

# JAVA Simplicia

1<sup>st</sup> Edition, Sudipta Kumar Das

```

JJJJJJJJJJ      AAA      VVVVVVVV      VVVVVVVV      AAA
J:~::~:~::~J      A::~A      V::~:~::~V      V::~:~::~V      A::~A
J:~::~:~::~J      A::~:A      V::~:~::~V      V::~:~::~V      A::~:A
JJ:~::~:~::~JJ      A::~:~::~A      V::~:~::~V      V::~:~::~V      A::~:~::~A
J:~::~J      A::~:~::~A      V::~:~::~V      V::~:~::~V      A::~:~::~A
J:~::~J      A::::A::::A      V::~:~::~V      V::~:~::~V      A::::A::::A
J:~::~J      A::::A      A::::A      V::~:~::~V      V::~:~::~V      A::::A      A::::A
J:~::~j      A::::A      A::::A      V::~:~::~V      V::~:~::~V      A::::A      A::::A
J:~::~J      A::::A      A::::A      V::::V V::::V      A::::A      A::::A
JJJJJJJJ      J:~::~J      A::::A~~~~~A::::A      V::::V V::::V      A::::A~~~~~A::::A
J:~::~J      J:~::~J      A::::A~~~~~A::::A      V::::V~~~~~V::::V      A::::A~~~~~A::::A
J:~::~J      J:~::~J      A::::A~~~~~A~~~~~A::::A      V::::V~~~~~V::::V      A::::A~~~~~A~~~~~A::::A
JJ:~::~:~::~JJ      J:~::~J      A::::A~~~~~A~~~~~A::::A      V::::V~~~~~V::::V      A::::A~~~~~A~~~~~A::::A
JJ:~::~:~::~JJ      JJ:~::~:~::~JJ      A::::A      A::::A      V::::V~~~~~V::::V      A::::A      A::::A
JJ:~::~:~::~JJ      JJ:~::~:~::~JJ      A::::A      A::::A      V::::V      VVV      A::::A      A::::A
JJJJJJJJJJ      AAAAAAA      AAAAAAA      VVV      AAAAAAA      AAAAAAA

```

SIMPLICA

# Contents

<b>I</b>	<b>Introduction</b>	<b>7</b>
	Preface	9
<b>1</b>	<b>History of JAVA</b>	<b>10</b>
<b>II</b>	<b>Pre-Basic of JAVA</b>	<b>12</b>
<b>2</b>	<b>Package &amp; Class Declaration</b>	<b>14</b>
2.1	package . . . . .	14
2.1.1	Syntax . . . . .	14
2.1.2	Example . . . . .	14
2.2	Access modifiers . . . . .	14
2.2.1	Public . . . . .	14
2.2.2	Private . . . . .	15
2.2.3	Protected . . . . .	15
2.2.4	Default . . . . .	15
2.2.5	Syntax . . . . .	15
2.2.6	Example . . . . .	15
2.3	Class Declaration . . . . .	15
2.3.1	Syntax . . . . .	15
2.3.2	Example . . . . .	15
2.4	Main Method . . . . .	16
2.4.1	Syntax . . . . .	16
2.4.2	Example . . . . .	16
2.5	Show Output in JAVA . . . . .	16
2.5.1	Syntax . . . . .	16
2.5.2	Example . . . . .	16
<b>3</b>	<b>Escape Sequence &amp; Format Specifier</b>	<b>18</b>
3.1	Escape Sequence . . . . .	18
3.1.1	Syntax . . . . .	18
3.1.2	Example . . . . .	18
3.2	Format Specifier . . . . .	19
3.2.1	Syntax . . . . .	19

3.2.2	Example . . . . .	19
3.3	Comments . . . . .	20
3.3.1	Single Line Comments . . . . .	21
3.3.2	Multi Line Comments . . . . .	21
3.3.3	Documentation Comments . . . . .	21
3.3.4	Syntax . . . . .	21
3.3.5	Example . . . . .	22
3.4	User Input from Console in JAVA . . . . .	22
3.4.1	Syntax . . . . .	22
3.4.2	Example . . . . .	23

### **III Basics of JAVA 24**

#### **4 Variables and Data Types 26**

4.1	Variables . . . . .	26
4.1.1	Variable Declaration . . . . .	26
4.1.2	Variable Initialization . . . . .	26
4.1.3	Syntax . . . . .	26
4.1.4	Example . . . . .	26
4.1.5	Rules to write Variables & Functions Name . . . . .	27
4.2	Data Types . . . . .	27
4.2.1	Syntax . . . . .	29
4.2.2	Example . . . . .	29

#### **5 Operators 30**

5.1	Arithmetic Operators . . . . .	31
5.2	Assignment Operators . . . . .	31
5.3	Unary Operators . . . . .	31
5.3.1	Prefix . . . . .	32
5.3.2	Postfix . . . . .	32
5.4	Relational Operators . . . . .	32
5.5	Bitwise Operators . . . . .	33
5.6	Logical Operators . . . . .	33
5.7	Ternary Operators . . . . .	33
5.8	Shift Operators . . . . .	33

#### **6 Control Statements 35**

6.1	Conditional Statements . . . . .	35
6.1.1	If-Else . . . . .	36
6.1.2	Example . . . . .	37
6.1.3	If, Else If, Else . . . . .	37
6.1.4	Example . . . . .	38
6.1.5	Switch . . . . .	38
6.1.6	Example . . . . .	40
6.2	Looping Statements . . . . .	42
6.2.1	For Loops . . . . .	42

6.2.2	While Loops . . . . .	42
6.2.3	Do-While Loops . . . . .	44
6.3	Jump Statements . . . . .	45
6.3.1	Break . . . . .	45

