JAVA Simplica

1st Edition, Sudipta Kumar Das

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.
Note :
To run the program in your machine/PC/Laptop, just save the file name "Test.java" for all the examples in this book.

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 !");
    }
}
```

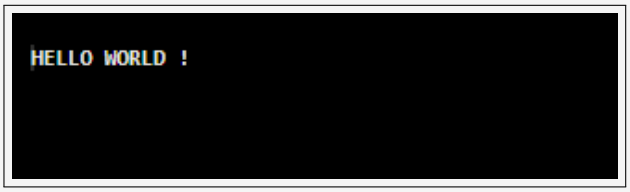

Figure 2.1: Show Output in JAVA[3]

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"

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

3.2.2 Example

```
public class Test {
    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);
    }
}
```

```
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");*/
    }
}
```

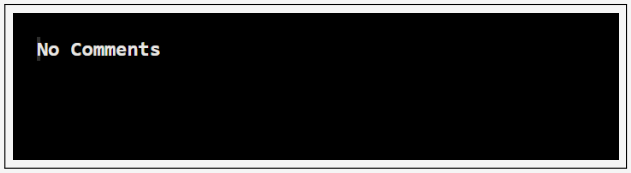

Figure 3.3: Comments[3]

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've to declare an
object of Scanner class. In short, we have to write this line must,And that line is,
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();`

3.4.1 Syntax

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

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

3.4.2 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);
    }
}
```

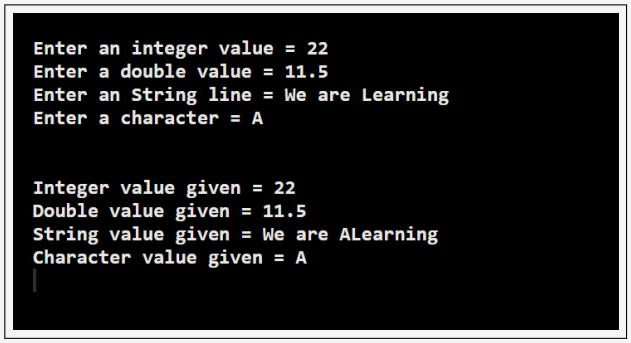

Figure 3.4: User Input[7][3]

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);
        }
    }
```

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 → 9), Underscore(_) & Dollar Sign($) for variable name.
- Variable names can not be started with Numerical values (0 → 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.

4.1.6 Kinds of Variables

We have seen the types of variables based on dataTypes. But there are some another kinds of variables. Those are,

- Static Variable
- Final Variable
- Local Variable
- Global Variable

Static Variable

Static variables are those public variables which can be accessed by the class
itself. That means static variables have to be declared as public. And all
objects of that class uses the same variable. In short if we have tp call a
variable normally then we use objects and call it, but in case of static
variable, we do not use objects to call that variables. As that variable
would be used by all objects so that's why that variable would be called
by the class. Syntax : static dataType variableName;

Final Variable

Final variables are those variables which has the final keyword before if.
Final variables basically means that once it is been initialized, no one can
change its value after that. It can be initialized just one time in a running
program. Syntax : final dataType variableName;

Local Variable

Those variables which we declare inside a method or structure. Which can't be
used outside of that structure.

Global Variable

Those variables which we declare inside a class not inside a method or
structure. Which can't be used outside of that structure.

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.
```

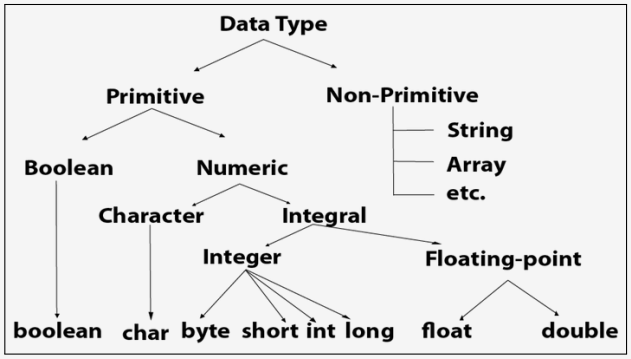

Figure 4.2: Data Types[6]

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 Default Values

- Default value of of integer is zero
- Default value of of String is null

4.2.2 Syntax

accessModifier dataType variableName;

4.2.3 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

5.1 Arithmetic Operators

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

5.2 Assignment Operators

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

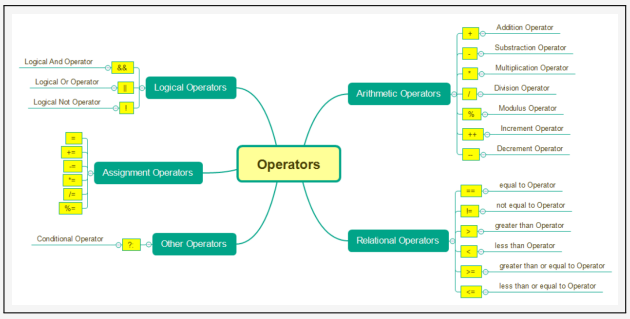

Figure 5.1: Operators[8]

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

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
^	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 semicolon after each and every statement finishes.

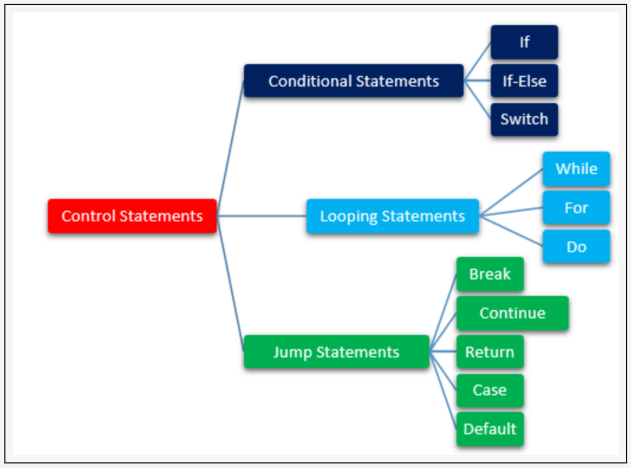

Figure 6.1: Control Statements[3]

6.1 Conditional Statements

We, the programmers need conditional statements the most to control the flow of the program. Conditional statements gives PC 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 confirms deal
and give a call, otherwise just cancel the deal. 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
    }
}
```

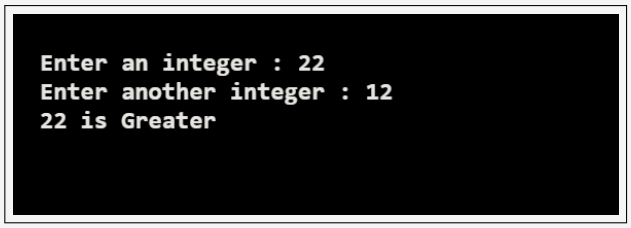

Figure 6.2: If-Else Condition

6.1.3 If, Else If, Else

In this statement there will be at-least one if(condition){}. And at-most 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);
        }

    }
}
```

