Cheat Sheet for comprehensive Java

Java Basics

Data Types

- **Primitive Types**:

- `byte`: 8-bit integer
- `short`: 16-bit integer
- `int`: 32-bit integer
- `long`: 64-bit integer
- `float`: 32-bit floating point
- `double`: 64-bit floating point
- `char`: 16-bit Unicode character
- `boolean`: true/false

- **Reference Types**:

- `String`: Immutable sequence of characters
- `Arrays`: Fixed-size collection of elements of the same type
- `Objects`: Instances of classes

Variables

- **Declaration**:

```
int number;
String name = "Java";
```

- **Final Variables**:

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

Operators

- **Arithmetic**: `+`, `-`, `*`, `/`, `%`
- **Comparison**: `==`, `!=`, `>`, `<`, `>=`, `<=`
- **Logical**: `&&`, `||`, `!`
- **Assignment**: `=`, `+=`, `-=`, `*=`, `/=`, `%=`

Control Flow

Conditional Statements

- **if-else**:

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

- **switch**:

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

Loops

- **for**:

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

- **while**:

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

- **do-while**:

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

- **Enhanced for loop**:

```
for (String item : array) {
    // code
}
```

Methods

Method Declaration

```
public static returnType methodName(parameterType parameter) {
    // code
    return returnValue;
}
```

Method Overloading

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

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

Classes and Objects

Class Declaration

```
public class MyClass {
    // Fields
    private int number;
    public String name;

    // Constructor
    public MyClass(int number, String name) {
        this.number = number;
        this.name = name;
    }

    // Methods
```

```
    public void display() {
        System.out.println("Number: " + number + ", Name: " + name);
    }
}
```

Object Creation

```
MyClass obj = new MyClass(10, "Java");
obj.display();
```

Inheritance and Polymorphism

Inheritance

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

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

Polymorphism

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

Exception Handling

try-catch-finally

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

throw and throws

```
public void method() throws Exception {
    if (condition) {
        throw new Exception("Exception message");
    }
}
```

Collections Framework

List

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

Set

```
Set<Integer> set = new HashSet<>();
set.add(1);
set.add(2);
```

Map

```
Map<String, Integer> map = new HashMap<>();
map.put("Java", 1);
map.put("Python", 2);
```

Generics

Generic Class

```
class Box<T> {
    private T item;

    public void setItem(T item) {
        this.item = item;
    }

    public T getItem() {
        return item;
    }
}
```

Generic Method

```
public <T> void printArray(T[] array) {
    for (T element : array) {
        System.out.println(element);
    }
}
```

Multithreading

Thread Creation

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

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

Runnable Interface

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

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

I/O Streams

File Reading

```
try (BufferedReader br = new BufferedReader(new
FileReader("file.txt"))) {
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

File Writing

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

Annotations

Built-in Annotations

- `@Override`: Indicates that a method overrides a superclass method.
- `@Deprecated`: Marks a method as deprecated.
- `@SuppressWarnings`: Suppresses compiler warnings.

Lambda Expressions

Basic Syntax

```
(parameter1, parameter2) -> {
    // code
}
```

Example

```
List<String> list = Arrays.asList("Java", "Python", "C++");
list.forEach(item -> System.out.println(item));
```

Stream API

Basic Operations

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
numbers.stream()
       .filter(n -> n % 2 == 0)
       .map(n -> n * 2)
       .forEach(System.out::println);
```

Date and Time API (Java 8+)

LocalDate

```
LocalDate date = LocalDate.now();
System.out.println(date);
```

LocalTime

```
LocalTime time = LocalTime.now();
System.out.println(time);
```

LocalDateTime

```
LocalDateTime dateTime = LocalDateTime.now();
System.out.println(dateTime);
```

Regular Expressions

Pattern Matching

```
String text = "Java is awesome";
Pattern pattern = Pattern.compile("Java");
Matcher matcher = pattern.matcher(text);

if (matcher.find()) {
    System.out.println("Match found!");
}
```

Java Concurrency Utilities

ExecutorService

```
ExecutorService executor = Executors.newFixedThreadPool(10);
executor.submit(() -> {
    // code
});
executor.shutdown();
```

Java Reflection

Class Information

```
Class<?> clazz = MyClass.class;
Method[] methods = clazz.getMethods();
for (Method method : methods) {
    System.out.println(method.getName());
}
```

Java Modules (Java 9+)

Module Declaration

```
module my.module {
    requires java.base;
    exports com.mypackage;
}
```

Java 11+ Features

String Methods

```
String text = "   Java   ";
System.out.println(text.isBlank()); // false
System.out.println(text.strip());   // "Java"
```

Files.readString and Files.writeString

```
String content = Files.readString(Path.of("file.txt"));
Files.writeString(Path.of("file.txt"), "New content");
```

Java 17+ Features

Sealed Classes

```
sealed class Shape permits Circle, Square {
    // code
}

final class Circle extends Shape {
    // code
}

final class Square extends Shape {
    // code
}
```

Java 21+ Features

Virtual Threads

```
Thread.startVirtualThread(() -> {
    // code
});
```

Best Practices

- **Use `final` for constants**: `final int MAX_VALUE = 100;`
- **Avoid raw types**: Use generics like `List<String>` instead of `List`.
- **Minimize scope of variables**: Declare variables as close as possible to their usage.
- **Use try-with-resources**: Automatically close resources like files and streams.
- **Follow naming conventions**: Class names in `UpperCamelCase`, method names in `lowerCamelCase`.

Debugging Tips

- **Use `System.out.println`**: For quick debugging.
- **Use IDE Debugger**: Set breakpoints, inspect variables, and step through code.
- **Check for `NullPointerException`**: Ensure all references are initialized.
- **Use logging frameworks**: Like Log4j or SLF4J for more sophisticated logging.

Performance Tips

- **Avoid premature optimization**: Focus on correctness first.
- **Use StringBuilder for string concatenation**: `StringBuilder` is more efficient than `+` operator.
- **Minimize object creation**: Reuse objects where possible.
- **Use primitive types where possible**: Avoid unnecessary boxing/unboxing.

Common Pitfalls

- **Null References**: Always check for `null` before accessing object properties.
- **Infinite Loops**: Ensure loop conditions will eventually be false.
- **Resource Leaks**: Always close resources like files and streams.
- **Concurrent Modification**: Avoid modifying collections while iterating over them.

Tools and Libraries

- **Maven/Gradle**: Build automation tools.
- **JUnit**: Unit testing framework.
- **Mockito**: Mocking framework for testing.
- **Spring Framework**: For building enterprise applications.

- **Hibernate**: ORM framework for database interactions.

Learning Resources

- **Oracle Java Documentation**: [docs.oracle.com](https://docs.oracle.com/en/java/)
- **Java Tutorials**: https://docs.oracle.com/javase/tutorial/
- **Stack Overflow**: stackoverflow.com
- **Books**: "Effective Java" by Joshua Bloch, "Java Concurrency in Practice" by Brian Goetz.

Conclusion

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

By Ahmed Baheeg Khorshid

ver 1.0