JAVA Simplica

1st Edition, Sudipta Kumar Das

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


  _____ _____ __ __ _____ _      _____ _____
 /  ___| |  _  | |  _  | |      |_   _/  __ \     /\
 \ `--.| | | | | | | | | |        | | | /  \/    /  \
  `--. \ | | | | | | | | |        | | | |       / /\ \
 /\__/ / |_| | | | | | | |____  _ | |_| \__/\   / ____ \
 \____/ |_____/ |_| |_| \_____/(_)\___/ \____/  /_/    \_\
```

2022, Publisher

Contents

III   Basics of JAVA   24

List of Figures

List of Tables

Part I

Introduction

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

Preface

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

Chapter 1

History of JAVA

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

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

Figure 1.1: James Gosling[2]

Part II

Pre-Basic of JAVA

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

Chapter 2

Package & Class Declaration

2.1 package

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

2.1.1 Syntax

import packageName.subPackageName.className

2.1.2 Example

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

2.2 Access modifiers

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

- Public
- Private
- Protected
- Default

2.2.1 Public

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

2.2.2 Private

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

2.2.3 Protected

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

2.2.4 Default

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

2.2.5 Syntax

```
Access_modifier dataType/returnType variableName/methodName()
```

2.2.6 Example

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

2.3 Class Declaration

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

2.3.1 Syntax

```
Access_modifier class className
```

2.3.2 Example

```
public class Mobile{

}
```

2.4   Main Method

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

2.4.1   Syntax

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

2.4.2   Example

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

    }
}
```

2.5   Show Output in JAVA

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

2.5.1   Syntax

$$System.out.println();$$

2.5.2   Example

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

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

$$"\backslash escapeCharacter"$$

3.1.2 Example

```
public class Test {
```

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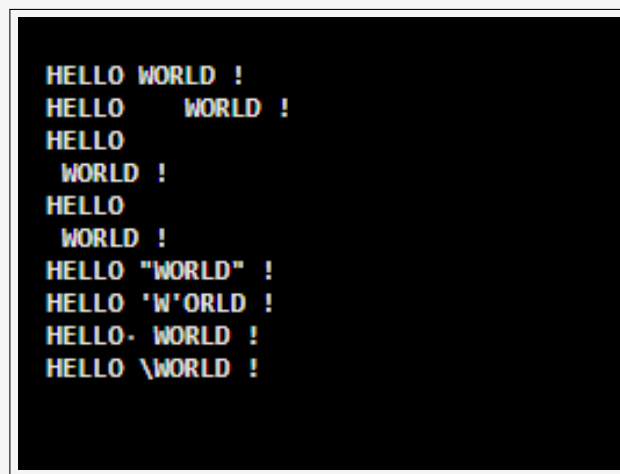

Figure 3.1: Escape Sequences[4][3]

3.2   Format Specifier

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

3.2.1   Syntax

$$"\%formatSpecifier"$$

3.2.2   Example

```
public class Test {
```

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

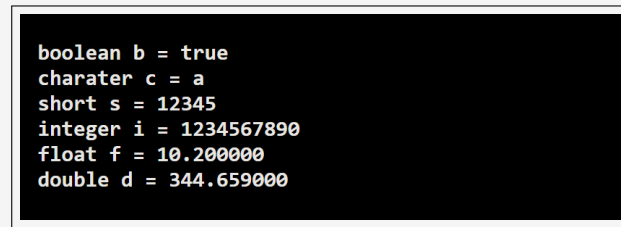Table 3.2: Format Specifier

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

3.3   Comments

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

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

Figure 3.2: Format Specifier[5][3]

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

- Single line comments

- Multi line comments

- Documentation comments

3.3.1 Single Line Comments

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

3.3.2 Multi Line Comments

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

3.3.3 Documentation Comments

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

3.3.4 Syntax

- Single Line Comments → //Comments

- Multi Line Comments → /*Comments*/

- Documentation Comments → /**Comments*/

3.3.5 Example

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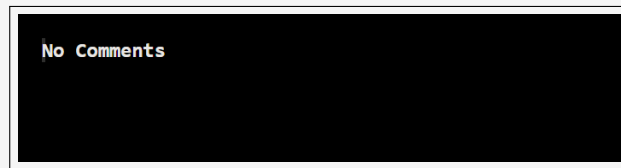

Figure 3.3: Comments[3]

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 appropiately. So, in that case, we have to take user inputs. For this we have to declare an object of Scanner class. In short, we have to write this line must *Scanner input = new Scanner(System.in);* . And to take input we have to use input.nextDataType(), here if we want to take integer, then we have to use input.nextInt();

