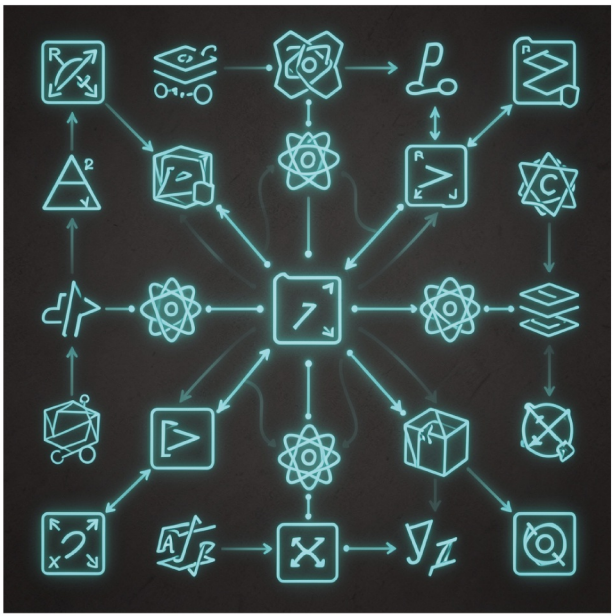
Programación orientada a objetos en Java



Lucien Sina

Programación orientada a objetos en Java

Lucien Sina

16 de febrero de 2026

Índice general

Prefacio

Este libro se escribió con un objetivo claro en mente: guiarlo a través de los conceptos básicos de la programación y avanzar de manera constante hacia los principios del diseño orientado a objetos, un paradigma esencial para el desarrollo de software moderno.

La programación no consiste únicamente en escribir código; se trata de resolver problemas, crear soluciones eficientes y aprender a pensar de forma lógica y sistemática. Este libro comienza con los conceptos básicos: presenta qué es la programación, cómo se forman los algoritmos y cómo Java, un lenguaje versátil y ampliamente utilizado, puede ser su herramienta para implementar soluciones.

A medida que avance en los capítulos, explorará estructuras de programación esenciales, como bucles, tipos de datos y métodos. Luego, el libro profundiza en los principios del diseño orientado a objetos, cubriendo conceptos como encapsulación, herencia, polimorfismo y clasificación. A lo largo del camino, ejercicios, ejemplos y soluciones lo ayudarán a reforzar su comprensión y lo

alentarán a aplicar lo que ha aprendido.

Este libro está diseñado para principiantes, pero no se detiene allí. Para aquellos que estén listos para dar el siguiente paso, ofrece una transición fluida hacia temas más avanzados. Además, he incluido un vistazo a libros futuros, como uno dedicado a habilidades de programación avanzadas, otro a algoritmos y estructuras de datos, y una exploración más profunda de temas teóricos como la computabilidad y la complejidad. Estos libros están destinados a complementar este, ayudándote a crecer como programador y solucionador de problemas.

Espero que este libro te sirva de ayuda en tu viaje hacia la programación. Si tienes comentarios, preguntas o sugerencias, no dudes en ponerte en contacto conmigo.

¡Feliz codificación!

Capítulo 1

Problemas, algoritmos y programas

1.1. Introducción

Todo programa comienza con un problema. En programación, un problema es una tarea o pregunta que requiere una solución, y un programa es la herramienta que proporciona esta solución a través de una serie de pasos lógicos. El objetivo de este capítulo es presentarle conceptos, problemas, algoritmos y programas fundamentales, y brindarle una descripción general de los lenguajes y paradigmas de programación que influyen en la forma en que se crean e implementan las soluciones.

1.2. ¿Qué es un problema?

En programación, un problema es cualquier tarea que necesitamos resolver o cualquier pregunta que requiere una respuesta. Por ejemplo: ¿Cómo ordeno una lista de números? o: ¿Cómo puedo crear un software que ayude a administrar el inventario en una tienda?

Un problema de programación normalmente tiene:

Entradas: datos o información proporcionada para la solución (por ejemplo, una lista de números). Salidas: el resultado deseado (por ejemplo, la lista ordenada en orden ascendente). Restricciones: reglas o límites dentro de los cuales debe operar la solución (por ejemplo, la lista no debe exceder una cierta longitud o el proceso de ordenación debe ser eficiente).

