OOP

Lab Manual (Lab 1)

****

Session: Spring 2024

LAB INSTRUCTOR: AYESHA MAJID ALI

**Objective:**

**Learning basics of Java.**

Java Example

Let's have a quick look at Java programming example. A detailed description of Hello Java example is available in next page.

**Simple.java**

1. **class** Simple{
2. **public** **static** **void** main(String args[]){
3. System.out.println("Hello Java");
4. }
5. }

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

Java Output / Print:

You can use the println() method to output values or print text in Java:

Example:

System.out.println("Hello World!");

**The Print() Method**

There is also a print() method, which is similar to println().

The only difference is that it does not insert a new line at the end of the output:

TASK 1: Write a JAVA program to print your name, session, age, gender on screen.

**Print Numbers:**

You can also use the println() method to print numbers.

However, unlike text, we don't put numbers inside double quotes:

public class Main {

public static void main(String[] args) {

System.out.println(3);

System.out.println(358);

System.out.println(50000);

}

}

You can also perform mathematical calculations inside the println() method:

Example

System.out.println(3 + 3);

System.out.println(2 \* 5);

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line Comments

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code:

public class Main {

public static void main(String[] args) {

// This is a comment

System.out.println("Hello World");

}

}

Java Multi-line Comments

Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

Example

/\* The code below will print the words Hello World

to the screen, and it is amazing \*/

System.out.println("Hello World");

Single or multi-line comments?

It is up to you which you want to use. Normally, we use // for short comments, and /\* \*/ for longer.

Java Variables

Variables are containers for storing data values.

In Java, there are different **types** of variables, for example:

* String - stores text, such as "Hello". String values are surrounded by double quotes
* int - stores integers (whole numbers), without decimals, such as 123 or -123
* float - stores floating point numbers, with decimals, such as 19.99 or -19.99
* char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
* boolean - stores values with two states: true or false

## **Declaring (Creating) Variables**

To create a variable, you must specify the type and assign it a value:

### **Syntax**

*type variableName = value;*

Where type is one of Java's types (such as int or String), and variableName is the name of the variable (such as **x** or **name**). The **equal sign** is used to assign values to the variable.

To create a variable that should store text, look at the following example:

### **Example**

Create a variable called **name** of type String and assign it the value "**John**":

String name = "John";

System.out.println(name);

To create a variable that should store a number, look at the following example:

### **Example**

Create a variable called **myNum** of type int and assign it the value **15**:

int myNum = 15;

System.out.println(myNum);

You can also declare a variable without assigning the value, and assign the value later:

### **Example**

int myNum;

myNum = 15;

System.out.println(myNum);

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

### **Example**

Change the value of myNum from 15 to 20:

int myNum = 15;

myNum = 20; // myNum is now 20

System.out.println(myNum);

## **Final Variables**

If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

### **Example**

final int myNum = 15;

myNum = 20; // will generate an error: cannot assign a value to a final variable

## **Other Types**

A demonstration of how to declare variables of other types:

### **Example**

int myNum = 5;

float myFloatNum = 5.99f;

char myLetter = 'D';

boolean myBool = true;

String myText = "Hello";

# **Java Print Variables**

## **Display Variables**

The println() method is often used to display variables.

To combine both text and a variable, use the + character:

### **Example**

String name = "John";

System.out.println("Hello " + name);

You can also use the + character to add a variable to another variable:

### **Example**

String firstName = "John ";

String lastName = "Doe";

String fullName = firstName + lastName;

System.out.println(fullName)

For numeric values, the + character works as a mathematical [operator](https://www.w3schools.com/java/java_operators.asp) (notice that we use int (integer) variables here):

### **Example**

int x = 5;

int y = 6;

System.out.println(x + y); // Print the value of x + y

From the example above, you can expect:

* x stores the value 5
* y stores the value 6
* Then we use the println() method to display the value of x + y, which is **11**

# **Java Declare Multiple Variables**

## **Declare Many Variables**

To declare more than one variable of the **same type**, you can use a comma-separated list:

### **Example**

Instead of writing:

int x = 5;

int y = 6;

int z = 50;

