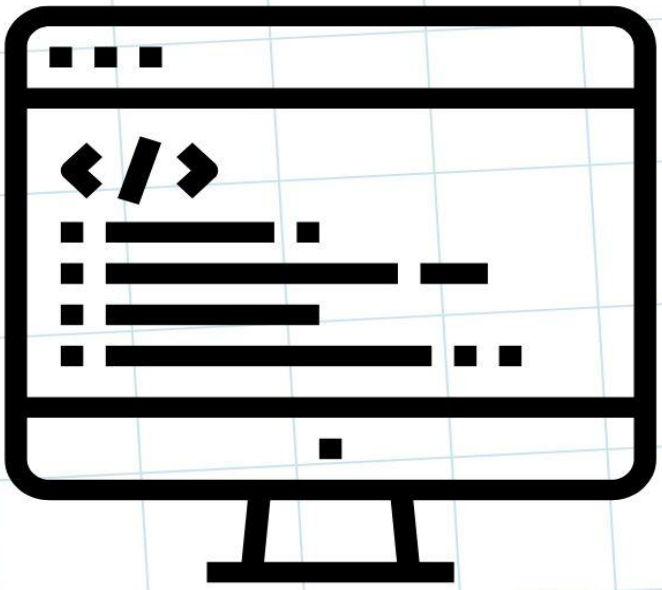
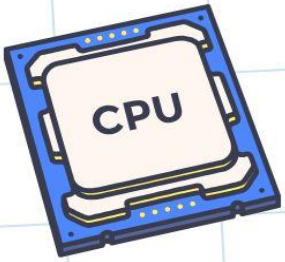
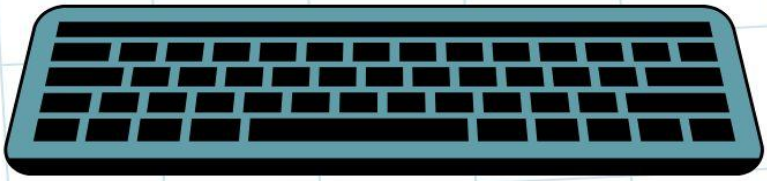
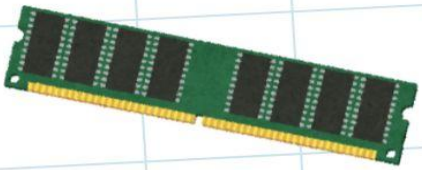
Memoria Técnica









1. Relación entre el programa Java, la memoria RAM, el procesador y los periféricos

Un programa Java interactúa con el procesador (CPU), la memoria RAM y los periféricos a través de un sistema jerárquico: la CPU ejecuta las instrucciones del código Java, la RAM proporciona un espacio de trabajo temporal y rápido para los datos e instrucciones, y los periféricos (como el teclado, la pantalla, etc.) son los dispositivos de entrada y salida con los que el programa interactúa. La CPU busca las instrucciones en la RAM, las ejecuta y devuelve los resultados, mientras que la interacción con los periféricos ocurre a través de las entradas/salidas de datos gestionadas por el sistema operativo y la CPU.

Relación entre los componentes:

- Procesador (CPU): Es el cerebro que ejecuta las instrucciones del programa Java. Busca las instrucciones y los datos en la memoria RAM, los procesa y calcula los resultados.
- Memoria RAM: Es el espacio de trabajo temporal donde se almacenan las instrucciones del programa Java y los datos con los que opera en un momento dado. La CPU accede a ella constantemente para obtener información rápida.
- Memoria Stack: Almacena variables locales y referencias a objetos, es de acceso rápido y corto plazo.
- Memoria Heap: Almacena objetos creados en tiempo de ejecución. Es de mayor tamaño y más lenta que la stack, con gestión dinámica.
- Periféricos: Son los dispositivos físicos con los que el programa interactúa para recibir información (entrada) o mostrar resultados (salida).
- Entrada: Un programa Java puede leer datos del teclado o de un archivo, que luego se almacenan temporalmente en la RAM para que la CPU pueda procesarlos.
- Salida: Los resultados del procesamiento del programa pueden mostrarse en la pantalla o guardarse en un archivo, enviando la información desde la RAM hacia el periférico correspondiente.
- Interconexión: Todos estos componentes están conectados a través de la placa base. La unidad de control del procesador es la encargada de gestionar el tráfico de información entre la CPU, la RAM y los periféricos.

