

PROMETEO

Unidad 1: Fundamentos y Entorno de Programación

Configurar un entorno de desarrollo profesional es mucho más que instalar software en tu ordenador.

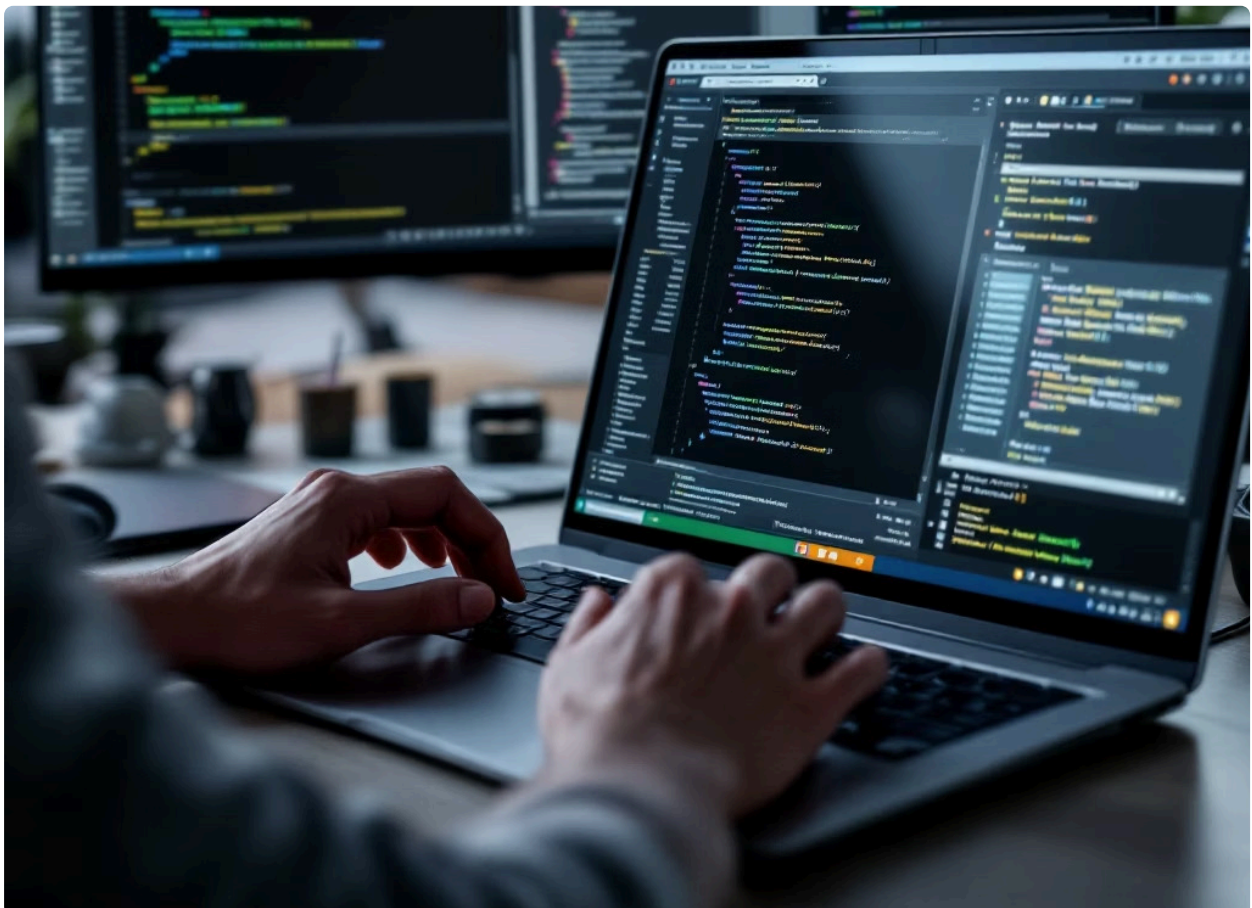
Sesión 1: Instalación de JDK, IDEs (Eclipse, IntelliJ, VSCode) y control de versiones con Git

Configurar un entorno de desarrollo profesional es mucho más que instalar software en tu ordenador. Es crear un ecosistema de trabajo que te acompañará durante toda tu carrera como desarrollador y que determinará tu productividad diaria. No se trata de una tarea que hagas una vez y olvides, sino de una inversión estratégica en tu futuro profesional.

