Coding Cheatsheet: Control Structures and String Handling

Estimated duration: 10 minutes

This reading provides a reference list of code that you'll encounter as you explore control structures and string handling. Understanding how to apply this code will help you write and debug your first Java programs.

- Using Conditional Statements
- Introduction to Loops in Java
- Working with Strings in Java
- Using Packages and Imports
- Implementing Functions andMethods

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!

Using Conditional Statements

if statement

The `if` statement checks a condition. If the condition is `true`, it executes the code inside the block. If the condition is `false`, the program skips the `if` block.

Description	Example
A Java class named `Main` with a `main` method. The `main` method is the entry point of the program.	`public class Main {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A variable `number` of type `int` is declared and initialized with the value `10`.	`int number = 10;`

The `if` statement checks if `number` is greater than `5`. If `true`, it prints "The number is greater than 5."	`if (number > 5) {`
	`  System.out.println("The number is greater than 5.");`
	`}`

Closing curly braces to end the `main` method and class definition.	`  }` `}`

Explanation: Since `number` is `10`, which is greater than `5`, the condition evaluates to `true`, and the program prints `"The number is greater than 5."`

---

if-else statement

The `if-else` statement gives an alternative if the condition is `false`.

Description	Example
A Java class named `Main` with a `main` method. The `main` method is the entry point of the program.	`public class Main {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A variable `number` of type `int` is declared and initialized with the value `3`.	`int number = 3;`

The `if` statement checks if `number` is greater than `5`. If `true`, it prints "The number is greater than 5."	`if (number > 5) {`
	`System.out.println("The number is greater than 5.");`
	`}`

The `else` block executes when the `if` condition is `false`, printing "The number is not greater than 5."	`else {`
	`System.out.println("The number is not greater than 5.");`
	`}`

Closing curly braces to end the `main` method and class definition.	`}` `}`

---

else if statement

You can check multiple conditions using `else if`.

Description	Example
A Java class named `Main` with a `main` method. The `main` method is the entry point of the program.	`public class Main {`

Description	Example
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A variable `number` of type `int` is declared and initialized with the value `5`.	`int number = 5;`

The `if` statement checks if `number` is greater than `5`. If `true`, it prints "The number is greater than 5."	`if (number > 5) {`
	`  System.out.println("The number is greater than 5.");`
	`}`

The `else if` statement checks if `number` is exactly `5`. If `true`, it prints "The number is equal to 5."	`else if (number == 5) {`
	`  System.out.println("The number is equal to 5.");`
	`}`

The `else` block executes when none of the above conditions are met, printing "The number is less than 5."	`else {`
	`  System.out.println("The number is less than 5.");`
	`}`

Closing curly braces to end the `main` method and class definition.	`  }` `}`

Explanation: Since `number` is exactly `5`, the program prints `"The number is equal to 5."`

---

switch statement

A `switch` statement checks a single variable against multiple values.

Description	Example
A Java class named `Main` with a `main` method. The `main` method is the entry point of the program.	`public class Main {`

Description	Example
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A variable `day` of type `int` is declared and initialized with the value `3`.	`int day = 3;`

The `switch` statement checks the value of `day` and compares it against the `case` labels. If `day` is `3`, it prints `"Wednesday"`.	`switch (day) {`
	`  case 1:`
	`  System.out.println("Monday");`
	`  break;`
	`  case 2:`
	`  System.out.println("Tuesday");`
	`  break;`
	`  case 3:`
	`  System.out.println("Wednesday");`
	`  break;`
	`  case 4:`
	`  System.out.println("Thursday");`
	`  break;`
	`  case 5:`
	`  System.out.println("Friday");`
	`  break;`
	`  default:`
	`  System.out.println("Weekend");`
	`}`

Closing curly braces to end the `main` method and class definition.	`  }` `}`

---

default case in a switch statement

When using a `switch` statement, it's a good practice to specify a `default` case. This case runs if none of the specified cases match, acting as a fallback option.

