Java Programming Cheat Sheet

Table of Contents

Basic Syntax

Hello World Program

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

Comments

java

```java
// Single line comment

/*
 * Multi-line comment
 * Multiple lines
 */

/**
 * JavaDoc comment
 * @param parameter description
 * @return return value description
 */
```

Package Declaration

java

```java
package com.company.projectname;
import java.util.*;
import java.io.File;
```

Data Types

Primitive Data Types

Type	Size	Range	Default
`byte`	8 bits	-128 to 127	0
`short`	16 bits	-32,768 to 32,767	0
`int`	32 bits	-2^{31} to $2^{31}-1$	0
`long`	64 bits	-2^{63} to $2^{63}-1$	0L
`float`	32 bits	±3.4E±38	0.0f
`double`	64 bits	±1.7E±308	0.0d
`char`	16 bits	0 to 65,535	'\u0000'
`boolean`	1 bit	true/false	false

Reference Data Types

java

```java
String text = "Hello";
int[] numbers = {1, 2, 3};
List<String> list = new ArrayList<>();
```

Variables

Declaration and Initialization

java

```java
// Declaration
int age;
String name;

// Initialization
age = 25;
name = "John";

// Declaration + Initialization
int score = 100;
final double PI = 3.14159; // Constant
```

Variable Scope

java

```java
public class ScopeExample {
    static int classVariable = 10;    // Class scope
    int instanceVariable = 20;       // Instance scope

    public void method() {
        int localVariable = 30;      // Method scope

        for (int i = 0; i < 5; i++) {
            int blockVariable = 40;  // Block scope
        }
    }
}
```

Operators

Arithmetic Operators

```java
int a = 10, b = 3;
int sum = a + b;        // Addition: 13
int diff = a - b;       // Subtraction: 7
int product = a * b;    // Multiplication: 30
int quotient = a / b;   // Division: 3
int remainder = a % b;  // Modulus: 1

// Increment/Decrement
a++;  // Post-increment
++a;  // Pre-increment
a--;  // Post-decrement
--a;  // Pre-decrement
```

Comparison Operators

```java
boolean result;
result = (a == b);  // Equal to
result = (a != b);  // Not equal to
result = (a > b);   // Greater than
result = (a < b);   // Less than
result = (a >= b);  // Greater than or equal
result = (a <= b);  // Less than or equal
```

Logical Operators

```java
boolean x = true, y = false;
boolean and = x && y;   // Logical AND
boolean or = x || y;    // Logical OR
boolean not = !x;       // Logical NOT
```

Assignment Operators

```java
int x = 10;
x += 5;  // x = x + 5
x -= 3;  // x = x - 3
x *= 2;  // x = x * 2
x /= 4;  // x = x / 4
x %= 3;  // x = x % 3
```

Control Structures

Conditional Statements

```java
// if-else
if (condition) {
    // code
} else if (anotherCondition) {
    // code
} else {
    // code
}

// Ternary operator
int result = (condition) ? valueIfTrue : valueIfFalse;

// switch statement
switch (variable) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code
        break;
}
```

Loops

```java
// for loop
for (int i = 0; i < 10; i++) {
    // code
}

// Enhanced for loop (for-each)
int[] array = {1, 2, 3, 4, 5};
for (int element : array) {
    System.out.println(element);
}

// while loop
while (condition) {
    // code
}

// do-while loop
do {
    // code
} while (condition);
```

Loop Control

```java
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;    // Exit loop
    }
    if (i == 2) {
        continue; // Skip current iteration
    }
    System.out.println(i);
}
```

Methods

Method Declaration

```java
// Syntax: [access modifier] [static] returnType methodName(parameters)
public static int add(int a, int b) {
    return a + b;
}

// Method overloading
public static double add(double a, double b) {
    return a + b;
}

// Void method
public void printMessage(String message) {
    System.out.println(message);
}
```

Method Parameters

```java
// Pass by value
public void changeValue(int x) {
    x = 100; // Original value unchanged
}

// Variable arguments (varargs)
public void printNumbers(int... numbers) {
    for (int num : numbers) {
        System.out.println(num);
    }
}
```

Classes and Objects

Class Definition

java

```java
public class Person {
    // Instance variables
    private String name;
    private int age;

    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Default constructor
    public Person() {
        this("Unknown", 0);
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // Setter methods
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        if (age >= 0) {
            this.age = age;
        }
    }

    // toString method
    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}
```

Object Creation and Usage

```java
// Creating objects
Person person1 = new Person("Alice", 25);
Person person2 = new Person();

// Using objects
person1.setAge(26);
System.out.println(person1.getName());
System.out.println(person1.toString());
```

Static Members

```java
public class Counter {
    private static int count = 0;  // Static variable

    public static void increment() {  // Static method
        count++;
    }

    public static int getCount() {
        return count;
    }
}

// Usage
Counter.increment();
System.out.println(Counter.getCount());
```

