Cheat Sheet for comprensive Java

Comprehensive Java Cheat Sheet

1. Basic Syntax

1.1 Hello World

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

1.2 Comments

```
// Single-line comment

/*
Multi-line comment
*/

/**
 * Documentation comment
 */
```

1.3 Variables

```
int age = 25;
double salary = 5000.50;
String name = "John Doe";
boolean isActive = true;
```

1.4 Constants

```
final int MAX_VALUE = 100;
```

---

2. Data Types

2.1 Primitive Data Types

Type	Size	Range
`byte`	1 byte	-128 to 127
`short`	2 bytes	-32,768 to 32,767
`int`	4 bytes	-2^31 to 2^31 - 1
`long`	8 bytes	-2^63 to 2^63 - 1
`float`	4 bytes	±3.40282347E+38F (6-7 significant digits)
`double`	8 bytes	±1.79769313486231570E+308 (15 significant digits)
`char`	2 bytes	0 to 65,535
`boolean`	1 bit	`true` or `false`

2.2 Reference Data Types

- `String`
- Arrays
- Classes
- Interfaces

---

3. Operators

3.1 Arithmetic Operators

```
+  // Addition
-  // Subtraction
*  // Multiplication
/  // Division
%  // Modulus
++ // Increment
-- // Decrement
```

3.2 Relational Operators

```
== // Equal to
!= // Not equal to
>  // Greater than
<  // Less than
```

```
>= // Greater than or equal to
<= // Less than or equal to
```

3.3 Logical Operators

```
&& // Logical AND
|| // Logical OR
!  // Logical NOT
```

3.4 Bitwise Operators

```
&  // AND
|  // OR
^  // XOR
~  // Complement
<< // Left shift
>> // Right shift
>>> // Unsigned right shift
```

3.5 Assignment Operators

```
=  // Assignment
+= // Add and assign
-= // Subtract and assign
*= // Multiply and assign
/= // Divide and assign
%= // Modulus and assign
```

---

4. Control Flow

4.1 Conditional Statements

```
if (condition) {
    // code
} else if (condition) {
    // code
} else {
    // code
}
```

4.2 Switch Statement

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

4.3 Loops

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

// While Loop
while (condition) {
    // code
}

// Do-While Loop
do {
    // code
} while (condition);
```

4.4 Break and Continue

```
// Break
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;
    }
}

// Continue
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue;
    }
}
```

---

5. Classes and Objects

5.1 Class Definition

```
public class MyClass {
    // Fields
    int myField;

    // Constructor
    public MyClass(int myField) {
        this.myField = myField;
    }

    // Methods
    public void myMethod() {
        // code
    }
}
```

5.2 Object Creation

```
MyClass obj = new MyClass(10);
```

5.3 Access Modifiers

- `public`: Accessible from anywhere.
- `private`: Accessible only within the class.
- `protected`: Accessible within the package and subclasses.
- `default`: Accessible within the package.

5.4 Static Members

```
public class MyClass {
    static int staticField;

    static void staticMethod() {
        // code
    }
}
```

---

6. Inheritance and Polymorphism

6.1 Inheritance

```
class Parent {
    // code
```

```
}

class Child extends Parent {
    // code
}
```

6.2 Method Overriding

```
class Parent {
    void display() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    @Override
    void display() {
        System.out.println("Child");
    }
}
```

6.3 Polymorphism

```
Parent obj = new Child();
obj.display(); // Output: Child
```

---

7. Interfaces and Abstract Classes

7.1 Interfaces

```
interface MyInterface {
    void myMethod();
}

class MyClass implements MyInterface {
    @Override
    public void myMethod() {
        // code
    }
}
```

7.2 Abstract Classes

```
abstract class MyAbstractClass {
    abstract void myMethod();
}

class MyClass extends MyAbstractClass {
    @Override
    void myMethod() {
        // code
    }
}
```

---

8. Exception Handling

8.1 Try-Catch Block

```
try {
    // code that may throw an exception
} catch (ExceptionType e) {
    // code to handle the exception
} finally {
    // code that always executes
}
```

8.2 Throwing Exceptions

```
throw new ExceptionType("Exception message");
```

8.3 Custom Exceptions

```
class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}
```

---

9. Collections Framework

9.1 List

```
List<String> list = new ArrayList<>();
list.add("Item1");
```

```
list.get(0);
list.remove(0);
```

9.2 Set

```
Set<String> set = new HashSet<>();
set.add("Item1");
set.contains("Item1");
set.remove("Item1");
```

9.3 Map

```
Map<String, String> map = new HashMap<>();
map.put("key1", "value1");
map.get("key1");
map.remove("key1");
```

9.4 Queue

```
Queue<String> queue = new LinkedList<>();
queue.add("Item1");
queue.peek();
queue.poll();
```

---

10. Generics

10.1 Generic Class

```
class MyGenericClass<T> {
    T myField;

    public MyGenericClass(T myField) {
        this.myField = myField;
    }

    public T getMyField() {
        return myField;
    }
}
```

10.2 Generic Method

```
public <T> void myGenericMethod(T param) {
    // code
}
```

---

11. Multithreading

11.1 Creating a Thread

```
class MyThread extends Thread {
    @Override
    public void run() {
        // code
    }
}

MyThread thread = new MyThread();
thread.start();
```

11.2 Runnable Interface

```
class MyRunnable implements Runnable {
    @Override
    public void run() {
        // code
    }
}

Thread thread = new Thread(new MyRunnable());
thread.start();
```

11.3 Synchronization

```
synchronized (lock) {
    // code
}
```

---

12. Input/Output (I/O)

12.1 Reading from Console

```
import java.util.Scanner;

Scanner scanner = new Scanner(System.in);
String input = scanner.nextLine();
```

12.2 Writing to File

```
import java.io.FileWriter;
import java.io.IOException;

try (FileWriter writer = new FileWriter("file.txt")) {
    writer.write("Hello, World!");
} catch (IOException e) {
    e.printStackTrace();
}
```

12.3 Reading from File

```
import java.io.FileReader;
import java.io.IOException;

try (FileReader reader = new FileReader("file.txt")) {
    int data = reader.read();
    while (data != -1) {
        System.out.print((char) data);
        data = reader.read();
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

---

13. Java Standard Library

13.1 String Manipulation

```
String str = "Hello";
str.length();
str.charAt(0);
str.substring(1, 3);
str.indexOf("e");
str.replace("H", "J");
```

13.2 Date and Time

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;

LocalDate date = LocalDate.now();
LocalTime time = LocalTime.now();
LocalDateTime dateTime = LocalDateTime.now();
```

13.3 Regular Expressions

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

Pattern pattern = Pattern.compile("\\d+");
Matcher matcher = pattern.matcher("123");
boolean matches = matcher.matches();
```

---

14. Miscellaneous Tips and Tricks

14.1 Debugging

- Use `System.out.println()` for quick debugging.
- Use IDE debugging tools for more advanced debugging.

14.2 Performance Tips

- Use `StringBuilder` for string concatenation in loops.
- Avoid creating unnecessary objects.
- Use primitive types when possible.

14.3 Best Practices

- Follow naming conventions (e.g., `camelCase` for variables, `PascalCase` for classes).
- Write clean and readable code.
- Use comments to explain complex logic.

This cheat sheet provides a comprehensive overview of essential Java concepts, syntax, and best practices. Use it as a quick reference to enhance your Java programming skills.

By Ahmed Baheeg Khorshid

ver 1.0