# List of Figures

1.1	James Gosling[2]	11
2.1	Show Output in JAVA[3]	17
3.1	Escape Sequences[4][3]	19
3.2	Format Specifier[5][3]	21
3.3	Comments[3]	22
3.4	User Input[7][3]	23
4.1	Variable[3]	27
4.2	Data Types[6]	28
4.3	Data Type Chart[3]	29
5.1	Operators[8]	30
6.1	Control Statements[3]	35
6.2	If-Else Condition	37
6.3	If-Else Condition	38
6.4	Vending Machine[10]	39
6.5	Switch[3]	41
6.6	For Loop	43
6.7	While Loop[3]	43
6.8	Do While Loop[3]	44

# List of Tables

3.1	Escape Sequences . . . . .	18
3.2	Format Specifier . . . . .	20
3.3	User Input Type . . . . .	22
4.1	Data Type . . . . .	28
5.1	Arithmetic Table . . . . .	31
5.2	Assignment Operators . . . . .	31
5.3	Unary Operators . . . . .	31
5.4	Prefix Operators . . . . .	32
5.5	Postfix Operators . . . . .	32
5.6	Relational Operators . . . . .	32
5.7	Bitwise Operators . . . . .	33
5.8	Logical Operators . . . . .	33
5.9	Ternary Operators . . . . .	33
5.10	Shift Operators . . . . .	34

**Part I**

**Introduction**

JAVA[1] is a Programming language which is used mostly in official softwares because of it's strong security system. It is a high-level language which uses JVM to convert the high-level code to a machine code. It is one of the most popular programming languages out there. Released in 1995 and still widely used today. Java has many applications, including software development, mobile applications, and large systems development. Knowing Java opens a lot of possibilities for us as a developer.



# Preface

JAVA[1] knowledge is vast. People most often have to go through most of the documentations of the JAVA code then they could think of writing something. Moreover, sometimes people loses their interest in learning JAVA or writing their codes in JAVA. So in that case they just give online posts and hire outworkers to complete their school/college projects, homeworks and others. This process's is both insecure and costly. In this book I just tried to teach JAVA in a simple way and by which people can start doing their school/college projects, homeworks and others by their own, having simple knowledge. Thus, they can learn the vast knowledge slowly and more interesting way.

# Chapter 1

## History of JAVA

Java[1] was originally developed by James Gosling[2] at Sun Microsystems and released in May 1995 as a core component of Sun Microsystems' Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun had relicensed most of its Java technologies under the GPL-2.0-only license. Oracle offers its own HotSpot Java Virtual Machine, however the official reference implementation is the OpenJDK JVM which is free open-source software and used by most developers and is the default JVM for almost all Linux distributions.

As of March 2022, Java 18 is the latest version, while Java 17, 11 and 8 are the current long-term support (LTS) versions. Oracle released the last zero-cost public update for the legacy version Java 8 LTS in January 2019 for commercial use, although it will otherwise still support Java 8 with public updates for personal use indefinitely. Other vendors have begun to offer zero-cost builds of OpenJDK 18 and 8, 11 and 17 that are still receiving security and other upgrades.



Figure 1.1: James Gosling[2]

# **Part II**

## **Pre-Basic of JAVA**

JAVA is a vast programming language, but it has some pre basic things, on which the whole language depends on. In this part we'll going to discuss it.

## Chapter 2

# Package & Class Declaration

### 2.1 package

Package is kind of a folder, where all the class files are present. We can use them by using the keyword *import packageName.subPackageName.className* or *import packageName.\**. Here \* means all the things. we can use predefined packages of jdk or we can also import our own packages in any class from another folder.

#### 2.1.1 Syntax

```
import packageName.subPackageName.className
```

#### 2.1.2 Example

```
java.io.File;
```

### 2.2 Access modifiers

Access modifiers basically used to control the access of the variables & methods from another class or package. It is mostly used in Encapsulation. There are basically 4 Access modifiers. Those are,

- Public
- Private
- Protected
- Default

#### 2.2.1 Public

Public Keyword is used to make the variables and methods Public that means those things can be accessed from anywhere, no matter where it is.

### 2.2.2 Private

Private Keyword is used to make the variables and methods inaccessible that means those thing can be access from nowhere, no matter where it is.

### 2.2.3 Protected

Protected Keyword is used to make the variables and methods only accessible from their children that means those thing can be access from nowhere except its child class, no matter where it is. IF a class is extended by another class then the class who extend in it, called child class of the class who got extended by the child class. And that class who got extended by the child class called parent Class.

### 2.2.4 Default

We don't need any access modifiers to make it default access. Default access is kind of private access modifier. Default access means that variable/methods can be accessible from anywhere inside the folder its in. And can not be accessible outside of the folder.

### 2.2.5 Syntax

*Access\_modifier dataType/returnType variableName/methodName()*

### 2.2.6 Example

```
public boolean isAccessible = true;
private String name = "Sudipta Kumar Das";
protected String carModel = "Toyota CHR";
int age = 22; \\ This is Default Access Modifier
```

## 2.3 Class Declaration

JAVA is an Object Oriented Programming(OOP) Language. Here we have to use lots of classes. To use classes we have to declare it. Class declaration has its own syntax

### 2.3.1 Syntax

*Access\_modifier class className*

### 2.3.2 Example

```
public class Mobile{

}
```

## 2.4 Main Method

JAVA is a high level language. It needs a compiler to convert the high level code into machine code. The compilers need to understand the starting point of the code conversion. Main method is the place from where the compilers start reading and start compiling. There should be only one main method for entire program or project. Classes can be many but main method must be one. main method is declared inside any one class.