Arrays

Array Declaration and Initialization

java

```java
// Declaration
int[] numbers;
String[] names;

// Initialization
numbers = new int[5];            // Array of size 5
names = new String[]{"A", "B"}; // Array literal

// Combined declaration and initialization
int[] scores = {90, 85, 78, 92, 88};
char[] letters = new char[10];
```

Array Operations

java

```java
int[] arr = {1, 2, 3, 4, 5};

// Access elements
int first = arr[0];
int last = arr[arr.length - 1];

// Modify elements
arr[2] = 10;

// Iterate through array
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}

// Enhanced for loop
for (int element : arr) {
    System.out.println(element);
}
```

Multidimensional Arrays

```java
// 2D array
int[][] matrix = new int[3][4];
int[][] table = {{1, 2}, {3, 4}, {5, 6}};

// Access elements
matrix[0][1] = 5;
int value = table[1][0];

// Iterate 2D array
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
}
```

Collections

ArrayList

```java
import java.util.ArrayList;
import java.util.List;

List<String> list = new ArrayList<>();

// Add elements
list.add("Apple");
list.add("Banana");
list.add(0, "Orange"); // Insert at index

// Access elements
String first = list.get(0);
int size = list.size();

// Remove elements
list.remove("Banana");
list.remove(0);

// Check if exists
boolean exists = list.contains("Apple");

// Iterate
for (String item : list) {
    System.out.println(item);
}
```

HashMap

```java
import java.util.HashMap;
import java.util.Map;

Map<String, Integer> map = new HashMap<>();

// Add key-value pairs
map.put("apple", 5);
map.put("banana", 3);

// Get value
Integer count = map.get("apple");

// Check if key exists
boolean hasKey = map.containsKey("banana");

// Remove entry
map.remove("apple");

// Iterate
for (Map.Entry<String, Integer> entry : map.entrySet()) {
    System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

HashSet

java

```java
import java.util.HashSet;
import java.util.Set;

Set<String> set = new HashSet<>();

// Add elements
set.add("Java");
set.add("Python");
set.add("Java"); // Duplicate ignored

// Check if exists
boolean exists = set.contains("Java");

// Remove element
set.remove("Python");

// Size
int size = set.size();
```

Exception Handling

Try-Catch-Finally

java

```java
try {
    // Code that might throw exception
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Division by zero: " + e.getMessage());
} catch (Exception e) {
    System.out.println("General exception: " + e.getMessage());
} finally {
    System.out.println("This always executes");
}
```

Throwing Exceptions

```java
public void validateAge(int age) throws IllegalArgumentException {
    if (age < 0) {
        throw new IllegalArgumentException("Age cannot be negative");
    }
}

// Custom exception
public class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
```

Try-with-Resources

```java
try (FileReader file = new FileReader("file.txt");
     BufferedReader buffer = new BufferedReader(file)) {

    String line = buffer.readLine();
    System.out.println(line);

} catch (IOException e) {
    System.out.println("Error reading file: " + e.getMessage());
}
```

Input/Output

Console Input/Output

java

```java
import java.util.Scanner;

// Output
System.out.println("Hello");           // Print with newline
System.out.print("Hello");             // Print without newline
System.out.printf("Number: %d%n", 42); // Formatted output

// Input
Scanner scanner = new Scanner(System.in);
System.out.print("Enter name: ");
String name = scanner.nextLine();

System.out.print("Enter age: ");
int age = scanner.nextInt();

scanner.close();
```

File I/O

```java
import java.io.*;
import java.nio.file.*;

// Write to file
try {
    Files.write(Paths.get("output.txt"),
                "Hello, World!".getBytes());
} catch (IOException e) {
    e.printStackTrace();
}

// Read from file
try {
    String content = Files.readString(Paths.get("input.txt"));
    System.out.println(content);
} catch (IOException e) {
    e.printStackTrace();
}

// Read file line by line
try (BufferedReader reader = Files.newBufferedReader(Paths.get("file.txt"))) {
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

String Operations

String Methods

```java
String str = "Hello, World!";

// Length and characters
int length = str.length();
char ch = str.charAt(0);

// Substrings
String sub1 = str.substring(7);      // "World!"
String sub2 = str.substring(0, 5);   // "Hello"

// Case operations
String upper = str.toUpperCase();
String lower = str.toLowerCase();

// Searching
int index = str.indexOf("World");
boolean contains = str.contains("Hello");
boolean starts = str.startsWith("Hello");
boolean ends = str.endsWith("!");

// Replacing
String replaced = str.replace("World", "Java");

// Splitting
String[] words = str.split(", ");

// Trimming
String trimmed = "  Hello  ".trim();

// Comparison
boolean equal = str.equals("Hello, World!");
boolean equalIgnoreCase = str.equalsIgnoreCase("hello, world!");
```

