JAVA Simplica

1st Edition, Sudipta Kumar Das

```
          JJJJJJJJJJJ                    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
JJJJJJJ     J:::::J             A:::::AAAAAAAAA:::::A                  V:::::V V:::::V                  A:::::AAAAAAAAA:::::A
J:::::J     J:::::J            A:::::::::::::::::::::A                  V:::::V:::::V                  A:::::::::::::::::::::A
J::::::J   J::::::J           A:::::AAAAAAAAAAAAA:::::A                  V:::::::::V                  A:::::AAAAAAAAAAAAA:::::A
J:::::::JJJ:::::::J          A:::::A             A:::::A                  V:::::::V                  A:::::A             A:::::A
 JJ:::::::::::::JJ          A:::::A               A:::::A                  V:::::V                  A:::::A               A:::::A
   JJ:::::::::JJ           A:::::A                 A:::::A                  V:::V                  A:::::A                 A:::::A
     JJJJJJJJJ            AAAAAAA                   AAAAAAA                  VVV                  AAAAAAA                   AAAAAAA
```

```
  _____   _____   __  __   _____    _        _____    _____
 / ____| |_   _| |  \/  | |  __ \  | |      |_   _|  / ____|     /\
| (___     | |   | \  / | | |__) | | |        | |   | |         /  \
 \___ \    | |   | |\/| | |  ___/  | |        | |   | |        / /\ \
 ____) |  _| |_  | |  | | | |      | |____   _| |_  | |____   / ____ \
|_____/  |_____| |_|  |_| |_|      |______| |_____|  \_____| /_/    \_\
```

2022, Publisher

Contents

List of Figures

List of Tables

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 looses 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 out 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 form 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 thing can be access 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 complier 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

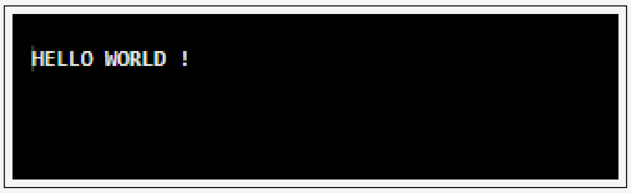
We use *System.out.println();* to print anything or show anything on console. Here println means print a newline also. That means the line will break and go to a new line after showing the output inside first bracket.

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 !");
    }
}
```



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

Table 3.1: Escape Sequences

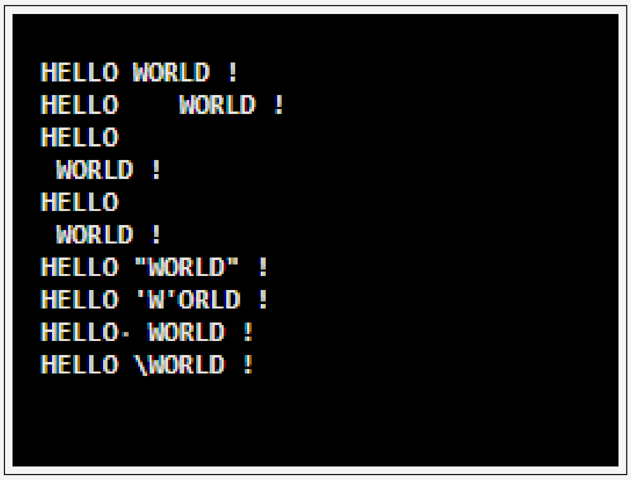
3.1.1 Syntax

"\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 !");
            }
        }
```



```
HELLO WORLD !
HELLO    WORLD !
HELLO
 WORLD !
HELLO
 WORLD !
HELLO "WORLD" !
HELLO 'W'ORLD !
HELLO♀ WORLD !
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"

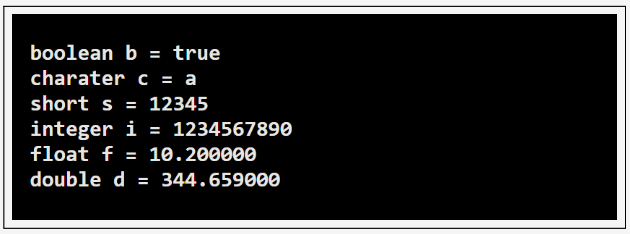
3.2.2 Example

```
        public class Test {
            public static void main(String args[]) {
                int i = 1234567890;
                boolean b = true;
                char c = ’a’;
```

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

```
        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);
    }
}
```



```
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]

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 postion, comments help to understand the workflow 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

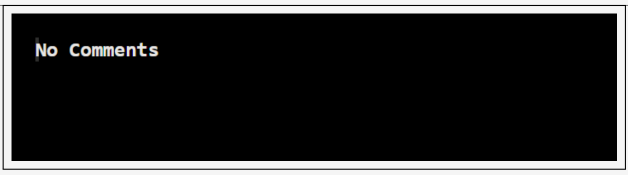
This comments contains just as many as line we take. But these kinds of comments are used for documentation purpose only. This kinds of comments started with just one forward slash and two star(/**) and ends with one star and one forward slash(*/)

3.3.4 Syntax

- Single Line Comments → //Comments
- Multi Line Comments → /*Comments*/
- Documentation Comments → /**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");*/
    }
}
```



```
No Comments
```

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 appropiately. 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
nextBoolean()	Reads Boolean values
nextByte()	Reads byte value
nextDouble()	Reads double value
nextFloat()	Reads float value
nextInt()	Reads int value
nextLine()	Reads String value
nextLong()	Reads long value
nextShort()	Reads short value
next().charAt(0)	Reads Char value

Table 3.3: User Input Type

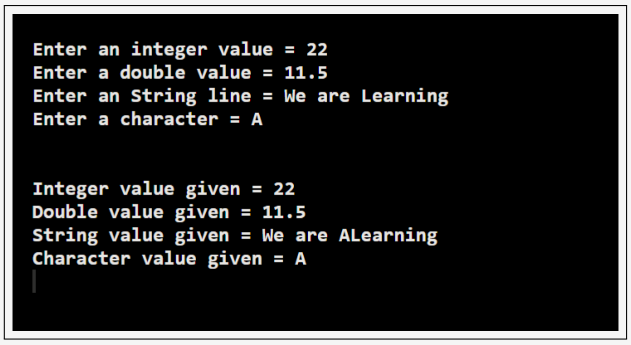
3.4.1 Syntax

```
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);
            }
        }
```



```
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.

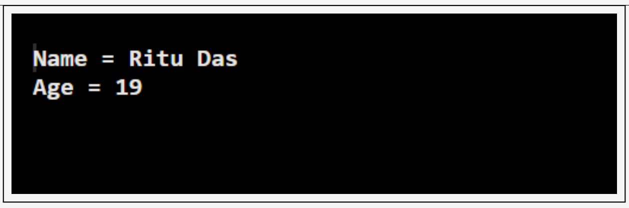
4.1.3 Syntax

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 charater 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.