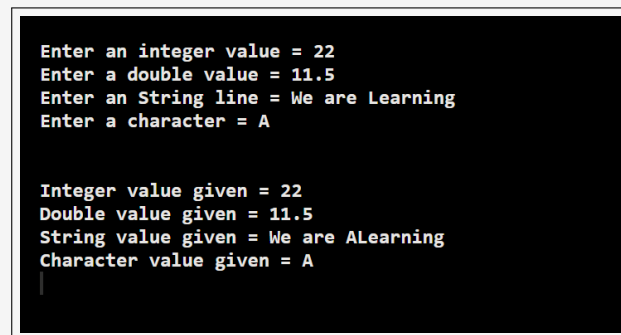| Method	Description
nextBoolean()	Reads Boolean values
nextByte()	Reads byte value
nextDouble()	Reads double value
nextFloat()	Reads float value
nextInt()	Reads int value
nextLine()	Reads String value
nextLong()	Reads long value
nextShort()	Reads short value
next().charAt(0)	Reads Char value

Table 3.3: User Input Type

3.4.1 Syntax

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

3.4.2 Example

```java
import java.util.Scanner;

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

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


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

Figure 3.4: User Input[7][3]

Part III

Basics of JAVA

JAVA is a vast programming language, but it has some basic things too, on whichs 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.

4.2   Data Types

Data Types[6] are basically types of pots that are used to store data inside it. As an example, if we want to store a football we can just use net(Not Internet) bags. But if we want to store water then we'll need bottles. Data Types are same. if we want to store integer numbers then we have to use integer type of variable not the boolean or another type.
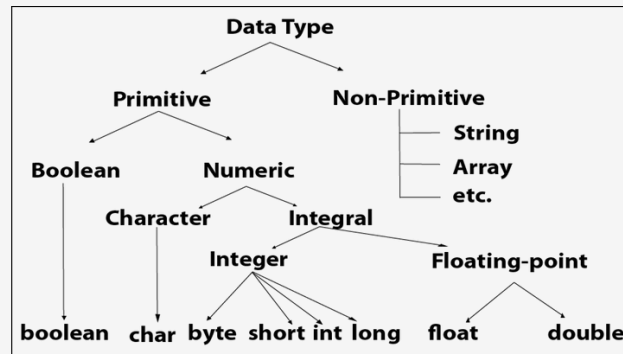
Figure 4.2: Data Types[6]

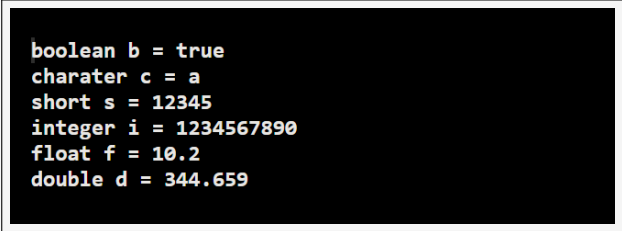| Type Name	Description	Size	Range	Simpel Declaration & Initialization
boolaen	true or false	1 Bit	{true,false}	boolean x = true;
char	Unicode Character	2 Byte	u0000 to uFFFF	char x = 'a';
byte	Signed Integer	1 Byte	-128 to 127	byte x = 12;
short	Signed Integer	2 Byte	-32768 to 32767	short x = 12345;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

Table 4.1: Data Type

4.2.1 Syntax

accessModifier dataType variableName;

4.2.2 Example

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

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

Figure 4.3: Data Type Chart[3]

Chapter 5

Operators

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

- Arithmetic Operators

- Assignment Operators

- Unary Operators

- Relational Operators

- Logical Operators

- Bitwise Operators
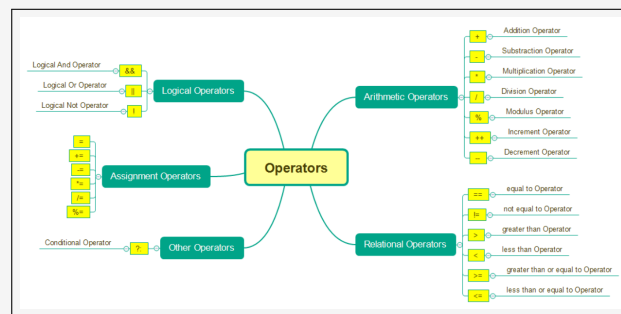
- Shift Operators

- Ternary Operators

Figure 5.1: Operators[8]

5.1   Arithmetic Operators

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

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

Table 5.1: Arithmetic Table

5.2   Assignment Operators

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

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

Table 5.2: Assignment Operators

5.3   Unary Operators

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

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

Table 5.3: Unary Operators

are,