Ejercicio 1.1: Identifique un problema del mundo real que encuentre, como encontrar la ruta más corta para ir al trabajo. Identifique:

Las entradas Las salidas deseadas Cualquier restricción Solución al Ejercicio 1.1: Ejemplo: Planificar la ruta más corta al trabajo.

Entradas: Ubicación de partida, destino, datos de tráfico. Salidas: La ruta más corta. Restricciones: Minimizar el tiempo de viaje, evitar peajes o carreteras cerradas.

1.3. Pasos para transformar un problema del mundo real en un programa

El viaje desde un problema del mundo real a un programa funcional implica una serie de transformaciones:

Problema del mundo real: Identificar la tarea que necesitamos resolver. Abstracción: Simplificar el problema del mundo real centrándose en los detalles relevantes e ignorando los innecesarios. Descripción del problema: Describir el problema en términos generales. Precisión y formalización: Refinar la descripción para especificar detalles y requisitos clave. Especificación del problema: Definir claramente las entradas, salidas y restricciones. Declarar el posible espacio de solución: Explorar formas potenciales de resolver el problema. Algoritmo: Crear un plan paso a paso para la solución. Programación en un lenguaje de orden superior: Traducir el algoritmo a un lenguaje de programación. Traducción a código de máquina: Convertir el código en instrucciones que el hardware de una computadora puede ejecutar. Entrada: Proporcionar los datos al programa. Salida/Resultado: El programa procesa la entrada para producir el resultado deseado.

Ejercicio 1.2: Piense en una tarea sencilla, como hornear un pastel. Resuma cada paso para transformar la tarea en una serie de instrucciones que una persona podría seguir, similares a los pasos anteriores.

Solución del Ejercicio 1.2: Ejemplo: Hornear un pastel.

Problema del mundo real: Elaboración de un pastel. Abstracción: Centrarse únicamente en los ingredientes y pasos necesarios. Descripción del problema: ¿Cómo se hace un pastel? Precisión: Especificar el tipo de pastel, los ingredientes y el método de cocción. Especificación: Receta exacta con ingredientes y pasos. Solución: Identificar todos los pasos necesarios (mezcla, horneado). Algoritmo: Instrucciones paso a paso. Lenguaje de orden superior: Tarjeta de receta o instrucciones de cocción. Código de máquina: Acciones realizadas por el panadero. Entrada: Ingredientes. Salida: Un pastel recién horneado.

1.4. ¿Qué es un algoritmo?

Un algoritmo es un conjunto claro de instrucciones paso a paso diseñadas para resolver un problema. Piense en un algoritmo como si fuera una receta: es una secuencia de pasos que debe seguir para lograr un resultado específico.

Por ejemplo, si su problema es ordenar una lista de números, un algoritmo proporciona los pasos para ordenar los números. Los algoritmos deben ser:

Claro: cada paso debe ser inequívoco y fácil de entender. Preciso: cada detalle está definido sin ambigüedad. Eficiente: idealmente, los pasos deben conducir a una solución de la manera más eficaz posible.

Ejercicio 1.3: Escribe un algoritmo para cepillarte los dientes. Cada paso debe ser claro y preciso.

Solución del ejercicio 1.3:

Coge el cepillo de dientes. Aplica pasta de dientes en el cepillo. Coloca el cepillo de dientes en tu boca. Pasa el cepillo por todos los dientes durante dos minutos. Enjuágate la boca y el cepillo de dientes.

1.5. ¿Qué es un programa?

Un programa es la implementación de un algoritmo de forma que una computadora pueda ejecutarlo. Los programas se escriben en lenguajes de programación que nos permiten traducir los algoritmos en instrucciones que una computadora puede entender. Por ejemplo, un programa que ordena números seguiría los pasos de un algoritmo de ordenación, expresado en un lenguaje que la computadora pueda ejecutar.

Los programas consisten en:

Código: Instrucciones escritas en un lenguaje de programación. Datos: Información procesada por el programa. Lógica: Toma de decisiones que determina las acciones del programa.

1.6. Lenguajes de programación y paradigmas

Los lenguajes de programación son herramientas que utilizamos para escribir programas, cada uno con su propia sintaxis (reglas para escribir instrucciones) y características. Por ejemplo:

Python: conocido por su simplicidad, lo que lo hace ideal para principiantes. Java (que utilizaremos en este libro): conocido por su confiabilidad y capacidad para ejecutarse en múltiples plataformas.

