

Programming Language I

CSE110 | Concise Java Notes



Inspiring Excellence

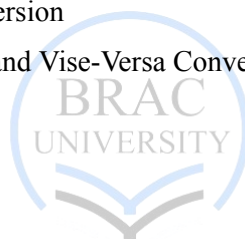
School of Data and Sciences | Brac/Abed University

Last Updated: July 11, 2023

The goal of this note is to aid the students, the instructors and any programming enthusiast in grasping the introductory Java programming concepts in a short amount of time. After covering the concise notes, enthusiasts are encouraged to go through the books as well as online materials.

TABLE OF CONTENTS

CHAPTER 1: FLOWCHART CONSTRUCTION	4
1.1 Introduction	4
1.2 Significance of the Shapes	4
1.3 Converting Scenarios into Flowcharts	4
1.4 Worksheet	7
CHAPTER 2: VARIABLES AND DATATYPES	8
2.1 Variables	8
2.2 Variable Naming Convention	8
2.3 Reserved Keywords in Java	8
2.4 Datatypes	8
2.5 Primitive vs Non-Primitive	9
2.6 Declaration, Assignment and Initialization	10
2.7 Datatype Conversion Basics	10
2.8 Numeric-to-Numeric Conversion	11
2.9 Numeric-to-Non-Numeric and Vice-Versa Conversion	12
2.10 Worksheet	13
CHAPTER 3: OPERATORS	14
4.1 What are Operators?	14
4.2 Arithmetic Operators	14
4.3 Assignment operators	14
4.4 Relational Operators	15
4.5 Logical Operators	15
4.6 Bitwise Operators	15
4.7 Unary Operators	15
4.8 Operator Precedence	16
4.9 Worksheet	17
CHAPTER 4: INTRODUCTION TO JAVA CODING	18
5.1 Essentials	18
5.2 Scopes	18
5.3 Printing	19
5.4 Comments	20
5.5 Worksheet	21
CHAPTER 5: USER INPUT	22



Inspiring Excellence

3.1	Creating a Scanner	22
3.2	Inputting Different Datatypes	22
3.3	Code Examples	22
3.4	Worksheet	24
CHAPTER 6: UNDERSTANDING ERRORS		25
6.1	Preamble	25
6.2	Syntax Errors	25
6.3	Logical Errors	25
6.4	Runtime Errors or Exceptions	26
6.5	Worksheet	26
CHAPTER 7: DECISION MAKING		27
7.1	“If” Single Selection Statement	27
7.2	“If - Else” Double Selection Statement	28
7.3	“If - Else If - Else” Multiple Selection Statement	30
7.4	Nested Decision Making	32
7.5	Switch Case Conditioning	33
7.6	Flowcharts	33
7.7	Worksheet	36
CHAPTER 8: REPETITIONS		38
8.1	Introducing Loops: Understanding When to Use Repetition	38
8.2	While Loop and For Loop	38
8.3	Do-While Loop	39
8.4	Infinite Loop	40
8.5	“Break” and “Continue” Keywords	40
8.6	Nested Loops	41
8.7	Definite vs Indefinite Repetition	41
8.8	Flowcharts	42
8.9	Worksheet	43
CHAPTER 9: STRINGS		44
9.1	Declaration and Initialization	44
9.2	String Input	44
9.3	String Escape Sequence	45
9.4	String Length	45
9.5	String Indexing	45



9.6	Character Array	46
9.7	Mutability of Strings	46
9.8	String Concatenation	46
9.9	Comparing Strings	47
9.10	Searching a Character in a String	48
9.11	String Slicing	48
9.12	Modifying a String	48
9.13	Trimming and Splitting a String	49
9.14	Case Conversion	49
9.15	Data Conversion in Strings	49
9.16	ASCII Values	50
9.17	Checking Substring Existence	50
9.18	Worksheet	51
CHAPTER 10: ARRAYS		53
10.1	Properties	53
10.2	Declaration, Creation and Initialization	53
10.3	Array Iteration	57
10.4	Operations on an Array	59
10.5	Multidimensional Arrays	59
10.6	Worksheet	60
CHAPTER 11: ARRAY SORTING		62
11.1	Selection Sort	62
11.2	Bubble Sort	64
11.3	Insertion Sort	67
11.4	Worksheet	71



Inspiring Excellence

CHAPTER 1: FLOWCHART CONSTRUCTION

1.1 INTRODUCTION

What is a flowchart?







A flowchart is a pictorial way of visualizing each step of a program/code using distinct shapes to represent different functionalities of the code.

Why do we use it?

Flowcharts help us to understand the chronological order of a program, that is, which line of code will be executed after which line.

1.2 SIGNIFICANCE OF THE SHAPES

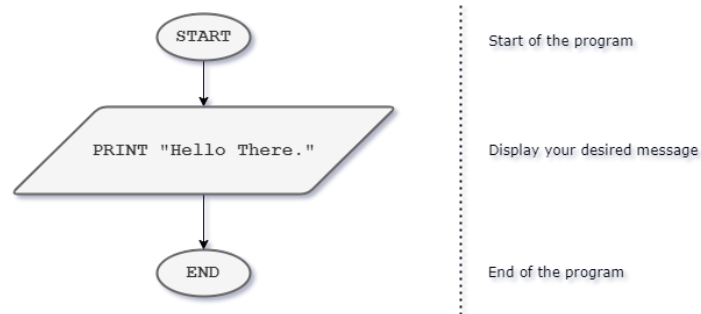
A flowchart uses different shapes to exhibit different operations of a code. A flowchart contains the following shapes:

Shape	Functionality	Visual Representation
Ellipse	Used to illustrate the “start” and “end” of a program	
Parallelogram	Used for inputs (prompt messages and user inputs) and outputs (print statements)	
Rectangle	Used for calculations and initializing variables	
Diamond	Used as conditional statements (if/else, loops)	
Circle	Used as connectors where multiple blocks of code either converge or diverge	
Arrow	Used to illustrate the sequence of the program by pointing from one shape to another	

