# **Java Cómo Programar**

Deitel & Deitel

Capítulo 9

**Bibliografía**

---



# 9

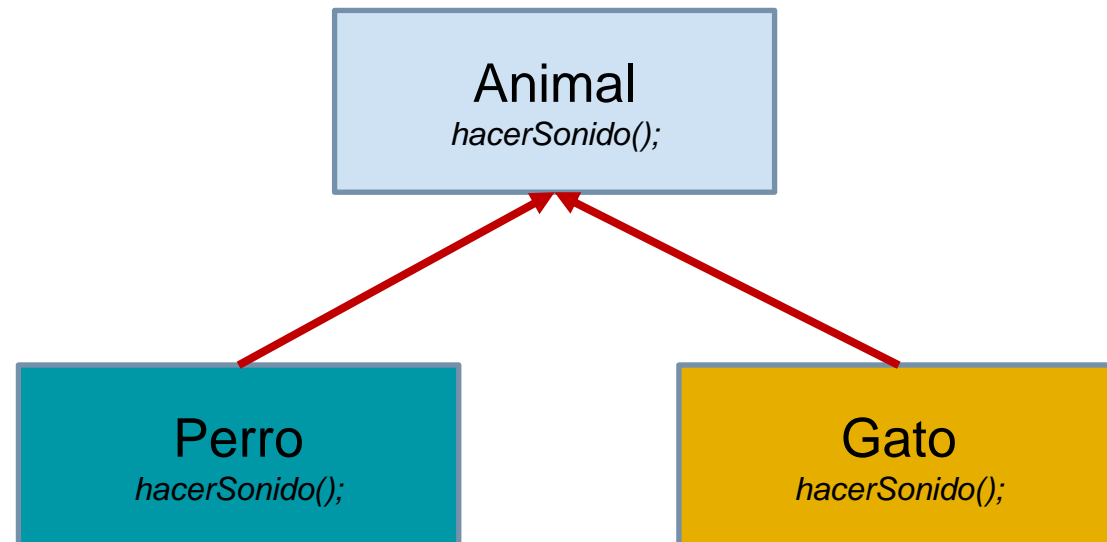
---

## Módulo 9

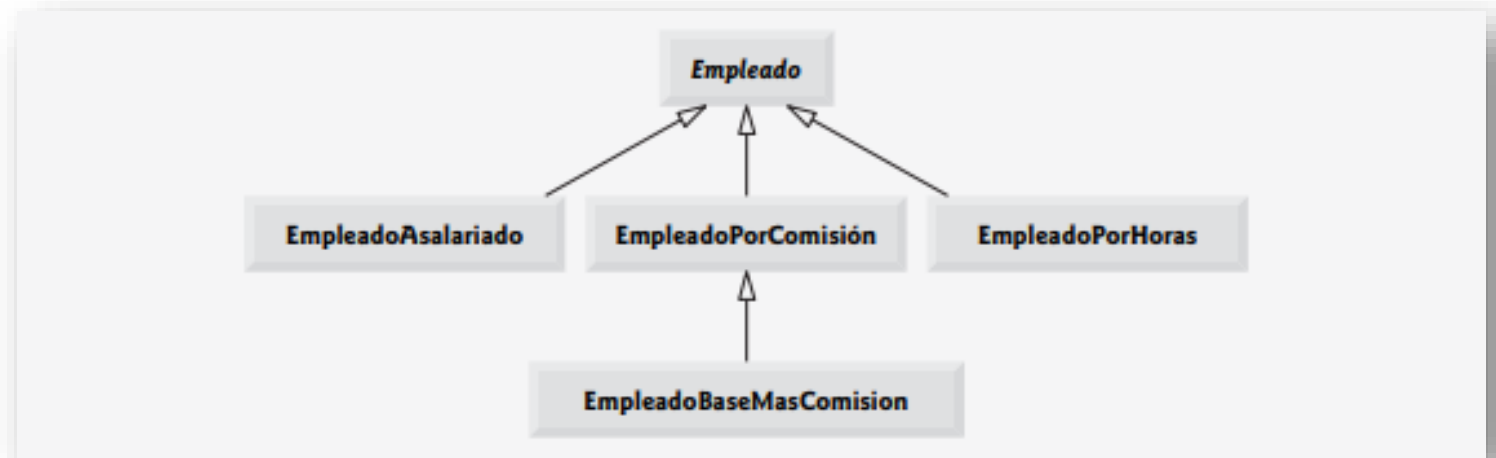
Polimorfismo

En programación orientada a objetos, polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros.