El JDK (Java Development Kit) es el conjunto completo de herramientas que necesitas para desarrollar aplicaciones Java. Incluye el compilador javac que transforma tu código fuente en bytecode ejecutable, la máquina virtual JVM que ejecuta ese bytecode, y una extensa biblioteca de clases que proporcionan funcionalidades básicas. La elección de la versión correcta es crucial: las versiones LTS (Long Term Support) como Java 8, 11, 17 y 21 ofrecen estabilidad y soporte a largo plazo, fundamentales para proyectos empresariales.

Los IDEs modernos han revolucionado la forma de programar. IntelliJ IDEA destaca por su inteligencia artificial integrada que sugiere correcciones, refactorizaciones automáticas, y detección de errores en tiempo real. Eclipse, con su arquitectura de plugins, es especialmente popular en entornos empresariales por su flexibilidad y herramientas de integración. VSCode, aunque técnicamente un editor, se ha convertido en una alternativa ligera pero potente gracias a su ecosistema de extensiones.

Git no es solo una herramienta técnica, es la base de la colaboración moderna en desarrollo de software. Permite rastrear cada cambio en tu código, crear ramas para experimentar sin riesgo, y coordinar el trabajo de equipos distribuidos globalmente. Plataformas como GitHub, GitLab y Bitbucket extienden Git con funcionalidades de revisión de código, integración continua, y gestión de proyectos.



Esquema visual

JDK

- Compilador javac
- JVM
- Bibliotecas

IDE

- IntelliJ IDEA
- Eclipse
- VSCode

Git

- Control de versiones
- Colaboración
- GitHub/GitLab

Este diagrama muestra la interrelación entre los componentes fundamentales del entorno de desarrollo. El JDK proporciona las herramientas básicas de compilación y ejecución, el IDE mejora la productividad del desarrollador, y Git garantiza el control de versiones y la colaboración. Cada componente tiene un propósito específico pero trabajan juntos para crear un entorno cohesivo y eficiente.





Caso de Estudio: Spotify y su Stack Tecnológico

Spotify, con más de 500 millones de usuarios activos, ha construido uno de los ecosistemas de desarrollo más sofisticados del mundo. Su equipo de más de 4,000 ingenieros utiliza Java como uno de sus lenguajes principales para servicios backend que procesan millones de streams simultáneos.

La estandarización del entorno de desarrollo fue crucial para su escalabilidad. Cada desarrollador configura su entorno local usando scripts automatizados que instalan JDK 17 LTS, IntelliJ IDEA Ultimate con plugins específicos de Spotify, y configuraciones Git preestablecidas. Esta automatización reduce el tiempo de onboarding de nuevos desarrolladores de semanas a días.

Su flujo de trabajo Git es especialmente sofisticado: utilizan trunk-based development con ramas de vida corta, revisiones de código obligatorias, y despliegues automatizados que pueden procesar más de 100,000 deployments por día. La integración continua ejecuta más de 150,000 tests automáticos diarios, garantizando que los cambios no rompan funcionalidades existentes.

El resultado de esta estandarización es impresionante: tiempo promedio de resolución de bugs reducido en 60%, productividad de desarrolladores incrementada en 40%, y capacidad de lanzar nuevas funcionalidades semanalmente a escala global.

Herramientas y Consejos

1

Instala SDKMAN para gestionar múltiples versiones de JDK sin conflictos. Este gestor de versiones te permite cambiar entre Java 8, 11, 17, o 21 según las necesidades de cada proyecto, algo esencial cuando trabajas en múltiples aplicaciones con diferentes requisitos.

2

Configura IntelliJ IDEA Community Edition con plugins esenciales: SonarLint para análisis de calidad de código en tiempo real, GitToolBox para integración Git avanzada, Rainbow Brackets para mejorar legibilidad de código anidado, y Key Promoter X para aprender atajos de teclado automáticamente.

3

Establece un flujo de trabajo Git consistente desde el principio: commits descriptivos con formato convencional, ramas con nomenclatura clara, y hooks pre-commit para verificar calidad de código antes de cada commit.