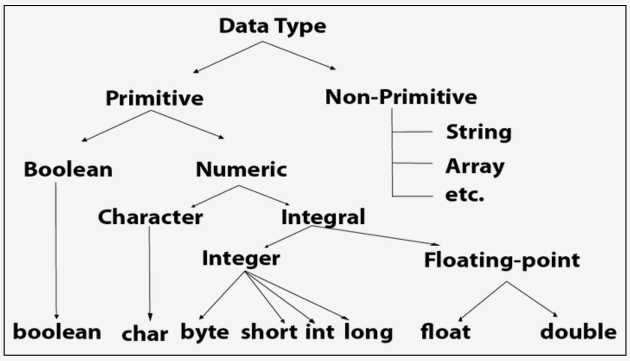


Figure 4.2: Data Types[6]

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	± 1.4E-45 to ± 3.44028235E+38	float x = 10.2f
double	IEEE 754 floating point	8 Byte	±4.9E-324 to ±1.7976931348623157E+308	double x = 21.3

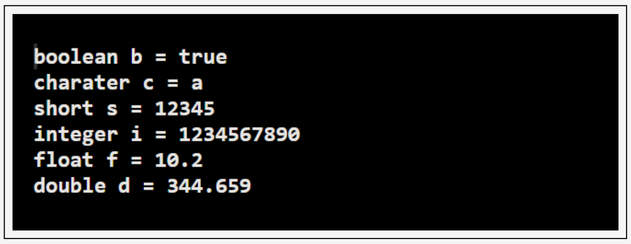
Table 4.1: Data Type

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

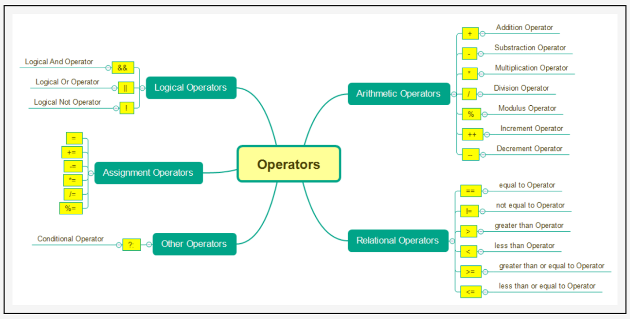


Figure 5.1: Operators[8]

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. It means these kinds of operators need just one varible to perform their tasks.

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

Table 5.3: Unary Operators

Unary operators are also 2 kinds. those are,

- Prefix
- Postfix

5.3.1 Prefix

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

Unary Operator	Meaning
++expr	Increment First
–expr	Decrement First

Table 5.4: Prefix Operators

5.3.2 Postfix

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

Unary Operator	Meaning
expr++	Increment Later
expr–	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
>	Op1>Op2	Greater Than
>=	Op1>=Op2	Greater Than Equal
<	Op1<Op2	Less Than
<=	Op1<=Op2	Less Than Equal
==	Op1==Op2	Both are Equal
!	Op1!=Op2	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
ˆ	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.

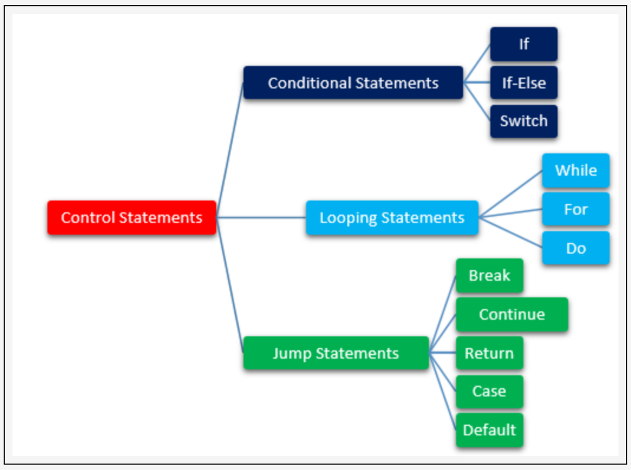


Figure 6.1: Control Statements[3]

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

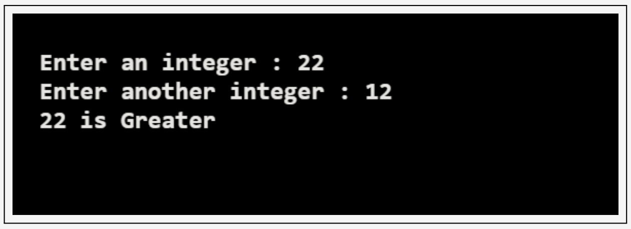
In this statement there will be atleast one if(condition){}. And atmost one else{}. It will be perfectly alright if we do not use else here.

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.

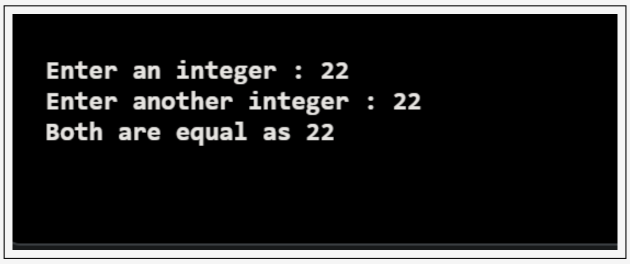
Syntax

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 automatic--ally. 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 → 9)
- Alphabets (A-Z) or (a-z)

Figure 6.4: Vending Machine[10]

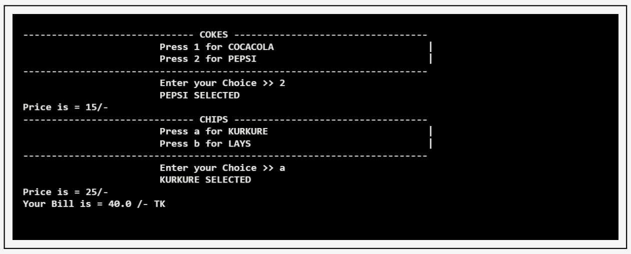
Syntax

```
// 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 numaric
        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

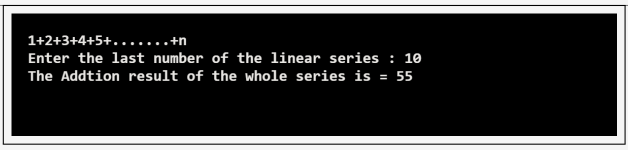
```
for(starting_point;ending_Point;Increment/Decrement){
    //Statements;
    //Statements;
}
```

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 Addtion 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.



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

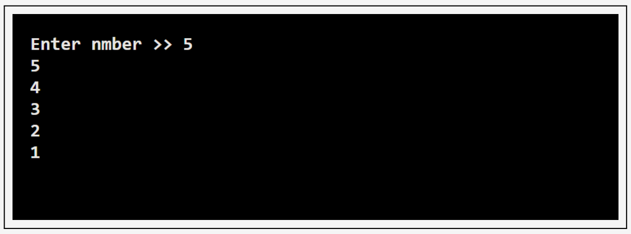
Figure 6.6: For Loop

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
        }
    }
}
```