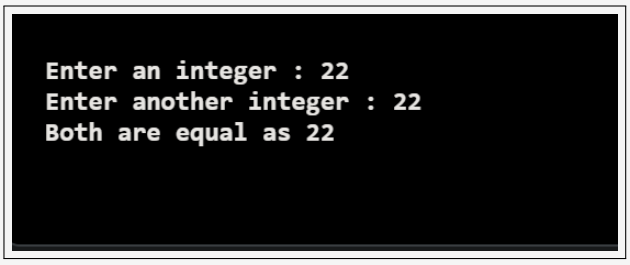

Figure 6.3: If-Else Condition

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 $\rightarrow$ 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 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");
    }
}
```

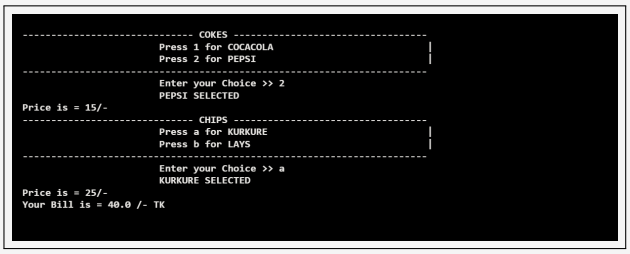

Figure 6.5: Switch[3]

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);
        }
    }
```

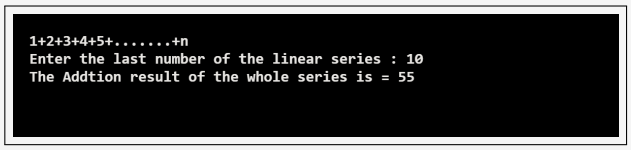

Figure 6.6: For Loop

6.2.2 While Loops

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

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
        }
    }
}
```

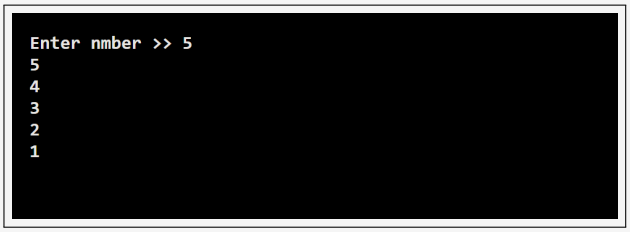

Figure 6.7: While Loop[3]

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 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]);
        }
    }
}
```

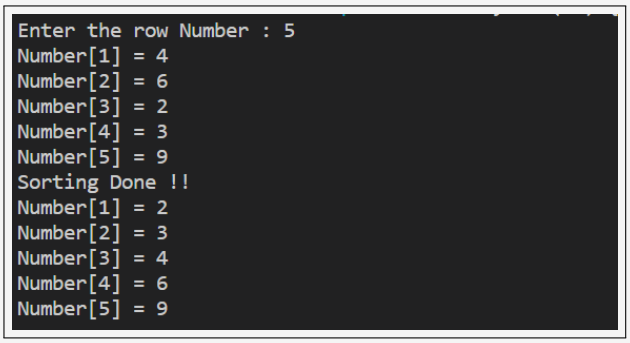

Figure 7.1: Array Sorting[3]

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: 2D Array

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);
            // Contains 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);
        }
    }
```

```
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]

7.3.3 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 $\rightarrow$ 5.

Syntax

arrayListName.set(index,value);

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);
    }
}
```

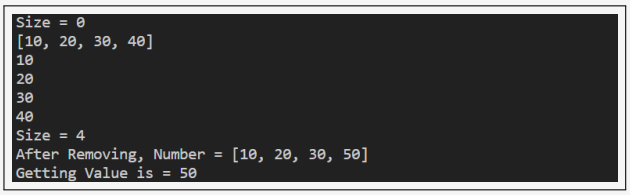

Figure 7.5: ArrayList Set Get[3]

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);
    }
}
```

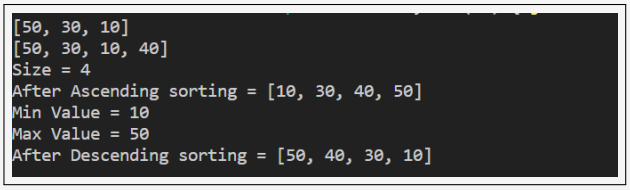

Figure 7.6: String Methods[3]

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);
        }
    }
}
```

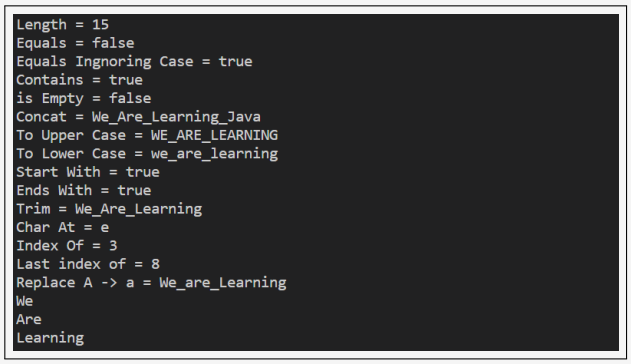

Figure 8.1: String Methods[3]

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);
    }
}
```

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);
    }
}
```

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

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

```
            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);
    }
}
```

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);
    }
}
```

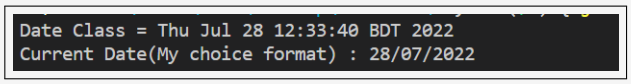

Figure 9.5: Date Class

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

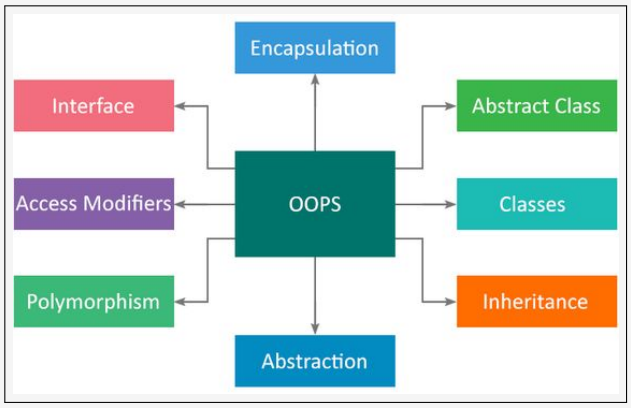

Figure 10.1: OOP concept overview

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

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

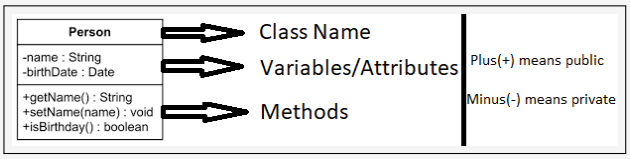

Figure 10.2: Class Structure

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 = "SIFAT RAHMAN AHONA";
        t1.gender = "Female";
        t1.salary = 1000000;
        t1.showInfo();
    }
}
```

