OOP

Lab Manual (Lab 2)

****

Session: Spring 2024

LAB INSTRUCTOR: AYESHA MAJID ALI

**Objective:**

* Java String concatenation
* Java Methods

## **Java Strings**

Strings are used for storing text.

A String variable contains a collection of characters surrounded by double quotes:

## **Example:**

Create a variable of type String and assign it a value:

String greeting = "Hello"

## **String Length**

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the length() method:

## **Example**

String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

System.out.println("The length of the txt string is: " + txt.length());

## 

## **More String Methods**

There are many string methods available, for example toUpperCase() and toLowerCase():

## **Example**

String txt = "Hello World";

System.out.println(txt.toUpperCase()); // Outputs "HELLO WORLD"

System.out.println(txt.toLowerCase()); // Outputs "hello world"

Note: camelCase naming convention is used for Lower case and upper-case methods.

A screen shot of a computer code

Description automatically generated

## **Finding a Character in a String**

The indexOf() method returns the **index** (the position) of the first occurrence of a specified text in a string (including whitespace):

## **Example**

String txt = "Please locate where 'locate' occurs!";

System.out.println(txt.indexOf("locate")); // Outputs 7

Java counts positions from zero.  
0 is the first position in a string, 1 is the second, 2 is the third ...

## **String Concatenation**

The + operator can be used between strings to combine them. This is called **concatenation**:

## **Example**

String firstName = "John";

String lastName = "Doe";

System.out.println(firstName + " " + lastName);

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.

You can also use the concat() method to concatenate two strings:

## **Example**

String firstName = "John ";

String lastName = "Doe";

System.out.println(firstName.concat(lastName));

## **Adding Numbers and Strings**

WARNING!

Java uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

## **Example**

int x = 10;

int y = 20;

int z = x + y; // z will be 30 (an integer/number)

If you add two strings, the result will be a string concatenation:

## **Example**

String x = "10";

String y = "20";

String z = x + y; // z will be 1020 (a String)

If you add a number and a string, the result will be a string concatenation:

## **Example**

String x = "10";

int y = 20;

String z = x + y; // z will be 1020 (a String)

**Java String Concatenation: Techniques, Tips, and Examples**

Are you finding it challenging to concatenate strings in Java? You’re not alone. Many developers find themselves puzzled when it comes to handling string concatenation in Java, but we’re here to help.

Think of Java string concatenation as a puzzle – each string is a piece that fits together to form a complete picture. It’s a fundamental concept in Java, but it can be tricky to master, especially when dealing with more complex scenarios.

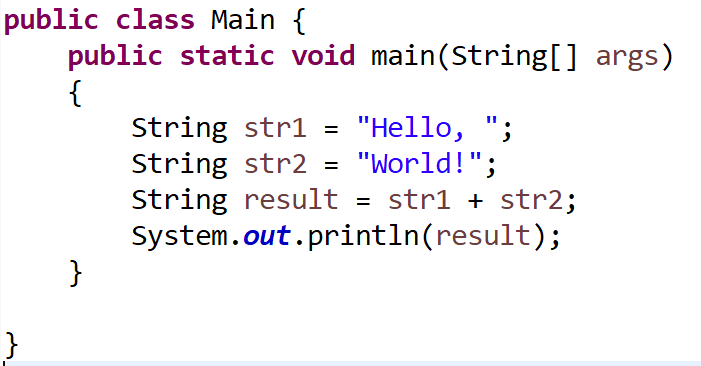
**In this guide, we’ll walk you through the process of string concatenation in Java**, from the basics to more advanced techniques. We’ll cover everything from using the ‘+’ operator, to more efficient methods like StringBuilder and StringBuffer, and even alternative approaches.

Let’s dive in and start mastering Java string concatenation!

**How Do I Concatenate Strings in Java?**

*The simplest way to concatenate strings in Java is using the ‘+’ operator. For instance, you can concatenate two strings,*str1*and*str2*, like this:*String result = str1 + str2;*.*

Here’s a simple example:



In this example, we’ve used the ‘+’ operator to concatenate two strings, str1 and str2. The result is a new string that combines the contents of str1 and str2, which is then printed to the console.

**Understanding the ‘+’ Operator in Java String Concatenation**

The ‘+’ operator is the most basic tool for string concatenation in Java. It’s simple to use and understand, making it a great starting point for beginners.

Here’s how it works: When you use the ‘+’ operator between two strings, Java combines them into a single string. This operation is known as string concatenation.

Let’s take a look at an example:

A screenshot of a computer program

Description automatically generated

In this example, we’ve used the ‘+’ operator to concatenate the strings str1 and str2. The result is a new string that combines the contents of str1 and str2, which is then printed to the console.

**Efficient Java String Concatenation with StringBuilder and StringBuffer:**

As you dive deeper into Java string concatenation, you’ll encounter more efficient methods, particularly when dealing with larger projects. Two such methods involve using the StringBuilder and StringBuffer classes.

**Understanding StringBuilder and StringBuffer**

StringBuilder and StringBuffer are mutable sequences of characters, meaning they can be modified after they are created. This feature makes them more efficient than the ‘+’ operator when concatenating multiple strings.

Let’s explore an example using StringBuilder:

A screenshot of a computer program

Description automatically generated

In this example, we create a StringBuilder object and use the append method to add strings. The result is a single string that combines the appended strings, which is then printed to the console.