Mitos y Realidades

✗ **Mito:** "Todos los IDEs son iguales, da igual cuál usar"

→ **FALSO.** Cada IDE tiene fortalezas específicas que impactan directamente tu productividad. IntelliJ IDEA destaca en refactoring inteligente y análisis de código, Eclipse en plugins empresariales y herramientas de debugging avanzado, VSCode en ligereza y personalización extrema. La elección correcta puede aumentar tu productividad entre 20-40% según estudios de JetBrains.

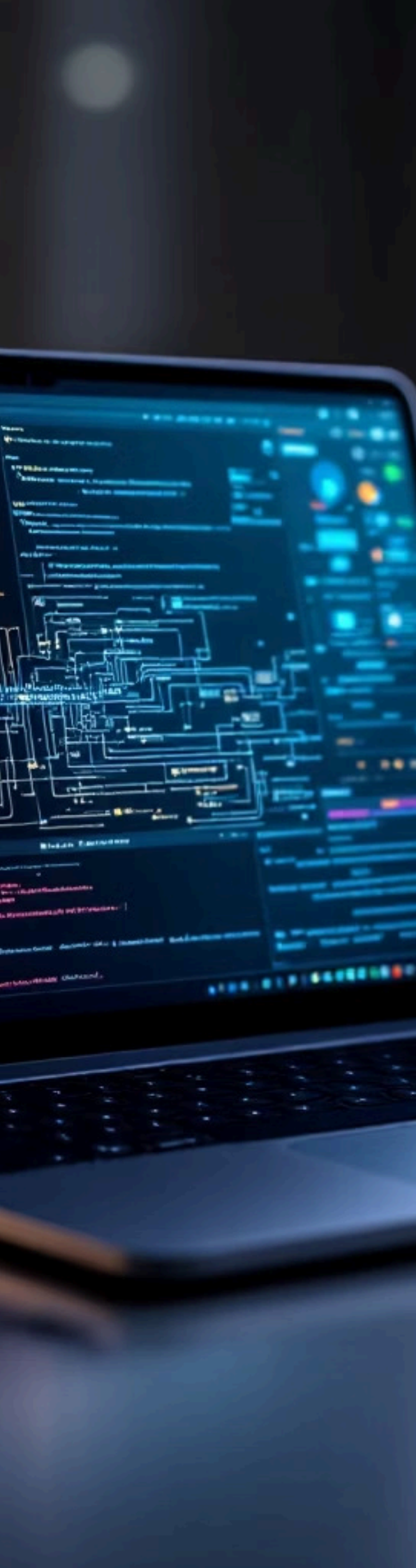
✗ **Mito:** "Git es solo para proyectos grandes o equipos"

→ **FALSO.** Git aporta valor incluso en proyectos personales pequeños. Te permite experimentar con ramas sin miedo a romper código funcional, mantener historial completo de tu evolución como programador, y tener backup automático en la nube. Además, dominar Git es requisito indispensable en cualquier entrevista técnica moderna.

Resumen final para examen

JDK proporciona herramientas de compilación y ejecución Java (javac, JVM, bibliotecas). IDEs modernos mejoran productividad con autocompletado, depuración, y refactoring. Git es esencial para control de versiones y colaboración profesional. La estandarización del entorno reduce onboarding y mejora productividad del equipo.





Sesión 2: Concepto de programa, datos, algoritmos y estructuras básicas

Un programa informático es mucho más que líneas de código en una pantalla. Es la materialización de una solución lógica a un problema específico, expresada en un lenguaje que la computadora puede interpretar y ejecutar. Cada programa transforma datos de entrada en resultados útiles siguiendo una secuencia de instrucciones precisas y deterministas.

Los datos son la materia prima de cualquier aplicación. Pueden ser números que representan temperaturas, cadenas de texto que contienen nombres de usuarios, imágenes digitales formadas por píxeles, o estructuras complejas que modelan entidades del mundo real. La forma en que organizas, almacenas y manipulas estos datos determina no solo la funcionalidad de tu aplicación, sino también su eficiencia y escalabilidad.