- Prefix

- Postfix

5.3.1   Prefix

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

Unary Operator	Meaning
++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/decrement
their value.

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

Table 5.5: Postfix Operators

5.4   Relational Operators

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

Operator	Use	Description
>	Op1>Op2	Greater Than
>=	Op1>=Op2	Greater Than Equal
<	Op1<Op2	Less Than
<=	Op1<=Op2	Less Than Equal
==	Op1==Op2	Both are Equal
!	Op1!=Op2	Are not Equal

Table 5.6: Relational Operators

5.5 Bitwise Operators

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

Operator	Description
&	Bitwise AND
ˆ	Bitwise Exclusive OR
|	Bitwise Inclusive OR

Table 5.7: Bitwise Operators

5.6 Logical Operators

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

Operators	Description
&&	Logical AND
||	Logical OR

Table 5.8: Logical Operators

5.7 Ternary Operators

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

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

Table 5.9: Ternary Operators

5.8 Shift Operators

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

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

Table 5.10: Shift Operators

Chapter 6

Control Statements

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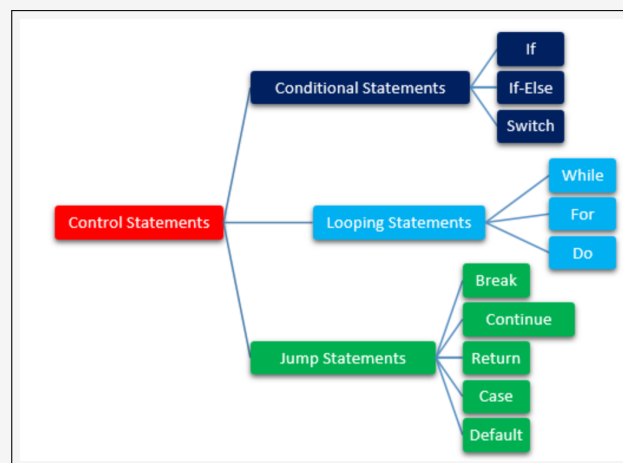after each and every statement finishes.

Figure 6.1: Control Statements[3]

6.1   Conditional Statements

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

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

- if-else

- if-else if-else

- switch

6.1.1   If-Else

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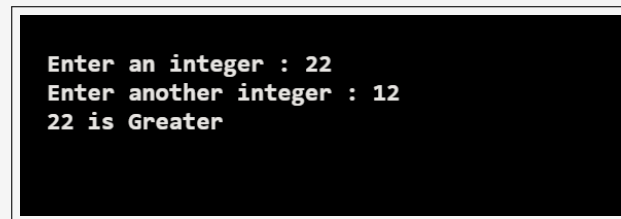It will be perfectly alright if we do not use else here.

Syntax

$$if(condition)\{statements;\}$$
$$else\{statements;\}$$

6.1.2 Example

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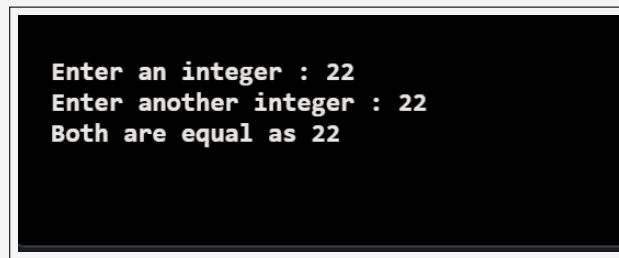
Syntax

$$if(condition)\{statements;\}$$
$$else \ if(condition)\{statements;\}$$
$$else \ if(condition)\{statements;\}$$
$$else\{statements;\}$$

6.1.4   Example

```java
import java.util.Scanner;

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

    }
}
```



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

Figure 6.3: If-Else Condition

6.1.5   Switch

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

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

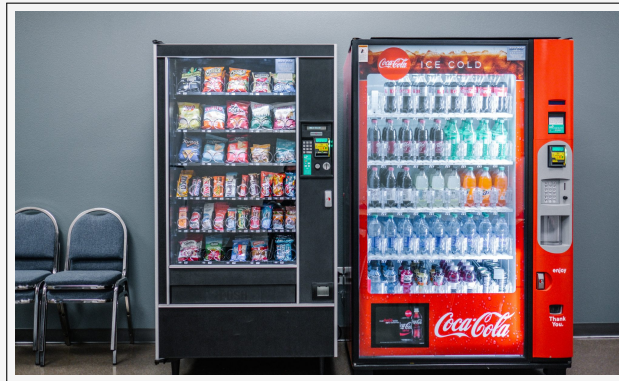- Single Digit Numbers (0 $\rightarrow$ 9)

- Alphabets (A-Z) or (a-z)

Figure 6.4: Vending Machine[10]

Syntax

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

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

6.1.6   Example