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

Figure 6.7: While Loop[3]

6.2.3 Do-While Loops

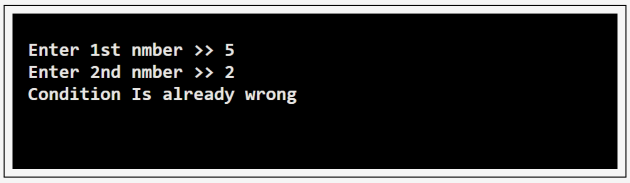
Do while loops are similar like while loop, just the difference is, while loop checks condition at the first, but in do while, it checks at the last. And if condition is wrong at the first, then while loop doesn't work, but as do while checks the condition at the last, so do while loop runs minimum one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



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

Figure 6.8: Do While Loop[3]

6.2.4 For Each Loops

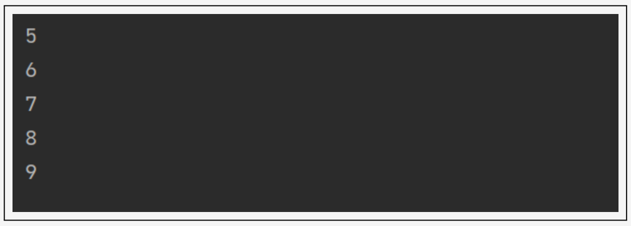
For Each loop is called enhanced for loop. It is basically used for arrays.

Syntax

```
for(dataType : variableName){
    \\ Statements;
    \\ Statements;
}
```

Example

```
import java.util.Scanner;
public class Test {
    public static void main(String args[]) {
        int [] numbers = {5,6,7,8,9};
        for(int x : numbers){
            // More Statements
            System.out.println(x);
        }
    }
}
```



```
5
6
7
8
9
```

Figure 6.9: For Each Loop

6.3 Jump Statements

6.3.1 Break

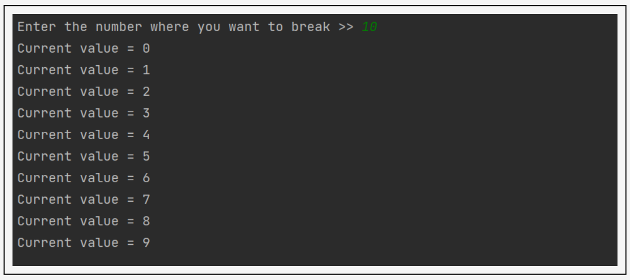
Break is mostly used in switches. Rather, It is also used in the loops too. Break basically ends the most inner loop a switch case. That means if a java compiler reads a break statement it immediately the most inner loop or case.

Syntax

break;

6.3.2 Example

```
import java.util.Scanner;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the number where you want to break >> ");
        int x = input.nextInt();
        int i = 0;
        while(true){
            if(x==i){
                break;
            }else{
                System.out.println("Current value = "+i);
            }
            i++;
        }
    }
}
```



```
Enter the number where you want to break >> 10
Current value = 0
Current value = 1
Current value = 2
Current value = 3
Current value = 4
Current value = 5
Current value = 6
Current value = 7
Current value = 8
Current value = 9
```

Figure 6.10: Break

6.3.3 Continue

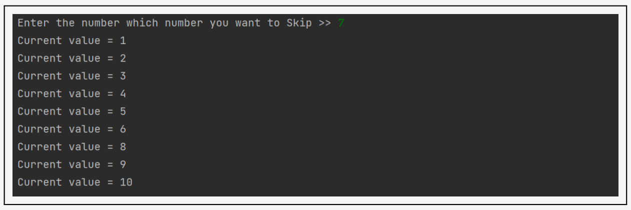
Continue keyword means kind of skip. That means if a java compiler founds that keyword in any loop then it skips that iteration(executing statements) & goes for the next iteration.

Syntax

continue;

Example

```
import java.util.Scanner;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the number which number you want to Skip >> ");
        int x = input.nextInt();
        int i = 1;
        while(i<20){
            if(x==i){
                i++;
                continue;
            }else{
                System.out.println("Current value = "+i);
            }
            i++;
        }
    }
}
```



```
Enter the number which number you want to Skip >> 7
Current value = 1
Current value = 2
Current value = 3
Current value = 4
Current value = 5
Current value = 6
Current value = 8
Current value = 9
Current value = 10
```

Figure 6.11: Continue

Chapter 7

Array

Array means declaration of bunch of variables of same type at a time. It means if we want to declare 50 variables for 50 data, and we don't use array, then we have to declare them one by one manually. But if we use array then we just declare once with data type and we'll tell it to declare 50 variables then it'll declare 50 variables automatically it's own.

To create an array we need new keyword. Sometimes the size of array can be pre defined sometimes the user have the choice to declare the size of an array. These are called dynamic memory allocation. This type of animation is also called non-primitives. To see the length of the array, we have to use .length keyword.

Syntax of Array Length

int variableName = arrayName.length;

Example of Array Length

```
int size = numbers.length;
```

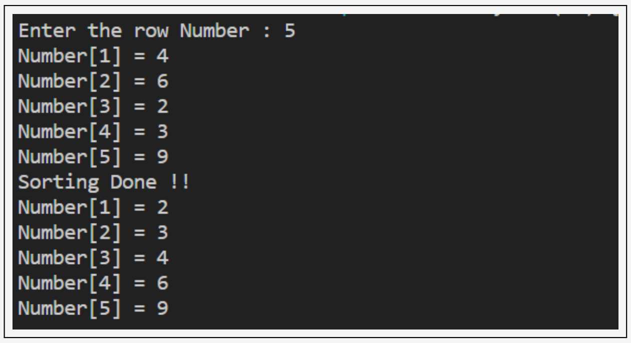
Syntax of Sorting an Array

Arrays.sort(arrayName);