### 2.4.1 Syntax

```
public static void main(String[] args){}
```

### 2.4.2 Example

```
public class Test{  
    public static void main(String[] args){  
  
    }  
}
```

## 2.5 Show Output in JAVA

We use *System.out.println()*; to print anything or show anything on console. Here println means print a newline also. That means the line will break and go to a new line after showing the output inside first bracket.

### 2.5.1 Syntax

```
System.out.println();
```

### 2.5.2 Example

```
public class Test{  
    public static void main(String[] args){  
        System.out.println("HELLO WORLD !");  
    }  
}
```





Figure 2.1: Show Output in JAVA[3]

## Chapter 3

# Escape Sequence & Format Specifier

### 3.1 Escape Sequence

Escape sequences[4] are some special characters who performs some special kinds of works on showing console output as like printing a backslash or a new line. Escape sequences are written after a backslash indicating it is a special character. And it is been written inside double quote marks("").

Escape Sequence	Meaning
<code>\b</code>	Backspace
<code>\t</code>	Tab (4 spaces at right)
<code>\n</code>	New Line/Break Line
<code>\r</code>	Carriage Return/ Break line & start from the left most after this line
<code>\"</code>	Print Double quote mark on console
<code>'</code>	Print Single quote mark on console
<code>\f</code>	Insert a form feed in the text at this point.
<code>\\</code>	Print Backslash on console

Table 3.1: Escape Sequences

#### 3.1.1 Syntax

`"\escapeCharacter"`

#### 3.1.2 Example

```
public class Test {
```

```

public static void main(String args[]) {
    System.out.println("HELLO\b WORLD !");
    System.out.println("HELLO\t WORLD !");
    System.out.println("HELLO\n WORLD !");
    System.out.println("HELLO\r WORLD !");
    System.out.println("HELLO \"WORLD\" !");
    System.out.println("HELLO \'W\'ORLD !");
    System.out.println("HELLO\f WORLD !");
    System.out.println("HELLO \\WORLD !");
}
}

```

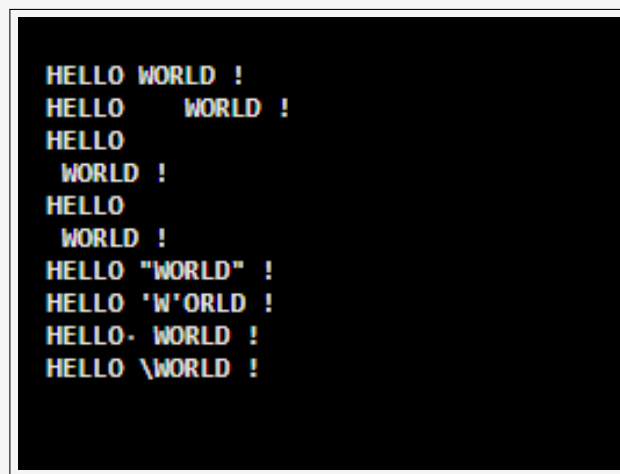


Figure 3.1: Escape Sequences[4][3]

## 3.2 Format Specifier

Format Specifier[5] is used to indicate the place where the value of a variable should appear in a string. That means sometimes we have to show the output inside a line, as like. Hii! I am {age} year old. here we want age = 22 or something just like that. So we'll write `System.out.println("Hii! I am %d year old.",age);` Here output will be if age = 22, Hii! I am 22 year old.

### 3.2.1 Syntax

*"%formatSpecifier"*

### 3.2.2 Example

```
public class Test {
```

Format Specifier	Usual Variable Type	Display As
%f%f	float or double	Signed Decimal
%o	int	unsigned Octal value
%u	int	unsigned Integer
%x	int	unsigned Hex value
%H	int	unsigned Decimal Integer
%S	array of char	Sequence of Characters
%%	-	Inserts a % sign
%f	float	Decimal floating-point
%e%E	-	Scientific Notation / Exponential Format
%g	-	Causes formatter to use either %f or %e which one is shorter
%h%H	-	Hash code of the Argument
%d		Decimal Integer
%c		Character
%b%B	boolean	Boolean
%a%A	-	Floating Point hexadecimal

Table 3.2: Format Specifier

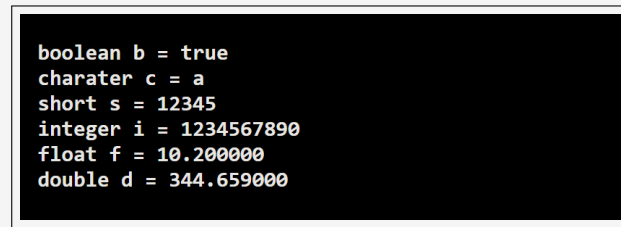
```

public static void main(String args[]) {
    int i = 1234567890;
    boolean b = true;
    char c = 'a';
    short s = 12345;
    float f = 10.2f;
    double d = 344.659;
    System.out.printf("boolean b = %b\n",b);
    System.out.printf("character c = %c\n",c);
    System.out.printf("short s = %d\n",s);
    System.out.printf("integer i = %d\n",i);
    System.out.printf("float f = %1f\n",f);
    System.out.printf("double d = %3f\n",d);
}
}

```

### 3.3 Comments

Comments are basically side notes, means the thing that is only needed for programmers not the endusers. Naturally programmers use comments to explain what the code is doing to himself or to other programmers. Sometimes the codes are too big and it becomes really very hard to understand what a specific portion of code is doing. On that position, comments help to understand the workflow



```
boolean b = true
charater c = a
short s = 12345
integer i = 1234567890
float f = 10.200000
double d = 344.659000
```

Figure 3.2: Format Specifier[5][3]

as all the codes looks like kind of same. These comments do no appear on output There are 3 kinds of comments in JAVA. Those are,

- Single line comments
- Multi line comments
- Documentation comments

### 3.3.1 Single Line Comments

This comments contains just one line. This kinds of comments are used by using just double forward slashes(`//`).

