**Software Design and Architecture**



Lab # 06

Getting started with Java (Basic structure, variables, data types, conditional statements)

Instructor: Mazhar Iqbal

Email: mazhar.iqbal@nu.edu.pk

Course Code: CL1002

Semester Spring 2023

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar Campus

# Basic Structure of a Java Program

public class MyProg

{

public static void main(String[] args)

{

System.out.println(“Hello World”);

}

}

|  |  |
| --- | --- |
| public class Hello | * This creates a class called Hello. * All class names must start with a capital letter. * The public word means that it is accessible from any other classes. |

|  |  |
| --- | --- |
| public static void main | * When the main method is declared public, it means that it can also be used by code outside of its class, due to which the main method is declared public. * The word static used when we want to access a method without creating its object, as we call the main method, before creating any class objects. * The word void indicates that a method does not return a value. main() is declared as void because it does not return a value. * main is a method; this is a starting point of a Java program.   You will notice that the main method code has been moved to some spaces left. It is called indentation which used to make a program easier to read and understand. |
| String[] args | It is an array where each element of it is a string, which has been named as "args". If your Java program is run through the console, you can pass the input parameter, and main() method takes it as input. |
| System.out.println(); | This statement is used to print text on the screen as output, where the system is a predefined class, and out is an object of the PrintWriter class defined in the system. The method println prints the text on the screen with a new line. You can also use print() method instead of println() method. All Java statement ends with a semicolon. The **System.out** object is an instance of the **PrintStream** class, which is a class that provides methods for printing text to a stream. |

## **Run a Java Program through command line**

First you will compile the source code (Java file) into byte code, so use the following line of code:

**javac FileName.java**

Now to convert the bytecode into an executable code, you will use the java command.

**java FileName**

**Java Variables**

* A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.
* Variable is a name of memory location. There are three types of variables in java: local, instance and static.
* There are two types of data types in Java: primitive and non-primitive.
* **Variable** is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.
* **int** data=10;      // Here data is variable

**Data Types in Java**

Data types specify the different sizes and values that can be stored in the variable.

**Data Types tell us:**

i) Type of data

ii) Memory reserved (amount of memory)

iii) Range of value

There are two types of data types in Java:

**1) Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.

**2) Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

# Java Data Types Operators in Java

Operator is a symbol which is used to perform some operation.

Operators are used to perform operations on variables and values.

**Types of operators**

1. Unary operators
2. Binary operators
3. Ternary operators

## Unary Operator

1. Increment (++)
2. Decrement (--)
3. Negation (!)

## Binary Operator

1. Arithmetic (+, -, \*, /,  %)
2. Relational (>, <, >=, <=, !=, ==)
3. Logical (&&, ||)
4. Assignment (=)
5. Arithmetic Assignment operator (+=, -=, \*=, /=, %=)

## Ternary Operator

Conditional operator (?:)

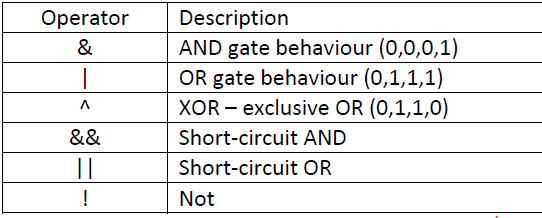
**Example**

   (condition)   **?** statement 1 **:** statement 2;

   int result= (n1>n2) ? n1 : n2;

# Mathematical Operators in Javahttps://lh3.googleusercontent.com/AaPYULUIhNUnSzAmOjzxbPcMCXmaOJiToOcIUFVI8Ry_2X7NSOmQZWEcZ60vr74vxgimS-0xhbTsfc_ijJNc0zvu4iScCnX1-k_XLdSqyptEB1y1aFMt3TH5QzJP0106Z6IDp8hUdcJZTDCeB2Tr

# Logical Operators in Java



# Runtime values

Values given to variable during program execution.

# Scanner Class

Scanner is built in class which is used to get data from users.

**Method in Scanner Class**

1. **nextInt()** To get integer values
2. **nextFloat()** To get float values
3. **nextDouble()** To get double values
4. **next()** To get String values. It will not accept space
5. **nextLine()** To get String values. It will accept space

## Scanner Class object

Scanner obj = new Scanner(System.in);

**Note:** Scanner class is contained in “ java.util ” package of java.

**Import Statement:**

Import java.util.Scanner;

# String Class

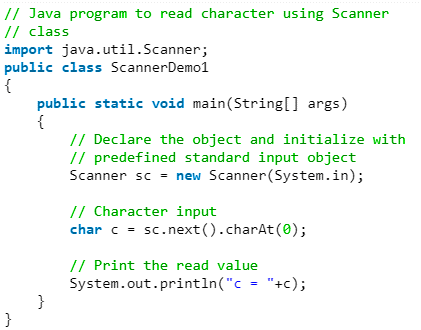
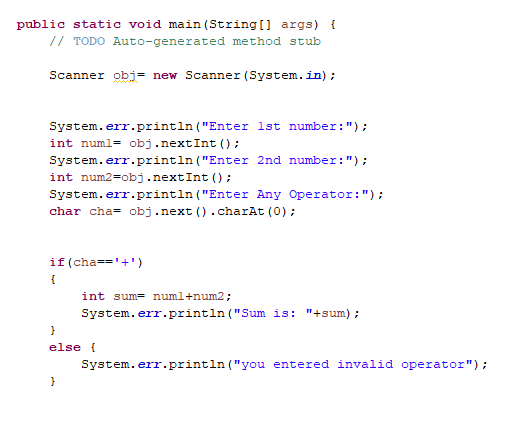
String is predefined class. String is treated as object in java.