Description	Example
A `switch` statement checks the value of a variable `color`.	`switch (color) {`

A `case` checks if `color` is `"red"`, printing `"Color is red."`	`  case "red":`
	`  System.out.println("Color is red.");`
	`  break;`

Description	Example

A `case` checks if `color` is `"blue"`, printing `"Color is blue."`	`case "blue":`
	`System.out.println("Color is blue.");`
	`break;`

The `default` case prints `"Unknown color."` if `color` doesn't match `"red"` or `"blue"`.	`default:`
	`System.out.println("Unknown color.");`
	`}`

Nested Conditional Statements

You can place conditional statements within each other to create more complex decisions. The process of placing conditional statements within other conditional statements is called nesting.

Description	Example
A Java class named `Main` with a `main` method. The `main` method is the entry point of the program.	`public class Main {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A variable `age` of type `int` is declared and initialized with the value `20`.	`int age = 20;`

The `if` statement checks if `age` is greater than or equal to `18`. If `true`, it prints `"You are an adult."`	`if (age >= 18) {`
	`System.out.println("You are an adult.");`
	`}`

Description	Example

Another `if` statement checks if `age` is greater than or equal to `65`, printing `"You are a senior citizen."` if true.	`if (age >= 65) {`
	`    System.out.println("You are a senior citizen.");`
	`}`

The `else` block executes if `age` is less than `18`, printing `"You are a minor."`	`else {`
	`    System.out.println("You are a minor.");`
	`}`

Closing curly braces to end the `main` method and class definition.	`    }` `}`

Introduction to Loops in Java

for Loop

The for loop is used when the number of iterations is known beforehand. It consists of three parts:

- Initialization: This sets a counter variable.
- Condition: This checks if the loop should continue executing.
- Increment/Decrement: This updates the counter variable after each iteration.

Description	Example
A Java class named `ForLoopExample` with a `main` method. The `main` method is the entry point of the program.	`public class ForLoopExample {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A for loop is initialized with `int i = 1`, which starts the counter at 1. The counter variable `i` is incremented by `i++` after each iteration.	`for (int i = 1; i <= 5; i++) {`
The loop checks if `i <= 5`, and if true, it prints the value of `i`.	`System.out.println(i);`

Description	Example
Close the `for` loop.	`}`
Close the `main` method.	`}`
Close the `ForLoopExample` class.	`}`

while Loop

The while loop is used when the number of iterations is not known in advance. It continues executing as long as the specified condition remains true.

Description	Example
A Java class named `WhileLoopExample` with a `main` method. The `main` method is the entry point of the program.	`public class WhileLoopExample {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A variable `i` is initialized to `1`.	`int i = 1;`
The while loop continues as long as `i <= 5`.	`while (i <= 5) {`

Description	Example

Inside the loop, the value of `i` is printed, then incremented by `i++`.	`System.out.println(i);`
	`i++;`
	`}`

The `main` method and class are closed with curly braces.	`}` `}`

do-while Loop

The do-while loop is similar to the while loop but guarantees that the code block executes at least once before checking the condition.

Description	Example
A Java class named `DoWhileLoopExample` with a `main` method. The `main` method is the entry point of the program.	`public class DoWhileLoopExample {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
A variable `i` is initialized to `1`.	`int i = 1;`
The do-while loop starts and executes at least once before checking if `i <= 5`.	`do {`
Inside the loop, the value of `i` is printed, then incremented by `i++`.	`System.out.println(i);` `i++;`

Description	Example
The condition `i <= 5` is checked after each iteration.	`} while (i <= 5);`
The `main` method and class are closed with curly braces.	`}` `}`

Nested loops

You can also use loops inside other loops, known as nested loops. This is useful for working with multi-dimensional data structures, like arrays or matrices.

Description	Example
A Java class named `NestedLoopsExample` with a `main` method. The `main` method is the entry point of the program.	`public class NestedLoopsExample {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
The outer loop controls the rows, running 10 times.	`for (int i = 1; i <= 10; i++) {`
The inner loop controls the columns, also running 10 times for each row.	`for (int j = 1; j <= 10; j++) {`
The product of `i * j` is printed for each combination of rows and columns.	`System.out.print(i * j + "\t");}`
After the inner loop, a newline is printed to separate the rows.	`System.out.println();}`

Description	Example
The `main` method and class are closed with curly braces.	`}` `}`

break statement

The break statement is used to terminate a loop immediately, regardless of the loop's condition. This can be useful when you want to exit a loop based on a specific condition that may occur during its execution.