```
Teacher Name = SIFAT RAHMAN AHONA
Teacher Gender = Female
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 = "SIFAT RAHMAN AHONA";
        t1.gender = "Female";
        t1.salary = 1000000;
        t1.showInfo(); // Calling a Method
    }
}
```

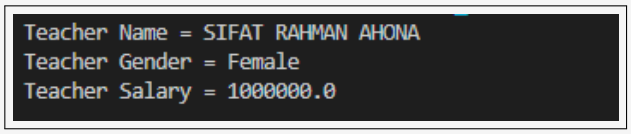

Figure 10.4: Constructor

10.3.3 Argument Passing in Method

Basically argument passing means to pass any value ina method, based on
which the method will perform some task. There are 2 kinds of argument passing. Those are,

- Pass by value / call by value
- Pass by reference / call by reference

Pass by value

In this kind of passing, the compiler just copies the value and pass it in the method. Now, if the value of that variable changes, then outside of the method the value of that variable won't change.

Pass by reference

In this kind of passing, the compiler takes that original value by its position
and pass it in the method. Now, if the value of that variable changes, then
outside of the method the value of that variable will also change.

10.3.4 Variable Length Argument

We can call variable length argument as varargs. Those methods which can take
variable number of arguments are called varargs methods. That means sometimes
we don't know how many variables we have to pass inside the method, when the
program is dynamic. In those cases, we use this varargs kinds of methods. Because it can take any number of same dataType variables as arguments.

Syntax

```
returnType methodName(dataType...variableName){
        // Statements
}
```

Example

```
class Demo {
    public void add(int ...num){
        int sum = 0;
        for(int x : num){
            sum = sum+1;
        }
        System.out.println("Sum = "+sum);
    }
}

public class Test {
    public static void main(String args[]) {
        Demo d1 = new Demo();
        d1.add(10,20);
        d1.add(10,20,30);
    }
}
```

```
Sum = 2
Sum = 3
```

Figure 10.5: Varargs

10.3.5 difference Between Constructor & Method

10.4 Recursion

Recursion means a function which can call itself. Recursion is kind of
loop but without formal loops like for, while do-while, foreach.

10.4.1 Example

```
// Factorial Code
```

Constructor	Method
Constructor name must be as same as class name	Method name can be anything excepts the keywords
No return type	Must have a return type. If nothing to return, then use void
No need to call. It call itself autometically when it's been initialized	It needs to be call either by objects or by class itself
It is used to initialize an object	It is used to show any behaviour or perform any task

Table 10.1: Constructor VS Methods

```
    class Demo {
    public int fact(int num){
        if(num == 1){
            return 1;
        }else{
            return num*fact(num-1); // Recursion here
        }
    }
}

public class Test {
    public static void main(String args[]) {
        Demo d1 = new Demo();
        int result = d1.fact(5);
        System.out.println("Factorial = "+result);
    }
}
```

```
Factorial = 120
```

Figure 10.6: Recursion

10.5 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

10.5.1 Empty Constructor

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

10.5.2 Parametrize Constructor

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

10.5.3 Example

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

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

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

public class Test {
    public static void main(String args[]) {
        Developer d1 = new Developer();
        Developer d2 = new Developer("Sudipta Kumar","Male",1000000);
        d2.showInfo(); // Calling a Method
```

```
    }
}
```

```
Empty Constructor
Developer Name = Sudipta Kumar
Currently living in = Switzerland
Developer Salary = 5022.0/- CHF (Swiss Franc)
```

Figure 10.7: Constructor

Chapter 11

Encapsulation

Encapsulation means encapsulate something. In another word we can say
encapsulation means to packaging two or more things. It is like medicine capsules, where medicines are kept together or make a packet of them. More
preciously, we can say in JAVA, encapsulation is a process in which all
the variables(which we have to encapsulate) of a class can not be accessible directly. There are some conditions to encapsulate. Those are,

- Variables must be in private
- All the private variables must have public methods (Setter & Getter) to access them from another class.

11.1 Setter Methods

In encapsulation, as the variables are in private, so they can not be initialize
from another class. Here, initialize means to put values in those variables.
In that case we use setter methods which return type is usually void particular variable anc access modifier is always public.

11.1.1 Syntax

```
 private dataType variablename;
public void setVariableName(dataType variablename){
        this.variablename = variablename;
}
```

11.1.2 Example

```
class Developer {
    // Attributes
    private String name;
    private String country;
    private double salary;

    // Constructor
    public Developer() {
        System.out.println("Empty Constructor");
    }

    public Developer(String name, String country, double salary) {
        setName(name);
        setCountry(country);
        setSalary(salary);

    }

    public void setName(String name) {
        this.name = name;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public void setSalary(double salary) {
        if (salary >= 0) {
            this.salary = salary;
        } else {
            System.out.println("Salary Can not be Negative values !");
        }
    }

    // Methods
    public void showInfo() { // Method
        System.out.println("Developer Name = " + name);
        System.out.println("Currently living in = " + country);
        System.out.println("Developer Salary = " + salary + "/- CHF (Swiss Franc)");
    }
}

public class Test {
    public static void main(String args[]) {
        Developer d1 = new Developer();
        Developer d2 = new Developer("Sudipta Kumar", "Switzerland", 5022.00);
        d2.showInfo(); // Calling a Method
    }
}
```

```
Empty Constructor
Developer Name = Sudipta Kumar
Currently living in = Switzerland
Developer Salary = 5022.0/- CHF (Swiss Franc)
```

Figure 11.1: Setter Methods

11.2 Getter Methods

In encapsulation, as the variables are in private, we can not get their values from another class. Here, getting means to call out the variables using objects. In that case we use getter methods which return type is usually same as the dataType of that variable which value we want get from it. And there should be no arguments Thus, access modifier is always public.

