Java Programming Cheatsheet

A comprehensive cheat sheet covering essential Java syntax, data structures, and commonly used libraries, designed to boost your coding efficiency.

Java Basics

Basic Syntax

`//`	Single-line comment.
`/* ... */`	Multi-line comment.
`public class Main { ... }`	Basic class definition.
`public static void main(String[] args) { ... }`	Main method - entry point of execution.
`System.out.println("Hello, World!");`	Prints text to the console.
`int x = 5;`	Integer variable declaration and initialization.

Data Types

`int`	Integer (4 bytes).
`double`	Double-precision floating-point number (8 bytes).
`boolean`	Boolean (true/false).
`String`	Sequence of characters.
`char`	Single character.
`float`	Single-precision floating-point number (4 bytes).

Operators

`+`, `-`, `*`, `/`, `%`	Arithmetic operators (addition, subtraction, multiplication, division, modulus).
`==`, `!=`, `>`, `<`, `>=`, `<=`	Comparison operators (equal to, not equal to, greater than, less than, greater than or equal to, less than or equal to).
`&&`, `||`, `!`	Logical operators (and, or, not).
`=`	Assignment operator.
`+=`, `-=`, `*=`, `/=`, `%=`	Compound assignment operators.
`++`, `--`	Increment and decrement operators.

Control Flow

Conditional Statements

if statement:

```java
if (condition) {
  // code to execute if condition is true
}
```

if-else statement:

```java
if (condition) {
  // code to execute if condition is true
} else {
  // code to execute if condition is false
}
```

if-else if-else statement:

```java
if (condition1) {
  // code to execute if condition1 is true
} else if (condition2) {
  // code to execute if condition2 is true
} else {
  // code to execute if all conditions are false
}
```

switch statement:

```java
switch (expression) {
 case value1:
  // code to execute if expression == value1
  break;
 case value2:
  // code to execute if expression == value2
  break;
 default:
  // code to execute if expression doesn't match any case
}
```

Looping Statements

for loop:

```java
for (initialization; condition; increment/decrement) {
 // code to execute repeatedly
}
```

while loop:

```java
while (condition) {
 // code to execute repeatedly while condition is true
}
```

do-while loop:

```java
do {
 // code to execute at least once
} while (condition);
```

enhanced for loop (for-each loop):

```java
for (DataType item : collection) {
 // code to execute for each item in the collection
}
```

Branching Statements

`break`	Terminates the loop or switch statement.
`continue`	Skips the current iteration and proceeds to the next iteration of the loop.
`return`	Exits from the current method.

Object-Oriented Programming

Classes and Objects

Class Definition:

```java
public class Dog {
  // Fields (attributes)
  String breed;
  int age;

  // Constructor
  public Dog(String breed, int age) {
    this.breed = breed;
    this.age = age;
  }

  // Method (behavior)
  public void bark() {
    System.out.println("Woof!");
  }
}
```

Object Creation:

```java
Dog myDog = new Dog("Golden Retriever", 3);
myDog.bark(); // Output: Woof!
```

Inheritance

Basic Inheritance:

```java
class Animal {
  String name;
  public void eat() {
    System.out.println("Animal is eating");
  }
}

class Dog extends Animal {
  public void bark() {
    System.out.println("Dog is barking");
  }
}
```

Using Inheritance:

```java
Dog myDog = new Dog();
myDog.name = "Buddy";
myDog.eat();  // Output: Animal is eating
myDog.bark(); // Output: Dog is barking
```

Polymorphism

Method Overriding:

```java
class Animal {
  public void makeSound() {
    System.out.println("Generic animal sound");
  }
}

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

Using Polymorphism:

```java
Animal myAnimal = new Dog();
myAnimal.makeSound(); // Output: Woof!
```

Encapsulation

Example of Encapsulation:

```java
class BankAccount {
  private double balance;

  public double getBalance() {
    return balance;
  }

  public void deposit(double amount) {
    balance += amount;
  }

  public void withdraw(double amount) {
    if (balance >= amount) {
    balance -= amount;
    } else {
    System.out.println("Insufficient funds");
    }
  }
}
```

Using Encapsulation:

```java
BankAccount account = new BankAccount();
account.deposit(1000);
account.withdraw(500);
System.out.println("Balance: " +
account.getBalance()); // Output: Balance:
500.0
```

Collections Framework

Lists

ArrayList:

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

List<String> list = new ArrayList<>();
list.add("Apple");
list.add("Banana");
System.out.println(list.get(0)); // Output:
Apple
```

LinkedList:

```java
import java.util.LinkedList;
import java.util.List;

List<String> linkedList = new LinkedList<>();
linkedList.add("Car");
linkedList.add("Bike");
System.out.println(linkedList.get(1)); //
Output: Bike
```

Sets

HashSet:

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

Set<String> set = new HashSet<>();
set.add("Red");
set.add("Blue");
System.out.println(set.contains("Red")); //
Output: true
```

TreeSet:

```java
import java.util.TreeSet;
import java.util.Set;

Set<String> treeSet = new TreeSet<>();
treeSet.add("Cat");
treeSet.add("Dog");
System.out.println(treeSet); // Output: [Cat,
Dog] (sorted order)
```

Maps

HashMap:

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

Map<String, Integer> map = new HashMap<>();
map.put("Alice", 25);
map.put("Bob", 30);
System.out.println(map.get("Alice")); //
Output: 25
```

TreeMap:

```java
import java.util.TreeMap;
import java.util.Map;

Map<String, Integer> treeMap = new TreeMap<>
();
treeMap.put("Charlie", 35);
treeMap.put("David", 40);
System.out.println(treeMap); // Output:
{Charlie=35, David=40} (sorted order by key)
```