Coding Cheat Sheet: Building Blocks of Java Programming

This reading provides a reference list of code that you'll encounter as you begin to learn and use Java. Understanding these concepts will help you write and debug your first Java programs. Let's explore the following Java coding concepts:

- Structuring Java Code and Comments
- Exploring Data Types in Java
- Introduction to Operators in Java
- Using Advanced Operators in Java
- Working with arrays

Keep this summary reading available as a reference as you progress through your course and refer to this reading as you begin coding with Java after this course!

Structuring Java Code and Comments

Types of comments

Description	Code Example
A Java statement used to print text or other data to the standard output, typically the console.	`System.out.println("Hello, World!")`
Use two forward slashes to precede a single line comment in Java	`// This is a single-line comment.`
All text after the two forward slash marks on this line is treated as a comment	`int number = 10; // This variable stores the number 10`
These comments start with `/*` and end with `*/`. They can span multiple lines.	`/* This is a multi-line comment It can be used to explain a block of code or provide detailed information`
This also is a multiline comment. Multiline comments start with `/*` and end with `*/`. They can span multiple lines.	`/* int sum = 0;` `This variable will hold the sum of numbers */.`

The documentation	`/**`
	` *This method calculates the square of a number.`
	` *@param number The number to be squared`

Description	Code Example

comments start with `/*` and ends with `*/` . These comments are used for generating documentation using tools like Javadoc.	`*@return The square of the input number`
	`*/`
	`public int square(int number) {`
	`return number * number;`
	`}`

Creating a package

Description	Code Example

To create a package, use the package keyword at the top of your Java source file.	`package com.example.myapp; // Declare a package`
	`public class MyClass {`
	`// Class code goes here`
	`}`

Folder structure for a package

Description	Folder Structure Example

The folder structure on your filesystem should match the package declaration. For instance, if your package is com.example.myapp, your source file should be located in the following path example.	`/src`
	`└── com`
	`└── example`
	`└─>└─>`
	`└─>└── └─>myapp`
	`└─>└── └─>└── MyClass.java`

Class and Methods Structure

Description	Code Example

In Java, a class is a blueprint for creating objects and can contain methods (functions) that define behaviors. For instance, the second line of this code displays the public class `library`, with methods like calculatePrice() or applyDiscount().	`package com.example.library;`
	`public class Library`
	`{`
	`private List <Book> books; // Private attribute to hold books`
	`public Library()`
	`{`
	`books = new ArrayList<>(); // Initialize the list in the constructor`
	`}`
	`public void addBook(Book book) {`
	`books.add(book); // Method to add a book to the library`
	`}`
	`}`

Creating methods

Description	Code Example

Every Java application needs an entry point, which is typically a main method within a public class. In this code, the second line of code identifies the method named Main in the public class.	package com.example.library;
	
	public class Main {
	
	public static void main(String[] args) {
	
	Library library = new Library(); // Create an instance of Library
	
	// Add books to the library
	library.addBook(new Book("1984", "George Orwell"));
	library.addBook(new Book("To Kill a Mockingbird", "Harper Lee"));
	
	// Display all books
	library.displayBooks();
	}
	}

Organizing source files in directories

Description	Directory Structure
As your project grows, organizing your source files into directories can keep your code manageable. The following example illustrates typical Java organizaton.	MyProject/ └── src/ # Source code goes here └── lib/ # External libraries/JARs └── resources/ # Configuration files, images, and others └── doc/ # Documentation └── test/ # Test files

Using imports

Description	Code Example
When you need to use classes from other packages, you need to import them at the top of your source file. These examples illustrate two examples of importing classes.	import java.util.List; import java.util.ArrayList; // Importing classes from Java's standard library </pre

Exploring Data Types in Java

Primitive data types

Description	Code Example
Use the `byte` data type when you need to save memory in large arrays where the memory savings are critical, and the values are between -128 and 127.	`byte age = 25; // Age of a person`
Use the `short` small integers data type for numbers from −32,768 to 32,767.	`short temperature = -5; // Temperature in degrees`
Use the `int` integer data type to store whole numbers larger than what byte and short can hold. This is the most commonly used integer type.	`int population = 1000000; // Population of a city`
Use the `long` data type when you need to store very large integer values that exceed the range of `int`.	`long distanceToMoon = 384400000L; // Distance in meters`
Use the `float` when you need to store decimal numbers but do not require high precision (for example, up to 7 decimal places).	`float price = 19.99f; // Price of a product`
Use the `double` data type when you need to store decimal numbers and require high precision (up to 15 decimal places).	`double pi = 3.141592653589793; // Value of Pi`
Use the `char` data type when you need to store a single character such as a single letter or an initial.	`char initial = 'A'; // Initial of a person's name`