## **Java Methods**

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once and use it many times.

## **Create a Method**

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Create a method inside Main:

public class Main {

static void myMethod() {

// code to be executed

}

}

#### **Example Explained**

* myMethod() is the name of the method
* static means that the method belongs to the Main class and not an object of the Main class. You will learn more about objects and how to access methods through objects later in this tutorial.
* void means that this method does not have a return value. You will learn more about return values later in this chapter

## **Call a Method**

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

In the following example, myMethod() is used to print a text (the action), when it is called:

### **Example**

Inside main, call the myMethod() method:

public class Main {

static void myMethod() {

System.out.println("I just got executed!");

}

public static void main(String[] args) {

myMethod();

}

}

// Outputs "I just got executed!"

A method can also be called multiple times:

### **Example**

public class Main {

static void myMethod() {

System.out.println("I just got executed!");

}

public static void main(String[] args) {

myMethod();

myMethod();

myMethod();

}

}

// I just got executed!

// I just got executed!

// I just got executed!

## **Parameters and Arguments**

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a String called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

### **Example**

public class Main {

static void myMethod(String fname) {

System.out.println(fname + " Refsnes");

}

public static void main(String[] args) {

myMethod("Liam");

myMethod("Jenny");

myMethod("Anja");

}

}

// Liam Refsnes

// Jenny Refsnes

// Anja Refsnes

## **Multiple Parameters**

You can have as many parameters as you like:

### **Example**

public class Main {

static void myMethod(String fname, int age) {

System.out.println(fname + " is " + age);

}

public static void main(String[] args) {

myMethod("Liam", 5);

myMethod("Jenny", 8);

myMethod("Anja", 31);

}

}

// Liam is 5

// Jenny is 8

// Anja is 31

## **Return Values**

The void keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

### **Example**

public class Main {

static **int** myMethod(int x) {

**return** 5 + x;

}

public static void main(String[] args) {

System.out.println(myMethod(3));

}

}

// Outputs 8 (5 + 3)

This example returns the sum of a method's **two parameters**:

### **Example**

public class Main {

static int myMethod(int x, int y) {

return x + y;

}

public static void main(String[] args) {

System.out.println(myMethod(5, 3));

}

}

// Outputs 8 (5 + 3)

You can also store the result in a variable (recommended, as it is easier to read and maintain):

### **Example**

public class Main {

static int myMethod(int x, int y) {

return x + y;

}

public static void main(String[] args) {

int z = myMethod(5, 3);

System.out.println(z);

}

}

// Outputs 8 (5 + 3)

## **A Method with If...Else**

It is common to use if...else statements inside methods:

### **Example**

public class Main {

// Create a checkAge() method with an integer variable called **age**

static void checkAge(int age) {

// If age is less than 18, print "access denied"

if (age < 18) {

System.out.println("Access denied - You are not old enough!");

// If age is greater than, or equal to, 18, print "access granted"

} else {

System.out.println("Access granted - You are old enough!");

}

}

public static void main(String[] args) {

checkAge(20); // Call the checkAge method and pass along an age of 20

}

}

// Outputs "Access granted - You are old enough!"

## **Method Overloading**

With**method overloading**, multiple methods can have the same name with different parameters:

### **Example**

int myMethod(int x)

float myMethod(float x)

double myMethod(double x, double y)

Consider the following example, which has two methods that add numbers of different type:

### **Example**

static int plusMethodInt(int x, int y) {

return x + y;

}

static double plusMethodDouble(double x, double y) {

return x + y;

}

public static void main(String[] args) {

int myNum1 = plusMethodInt(8, 5);

double myNum2 = plusMethodDouble(4.3, 6.26);

System.out.println("int: " + myNum1);

System.out.println("double: " + myNum2);

}

Instead of defining two methods that should do the same thing, it is better to overload one.

In the example below, we overload the plusMethod method to work for both int and double:

### **Example**

static int plusMethod(int x, int y) {

return x + y;

}

static double plusMethod(double x, double y) {

return x + y;

}

public static void main(String[] args) {

int myNum1 = plusMethod(8, 5);

double myNum2 = plusMethod(4.3, 6.26);

System.out.println("int: " + myNum1);

System.out.println("double: " + myNum2);

}

**Lab Tasks:**

**Task 1: Create a calculator using Java Methods. Create separate methods for sum, add, division, multiplication.**

**Task 2: Write a Java program to concatenate your last name after your first name.**

**Task 3: Write a Java program to create a mini hotel management system with the following menu.**

Menu

1. Ice cream 200/-
2. Pasta 500/-
3. Tea 100/-
4. Samosa 100/-

Use Java method to display the menu.

The code should not terminate unless the user presses n key.

Print the total bill in the end user can select multiple items.

After user presses n. Pass the total bill as parameter to a method name taxCal and apply 17% tax on the total bill print the total bill in the end.

**Sample Output:**

Menu

1. Ice cream 200/-
2. Pasta 500/-
3. Tea 100/-
4. Samosa 100/-

What do you want to order?

User input: 1

Total =200

Do you want to order more items?

User Input: y

What do you want to order?

User input: 1

Total=400

Do you want to order more items?

User Input: y

What do you want to order?

User input 3

Total=900

Do you want to order more items?

User Input: n

Your total bill is 1053 with 17% GST

Code Terminates