11.2.1 Syntax

```
 private dataType variablename;
public dataType geetVariableName(){
      return variablename;
}
```

11.2.2 Example

```
class Developer {
    // Attributes
    private String name;
    private String country;
    private double salary;

    // Constructor
    public Developer() {
        System.out.println("Empty Constructor");
    }

    public Developer(String name, String country, double salary) {
        setName(name);
        setCountry(country);
        setSalary(salary);

    }

    // Setter Methods

    public void setName(String name) {
        this.name = name;
```

```
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public void setSalary(double salary) {
        if (salary >= 0) {
            this.salary = salary;
        } else {
            System.out.println("Salary Can not be Negative values !");
        }
    }

    // Getter Methods

    public String getName() {
        return name;
    }

    public String getCountry() {
        return country;
    }

    public double getSalary() {
        return salary;
    }

}

public class Test {
    public static void main(String args[]) {
        Developer d1 = new Developer();
        Developer d2 = new Developer("Sudipta Kumar", "Switzerland", 5022.00);
        System.out.println("Developer Name = " + d2.getName());
        System.out.println("Currently living in = " + d2.getCountry());
        System.out.println("Developer Salary = "+d2.getSalary()+"/- CHF(Swiss Franc)");
    }
}
```

```
Empty Constructor
Developer Name = Sudipta Kumar
Currently living in = Switzerland
Developer Salary = 5022.0/- CHF (Swiss Franc)
```

Figure 11.2: Setter Methods

Chapter 12

Inheritance

Inheritance means legacy. That means to get something which we don't have but
we can get them from our parents or ancestors. In java OOP, the task of the
inheritance is kind of same. We need to use the keyword named ‘‘extend’’ in
child class to get the abilities of the parent class. Here, the class who uses
the word extends it will be the child class. And the another one will be the
parent class.

Importance of Inheritance

- Code reusability.
- Method overriding.
- Implement child-parent relationship.

12.0.1 Syntax

```
class childClassName extends parentClassName{
   //statements
}
```

Types of Inheritance

There are 4 types of inheritance available. Those are,

- Single level inheritance
- Multi-level inheritance

- Hierarchical inheritance
- Multiple inheritances
- Hybrid

12.0.2 Single Level Inheritance

In this type of inheritance, there will be only one parent and one child.
More easily, we can say the father-child relationship. As like
public B extends A{ // Statements }

12.0.3 Multi-Level Inheritance

In this kind of inheritance, there will be grandfather-father-child relationship. *public B extends A{ // Statements }*
public C extends B{ // Statements }

12.0.4 Hierarchical Inheritance

In this inheritance, we can say this is a full family. Where a father has
multiple children. As Like, *public B extends A{ // Statements }*
public C extends A{ // Statements }

12.0.5 Multiple Inheritance

In this inheritance, we can say we'll have multiple parents but only one child
That means one class will extend 2 or multiple different classes, by using just
coma (‘,’).
public B extends A, C, D{ // Statements }

12.0.6 Hybrid Inheritance

This inheritance is the mix version of all other inheritances. That means we
can have multiple, multi-level, hierarchical inheritance at a time in specific
number of classes.

12.1 Method Overriding

Basically, method overriding means to override a function and customize as the
child class perspective, which function has already been defined in parent
class. To use method overriding in polymorphism, we need inheritance
overriding system.

12.1.1 Rules of Method Overriding

- Name, SignatureType, parameters all must be same.
- Can't override functions like static/final.
- Constructors can not be overridden.
- 2 class who has that same named function,
  must've Inheritance relationship.

12.1.2 Method Overloading VS Method Overriding

Method Overloading	Method Overriding
Parameter must be different inside a same class	Parameter must be different inside multiple class
Inheritance relationship is not mandatory	Inheritance relationship is not mandatory
Return type can be same or not	Return type must be same
A method never hides the another one	Child method hides the parents one

Table 12.1: Method Overloading VS Method Overriding

Chapter 13

Polymorphism

Polymorphism is to use the same function in different time and different type
situations to get different job done. Polymorphism occurs from 2 greek words
Poly (many) and Morph (form).

Just imagine you have a function which is used in different situations. That
means you can use this same named function but with different parameter types
and number of the parameters. As and example the '+' operator is used to add 2
numbers but if you can use this function to do something else when it's used in
between 2 different objects like 2 strings then it is called polymorphism.

Moreover we can use polymorphism by putting child class objects in parent
class type variableNames. By doing this that variable can use both parent
& child class functions and attributes.

13.1 Types of Polymorphism

There are 2 types of polymorphism available. Those are,

- Static or Compile-time Polymorphism
- Dynamic or Run-time Polymorphism

13.1.1 Static Polymorphism

Static polymorphism is the polymorphism which is used when we have a function, which is used in different situations. That means we can use this same named function but with different parameter types and number of the parameters.

13.1.2 Dynamic Polymorphism

It's called dynamic polymorphism when we put child class objects in parent class type variableNames. By doing this that variable can use both parent & child class functions and attributes.

13.1.3 Example

```
        class developer {

            public void add(int a, int b) {
                System.out.println(a + b);
            }

            public void add(int a, int b, int c) {
                System.out.println(a + b + c);
            }

            public void add(String string, String string2) {
                System.out.println("This Text in parent class");
            }

        }

        class Junior extends developer {

            public void add(String a, String b) {
                System.out.println(a + " " + b);
            }

        }

        public class Test {
            public static void main(String args[]) {
                // ********** Method Overloading **********
```

```
            developer d = new developer();
            d.add(2, 3);
            d.add(2, 3, 4);
            // ********* Method Overriding **********
            developer d2 = new Junior();
            d2.add("Hello", "World");
            d2.add(3, 3);
            d2 = new developer();
            d2.add("Hello", "World");

        }
    }
```

```
5
9
Hello World
6
This Text in parent class
```

Figure 13.1: Polymorphism[3]

Chapter 14

Abstraction

Abstraction is to hide the details of the implementation and show only the
things that a we want to show. Abstraction is to hide the implementation of
the process and show only the functionalities to the users. As an example,
we can that we all see the vending machine . We know that we put money, select
items and push the button, then we get that item, but we don't know how the
machine actually selects the specific ones and how it comes to us. That is an
example of abstraction. Where we can't see the actual process but kind of.
Abstraction has some features, which are,

- Abstract Class
- Abstract Method

14.1 Abstract Class

Abstraction has rules, which are,

- Abstract class have abstract ad non-abstract methods.
- we can not create object(new className) of an abstract class but, can
  create reference variable.
- Abstract class has abstract keyword before the class name.

