
Java™

Contents

La Llave del Vivero 3

I. Prólogo 4

II. La Regla de Oro de la Sintaxis 5

V. Ejercicio 0: El Mapa de Visibilidad 6

VI. Ejercicio 1: La Puerta Abierta 6

VII. Ejercicio 2: El Candado 7

VIII. Ejercicio 3: El Vacío 7

IX. Ejercicio 4: Estático vs Instancia 8

XIII. Ejecución 9

La Llave del Vivero

Antes de plantar árboles complejos, debemos aprender quién tiene permiso para entrar al vivero. Aprenderás modificadores de acceso, tipos de retorno y la anatomía de una clase. En Java, cada elemento tiene una identidad definida por tres preguntas: ¿Quién puede verlo? (Modificador), ¿Qué es? (Tipo) y ¿Cómo se llama? (Nombre).

Instrucciones Generales

- ✓ ***Versión:*** Java 17+ (OpenJDK).
- ✓ ***Estructura:*** Una clase por archivo (Puedes usar package).
- ✓ ***Linter Obligatorio:*** Checkstyle (Google Java Style). Errores de estilo = KO.

- ✓ ***Comentarios:*** Obligatorios, solo en inglés y siguiendo formato Javadoc.
- ✓ **Nomenclatura:** Clases en PascalCase, métodos y variables en camelCase.
- ✓ ***IDE:*** IntelliJ IDEA (recomendado), VS Code, Eclipse, etc.
- ✓ ***Compilación:*** Vía terminal: javac Clase.java y java Clase.

Entrega y Evaluación

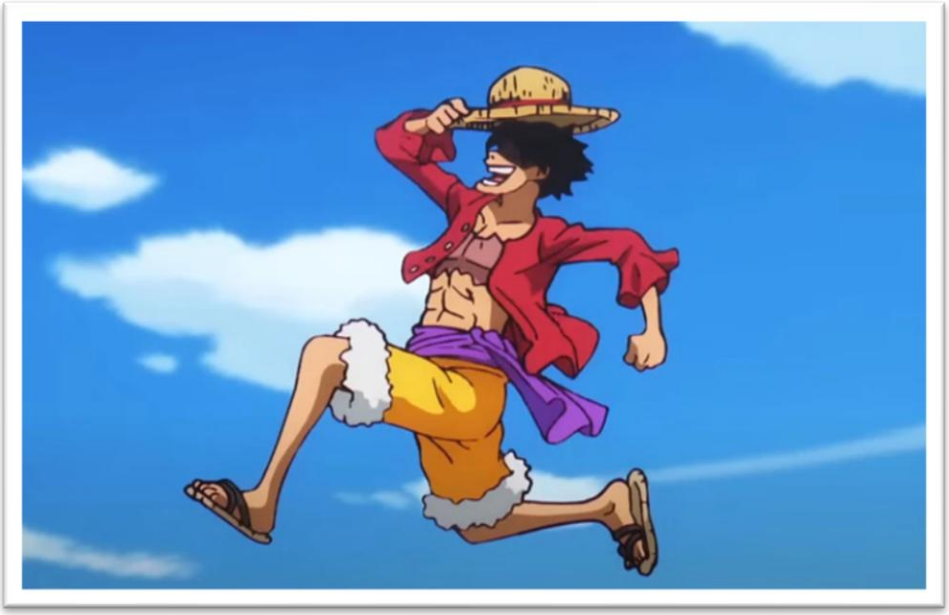
- Entrega en repositorio Git.

I. Prólogo

En el mundo de la programación, así como en un huerto comunitario, existen reglas invisibles que dictan quién puede acceder a qué recurso. No cualquiera puede entrar a una zona de semillas delicadas, ni todos los voluntarios tienen permiso para modificar el sistema de riego principal. En Java, estas reglas se definen mediante la ***Sintaxis de Acceso***.

Entender la anatomía de una clase no es solo una cuestión de gramática técnica; es aprender a diseñar un ecosistema donde la información está protegida y las acciones están bien definidas. Un código bien estructurado es como un jardín con vallas y senderos claros: evita que los datos se corrompan accidentalmente y permite que diferentes componentes trabajen juntos en armonía.

A medida que avances en este módulo introductorio, descubrirás que cada palabra clave (public, private, static) es una herramienta de organización. Aprenderás que programar con precisión es como preparar el suelo: una base sólida y bien delimitada es lo que permite que las aplicaciones futuras crezcan de forma robusta y escalable.



II. La Regla de Oro de la Sintaxis

*¡Te damos la bienvenida a **La Anatomía del Código**!*

En Java, la declaración de cualquier cosa sigue casi siempre este orden:

[Visibilidad] [Modificador_Extra] [Tipo_de_Dato/Retorno] [Nombre]

1. Clases

- **Sintaxis:** public class MyClass { ... }
- **Regla:** El nombre de la clase debe ser igual al del archivo.