**Syntax:**

* String variableName;

## Character in Java

* Scanner class in Java supports nextInt(), nextLong(), nextDouble() etc. **But there is no nextChar()**
* To read a char, we use **next().charAt(0)**. next() function returns the next token/word in the input as a string and **charAt(0)** function returns the first character in that string.
* char letterA = 'A’;
* **Example 1:**

  
  
  
**Example **

# Strings

* A *string* is commonly considered to be a sequence of characters stored in memory and accessible as a unit.
* Strings in java are represented as **objects**.

## String Concatenation

* “+” operator is used to concatenate strings

**For example:** System.out.println(“Hello” + “World”) will print Hello World on console

* String concatenated with any other data type such as int will also convert that datatype to String and the result will be a concatenated String displayed on console.

**For example**

* int i = 4;
* int j = 5;
* System.out.println (“Hello” + i);    // will print Hello 4 on screen

**However**, System.out.println(i+j);    //will print 9 on the console because both i and j are of type int.

## Comparing Strings

**Class:** No Data

**Object:** Data

* The space reserved in memory for class.
* Instance of a class is called object.

Instance--> single occurrence.

A    obj = new A()

if(OXFC00 **==** OXFC08**)**

{

println(“Yes”)

}

else     {

print(“No”)

}

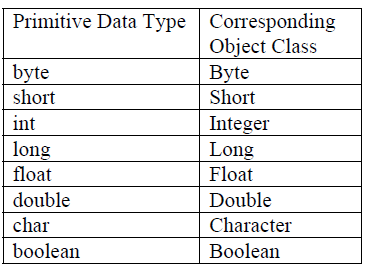
For comparing Strings never use == operator, use *equals* method of String class.

• == operator compares addresses (shallow comparison) while equals compares values (deep comparison)

**E.g.** string1.equals(string2)

**Example Code: String concatenation and comparison**

# Wrapper Classes

Each primitive data type has a corresponding object (wrapper class). These wrapper classes provide additional functionality (conversion, size checking etc.), which a primitive data type cannot provide.

**Wrapper Class**

* You can create an object of Wrapper class using a String or a primitive data type.
* • Integer num = new Integer(4);
* OR
* • Integer num = new Integer(“4”);
* **Note:** num is an object over here not a primitive data type
* You can get a primitive data type from a Wrapper using the corresponding value function
* • int primNum = num.intValue();

## Converting Strings to Numeric Primitive Data Types

* To convert a string containing digits to a primitive data type, wrapper classes can help.
* *parseXxx* method can be used to convert a String to the corresponding primitive data type.
* • String value = “532”;
* int d = Integer.parseInt(value);
* • String value = “3.14e6”;

double d = Double.parseDouble(value);

# Decision Making Statements OR Decision Control Statements

* Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
* Used to change the flow.
* In these statements (conditions) order or sequence of the statements are changed.

## If Statement

**if statement** will execute or skip or ignore a block of code depending one condition.

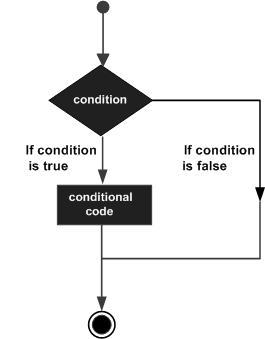
**Syntax:**

if(conditional expression)

{

 // Statements will execute if the Boolean expression is true

}



**Example**:

public class Test {

 public static void main(String args[]) {

int x = 10;

 if( x < 20 )

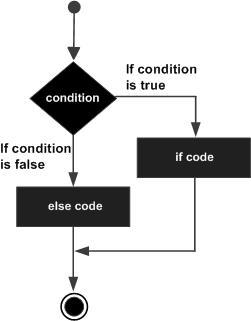
{

  System.out.print("This is if statement");

}   }    }

## if...else statement

* Used for making two-way decision.
* It will execute if block if condition is true and will execute another block (else block) if condition is false.
* It will take one action if the condition is true and take another action if condition is false.

if(condition) {

 // Executes this block when the Boolean expression is true

  }

else {

// Executes this block when the Boolean expression is false

}

**Example**:

int x = 30;

if( x < 20 ) {

 System.out.print("This is if statement");

}

else

{

System.out.print("This is else statement");

 }

Nested if statement

You can use one **if** or **else if** statement inside another **if** or **else if** statement(s)

public class Test {

public static void main(String args[]) {

 int x = 30;

 int y = 10;

if( x == 30 ) {

if( y == 10 ) {

System.out.print("X = 30 and Y = 10");

}

}

}

}

## Multiple if statement

* This statement is used when we have to check multiple conditions.
* Used for multiple way decision making.