1.3 CONVERTING SCENARIOS INTO FLOWCHARTS

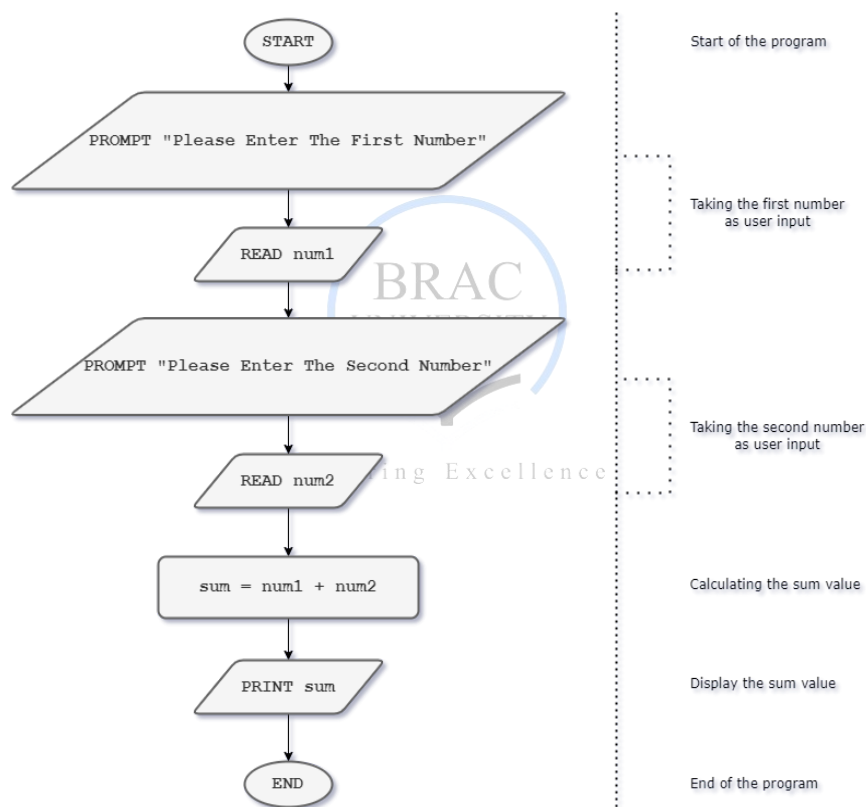
Till now, you have the basic introduction to flowcharts and you are familiar with the shapes of a flowchart. Now let's apply our knowledge in some scenarios.

- ✓ **Scenario 1:** Let's design a flowchart of a program that will display the message “Hello There.” and observe the steps carefully below.



For displaying anything, we use the PRINT statement. Simple, isn't it? Now can you design a flowchart of a program that displays your name? Try it by yourself.

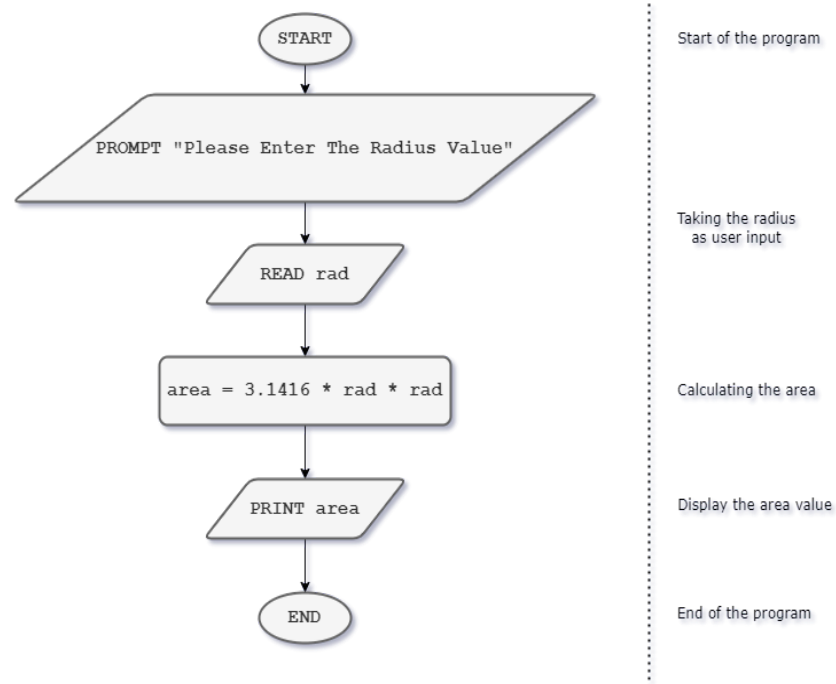
- ✓ **Scenario 2:** Now, let's talk about some numbers. Suppose, we want to display the summation of two numbers. You need to take these two numbers as user input and store them somewhere. Let's see the flowchart below:



To take user inputs, we need PROMPT and READ. PROMPT is used for displaying a prompt message to the user and READ is used for storing the value from a variable (we will learn all the details about variables in Chapter 2). In this scenario, after taking two number inputs, we can access those values from variables num1 and num2 respectively. We can then store the summation of these two numbers in another variable called sum and display it.

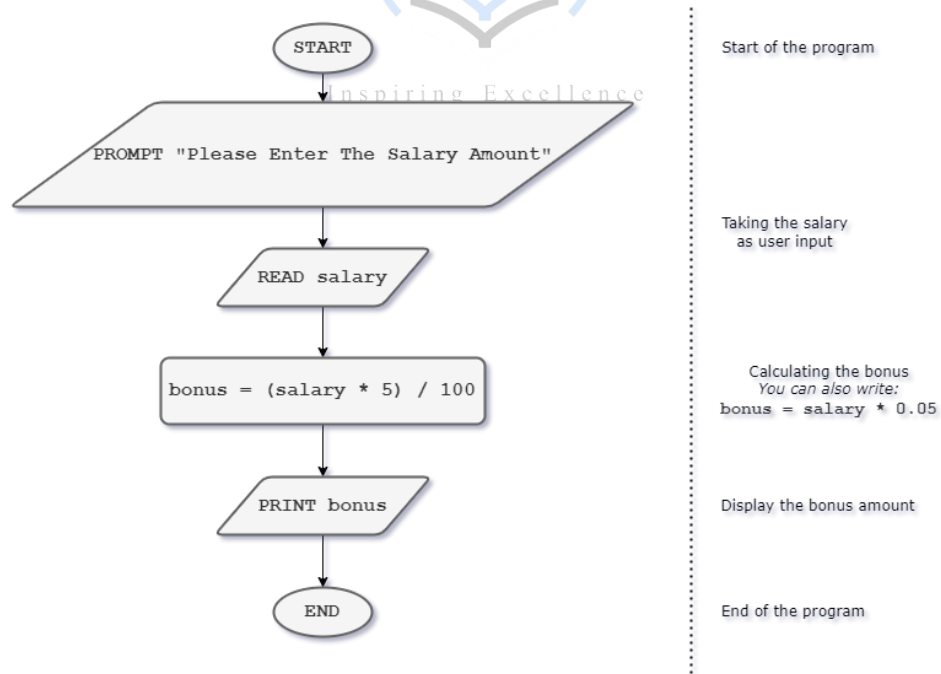
Now, hopefully you can design flowcharts of adding, subtracting, multiplying two or more numbers and many more.

