Referencia Rápida de Java

POO, colecciones, streams, manejo de excepciones

Básico

Hola Mundo

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

Compilar y ejecutar

```
javac Main.java      # compile
java Main             # run
java Main.java        # single-file (Java 11+)
```

Convenciones de nomenclatura

`ClassName`	PascalCase para clases e interfaces
`methodName`	camelCase para métodos y variables
`CONSTANT_NAME`	UPPER_SNAKE para constantes
`com.example.pkg`	Dominio invertido en minúsculas para paquetes

Tipos de datos

Primitivos

`byte`	8 bits con signo (-128 a 127)
`short`	16 bits con signo
`int`	32 bits con signo (entero por defecto)
`long`	64 bits con signo (sufijo **`L`**)
`float`	IEEE-754 de 32 bits (sufijo **`f`**)
`double`	IEEE-754 de 64 bits (decimal por defecto)
`boolean`	**`true`** / **`false`**
`char`	Carácter Unicode de 16 bits

Strings

```
String s = "hello";
String joined = s + " world";      // concatenation
int len = s.length();
String sub = s.substring(0, 3);    // "hel"
boolean eq = s.equals("hello");    // content equality
```

Conversión de tipos

```
int i = (int) 3.14;               // narrowing cast
double d = i;                      // widening (auto)
int n = Integer.parseInt("42");    // string to int
String s = String.valueOf(42);     // int to string
```

Arrays

```
int[] nums = {1, 2, 3};
String[] names = new String[5];
int[][] matrix = new int[3][4];
Arrays.sort(nums);
```

Flujo de control

If / Else

```
if (x > 0) {
    System.out.println("positive");
} else if (x == 0) {
    System.out.println("zero");
} else {
    System.out.println("negative");
}
```

Switch

```
// Traditional
switch (day) {
    case "Mon": doWork(); break;
    case "Sat": case "Sun": rest(); break;
    default: routine();
}
// Switch expression (Java 14+)
String type = switch (day) {
    case "Sat", "Sun" -> "weekend";
    default -> "weekday";
};
```

Bucles

```
for (int i = 0; i < 10; i++) { }
for (String s : list) { }         // enhanced for
while (condition) { }
do { } while (condition);
```

Métodos

Definición

```
public static int add(int a, int b) {
    return a + b;
}
```

Varargs y sobrecarga

```
static int sum(int... nums) {
    int total = 0;
    for (int n : nums) total += n;
    return total;
}
// sum(1, 2)  sum(1, 2, 3)  both work
```

Modificadores de acceso

`public`	Accesible desde cualquier lugar
`protected`	Mismo paquete + subclases
`(default)`	Solo mismo paquete (sin palabra clave)
`private`	Solo la misma clase

Clases y objetos

Definición de clase

```
public class User {
    private String name;
    private int age;
    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() { return name; }
}
```

Records (Java 16+)

```
public record Point(double x, double y) {
    // auto: constructor, getters, equals, hashCode, toString
    public double distance() {
        return Math.sqrt(x * x + y * y);
    }
}
```

Static y Final

`static`	Pertenece a la clase, no a la instancia
`final field`	No puede reasignarse después de la inicialización
`final method`	No puede ser sobreescrito
`final class`	No puede ser heredado

Herencia

Extends

```
public class Animal {
    public void speak() { System.out.println("..."); }
}
public class Dog extends Animal {
    @Override
    public void speak() { System.out.println("Woof!"); }
}
```

Clases abstractas

```
public abstract class Shape {
    abstract double area();
    public void describe() {
        System.out.println("Area: " + area());
    }
}
```

Conceptos clave

`super`	Llamar al constructor o método padre
`@Override`	Verificación de sobreescritura en tiempo de compilación
`instanceof`	Verificación de tipo en tiempo de ejecución
`sealed (17+)`	Restringir qué clases pueden extender

Interfaces

Definición

```
public interface Printable {
    void print();                    // abstract
    default String format() {        // default method
        return toString();
    }
    static Printable of(String s) {  // static method
        return () -> System.out.println(s);
    }
}
```

Implementación

```
public class Report implements Printable, Serializable {
    @Override
    public void print() {
        System.out.println("Report");
    }
}
```