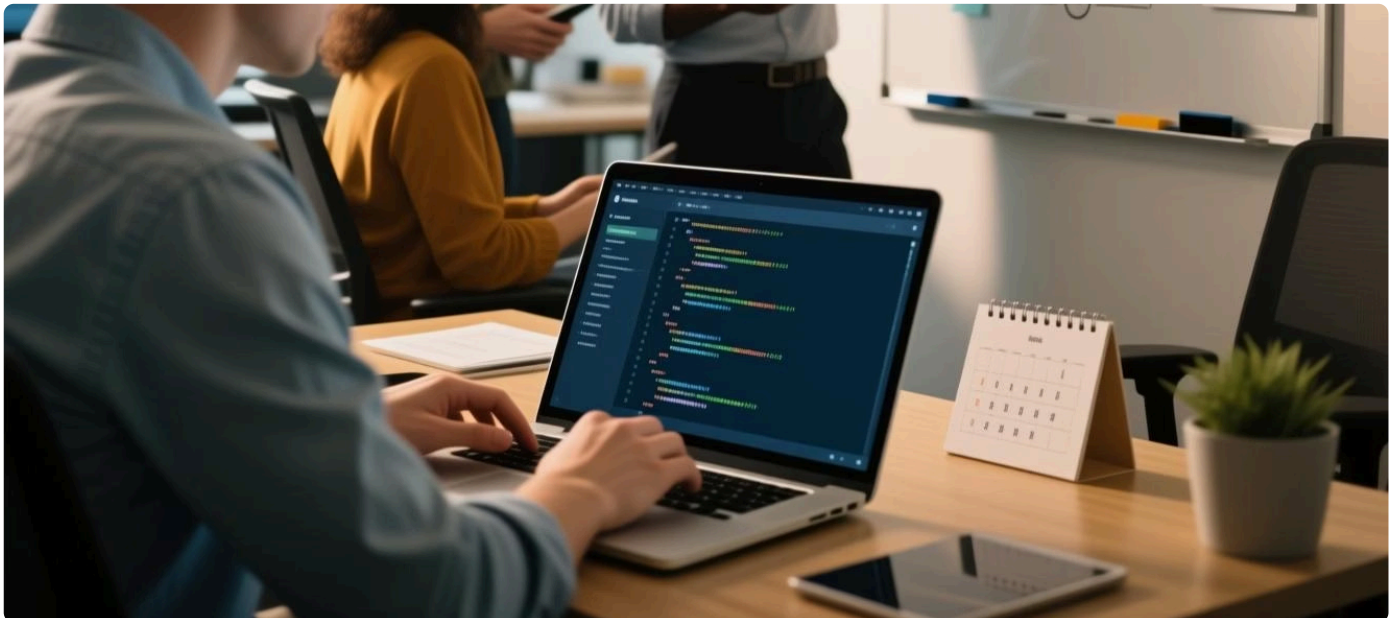
Un algoritmo es una secuencia lógica, finita y no ambigua de pasos diseñada para resolver un problema específico. Antes de escribir una sola línea de código, debes diseñar el algoritmo usando herramientas como pseudocódigo, diagramas de flujo, o simplemente pensamiento estructurado. Esta planificación previa es lo que diferencia a un programador profesional de alguien que simplemente "hace que funcione".

Las estructuras básicas de programación son los bloques fundamentales con los que construyes cualquier algoritmo: secuencia (instrucciones ejecutadas una tras otra), selección (decisiones basadas en condiciones), e iteración (repetición controlada de instrucciones). Según el teorema de Böhm-Jacopini, estas tres estructuras son suficientes para expresar cualquier algoritmo computable.

Esquema Visual



Este esquema ilustra cómo los conceptos fundamentales se interrelacionan. Un programa utiliza algoritmos para transformar datos en resultados, y estos algoritmos se construyen combinando las tres estructuras básicas. La comprensión de esta jerarquía conceptual es esencial para desarrollar pensamiento algorítmico sólido.





Caso de Estudio: Algoritmo de Recomendación de Amazon

Amazon procesa más de 12 petabytes de datos diarios para generar recomendaciones personalizadas que impulsan el 35% de sus ingresos totales. Su algoritmo de recomendación, aunque complejo en implementación, se basa en estructuras básicas de programación aplicadas a escala masiva.

El algoritmo sigue una estructura secuencial clara: primero recopila datos del usuario (historial de compras, tiempo en páginas, búsquedas realizadas), luego aplica estructuras de selección para categorizar el comportamiento (si compró libros, entonces analizar preferencias literarias), y finalmente utiliza iteración para comparar con patrones de usuarios similares.