Paradigmas de programación Un paradigma de programación es una forma de organizar y pensar el código. Diferentes paradigmas se adaptan a distintos tipos de problemas y comprenderlos nos ayuda a elegir el mejor enfoque para una tarea determinada. Veamos algunos paradigmas comunes:

-Programación imperativa: escribir secuencias de instrucciones que le indican a la computadora cómo lograr un resultado. Algunos ejemplos incluyen: -Programación procedimental: enfocarse en funciones que realizan operaciones en una secuencia (por ejemplo, pasos para resolver un problema matemático).

Programación Orientada a Objetos (POO): Organizar el código en objetos que representan cosas del mundo real, como crear un sistema de biblioteca con objetos de libros y de usuarios. Programación Declarativa: Centrarse en cuál debería ser el resultado, dejando el cómo al lenguaje o entorno de programación. Algunos ejemplos incluyen:

Programación funcional: define las operaciones como una serie de funciones, a menudo sin cambiar el estado o los datos. Es común en aplicaciones matemáticas y pone énfasis en las relaciones claras entre las entradas y las salidas. Programación lógica: utiliza la lógica y las reglas para especificar los resultados deseados. Un ejemplo es Prolog, donde se utilizan relaciones y hechos

para inferir nueva información. Programación orientada a aspectos (AOP): separa las preocupaciones transversales (por ejemplo, el registro o el manejo de errores) para que el código sea más modular y más fácil de mantener. La AOP ayuda a aislar la funcionalidad que afecta a múltiples áreas de un programa.

Ejercicio 1.4: Identifique tres problemas diferentes y piense qué paradigma de programación podría ser más adecuado para cada uno. Justifique sus elecciones.

Solución del ejercicio 1.4:

Ordenar una lista de números: procedimental, ya que implica una secuencia sencilla de pasos. Sistema de gestión de inventario: orientado a objetos, ya que los artículos y las operaciones de inventario se pueden representar como objetos. Sistema experto para diagnóstico médico: programación lógica, ya que puede utilizar reglas y hechos para inferir posibles diagnósticos.

1.7. Resumen

En este capítulo, definimos los elementos fundamentales de la programación: problemas, algoritmos y programas. Exploramos cómo un problema del mundo real se transforma en una solución programada a través de una serie de pasos. También presentamos lenguajes y paradigmas de programación, brindando contexto sobre cómo varían los enfoques de programación y por qué son adecuados para diferentes tareas. Ahora, con estos conceptos básicos en mente, ¡está listo para profundizar

en la programación y comenzar a crear soluciones!

Capítulo 2

Configuración JDK

2.1. Introducción

Antes de sumergirnos en la programación Java, es esencial configurar un entorno de desarrollo para escribir, compilar y ejecutar código Java de manera eficiente. En este capítulo, lo guiaremos en la instalación del Kit de desarrollo de Java (JDK) y la configuración de un entorno de desarrollo integrado (IDE). Cubriremos dos opciones populares, Eclipse y Android Studio, para que pueda elegir la que mejor se adapte a sus necesidades. Al final de este capítulo, tendrá un entorno completo listo para programar.

2.2. Instalación del kit de desarrollo de Java (JDK)

El JDK es necesario para compilar y ejecutar programas Java, y proporciona las bibliotecas principales necesarias para trabajar con Java.

Guía paso a paso para instalar el JDK: Descargue el JDK:

Visite la página oficial de descarga de Oracle JDK o instale OpenJDK desde el sitio web de OpenJDK. Elija la última versión de JDK compatible con su sistema operativo (Windows, macOS o Linux). Instale el JDK:

Siga las instrucciones de instalación que aparecen en la página de descarga. La mayoría de los instaladores ofrecen una ruta predeterminada, que puede conservar, o puede elegir una ruta personalizada. Establecer variables de entorno (solo para Windows):

Abra Propiedades del sistema > Configuración avanzada del sistema > Variables de entorno. En "Variables del sistema", busque Ruta, selecciónela y haga clic en ."Editar". Agregue una nueva ruta al directorio bin de su JDK, por ejemplo

```
C:\Program Files\Java\jdk-xx\bin
```

Verifique la instalación:

Abra una terminal o un símbolo del sistema y escriba java -version. Si el JDK está instalado correctamente, verá la versión del JDK en pantalla. Ejercicio 2.1: Abra su terminal o símbolo del sistema y ejecute java -version y

javac -version. Confirme que ambos comandos reconocen la instalación de Java. Si ve la información de la versión, su JDK está instalado correctamente.