```
import java.util.Scanner;

public class Test{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        int x, numberofCokes = 0, numberofChips = 0;
        char y = 'a';
        double price = 0;
        System.out.println("-------------------- COKES -----------------------");
        System.out.println("                    Press 1 for COCACOLA                 |");
        System.out.println("                    Press 2 for PEPSI                    |");
        System.out.println("--------------------------------------------------------");
        System.out.print("              Enter your Choice >> ");
        x = input.nextInt();
        // If input type is 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 Addtion result of the whole series is = " + result);
}
}
```

6.2.2   While Loops

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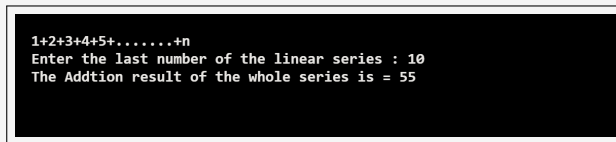As an exaple, we can do a repeated task until the user put a specific value.

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

Figure 6.6: For Loop

Syntax

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

Example

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

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

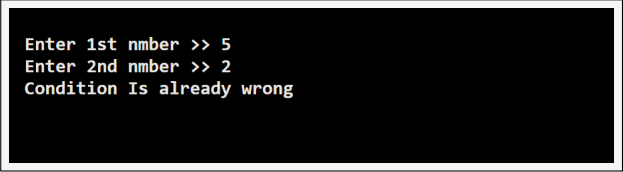Figure 6.7: While Loop[3]

6.2.3   Do-While Loops

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

Syntax

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

Example

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

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


```

Figure 6.8: Do While Loop[3]

6.2.4   For Each Loops

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

Syntax

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

Example

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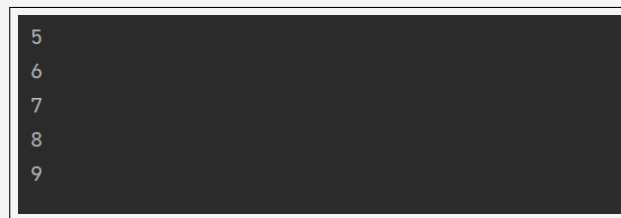

Figure 6.9: For Each Loop

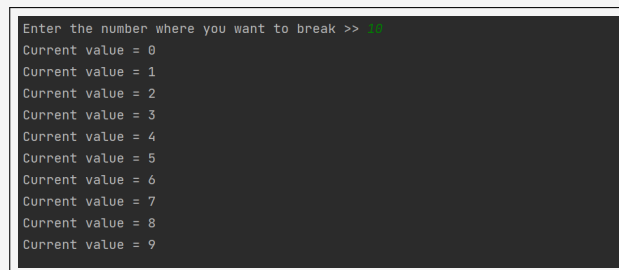## 6.3   Jump Statements

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

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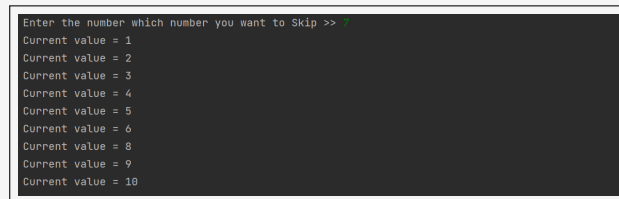

Figure 6.10: Break

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

```java
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++;
        }
    }
}
```


Figure 6.11: Continue

Chapter 7

Array

Array means declaration of bunch of variables of same type at a time. It means if
we want to declare 50 variables for 50 data, and we don't use array, then we have to
declare them one by one manually. But if we use array then we just declare once with
data type and we'll tell it to declare 50 variables then it'll declare 50 variables
automatically it's own.
To create an array we need new keyword. Sometimes the size of array can be pre defined
sometimes the user have the choice to declare the size of an array. These are called
dynamic memory allocation. This type of animation is also called non-primitives.
To see the length of the array, we have to use .length keyword.