La implementación utiliza las tres estructuras básicas de forma sofisticada: secuencias de filtros que procesan millones de productos, selecciones múltiples que evalúan decenas de factores de relevancia, e iteraciones que refinan recomendaciones en tiempo real. El resultado es un sistema que procesa 300 millones de productos y genera recomendaciones personalizadas para 200 millones de usuarios activos en menos de 100 milisegundos.

La clave del éxito no está en la complejidad del código, sino en la elegancia del diseño algorítmico: descomponer un problema complejo (recomendar productos relevantes) en estructuras básicas bien organizadas y optimizadas.



Herramientas y Consejos

Utiliza herramientas de diagramación como Lucidchart, Draw.io, o Miro para visualizar algoritmos antes de programar. La representación gráfica te ayuda a identificar problemas de lógica y optimizaciones potenciales antes de invertir tiempo en código.

Practica el pseudocódigo como paso intermedio entre la idea y el código. Escribir la lógica en lenguaje natural estructurado te permite concentrarte en la solución sin distraerte con sintaxis específica del lenguaje de programación.

Aplica la metodología "divide y vencerás": descompón problemas complejos en subproblemas más simples que puedas resolver con estructuras básicas. Esta aproximación no solo simplifica el desarrollo, sino que también facilita el testing y mantenimiento posterior.

Mitos y Realidades

✗ **Mito:** "Los buenos programadores no necesitan planificar, programan directamente"

→ **FALSO.** Los programadores expertos dedican entre 60-80% de su tiempo a análisis y diseño, y solo 20-40% a escribir código. Empresas como Google, Microsoft y Meta han demostrado que la planificación previa reduce el tiempo total de desarrollo entre 40-60% y disminuye bugs en producción hasta en 80%.

✗ **Mito:** "Los diagramas de flujo son obsoletos en programación moderna"

→ **FALSO.** Gigantes tecnológicos como Google, Amazon y Netflix siguen utilizando diagramas de flujo para documentar algoritmos críticos, comunicar lógica compleja entre equipos, y onboarding de nuevos desarrolladores. La visualización sigue siendo la forma más eficiente de comunicar lógica algorítmica compleja.

Resumen final para el examen

📌 Programa: instrucciones que transforman datos en resultados. Algoritmo: secuencia lógica, finita y no ambigua de pasos. Estructuras básicas: secuencia, selección, iteración (suficientes para cualquier algoritmo). Planificación previa reduce tiempo de desarrollo y bugs. Amazon ejemplifica aplicación exitosa a escala masiva.

