1. Cuatro formas de gestionar constantes en un proyecto Java "puro"

Para un proyecto Java estándar sin frameworks (como Spring) ni gestores de dependencias (como Maven/Gradle), se pueden usar las siguientes alternativas:
• Clase o Interfaz de Constantes: Definir una clase final con miembros public static final (ej. public static final String CSV_SEPARATOR = ";";).
• Variables de Entorno del Sistema: Utilizar variables del sistema operativo y leerlas mediante System.getenv("VARIABLE_NAME").
• Argumentos de Línea de Comandos / Propiedades del Sistema: Pasar parámetros al arrancar la máquina virtual (java -Dmi.propiedad=valor -jar app.jar) y leerlos con System.getProperty("mi.propiedad").
• Archivos de texto planos personalizados (TXT/CSV): Crear un archivo propio con un formato básico (como clave=valor) y parsearlo manualmente usando BufferedReader y String.split().

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

• Modificación sin recompilar: Permite cambiar la configuración (como una IP o una ruta) directamente en el archivo sin tener que volver a compilar el código Java.
• Separación de entornos: Facilita cambiar entre producción, desarrollo o pruebas simplemente sustituyendo el archivo de configuración.
• Limpieza del código: Mantiene el código fuente libre de textos fijos ("hardcoded") y centrado en la lógica de negocio.
• Centralización: Todos los parámetros del sistema se encuentran organizados en un único lugar.

3. Valores adecuados e inadecuados para guardar en un .properties

• Valores adecuados: Parámetros que cambian según el entorno o la instalación.
    • Ejemplos: Rutas de archivos, credenciales de bases de datos, puertos de red, separadores de archivos CSV y URLs de APIs.
• Valores inadecuados:
    • Datos altamente sensibles sin cifrar: Contraseñas en producción o claves privadas (por motivos de seguridad).
    • Lógica de negocio o constantes inmutables: Valores matemáticos (como el valor de Pi) o estados fijos del programa que jamás van a cambiar; estos deben ir en el código.
    • Estructuras de datos complejas: Listas anidadas u objetos jerárquicos pesados.

4. Limitaciones de .properties y alternativas modernas

• Limitaciones:
    • Solo soporta texto plano (clave-valor) en una estructura plana (sin jerarquías ni anidamientos).
    • Soporte nativo de codificación limitado (históricamente ISO-8859-1, requiriendo secuencias Unicode para caracteres especiales).
    • No valida tipos de datos de forma automática (todo se lee como un String).
• Alternativas modernas:
    • JSON / YAML: Soportan estructuras de datos complejas, listas y jerarquías legibles.
    • Bases de datos: Ideales si las configuraciones deben cambiar dinámicamente en tiempo de ejecución.
    • Spring @Value / MicroProfile Config: Inyección automática de dependencias y gestión avanzada de entornos mediante anotaciones.