**Syntax:**

       if(condition)

{     statements(s);    }

        if(condition)

{     statements(s);    }

        if(condition)

{     statements(s);    }

## if else if ladder statement

* This statement is used to check multiple conditions.
* Used to execute one condition from multiple statements.
* **Syntax:**

       if(condition)

{     statements(s);    }

        else if(condition)

{     statements(s);    }

        else if(condition)

{     statements(s);    }

         else    {     statements(s);    }

## if else if ladder statement VS multiple if

In multiple if statement all conditions are checked due to which compiler is bored and it is difficult task for compiler.

That’s why we are using if else if ladder statement instead of multiple if.

**Note:**

1. check boxes is the example of multiple if

2) Radio buttons are the example of if else if ladder statement.

## 6) Conditional Operator (? :)

* It is ternary operator and work on three operands.
* It works like if else statement.
* **Syntax:**

       (condition)   ? statement 1 : statement 2;

Statement must be single and it is the limitation of conditional operator.

**Example:**

int n1=4;

int n2=5;

int result= (n1>n2) ? n1 : n2;

print(result);

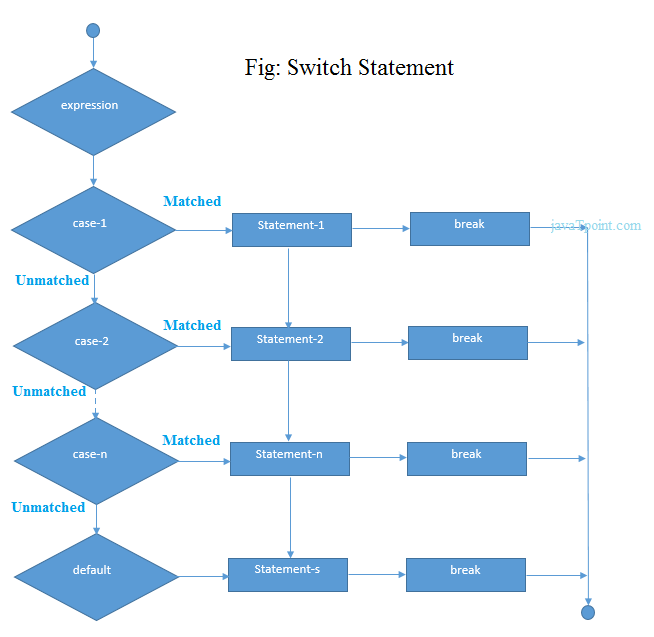
## Logical Operators

Used for compound condition or expression

1. AND (&&)
2. OR (||) pip sign

## Switch statement

* Used when multiple choices are given and one is to be selected. It is like if else if ladder statement. Used to select one several actions based on the value of variable or expression.
* **Switch case statement** is used when we have number of options (or choices) and we may need to perform a different task for each choice.



**Syntax**:

switch(variable/expression)

{

case value 1:

Statement(s);   // code to be executed

break;

case value 2:

Statement(s);   // code to be executed

break;

**Syntax**:

.

.

.

case value n:

Statement(s);      // code to be executed

break;

default:

Statement(s);      // code to be executed if all cases are not matched

}

**Example**

int num=2;

switch(num+2) {

case 1:

System.out.println("Case1: Value is: "+num);

break;

case 2:

System.out.println("Case2: Value is: "+num);

break;

case 3:

  System.out.println("Case3: Value is: "+num);

break;

default:

System.out.println("Default: Value is: "+num);

}

## Switch statement VS if else if

* If a program contains conditions or compound conditions then we use if else if
* If program contains single variable or expression then we use switch statement.

**Compound conditions**

1. if(a>b && a>c)
2. if)(a>b || a>c)

## Java Switch statement is fall through

It means it executes all statements after match if break statement is not used with switch cases.

**Example:**

alphabet= input.nextLine():

switch(alphabet)

{

case “a” :

case “e” :

case “i” :

case “o” :

case “u” :

case “A” :

case “E” :

case “I” :

case “O” :

case “U” :

System.out.println(“You entered vowel”);

break;

default:

System.out.println(“You entered consonant”);

}   // switch body Closed

**Break Statement**

* The break statement is used to exit from the body of the switch structure or loop structure.
* The break statement terminates the execution of the loop when it is used inside the body of the loop.

**Syntax:** break;

## boolean variable example

Scanner input= new Scanner(System.in);

int number;

System.out.println(“Enter any number”);

number = input.nextInt();

boolean even;

even = (n%2==0);

if(even)

{ System.out.println(“Even number”) }

else

{

System.out.println(“Odd Number”);

}