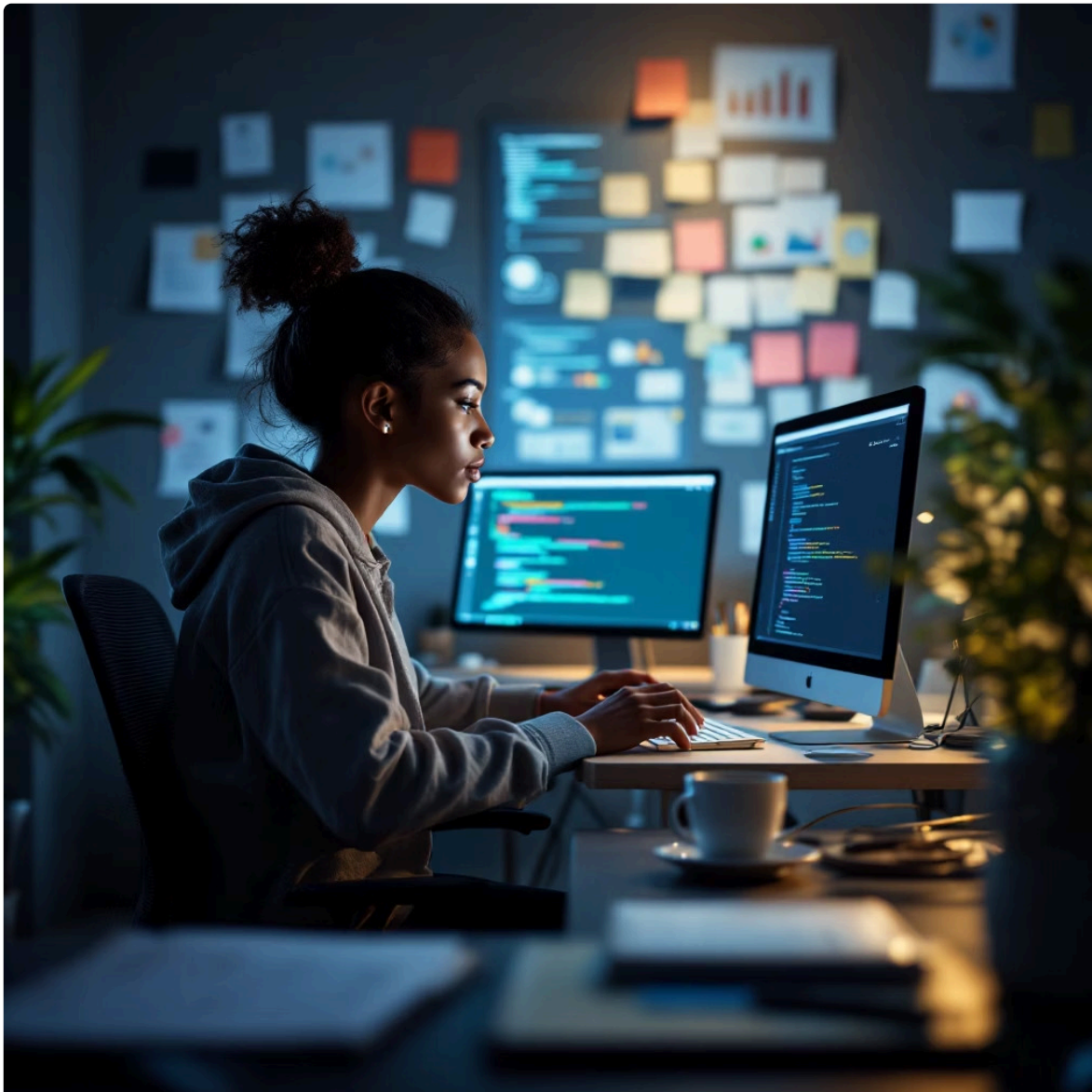
Sesión 3: Ejecución de programas en terminal e IDE. Uso inicial de Git

La ejecución de programas Java puede realizarse de dos formas fundamentales, cada una con ventajas específicas según el contexto profesional. El terminal te proporciona control granular sobre el proceso de compilación y ejecución, mientras que los IDEs automatizan estas tareas añadiendo funcionalidades avanzadas de desarrollo.

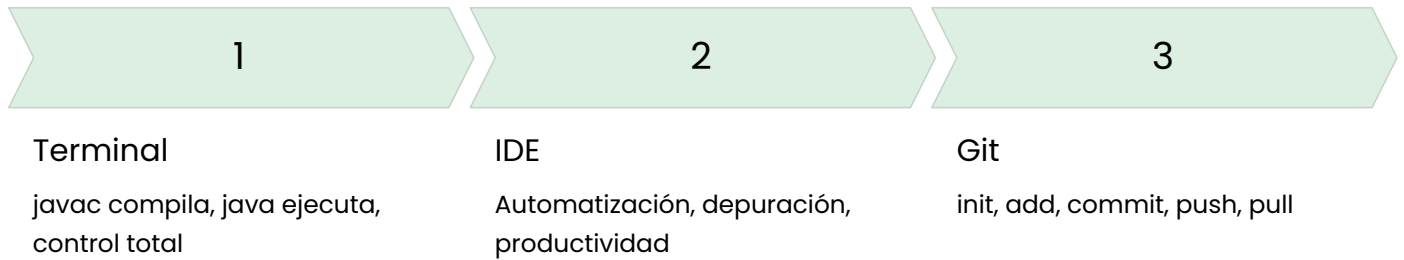
En el terminal, el proceso es explícito y educativo: utilizas `javac NombreClase.java` para compilar tu código fuente en bytecode, generando archivos `.class` que contiene las instrucciones que la JVM puede ejecutar. Posteriormente, `java NombreClase` ejecuta el programa cargando el bytecode en la máquina virtual. Este método es fundamental para entender qué ocurre realmente "bajo el capó" y es imprescindible en entornos de producción, servidores remotos, y sistemas de integración continua.