Syntax of Array Length

int variableName = arrayName.length;

Example of Array Length

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

Syntax of Sorting an Array

Arrays.sort(arrayName);

Example of Sorting an Array

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

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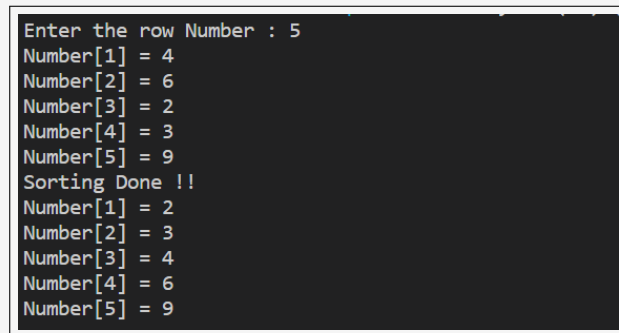
```
            int[] arr = new int[n];

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

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

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

Figure 7.1: Array Sorting[3]

Classification of Arrays

```
There are 2 kinds of arrays.  Those are,
```

- One Dimensional

- Two Dimensional

7.1 One Dimensional Array

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

7.1.1   Syntax

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

7.1.2   Example

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


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[][];
```

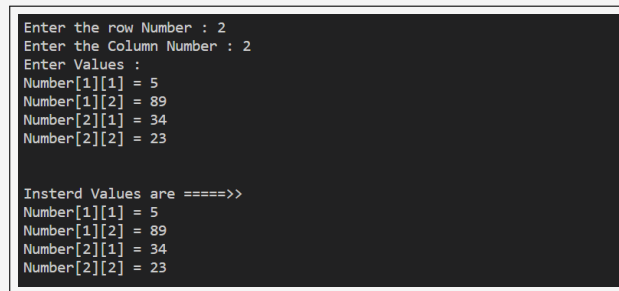### 7.2.2   Example

```
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();
```

```java
            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]);
                }
            }
        }
    }
```


Figure 7.3: temp[3]

7.3 ArrayList

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

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

Table 7.1: Array VS ArrayList

7.3.1   Syntax

```
ArrayList<dataType>variableName = new ArrayList();
```
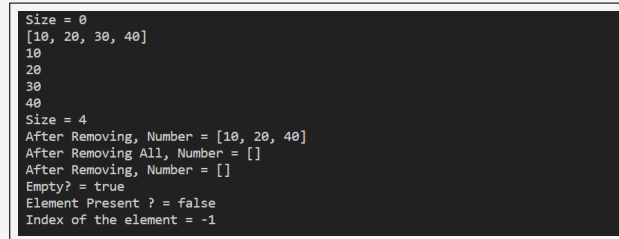
7.3.2   Example

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

import javax.print.attribute.standard.NumberUp;
public class Test {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        ArrayList<Integer>number = new ArrayList<>();
        System.out.println("Size = "+number.size());
        number.add(10);
        number.add(20);
        number.add(30);
        number.add(40);
        System.out.println(number); // Horizontal Print
        //using for each loop
        for(int x: number){
            System.out.println(x);
        }
        System.out.println("Size = "+number.size());
        // Removing Elements
        number.remove(2);
        System.out.println("After Removing, Number = "+number);
        // REmoving all elements
        number.removeAll(number);
        System.out.println("After Removing All, Number = "+number);
        // Removing all Elements
        number.clear();
        System.out.println("After Removing, Number = "+number);
        boolean b = number.isEmpty();// Return true if empty
        System.out.println("Empty? = "+b);
        // Contaisn 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 {
```
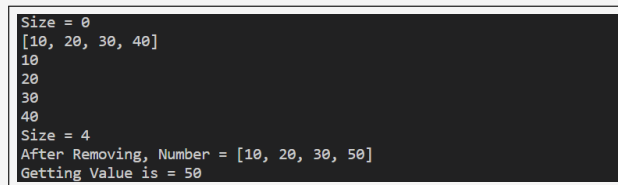
```
public static void main(String args[]) {
    Scanner input = new Scanner(System.in);
    ArrayList<Integer>number = new ArrayList<>();
    System.out.println("Size = "+number.size());
    number.add(10);
    number.add(20);
    number.add(30);
    number.add(40);
    System.out.println(number); // 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);
}
}
```

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

Figure 7.5: ArrayList Set Get[3]

7.3.6   ArrayList Methods

7.3.7   Sorting ArrayList

Syntax

7.3.8   Syntax

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

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

Index

Bibliography

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

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

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

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

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

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

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

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

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

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