- ✓ **Scenario 3:** Let's calculate the area of a circle now. You know the formula of calculating its area. Your task is to take the value of the circle radius from the user and then calculate the area. For simplicity, assume the value of pi will be 3.1416 for this problem. Let's design the flowchart:



Similarly, you will be able to calculate the circumference of the circle. Additionally, you can design flowcharts by calculating the area of squares, triangles, rectangles and so on.

- ✓ **Scenario 4:** Suppose, you are a loyal employee of a renowned company. You need to calculate the bonus you will receive during the upcoming festive season. The bonus will be 5% of your monthly salary. Let's design a simple flowchart for that.



Now, let's assume, at the end of each month, you have to pay 2.5% of your monthly salary as income tax. So, your festival bonus is now calculated using 5% of your monthly salary after deducting the income tax. Think what steps need to be modified and re-design the above flowchart.



Inspiring Excellence

1.4 WORKSHEET

- A. Write a short description and draw a flowchart to make a cup of tea or coffee. The steps could vary person to person.
- B. Draw a flowchart to find the result of addition of two numbers given by a user. Also, find their average.
- C. Draw a flowchart to take temperature in Fahrenheit and convert it to Celsius and print it.
- D. Draw a flowchart which takes a number from the user and then multiply it with 10, then add 2050 to the result and divide the number by 5 and show the final result as output.
- E. Write a flowchart that finds the number of hours, minutes, and seconds in a given number of seconds. For example, how many hours, minutes, and seconds are there in 10,000 seconds?
- F. Write a flowchart to print the area of a triangle taking base and height input from the user.
- G. Write a flowchart that reads the values for the three sides x, y, and z of a triangle, and then calculates its area. The area is calculated as follows:
$$\text{Area} = s(s - x)(s - y)(s - z), \text{ where } s \text{ is } (x + y + z)/2$$
- H. Assume there are two variables x and y. Take Values of these variables from the user. For example, the user gave the following two values.
 $x = 46;$
 $y = 27;$
Now exchange / swap values in such a way so that printing the variable x gives 27 and y gives 46.
- I. Take the value of a, b, c from the user. Then swap the values and print in such a way that:
Value of a goes to b
Value of b goes to c
Value of c goes to a
- J. Suppose a government employee has to pay 3000 taka plus 2.5% tax of his yearly salary. Now take the salary input from a user and print his tax.

CHAPTER 2: VARIABLES AND DATATYPES

2.1 VARIABLES

What are variables?

Variables are identifiers that are used to store data values. They are used to reserve a storage space, so that data can be accessed and modified.

2.2 VARIABLE NAMING CONVENTION

In order to create variables in java, certain rules or conventions must be followed. They are:

- Must begin with either letters (A-Z or a-z), dollar sign (\$) or underscore (_).
- They are case sensitive, i.e.- “name” and “Name” are two separate variables.
- Cannot have any white spaces.
- Can have numbers (0-9).
- Cannot begin with a number or any special signs.
- Can use **camel casing** (yourFirstName) or **snake casing** (your_first_name) for multiple words in a variable.
- Cannot be a reserved keyword (table given below).
- Should be meaningful.
- Should not be a single character.

Valid		Invalid	
a123bc	abc123	123Abc	ab cd
first_name	firstName	first name	Ab.cd
Int	\$ami	#number	if
_ami	class_1	class	int

Always try to keep your variable names meaningful and close to the context. Otherwise, the code will not be readable by you or anyone else.

2.3 RESERVED KEYWORDS IN JAVA

The following table contains a list of words that have a special meaning in java and cannot be used as the name of an identifier or a variable. Java is a case-sensitive language. Therefore, you can use the reserved words after changing the case of any of the characters. However, this is not recommended. While writing your code, you should be cautious towards not using any of the reserved keywords.

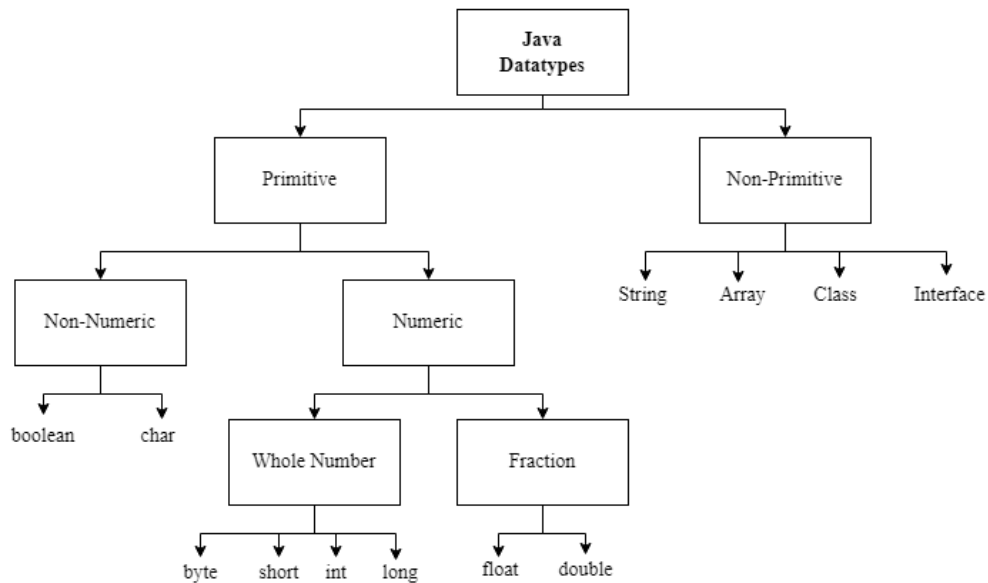
abstract	assert	boolean	synchronized	byte	case	catch	char	class
continue	default	do	double	else	enum	extends	final	finally
float	for	if	implements	import	int	long	native	new
null	package	private	instanceOf	public	return	short	static	strictfp
super	switch	break	interface	this	throw	throws	transient	try
void	volatile	while	protected	const	goto	true	false	null
exports*	module*	requires*	var**					

