



CONCEPTOS BÁSICOS

Agenda

- Modificadores
- Clases y Métodos

Abstractos

- Interfaces
- JavaDoc

Modificadores

- De acceso
 - Public
 - Private
 - Protected
- Static
- Final

Modificadores de acceso – “Public”

- Un objeto, atributo o método **Público** puede ser accedido por cualquier otro objeto o método fuerá de su clase. Su acceso no se limita a su scope.

```
3 public class Robot {  
4  
5     public String name; // Atributo público  
6     public String type;  
7     public int year;  
8  
9     public Robot(String name, String type, int year) {  
10        this.name = name;  
11        this.type = type;  
12        this.year = year;  
13    }  
14  
15    public String getName() { // Método público  
16        return name;  
17    }  
18}
```

```
public class Main {  
    public static void main(String[] args) {  
        Robot robot = new Robot( name: "Tom", type: "metal", year: 2017);  
        //Quiero acceder al nombre del robot  
        String nombre1 = robot.name;  
        String nombre2 = robot.getName();  
    }  
}
```

Modificadores de acceso – “Private”

- Un objeto, atributo o método **Privado NO** puede ser accedido por cualquier otro objeto o método fueras de su clase. Su acceso no se limita a su scope.

```
public class Robot {  
  
    private String name;  
    private String type;  
    private int year;  
  
    public Robot(String name, String type, int year) {  
        this.name = name;  
        this.type = type;  
        this.year = year;  
    }  
  
    public String getName() {  
        return name;  
    }  
    public String getType() {  
        return type;  
    }  
    public int getYear() {  
        return year;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Robot robot = new Robot( name: "Tom", type: "metal", year: 2017);  
        //Quiero acceder al tipo del robot  
        String nombre1 = robot.type;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Robot robot = new Robot( name: "Tom", type: "metal", year: 2017);  
        //Quiero acceder al tipo del robot  
        String nombre1 = robot.getType();  
    }  
}
```

Modificadores de acceso – “Protected”

- Un objeto, atributo o método **Protegido NO** puede ser accedido por cualquier otro objeto o método fueras de su clase, EXCEPTO por sus subclases.

```
public class Robot {  
  
    protected String name;  
    private String type;  
    private int year;  
  
    public Robot(String name, String type, int y  
        this.name = name;  
        this.type = type;  
        this.year = year;  
    }  
}
```

```
public class Drone extends Robot {  
  
    public Drone(String name, String type, int year) {  
        super(name, type, year);  
        this.name = "drone - " + name;  
        this.type = "drone - " + type;  
    }  
}
```

Modificadores- “Static”

- Un modificador **Estático**, hace que un atributo o método, pertenezca a la CLASE y no a la INSTANCIA de la clase.
- Es posible su uso SIN tener una instancia de la clase.

```
public class Main {  
    public static void main(String[] args) {  
        int a = 10, b = 5;  
        Calculadora calculadora = new Calculadora();  
        int suma = calculadora.sumaEntera(a, b);  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        int a = 10, b = 5;  
        int suma = Calculadora.sumaEntera(a, b);  
    }  
}
```

```
public class Calculadora {  
  
    public Calculadora() { }  
  
    public int sumaEntera(int a, int b) {  
        return a+b;  
    }  
}
```

```
public class Calculadora {  
  
    public Calculadora() { }  
  
    public static int sumaEntera(int a, int b) {  
        return a+b;  
    }  
}
```

Modificadores- “Final”

- Un modificador **Final**, es inicializado con un valor que NO puede cambiar en tiempo de ejecución.

```
public class Main {  
    public static void main(String[] args) {  
        Calculadora.numero = 9;  
        Calculadora.PI = 34;  
    }  
}  
  
public class Calculadora {  
    public static final double PI = 3.1415;  
    public static final double E = 2.7182;  
    public static int numero = 10;  
  
    public Calculadora() { }  
}
```