2. Variables (Atributos)

- **Sintaxis:** private int treeHeight;
- **Regla:** Define primero la visibilidad para proteger el dato.

3. Métodos (Funciones)

- **Sintaxis:** public void grow() { ... }
- **Regla:** Si no devuelve nada, usa void. Si devuelve algo, pon el tipo (ej. int).

Los Modificadores de Acceso

- **public**: Todo el mundo puede verlo (La puerta del vivero está abierta).
- **private**: Solo se ve dentro de esa clase (Es tu diario personal).
- **protected**: Solo lo ven los hijos (herencia) y los vecinos del mismo paquete.

Default (sin nada): Solo lo ven los vecinos del mismo paquete (el "barrio").

Si no pones nada (solo class Nombre), Java aplica el acceso **Package-Private**. Esto significa que la clase solo es visible para otras clases que estén dentro de su misma carpeta (paquete). El resto del proyecto no podrá acceder a ella ni utilizarla.

En resumen: Es como si fuera un "miembro del barrio"; solo tus vecinos inmediatos saben que existes.

V. Ejercicio 0: El Mapa de Visibilidad

- **Directorio:** ex0/
- **Archivo:** SyntaxMap.java
- **Objetivo:** Identificar y escribir correctamente los componentes.
- **Funciones Autorizadas:** System.out.print/ln

Escribe una clase donde declares:

1. Una constante **pública** y **estática** (entera) llamada MAX_TREES con valor 100.
2. Una variable **privada** (String) llamada speciesName.
3. Un método **público** que devuelva el nombre de la especie.

```
Maximum trees allowed: 100
Current species: Oak
```

VI. Ejercicio 1: La Puerta Abierta

- **Directorio:** ex1/
- **Archivo:** PublicGate.java
- **Objetivo:** Entender que public permite el acceso total.
- **Funciones Autorizadas:** System.out.print/ln

Crea una clase llamada PublicGate.

1. Declara una variable **pública** de tipo String llamada message.
2. En tu método main, crea un objeto de esa clase y cambia el mensaje directamente.

Teoría rápida: Usa public solo para clases y métodos que necesiten ser llamados desde fuera. **Casi nunca** para variables.

```
The gate is open: Welcome!
```

VII. Ejercicio 2: El Candado

- **Directorio:** ex2/
- **Archivo:** PrivateSeed.java
- **Objetivo: Entender que private oculta la información.**
- **Funciones Autorizadas:** System.out.print/ln

Crea una clase PrivateSeed.

1. Declara una variable **privada** llamada seedName.
2. Intenta cambiarla directamente desde otra clase. ¡Verás que el compilador falla!
3. Crea un método **público** llamado setSeedName para poder cambiarla de forma segura.

Teoría rápida: Esto se llama **Encapsulamiento**. Protegemos el dato (private) y damos una herramienta para usarlo (public method).

```
Accessing private seed: Oak
```

VIII. Ejercicio 3: El Vacío

- **Directorio:** ex3/
- **Archivo:** WaterAction.java
- **Objetivo:** Entender cuándo usar void y cuándo no.
- **Funciones Autorizadas:** System.out.print/ln

En la clase WaterAction, crea dos métodos:

1. ***public void pourWater():*** Solo imprime "Pouring water...". No devuelve nada (void).
2. ***public int getWaterAmount():*** No imprime nada, solo **devuelve** el número 10.

Teoría rápida:
void es una acción (haz algo). Si no tiene void, es una pregunta (dame algo).

```
Action: Pouring water...
Data: Water amount is 10 liters.
```

IX. Ejercicio 4: Estático vs Instancia

- **Directorio:** ex4/
- **Archivo:** ForestCounter.java
- **Objetivo:** ¿Por qué a veces usamos static?
- **Funciones Autorizadas:** System.out.print/ln

Crea una clase ForestCounter.

1. Crea una variable **estática** totalTrees.
2. Crea una variable **no estática** treeName.
3. Observa que totalTrees es la misma para todos los árboles, pero cada uno tiene su propio treeName.

```
Tree 1: Pine
Tree 2: Oak
Total trees in forest: 2
```

XIII. Ejecución

El flujo real en tu terminal

Entra en la carpeta:

Primero tienes que estar donde está el archivo. Si no, el comando javac te dirá "***archivo no encontrado***" y te entrarán ganas de tirar el PC por la ventana.

- ✓ cd Desktop/proyectos_java/ex00
- ✓ ***javac*** FtHelloForest.java **-> La Compilación (El traductor)**
- ✓ ***java*** FtHelloForest.java **-> La Ejecución (El lanzamiento)**