*New in Java 9 | **New in Java 10

2.4 DATATYPES

Every variable in Java must be of a specific datatype. Datatypes can be categorized as follows:

- Primitive Data Types: Holds a value, represents the size and type of a variable
- Non-Primitive Data Types: Holds a reference to an object or location



Category	Name	Size (bytes)	Possible Values/Explanation	Examples
Primitive	byte	1	Whole numbers from -128 or $-(2^7)$ to 127 or $(2^7)-1$	byte b1 = 51;
	short	2	Whole numbers from -32,768 or $-(2^{15})$ to 32,767 or $(2^{15})-1$	short s1 = 513;
	int	4	Whole numbers from -2,147,483,648 or $-(2^{31})$ to 2,147,483,647 or $(2^{31})-1$	int i1 = 5456;
	long	8	Whole numbers from -2^{63} to $2^{63}-1$	long l1= 12345788;
	float	4	Fractional numbers with at most 6 to 7 decimal digits	float f1= 5.0f;
	double	8	Fractional numbers with at most 15 decimal digits	double d1= 123.4558428743d;
	boolean	1	Either true or false	boolean b1 = true;
	char	2	A single character from the ASCII code	char c1 = 'A';
Non-Primitive	String	Same size	A sequence of characters	String s1 = "Hello";
	Arrays	Same size	Multiple data of the same datatype	int [] arr = {1, 2, 3};

Double and double, Integer and int, Float and float all exist in Java. However, Double, Integer, and Float etc are non-primitive datatypes (classes) where double, int and float are primitive. We shall learn more about classes and interfaces later on.

2.5 PRIMITIVE VS NON-PRIMITIVE

The differences of primitive and non-primitive datatype are shown below:

Primitive	Non-Primitive
Predefined, no need to create before using	Must be created (except for String) before using
Contains a value	Contains a reference
Cannot be null	Can be null
The size of a primitive type depends on the data type	All non-primitive types have the same size.

2.6 DECLARATION, ASSIGNMENT AND INITIALIZATION

Declaration: Allocating memory space while specifying the datatype of a variable.

Assignment: Assigning a value to a variable.

Initialization: Allocating memory space while specifying the datatype and assigning a value.

int a; //declaration a= 5; //assignment	int a = 5; //initialization
--	-----------------------------

How to initialize a variable?

<datatype> <variable name> = <value> < format specifiers >; // format specifiers are only for double and float

Examples:

```
byte b1 = 51;
short s1 = 513;
int i1 = 5456;
long l1 = 12345788;
float f1 = 5.0f;
double d1 = 123.4558428743d;
boolean b1 = true;
char c1 = 'A';
String s1 = "Hello";
int [] a = {10, 20, 100};
```

If you paid attention, you should have noticed that for float and double there are an f and d respectively after the values. These are format specifiers for float and double literals. You do not need format specifiers for any other datatype. Values of char datatype are enclosed in single quotation (') and values of String datatype are enclosed in double quotation ("). Length of a String can be 1 or more but char can have only one character.

String a = "ABC"; //Valid	byte a = 1234; //Invalid
String b = "A"; //Valid	byte a = 123; //Valid
String c = 'A'; //Invalid, wrong quotation	int a = 12.3; //Invalid
char d = 'C'; //Valid	float a = 12.3; //Invalid, f missing
char e = "C"; //Invalid, wrong quotation	double a = 12.3f; //Valid, datatype converted
char f = "ABC"; //Invalid, more than 1 character	float a = 12.3d; //Invalid, lossy conversion
boolean b = True; //Invalid, T should be smaller case	int [] a = {5.3, 1, true} //Invalid, all data must be of int type

~ Self-Study ~

Refer to the table above and experiment to understand why some of the lines were invalid. You shall learn more about lossy conversion in 2.7 Datatype Conversion (Type Casting).

2.7 DATATYPE CONVERSION BASICS

Type refers to the data type of a variable or entity and casting is the process of changing an entity's data type to another datatype. Moreover, type casting is the process of converting a value between two different data types. Only the data type can be changed by casting; the data itself cannot be changed.

In Java, there are two types of casting:

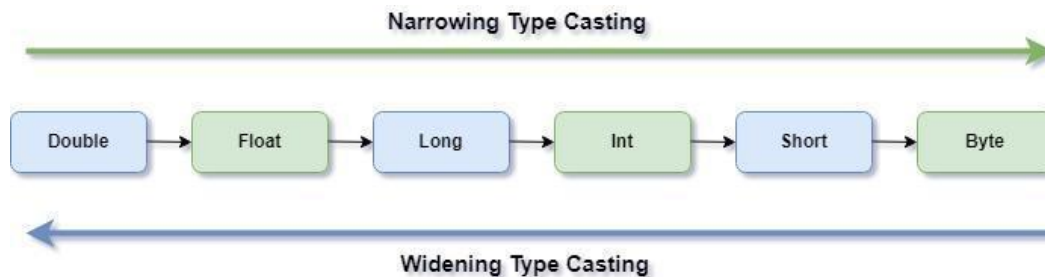
- Widening/Implicit Casting

- Narrowing/Explicit Casting

Widening/Implicit Casting:

Converting a lower data type to a higher data type (from right to left in the following diagram) is known as widening typecasting. As a result, there is no data loss. Since Java does this type of casting automatically and without any explicit coding, it is often referred to as automatic type casting. To implement widening type casting, we do not need to create any new code or modify any of the ones we already have. Example:

```
int num = 10;
double data = num;
```



Narrowing/Explicit Casting:

Narrowing type casting is the process of converting higher data types into lower data types (from left to right in the following diagram). As we convert larger size data types into smaller ones, data loss occurs. Because of this, the conversion has been made manual rather than automatic. This type conversion is also known as Explicit type conversion. Example:

```
double num = 10.99;
int data = (int)num;
```

Data Type Form/To	Byte	Char	Short	Int	Long	Float	Double
Byte	Auto Assignable	Cast Needed	Cast Needed	Cast Needed	Cast Needed	Cast Needed	Cast Needed
Char	Cast Needed	Auto Assignable	Cast Needed	Cast Needed	Cast Needed	Cast Needed	Cast Needed
Short	Auto Assignable	Cast Needed	Auto Assignable	Cast Needed	Cast Needed	Cast Needed	Cast Needed
Int	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Cast Needed	Cast Needed	Cast Needed
Long	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Cast Needed	Cast Needed
Float	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Cast Needed
Double	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable

Here, auto assignable represents the widening or implicit type casting and cast needed represents the narrowing or explicit type casting.

2.8 NUMERIC-TO-NUMERIC CONVERSION

Conversion	Type	Code	Explanation
int to float and double	Widening/Implicit	int num = 10; double data = num; //10.0 float data2 = num; //10.0	Assigning the int type variable named num to a double type variable named data. Here, Java first converts the int type data into the double type. And then assigns it to the double variable.
double and float to int	Narrowing / Explicit	double num = 10.99d; float num2 = 10.25f; int data = (int)num; //10	Assigning the double type variable named num to an int type variable named data. Here, the int

		<code>int data2= (int)num2 //10</code>	keyword inside the parentheses indicates that the num variable is converted into the int data type.
float to double	Widening/ Implicit	<code>float num = 127.45f; double d = (double) num; System.out.println(d); // 127.44999694824219</code>	Assigning the float type variable named num to a double type variable named d. Here, Java first converts the float type data into the double type. And then assigns it to the double variable.
double to float	Narrowing / Explicit	<code>double num=1.123456789d; float f= (float)num; System.out.println(f); // 1.1234568</code>	Assigning the double type variable named num to a float type variable named f. Here, the float keyword inside the parentheses indicates that the num variable is converted into the float data type.
int to byte	Narrowing / Explicit	<code>int num = 127; byte b = (byte) num; System.out.println(b); //127</code>	This will only work for int values between -128 to 127. Otherwise, it will give an error because byte can only store numbers from -128 to 127.

2.9 NUMERIC-TO-NON-NUMERIC AND VISE-VERSA CONVERSION

Conversion	Code	Explanation
integer to String	<code>String x= String.valueOf(15); String y= Integer.toString(20); System.out.println(x+y); //1520</code>	String.valueOf() and Integer.toString() both converts the int to a String and returns it. Therefore, the output is "15"+"20"="1520".
String to Integer	<code>int n = Integer.parseInt("25"); int m = Integer.valueOf("35"); System.out.println(m+n); //60</code>	Integer.parseInt() method returns the string as a primitive type int. If the string does not contain a valid integer, then it will throw a <code>NumberFormatException</code> . Integer.valueOf() method returns the string as an integer object. If the string does not contain a valid integer, then it will also throw a <code>NumberFormatException</code> . Therefore, the output is 25+35= 60.
int to char	<code>int a=65; char c=(char)a; System.out.println(c); //A</code>	The ASCII character of the given value is stored as the character.
char to int	<code>char a='Z'; int c=(int)a; System.out.println(c); //90</code>	The ASCII value of the given character is stored as the integer.

You will learn how to convert char to String and vise-versa later on in this course in Chapter 9: Strings. You are also not familiar with how methods work. For now, just keep in mind that methods may take a certain value, perform certain operations, and may return a different value.

~ Self-Study ~

Experiment on every possible kind of datatype conversion. Look up lossy conversion on the internet.

2.10 WORKSHEET

A. Trace the following code and write the outputs.

Code	Output
<pre> class B { public static void main(String[] args) { int a= 15; //Value of a is 5 int b= 4; //Value of b is 7 System.out.println(a+b); float a1= a; float b1= b; String a2= Integer.toString(a); String b2= Integer.toString(b); System.out.println(a1+b1); System.out.println(a1 == a); System.out.println(a2+b2); System.out.println(a2+"Hi"+b2); System.out.println("Hi"+Integer.toString(b+1)); } } </pre>	

B. Imagine you have the following lines of codes. Your task is to complete the code so that the desired output is generated.

Code	Output
<pre> class A { public static void main(String[] args) { int x= 5; //Value of x is 5 int y= 7; //Value of y is 7 System.out.println(x+y); //Your code here System.out.println(x1+y1); System.out.println(x2+y2); } } </pre>	12 12.0 57

C. Which of the following are invalid variable names and why?

N1, num2, 2num, num_1, firstName, first1name, hi 1, first_1_name, first-name, import, long, import1

Invalid Variable names	Reason for Being Invalid

- D. What is the difference between **declaring**, **assigning** and **initializing** a variable? Can we declare, assign or initialize the same variable multiple times in Java?
- E. Explain lossy conversion with code examples.



Inspiring Excellence

CHAPTER 3: OPERATORS

4.1 WHAT ARE OPERATORS?

Operators are symbols that are used to perform certain operations and changes to one or more operands. For example, 5+6 here 5 and 6 are operands and “+” is an operator. Operators in Java can be categorized as follows:

- Arithmetic
- Assignment
- Relational
- Logical
- Bitwise
- Unary and more

We shall only be focusing on the above six types of operators in this course.

4.2 ARITHMETIC OPERATORS

Used to perform arithmetic operations between two operands.