Solución al ejercicio 2.1: Si la instalación fue exitosa, debería ver un resultado similar a este: java version "17.0.1"2021-10-19 LTS Java(TM) SE Runtime Environment (build 17.0.1+12-39) Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-39, mixed mode, sharing)

2.3. Elección de un IDE: Eclipse vs. Android Studio

Un entorno de desarrollo integrado (IDE) es esencial para la programación, ya que proporciona herramientas para editar, compilar y depurar código de manera eficiente. Comparemos brevemente Eclipse y Android Studio para ayudarlo a elegir el que mejor se adapte a sus necesidades.

Eclipse:

Ventajas: Altamente personalizable, ligero, popular para una amplia gama de aplicaciones Java. Desventajas: Puede requerir complementos para ciertas funcionalidades, menos optimizado para el desarrollo de Android. Ideal para: Desarrollo general de Java y aplicaciones más grandes.

Estudio Android:

Pros: Excelente para el desarrollo de Android, incluye bibliotecas y herramientas específicas de Android. Contras: Requiere más recursos del sistema, ideal para

proyectos centrados en Android. Ideal para: Principiantes interesados en el desarrollo de aplicaciones móviles con Java. Elige el IDE que se ajuste a tus objetivos. Si eres nuevo en programación, es posible que te resulte más fácil comenzar con la interfaz más sencilla de Eclipse. Si te interesa el desarrollo de aplicaciones móviles, Android Studio te proporcionará una base sólida.

2.4. Configuración de Eclipse

Guía paso a paso para instalar y configurar Eclipse Descargar Eclipse:

Vaya a la página de descarga de Eclipse. Seleccione "Eclipse IDE for Java Developers" y descargue el instalador para su sistema operativo. Instale Eclipse:

Ejecute el instalador descargado y seleccione las opciones de instalación predeterminadas. Una vez instalado, inicie Eclipse. Cree un nuevo proyecto Java:

Abra Eclipse y seleccione Archivo > Nuevo proyecto Java > . Asigne un nombre al proyecto (por ejemplo, "Hola Mundo") y haga clic en "Finalizar". Escriba su primer programa:

Haz clic derecho en la carpeta src en el Explorador de proyectos. Selecciona Nueva clase > , nómbrala HelloWorld y marca "public static void main(String[] args)" para incluir el método principal. Dentro del método principal, escribe:

```
1 public class HelloWorld {
2     public static void main(String[] args) {
```

```
3         System.out.println("Hello, World!");
4     }
5 }
```

Guarda el archivo y ejecútalo haciendo clic derecho y seleccionando Ejecutar como > Aplicación Java. Ejercicio 2.2: Escribe y ejecuta tu primer programa "¡Hola, mundo!" en Eclipse, siguiendo los pasos anteriores. Confirma que el resultado se muestra en la consola.

2.5. Configuración de Android Studio

Guía paso a paso para instalar y configurar Android Studio Descargar Android Studio:

Visita la página de descarga de Android Studio y descarga el instalador para tu sistema operativo. Instalar Android Studio:

Ejecute el instalador y siga las instrucciones para completar la instalación. Android Studio también descargará los SDK y las herramientas necesarios para el desarrollo de Android. Cree un nuevo proyecto:

Abre Android Studio y selecciona Nuevo proyecto. Elige una plantilla de proyecto, como "Sin actividad" para un programa Java simple o "Actividad vacía" si te interesan las aplicaciones de Android. Ponle un nombre a tu proyecto (por ejemplo, "Hola Mundo") y elige "Java" como lenguaje. Escribe tu primer programa:

En el archivo Java principal del proyecto, crea una clase HelloWorld (o edita la actividad principal para

proyectos Android). En el método principal, escribe:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Si se trata de una aplicación de Android, deberá configurar un TextView en el diseño XML para mostrar "¡Hola, mundo!.[en] la pantalla. Ejecute su programa:

Para proyectos Java que no sean de Android, ejecute el programa haciendo clic derecho en el archivo Java principal y seleccionando Ejecutar. Para aplicaciones de Android, seleccione Ejecutar > Ejecutar 'aplicación' para probar la aplicación en un emulador o dispositivo conectado. Ejercicio 2.3: Cree y ejecute su programa "Hello, World!.[en] Android Studio. Si trabaja con una aplicación de Android, muestre el mensaje en la pantalla usando un widget TextView.

