
Java™

Contents

Ingeniería de Datos para Biosferas Planetarias

En el espacio, un error de datos no solo es un bug, es una catástrofe. Aprenderás a construir tuberías de datos (pipelines) resilientes para gestionar estaciones biológicas en Marte y Europa, manejando fallos de sensores y tormentas solares sin que el sistema colapse.

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.
- ✓ **Regla de Supervivencia:** Tus programas ***nunca deben cerrarse inesperadamente (crash)***. Debes capturar los errores y recuperarte.

Entrega y Evaluación

- Entrega en repositorio Git.

I. Prólogo

El Centinela del Vacío

En la inmensidad del espacio, los datos son el oxígeno de la supervivencia. Building en tus bases de monitorización (Módulo 01), ahora te enfrentas a los desafíos reales de la agricultura inteligente en entornos hostiles. En una estación orbital o en una base en Marte, los datos fluyen como el agua a través de sistemas de irrigación: lecturas de sensores constantes, previsiones de radiación solar y telemetría de humedad en invernaderos digitales.

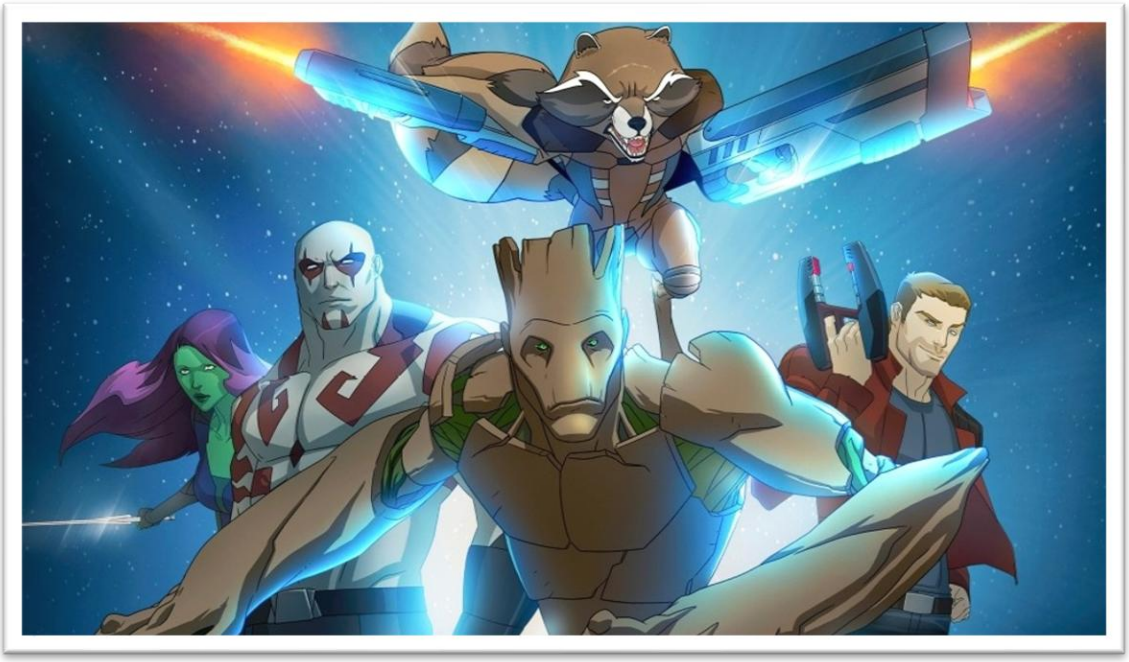
Pero ¿qué sucede cuando los sensores fallan en plena cosecha? ¿Qué ocurre cuando las conexiones se pierden durante una tormenta solar crítica o cuando datos corruptos amenazan con activar ciclos de riego falsos que agotarían nuestras reservas? Los ingenieros de datos espaciales profesionales saben que los sistemas robustos no se construyen para evitar fallos, sino para manejarlos con elegancia cuando ocurre lo inesperado.

Tu invernadero digital debe ser tan resiliente como la propia naturaleza. El sistema de manejo de excepciones de Java es tu caja de herramientas para construir tuberías de datos a prueba de balas. Aprenderás a capturar anomalías, crear alertas personalizadas y asegurar la integridad de la misión incluso cuando las leyes de Murphy ataquen en el vacío. En este proyecto, evolucionarás de un programador básico a un ingeniero de datos capaz de mantener la vida creciendo sin importar lo que pase.

II. Introducción

¡Te damos la bienvenida a **Garden Guardian: ¡Ingeniería de Datos Espaciales!**

Partiendo de los cimientos de monitorización que construiste anteriormente, ahora dominarás las habilidades críticas para la ingeniería de tuberías de datos resilientes en sistemas de soporte vital.



A lo largo de este módulo, descubrirás:

- Cómo validar y limpiar flujos de datos agrícolas en tiempo real.
- Qué tipos de modos de fallo existen en las redes de sensores interestelares.
- Cómo crear alertas personalizadas para cultivos específicos en condiciones de baja gravedad.
- Técnicas esenciales para la tolerancia a fallos y la recuperación de sistemas críticos.
- Cómo garantizar la integridad de los datos en sistemas de cultivo distribuidos por toda la galaxia.

Cada ejercicio construye un componente de tu plataforma de datos de agricultura espacial, progresando desde la validación básica de sensores hasta sistemas de monitorización integrales y automáticos. Tus programas deben demostrar cómo construir sistemas que manejen los escenarios del mundo real con gracia y precisión.

V. Ejercicio 0: El Punto de Entrada

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

Objetivo: Filtrar lecturas corruptas de los sensores de la base.

- Crea un método checkOxygenLevel(String input) que intente convertir la lectura a un número.
- **Rango seguro:** 18% a 25%. Si está fuera, lanza una excepción.

Salida esperada

```
Testing Oxygen: 21 -> Perfect for breathing!
Testing Oxygen: 'error' -> Error: Invalid sensor data format
Testing Oxygen: 10 -> Error: 10% is too low! Evacuate!
```

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:** ex6/
- **Archivo:** ForestAnalytics.java, ForestManager.java, Tree.java, FruitTree.java, PrizeAppleTree.java.
- **Imports:**
  import java.util.ArrayList;
  import java.util.List;
- **Funciones Autorizadas:** static, final, ArrayList (opcional), Clases anidadas, private final List<Tree> tres, 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
Total growth: 4cm

=== Northern Forest Report ===
Trees in forest:
- Pine (52cm, 100days)
- Apple Tree (182cm, 200days) - Premium fruit tree, Apple,
Harvest ready (A+, Red, Ready)
Trees added: 2
Total forests managed: 1
Total trees created in system: 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)**