Los IDEs transforman este proceso manual en una experiencia fluida y productiva. Con un simple clic o atajo de teclado, el IDE compila, ejecuta, y presenta los resultados, pero además ofrece capacidades avanzadas: depuración paso a paso con puntos de interrupción, inspección de variables en tiempo real, refactoring automático, y detección de errores antes de la compilación.

Git, en este nivel inicial, se centra en cinco comandos fundamentales que cubren el 80% del uso diario: `git init` crea un repositorio, `git add` prepara cambios para commit, `git commit` guarda una versión, `git push` sube cambios al repositorio remoto, y `git pull` descarga actualizaciones. Dominar estos comandos básicos es suficiente para trabajar efectivamente en equipos de desarrollo.



Esquema visual



El diagrama muestra cómo estas herramientas se complementan en el flujo de trabajo diario. El terminal proporciona comprensión profunda y control preciso, el IDE acelera el desarrollo y facilita el debugging, y Git garantiza la integridad y colaboración del código.

Caso de Estudio: Netflix y Desarrollo Distribuido

Netflix opera con más de 2,500 desarrolladores distribuidos globalmente trabajando simultáneamente en cientos de microservicios que sirven contenido a 230+ millones de suscriptores. Su estrategia de desarrollo combina inteligentemente ejecución local en IDEs para desarrollo rápido y terminal para deployment en contenedores Docker.

Durante el desarrollo, cada ingeniero ejecuta pruebas localmente en IntelliJ IDEA o VSCode, aprovechando las capacidades de depuración para resolver problemas complejos rápidamente. Sin embargo, su pipeline de CI/CD (Integración Continua/Despliegue Continuo) utiliza exclusivamente terminal para compilar, testear, y desplegar automáticamente, procesando más de 4,000 deployments diarios.



Su estrategia Git es especialmente sofisticada: implementan "trunk-based development" donde todos los desarrolladores trabajan en la rama principal con commits pequeños y frecuentes. Realizan más de 4,000 commits diarios coordinados entre equipos en diferentes zonas horarias, utilizando herramientas automatizadas que ejecutan tests y validaciones antes de cada merge.

Esta metodología híbrida (IDEs para desarrollo, terminal para producción, Git para coordinación) les permite mantener velocidad de innovación sin sacrificar estabilidad: lanzan nuevas funcionalidades semanalmente mientras mantienen 99.9% de disponibilidad del servicio.

Herramientas y Consejos

Domina los atajos de teclado esenciales de tu IDE:
Ctrl+Shift+F10 para ejecutar en IntelliJ, F5 para debug, Ctrl+Shift+T para crear tests. Estos atajos pueden ahorrarte horas semanales y son indicadores de profesionalidad en entrevistas técnicas.

Configura alias de Git para comandos frecuentes: git config --global alias.st status permite usar git st en lugar de git status. Otros alias útiles incluyen git config --global alias.co checkout y git config --global alias.br branch.

Practica la ejecución en terminal regularmente, incluso si usas IDE diariamente. Esta habilidad es imprescindible para trabajar con servidores remotos, contenedores Docker, y sistemas de integración continua donde no hay interfaz gráfica disponible.

Mitos y Realidades

✗ Mito: "Solo necesito saber usar el IDE, el terminal es obsoleto"

→ **FALSO.** En entornos profesionales, trabajarás frecuentemente con servidores Linux sin interfaz gráfica, contenedores Docker, pipelines de CI/CD, y herramientas de deployment que requieren terminal. Según encuestas de Stack Overflow, 87% de desarrolladores profesionales usan terminal diariamente.

✗ Mito: "Git es demasiado complicado para empezar"

→ **FALSO.** Los cinco comandos básicos (init, add, commit, push, pull) son suficientes para ser productivo inmediatamente. La complejidad adicional (branches, merges, rebases) se aprende gradualmente según necesidades. GitHub reporta que el 70% de desarrolladores usan menos de 10 comandos Git regularmente.

Resumen final para el examen

Resumen final

Terminal: javac compila, java ejecuta, control total del proceso. IDE: automatización, depuración, productividad mejorada. Git básico: init, add, commit, push, pull cubren % uso diario. Netflix ejemplifica uso profesional híbrido con , commits diarios. Ambos métodos son complementarios y necesarios.