Example of Sorting an Array

```
import java.util.Arrays;
import java.util.Scanner;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter the row Number : ");
        int n = input.nextInt();
```

```
            int[] arr = new int[n];

            for(int i=0;i<n;i++){
                System.out.print("Number["+(i+1)+"] = ");
                arr[i] = input.nextInt();
            }
            Arrays.sort(arr); // Code For Sorting a array Low to High
            System.out.println("Sorting Done !!");

            for(int i=0;i<n;i++){
                System.out.println("Number["+(i+1)+"] = "+arr[i]);
            }
        }
    }
```



```
Enter the row Number : 5
Number[1] = 4
Number[2] = 6
Number[3] = 2
Number[4] = 3
Number[5] = 9
Sorting Done !!
Number[1] = 2
Number[2] = 3
Number[3] = 4
Number[4] = 6
Number[5] = 9
```

Figure 7.1: Array Sorting[3]

Classification of Arrays

There are 2 kinds of arrays. Those are,

- One Dimensional
- Two Dimensional

7.1 One Dimensional Array

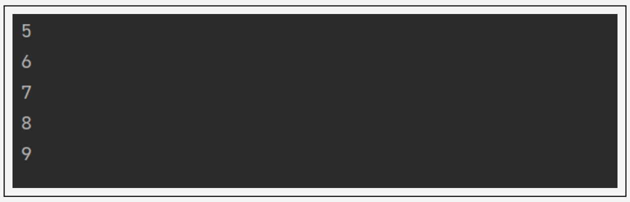
One Dimensional array has just one row of variables in a matrix.Or, we can say that one dimensional array is just a simple array with defined number of variables. And how many variables it'll create, that defined number depend on us.

7.1.1 Syntax

```
dataType [] arrayName = new dataType[Size]; // Declaration only
dataType [] variableName = {} //Declaration and Initialization
```

7.1.2 Example

```
import java.util.Scanner;
public class Test {
    public static void main(String args[]) {
        int [] numbers = {5,6,7,8,9};
        for(int i=0;i<numbers.length; i++){
            System.out.println(numbers[i]);
        }
    }
}
```



```
5
6
7
8
9
```

Figure 7.2: One Dimensional Array

7.2 Two Dimensional Array

Two dimensional arrays are like matrix table. But each and every element or position is a variable.

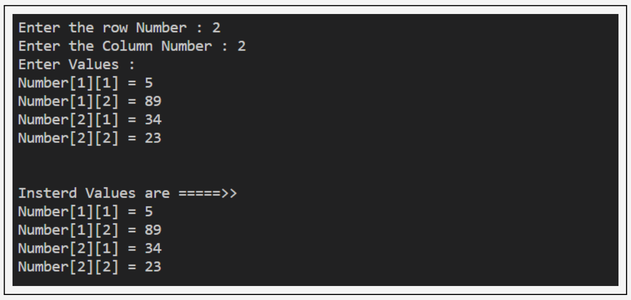
7.2.1 Syntax

dataType[][] arrayName = new dataType[][];

7.2.2 Example

```
import java.util.Scanner;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the row Number : ");
        int m = input.nextInt();
        System.out.print("Enter the Column Number : ");
        int n = input.nextInt();
        System.out.println("Enter Values : ");
        int [][] numbers = new int[m][n];
        for(int row=0;row<m;row++){
            for(int col=0;col<n;col++){
                System.out.print("Number["+(row+1)+"]["+(col+1)+"] = ");
                numbers[row][col] = input.nextInt();
            }
        }

        System.out.println("\n\nInsterd Values are =====>>");
```

```
            for(int row=0;row<m;row++){
                for(int col=0;col<n;col++){
                    System.out.println("Number["+(row+1)+"]["+(col+1)+"] = "+numbers[row][col]);
                }
            }
        }
    }
```



```
Enter the row Number : 2
Enter the Column Number : 2
Enter Values :
Number[1][1] = 5
Number[1][2] = 89
Number[2][1] = 34
Number[2][2] = 23


Insterd Values are =====>>
Number[1][1] = 5
Number[1][2] = 89
Number[2][1] = 34
Number[2][2] = 23
```

Figure 7.3: temp[3]

7.3 ArrayList

In short, arrayList is a dynamic array, where you can get as much as variables but no overflow or null/underflow. As an example, suppose we need 5 variables to store the marks of 5 students. but suddenly 3 students appears now if we use normal array, then the storage is fixed with length 5 which means we have start from the beginning again.But, if we use the arrayList, then we can add those 3 students also without writing the code/software again. That's why it is also called the better version of arrays.
To add values in arrayList we need to use *arrayListName.add(value);*

Array	ArrayList
Not Resizable	Resizable
For/For Each Loop	For Each Loop/Iterator
Fast	Slow
arrayName.length;	arrayName.size();
Static	Dynamic

Table 7.1: Array VS ArrayList

7.3.1 Syntax

ArrayList<dataType>variableName = new ArrayList();

7.3.2 Example

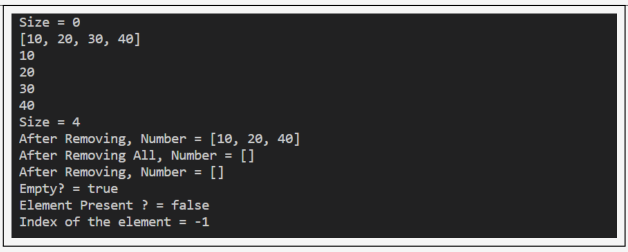
```
import java.util.ArrayList;
import java.util.Scanner;

import javax.print.attribute.standard.NumberUp;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        ArrayList<Integer>number = new ArrayList<>();
        System.out.println("Size = "+number.size());
        number.add(10);
        number.add(20);
        number.add(30);
        number.add(40);
        System.out.println(number); // Horizontal Print
        //using for each loop
        for(int x: number){
            System.out.println(x);
        }
        System.out.println("Size = "+number.size());
        // Removing Elements
        number.remove(2);
        System.out.println("After Removing, Number = "+number);
        // REmoving all elements
        number.removeAll(number);
        System.out.println("After Removing All, Number = "+number);
        // Removing all Elements
        number.clear();
        System.out.println("After Removing, Number = "+number);
        boolean b = number.isEmpty();// Return true if empty
        System.out.println("Empty? = "+b);
        // Containsn checks if the element is present or not
        boolean b1 = number.contains(30);
        System.out.println("Element Present ? = "+b1);
        // Index of shows index of any value, if not found then gives -1
        int i = number.indexOf(40);
        System.out.println("Index of the element = "+i);
    }
}
```

7.3.3 Set Methods

.set methods is used to replace any existing value present in any index of arrayList. That means if we want to change the value of 5th index then the arrayList should have the values from index 0 → 5.