Es una característica de la programación orientada a objetos que permite llamar a métodos con igual nombre pero que pertenecen a clases distintas.



Implementar el siguiente diagrama de clases



Cada clase debe contener:

1. Atributos
2. Métodos setter y getter
3. Constructor(es)

Cada clase tiene las siguientes especificaciones:

	ingresos	toString
Empleado	abstract	<i>primerNombre apellidoPaterno</i> número de seguro social: <i>NSS</i>
Empleado-Asalariado	salarioSemanal	empleado asalariado: <i>primerNombre apellidoPaterno</i> número de seguro social: <i>NSS</i> salario semanal: <i>salarioSemanal</i>
EmpleadoPor-Horas	if horas <= 40 sueldo * horas else if horas > 40 40 * sueldo + ( horas - 40 ) * sueldo * 1.5	empleado por horas: <i>primerNombre apellidoPaterno</i> número de seguro social: <i>NSS</i> sueldo por horas: <i>sueldo</i> ; horas trabajadas: <i>horas</i>
EmpleadoPor-Comisión	tarifaComisión * ventasBrutas	empleado por comisión: <i>primerNombre apellidoPaterno</i> número de seguro social: <i>NSS</i> ventas brutas: <i>ventasBrutas</i> ; tarifa de comisión: <i>tarifaComisión</i>
Empleado-BaseMas-Comision	( tarifaComision * ventasBrutas ) + salarioBase	empleado por comisión con salario base: <i>primerNombre apellidoPaterno</i> número de seguro social: <i>NSS</i> ventas brutas: <i>ventasBrutas</i> ; tarifa de comisión: <i>tarifaComision</i> ; salario base: <i>salarioBase</i>

## Ejercicio 1

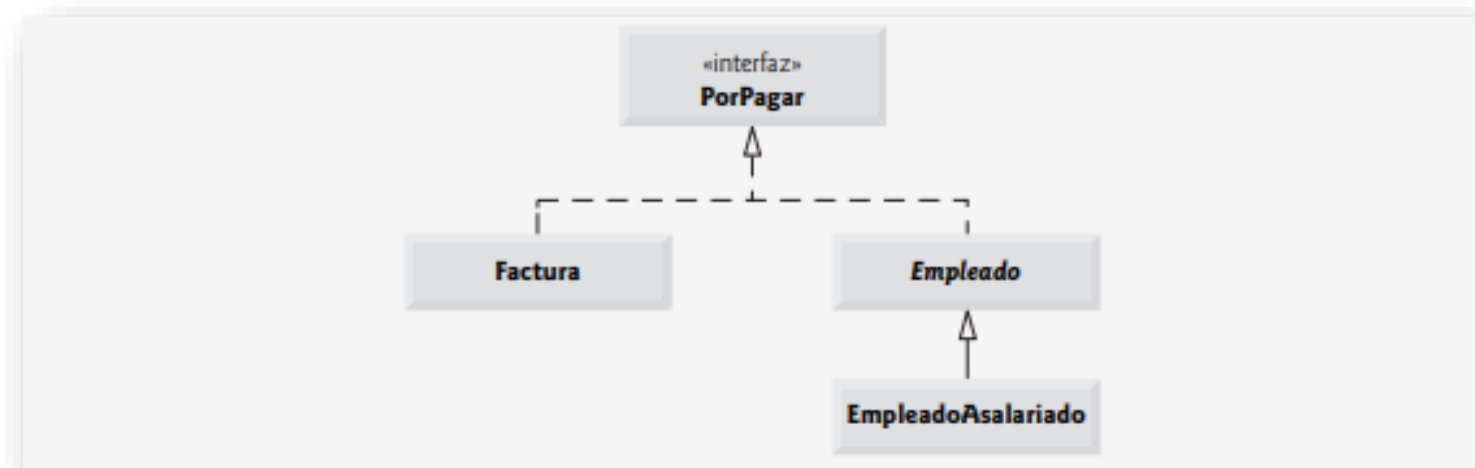
Es una colección de métodos abstractos y propiedades constantes y es en esta donde se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describen la lógica del comportamiento de los métodos.

```
public interface PorPagar {  
    public double obtenerMontoPago();  
}
```

```
public class Factura implements PorPagar {  
  
    @Override  
    public double obtenerMontoPago() {  
        return 0.0;  
    }  
}
```