Description	Example
A Java class named `BreakExample` with a `main` method. The `main` method is the entry point of the program.	`public class BreakExample {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
An array `numbers` is declared and initialized.	`int[] numbers = {1, 3, 5, 7, 9, 2, 4};`
The loop iterates through the array, checking if any number is greater than 5.	`for (int num : numbers) {`

When a number greater than 5 is found, it is printed and the loop exits with the `break` statement.	`if (num > 5) {`
	`System.out.println(num);`
	`break;`
	`}`

The `main` method and class are closed with curly braces.	`}` `}` `}`

Description	Example

continue statement

The continue statement is used to skip the current iteration of a loop and move to the next iteration. It's useful when you want to skip certain conditions but continue executing the rest of the loop.

Description	Example
A Java class named `ContinueExample` with a `main` method. The `main` method is the entry point of the program.	`public class ContinueExample {`
The `main` method is declared using `public static void main(String[] args)`. This method is required for execution in Java programs.	`public static void main(String[] args) {`
The loop iterates through the numbers from 1 to 10.	`for (int i = 1; i <= 10; i++) {`

When `i == 5`, the `continue` statement is executed, skipping the `System.out.println(i);` statement for that iteration.	`if (i == 5) {`
	` continue;`
	`}`

The value of `i` is printed for all numbers except 5.	`System.out.println(i);`
The `main` method and class are closed with curly braces.	`}` `}` `}`

Working with Strings in Java

Creating strings

You can create a string in Java in two main ways:

Using string literals: This means writing the string directly inside double quotes.

Description	Example
Create a string using string literal.	`String greeting = "Hello, World!";`

In this example, we created a string called greeting that contains "Hello, World!".

Using the new Keyword: This method involves using the new keyword to create a string object.

Description	Example
Create a string using the new keyword.	`String message = new String("Hello, World!");`

Although this works, it's more common to use string literals because it's simpler.

String length

To find out how many characters are in a string, you can use the length() method. This method tells you the total number of characters in the string.

Description	Example
Create a string and use length() to get the number of characters.	`String text = "Java Programming";`
Use the length() method to find the string length.	`int length = text.length();`
Print the length of the string.	`System.out.println("Length of the string: " + length); // Output: 16`

Here, we created a string called text and then checked its length. The output tells us that there are 16 characters in "Java Programming".

Accessing characters

If you want to look at individual characters in a string, you can use the charAt() method. This method allows you to get a character at a specific position in the string.

Description	Example
Create a string and access a character using charAt().	`String word = "Java";`
Access the first character of the string.	`char firstChar = word.charAt(0);`
Print the first character of the string.	`System.out.println("First character: " + firstChar); // Output: J`

In this example, we accessed the first character of the string "Java". Remember that counting starts at 0, so charAt(0) gives us 'J'.

Concatenating strings

Sometimes you might want to combine two or more strings together. You can do this easily using the + operator or the concat() method.

Description	Example
Combine two strings using the + operator.	`String firstName = "John";`
Combine two strings using the + operator.	`String lastName = "Doe";`
Concatenate first and last names using the + operator.	`String fullName = firstName + " " + lastName;`
Print the full name.	`System.out.println("Full Name: " + fullName); // Output: John Doe`

Here, we combined firstName and lastName using the + operator and added a space between them.

You can also use the concat() method:

Description	Example
Combine strings using the concat() method.	`String anotherFullName = firstName.concat(" ").concat(lastName);`
Print the concatenated string.	`System.out.println("Another Full Name: " + anotherFullName); // Output: John Doe`

String comparison

When you want to check if two strings are the same, use the equals() method. This checks if both strings have identical content.