2. Diferencia entre código fuente, objeto y ejecutable

Característica	Código Fuente	Código Objeto	Código Ejecutable
Descripción	Texto escrito en un lenguaje de programación de alto nivel que es legible por humanos.	Resultado de la compilación del código fuente; es lenguaje máquina, pero está incompleto o no está enlazado.	Resultado del enlazado de uno o más archivos de código objeto y bibliotecas; está completo y listo para ejecutarse.
Comprensión	Legible por humanos, pero no por la computadora.	Se acerca más a ser legible por la computadora, pero no se puede ejecutar directamente.	Es directamente ejecutable por la computadora.
Proceso	Escrito por un programador.	Generado por un compilador.	Generado por un enlazador (linker).

En el contexto del proyecto de la lista de la compra, se diferencian claramente tres tipos de archivos:
• Código fuente: Es el archivo .java donde el programador escribe las instrucciones del programa. Ejemplo: GestorListaCompra.java.
• Código objeto: Es el bytecode generado automáticamente por el compilador al traducir el código fuente. Tiene extensión .class.
• Ejecutable: Es el archivo que permite ejecutar el programa. En Java, no existe un .exe como tal, pero el programa es ejecutable a través del comando 'java NombreClase' o mediante un archivo .jar.

3. Evalúa la funcionalidad que te ha proporcionado Eclipse como IDE.

Durante el desarrollo del proyecto, se ha utilizado un entorno de desarrollo integrado (IDE), el cual facilita la programación gracias a varias herramientas. Algunos aspectos evaluados son:
• Ventajas: autocompletado, depuración de errores, ejecución rápida del programa, árbol de archivos visible y fácil gestión del proyecto.
• Desventajas: Ha requerido configuraciones adicionales, como la instalación de JDK25 y JUnit para poder hacer el testing (Esto ya lo tenía instalado de antes pero en caso de no haberlo instalado antes, hubiera tenido que instalarlo para hacer este proyecto) y en ocasiones genera archivos automáticos que pueden confundir.
En general, el uso del IDE ha mejorado la productividad y ha facilitado la comprensión del proyecto.

4. Explica si tu proceso de desarrollo se ajustó a una metodología ágil (como Scrum o Kanban) y justifica tu respuesta.

La metodología que utilicé no fue ni waterfall, ni scrum ni kamban en su estado puro. Mezclé las tres metodologías.

El modelo en cascada se caracteriza por avanzar por fases lineales: análisis-diseño-implementación-pruebas-entrega.

En este proyecto se aprecia lo siguiente:

1. **Planificación inicial del proyecto**: Antes de empezar a programar se definió qué funcionalidades debía tener la aplicación (añadir, eliminar, buscar, listar, vaciar productos).
   Esto corresponde a la fase de análisis del modelo en cascada.

2. **Estructuración del proyecto antes del código**:
   Se creó la estructura /src y /test, se preparó el repositorio y se configuró el entorno de trabajo, lo cual se parece a la fase de diseño del modelo en cascada.

3. **Implementación ordenada por funcionalidades**: Aunque se hizo de forma incremental mediante ramas (método ágil), las funcionalidades fueron programadas siguiendo un orden lógico previamente pensado, lo que refleja la secuencia lineal típica del método waterfall.

4. **Pruebas después de implementar cada parte**: La verificación del programa y la integración final se hizo al terminar varias funcionalidades y al realizar merges hacia develop y main. Esto tiene relación con la fase de pruebas del modelo en cascada.

5. **Entrega final tras completar todas las fases**: Se generó la release v1.0 y se prepararon los documentos finales (memoria, incidencias, README). Esta parte se asemeja a la fase de entrega del método waterfall.

Por otro lado, durante el desarrollo del proyecto se ha aplicado la **metodología ágil**, aunque no se ha aplicado una de ellas de forma completa o formal como Scrum o Kanban. Sin embargo, sí se han utilizado varios principios ágiles.