## Interfaces

Modificar el ejercicio 1 para implementar la Interfaz PorPagar



# **Java Cómo Programar**

Deitel & Deitel

Capítulo 10

**Bibliografía**

---

# 10

---

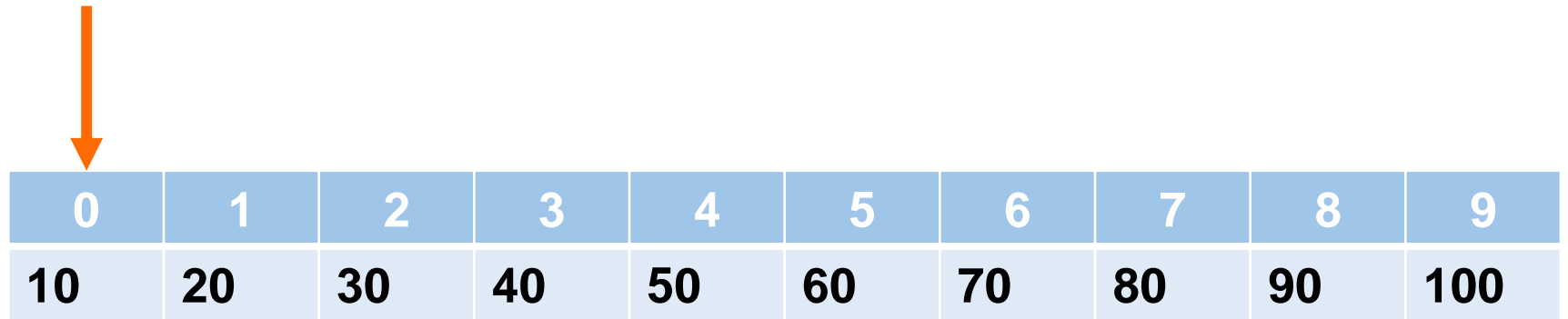
## Módulo 10

Arreglos



Un array es una estructura de datos que nos permite almacenar una ristra de datos de un mismo tipo. El tamaño de los arrays se declara en un primer momento y no puede cambiar en tiempo de ejecución

Índice



The diagram illustrates an array structure. It consists of a horizontal row of 10 cells. The top row of cells contains indices from 0 to 9, with an orange arrow pointing down to the cell containing '0'. The bottom row of cells contains corresponding values: 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. An orange arrow points up to the cell containing '20'.

0	1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90	100

Valor

Declaración por tamaño



```
//Declaración de un arreglo  
int arreglo1[] = new int[10];  
int arreglo2[] = {1, 2, 3, 4, 5};
```



Declaración por elementos

Lectura de una posición del arreglo



```
//Lectura de una posicion del arreglo  
int valor1 = arreglo1[5];  
//Asignacion de un valor a una posicion  
arreglo1[5] = 20;
```



Asignación de un valor a una posición

## Recorrer un arreglo

```
public class Arreglos {  
  
    public static void main(String[] args) {  
  
        //Declaración de un arreglo  
        String arreglo2[] = {"Este", "es", "un", "arreglo", "de", "String"};  
  
        for (int indice = 0; indice < arreglo2.length; indice++) {  
            System.out.println(String.format(  
                "Indice: %d, Valor: %s",  
                indice, arreglo2[indice]));  
        }  
    }  
}
```

## Recorrer un arreglo

```
public class Arreglos {  
  
    public static void main(String[] args) {  
  
        //Declaración de un arreglo  
        String arreglo2[] = {"Este", "es", "un", "arreglo", "de", "String"};  
  
        for(String elemento:arreglo2) {  
            System.out.println(String.format("Valor: %s", elemento));  
        }  
    }  
}
```

El ordenamiento de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con la siguiente posición o elemento, intercambiándolos de posición si están en el orden equivocado.

Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

0	1	2
50	30	20

1ª Pasada

0	1	2
30	20	50

2ª Pasada

0	1	2
20	30	50

3ª Pasada

0	1	2
20	30	50

**Ejercicio**

Algoritmo de burbuja

El método de burbuja es ineficiente para arreglos grandes. Haga las siguientes modificaciones para mejorar el rendimiento del algoritmo:

- a) Después de la 1ª Pasada, se garantiza que el número más grande está en el elemento del arreglo con el índice más grande, después de la 2ª Pasada los dos números más grandes, se encuentran “ordenados” etc. En vez de realizar  $n$  comparaciones en cada pasada, modifique el algoritmo para hacer  $n - 1$  comparaciones en cada pasada.
- b) Los datos en el arreglo tal vez ya se encuentren en el orden correcto, entonces, ¿para qué hacer  $n$  comparaciones si se pueden hacer con menos? Modifique el algoritmo para comprobar al final de cada pasada si el arreglo ya se encuentra ordenado.

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 7

**Bibliografía**

---



11

# Módulo 11

Colecciones

Una colección representa un grupo de objetos. Estos objetos son conocidos como elementos. Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos.

En Java, se emplea la interfaz genérica `Collection` para este propósito. Gracias a esta interfaz, podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes, como pueden ser: añadir, eliminar, obtener el tamaño de la colección.

La interfaz **Set** define una colección que no puede contener elementos duplicados. Esta interfaz contiene, únicamente, los métodos heredados de Collection añadiendo la restricción de que los elementos duplicados están prohibidos.

### Implementaciones:

1. HashSet
2. TreeSet
3. LinkedHashSet

La interfaz **Map** asocia claves a valores. Esta interfaz no puede contener claves duplicadas y; cada una de dichas claves, sólo puede tener asociado un valor como máximo.

### Implementaciones:

1. HashMap
2. TreeMap
3. LinkedHashMap

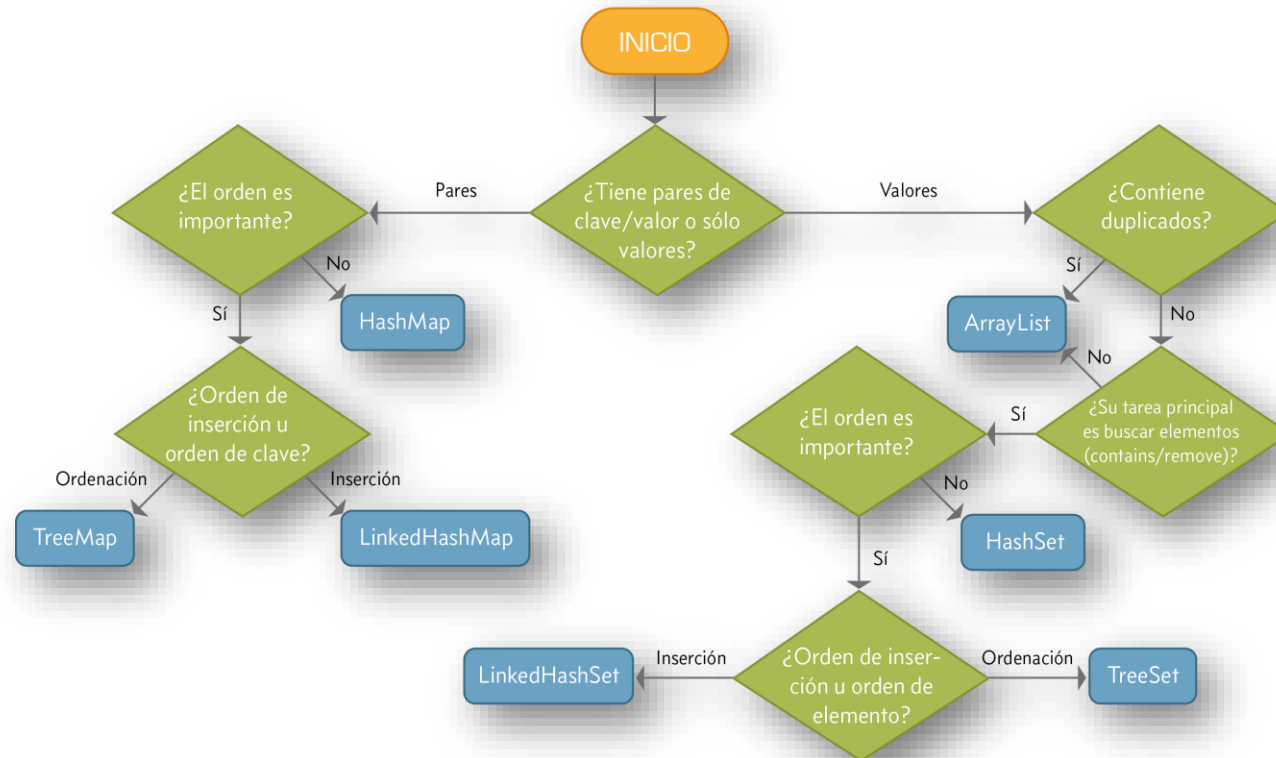
La interfaz **List** define una sucesión de elementos. A diferencia de la interfaz Set, la interfaz List sí admite elementos duplicados.