Description	Example
Create three strings to compare.	`String str1 = "Hello";`
Create another string to compare.	`String str2 = "Hello";`
Create a third string to compare.	`String str3 = "World";`
Check if str1 is equal to str2.	`boolean isEqual = str1.equals(str2);`
Print comparison result.	`System.out.println("str1 equals str2: " + isEqual); // Output: true`
Check if str1 is equal to str3.	`boolean isNotEqual = str1.equals(str3);`

Description	Example
Print comparison result.	`System.out.println("str1 equals str3: " + isNotEqual); // Output: false`

In this example, isEqual returns true because both strings are "Hello". However, isNotEqual returns false since "Hello" and "World" are different.

String immutability

One important thing to know about strings in Java is that they are immutable. This means that once a string is created, it cannot be changed. If you try to change it, you will actually create a new string instead.

Description	Example
Create an original string.	`String original = "Hello";`
Add to the string (creates a new string).	`original = original + " World";`
Print the new string.	`System.out.println(original); // Output: Hello World`

In this case, we added "World" to original, but instead of changing the original string, we created a new string that combines both parts.

Finding substrings

You may want to get a smaller part of a string. You can do this using the substring() method, which allows you to specify where to start and where to stop.

Description	Example
Create a string and extract a substring.	`String phrase = "Java Programming";`
Extract a substring from the string.	`String sub = phrase.substring(5, 16);`

Description	Example
Print the extracted substring.	`System.out.println("Substring: " + sub); // Output: Programming`

In this example, we started at index 5 and went up to (but did not include) index 16 to extract "Programming".

String methods

Java has many built-in methods for strings that help you manipulate and process them. Here are some useful ones:

toUpperCase(): This method converts all letters in a string to uppercase.

Description	Example
Create a string.	`String text = "hello";`
Convert the string to uppercase.	`System.out.println(text.toUpperCase()); // Output: HELLO`

toLowerCase(): This converts all letters in a string to lowercase.

Description	Example
Create a string.	`String text = "WORLD";`
Convert the string to lowercase.	`System.out.println(text.toLowerCase()); // Output: world`

trim(): This method removes any extra spaces at the beginning or end of a string.

Description	Example
Create a string with extra spaces and trim it.	`String textWithSpaces = "   Hello   ";`

Description	Example
Remove spaces from the string.	`System.out.println(textWithSpaces.trim()); // Output: Hello`

replace(): If you want to change all instances of one character or substring to another, use this method.

Description	Example
Create a sentence and replace a word.	`String sentence = "I like cats.";`
Replace a word in the sentence.	`String newSentence = sentence.replace("cats", "dogs");`
Print the new sentence.	`System.out.println(newSentence); // Output: I like dogs.`

Splitting strings

You can break a string into smaller pieces using the split() method. This is useful when you have data separated by commas or spaces.

Description	Example
Create a CSV string and split it.	`String csv = "apple,banana,cherry";`
Split the string at each comma.	`String[] fruits = csv.split(",");`

Print each fruit in the array.	`for (String fruit : fruits) {`
	`  System.out.println(fruit);`
	`}`

Description	Example
Output:	apple banana cherry

Joining strings

If you have an array of strings and want to combine them back into one single string, you can use the String.join() method.

Description	Example
Create an array of strings.	`String[] colors = {"Red", "Green", "Blue"};`
Join the strings with a separator.	`String joinedColors = String.join(", ", colors);`
Print the joined string.	`System.out.println(joinedColors); // Output: Red, Green, Blue`

Using Packages and Imports

Creating a package

To create a package, you simply declare it at the top of your Java source file using the package keyword followed by the package name. For example:

Description	Example
Declare a package at the top of the file.	`package com.example.myapp;`

In this example, com.example.myapp is the name of the package. It's common practice to use a reverse domain name as the package name to ensure uniqueness.

Description	Example

Define a class inside the package.	`public class MyClass {`
	`  // Class code here`
	`}`

Creating and using a package

Description	Example
Define a class inside the shapes package.	`package shapes;`

Create the Circle class with a constructor and a method.	`public class Circle {`
	`  private double radius;`
	
	`  public Circle(double radius) {`
	`  this.radius = radius;`
	`  }`
	
	`  public double area() {`
	`  return Math.PI * radius * radius;`
	`  }`
	`}`

To use this class in another Java file, you need to import it.