StringBuilder

```java
StringBuilder sb = new StringBuilder();

sb.append("Hello");
sb.append(" ");
sb.append("World");
sb.insert(5, ",");
sb.delete(5, 6);
sb.reverse();

String result = sb.toString();
```

Inheritance

Basic Inheritance

java

```java
// Parent class
public class Animal {
    protected String name;

    public Animal(String name) {
        this.name = name;
    }

    public void eat() {
        System.out.println(name + " is eating");
    }

    public void sleep() {
        System.out.println(name + " is sleeping");
    }
}

// Child class
public class Dog extends Animal {
    private String breed;

    public Dog(String name, String breed) {
        super(name); // Call parent constructor
        this.breed = breed;
    }

    @Override
    public void eat() {
        System.out.println(name + " the dog is eating dog food");
    }

    public void bark() {
        System.out.println(name + " is barking");
    }
}
```

Abstract Classes

java

```java
public abstract class Shape {
    protected double width, height;

    public Shape(double width, double height) {
        this.width = width;
        this.height = height;
    }

    // Abstract method
    public abstract double calculateArea();

    // Concrete method
    public void display() {
        System.out.println("Shape with area: " + calculateArea());
    }
}

public class Rectangle extends Shape {
    public Rectangle(double width, double height) {
        super(width, height);
    }

    @Override
    public double calculateArea() {
        return width * height;
    }
}
```

Interfaces

Interface Definition and Implementation

java

```java
public interface Drawable {
    // Constants (public static final by default)
    int MAX_SIZE = 100;

    // Abstract methods (public abstract by default)
    void draw();
    void resize(double factor);

    // Default method (Java 8+)
    default void print() {
        System.out.println("Drawing shape");
    }

    // Static method (Java 8+)
    static void info() {
        System.out.println("Drawable interface");
    }
}

public class Circle implements Drawable {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public void draw() {
        System.out.println("Drawing circle with radius " + radius);
    }

    @Override
    public void resize(double factor) {
        radius *= factor;
    }
}
```

Multiple Interfaces

java

```java
public interface Movable {
    void move(int x, int y);
}

public class MovableCircle extends Circle implements Drawable, Movable {
    private int x, y;

    public MovableCircle(double radius, int x, int y) {
        super(radius);
        this.x = x;
        this.y = y;
    }

    @Override
    public void move(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Common Libraries

Math Operations

java

```java
import java.lang.Math;

double result;
result = Math.abs(-5);          // Absolute value: 5
result = Math.max(10, 20);      // Maximum: 20
result = Math.min(10, 20);      // Minimum: 10
result = Math.pow(2, 3);        // Power: 8.0
result = Math.sqrt(16);         // Square root: 4.0
result = Math.random();         // Random 0.0-1.0

// Rounding
result = Math.ceil(4.3);        // Ceiling: 5.0
result = Math.floor(4.7);       // Floor: 4.0
result = Math.round(4.6);       // Round: 5
```

Date and Time (Java 8+)

```java
import java.time.*;
import java.time.format.DateTimeFormatter;

// Current date and time
LocalDate today = LocalDate.now();
LocalTime now = LocalTime.now();
LocalDateTime dateTime = LocalDateTime.now();

// Create specific date/time
LocalDate birthday = LocalDate.of(1990, 5, 15);
LocalTime meeting = LocalTime.of(14, 30);

// Formatting
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
String formatted = dateTime.format(formatter);

// Parsing
LocalDate parsed = LocalDate.parse("2023-12-25");
```

Random Numbers

```java
import java.util.Random;

Random random = new Random();

int randomInt = random.nextInt();           // Any integer
int boundedInt = random.nextInt(100);       // 0 to 99
double randomDouble = random.nextDouble();  // 0.0 to 1.0
boolean randomBoolean = random.nextBoolean();
```

Quick Reference

Access Modifiers

- `public`: Accessible everywhere
- `protected`: Accessible within package and subclasses

- `default` (no modifier): Accessible within package
- `private`: Accessible only within the same class

Keywords

- `static`: Belongs to class, not instance
- `final`: Cannot be changed/overridden
- `abstract`: Must be implemented by subclass
- `synchronized`: Thread-safe method/block
- `volatile`: Variable may be modified by multiple threads
- `transient`: Not serialized
- `native`: Implemented in native code

Naming Conventions

- **Classes**: PascalCase (e.g., `MyClass`)
- **Methods/Variables**: camelCase (e.g., `myMethod`)
- **Constants**: UPPER_SNAKE_CASE (e.g., `MAX_SIZE`)
- **Packages**: lowercase (e.g., `com.company.project`)

This cheat sheet covers the essential Java programming concepts. For more advanced topics like generics, lambda expressions, streams, and concurrency, refer to official Java documentation.