### Implementaciones:

1. ArrayList: Se basa en un array redimensionable que aumenta su tamaño según crece la colección de elementos. Es la que mejor rendimiento tiene sobre la mayoría de situaciones.
2. LinkedList: Esta implementación se basa en una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento.

# ¿Cómo elegir la mejor colección?

Diagrama de decisión para uso de colecciones Java



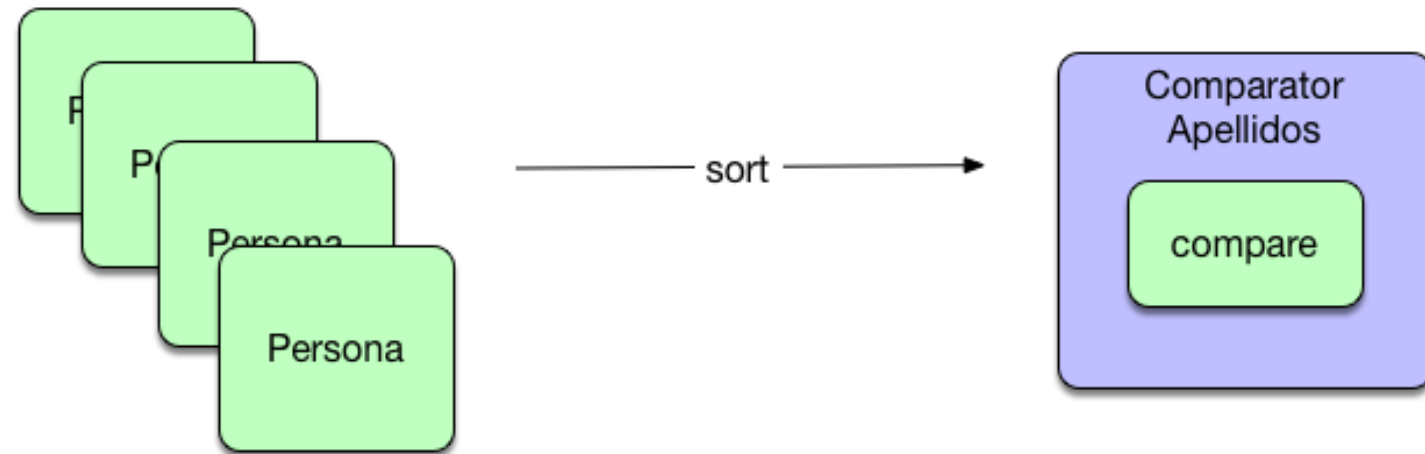
## List

### Principales métodos

<code>boolean add(E e)</code>	Se utiliza para agregar el elemento especificado al final de una lista.
<code>void clear()</code>	Se utiliza para eliminar todos los elementos de esta lista.
<code>E get(int index)</code>	Se utiliza para obtener el elemento de la posición particular de la lista.
<code>boolean isEmpty()</code>	Devuelve verdadero si la lista está vacía, de lo contrario, devuelve falso.
<code>E remove(int index)</code>	Se utiliza para eliminar el elemento presente en la posición especificada en la lista.
<code>E set(int index, E element)</code>	Se utiliza para reemplazar el elemento especificado en la lista, presente en la posición especificada.