Interfaces funcionales

`Runnable`	**`() -> void`**
`Supplier<T>`	**`() -> T`**
`Consumer<T>`	**`T -> void`**
`Function<T,R>`	**`T -> R`**
`Predicate<T>`	**`T -> boolean`**
`Comparator<T>`	**`(T, T) -> int`**

Colecciones

List

```
List<String> list = new ArrayList<>();
list.add("a");
list.get(0);                  // "a"
list.size();                  // 1
List<String> immutable = List.of("a", "b", "c");
```

Referencia Rápida de Java

Map

```
Map<String, Integer> map = new HashMap<>();
map.put("key", 42);
map.getOrDefault("key", 0);   // 42
map.containsKey("key");       // true
map.forEach((k, v) -> { });
```

Set

```
Set<String> set = new HashSet<>();
set.add("a");
set.contains("a");            // true
Set<String> immutable = Set.of("a", "b", "c");
```

Implementaciones comunes

`ArrayList`	Array redimensionable, acceso aleatorio rápido
`LinkedList`	Doblemente enlazada, inserción/eliminación rápida
`HashMap`	Tabla hash, get/put O(1)
`TreeMap`	Ordenado por clave, O(log n)
`HashSet`	Elementos únicos, búsqueda O(1)
`LinkedHashMap`	HashMap con orden de inserción

Manejo de excepciones

Try / Catch / Finally

```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.err.println(e.getMessage());
} finally {
    // always executes
}
```

Try con recursos

```
try (var reader = new BufferedReader(new FileReader(path))) {
    String line = reader.readLine();
}   // auto-closes reader
```

Jerarquía de excepciones

`Throwable`	Raíz de todos los errores y excepciones
`Error`	Problemas graves (OutOfMemoryError)
`Exception`	Excepciones verificadas (deben manejarse)
`RuntimeException`	No verificadas (NullPointer, IndexOutOfBounds)

Excepción personalizada

```
public class AppException extends Exception {
    public AppException(String msg) { super(msg); }
    public AppException(String msg, Throwable cause) {
        super(msg, cause);
    }
}
```

Streams y Lambdas

Sintaxis lambda

```
Comparator<String> byLen = (a, b) -> a.length() - b.length();
Runnable task = () -> System.out.println("run");
Function<String, Integer> len = String::length;  // method ref
```

Pipeline de stream

```
List<String> result = names.stream()
    .filter(n -> n.length() > 3)
    .map(String::toUpperCase)
    .sorted()
    .collect(Collectors.toList());
```

Operaciones comunes de stream

`.filter(pred)`	Mantener elementos que coinciden con el predicado
`.map(func)`	Transformar cada elemento
`.flatMap(func)`	Mapear y aplanar streams anidados
`.sorted()`	Ordenar (natural o con Comparator)
`.distinct()`	Eliminar duplicados
`.limit(n)`	Tomar los primeros n elementos
`.collect()`	Terminal: reunir en colección
`.forEach()`	Terminal: ejecutar acción en cada elemento
`.reduce()`	Terminal: combinar en un único valor
`.count()`	Terminal: contar elementos

Genéricos

Clase y método genérico

```
public class Box<T> {
    private T value;
    public Box(T value) { this.value = value; }
    public T get() { return value; }
}
public static <T> List<T> listOf(T... items) {
    return List.of(items);
}
```

Tipos acotados y wildcards

`<T extends Number>`	T debe ser Number o subclase
`<T extends A & B>`	Múltiples límites (clase + interfaces)
`<?>`	Tipo desconocido (solo lectura)
`<? extends T>`	Wildcard con límite superior (productor)
`<? super T>`	Wildcard con límite inferior (consumidor)

Optional y Java moderno

Optional

```
Optional<String> opt = Optional.ofNullable(getValue());
String result = opt.orElse("default");
opt.ifPresent(v -> System.out.println(v));
String upper = opt.map(String::toUpperCase).orElse("");
```

Text Blocks (Java 15+)

```
String json = """
       { "name": "Alice", "age": 30 }
       """;
```

Utilidades modernas

`var (10+)`	Inferencia de tipo para variables locales
`record (16+)`	Clase portadora de datos inmutable
`sealed (17+)`	Jerarquías de clases restringidas
`pattern matching (21+)`	**`instanceof`** con conversión automática
`virtual threads (21+)`	Hilos ligeros mediante **`Thread.ofVirtual()`**