Actividad 1.7: Tratamiento ficheros Formación. Uso de fichero ”properties”

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

1. Cuatro formas distintas de gestionar valores constantes

1. **Clase o interfaz de constantes (`public static final`):** Definir atributos `public static final String RUTA = "..."` dentro de una clase de utilidades o una interfaz.
2. **Archivos de propiedades (`.properties`) con la API de Java:** Utilizar la clase nativa `java.util.Properties` para leer pares clave-valor desde un archivo de texto externo cargado con `FileInputStream` o `ClassLoader.getResourceAsStream()`.
3. **Variables de entorno del sistema (`System.getenv()`):** Leer constantes mediante `System.getenv("NOMBRE_VAR")`, configuradas directamente en el sistema operativo o en el entorno de ejecución.
4. **Propiedades del sistema de la JVM (`System.getProperty()`):** Pasar constantes mediante argumentos en la línea de comandos al arrancar la máquina virtual de Java (ejemplo: `java -Dconfig.ruta=/path/fichero PrincipalApp`).

2. Ventajas de usar .properties frente a constantes en código fuente

- **Desacoplamiento y mantenimiento:** Permite cambiar la configuración (rutas, separadores CSV, etc.) sin necesidad de modificar el código fuente ni recompilar la aplicación.
- **Reutilización y despliegue:** Facilita adaptar el comportamiento de la aplicación entre diferentes entornos (desarrollo, pruebas, producción) cambiando únicamente el archivo de configuración.
- **Separación de roles:** Permite a personal técnico, administradores de sistemas o usuarios finales ajustar parámetros sin tener que acceder al código fuente.

3. Tipos de valores adecuados y no adecuados

- **Valores adecuados:**
  - Rutas de archivos e identificadores de directorio.
  - Separadores de formato (como el carácter delimitador en archivos CSV).
  - Parámetros de conexión a bases de datos o URLs de servicios externos.
  - Mensajes de la interfaz de usuario e idiomas (internacionalización/i18n).
- **Valores NO adecuados:**
  - **Credenciales, claves secretas o claves API sin encriptar:** Al guardarse en texto plano, suponen un riesgo de seguridad elevado si se suben a repositorios de código.
  - **Lógica de negocio o estructuras de datos complejas:** No es adecuado almacenar arrays, objetos anidados o reglas de control de flujo.

4. Limitaciones de `.properties` y alternativas modernas

- **Limitaciones de `.properties`:**
  - Soporta únicamente pares clave-valor simples en formato texto (`String`).
  - No admite estructuras jerárquicas o anidadas de forma nativa.
  - No maneja tipos de datos fuertemente tipados (todos los valores se leen como texto y deben parsearse manualmente).
- **Alternativas modernas:**
  - **JSON / YAML:** Soportan estructuras jerárquicas, anidamiento y listas estructuradas.
  - **Bases de datos:** Útiles para configuraciones dinámicas que requieren persistencia y actualización en tiempo de ejecución.
  - **Inyección de dependencias (`Spring @Value` / `@ConfigurationProperties`):** Permiten inyectar configuraciones mapeadas a objetos directamente en el código de forma fuertemente tipada.

5. Carga de un `.properties` en Java y gestión de claves inexistentes

Para cargar un archivo `.properties` y acceder a sus valores mediante Java puro:

```
import java.io.InputStream;

import java.util.Properties;

import java.util.Scanner;


public class CargadorConfiguracion {


    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Properties props = new Properties();


        // Carga del archivo desde la carpeta de recursos/configuración

        try (InputStream input = CargadorConfiguracion.class.getClassLoader()

                .getResourceAsStream("config/application.properties")) {


            if (input == null) {

                System.out.println("Error: No se encontró el archivo de propiedades.");
```

```
            return;
        }

        // Cargar los pares clave-valor
        props.load(input);

        // 1. Obtener un valor existente
        String separador = props.getProperty("csv.separador");
        System.out.println("Separador CSV: " + separador);

        // 2. Obtener un valor definiendo un valor por defecto
        String ruta = props.getProperty("ruta.continente", "rutas/continentes.csv");
        System.out.println("Ruta configurada: " + ruta);

    } catch (Exception e) {
        System.err.println("Ocurrió un error al procesar el archivo: " + e.getMessage());
    } finally {
        sc.close();
    }
  }
}
```