Un Comparator es una Clase que el método compare de la interfaz Comparator para determinar el ordenamiento.

- Si el método compare devuelve un positivo, entonces  $\text{valor1} > \text{valor2}$ .
- Si el método compare devuelve un negativo, entonces  $\text{valor1} < \text{valor2}$ .





1. Crear una lista de objetos tipo String e iterar la lista.
2. Crear una lista de objetos de tipo Estudiante e iterar la lista
3. Convertir la lista de tipo Estudiante a String.
4. Cambiar y obtener elementos de la lista por medio de su indice
5. Ordenar la lista de elementos de tipo String
6. Iterar una lista mediante clave – valor
7. Ordenar la lista de objetos de tipo Estudiante de manera ascendente y descendente

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 16

**Bibliografía**

---

# 12

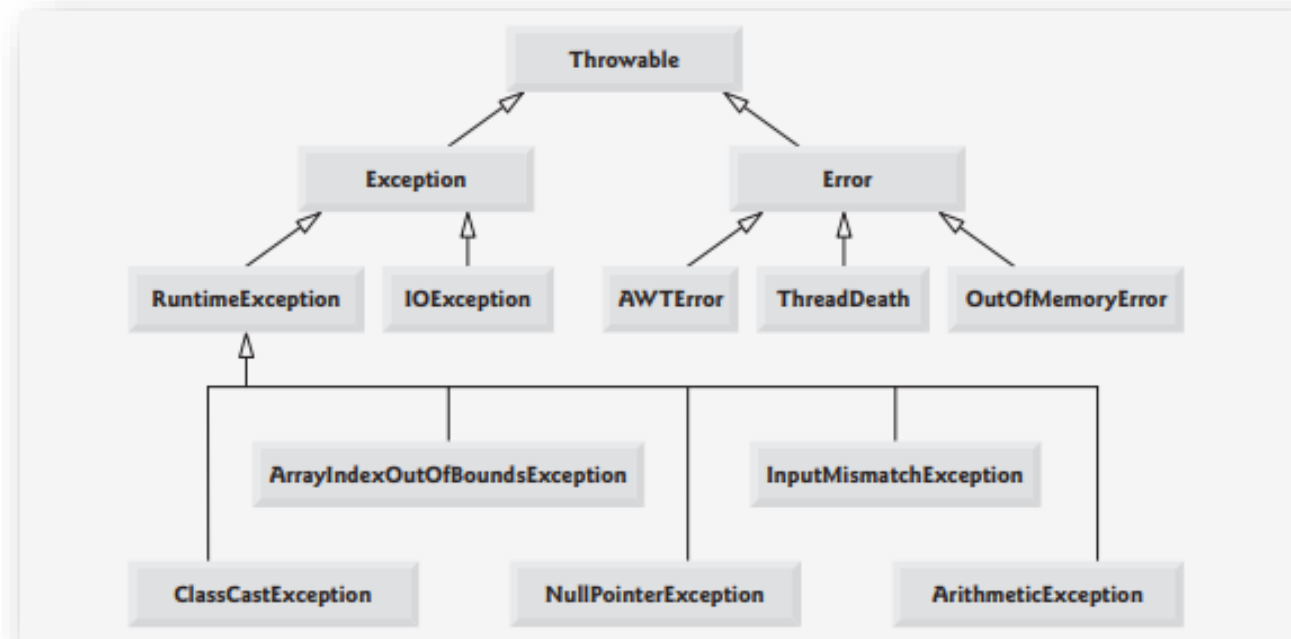
---

## Módulo 12

Excepciones

Una excepción es un error que ocurre en tiempo de ejecución.