### 3.3.2 Multi Line Comments

This comments contains just as many as line we take. This kinds of comments started with just one forward slash and one star(`/*`) and ends with one star and one forward slash(`*/`)

### 3.3.3 Documentation Comments

This comments contains just as many as line we take. But these kinds of comments are used for documentation purpose only. This kinds of comments started with just one forward slash and two star(`/**`) and ends with one star and one forward slash(`*/`)

### 3.3.4 Syntax

- Single Line Comments  $\rightarrow$  `//Comments`
- Multi Line Comments  $\rightarrow$  `/*Comments*/`
- Documentation Comments  $\rightarrow$  `/**Comments*/`

### 3.3.5 Example

```
public class Test {
    public static void main(String args[]) {
        System.out.println("No Comments");
        //System.out.println("Single Line Comment");
        /*System.out.println("Multi Line Comment");*/
        /** System.out.println("Documentation Comment");*/
    }
}
```

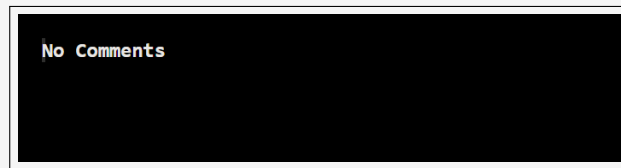


Figure 3.3: Comments[3]

## 3.4 User Input from Console in JAVA

A program is successful when it can take user inputs[7] and perform their task based on it appropriately. So, in that case, we have to take user inputs. For this we have to declare an object of Scanner class. In short, we have to write this line must *Scanner input = new Scanner(System.in);*. And to take input we have to use *input.nextDataType()*, here if we want to take integer, then we have to use *input.nextInt()*;

Method	Description
<i>nextBoolean()</i>	Reads Boolean values
<i>nextByte()</i>	Reads byte value
<i>nextDouble()</i>	Reads double value
<i>nextFloat()</i>	Reads float value
<i>nextInt()</i>	Reads int value
<i>nextLine()</i>	Reads String value
<i>nextLong()</i>	Reads long value
<i>nextShort()</i>	Reads short value
<i>next().charAt(0)</i>	Reads Char value

Table 3.3: User Input Type

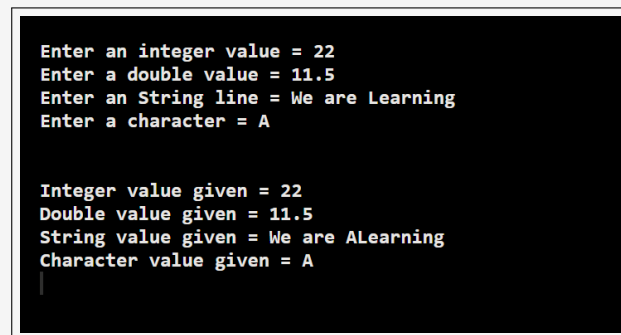
### 3.4.1 Syntax

```
Scanner input = new Scanner(System.in); variableName = input.nextDataType();
```

### 3.4.2 Example

```
import java.util.Scanner;

public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        int i;
        double d;
        String s;
        char c;
        System.out.print("Enter an integer value = ");
        i = input.nextInt();
        System.out.print("Enter a double value = ");
        d = input.nextDouble();
        input.nextLine();
        System.out.print("Enter an String line = ");
        s = input.nextLine();
        System.out.print("Enter a character = ");
        c = input.next().charAt(0);
        System.out.println();
        System.out.println();
        System.out.println("Integer value given = "+i);
        System.out.println("Double value given = "+d);
        System.out.println("String value given = "+s);
        System.out.println("Character value given = "+c);
    }
}
```

A screenshot of a Java program execution in a terminal window. The program prompts the user for four inputs: an integer, a double, a string, and a character. The user enters '22' for the integer, '11.5' for the double, 'We are Learning' for the string, and 'A' for the character. The program then prints the values back to the user, confirming the input for each variable.

```
Enter an integer value = 22
Enter a double value = 11.5
Enter an String line = We are Learning
Enter a character = A

Integer value given = 22
Double value given = 11.5
String value given = We are ALearning
Character value given = A
```

Figure 3.4: User Input[7][3]

# Part III

## Basics of JAVA



JAVA is a vast programming language, but it has some basic things too, on which the whole language also depends on. In this part we'll going to discuss those things.

## Chapter 4

# Variables and Data Types

### 4.1 Variables

Variables means a place where we store some data. As like if we want to store water, then we'll take a pot like bottles. Variables are like similar pots but we store data here. To write variables, we need dataTypes.

#### 4.1.1 Variable Declaration

Variable declaration means just declare where the data we want to store, but not store at the same time. we should store later there.

#### 4.1.2 Variable Initialization

Variable initialization means store values in variables which has already been declared previously by us. After that, we can store later there.

#### 4.1.3 Syntax

*accessModifier dataType variableName; // Variable Declaration*

*accessModifier dataType variableName = value; // Variable Initialization*

#### 4.1.4 Example

```
public class Test {
    public static void main(String args[]) {
        String name ; // Variable Declaration
        int age = 19; // Variable Initialization

        name = "Ritu Das";
        System.out.println("Name = "+name);
        System.out.println("Age = "+age);
    }
}
```

```
}  
}
```



```
Name = Ritu Das  
Age = 19
```

Figure 4.1: Variable[3]

#### 4.1.5 Rules to write Variables & Functions Name

- We can use Alphabets both Capital Letter(A-Z) & Small Letter(a-z) for variable name.
- We can use Numerical values ( $0 \rightarrow 9$ ), Underscore(\_) & Dollar Sign(\$) for variable name.
- Variable names can not be started with Numerical values ( $0 \rightarrow 9$ ) or any character except Alphabets both Capital Letter(A-Z) & Small Letter(a-z).
- Any keyword can not be a variable name.
- There can not be any spaces inside a variable/function name.
- We can use maximum 31 characters for the name of variables/functions. But using maximum 8 characters is standard.