Description	Code Example
Use `boolean` when you need to represent a true/false value. Boolean is often used for conditions and decisions.	`boolean isLoggedIn = true; // User login status`

Reference data types

Description	Code Example
A `string` data type is a sequence of characters. The `string` data type is very useful for handling text in your programs.	`String greeting = "Hello, World!";`
An `array` is a collection of multiple values stored under a single variable name. All the values in an array must be of the same type. Arrays are great for storing lists of items, like student scores or names. The following code defines an integer array type of scores that include 85, 90, 78, and 92.	`int[] scores = {85, 90, 78, 92};`

The reference data type `class` is like a blueprint for creating objects. You can see the class identified in the first line of code.	`class Car {`
	` String color;`
	` int year;`
	` void displayInfo() {`
	` System.out.println("Color: " + color + ", Year: " + year);`
	` }`
	`}`

Objects are classes that contain both data and functions. In this code `Car myCar = new Car();` is the object.	`public class Main {`
	` public static void main(String[] args) {`
	`  Car myCar = new Car();`
	`  myCar.color = "Red";`
	`  myCar.year = 2026;`
	`  myCar.displayInfo(); // Output: Color: Red, Year: 2026`
	`  }`
	`}`

When you create an interface, you only declare the methods without providing their actual code. All methods in an interface are empty by default. Here's an example of an interface called MyInterfaceClass with three methods.	`// The interface class`
	`interface MyInterfaceClass {`
	`  void methodExampleOne();`
	`  void methodExampleTwo();`
	`  void methodExampleThree();`
	`}`

Description	Code Example

An `enum` is a special data type that defines a list of named values. An enum is useful for representing fixed sets of options, such as days of the week or colors.	`enum DaysOfWeek {`
	`MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;`
	`}`

Introduction to Operators in Java

Arithemtic operators

The following operators perform basic mathematical functions.
In these code examples `int a = 10;` and `int b = 5;`

Description	Example
The plus symbol `+` performs addition operations.	`System.out.println("Addition: " + (a + b));  // 15`
The minus symbol `-` performs subtraction operations.	`System.out.println("Subtraction: " + (a - b));  // 5`
The asterisk symbol `*` performs multiplication operations.	`System.out.println("Multiplication: " + (a * b));  // 50`
The division symbol `/` performs division operations.	`System.out.println("Division: " + (a / b));  // 2`
The percentage symbol `%` performs modulus (percentage) operations.	`ystem.out.println("Modulus: " + (a % b));  // 0`

Description	Example

Relational operators

You use relational operators to compare two values. These operators return a boolean result (true or false).

In the following examples `int a = 10;` and `int b = 5;`.

Description	Example
The double equal symbols == check for equality.	`System.out.println("Is a equal to b? " + (a == b));  // false`
The exclamation point and equal symbol together != check for a not equal result.	`System.out.println("Is a not equal to b? " + (a != b));  // true`
The greater than symbol > checks for the greater than result.	`System.out.println("Is a greater than b? " + (a > b));  // true 0`
The less than symbol < checks if the result for a less than state.	`System.out.println("Is a less than  b? " + (a < b)); // false`
The greater than equal symbols >= compares the values to check for a greater than or equal to result.	`System.out.println("Is a greater than or equal to b? " + (a >= b)); // true`
The less than equal symbols <= compares the values to check for a less than or equal to result.	`System.out.println("Is a less than or equal to b? " + (a <= b)); // false`

Description	Example

Logical operators

You can use logical operators to combine boolean expressions. The following code examples use `boolean x = true;` and `boolean y = false;`.

Description	Example
The double ampersands && combines two boolean expressions (both must be true).	`if (a > b && b < c) { ... }`
The double vertical bar symbols `||` used with boolean values do not evaluate the second expression if the first one is true; referred as short-circuit evaluation.	`if (a > b || b < c) { ... }`
The single exclamation point symbol `!` operator in Java is the logical NOT operator. This operator is used to negate a boolean expression, meaning it reverses the truth value.	`if (!(a > b)) { ... }`

Using Advanced Operators in Java

Assignment operators

You can use assignment operators to assign values to a variable.

