// preview del temario

Aprende Java de verdad, este verano.

Preview gratuita del temario completo · Acceso anticipado a módulos 01, 02, 03 y 08 con ejemplos reales · El resto de módulos se desbloquean al reservar plaza.

4 Niveles	3 Modalidades	11 Módulos	100% Práctico

// temario completo

Para todos los niveles

Los temas marcados con ✓ incluyen ejemplos en esta preview. El resto muestra los contenidos pero los ejemplos se desbloquean al reservar.

BÁSICO

01 · Fundamentos de Java

- ■ Tipos primitivos (int, double, boolean, char)
- ■ String y concatenación
- ■ Operadores aritméticos y lógicos
- ■ Control de flujo: if / else
- ■ Bucles: for, while, do-while
- ■ Arrays unidimensionales y bidimensionales
- ■ Métodos: parámetros y retorno
- ■ Scope y paso por valor

```java
// Tipos primitivos — los más usados
int edad = 25; // entero (32 bits)
double precio = 9.99; // decimal (64 bits)
boolean activo = true; // true / false
char letra = 'A'; // un solo carácter

// String — no es primitivo, es un objeto
String nombre = "Ana García";
String saludo = "Hola, " + nombre;

// Control de flujo — if / else
if (edad >= 18) {
System.out.println("Mayor de edad");
} else {
System.out.println("Menor de edad");
}

// ... resto de fundamentos en la clase completa ■
```

BÁSICO

02 · Orientación a Objetos

- Clases y objetos: atributos y métodos
- Encapsulación: private + getters/setters
- Herencia: extends y @Override
- Polimorfismo en tiempo de ejecución
- Interfaces y clases abstractas
- Constructores y this
- equals(), hashCode() y toString()

```
// Encapsulación — atributos privados + getters/setters
public class Persona {
private String nombre;
private int edad;

public Persona(String nombre, int edad) {
this.nombre = nombre;
this.edad = edad;
}

public String getNombre() { return nombre; }
public void setEdad(int e) { this.edad = e; }
}

// Herencia — subclase extiende comportamiento
public class Estudiante extends Persona {
private String carrera;

@Override
public String toString() {
return getNombre() + " estudia " + carrera;
}
}

// ... Polimorfismo e Interfaces en la clase completa ■
```

INTERMEDIO

03 · Colecciones y Generics

- Genéricos: clases y métodos parametrizados
- Type safety en tiempo de compilación
- Uso con List, Set, Map
- ArrayList vs LinkedList vs ArrayDeque
- HashSet vs TreeSet vs LinkedHashSet
- HashMap: put, get, entrySet, merge
- Bounded wildcards: ,

```
// Generics — el compilador protege el tipo
public class Caja<T> {
private T contenido;

public void guardar(T item) { this.contenido = item; }
public T obtener() { return contenido; }
}

// Uso: la misma clase, distintos tipos sin castear
Caja<String> c1 = new Caja<>() ;
c1.guardar("Hola");

Caja<Integer> c2 = new Caja<>() ;
c2.guardar(42);

// ... List, Set, Map y Wildcards en la clase completa ■
```

INTERMEDIO

04 · SOLID y Clean Code

- Single Responsibility Principle (SRP)
- Open/Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation (ISP)
- Dependency Inversion (DIP)
- Refactoring y code smells
- Naming y legibilidad

■ *Ejemplos y ejercicios disponibles al reservar plaza*

INTERMEDIO

05 · Control de Versiones con Git

- Repositorios, commits y staging
- Ramas y estrategias de branching
- Merge vs Rebase
- Resolución de conflictos
- GitHub: PRs y code review
- Flujos de trabajo profesionales

■ *Ejemplos y ejercicios disponibles al reservar plaza*

AVANZADO

06 · Java Funcional

- Lambdas y expresiones funcionales
- Stream API: filter, map, reduce, collect
- Interfaces funcionales: Predicate, Function, Consumer
- Optional para evitar NullPointerException
- Method references
- Collectors avanzados

Ejemplos y ejercicios disponibles al reservar plaza

AVANZADO

07 · Testing con JUnit 5

- TDD: Red–Green–Refactor
- JUnit 5: @Test, @BeforeEach, @ParameterizedTest
- Mockito: mocks, stubs y spies
- AssertJ: aserciones fluidas
- Test doubles y cobertura
- Testing de servicios con dependencias