## 4.2 Data Types

Data Types[6] are basically types of pots that are used to store data inside it. As an example, if we want to store a football we can just use net(Not Internet) bags. But if we want to store water then we'll need bottles. Data Types are same. if we want to store integer numbers then we have to use integer type of variable not the boolean or another type.

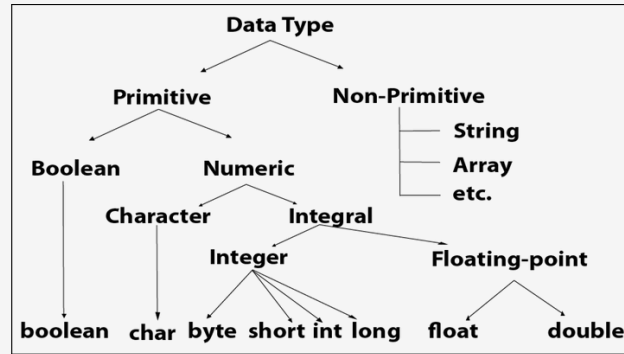


Figure 4.2: Data Types[6]

Type Name	Description	Size	Range	Simpel Declaration & Initialization
boolaen	true or false	1 Bit	{true,false}	boolean x = true;
char	Unicode Character	2 Byte	u0000 to uFFFF	char x = 'a';
byte	Signed Integer	1 Byte	-128 to 127	byte x = 12;
short	Signed Integer	2 Byte	-32768 to 32767	short x = 12345;
int	Signed Integer	4 Byte	-2147483648 to 2147483647	int x = 123456
long	Signed Integer	8 Byte	-9223372036854775808 to 9223372036854775807	long x = 0;
float	IEEE 754 floating point	4 Byte	$\pm 1.4\text{E-}45$ to $\pm 3.44028235\text{E+}38$	float x = 10.2f
double	IEEE 754 floating point	8 Byte	$\pm 4.9\text{E-}324$ to $\pm 1.7976931348623157\text{E+}308$	double x = 21.3

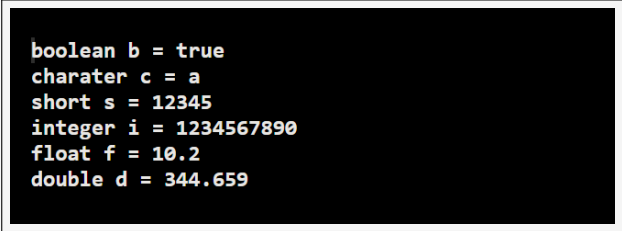
Table 4.1: Data Type

### 4.2.1 Syntax

*accessModifier dataType variableName;*

### 4.2.2 Example

```
public class Test {  
    public static void main(String args[]) {  
        int i = 1234567890; //10 Digit storable, shouldn't put 0 at first  
        boolean b = true;   // Only true/false or 1/0 allowed  
        char c = 'a';       // 1 character storable at a time & single quote must  
        short s = 12345;     // 5 digits storable  
        float f = 10.2f;     // We have to put f at the edge of the value for float  
        double d = 344.659; // JAVA's default decimal type is double  
        System.out.println("boolean b = "+b);  
        System.out.println("character c = "+c);  
        System.out.println("short s = "+s);  
        System.out.println("integer i = "+i);  
        System.out.println("float f = "+f);  
        System.out.println("double d = "+d);  
    }  
}
```



```
boolean b = true  
charater c = a  
short s = 12345  
integer i = 1234567890  
float f = 10.2  
double d = 344.659
```

Figure 4.3: Data Type Chart[3]

## Chapter 5

# Operators

Operators[8] are some special characters which performs some special tasks like data assignation addition subtraction or decides equal or not greater or not. There are 8 kinds of operators. Those are,

- Arithmetic Operators
- Assignment Operators
- Unary Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Shift Operators
- Ternary Operators

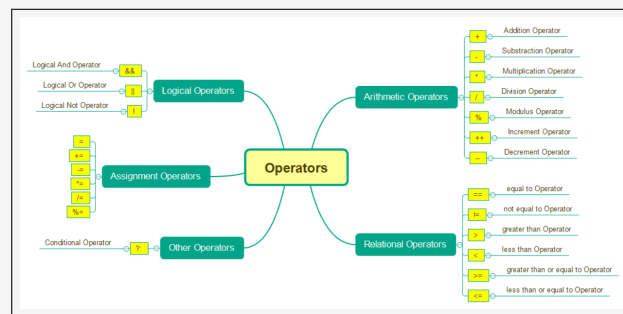


Figure 5.1: Operators[8]

## 5.1 Arithmetic Operators

Arithmetic operators[8] are those we use for arithhmatc operations like addition, subtraction, multiplication etc.

Operator	Task	Example	Output
+	Addition	X=15+6	X=21
-	Subtraction	X=15-6	X=9
*	Multiplication	X=15*6	X=90
/	Division	X=15/6	X=2
%	Modulus	X=15%6	X=3

Table 5.1: Arithmetic Table

## 5.2 Assignment Operators

Assignment operators[8] are those we use for assigning values into variables like Equal to etc.