Syntax

arrayListName.set(index,value);



```
Size = 0
[10, 20, 30, 40]
10
20
30
40
Size = 4
After Removing, Number = [10, 20, 40]
After Removing All, Number = []
After Removing, Number = []
Empty? = true
Element Present ? = false
Index of the element = -1
```

Figure 7.4: ArrayList[3]

7.3.4 Get Methods

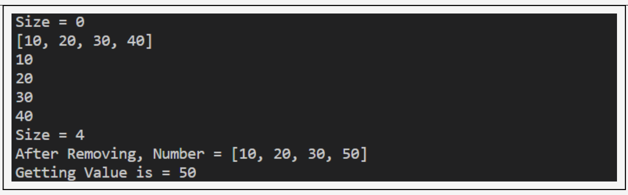
.get methods is used to print any value present in any index of arrayList.

Syntax

dataType variableName = arrayListName.get(indexNumber);

7.3.5 Example

```
import java.util.ArrayList;
import java.util.Scanner;

import javax.print.attribute.standard.NumberUp;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        ArrayList<Integer>number = new ArrayList<>();
        System.out.println("Size = "+number.size());
        number.add(10);
        number.add(20);
        number.add(30);
        number.add(40);
        System.out.println(number); // Horizontal Print
        //using for each loop
        for(int x: number){
            System.out.println(x);
        }
        System.out.println("Size = "+number.size());
        // Removing Elements
        number.set(3,50);
        System.out.println("After Removing, Number = "+number);
        // REmoving all elements
        int x = number.get(3);
        System.out.println("Getting Value is = "+x);
    }
}
```



```
Size = 0
[10, 20, 30, 40]
10
20
30
40
Size = 4
After Removing, Number = [10, 20, 30, 50]
Getting Value is = 50
```

Figure 7.5: ArrayList Set Get[3]

7.3.6 ArrayList Methods

Method	Task
size()	it shows the length/size
add()	add value or element
remove()	removes a specific element
removeAll()	removes every elements
clear()	removes every elements
isEmpty()	if yes, returns true
contains()	if that specific element available, return true
indexOf()	returns the index value if found, else return -1
set()	replace value of given index
get()	shows value of a specific index
equal()	shows if the arrayLists are equal or not
addAll()	Merge a arrayList into another

Table 7.2: ArrayList Methods

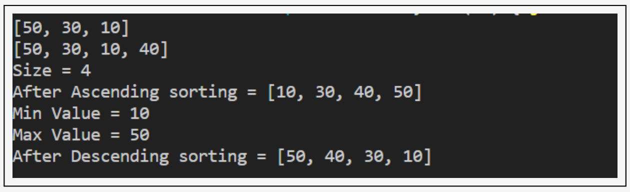
7.3.7 Sorting ArrayList

Syntax

```
Collections.sort(arrayListName);//Ascending
Collections.sort(arrayListName,Collections.reverseOrder());//Reverse
int variableName = ArrayListName.get(0);//Min
int variableName = ArrayListName.size()-1);// Max
```

Example

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Scanner;

import javax.print.attribute.standard.NumberUp;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        ArrayList<Integer>numbers = new ArrayList<>();//Declare ArrayList
        // Adding values to numbers
        numbers.add(50);
        numbers.add(30);
        numbers.add(10);
        System.out.println(numbers);
        numbers.add(3,40); // Adding in a specific position
        System.out.println(numbers);
        System.out.println("Size = "+numbers.size());
        // Ascending Sorting
        Collections.sort(numbers);
        System.out.println("After Ascending sorting = "+numbers);
        // Gettting Lowest value of ArrayList
        int min = numbers.get(0);
        System.out.println("Min Value = "+min);
        // Gettting Biggest value of ArrayList
        int max = numbers.get(numbers.size()-1);
        System.out.println("Max Value = "+max);
        // Descendind Sorting
        Collections.sort(numbers,Collections.reverseOrder());
        System.out.println("After Descending sorting = "+numbers);
    }
}
```



```
[50, 30, 10]
[50, 30, 10, 40]
Size = 4
After Ascending sorting = [10, 30, 40, 50]
Min Value = 10
Max Value = 50
After Descending sorting = [50, 40, 30, 10]
```

Figure 7.6: String Methods[3]

Chapter 8

String

Sequence of characters are together called strings. In simple words, words and sentences are called strings.

8.1 String Methods

8.1.1 Syntax

`stringVariableName.methodName()`

8.1.2 String Basic Methods

Method	Description
length()	shows the string length
equals()	checks, 2 strings are same or not
equalsIgnoreCase()	matches the characters, doesn't matter if it's capital or small
contains()	checks if the word is present in the given string or not
isEmpty()	checks if the string is Empty [("") or null] or not
concat()	to add to strings together and make one string
toUpperCase()	It turns all the characters into upper case characters
toLowerCase()	It turns all the characters into lower case characters
startsWith()	It checks if the string started with given character or word
endsWith()	It checks if the string ended with given character or word

Table 8.1: String Basic Methods

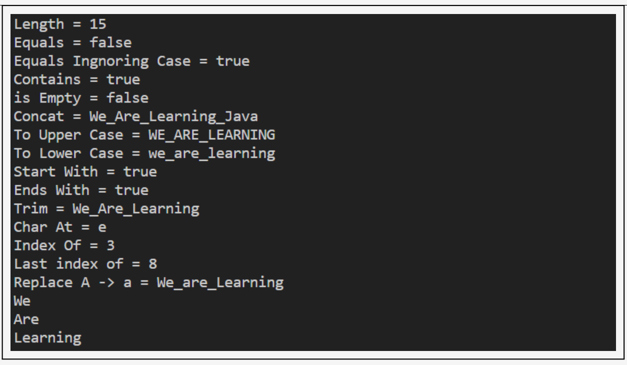
8.1.3 String Special Methods

Method	TaskDescription
trim()	Removes the previous after spaces in of a string
charAt()	Takes an integer value as index & return thing holding in that index variable
charPointAt()	Shows ASCII value based on input parameter
indexOf	Shows the index value of any character/word
lastIndexOf()	If there is same character or word in a string then shows the index of the last occurrence of that word/character
replace()	It replaces all words that matches with the given word and puts new word
split()	It breaks when it founds the specific character and crops data till there

Table 8.2: String Special Methods

8.1.4 Example

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Scanner;

import javax.print.attribute.standard.NumberUp;
public class Test {
    public static void main(String args[]) {
        String message2 = "We_are_Learning";
        String message = "We_Are_Learning";
        System.out.println("Length = "+message.length());
        System.out.println("Equals = "+message.equals(message2));
        System.out.println("Equals Ingnoring Case = "+message.equalsIgnoreCase(message2));
        System.out.println("Contains = "+message.contains("Learning"));
        System.out.println("is Empty = "+message.isEmpty());
        System.out.println("Concat = "+message.concat("_Java"));
        System.out.println("To Upper Case = "+message.toUpperCase());
        System.out.println("To Lower Case = "+message.toLowerCase());
        System.out.println("Start With = "+message.startsWith("We"));
        System.out.println("Ends With = "+message.endsWith("Learning"));

        System.out.println("Trim = "+message.trim());
        System.out.println("Char At = "+message.charAt(5));
        System.out.println("Index Of = "+message.indexOf("Are"));
        System.out.println("Last index of = "+message.lastIndexOf("e"));
        System.out.println("Replace A -> a = "+message.replace('A','a'));
        String[] words = message.split("_");
        for (String string : words) {
            System.out.println(string);
        }
    }
}
```



```
Length = 15
Equals = false
Equals Ingnoring Case = true
Contains = true
is Empty = false
Concat = We_Are_Learning_Java
To Upper Case = WE_ARE_LEARNING
To Lower Case = we_are_learning
Start With = true
Ends With = true
Trim = We_Are_Learning
Char At = e
Index Of = 3
Last index of = 8
Replace A -> a = We_are_Learning
We
Are
Learning
```

Figure 8.1: String Methods[3]

8.2 String Buffer

The only difference between string and string Buffer is, for string we can not change the string and store at the same variable. Because we just make association by just declaring string variables. But if we initialize string object that means this is only operated inside that class no where else so that's why we can change the string and store in that same variable.

Method	TaskDescription
append()	Add a string with the existing string
reverse()	It returns that same string but printed backwardly as like (learn => nrael) Also called Palindrome
delete(start,end)	It deletes a specific portion of a string based on the indexes
setLength(parameter)	It show the strings from starting index 0 to parameter(integer)

Table 8.3: String Buffer Methods

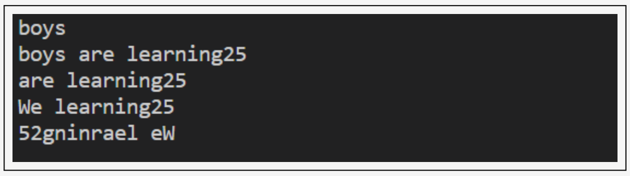
8.2.1 Syntax

StringBuffer variableName = new StringBuffer(stringVariableName);
variableName.methodName();

8.2.2 Example

```
public class Test {
    public static void main(String args[]) {
        String s1 = "boys";
        StringBuffer sb = new StringBuffer(s1);
        System.out.println(sb);
        sb.append(" are learning");
        sb.append(25);
        System.out.println(sb);
```

```
                sb.delete(0, 5);
                System.out.println(sb);
                sb.replace(0, 3, "We");
                System.out.println(sb);
                sb.reverse();
                System.out.println(sb);
            }
        }
```



```
boys
boys are learning25
are learning25
We learning25
52gninrael eW
```

Figure 8.2: String Buffer

8.3 String Builder

We can store string in 3 ways, those are,

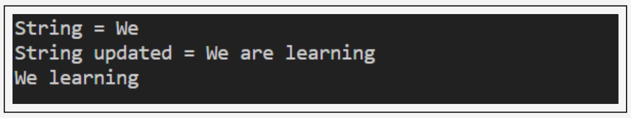
- String
- StringBuffer
- StringBuilder

Of them, StringBuffer and StringBuilder are similar.

8.3.1 Syntax

StringBuilder variableName = new StringBuider(stringVariableName);
variableName.methodName();

```
        public class Test {
            public static void main(String args[]) {
            String s1 = "boys";
            StringBuilder message = new StringBuilder("We");
            System.out.println("String = "+message);
            message.append(" are learning");
            System.out.println("String updated = "+message);
            message.delete(2, 6);
            System.out.println(message);
        }
    }
```



```
String = We
String updated = We are learning
We learning
```

Figure 8.3: String Builder

Chapter 9

Wrapper Class

We know, int, charAt, double, these are primitive dataTypes. So, if we want to convent these into a object or we want to convert an object Introduction primitive dataTypes, then we'll need wrapper classes.

Primitive	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Table 9.1: Wrapper Classes

AutoBoxing

AutoBoxing means to convert a primitive dataType to an object.

UnBoxing

AutoBoxing means to convert an object to a primitive dataType.

Syntax

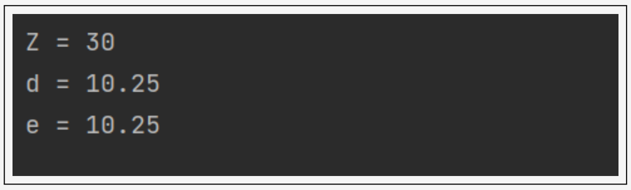
Auto Boxing Process-1 :

```
 dataType variableName = value;
wrapperClassType wrapperVariableName = wrapperClassType.valueOf(variableName);
```

```
Auto Boxing Process-2 :
 dataType variableName = value;
wrapperClassType wrapperVariableName = variableName;

Un-Boxing Process-1 :
 wrapperClassType wrapperVariableName = new wrapperClassType(value); dataType
variableName = wrapperVariableName.variableNameValue();

Un-Boxing Process-2 :
 wrapperClassType wrapperVariableName = new wrapperClassType(value);
dataType variableName = wrapperVariableName;
```

Example

```
public class Test {
    public static void main(String args[]) {
        int x = 30;
        Integer z = x; // Auto-Boxing
        System.out.println("Z = "+z);
        Double d = new Double(10.25);
        System.out.println("d = "+d);
        double e = d; // UnBoxing
        System.out.println("e = "+e);
    }
}
```



```
Z = 30
d = 10.25
e = 10.25
```

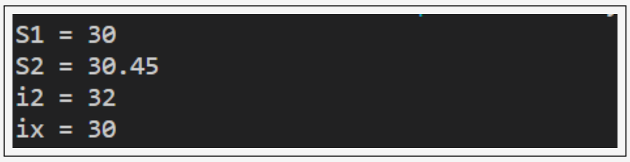
Figure 9.1: AutoBoxing Unboxing

9.1 Conversation between String & Primitive

9.1.1 Example

```
public class Test {
    public static void main(String args[]) {
    int i = 30;
    String s1 = Integer.toString(i); // int --> String
    System.out.println("S1 = "+s1);

    double d = 30.45;
```

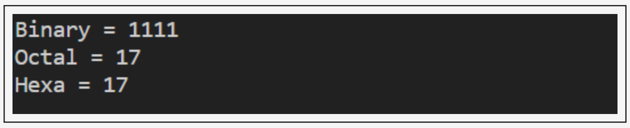
```
        String s2 = Double.toString(d); // double --> String
        System.out.println("S2 = "+s2);

        String s3 = "32";
        int i2 = Integer.parseInt(s3); // String --> int
        System.out.println("i2 = "+i2);
        int ix = Integer.valueOf("100");
        System.out.println("ix = "+i);
    }
}
```



```
S1 = 30
S2 = 30.45
i2 = 32
ix = 30
```