- **Trabajo iterativo e incremental:** Las funcionalidades se desarrollaron en pequeñas partes independientes mediante ramas feature (feature/add-product, feature/remove-product, feature/get-list, etc.). Cada iteración añadía una funcionalidad concreta y probada antes de integrarse en la rama develop.
  Este enfoque es de la metodologías ágil, donde se priorizan entregas pequeñas y continuas.
- **Integración frecuente:** Se realizaron merges frecuentes hacia develop y posteriormente hacia main, reduciendo riesgos y permitiendo validar los avances de forma constante, algo característico de Scrum y Kanban.

- **Gestión visual del trabajo (similar a Kanban):** Aunque no se usó un tablero formal, las tareas estaban claramente separadas por funcionalidades (añadir, eliminar, listar, buscar…).
  Esto es equivalente a dividir el trabajo en unidades manejables, tal como propone Kanban.
- **Flexibilidad ante cambios:** Cuando surgieron incidencias (como conflictos, reorganización de ramas, errores en el push), se resolvieron rápidamente y se ajustó el flujo de trabajo, siguiendo el principio ágil de adaptación sobre planificación estricta.
- **Control de versiones como herramienta ágil:** Git y GitHub se usaron para mantener un historial claro, con ramas separadas, releases (v1.0), merges y resolución de conflictos, lo cual encaja con la metodología ágil.

En conclusión:

El proyecto aplicó varios principios ágiles:

• Trabajo por funcionalidades pequeñas mediante ramas "feature".

• Integraciones frecuentes y revisiones continuas mediante merges a develop.

• Flexibilidad para modificar código y corregir errores durante el desarrollo.

También se utilizaron elementos del modelo en cascada:

• Se definió al principio qué debía hacer el programa (fase de análisis).

• Se planificó la estructura del proyecto y del repositorio (diseño).

• Se implementaron las funcionalidades en un orden lógico (implementación).

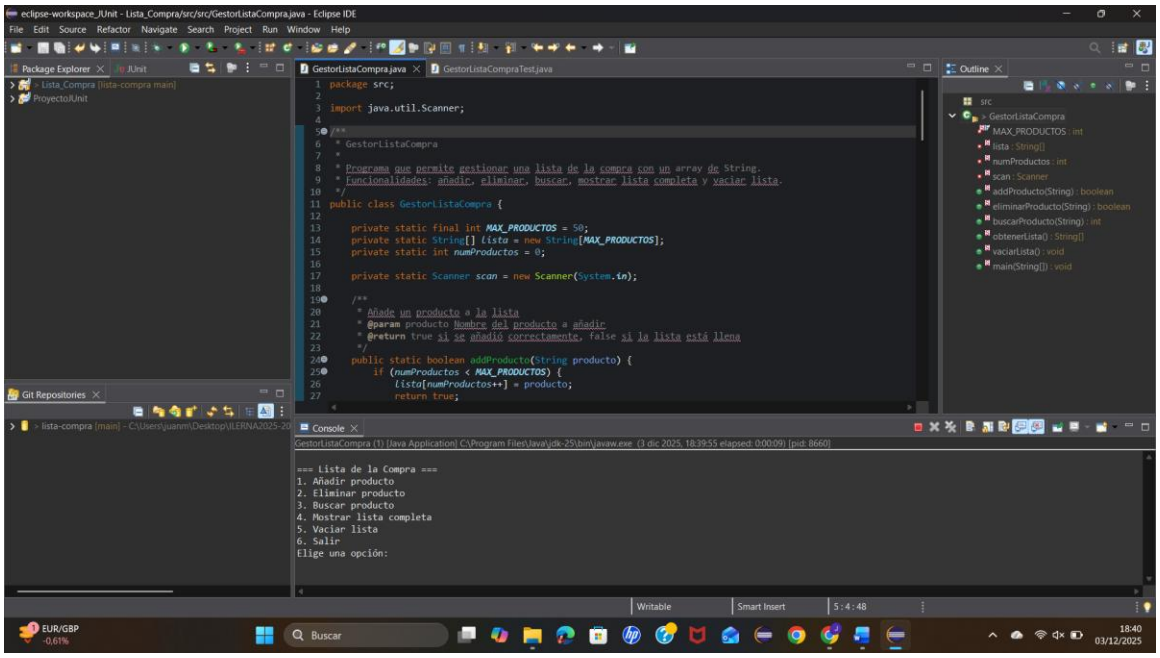
• Las pruebas se realizaron después de integrar partes del código (pruebas).