Operator	Example	Full Form
=	y=x+5	y=x+5
+=	x+=5	x=x+5
-=	x-=5	x=x-5
=	x*=5	x=x*5
/=	x/=5	x=x/5

Table 5.2: Assignment Operators

## 5.3 Unary Operators

Unary operators[8] are also called single operators. I means these kinds of operators need just one variable to perform their tasks. Unary operators are also 2 kinds. those

Unary Operator	Meaning
+	Unary Plus
-	Unary Minus
++	Increment
-	Decrement

Table 5.3: Unary Operators

are,

- Prefix
- Postfix

### 5.3.1 Prefix

These kinds of operators[8] increment/decrement their value first then perform their tasks.

Unary Operator	Meaning
<code>++expr</code>	Increment First
<code>--expr</code>	Decrement First

Table 5.4: Prefix Operators

### 5.3.2 Postfix

These kinds of operators[8] perform their task first then they increment/decrement their value.

Unary Operator	Meaning
<code>expr++</code>	Increment Later
<code>expr--</code>	Decrement Later

Table 5.5: Postfix Operators

## 5.4 Relational Operators

These kinds of operators[8] are needed to create relations between 2 variables. As like which one is greater or smaller between 2 operators etc.

Operator	Use	Description
<code>&gt;</code>	<code>Op1&gt;Op2</code>	Greater Than
<code>&gt;=</code>	<code>Op1&gt;=Op2</code>	Greater Than Equal
<code>&lt;</code>	<code>Op1&lt;Op2</code>	Less Than
<code>&lt;=</code>	<code>Op1&lt;=Op2</code>	Less Than Equal
<code>==</code>	<code>Op1==Op2</code>	Both are Equal
<code>!</code>	<code>Op1!=Op2</code>	Are not Equal

Table 5.6: Relational Operators



## 5.5 Bitwise Operators

Bitwise operators[8] are used to performing the manipulation of individual bits of a number. They can be used with any integral type (char, short, int, etc.). They are used when performing update and query operations of the Binary indexed trees.

Operator	Description
&	Bitwise AND
^	Bitwise Exclusive OR
	Bitwise Inclusive OR

Table 5.7: Bitwise Operators

## 5.6 Logical Operators

Logical operators[8] are used to check whether an expression is true or false . They are used in decision making.

Operators	Description
&&	Logical AND
	Logical OR

Table 5.8: Logical Operators

## 5.7 Ternary Operators

The Java ternary operator[8] lets us write an if statement on one line of code. A ternary operator can either evaluate to true or false. It returns a specified value depending on whether the statement evaluates to true or false. We use Java if...else statements to control the flow of a program.

Operators	Syntax	Example
? :	x<=y?true:false	x<=y?System.out.println("X bigger") : System.out.println("Y bigger")

Table 5.9: Ternary Operators

## 5.8 Shift Operators

The shift operator[8] is used when we're performing logical bits operations, as opposed to mathematical operations. It can be used for speed, being significantly faster than division/multiplication when dealing with operands that are powers of two, but clarity of code is usually preferred over raw speed.

Operators	Description
<<	Left Shift
>>	Right Shift
>>>	Changes parity bit (MSB) to 0 For Negative Numbers

Table 5.10: Shift Operators

## Chapter 6

# Control Statements

To know about control statements, we have know about the statements first. So, what is an statement? Any meaningful expression is called a statement. There must a semicolont after each and every statement finishes.

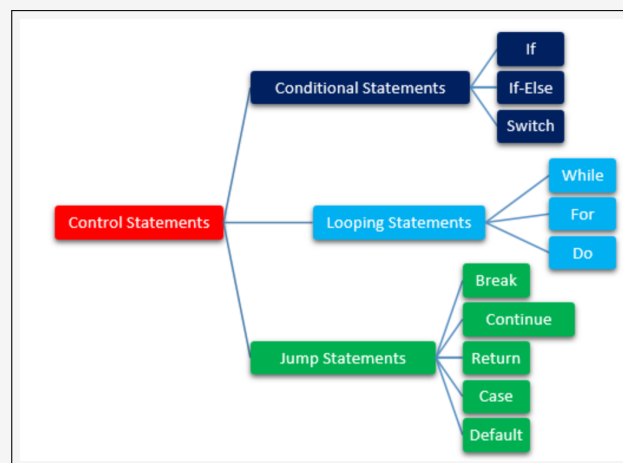


Figure 6.1: Control Statements[3]

### 6.1 Conditional Statements

We, the programmers need conditional statements the most to control the flow of the program. Conditional statements gives machines the ability to take decisions depends On the situations. Those decisions are already pre-built by us but the program will execute the right decision at the right situation. As an example, we can say that if he accepts all the terms and conditions then confirm the deal and give a call, otherwise just cancel the deal and shutdown. Now it depends on the customer, if he accepts or

not. But the computer knows what he should do after the decision customer takes. These kinds of tasks also can be done by computers using those conditional statements. Conditional statements have 3 sub-parts. Those are,

- if-else
- if-else if-else
- switch

### 6.1.1 If-Else

In this statement there will be atleast one `if(condition){}`. And atmost one `else{}`. It will be perfectly alright if we do not use else here.

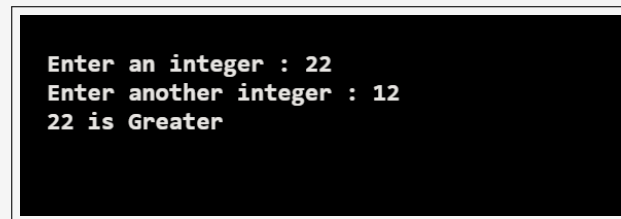
#### Syntax

```
if(condition){statements;}  
    else{statements;}
```

### 6.1.2 Example

```
import java.util.Scanner;

public class Test{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        int x, y;
        System.out.print("Enter an integer : ");
        x = input.nextInt();
        System.out.print("Enter another integer : ");
        y = input.nextInt();
        if (y != x){
            if (x < y){
                System.out.println(y + " is Greater");
            }else{
                System.out.println(x + " is Greater");
            }
        } // We didn't use else here. But it's Working
    }
}
```



```
Enter an integer : 22
Enter another integer : 12
22 is Greater
```

Figure 6.2: If-Else Condition

### 6.1.3 If, Else If, Else

In this statement there will be atleast one `if(condition){}`. And atmost one `else{}`. There can be multiple `else if(condition){}`. It will be perfectly alright if we do not use `else` here.