Figure 9.2: String & Primitive

```
    public class Test {
        public static void main(String args[]) {
        int decimal = 15;
        String binary = Integer.toBinaryString(decimal);//decimal --> Binary
        System.out.println("Binary = "+binary);

        String octal = Integer.toOctalString(decimal); // decimal --> Octal
        System.out.println("Octal = "+octal);

        String hexa = Integer.toOctalString(decimal); // decimal --> Octal
        System.out.println("Hexa = "+hexa);
    }
}
```

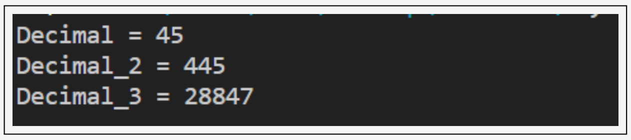


```
Binary = 1111
Octal = 17
Hexa = 17
```

Figure 9.3: Decimal −− > Binary/Octal/Hexa-Decimal

```
    public class Test {
        public static void main(String args[]) {
            String binary = "101101";
            int decimal = Integer.parseInt(binary,2); // binary --> Decimal
```

```
                System.out.println("Decimal = "+decimal);

                String octal = "675";
                int decimal_2 = Integer.parseInt(octal,8); // octal --> Decimal
                System.out.println("Decimal_2 = "+decimal_2);

                String hexa = "70AF";
                int decimal_3 = Integer.parseInt(hexa,16); // hexa --> Decimal
                System.out.println("Decimal_3 = "+decimal_3);
        }
    }
```



```
Decimal = 45
Decimal_2 = 445
Decimal_3 = 28847
```

Figure 9.4: Binary/Octal/Hexa-Decimal −− > Decimal

9.2 Date Class

This ‘Date’ class is the only class which we can use to see the normal date by formatting.

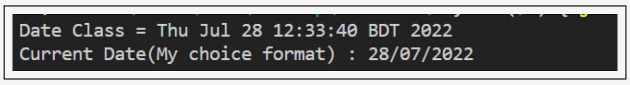
9.2.1 Syntax

System-1 :
Date date = new Date();

System-2 :
Date date = new Date();
DateFormat dateFormat = new SimpleDateFormat("dd/MM/YYYY");
String currentDate = dateFormat.format(date);

9.2.2 Example

```
        import java.text.DateFormat;
        import java.text.SimpleDateFormat;
        import java.util.Date;

        public class Test {
            public static void main(String args[]) {
                Date date = new Date();
            System.out.println("Date Class = "+date);

            // Formatting Time (My choice)
            DateFormat dateFormat = new SimpleDateFormat("dd/MM/YYYY");
            String currentDate = dateFormat.format(date);
            System.out.println("Current Date(My choice format) : "+currentDate);
        }
    }
```



```
Date Class = Thu Jul 28 12:33:40 BDT 2022
Current Date(My choice format) : 28/07/2022
```

Figure 9.5: Date Class

9.3 Time Class

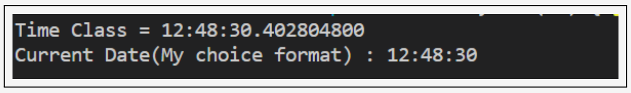
Time class is used to receive current time.

9.3.1 Example

```
System-1 :
 LocalTime time = LocalTime.now();

System-2 :
 LocalTime time = LocalTime.now();
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("hh:mm:ss");
String currentTime = time.format(formatter);
```

9.3.2 Example

```
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;

public class Test {
    public static void main(String args[]) {
        public static void main(String args[]) {
        LocalTime time = LocalTime.now();
        System.out.println("Time Class = "+time);

        // Formatting Time (My choice)
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("hh:mm:ss");
        String currentTime = time.format(formatter);
        System.out.println("Current Date(My choice format) : "+currentTime);
    }
}
```



```
Time Class = 12:48:30.402804800
Current Date(My choice format) : 12:48:30
```

Figure 9.6: Time Class

9.4 Random Number

Random number means the program will generate any digit or multi-digit number automatically. We can do this by using,

- Random Class
- Math Class

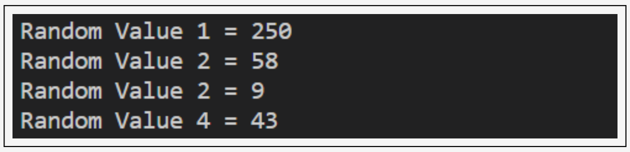
Syntax

By using Random Class :
Random rand = new Rand();
System.out.println$(rand.nextInt(rangeValueFrom_0))$*);*

By using Math Class :
System.out.println$(Math.random() * rangeValueFrom_0)$*;*

Example

```
import java.util.Random;
public class Test {
    public static void main(String args[]) {
        // By using Random Class
        Random rand = new Random();
        int randomNumber = rand. nextInt(10);// from 0 --> 10
        System.out.println("Random Value 1 = "+randomNumber+50);
        int randomNumber1 = rand. nextInt(10)+50;
        System.out.println("Random Value 2 = "+randomNumber1); // from 50 --> (50+10 = 60)

        // By using Math Class
        int randomNumber2 = (int) (Math.random()*20); // from 0 --> 20
        System.out.println("Random Value 2 = "+randomNumber2);
        int randomNumber3 = (int) (Math.random()*20)+30; // from 30 --> (30+20 = 50)
        System.out.println("Random Value 4 = "+randomNumber3);
    }
}
```



```
Random Value 1 = 250
Random Value 2 = 58
Random Value 2 = 9
Random Value 4 = 43
```

Figure 9.7: Random Number

Part IV

Object Oriented Programming

As the name suggests, object-oriented programming or OOPs refers to languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks assigned by you.

Chapter 10

Introduction to Object Oriented Programming

In this chapter we'll learn about the basic things of OOP(Object Oriented Programming). Mainly OOP is based on some core features. Those are,

- Encapsulation
- Classes
- Inheritance
- Abstraction
- Polymorphism
- Access modifiers
- Interface

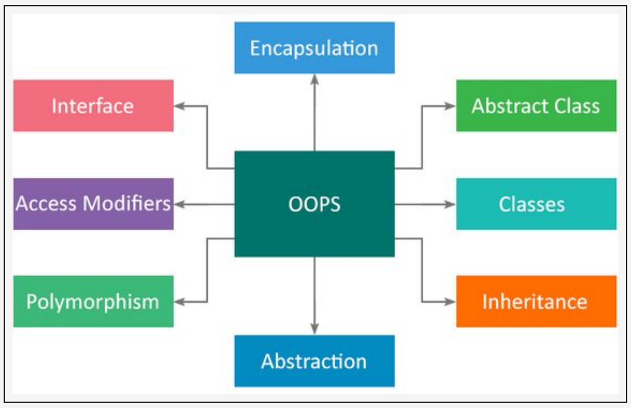


Figure 10.1: OOP concept overview

Object

Everything we can see around us is called an object. More specifically, Variables or instance of any class is called an object.