14.2 Abstract Method

Abstraction has rules, which are,

- Abstract method has no body
- Abstract method must be ends with semicolon(';')
- Abstract method can be parametrized of non-parametrized
- Abstract method must be in abstract class
- If any class extends abstract class then it has to override those abstract methods also that is present in the abstract class
- Abstract method can not be declared as static or final

14.2.1 Example

```
abstract class developer {
    abstract void display();

    public void writeCode() {
        System.out.println("Writing code");
    }
}

class Junior extends developer {

    public void add(String a, String b) {
        System.out.println(a + " " + b);
    }

    @Override
    void display() {
        System.out.println("I am a junior developer");
    }

}

public class Test {
    public static void main(String args[]) {
        Junior j = new Junior();
        j.display();
```

```
    }
}
```

```
I am a junior developer
```

Figure 14.1: Abstract Class

14.3 Interface

Interface is just like the abstract class. The difference is, In the interface
we can not use normal methods. Here we can only use abstract methods. And the
other procedure is same as the abstract class. That's why it is called full
abstraction. To use this, we have to use ''interface'' keyword before the name
of the class & we've to use the keyword ''implements'' instead of ''extends''
keyword

14.3.1 Syntax

```
interface interfaceName {
    abstract void methodName();
}
```

14.3.2 Example

```
interface developer {
    void display();
}

class Junior implements developer {
    public void display() {
        System.out.println("I am a Junior");
    }
}

public class Test {
```

```
    public static void main(String args[]) {
        Junior j = new Junior();
        j.display();

    }
}
```

```
I am a junior developer
```

Figure 14.2: Interface Class

Chapter 15

Association

There is two most basic idea that you have to remember, of them first is, so
we all know about variables. int, String and lots more. Have you ever thing
that those dataTypes is also been written somewhere as a class. So, we also
can use our custom made classes as dataTypes. And the second is, this is
similar to the concept of RDBMS, means "Relational Database Management
System". where we can apply one to one relationship one to many and so on.
There are two types of association, which are,

- Composition (Strong Association)
- Aggregation(Weak Association)

15.1 Composition

Composition is also called strong association. It is, declared and initialized
in the class. That means, suppose, we created a class. Now we want to use it
as a dataType of a variable. so, first we have to declare the variable. Now
as we initialize the class objects by using new keyword, here we have to do
the same. By doing so, we are creating that kind of variable which data is
attached with the class in which it has been declared. that means

if we delete
the class object, then the data that the custom variable(class object)
holds will also get deleted automatically.

15.1.1 Syntax

```
className variableName = new className();
```

15.1.2 Example

```
class Address {
    String city;
    String state;
    String zip;
    Address() {
        // System.out.println("Empty Constructor");
    }
    Address(String city, String state, String zip) {
        this.city = city;
        this.state = state;
        this.zip = zip;
    }
    public String toString() {
        return "City = " + city + ", State = " + state + ", Zip = " + zip;
    }
}
class Person {
    Address address;
    String name;
    Person(String name, String city, String state, String zip) {
        this.name = name;
        address = new Address(city, state, zip);
    }
    public String toString() {
        return "Name = " + name + ", Address = " + address;
    }
}
public class Test {
    public static void main(String[] args) {
        Person person = new Person("Sudipta Kumar Das", "Zermatt", "Switzerland", "3920");
        System.out.println(person);
        /* Deleting Person Object and as the address is initialized inside the class,
        * it also get deleted automatically. */
        person = null;
        System.out.println("After Removing Person Object, Person is = " + person);
    }
}
```

```
Name = Sudipta Kumar Das, Address = City = Zermatt, State = Switzerland, Zip = 3920
After Removing Person Object = null
```

Figure 15.1: Composition Example[3]

15.2 Aggregation

Aggregation is also called weak association. It is just, declared and but not
initialized in the class. That means, suppose, we created a class. Now we want
to use it as a dataType of a variable. so, first we have to declare the
variable. Now as we won't initialize the class objects by using new keyword,
By doing so, we are creating that kind of variable which data is not attached
with the class in which it has been declared. that means if we delete the class
object, then the data that the custom variable(class object) holds won't get
deleted automatically.

15.2.1 Syntax

className variableName;

15.2.2 Example

```
class Address {
    String city;
    String state;
    String zip;
    Address() {
        // System.out.println("Empty Constructor");
    }
    Address(String city, String state, String zip) {
        this.city = city;
        this.state = state;
        this.zip = zip;
    }
    public String toString() {
        return "City = " + city + ", State = " + state + ", Zip = " + zip;
    }
}
class Person {
    Address address = new Address();
    String name;
    Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }
    public String toString() {
        return "Name = " + name + ", Address = " + address;
    }
}
public class Test {
```

```
    public static void main(String[] args) {
        Address address = new Address("Zermatt", "Switzerland", "3920");
        Person person = new Person("Sudipta Kumar Das", address);
        System.out.println(person);
        /* Deleting Person Object and as the address is not initialized inside the class,
        * it won't get deleted, but the person object will be deleted. */

        person = null;
        System.out.println("After Removing Person Object, Person is = " + person);
        System.out.println("After Removing Person Object, Address is = " + address);
    }
}
```

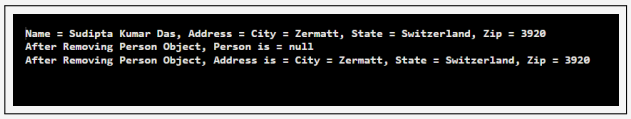

Figure 15.2: Aggregation Example[3]

Chapter 16

Keywords

There are some words which plays Important tasks in programming. These are
called keywords. There are lots of keywords in java. You'll get to know all
of them in future day by day. Now we will see some of them.

- Super keyword
- this keyword
- final keyword
- static Keyword

16.1 Super Keyword

To call any attribute or function or the constructor of the parent class, we
need to use ‘‘super" keyword. Moreover, we've to use ‘‘super" keyword to get
return any value from parent class inside the child class, so that the child
class can use it. We can use it in some situations. Those are,

- To call the constructor of the parent class.
- To call the function of the parent class.
- To call the attribute of the parent class.
- To send the value to the parent class.

16.1.1 Call the constructor of the parent class

super();

Example

```
class A {
    A() {
        System.out.println("Class A");
    }

    A(String X) {
        System.out.println("Class A with " + X);
    }
}

class B extends A {
    B() {
        super("Devs");
        System.out.println("Class B");
    }
}

public class Test {
    public static void main(String args[]) {
        B b = new B();
    }
}
```

```
Class A with Devs
Class B
```

Figure 16.1: Super Keyword Constructor Call

16.1.2 Call the attribute of the parent class

super.variableName;

Example

```
class A {
    int a = 10;
```

```
}

class B extends A {
    int a = 5;

    void display() {
        System.out.println(super.a);
        System.out.println(a);
    }
}

public class Test {
    public static void main(String args[]) {
        B b = new B();
        b.display();
    }
}
```

```
10
5
```

Figure 16.2: Super Keyword Variable Call

16.1.3 Insert Value into the parent class variables

super.variableName;

Example

```
class A {
    int a = 10;
}

class B extends A {
    int a = 5;

    void insert(int x) {
        super.a = x;
    }

    void show() {
```

```
        System.out.println(super.a);
    }
}

public class Test {
    public static void main(String args[]) {
        B b = new B();
        b.insert(8);
        b.show();
    }
}
```

```
8
```

Figure 16.3: Super Keyword Variable insert

16.1.4 Call the function of the parent class

super.functionName();

