
Java™

Contents

Raíces del Código

Java es un lenguaje de programación **orientado a objetos (POO)** que se usa para crear desde aplicaciones de escritorio, móviles (como aplicaciones Android) hasta servidores y sistemas empresariales. Su filosofía principal es "**escribe una vez, corre en cualquier lugar**" porque funciona sobre la **Java Virtual Machine (JVM)**, que traduce el código Java a un lenguaje que cualquier sistema operativo puede ejecutar.

Fundamentos de Java a través de la gestión de viveros forestales

Construye un ecosistema digital de gestión forestal mientras descubres conceptos avanzados de Java como Clases, Herencia, Encapsulamiento y Polimorfismo

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 la era digital, la gestión forestal requiere sistemas inteligentes que reflejen la complejidad de la naturaleza. Como ocurre en un bosque próspero, el código bien estructurado crece capa a capa, donde cada componente tiene un propósito y lugar adecuado.

Las plantas no crecen aisladas; forman comunidades interconectadas donde cada especie aporta algo único mientras comparte necesidades comunes. Este ecosistema refleja el funcionamiento del software basado en objetos: componentes individuales con roles distintos que trabajan juntos en armonía para construir aplicaciones robustas. Cada línea de código que escribas en este módulo es una semilla plantada en el suelo de las posibilidades digitales.

II. Introducción

*¡Te damos la bienvenida a **Raíces del Código 01**!*

¡Te damos la bienvenida a la segunda etapa de tu formación! Tras dominar los fundamentos básicos, ahora abordarás desafíos más complejos mediante la **Programación Orientada a Objetos (POO)**.

La POO es un paradigma que nos permite modelar sistemas del mundo real —como nuestro vivero forestal— creando "objetos" digitales. En lugar de manejar datos sueltos, agrupamos la información y el comportamiento dentro de una misma entidad.

- **Clases (El Molde):** Imagina que tienes un plano botánico que define qué es un "Árbol". Este plano especifica que todo árbol tiene una altura y una especie.
- **Objetos (El Ejemplar):** A partir de ese plano, puedes plantar ejemplares reales: un Pino de 50cm o un Roble de 200cm. Cada uno es un "objeto" independiente creado desde la misma clase.
- **Uso y Ventajas:** Este enfoque te permitirá crear componentes de código reutilizables, proteger la integridad de tus datos forestales y diseñar arquitecturas de software que puedan adaptarse y ampliarse fácilmente a medida que tu "bosque digital" crece.

Este módulo progresa desde la estructura básica hasta la creación de sistemas complejos, donde cada ejercicio añade nuevas funcionalidades a tu ecosistema de gestión.

V. Ejercicio 0: El Punto de Entrada

- **Directorio:** ex0/
- **Archivo:** FtForestIntro.java
- **Método:** Any.
- **Funciones Autorizadas:** System.out.print/ln, public static void main(String[] args).

Crea un programa que represente la "semilla" de tu sistema. Debe utilizar el punto de entrada estándar de Java para mostrar información básica de un árbol inicial. Si no usa parámetros la salida será: **"*Usage: java FtForestIntro <Type> <Height> <Age>*"**
Requisito extra: Investiga qué significa el parámetro String[] args en el método main.

```
=== Welcome to the Forest Management System ===
Tree: Pine
Height: 50cm
Age: 100 days

=== End of Program ===
```

VI. Ejercicio 1: El Modelo del Árbol

- **Directorio:** ex1/
- **Archivo:** Tree.java, FtForestData.java
- **Método:** Any.
- **Funciones Autorizadas:** Constructores, System.out.print/ln, public static void main(String[] args), class, this, toString(), @Override.

El sistema necesita registrar múltiples árboles. Crea una clase ***Tree*** con atributos: nombre, altura y edad. Crea una instancia para un Pine, un Oak y un Cedar.

Requisito extra

- ✓ ¿Qué sucede si imprimes el objeto directamente (System.out?println(myTree))?
- ✓ Investiga el método toString() y this.

```
=== Forest Tree Registry ===
Pine: 50cm, 100 days old
Oak: 200cm, 365 days old
Cedar: 150cm, 500 days old
```

VII. Ejercicio 2: Comportamiento Forestal

- **Directorio:** ex2/
- **Archivo:** Tree.java, FtForestGrowth.java
- **Método:** Any
- **Funciones Autorizadas:** Constructores, System.out.print/ln, public static void main(String[] args), class, this, toString(), @Override.

Las plantas deben tener comportamientos propios.