• Se generó una versión final y documentación (entrega).

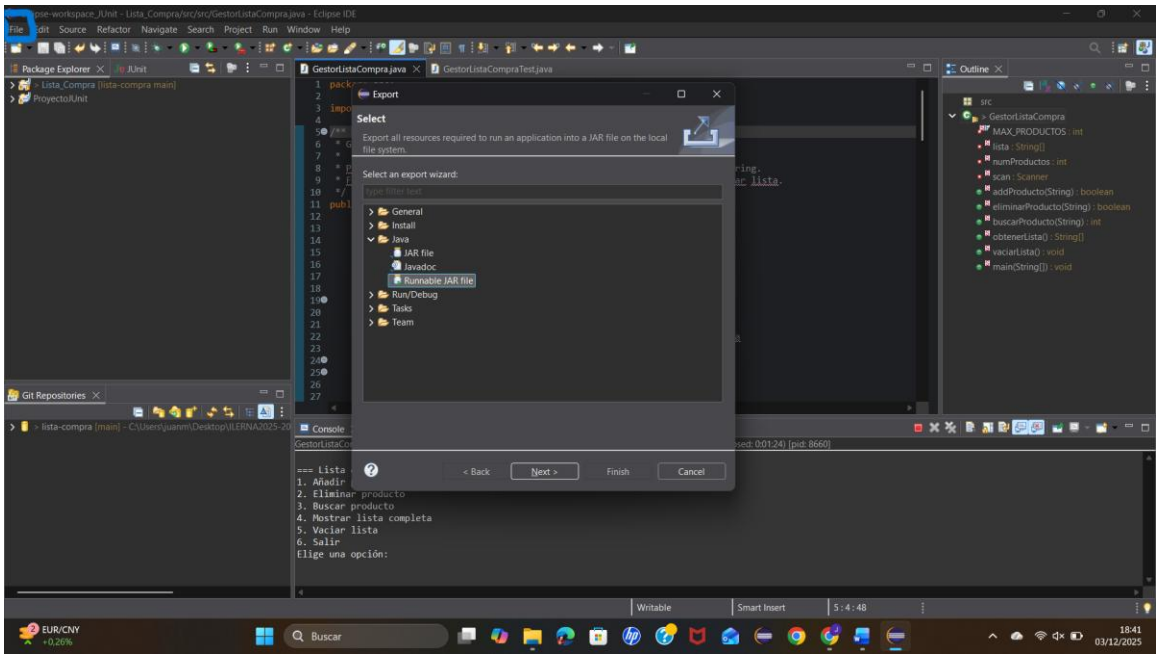
5. Cómo generar ejecutables desde Eclipse y desde otros IDES (al menos uno), o desde terminal.

Para generar un ejecutable en eclipse tienes que seguir los siguientes pasos:

1. **Abre tu proyecto en Eclipse.**



2. **En el menú (la barra de arriba), dale a file (lo marcado en azul), luego le das a export, te saldrá una ventana emergente con muchas carpetitas, le das a la carpeta donde pone "Java" y finalmente dentro de java le tienes que dar a "Runnable JAR file" (el recuadro que tengo marcado).**



3. Selecciona la **clase principal** (es la que contiene main, prácticamente donde hemos creado todo nuestro proyecto).

4. Elige la ubicación donde quieres guardar el archivo .jar, yo en mi caso he elegido el escritorio.

5. Configura las opciones de librerías (por ejemplo “Extract required libraries into generated JAR”). Simplemente tienes que seleccionar el cuadrito que hay al lado de la frase.

6. Haz clic en **Finish**.

- Esto genera un archivo .jar ejecutable que puedes abrir en cualquier sistema con Java instalado.

- Para ejecutarlo, basta con hacer doble clic o usar: