

JAVA RIFERIMENTO RAPIDO

OOP, collezioni, stream, gestione delle eccezioni

Basi

Hello World

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

Compilazione ed Esecuzione

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

Convenzioni di Nomenclatura

ClassName PascalCase per classi e interfacce
methodName camelCase per metodi e variabili
CONSTANT_NAME UPPER_SNAKE per le costanti
com.example.pkg Dominio inverso minuscolo per i package

Tipi di Dato

Primitivi

byte 8 bit con segno (-128 a 127)
short 16 bit con segno
int 32 bit con segno (intero predefinito)
long 64 bit con segno (suffisso 'L')
float IEEE-754 32 bit (suffisso 'f')
double IEEE-754 64 bit (decimale predefinito)
boolean 'true' / 'false'
char Carattere Unicode 16 bit

Stringhe

```
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
```

Casting di Tipo

```
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
```

Array

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

Controllo del Flusso

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";
};
```

Cicli

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

Metodi

Definizione

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

Varargs e Overloading

```
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
```

Modificatori di Accesso

public Accessibile da qualsiasi punto
protected Stesso package + sottoclassi
(default) Solo stesso package (nessuna parola chiave)
private Solo stessa classe

Classi e Oggetti

Definizione di Classe

```
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; }
}
```

Record (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 e Final

static Appartiene alla classe, non all'istanza
final field Non può essere riassegnato dopo l'iniz
final method Non può essere sovrascritto
final class Non può essere sottoclassato

Ereditarietà

Extends

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

Classi Astratte

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

Concetti Chiave

super Chiama il costruttore o il metodo del genitore
@Override Controllo di override a tempo di compilazione
instanceof Controllo del tipo a runtime
sealed (17+) Limita quali classi possono estendere

Interfacce

Definizione

```
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);
    }
}
```

Implementazione

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

Interfacce Funzionali

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

Collezioni

List

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

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");
```

Implementazioni Comuni

ArrayList Array ridimensionabile, accesso casuale veloce
LinkedList Doppia mente collegata, inserzione/rimozione veloce
HashMap Tabella hash, get/put O(1)
TreeMap Ordinata per chiave, O(log n)
HashSet Elementi unici, ricerca O(1)
LinkedHashMap HashMap con ordine di inserzione

Gestione delle Eccezioni

Try / Catch / Finally

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

Try-with-Resources

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

Gerarchia delle Eccezioni

Throwable Radice di tutti gli errori ed eccezioni
Error Problemi gravi (OutOfMemoryError)
Exception Eccezioni checked (da gestire)

RuntimeException Unchecked (NullPointerException, IndexOutOfBoundsException)

Eccezione Personalizzata

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

Stream e Lambda

Sintassi 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 Stream

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

Operazioni Stream Comuni

.filter(pred) Mantiene gli elementi che soddisfano il predicato
.map(func) Trasforma ogni elemento
.flatMap(func) Mappa e appiattisce stream annidati
.sorted() Ordina (naturale o con Comparator)
.distinct() Rimuove i duplicati
.limit(n) Prende i primi n elementi
.collect() Terminale: raccoglie in una collezione
.forEach() Terminale: esegue un'azione su ogni elemento
.reduce() Terminale: combina in un singolo valore
.count() Terminale: conta gli elementi

Generici

Classe e Metodo Generico

```
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);
}
```

Tipi Limitati e Wildcard

<T extends Number> T deve essere Number o sottoclasse
<T extends A & B> Limiti multipli (classe + interfacce)
<?> Tipo sconosciuto (sola lettura)
<? extends T> Wildcard con limite superiore (produttore)
<? super T> Wildcard con limite inferiore (consumatore)

Optional e 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 Block (Java 15+)

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

Utilità Utili

var (10+) Inferenza del tipo per variabili locali
record (16+) Classe portadati immutabile
sealed (17+) Gerarchie di classi ristrette
pattern matching (21+) 'instanceof' con cast automatico
virtual threads (21+) Thread leggeri tramite Thread.ofVirtual()