2.6. Solución de problemas de configuración habituales

La configuración de un IDE puede provocar a veces problemas inesperados. A continuación, se indican algunos problemas habituales y sus soluciones:

Español:JDK no reconocido: asegúrese de que el JDK esté instalado correctamente y de que las variables de entorno estén configuradas (usuarios de Windows). El

IDE falla o no se abre: verifique si su sistema cumple con los requisitos mínimos del IDE. El programa no se está ejecutando: verifique que el método principal esté escrito correctamente como public static void main(String[] args). Asegúrese de estar ejecutando el archivo correcto. Consejo: si los problemas persisten, consulte la documentación oficial o los foros en línea para obtener consejos sobre cómo solucionar problemas. Tanto Eclipse como Android Studio tienen grandes comunidades, por lo que la mayoría de los problemas tienen soluciones publicadas en línea.

2.7. Resumen

En este capítulo, configuramos las herramientas esenciales necesarias para la programación Java. Ya sea que elija Eclipse o Android Studio, ahora tiene un entorno donde puede escribir, ejecutar y solucionar problemas de código Java. Esta configuración es la base para su viaje hacia la programación.

En el próximo capítulo, exploraremos la estructura del programa Java, aprenderemos sobre diagramas de sintaxis, encabezados de programas y otros aspectos fundamentales del código Java. ¡Felicitaciones por configurar y escribir su primer programa!

Capítulo 3

Estructura de los programas Java

3.1. Introducción a la estructura del programa

Un programa Java tiene una estructura específica que permite compilarlo y ejecutarlo. Comprender esta estructura le ayudará a leer, escribir y depurar código de manera eficaz. En este capítulo, desglosaremos las partes de un programa Java, exploraremos cómo representar visualmente la sintaxis del programa y presentaremos conceptos clave como encabezados de programa, bloques, partes de declaración y partes de instrucción.

3.2. Diagramas de sintaxis: visualización de la sintaxis de Java

Los diagramas de sintaxis (también llamados diagramas de ferrocarril) son una forma útil de visualizar la estructura de los lenguajes de programación. Ilustran las reglas para combinar elementos en Java y ayudan a aclarar cómo encajan los diferentes componentes del código.

Conceptos clave: Terminales: elementos fijos del lenguaje, como palabras clave (class, public, static) y símbolos (, , ;), que deben aparecer exactamente como están escritos. No terminales: elementos variables que representan marcadores de posición o patrones que deben llenarse con valores específicos, como identificadores (por ejemplo, nombres de variables) o expresiones. Ejemplo: un diagrama de sintaxis para una definición de clase Java podría mostrar que una clase comienza con la palabra clave class, seguida de un identificador (el nombre de la clase) y luego el bloque de código ... .

3.3. Estructura básica de un programa Java

La estructura de un programa Java generalmente consta de las siguientes partes clave:

1. Encabezado del programa (definición de clase):

Todo programa Java comienza con una definición de clase. El encabezado del programa incluye palabras clave

como public y class , seguidas del nombre de la clase, y es la estructura de nivel superior. Ejemplo:

```
public class HelloWorld {
    // code block
}
```

Diagrama de sintaxis para la definición de clase

público (opcional) -> clase -> NombreClase -> -> Bloque de código ->

Explicación:

public (opcional): El modificador de acceso, que es opcional aquí. class: La palabra clave para definir una clase. ClassName: Marcador de posición para el nombre de la clase (no terminal). : Las llaves encierran el bloque de código, que contiene el cuerpo principal de la clase.

2. Método principal (Punto de entrada del programa):

El método principal, public static void main(String[] args), es el punto de entrada para los programas Java. Este método define lo que ejecutará el programa cuando se ejecute. Ejemplo:

```
public static void main(String[] args) {
    // statements
}
```

3. Bloque de código:

Un bloque de código es una sección de código encerrada en , que agrupa instrucciones o declaraciones. Los bloques definen el alcance de las secciones de código, lo que significa que las variables y las instrucciones dentro