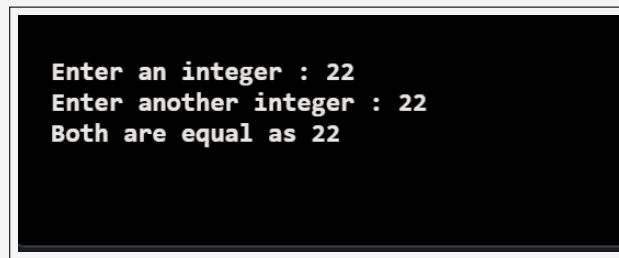
#### Syntax

```
if(condition){statements;}
else if(condition){statements;}
else if(condition){statements;}
else{statements;}
```

### 6.1.4 Example

```
import java.util.Scanner;

public class Test{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        int x, y;
        System.out.print("Enter an integer : ");
        x = input.nextInt();
        System.out.print("Enter another integer : ");
        y = input.nextInt();
        if(x < y){
            System.out.println(y + " is Greater");
        }else if(x > y){
            System.out.println(x + " is Greater");
        }else{
            // else works when any option didn't matched.
            System.out.println("Both are equal as " + x);
        }
    }
}
```



```
Enter an integer : 22
Enter another integer : 22
Both are equal as 22
```

Figure 6.3: If-Else Condition

### 6.1.5 Switch

Switch is a conditional statement which is used to take decision from multiple options. It is kind of a list contains multiple decisions based on multiple situations. As an example, we all have seen the vending machine. There are lots of cokes placed on the shelves. We have to put money and press the button of the coke we want. Then the coke from a specific shelf comes down automatically. Switch case is kind of same. User have to push a button or select from multiple choice. The program will perform

the task assigned for that specific option from those multiple choices. There are 2 types of inputs for switches. Those are,

- Single Digit Numbers ( $0 \rightarrow 9$ )
- Alphabets (A-Z) or (a-z)



Figure 6.4: Vending Machine[10]

### Syntax

```
// If input type is numeric
switch(integer_variable){
    case 1:{
        statements;
        statements;
        break;
    }
    case 2:{
        statements;
        statements;
        break; // we use breaks so that after executing one case,
               //the switch stops and don,t go to another one
    }
    default:{
        statements;
        statements;
        // Default works when any option didn't matched.
        break;
    }
}
```

```

// If input type is character
switch(character_variable){
    case 'a':{
        statements;
        statements;
        break;
    }
    case 'B':{
        statements;
        statements;
        break;
    }
    default:{
        statements;
        statements;
        break;
    }
}

```

### 6.1.6 Example

```

import java.util.Scanner;

public class Test{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        int x, numberOfCokes = 0, numberOfChips = 0;
        char y = 'a';
        double price = 0;
        System.out.println("----- COKES -----");
        System.out.println("                        Press 1 for COCACOLA                |");
        System.out.println("                        Press 2 for PEPSI                  |");
        System.out.println("-----");
        System.out.print("                        Enter your Choice >> ");
        x = input.nextInt();
        // If input type is numeric
        switch(x){
            case 1:{
                System.out.println("COCACOLA SELECTED");
                System.out.println("Price is = 25/-");
                numberOfCokes++;
                price = price + 25 * numberOfCokes;

                break;
            }
            case 2:{
                System.out.println("PEPSI SELECTED");
                System.out.println("Price is = 15/-");
                numberOfCokes++;
                price = price + 15 * numberOfCokes;
            }
        }
    }
}

```



```

        break;
    }
    default:{
        System.out.println("\nPLEASE ENTER A VALID INPUT");
        break;
    }
}

System.out.println("----- CHIPS -----");
System.out.println("                Press a for KURKURE                |");
System.out.println("                Press b for LAYS                |");
System.out.println("-----");
System.out.print("                Enter your Choice >> ");
y = input.next().charAt(0);
// If input type is character
switch(y){
    case 'a':{
        System.out.println("KURKURE SELECTED");
        System.out.println("Price is = 25/-");
        numberOfChips++;
        price = price + 25 * numberOfChips;
        break;
    }
    case 'b':{
        System.out.println("KURKURE SELECTED");
        System.out.println("Price is = 20/-");
        numberOfChips++;
        price = price + 20 * numberOfChips;
        break;
    }
    default:{
        System.out.println("\nPLEASE ENTER A VALID INPUT");
        break;
    }
}
System.out.println("Your Bill is = " + price + " /- TK");
}
}

```

```

----- COKEs -----
Press 1 for COCACOLA
Press 2 for PEPSI
Enter your Choice >> 2
PEPSI SELECTED
Price is = 15/-

----- CHIPS -----
Press a for KURKURE
Press b for LAYS
Enter your Choice >> a
KURKURE SELECTED
Price is = 25/-
Your Bill is = 40.0 /- TK

```

Figure 6.5: Switch[3]

## 6.2 Looping Statements

Looping means iterations. That means doing any specific task again & again or multiple times. Sometimes we have to do many tasks again and again to complete it. In these situations we use loops to finish it. because it's impossible to write same code 100 of time if we have to repeat it 100 times. So, we creates loops and tell it to run and do same task repeatedly 100 times. sometimes we need to repeat same task till a specific environment occurs. In that case we can also use conditions in the loops. There are 3 kinds of loops in JAVA. Those are,

- For Loops
- While Loops
- Do-While Loops

### 6.2.1 For Loops

For Loops are also called incremental loops. We use this loop, when we know exactly how many times we want to repeat the task.