Clases y Métodos Abstractos

- Una **clase abstracta** tiene su razón de ser en la herencia. Una superclase abstracta NO tendrá instancias, sino que sus subclases tendrán instancias, y deberán implementar todos los métodos abstractos de la superclase

```
public abstract class Robot {  
  
    protected String name;  
    private String type;  
    private int year;  
  
    public abstract int getSerialNumber();  
  
    public Robot(String name, String type, int year) {  
        this.name = name;  
        this.type = type;  
        this.year = year;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

Puede
implementa
r métodos

```
public class Drone extends Robot {  
  
    public Drone(String name, String type, int year) {  
        super(name, type, year);  
        this.name = "drone - " + name;  
    }  
  
    @Override  
    public int getSerialNumber() {  
        return 33777;  
    }  
}
```

Interfaces

- Una **Interface NO** posee implementaciones de métodos (*). Solo posee **DEFINICIONES** que obligatoriamente tienen que implementar las clases que implementan esa interfaz.
- Son utilizadas para unificar clases.
- Es una colección de métodos abstractos y propiedades constantes (*)
(*) En java 9 cambia.

```
public interface IRobot {  
  
    void mover();  
    String getType();  
    int getPosX();  
    int getPosY();  
}
```

```
public class Drone implements IRobot {  
  
    private int x;  
    private int y;  
  
    public Drone(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    @Override  
    public void mover() {  
        x += 5;  
        y += 2;  
    }  
  
    @Override  
    public int getPosX() {  
        return x;  
    }  
  
    @Override  
    public int getPosY() {  
        return y;  
    }  
}
```

JavaDoc

- JavaDoc posee toda la documentación de las librerías y objetos que posee Java. Explica detalladamente cada objeto, con sus atributos, métodos, formas de uso y ejemplos.
- <https://docs.oracle.com/javase/8/docs/api/>

The screenshot shows a Java API documentation page for the `ThreadLocalRandom` class. The top navigation bar includes links for Overview, Package, Class (which is highlighted), Use, Tree, Deprecated, Index, and Help. Below the navigation, there are links for Prev Class, Next Class, Frames, and No Frames. The main content area starts with a summary of nested classes, fields, constructors, and methods. It then defines the `ThreadLocalRandom` class, which extends `java.lang.Object` and implements `java.util.Random`. The class is described as a random number generator isolated to the current thread. The sidebar on the left lists various Java packages and classes, such as `java.awt`, `java.util.concurrent`, and `java.util.ThreadLocal`.

Java™ Platform Standard Ed. 8

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.util.concurrent

Class ThreadLocalRandom

java.lang.Object
java.util.Random
java.util.concurrent.ThreadLocalRandom

All Implemented Interfaces:

Serializable

public class ThreadLocalRandom
extends Random

A random number generator isolated to the current thread. Like the global `Random` generator used by the `Math` class, a

Java™ Platform Standard Ed. 8

JDK Versiones

Versión	Fecha de Lanzamiento	Principales Funcionalidades
JDK 8 (LTS)	Marzo 2014	Lambda expressions - Streams API - Default Methods Optional Class - Date and Time API
JDK 9 - JDK 10	2017 - 2018	JShell - Modules System - Reactive Streams - Local Variable Type Interface Parallel Full GC for G1- Application CDS (Class-Data Sharing)
JDK 11 (LTS)	Septiembre 2018	HTTP Client - Local-Variable for Lambda Parameters Epsilon GC - Single-File Source-Code Programs
JDK 12 - JDK 16	2019 - 2021	Switch Expressions - Sealed Classes - Records - Text Blocks Pattern Matching for instanceof - Helpful NullPointerExceptions
JDK 17 (LTS)	Septiembre 2021	Sealed Classes (Standard) - Pattern Matching for switch Strong Encapsulation for JDK Internals - New macOS Rendering Pipeline
JDK 18 - JDK 20	2022 - 2023	Simple Web Server - Foreign Functions & Memory API Structured Concurrency - Vector API
JDK 21 (LTS)	Septiembre 2023	String Template (Preview) - Virtual Threads - Structured Concurrency Unnamed Patterns (Preview) - Scoped Values - Sequenced Collections

¿PREGUNTA
S?