Example

```
class A {
    String getValue() {
        return "Hello From Parent";
    }
}

class B extends A {
    void show() {
        System.out.println(super.getValue());
    }
}

public class Test {
    public static void main(String args[]) {
        B b = new B();
        b.show();
    }
}
```

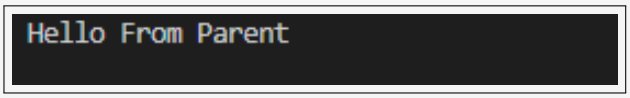

Figure 16.4: Super Keyword Function Call

16.2 This Keyword

By using ‘‘this" keyword, we can access the attributes and functions of the
same scope(Scope means {}). We can call the constructor of the current class
by this keyword. And also can pass it as an argument to the function

16.2.1 Syntax

this.functionName();
this.variableName();

Example

```
class developer {

    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public void show() {
        System.out.println("Hello " + name);
    }

}

public class Test {
    public static void main(String args[]) {
        developer d = new developer();
        d.setName("KUMAR");
        d.show();
    }
}
```

Figure 16.5: This Keyword

16.3 final Keyword

By using ‘‘final" keyword, we can make the variable as constant and restrict
the value of the variable to be changed after one time declaration and
initialization. Final can be used in
several cases like,

- Final Variable
- Blank Final Variable
- Static Blank Final Variable

16.3.1 Final Variable

If we use final keyword before any variable declaration, then the variable can
be used as constant and can not change the value. It’s better to use capital
letters for the final variable name.

16.3.2 Blank Final Variable

This is mainly used to get user input in a final variable.

16.3.3 Static Blank Final Variable

This kind of variable is used to store the static values but only one time. To
initialize the static variable, we have to use static block or constructor.

16.4 Static Keyword

Static keyword is basically used to access that particular thing by all the
objects. Static can be used in several cases like,

- Static variable
- Static Method
- Static Block

16.4.1 Static Method

Static methods are also used for the same case, that is can be accessed by
all the objects of that class. But a static method can only access static
variables. That, means non-static variables can not be used inside static
method.

Syntax

```
public static returnType methodName(){
    // Statements
}
```

16.4.2 Static Block

Static block is a structure which is used to initialize variables.

Syntax

```
static{
    // Statements
 }
```

Part V

Conclusion

Finally we are at the edge of learning the basics of JAVA. We have last few
things to learn. Now we'll learn about some error handling so that out
program do not get crashed and file handling for data storage in a simple
way. After that we'll learn some advanced things too, like linked-list and
hashmap.

Chapter 17

Exception Handling

To learn about the exception handling, first we have to know about what is an exception. So exception is kind of run time error. That means exception is an abnormal situation which we can not understand always on compile time. it occurs run time when the user or any network issues occurred. There are a few kinds of exceptions in java. Those are,

- Arithmetic exception
- Null Pointer exception
- String index out of bound exception
- Number format exception
- File not found exception
- Array index out of bound exception
- class not found exception
- IO exception
- No such method exception

17.1 Arithmetic Exception

This exception occurs when we try to divide a number by zero.

17.1.1 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: ");
        double num1 = input.nextDouble();
        System.out.print("Enter 2nd number: ");
        double num2 = input.nextDouble();
        double div = num1 / num2;
        System.out.println("Division is: " + div);
    }
}
```

Figure 17.1: Arithmetic Exception

17.2 Null Pointer exception

This exception occurs when we try to access a null value.

17.2.1 Example

```
public class Test {
    public static void main(String args[]) {
        String name = null;
        System.out.println(name.charAt(0));
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException
        at Test.main(Main.java:4)
```

Figure 17.2: Null pointer Exception

17.3 String index out of bound exception

This exception occurs when we try to access a string index which is out of
the range.

17.3.1 Example

```
public class Test {
    public static void main(String args[]) {
        String name = "KUMAR";
        System.out.println(name.charAt(6));
    }
}
```

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out
        at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)
        at java.base/java.lang.String.charAt(String.java:711)
        at Test.main(Main.java:4)
```

Figure 17.3: String Index Out of Bound Exception

17.4 Number Format Exception

This exception occurs when we try to convert a string to a number.

17.4.1 Example

```
public class Test {
    public static void main(String args[]) {
        int number = Integer.parseInt("KUMAR");
        System.out.println(number);
    }
}
```

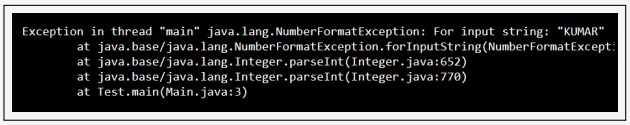

Figure 17.4: Number Format Exception

17.5 File Not Found Exception

This exception occurs when we try to access a file which is not present in
the location that we have given input.

17.5.1 Example

```
import java.io.File;
public class Test {
    public static void main(String args[]) {
        File file = new File("C://file.txt");
        System.out.println(file.canRead());
    }
}
```

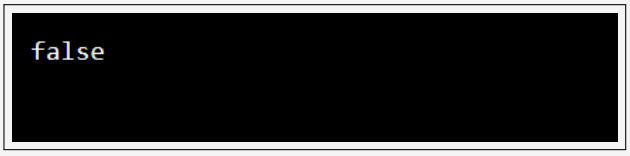

Figure 17.5: File Not Found Exception

17.6 Array Index Out of Bound Exception

This exception occurs when we try to access an array index which is out of
the size or range.

17.6.1 Example

```
import java.io.File;
public class Test {
    public static void main(String args[]) {
        int a[] = new int[5];
        a[7] = 32;
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 5
        at Test.main(Main.java:5)
```

Figure 17.6: Array Index Out of Bound Exception

17.7 Class Not Found Exception

This exception occurs when we try to access a class which is not present
in
the location that we have given input.

17.7.1 Example

```
public class Test {
    public static void main(String args[]) {
        Class.forName("NoClassExist");
    }
}
```

```
Main.java:3: error: unreported exception ClassNotFoundException; must be caught or declared to be thrown
        Class.forName("NoClassExist");
                     ^
1 error
```

Figure 17.7: Class Not Found Exception

17.8 IO Exception

This exception occurs when we try to read or write a file which is
not present
in the location that we have given input.

17.8.1 Example

```
import java.io.File;
import java.io.FileInputStream;
public class Test {
    public static void main(String args[]) {
        FileInputStream fileInputStream = new FileInputStream(new File("C://file.txt"));
        fileInputStream.read();
    }
}
```

```
Main.java:6: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
        FileInputStream fileInputStream = new FileInputStream(new File("C://file.txt"));
                                          ^
Main.java:7: error: unreported exception IOException; must be caught or declared to be thrown
        fileInputStream.read();
                        ^
2 errors
```

Figure 17.8: IO Exception

17.9 No Such Method Exception

This exception occurs when we try to access a method which is not present in
the class.

17.9.1 Example

```
class Chef{
    public void makeDesert(String name){
        System.out.println("Chef makes " + name);
    }
}

public class Test {
    public static void main(String[] args) {
        Chef chef = new Chef();
        chef.make("cake");
    }
}
```

17.10 Try-Catch-Throw & finally

You have already seen that if any kinds of exception occurs then our program

```
Main.java:10: error: cannot find symbol
        chef.make("cake");
            ^
  symbol:   method make(String)
  location: variable chef of type Chef
1 error
```

Figure 17.9: No Such Method Exception

gets terminated. To overcome this problem we use try-catch block.

17.10.1 Try block

The try block is used to place the code which may throw an exception.

Syntax

```
try {
   //code which may throw an exception
}
```

17.10.2 Catch block

The catch block is used to handle the exception. That means by catch block
we can see what kind of exception has occurred and we can handle it or not.
And the thing that should be done if any exception occurs.

Syntax

```
catch(Exception ex) {
   System.out.println("Error = "+ex.getMessage());
}
```

17.10.3 Finally block

We know that if exception occurs than the whole program gets terminated.
so to stop this, that means to execute the remaining code even if any exception occurs we use finally block. It indicates no matter what kinds

of exception occurs, the rest of the code will be executed.

Syntax

```
finally {
   //code which may throw an exception
}
```

17.10.4 throw

We know that if exception occurs than the whole program gets terminated. so to stop this, that means to execute the remaining code even if any exception occurs we use finally block. It indicates no matter what kinds
of exception occurs, the rest of the code will be executed.

Syntax

```
throw new errorName("Error message that you wanna show");
```

17.10.5 Example

```
class Developer{
    public static void giveError(){
        throw new RuntimeException("OOPS! We got an error");
    }
}
public class Test {
    public static void main(String[] args) {
        System.out.println("Before getting exception");
        try{
            int X = 2/0;
        }catch(Exception ex){
            System.out.println("Error = "+ex.getMessage());
        }finally{
            System.out.println("This part will always execute");
        }
        Developer.giveError();
    }
}
```

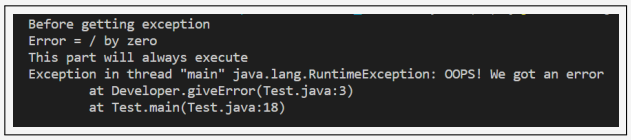

Figure 17.10: temp

Chapter 18

Advanced Java

18.1 Decimal Number Formatting

Decimal number formatting means to show how many digits after the decimal point.

18.1.1 Example

```
import java.text.DecimalFormat;
public class Test {
    public static void main(String[] args) {
        double x = 2.9875488;
        // C programming Style
        System.out.printf("X = %.2f", x);
        System.out.println("\n");
        // JAVA Style 2 Decimal places
        DecimalFormat df = new DecimalFormat("#.##");
        System.out.println("X = " + df.format(x));
    }
}
```

18.2 To String Method

The toString() method is used to convert any data type into string.
To be more specific, it is used to show all the attributes of an object.
t together.

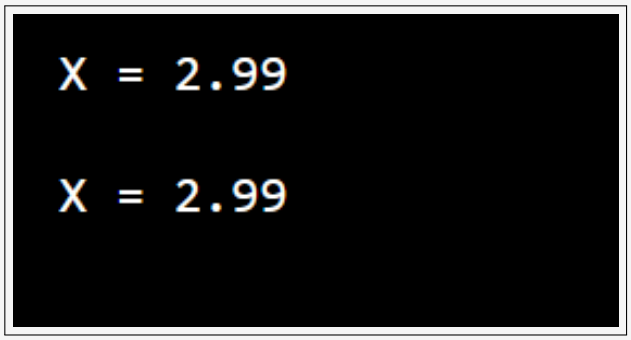

Figure 18.1: Decimal Number Formatting

18.2.1 Example

```
import java.text.DecimalFormat;
class Person{
    String name;
    int age;
    Person(String name, int age){
        this.name = name;
        this.age = age;
    }
    @Override
    public String toString(){
        return "Name: " + name + ", Age: " + age;
    }
}
public class Test {
    public static void main(String[] args) {
        Person p1 = new Person("KUMAR", 22);
        Person p2 = new Person("John", 25);
        System.out.println(p1);
        System.out.println(p2.toString());
    }
}
```

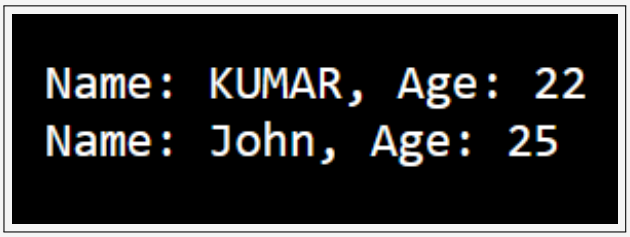

Figure 18.2: To String Example

18.3 Linked List

Basically it's also a dynamic array. Linked list means doubly linked list.
Doubly linked list has 3 different parts, of them 1st is store previous node
address. 2nd part is to store the value that we want to store. 3rd part is to
store next node address. And the other things are same as dynamic array or
array list.

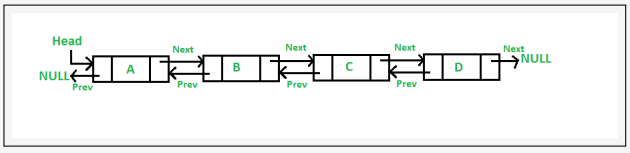

Figure 18.3: Linked List Diagram[11]

18.3.1 Example

```
import java.util.LinkedList;
public class Test {
    public static void main(String[] args) {
        LinkedList<String> countryNames = new LinkedList<String>();
        countryNames.add("India");
        countryNames.add("Sweden");
        countryNames.add("Finland");
        countryNames.add("Switzerland");
        System.out.println(countryNames);
    }
}
```

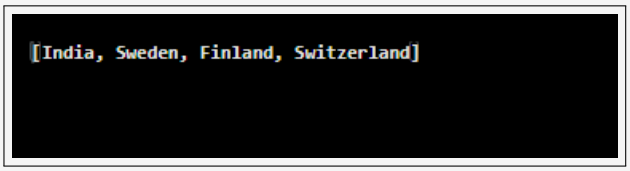

Figure 18.4: Linked List Example[3]

18.4 Hashmap

Hashmap is also a dynamic array.But we call it as a database. That means in
the hashmap, we can store the data but we have to use individual id to store
each data as like database. We also can call it as 2D hash table.