En Java, todas las excepciones están representadas por clases. Todas las clases de excepción se derivan de una clase llamada Throwable. Por lo tanto, cuando se produce una excepción en un programa, se genera un objeto de algún tipo de clase de excepción.



1. Las excepciones de tipo Error están relacionadas con errores que ocurren en la Máquina Virtual de Java y no en el programa.

2. Los errores que resultan de la actividad del programa están representados por subclases de Exception. Por ejemplo:

- a) Dividir por cero
- b) Límite de matriz
- c) Errores de entrada/salida

## Manejo de excepciones

Bloque “try” donde se lanza la excepción

Bloques “catch” donde se atrapa una Excepcion (de lo mas general a lo más específico)

```
try{
    //bloque de código para monitorear errores
}
catch (TipoExcepcion1 exOb){
    //Manejador para TipoExepción1
}
catch (TipoExcepcion2 exOb){
    //Manejador para TipoExepción2
}
```

## La clase Throwable

Método	Sintaxis	Descripción
getMessage	String getMessage()	Devuelve una descripción de la excepción.
getLocalizedMessage	String getLocalizedMessage()	Devuelve una descripción localizada de la excepción.
toString	String toString()	Devuelve un objeto String que contiene una descripción completa de la excepción. Este método lo llama println() cuando se imprime un objeto Throwable.
printStackTrace()	void printStackTrace()	Muestra el flujo de error estándar.



## Atrapar excepciones desde un método

```
public class Excepciones {  
  
    public static void main(String[] args) {  
  
        int nums[] = new int[4];  
  
        try {  
            System.out.println("Antes de que se genere la excepción.");  
            //generar una excepción de índice fuera de límites  
            Excepciones.dividir();  
        } catch (Exception exc){  
            //Capturando la excepción  
            System.out.println(";Índice fuera de los límites!");  
        }  
        System.out.println("Despues de que se genere la excepcion.");  
    }  
  
    public static double dividir() {  
        return 10/0;  
    }  
}
```

Modificar el ejercicio anterior para que la excepción lanzada pase por una jerarquía de 3 métodos.

## Lanzar manualmente una excepción

```
public class Excepciones {  
  
    public static void main(String[] args) {  
  
        try{  
            System.out.println("Antes de lanzar excepción.");  
            throw new ArithmeticException(); //Lanzar una excepción  
        }catch (ArithmeticException exc){  
            //Capturando la excepción  
            System.out.println("Excepción capturada.");  
        }  
        System.out.println("Después del bloque try/catch");  
    }  
}
```

## Re-lanzar una excepción

Una excepción capturada por una declaración catch se puede volver a lanzar para que pueda ser capturada por un catch externo. La razón más probable para volver a lanzar de esta manera es permitir el acceso de múltiples manejadores/controladores a la excepción.

```
public class Excepciones {  
  
    public static void main(String[] args) {  
  
        try{  
            System.out.println("Antes de lanzar excepción.");  
            Lanza.lanzarExcepcion();  
        } catch (ArithmeticException exc){  
            //Capturando la excepción  
            System.out.println("Excepción capturada.");  
        }  
        System.out.println("Después del bloque try/catch");  
    }  
}  
  
class Lanza {  
  
    static void lanzarExcepcion() {  
        try {  
            int division = 10 / 0;  
        } catch (ArithmeticException e) {  
            System.out.println("Se atrapa la excepcion "  
                + "y se lanza nuevamente");  
            throw e;  
        }  
    }  
}
```