Category	Name	Symbol	Operation	Example
Additive	Addition	+	Adds two numbers	5+6 is 11
	Subtraction	-	Subtracts the second number from the first	5-6 is -1
Multiplicative	Multiplication	*	Multiplies the first number with the second	5*6 is 30
	Division	/	Divides the first number with the second	12/5 is 2
	Modulus	%	Finds the remainder after dividing the first number with the second	12%6 is 0

To perform any of the arithmetic operations both operands must be of the same type. Which means, the operands must be both int, both float or both double.

While dividing an int with another int, the whole part is taken. On the other hand, while dividing a float/double with a float/double, the decimal part is also taken into account. Therefore, float/double datatypes provide more accurate results in some arithmetic operations. For example, 5/2 is 2 whereas 5.0/2.0 is 2.5.

4.3 ASSIGNMENT OPERATORS

Used to assign a changed or unchanged value from the operand on its right to the operand on its left.

Name	Symbol	Identity	Operation
Assignment	=	x= 5;	Assigns the value of the operand on its right to its left
Addition Equals	+=	x+= 5; x= x+5;	Adds both of the operands on its right and left and assigns the addition to the operand on its left
Subtraction Equals	-=	x-= 2; x= x-2;	Subtracts the operand on its right from the operand on its left and assigns the subtraction to the operand on its left
Multiplication Equals	*=	x*= 2; x= x*2;	Multiplies both of the operands and assigns the multiplication to the operand on its left
Division Equals	/=	x/= 2; x= x/2;	Divides the operand on its left with the operand on its right and assigns the subtraction to the operand on its left

Modulus Equals	%=	x%= 2; x= x%2;	Assigns the remainder to the operator on its left after performing modulus operation on the operator on its left with the operator on its right.
-------------------	----	-------------------	--

4.4 RELATIONAL OPERATORS

Used to compare between two operands. Always results in either true or false.

Category	Name	Symbol	Examples
Comparison	Greater Than	>	System.out.println(5>3); //Output: true
	Less Than	<	System.out.println(5<3); //Output: false
	Greater Than or Equals	>=	System.out.println(5>=5); //Output: true
	Less Than or Equals	<=	System.out.println(5<=3); //Output: false
Equality	Equals	==	System.out.println(5==3); //Output: false
	Not Equals	!=	System.out.println(5!=3); //Output: true

4.5 LOGICAL OPERATORS

Used to determine the logic between two operands. Always results in either true or false.

Name	Symbol	Operation	Examples
Logical AND	&&	Denotes true if both operands before and after are true, otherwise denotes false.	System.out.println(5>3 && 2<3); //Output: true
Logical OR		Denotes true if at least one of the operands before or after is true, otherwise denotes false.	System.out.println(5<3 2!=2); //Output: false
Logical NOT	!	Reverses the logical value of the operand on its right.	System.out.println(!(2==3)); //Output: true

4.6 BITWISE OPERATORS

Bitwise operators are similar to logical operators but can be used on both int and Boolean datatypes (except bitwise complement). If the operands are int, the output is also int; if the operands are Boolean, the output is also Boolean.

Moreover, bitwise operator checks both the operands regardless of the value of the first operand. On the other hand, logical AND (&&) does not check the value of the second operand if the first operand is false. Logical OR (||) does not check the value of the second operand if the first operand is true. Bitwise Exclusive OR provides true if one of the operands is true and the other one is false, otherwise, it provides false. Bitwise Complement only works for Boolean and is also a unary operator as it needs only one operand.

Name	Symbol	Operation	Examples
Bitwise AND	&	Performs binary AND	System.out.println(8 & 7); //Output: 0 System.out.println(false & true); //Output: false
Bitwise Exclusive OR	^	Performs binary XOR	System.out.println(8 ^ 7); //Output: 15 System.out.println(true ^ false); //Output: true
Bitwise Inclusive OR		Performs binary OR	System.out.println(9 7); //Output: 15 System.out.println(false true); //Output: false
Bitwise Complement	~	Returns the two's complement representation of the input value	System.out.println(~5); //Output: -6

4.7 UNARY OPERATORS

Unary operators only need one operand. These types of operators are mainly used for counting purposes. There are two variations of unary operators:

- Prefix: The value is changed first and then assigned
- Postfix: the value is assigned first and then changed

Type	Name	Symbol	Operation	Example
Prefix	Negation	-	Negates an operand	x=5; y= -x; Here y is -5
	Not	!	Reverses the logical value of an operand	x= !true; Here x is false
	Increment	++	Increments the operand by 1	x= 5; y= ++x; Here x and y both are 6
	Decrement	--	Decrements the operand by 1	x= 5; y= --x; Here x and y both are 4
Postfix	Increment	++	Increments the operand by 1	x= 5; y= x++; Here x is 6 but y is 5
	Decrement	--	Decrements the operand by 1	x= 5; y= x--; Here x is 4 but y is 5

4.8 OPERATOR PRECEDENCE

The precedence is higher at the top and decreases as we move towards the bottom. If multiple operators with the same precedence are placed adjacently, operators are executed from left to right. If the first brackets “()” are present, we must go to the innermost first bracket and gradually move towards the outer ones.

Operator Type	Operator Category/Name	Symbol
Unary	Postfix	++, --
	Prefix	++, --, !, -
Arithmetic	Multiplicative	*, /, %
	Additive	+, -
Relational	Comparison	<, >, <=, >=
	Equality	==, !=
Bitwise	Bitwise AND	&
	Bitwise Exclusive OR	^
	Bitwise Inclusive OR	
Logical	Logical AND	&&
	Logical OR	
Assignment	All assignment operators	=, +=, -=, *=, /=, %=

Now, let us find the output of the following Java code lines:

```
System.out.println(5*(5+5/5%(5*5))); //Output: 30; Here 5*5 was executed first, then 5/5.
System.out.println(5+1==(5+5/5%(5*5))); //Output: true.
System.out.println(-(-5-1)==(5+5/5%(5*5))); //Output: true; Here -5-1 and 5*5 were executed first.
System.out.println(5!=5 && 5%1!=1 || 1==(5/2)); //Output: true.
```

From the above lines of codes, we can see that if operators with same precedence are placed adjacently such as: $5/5*5$, we must calculate from left to right. Which means performing the division first and then the multiplication.