Ejemplos y ejercicios disponibles al reservar plaza

EXPERTO

08 · Patrones de Diseño

- Factory Method: creación desacoplada
- Interfaces como contrato entre capas
- Switch expressions (Java 14+)
- Extensibilidad sin tocar el cliente
- Strategy — comportamiento intercambiable en tiempo de ejecución
- Observer — eventos y suscriptores desacoplados
- Builder — construcción fluida de objetos complejos
- Decorator — extensión sin herencia
- Singleton — instancia única garantizada

```
// Contexto real: una app necesita notificar por distintos canales
// Sin Factory cada parte del código sabe cómo construir cada Notificador → acoplamiento

// 1. Interfaz común
public interface Notificador {
void enviar(String mensaje);
}

// 2. Implementaciones concretas
public class EmailNotificador implements Notificador {
@Override public void enviar(String msg) {
System.out.println("■ Email: " + msg);
}}

public class SMSNotificador implements Notificador {
@Override public void enviar(String msg) {
System.out.println("■ SMS: " + msg);
}}

// 3. La Factory — un único lugar que sabe cómo construir
public class NotificadorFactory {
public static Notificador crear(String tipo) {
return switch (tipo) {
case "email" -> new EmailNotificador();
case "sms" -> new SMSNotificador();
default -> throw new IllegalArgumentException(tipo);
};
}}

// 4. El cliente no sabe nada del tipo concreto
Notificador n = NotificadorFactory.crear("email");
n.enviar("Reserva confirmada");

// ... Strategy, Observer, Builder en la clase completa ■
```

EXPERTO

09 · Algoritmos y Estructuras de Datos

- Complejidad temporal y espacial: Big O
- Sorting: QuickSort, MergeSort, HeapSort
- Búsqueda binaria e iterativa
- Pilas, colas y listas enlazadas
- Árboles BST y recorridos
- Grafos: BFS, DFS
- Dynamic Programming

■ *Ejemplos y ejercicios disponibles al reservar plaza*

EXPERTO

10 · Arquitectura de Software

- MVC y separación de responsabilidades
- Arquitectura Hexagonal (Ports & Adapters)
- Value Objects e Immutability
- Introducción a Microservicios
- API REST con Spring Boot
- Gestión de dependencias con Maven

Ejemplos y ejercicios disponibles al reservar plaza

PROYECTO

11 · Aplicación Real Completa

- Diseño del dominio desde cero
- Arquitectura por capas (domain / application / infra)
- API REST funcional
- Base de datos con H2 / PostgreSQL
- Testing de integración
- Deploy y presentación del proyecto

Ejemplos y ejercicios disponibles al reservar plaza

Los módulos 04–11 están disponibles al reservar plaza. Incluyen ejemplos completos, ejercicios guiados y código real comentado.

// retos de práctica

Pon a prueba lo que has aprendido

Dos retos al estilo Codewars. El básico aparece resuelto para que puedas analizarlo; el intermedio está para que lo intentes tú.

Reto 1 · Resuelto

● BÁSICO

Escribe un método `esPalindromo(String s)` que devuelva `true` si la cadena se lee igual al derecho que al revés (ignorando mayúsculas).

Ejemplos: "Radar" → true · "Java" → false · "Ana" → true

```
public static boolean esPalindromo(String s) {
String limpia = s.toLowerCase();
String inversa = new StringBuilder(limpia).
reverse().toString();
return limpia.equals(inversa);
}

// Test rápido
System.out.println(esPalindromo("Radar")) ; // true
System.out.println(esPalindromo("Java")); // false
```

Pistas usadas: String.toLowerCase(), StringBuilder.reverse(), String.equals()

Reto 2 · Para resolver

◆ INTERMEDIO

Dado un `List<String>` de palabras, devuelve un `Map<Character, Long>` con la frecuencia de la **primera letra** de cada palabra (ignorando mayúsculas). Usa Stream API.

Ejemplo: ["Ana","Ángel","Bruno","Beatriz","Carlos"] → {a=2, b=2, c=1}

```java
public static Map<Character, Long>
frecuenciaPrimeraLetra(List<String> palabras) {

// TODO: implementa con palabras.stream()
// usa .map(), .collect() y Collectors.groupingBy()
// convierte la primera letra a minúscula antes de agrupar

return null; // reemplaza esto
}
```

■ Pistas

1. `s.toLowerCase().charAt(0)` — primera letra en minúscula de cada String.
2. `Collectors.groupingBy(fn, Collectors.counting())` — agrupa y cuenta en un solo paso.
3. El lambda de groupingBy recibe cada String del stream: `s -> s.toLowerCase().charAt(0)`.

La solución completa y variantes adicionales se trabajan en el Módulo 06 · Java Funcional ■