Importing classes

To import a specific class from a package, use the following syntax:

Description	Example
Import a specific class from a package.	`import package_name.ClassName;`

Importing all classes from a package

You can also import all classes from a package using an asterisk (*).

Description	Example
Import all classes from the shapes package.	`import shapes.*;`

This imports all classes in the shapes package, allowing you to use any class without needing to import them individually.

Implementing Functions and Methods

Function structure

Description	Example

The structure of a function in Java.	`returnType functionName(parameter1Type parameter1, parameter2Type parameter2) {`
	`  // code to be executed`
	`  return value; // optional`
	`}`

Example of a Simple Function

Let's create a simple function that adds two numbers:

Description	Example

Create a function that adds two numbers.	`public static int add(int a, int b) {`
	`  return a + b;`
	`}`

Call the add function in the main method and print the result.	`int sum = add(5, 3);` `System.out.println("The sum is: " + sum);`

Example of a simple method

Let's create a method within a class:

Description	Example
Define a multiply method inside the Calculator class.	`public class Calculator {`
The multiply method takes two integers and returns their product.	`public int multiply(int x, int y) {`
Close the multiply method.	`return x * y;` `}`

Description	Example
Define the main method, which is the program's entry point.	`public static void main(String[] args) {`
Create an instance of the Calculator class in the main method.	`Calculator calc = new Calculator();`
Call the multiply method with 4 and 5 and store the result.	`int product = calc.multiply(4, 5);`
Print the result to the screen.	`System.out.println("The product is: " + product);`
Close the main method and class.	`}` `}`

Parameters and arguments

Parameters are the inputs to functions or methods, while arguments are the values passed when calling these functions or methods.

Description	Example

Define a method with multiple parameters.	`public void greet(String name, int age) {`
	`    System.out.println("Hello " + name + ", you are " + age + " years old.");`
	`}`

Call the greet method with arguments in the main method.	`Greeting greeting = new Greeting();` `greeting.greet("Alice", 30);`

Description	Example

Return values

Functions and methods can return values or perform actions without returning anything.

Description	Example

Define a method that returns a value.	`public double area(double length, double width) {`
	`  return length * width;`
	`}`

Call the area method and print the returned value.	`Rectangle rect = new Rectangle();`
	`double area = rect.area(4.5, 3.0);`
	`System.out.println("The area of the rectangle is: " + area);`

Overloading methods

Java allows defining multiple methods with the same name but different parameters. This is known as method overloading.

Description	Example
Define the `Display` class.	`public class Display {`
Define an overloaded method that takes an `int`.	`  public void show(int number) {`
Print the number.	`  System.out.println("Number: " + number);`
Close the first method.	`  }`

Description	Example
Define an overloaded method that takes a `String`.	`public void show(String text) {`
Print the text.	`System.out.println("Text: " + text);`
Close the second method.	`}`
Define the `main` method to call the overloaded methods.	`public static void main(String[] args) {`
Create a `Display` object.	`Display display = new Display();`
Call `show(int)` and `show(String)`.	`display.show(10);`
display.show("Hello World");	
Close the `main` method.	`}` `}`

Scope of identifiers

The scope of an identifier refers to the part of the program where the identifier can be accessed.

Description	Example
Local Scope: Identifiers are accessible only within the method or block.	`int x = 10; // x is local to this block`
Instance Scope: Variables are accessible by all methods in the class.	`private int x; // x is accessible by all methods`
Static Scope: Static variables belong to the class and are accessible throughout the class.	`private static int count;`

Void methods

A void method does not return a value.

Description	Example

Define a void method that prints a message.	`public void printMessage() {`
	`  System.out.println("Hello, World!");`
	`}`

Empty parameter lists

An empty parameter list means the method does not take any parameters.

Description	Example

Define a method with an empty parameter list.	`public void show() {`
	`  System.out.println("No parameters here.");`
	`}`

Author(s)

[Ramanujam Srinivasan]
[Lavanya Thiruvali Sunderarajan]

Skills Network