Inspiring Excellence

4.9 WORKSHEET

A. Trace the following lines of codes and write the outputs.

Code	Output
class A {	
public static void main(String[] args) {	
int x= 5; //Value of x is 5	
int y= 7; //Value of y is 7	
System.out.println(x); //Print current value of x	
System.out.println(y); //Print current value of y	
x-=1;	
y+=1;	
System.out.println(x);	
System.out.println(y);	
x= ++y;	
y= x--;	
System.out.println(x);	
System.out.println(y);	
System.out.println(x==y);	
System.out.println(x%y==4*2);	
System.out.println(x&y);	
System.out.println(x y);	
x= -x;	
System.out.println(x);	
x= ~x;	
System.out.println(x);	
y*=x;	
System.out.println(y);	
System.out.println(x%2==1 && y%2!=0);	
}	
}	

Inspiring Excellence

B. Answer the following questions.

- ❖ What is the difference between bitwise operators and logical operators?
 - ❖ What is the output of the following lines of codes? Why do the values of x and y differ?
- ```

int x= 5;
int y= 7;
y= ++x;
x= y--;
System.out.println(x);
System.out.println(y)

```

## CHAPTER 4: INTRODUCTION TO JAVA CODING

### 5.1 ESSENTIALS

In order to start writing code in java, it is essential to understand the meaning and use of certain keywords that are the building blocks of any java code.

| Keywords      | Meaning                                                                                                         | Example Code                                                                                                                        |
|---------------|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| public        | An access modifier that allows classes, attributes, methods and constructors to be accessible by other classes. | <pre>public class Example1 {     public static void main(String args[]) {         System.out.println("Hello World.");     } }</pre> |
| class         | This keyword defines a class.                                                                                   |                                                                                                                                     |
| main          | This defines the main method.                                                                                   |                                                                                                                                     |
| args          | It represents arguments passed in the java command-line.                                                        |                                                                                                                                     |
| static        | It is used for memory management. It is used to share properties of a class with other classes.                 |                                                                                                                                     |
| void          | It is used during method declaration to indicate that a method does not return any type.                        |                                                                                                                                     |
| String[] args | It represents an array of strings that stores the arguments passed in the java command-line.                    |                                                                                                                                     |

### 5.2 SCOPES

In java, a block of code is the code inside the curly braces, {}. The scope of a variable represents the space or block of code in which the variable can be accessed and modified. Usually, the scope of a variable is the block of code in which it is created, for example- (1) A variable declared inside the curly braces of a loop is only valid (can be accessed and modified) within the scope of the loop, (2) A variable declared inside the curly braces of a method is only valid within the scope of the method.

| Example Code                                                                                                                                                                                                                                                                                                                                                                                                                            | Output                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| <pre>public class Example1 {     public static void main(String args[]) {         int x=10;         int y=25;         int z=x+y;         while (x&gt;5){             int m=3; //m initialized inside loop.             int n=m+x; //n initialized inside loop.             x--;         }          System.out.println("Sum of x+y = " + z);         System.out.println(m+" "+n); //m &amp; n not accessible outside loop.     } }</pre> | error: cannot find symbol      |
| <pre>public class Example2 {     public static void main(String args[]) {         int x=10;         int y=25;</pre>                                                                                                                                                                                                                                                                                                                     | <pre>3 13 3 12 3 11 3 10</pre> |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| <pre> int z=x+y; while (x&gt;5){     int m=3; //m initialized inside loop.     int n=m+x; //n initialized inside loop.     System.out.println(m+" "+n); //m &amp; n accessible only inside loop.     x--; }  System.out.println("Sum of x+y = " + z); } </pre>                                                                                                                                                                                                                                        | <pre> 3 9 Sum of x+y = 35 </pre>                     |
| <pre> public class Example3 {     public static void main(String args[]) {         int x=10;         int y=25;         int z=x+y;         while (x&gt;5){             int m=3; //m initialized inside loop.             int n=m+x; //n initialized inside loop.             x--;         }          System.out.println("Sum of x+y = " + z);         m+=5; //m cannot be modified outside loop.         System.out.println(m+" "+n); //m &amp; n not accessible outside loop.     } } </pre>          | <pre> error: cannot find symbol </pre>               |
| <pre> public class Example4 {     public static void main(String args[]) {         int x=10;         int y=25;         int z=x+y;         while (x&gt;5){             int m=3; //m initialized inside loop.             int n=m+x; //n initialized inside loop.             m+=5; //m can only be modified inside loop.             System.out.println(m+" "+n); //m &amp; n accessible only inside loop.             x--;         }          System.out.println("Sum of x+y = " + z);     } } </pre> | <pre> 8 13 8 12 8 11 8 10 8 9 Sum of x+y = 35 </pre> |

### 5.3 PRINTING

In order to show an output of a program, we use the print statement, **System.out.println("Text to be printed.")**. The **System.out** command basically invokes the System class to generate an output. The **println()** is a built-in public method used to show an output. It prints the text inside the method parameter, as well as a new line. To avoid adding the new line at the end, **print()** can be used instead as **System.out.print()**.

| Example Code                                                                                                                                                                                   | Output                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| <pre> public class Example1 {     public static void main(String args[]) {         int x=10;         while (x&gt;5){             System.out.println(x); //New line is added at the end. </pre> | <pre> 10 9 8 7 6 </pre> |

|                                                                                                                                                                                                                                    |        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| <pre>         x--;     } } } </pre>                                                                                                                                                                                                |        |
| <pre> public class Example1 {     public static void main(String args[]) {         int x=10;         while (x&gt;5){             System.out.print(x); //No new line is added at the end.             x--;         }     } } </pre> | 109876 |

## 5.4 COMMENTS

In a program, comments are lines which are not executed but are there to help someone who is not familiar with the code to understand how the code works. The compiler ignores these lines while compiling. Writing comments can also help to debug code easily.

