

ACTIVIDAD 1.7 - Tratamiento de ficheros Formación. Uso de fichero “properties”

El objetivo de la actividad es el de utilizar un fichero de configuración (”.properties”) para almacenar los literales que están fijos en el código.

Así mismo, en la actividad se realizará una refactorización del código para adaptarlo al uso de “ficheros properties”.

▼

Investigación sobre el uso de los ficheros “properties” de configuración

4 formas de gestionar constantes en un proyecto Java “puro”

- **Clases de Constantes (`public static final`)**

  Es el estándar nativo del lenguaje Java para valores inmutables en tiempo de compilación. Su gran desventaja es que acopla fuertemente la configuración al código fuente.

- **Variables de Entorno (`System.getenv()`)**La metodología **"The Twelve-Factor App"** (el estándar de la industria más reconocido para crear aplicaciones modernas y escalables) establece en su **Factor III (Configuración)** que todo lo que varía entre despliegues debe almacenarse estrictamente en variables de entorno.

- **Propiedades del Sistema de la JVM (`System.getProperty()`)**

  Se inyectan en la Máquina Virtual de Java desde la línea de comandos usando el flag `-D` al momento de la ejecución. La propia documentación de Oracle lista decenas de propiedades

globales del sistema que se cargan mediante este mecanismo.

- **Archivos `.properties` (`java.util.Properties`)**

  Utilizan la clase `java.util.Properties`, presente en el JDK desde su versión 1.0. Esta clase proporciona los métodos oficiales `load()` y `store()` para leer y guardar colecciones persistentes de propiedades.

Ventajas apoyadas por la arquitectura del software

El uso de un archivo `.properties` o de variables de entorno obedece a un principio fundamental de la ingeniería de software moderna: **separar estrictamente la configuración del código**.

- **Evita la recompilación y facilita el despliegue:** Según la metodología *Twelve-Factor*, la configuración incluye todo aquello susceptible de variar entre entornos (desarrollo, preproducción, producción), como por ejemplo, URLs de bases de datos o claves de Amazon S3. Extraer esto a un archivo de texto permite cambiar a dónde apunta la aplicación sin tener que recompilar el proyecto.
- **Seguridad:** Los archivos que contienen credenciales pueden y deben ser excluidos del control de versiones (por ejemplo, mediante `.gitignore`). La literatura arquitectónica condena la práctica de escribir credenciales directamente en el código fuente (hardcoding).

Valores adecuados VS inadecuados

- **Adecuados:** Cadenas de texto simples como credenciales, URLs, direcciones IP, tiempos de espera (*timeouts*) o separadores.
- **Inadecuados:** Estructuras de datos jerárquicas o complejas. El formato `.properties` es inherentemente plano (consiste en pares de clave y valor almacenados línea por línea). Tampoco es adecuado para datos binarios o bloques de texto multilínea muy extensos.

Limitaciones históricas y alternativas modernas

- **El problema de la codificación (Resuelto en Java 9):** Históricamente, la especificación de Java establecía que los archivos `.properties` debían estar codificados en **ISO-8859-1 (Latin-1)**. Si querías usar una tilde, un carácter asiático o una "ñ", estabas obligado a usar caracteres de escape Unicode (ej. `\u00F1`). Esto cambió de forma oficial a partir de Java 9 con la aprobación del **JEP 226 (JDK Enhancement Proposal)**, el cual dictaminó que la codificación predeterminada pasara a ser **UTF-8**, permitiendo la escritura natural de cualquier carácter.
- **Alternativas modernas:** Debido a la limitación de ser un formato estrictamente plano y sin tipado (todo es texto), la industria ha adoptado masivamente **YAML** y **JSON**. Estos formatos soportan anidación jerárquica y tipos de datos nativos (booleanos, listas, enteros).

Lectura desde Java y manejo de nulos

La clase oficial `java.util.Properties` hereda de `Hashtable` y requiere un flujo de datos (`InputStream`) para leer el archivo de texto.

```
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.Properties;

public class LectorConfig {
    public static void main(String[] args) throws Exception {
        Properties propiedades = new Properties();

        try (InputStream input = new FileInputStream("config.propertie
s")) {
            propiedades.load(input); // Lee los pares clave-valor
            String url = propiedades.getProperty("db.url");
        }
    }
}
```

¿Qué ocurre si la clave no existe?
La API de Java especifica que si llamas a `getProperty("clave_inexistente")` y la clave no se encuentra en el archivo, el método no lanza ninguna excepción, simplemente **devuelve** `null`.

Para evitar errores en tiempo de ejecución (como un `NullPointerException`), Java proporciona la sobrecarga `getProperty(String key, String defaultValue)`, que te permite definir un valor seguro por defecto si la configuración falta en el archivo original.