Description	Example
The single equal symbol = assigns the right operand to left	`a = 10`
The plus sign and equal symbols combined += adds and assigns values to variables	`a += 5`
The minus sign and equal symbols combined `-=` subtracts values to variables	`a -= 2`

Description	Example
The asterisk sign and equal symbols combined `*=` multipies and assigns values to variables	`a *= 3`
The forward slash sign and equal symbols combined `/=` divides and assigns values to variables	`a /= 2`
The percentage and equal symbols combined `%=` take the modulus (percentage) and assigns values to the variables	`a %= 4`

Unary operators

A unary operation is a mathematical or programming operation that uses only one operand or input. Developers use unary operations to manipulate data and perform calculations. You would use the following assignment operators to assign values to variables.

Description	Example

Use the plus symbol to indicate a positive value.	`int positive = +a;`
	
	`int a = 10;`
	`System.out.println("Unary plus: " + (+a));  // 10`

Ternary operators

Ternary operators are a shorthand form of the conditional statement. They can use three operands.

Description	Example

Ternary operators uses the Syntax: condition ? expression1 : expression2;. In the following code, `int max = (a > b) ? a : b;`	`int a = 10;`
	`int b = 20;`
	`int max = (a > b) ? a : b; // If a is greater than b, assign a; otherwise assign b`
	`System.out.println("Maximum value is: " + max); // 20`

Working with Arrays

Declaring an array

Description	Example
To declare an array in Java, you use the following syntax: `dataType[] arrayName;`. The following doce displays the data type of int and creates an array named `numbers`.	`int[] numbers;`

Initializing an array

After you declare an array, you need to initialize the array to allocate memory for it.

Description	Example

These code samples display three methods used to create an array of 5 integers. You can initialize an array using the `new` keyword. You can also declare and initialize an array in a single line. Alternatively, you can create an array and directly assign values using curly braces.	`numbers = new int[5];`
	
	`int[] numbers = new int[5];`
	
	`int[] numbers = {1, 2, 3, 4, 5};`

Accessing an array

You can access individual elements in an array by specifying the index inside square brackets.

Description	Example

Remember that indices start at zero. Here are two examples:	`System.out.println(numbers[0]);System.out.println(numbers[0]); // Outputs: 1`
	
	`System.out.println(numbers[4]); // Outputs: 5`

Modifying array elements

Description	Example

Modify the array element value by accessing it within its index.	`numbers[2] = 10; // Changing the third element to 10`
	
	`System.out.println(numbers[2]);>System.out.println(numbers[2]); // Outputs: 10`

Verify the array length

Description	Example
Verify the array length by using the length property	`System.out.println("The length of the array is: " + numbers.length);`

Using a for loop to iterate through an array

Description	Example

You can use a `for` loop for to iterarate through and array. Here's an example that prints all elements in the numbers array. The `for` loop includes this code `for (int i = 0` in the following code snippet.	`for (int i = 0; i < numbers.length; i++) {`
	`System.out.println(numbers[i]);`
	`}`

Description	Example

Using a for each loop to iterate through an array

Description	Example

You can also use the enhanced `for` loop, known as the "for-each" loop. The enhanced `for each` loop code include this portion, `for (int` of the following code snippet.	`for (int number : numbers) {`
	`System.out.println(number);`
	`}`

Declare and intialize a multidimensional array

Java supports multi-dimensional arrays, which are essentially arrays of arrays. The most common type is the two-dimensional array.

Description	Example

Here's how to declare and initialize a 2D array. You will declare the integer data type and create the matrix array.	`int[][] matrix = {`
	`{1,{1, 2, 3},`
	`{4, 5, 6},`
	`{7, 8, 9}`
	`};`

You can access elements in a two-dimensional array by specifying both indices, which are shown in this code inside of the square brackets.	`System.out.println(matrix[0][1]); // Outputs: 2`

Iterating Through a 2D Array

Description	Example

You can use nested loops to iterate through all elements of a 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.print(matrix[i][j] + " ");`
	`}`
	`System.out.println(); // Move to the next line after each row`

Author(s)

Ramanujam Srinivasan
Lavanya Thiruvali Sunderarajan