## Uso de finally

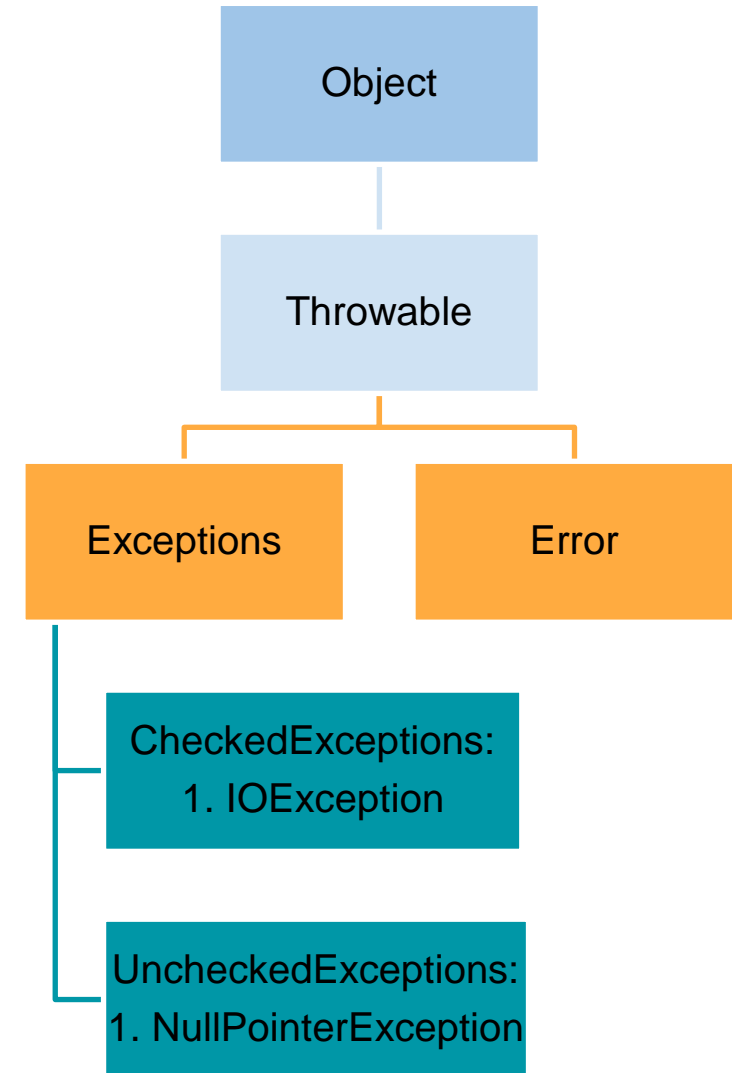
Algunas veces querrá definir un bloque de código que se ejecutará cuando quede un bloque try/catch. Por ejemplo, una excepción puede causar un error que finaliza el método actual, causando su devolución prematura. Sin embargo, ese método puede haber abierto un archivo o una conexión de red que debe cerrarse..

```
//Uso de finally
public class UsoFinally {
    public static void genExcepcion(int rec) {
        int nums[]=new int[2];
        System.out.println("Recibiendo " + rec);
        try {
            switch (rec){
                case 0:
                    int t= 10 / rec;
                    break;
                case 1:
                    nums[4] = 4; //Genera un error de indexación
                    break;
                case 2:
                    return; //Retorna desde el blorec try
            }
        } catch (ArithmeticException exc){
            //Capturando la excepción
            System.out.println("No se puede dividir por cero!");
            return; //retorna desde catch
        } catch (ArrayIndexOutOfBoundsException exc){
            //Capturando la excepción
            System.out.println("Elemento no encontrado");
        } finally {
            //esto se ejecuta al salir de los blores try/catch
            System.out.println("Saliendo de try.");
        }
    }
}
```

## Tipos de excepciones

**Checked Exceptions:** Estas son las excepciones que se comprueban en tiempo de compilación.

**Unchecked Exceptions:** Estas son las excepciones que no se verifican en tiempo de compilación



## Tipos de excepciones

```
public class LeerArchivo {  
    public static void main(String[] args) {  
        leerArchivo();  
    }  
    public static leerArchivo() {  
  
        // Crear una referencia al archivo  
        FileReader file = new FileReader("C:\\\\test\\\\a.txt");  
  
        // Crear un Buffer para leer el archivo  
        BufferedReader fileInput = new BufferedReader(file);  
  
        //Leer el archivo e imprimir las primeras 3 lineas  
        for (int counter = 0; counter < 3; counter++)  
            System.out.println(fileInput.readLine());  
  
        //Cerrar el archivo  
        fileInput.close();  
    }  
}
```

# **Java Cómo Programar**

Deitel & Deitel

Capítulo 11

**Bibliografía**

---



**¡Gracias!**