- ✓ **Requisitos:** Implementa los métodos grow() (aumenta altura) y age() (aumenta edad).
- ✓ **Simulación:** Tu programa debe mostrar el estado inicial y el estado tras una "semana de crecimiento".

```
=== Day 1 ===
Pine: 50cm, 100 days old
=== Day 7 ===
Pine: 56cm, 106 days old
Growth this week: +6cm
```

VIII. Ejercicio 3: Constructor de Vivero

- **Directorio:** ex3/
- **Archivo:** Tree.java, FtTreeFactory.java
- **Método:** Any
- **Funciones Autorizadas:** Constructores, System.out.print/ln, public static void main(String[] args), class, this, toString(), @Override, throw new IllegalArgumentException, trim(), isEmpty(), null.

Optimiza la creación de árboles para que nazcan con sus valores iniciales listos.

- ✓ **Requisito:** No permitas que un árbol se cree "vacío"; el constructor debe exigir los datos obligatorios.
- ✓ **Extra-Teórico:** Investiga la **sobrecarga de constructores** (Constructor Overloading). ¿Podrías crear un árbol solo con el nombre y que el resto sea por defecto?

```
=== Tree Factory Output ===
Created: Pine (50cm, 100 days)
Created: Oak (200cm, 365 days)
Created: Cedar (150cm, 500 days)
Created: Fir (30cm, 45 days)
Created: Willow (120cm, 200 days)
Total trees created: 5
```

IX. Ejercicio 4: Seguridad y Encapsulamiento

- **Directorio:** ex4/
- **Archivo:** SecureTree.java, FtForestSecurity.java
- **Método:** Any
- **Funciones Autorizadas:** private, public, Getters/Setters, todas las funciones de los ejercicios anteriores, no posteriores.

Evita que se corrompan los datos con valores imposibles (alturas o edades negativas).

- ✓ **Requisitos:** Usa el modificador private para los atributos.
- ✓ **Validación:** Los métodos setHeight y setAge deben rechazar valores negativos con un mensaje de error.

```
=== Forest Security System ===
Tree created: Pine
Height updated: 55cm [OK]
Age updated: 101 days [OK]
Invalid operation attempted: height -10cm [REJECTED]
Security: Negative height rejected
Current tree: Pine (55cm, 101days)
```

X. Ejercicio 5: Especialización de Especies

- **Directorio:** ex5/
- **Archivo:** Tree.java, Conifer.java, FruitTree.java, FtTreeSpecies.java
- **Funciones Autorizadas:** extends, super(), System.out.println, todas las funciones de los ejercicios anteriores, no posteriores.

Crea tipos especializados que hereden de la clase base Tree:

- ✓ **Conifer:** Añade el atributo stneedleType y el método dropCones() *(Solo muestra).*
- ✓ **FruitTree:** Añade el atributo fruitType y el método harvest() *(Solo muestra)*.
- ✓ **Usar:** Encapsulamiento (Obligatorio).
- ✓ **Extra-Teórico:** Entender la relación **"Es un"** (Is-a).

```
=== Forest Tree Types ===
Pine (Conifer): 50cm, 100 days, Needle type: Long
Pine is dropping cones on the forest floor!

Apple Tree (FruitTree): 180cm, 400 days, Fruit: Red Apple
Harvesting Red Apple... Nutritional value: High
```

XI. Ejercicio 6: Plataforma Analítica

- **Directorio:** ex06/
- **Archivo:** ForestAnalytics.java, ForestManager.java, Tree.java, FruitTree.java, PrizeAppleTree.java.
- **Método:** Any
- **Funciones Autorizadas:** static, final, ArrayList (opcional), Clases anidadas, todas las funciones anteriores.

Este ejercicio reúne todo lo aprendido.

- ✓ **Requisitos:** Crea un ForestManager que gestione una colección de árboles.
- ✓ **Métodos de Clase:** Implementa un método static para contar cuántos árboles totales se han creado en el sistema.
- ✓ **Jerarquía Completa:** Crea una cadena: Tree -> FruitTree -> PrizeAppleTree.
- ✓ **Usar:** Encapsulamiento (Obligatorio).

- ✓ Tener en cuenta:
  - ***Tree:*** name, height, age
  - ***Fruit:*** Tree atributos + *typeFruit*, *harvestStatus*, description.
  - ***PrizeAppleTree:*** Fruit atributos + quality, color, state.

```
=== Forest Management System Demo ===
Added Pine to Northern Forest
Added Apple Tree to Northern Forest
Northern Forest is nurturing all trees...
Pine grew 2cm
Apple Tree grew 2cm

=== Northern Forest Report ===
Trees in forest:
- Pine: 52cm, Long needles
- Apple Tree: 182cm, Red Apple (Harvest ready)
Trees added: 2, Total growth: 4cm
Validation test (Positive height): True
Total forests managed: 1

Trees in forest:
- Pine: 52cm, Long needles
- Apple Tree: 182cm, Red Apple (Harvest ready)
Trees added: 2, Total growth: 4cm
Validation test (Positive height): True
Total forests managed: 1
```

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)**