#### Syntax

```
for(starting_point;ending_Point;Increment/Decrement){
    //Statements;
    //Statements;
}
```

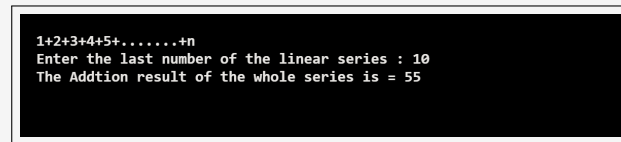
#### Example

```
import java.util.Scanner;

public class Test{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        int result = 0;
        System.out.println("1+2+3+4+5+.....+n");
        System.out.print("Enter the last number of the linear series : ");
        int n = input.nextInt();
        for(int i = 1; i <= n; i++){
            result = result + i;
        }
        System.out.println("The Addition result of the whole series is = " + result);
    }
}
```

### 6.2.2 While Loops

While loops are also called conditional loops. Because we use conditions in this loop. As an exaple, we can do a repeated task until the user put a specific value.

A screenshot of a Java program's output. The first line shows the formula for the sum of the first n natural numbers: 1+2+3+4+5+.....+n. The second line shows the user input: "Enter the last number of the linear series : 10". The third line shows the result: "The Addition result of the whole series is = 55".

```
1+2+3+4+5+.....+n
Enter the last number of the linear series : 10
The Addition result of the whole series is = 55
```

Figure 6.6: For Loop

### Syntax

```
initialization;
while(condition){
    //statements;
    //statements;
    increment/decrement;
}
```

### Example

```
import java.util.Scanner;
public class Test{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        System.out.print("Enter nmber >> ");
        int x = input.nextInt(); // Initialization
        while (x > 0){
            System.out.println(x); // Statements
            x--; // Decrement
        }
    }
}
```

A screenshot of a Java program's output. The first line shows the prompt: "Enter nmber >> 5". The subsequent lines show the numbers 5, 4, 3, 2, and 1, each on a new line, representing the values of x during the while loop iteration.

```
Enter nmber >> 5
5
4
3
2
1
```

Figure 6.7: While Loop[3]

### 6.2.3 Do-While Loops

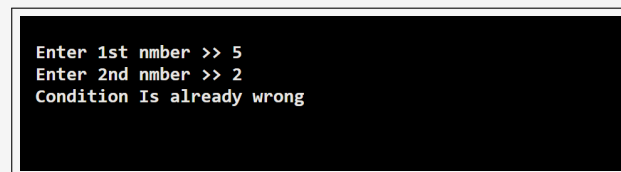
Do while loops are similar like while loop, just the difference is, while loop checks condition at the first, but in do while, it checks at the last. And if condition is wrong at the first, then while loop doesn't work, but as do while checks the condition at the last, so do while loop runs minimum one time though the condition is wrong.

#### Syntax

```
initialization;
do{
    //statements;
    //statements;
    increment/decrement;
}while(condition);
```

#### Example

```
import java.util.Scanner;
public class Test{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        System.out.print("Enter 1st number >> ");
        int x = input.nextInt();
        System.out.print("Enter 2nd number >> ");
        int y = input.nextInt();
        do{
            System.out.println("Condition Is already wrong");
            x--;
        }while (x / y == 0);
    }
}
```

A screenshot of a terminal window with a black background and white text. It shows the execution of a Java program. The first two lines are prompts for user input: "Enter 1st number >> 5" and "Enter 2nd number >> 2". The third line is the output of the program: "Condition Is already wrong".

```
Enter 1st number >> 5
Enter 2nd number >> 2
Condition Is already wrong
```

Figure 6.8: Do While Loop[3]

## 6.3 Jump Statements

### 6.3.1 Break

Break is mostly used in switches. Rather, It is also used in the loops too. Break basically ends the most inner loop a switch case. That means if a java compiler reads a break statement it immediately

```
1.44614pt
1.44614pt
```

# Index

Access, 14

Child Class, 15  
class libraries, 10  
Comment, 20

Default, 15

Encapsulation, 14  
Escape Sequences, 18

James Gosling, 10  
Java compilers, 10  
Java technologies, 10

JVM, 10

Linux distributions, 10

Main Method, 16

OpenJDK, 10

Parent Class, 15  
Private, 15  
Public, 14

virtual machines, 10

# Bibliography

- [1] Java, [https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
- [2] James Gosling, [https://en.wikipedia.org/wiki/James\\_Gosling](https://en.wikipedia.org/wiki/James_Gosling)
- [3] Java Compiler, <https://www.jdoodle.com/online-java-compiler/>
- [4] Escape Sequence, <https://docs.oracle.com/javase/tutorial/java/data/characters.html>
- [5] Format Specifier, <https://www.geeksforgeeks.org/format-specifiers-in-java/>
- [6] DataType, <https://www.javatpoint.com/java-data-types>
- [7] User Input, [https://www.w3schools.com/java/java\\_user\\_input.asp](https://www.w3schools.com/java/java_user_input.asp)
- [8] Operators, <https://www.qafox.com/java-for-testers-different-types-of-operators-in-java/>
- [9] Control Statements, <http://www.testingtools.co/java/11-keywords-to-learn-loops-and-conditional-statements-in-java>
- [10] Vending Machine, <http://benchmarkreporter.com/buying-vs-renting-a-vending-machine-which-option-is-better/>