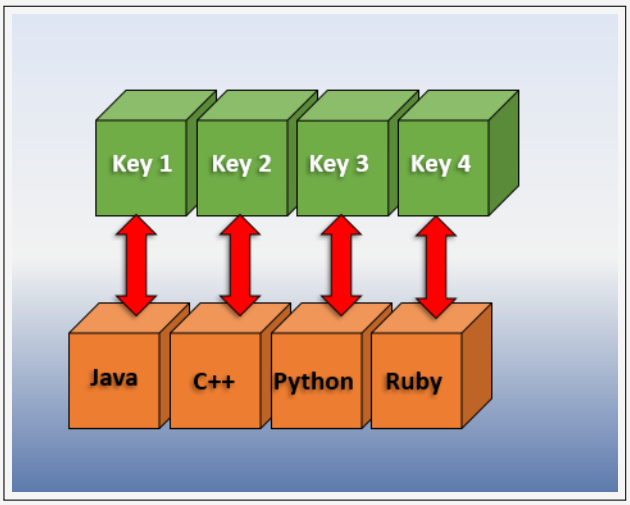

Figure 18.5: Linked List Diagram[11]

18.4.1 Example

```
import java.util.HashMap;
public class Test {
    public static void main(String[] args) {
        HashMap<String, String> countryNames = new HashMap<String, String>();
        countryNames.put("Key_1", "India");
        countryNames.put("Key_2", "Sweden");
```

```
            countryNames.put("Key_3", "Finland");
            countryNames.put("Key_4", "Switzerland");
            System.out.println(countryNames);
        }
    }
```

Figure 18.6: HashMap Example[3]

18.5 HashSet

The basic difference between arrayList and hashSet is that there can be
duplicate values in arrayList but not in hashSet.

18.5.1 Example

```
import java.util.HashSet;
public class Test {
    public static void main(String[] args) {
        HashSet<String> countryNames = new HashSet<String>();
        countryNames.add("India");
        countryNames.add("Sweden");
        countryNames.add("Finland");
        countryNames.add("Switzerland");
        countryNames.add("Switzerland");
        System.out.println(countryNames);
        countryNames.remove("Sweden");
        System.out.println(countryNames);
        countryNames.clear();
        System.out.println(countryNames);
    }
}
```

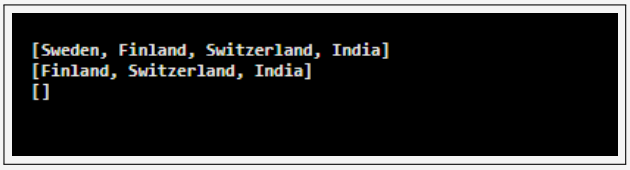

Figure 18.7: HashSet Example[3]

Chapter 19

File Handling

File handling exactly means is to read and write data from and to file.Though we can use database to store data, but we can't use database to store logfile data. So, that's why we use file to store logfile data.

19.1 Create a file

To create a file, we use File class.

19.1.1 Syntax

```
File file = new File("filename.txt");
file.mkdir();
```

19.1.2 Example

```
import java.io.File;
public class Test {
    public static void main(String[] args) {
        /**
        * Creating Directory in the same folder
        * where the java file is located
        */
        File dir = new File("file.txt");
        dir.mkdir(); // creates the directory
        /** To get the absolute location of the folder */
        System.out.println("Directory Path = " + dir.getAbsolutePath());
        System.out.println("File Name = " + dir.getName());
        System.out.println("Creating another file in same location...");
        File file2 = new File(dir.getAbsolutePath() + "/file2.txt");
        file2.mkdir();
        System.out.println("Directory Path_2= " + file2.getAbsolutePath());
        dir.delete(); // deletes the directory
```

```
            // File Creation should be inside try catch block
        }
    }
```

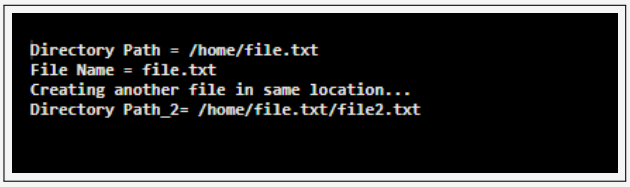

Figure 19.1: Create File Example[3]

19.2 Write a file

To write a file, we use Formatter class.

19.2.1 Syntax

```
Formatter formatter = new Formatter("filename.txt");
formatter.format("%s %s \r\n", "value1", "value2");
```

19.2.2 Example

```
import java.io.File;
import java.util.Formatter;
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        // Formatting means Re-Write into a already created file
        // Formatter should be inside the try catch block
        try {
            Formatter formatter = new Formatter("file.txt");
            Scanner input = new Scanner(System.in);
            System.out.print("How many Students = ");
            int number = input.nextInt();
            input.nextLine();
            int flag = 0;
            for (int i = 0; i < number; i++) {
                System.out.print("Enter Student Name = ");
                String name = input.nextLine();
                System.out.print("Enter Student ID = ");
                String ID = input.nextLine();
                formatter.format("%s - %s \r\n", name, ID);
                // %S indicates to write a value in string format
                flag = 1;
            }
```

```
            if (flag == 0) {
                System.out.println("Could not Be inserted into file");
            } else {
                System.out.println("Successfully inserted into file");
                formatter.close(); // Closing the formatter
            }
        } catch (Exception e) {
            System.out.println("Error = " + e.getMessage());
        }
    }
}
```

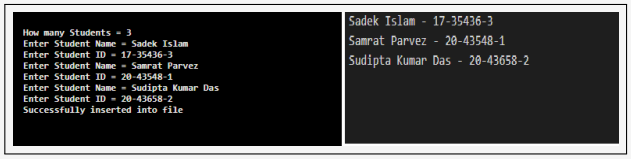

Figure 19.2: Write File Example[3]

19.3 Read a file

To read a file, we use Scannerclass.

19.3.1 Syntax

```
File file = new File("filename.txt");
Scanner scanner = new Scanner(file);
scanner.next();
```

19.3.2 Example

```
import java.io.File;
import java.util.Formatter;
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        // Formatting means Re-Write into a already created file
        // Formatter should be inside the try catch block
        try {
            File file = new File("file.txt");
            Scanner scanner = new Scanner(file);
            while (scanner.hasNext()) {
                String name = scanner.nextLine();
                System.out.println(name);
            }
```

```
        } catch (Exception e) {
            System.out.println("Error = " + e.getMessage());
        }
    }
}
```

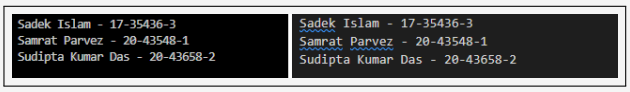

Figure 19.3: Read File Example[3]

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/

[11] Linked List, https://www.geeksforgeeks.org/doubly-linked-list/