System.out.println(x + y + z);

You can simply write:

int x = 5, y = 6, z = 50;

System.out.println(x + y + z);

## **One Value to Multiple Variables**

You can also assign the **same value** to multiple variables in one line:

### **Example**

int x, y, z;

x = y = z = 50;

System.out.println(x + y + z);

# **Java Identifiers**

## **Identifiers**

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

### **Example**

// Good

int minutesPerHour = 60;

// OK, but not so easy to understand what **m** actually is

int m = 60;

The general rules for naming variables are:

* Names can contain letters, digits, underscores, and dollar signs
* Names must begin with a letter
* Names should start with a lowercase letter, and cannot contain whitespace
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case-sensitive ("myVar" and "myvar" are different variables)
* Reserved words (like Java keywords, such as int or boolean) cannot be used as names

# **Java Data Types**

## **Java Data Types**

As explained in the previous chapter, a [variable](https://www.w3schools.com/java/java_variables.asp) in Java must be a specified data type:

### **Example**

int myNum = 5; // Integer (whole number)

float myFloatNum = 5.99f; // Floating point number

char myLetter = 'D'; // Character

boolean myBool = true; // Boolean

String myText = "Hello"; // String

Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp) (you will learn more about these in a later chapter)

## **Primitive Data Types**

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| byte | 1 byte | Stores whole numbers from -128 to 127 |
| short | 2 bytes | Stores whole numbers from -32,768 to 32,767 |
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| boolean | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter or ASCII values |

# **Java If ... Else**

## **Java Conditions and If Statements**

You already know that Java supports the usual logical conditions from mathematics:

* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b
* Equal to a == b
* Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

Java has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

### **Syntax**

if (condition1) {

// block of code to be executed if condition1 is true

} else if (condition2) {

// block of code to be executed if the condition1 is false and condition2 is true

} else {

// block of code to be executed if the condition1 is false and condition2 is false

}

# **Java Switch**

## **Java Switch Statements**

Instead of writing **many** if..else statements, you can use the switch statement.

The switch statement selects one of many code blocks to be executed:

### **Syntax**

switch(expression) {

case x:

// code block

break;

case y:

// code block

break;

default:

// code block

}

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.

## **The break Keyword**

When Java reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

## **The default Keyword**

The default keyword specifies some code to run if there is no case match:

### **Example**

int day = 4;

switch (day) {

case 6:

System.out.println("Today is Saturday");

break;

case 7:

System.out.println("Today is Sunday");

break;

default:

System.out.println("Looking forward to the Weekend");

}

// Outputs "Looking forward to the Weekend"

# **Java While Loop**

## **oops**

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

## **Java While Loop**

The while loop loops through a block of code as long as a specified condition is true:

### **Syntax**

while (condition) {

*// code block to be executed*

}

## **The Do/While Loop**

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### **Syntax**

do {

*// code block to be executed*

}

while (condition);

## **Java For Loop**

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

### **Syntax**

for (*statement 1*; *statement 2*; *statement 3*) {

*// code block to be executed*

}

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

## **For-Each Loop**

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an [**array**](https://www.w3schools.com/java/java_arrays.asp):

### **Syntax**

for (type variableName : arrayName) {

*// code block to be executed*

}

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

### **Example**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

**TASKS:**

1. **Create a simple calculator using JAVA. Display a menu to the user from which user can select whether he/she wants to perform addition, subtraction, multiplication, division. Use switch case here. Syntax and usage of switch case is same as that in C++ programming language. Note (Google search how you can take char input in java, in class we have studied how we can take input in int, float, double etc)**
2. **Print your name 100 times using for loop in Java. Make sure you take name as an input from the user. Again, syntax for “for loop” is same in Java.**
3. **Take input from the user and check whether the value that is entered is even or odd. If the number is even multiply it with 10 and print it on screen. If the number is odd find it’s square and print it on screen. Use if else for this. The syntax for if else is again same as that we followed in C++.**

**References:**

[**https://www.w3schools.com/java/**](https://www.w3schools.com/java/)