Class

Common collection of many objects is called a class. More specifically, class is a thing which holds variables and methods inside it.

10.1 Introducing Class

A class in java is basically a template that we can use multiple times. There are 3 major portion of a class. Those are,

- Class Name
- Attributes/Variables
- Methods

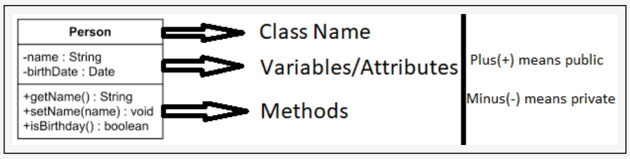


Figure 10.2: Class Structure

10.1.1 Class Name

Every class must have a class name. Class name has also a access modifier public/private/default. If we use default or no access modifier, then it can oly be accessed from that folder. So to use this globally we need to declare it as public. And if we declare it as public, then we have to keep the fileName.java as same as the className.
That means file name would be == className.java

Syntax

```
public class className{
   // Attributes
   //Methods
}
```

10.1.2 Attributes/Variables

After declaring the class name. Now heading part of any class is basically attributes or in other words, you can say here we normally declare all our variables and objects classes that is going to be used in the whole class.

10.1.3 Methods

After declaring all the variables, that we need initially, then we'll go for methods. What is called methods we will learn next.

10.2 Object Declaration & Creation

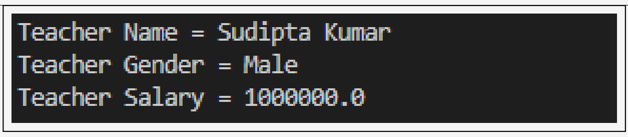
Object of a class means creating an instance of that class. That means all the things of that class which is public, we can access them by using that instance or object.

10.2.1 Syntax

```
className objectName; // Object Declaration
objectName = new className(); // Object Initialization
// Object Declaration & initialization
className objectName = new className();
```

10.2.2 Example

```
class Teacher {
    // Attributes
    public String name;
    public String gender;
    public double salary;

    // Methods
    public void showInfo() {
        System.out.println("Teacher Name = " + name);
        System.out.println("Teacher Gender = " + gender);
        System.out.println("Teacher Salary = " + salary);
    }
}

public class Test {
    public static void main(String args[]) {
        Teacher t1 = new Teacher();
        t1.name = "Sudipta Kumar";
        t1.gender = "Male";
        t1.salary = 1000000;
        t1.showInfo();
    }
}
```



```
Teacher Name = Sudipta Kumar
Teacher Gender = Male
Teacher Salary = 1000000.0
```

Figure 10.3: OOP Object Creation

10.3 Methods in JAVA

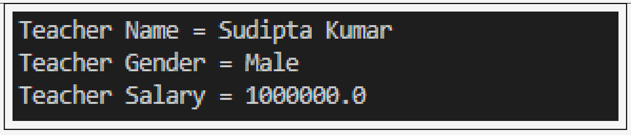
Several things that work together to perform a single task is called a method. That means, method is a structure, where we write some codes which get executed together to perform a single task.

10.3.1 Syntax

```
    methodReturnType methodName(parameter){
        //Statements
    }
```

10.3.2 Example

```
class Teacher {
    // Attributes
    public String name;
    public String gender;
    public double salary;

    // Methods
    public void showInfo() { // Method
        System.out.println("Teacher Name = " + name);
        System.out.println("Teacher Gender = " + gender);
        System.out.println("Teacher Salary = " + salary);
    }
}

public class Test {
    public static void main(String args[]) {
        Teacher t1 = new Teacher();
        t1.name = "Sudipta Kumar";
        t1.gender = "Male";
        t1.salary = 1000000;
        t1.showInfo(); // Calling a Method
    }
}
```



```
Teacher Name = Sudipta Kumar
Teacher Gender = Male
Teacher Salary = 1000000.0
```

Figure 10.4: Constructor

10.4 Constructor

Constructor is the thing which basic job is to initialize the objects. Constructor is used to initialize any object while declaration. To pass the data at the initialization, we use parameters. It is basically a method, but has no return type. Constructor name is as same as the class name. There are 2 kinds of constructor in java. Those are,

- Empty Constructor
- Parametrize Constructor

10.4.1 Empty Constructor

The constructor that doesn't have any parameter to pass, called an empty constructor

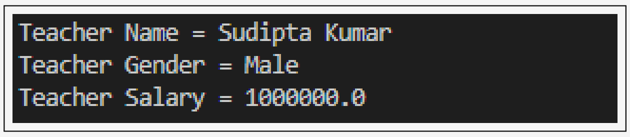
10.4.2 Parametrize Constructor

The constructor that do have any parameter to pass, called an parametrize constructor

10.4.3 Example

```
clas Teacher {
   // Attributes
   public String name;
   public String gender;
   public double salary;

   // Constructor
   public Teacher(){System.out.println("Empty Constructor");}
   public Teacher(String name,String gender,double salary){
       this.name = name;
       this.gender = gender;
       this.salary = salary;
   }

   // Methods
   public void showInfo() { // Method
       System.out.println("Teacher Name = " + name);
```

```
            System.out.println("Teacher Gender = " + gender);
            System.out.println("Teacher Salary = " + salary);
        }
    }

    public class Test {
        public static void main(String args[]) {
            Teacher t1 = new Teacher("Sudipta Kumar","Male",1000000);
            t1.showInfo(); // Calling a Method
        }
    }
```



```
Teacher Name = Sudipta Kumar
Teacher Gender = Male
Teacher Salary = 1000000.0
```

Figure 10.5: Constructor

Index

Bibliography

[1] Java, en.wikipedia.org/wiki/Java_(programming_language)

[2] James Gosling, en.wikipedia.org/wiki/James_Gosling

[3] Java Compiler, www.jdoodle.com/online-java-compiler/

[4] Escape Sequence, docs.oracle.com/javase/tutorial/java/data/characters.html

[5] Format Specifier, www.geeksforgeeks.org/format-specifiers-in-java/

[6] DataType, www.javatpoint.com/java-data-types

[7] User Input, www.w3schools.com/java/java_user_input.asp

[8] Operators, www.qafox.com/java-for-testers-different-types-of-operators-in-java/

[9] Control Statements, www.testingtools.co/java/11-keywords-to-learn-loops-and-conditional-statements-in-java

[10] Vending Machine, www.benchmarkreporter.com/buying-vs-renting-a-vending-machine-which-option-is-better/