In java, comments are 3 types:

1. **Single-line comments:** These are comments written in a single line and are usually short. The comment is initiated using two forward slashes, //. Anything written after the second slash in the same line will be considered as a comment.
2. **Multi-line comments:** These are comments written when descriptions need to be written for methods or loops in multiple lines. Writing multi-line comments using // is possible, but it becomes tedious when there are several lines, and // needs to be written before each line. In order to make writing multi-line comments easier, use a forward slash, /, followed by an asterisk, \*, then write the comment, and end using an asterisk, and a forward slash.
3. **Documentation comments:** It is often referred to as doc comment. It is usually used when writing code for a software or package to generate a documentation page for reference. It follows the same convention as a multi-line comment, only an exception of an additional asterisk at the beginning of every new line.

| Comment type  | Example                                                                                    |
|---------------|--------------------------------------------------------------------------------------------|
| Single line   | //This is a single-line comment.                                                           |
| Multi-line    | /*Multi-line comment starts<br>This is a multi-line comment.<br>Multi-line comment ends.*/ |
| Documentation | /** Doc comment starts<br>*continue<br>*add tags<br>*doc comment ends*/                    |



## 5.5 WORKSHEET

- A. Take two integer numbers from the user and print the summation and average.
- B. Take the values of x, y and z and solve the result of the following equation:  

$$\text{result} = x + (x \% 2 + y * z^{(4/2)})$$
- C. Take the first name and last name from the user and print the outputs accordingly.

|                                     |                                    |
|-------------------------------------|------------------------------------|
| Sample Input #1:<br>Samiha<br>Haque | Output:<br>Full Name: Samiha Haque |
| Sample Input #2:<br>Sabbir<br>Ahmed | Output:<br>Full Name: Sabbir Ahmed |

- D. Take the first name, middle name and last name from the user and print the outputs accordingly. Keep in mind that a user may or may not have a middle name. If the user does not have one, they must enter an empty string as the middle name.

|                                                |                                            |
|------------------------------------------------|--------------------------------------------|
| Sample Input #1:<br>Aiman<br>Jabeen<br>Ramisa  | Output:<br>Full Name: Aiman Jabeen Ramisa  |
| Sample Input #2:<br>Dibyo<br>Fabian<br>Dofadar | Output:<br>Full Name: Dibyo Fabian Dofadar |
| Sample Input #3:<br>Sifat<br><br>Tanvir        | Output:<br>Sifat Tanvir                    |

## CHAPTER 5: USER INPUT

### 3.1 CREATING A SCANNER

The Scanner is a class which is used to get user input, and it is found in the java.util package. Whenever, we shall take inputs from the user, we have to import this package before starting the class. This is how we import the Scanner class:

```
import java.util.Scanner; // Import the Scanner class
```

Now, inside the class we have to create an object of the Scanner class, so that we can use all its properties. You will not understand the process of creating an object or its purposes now, as it is a concept of Object-Oriented Programming. For now, just assume that the following line is necessary for taking user input. Using the variable sc we can now take different types of user inputs. The **System**

```
Scanner sc = new Scanner(System.in); // Creating a Scanner
```

### 3.2 INPUTTING DIFFERENT DATATYPES

After taking user inputs, we have to assign those inputs in variables. While declaring variables, we have to specify its datatype. Therefore, while reading user inputs, we must also specify what kind of input we are reading. The reading method will be different for each kind of input.

| Method        | Datatype |
|---------------|----------|
| nextBoolean() | Boolean  |
| nextByte()    | Byte     |
| nextShort()   | Short    |
| nextLong()    | Long     |
| nextInt()     | Integer  |
| nextFloat()   | Float    |
| nextDouble()  | Double   |
| nextLine()    | String   |

### 3.3 CODE EXAMPLES

| Questions                                       | Codes                                                                                                                                                                                                                   | Input(s) | Output(s) |
|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-----------|
| Take a float number from the user and print it. | <pre>import java.util.Scanner; class Main {     public static void main(String[] args) {         Scanner sc = new Scanner(System.in);         float num= sc.nextFloat();         System.out.println(num);     } }</pre> | 5.5      | 5.5       |

|                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                       |                                                                              |
|---------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|------------------------------------------------------------------------------|
| Take the name (String), the age (int) and the salary (double) from the user and print these.                        | <pre>import java.util.Scanner;  class Main {     public static void main(String[] args) {         Scanner sc = new Scanner(System.in);          System.out.println("Enter name, age and salary:");         String name = sc.nextLine();         int age = sc.nextInt();         double salary = sc.nextDouble();          System.out.println("Name: " + name);         System.out.println("Age: " + age);         System.out.println("Salary: " + salary);     } }</pre> | Sami<br>23<br>1200.5  | Enter name,<br>age and<br>salary:<br>Name: Sami<br>Age: 23<br>Salary: 1200.5 |
| Take a number from the user and see if it is divisible by both 3 and 7. If yes, print "Yes", otherwise, print "No". | <pre>import java.util.Scanner; class Main {     public static void main(String[] args) {         Scanner sc = new Scanner(System.in);         int num= sc.nextInt(); //integer input         if (num%3==0 &amp;&amp; num%7==0){             System.out.println("Yes");         }else{             System.out.println("No");         }     } }</pre>                                                                                                                      | 21                    | Yes                                                                          |
|                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 18                    | No                                                                           |
| Take two Strings from the user, merge them and print the merged strings.                                            | <pre>import java.util.Scanner; class NumberChecker {     public static void main(String[] args) {         Scanner sc = new Scanner(System.in);         String s1= sc.nextLine();         String s2= sc.nextLine();         System.out.println(s1+s2);     } }</pre>                                                                                                                                                                                                      | Hello<br>Darknes<br>s | HelloDarknes<br>s                                                            |
|                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 5<br>7                | 57                                                                           |

### 3.4 WORKSHEET

- A. Write a program that takes input of the user's last name. Then prints it back as shown in the examples below.

| Sample Input | Sample Output                    |
|--------------|----------------------------------|
| John         | This user's last name is John.   |
| Rahman       | This user's last name is Rahman. |

- B. Write a Java program that reads two integers from the user and prints the sum, difference, product and the average.

| Sample Input | Sample Output                                               |
|--------------|-------------------------------------------------------------|
| 10<br>12     | Sum: 22<br>Difference: 2<br>Product: 120.0<br>Average: 11.0 |
| 5<br>-2      | Sum: 3<br>Difference: 7<br>Product: -10.0<br>Average: 1.5   |

- C. Write a Java program that reads the radius of a circle and prints the area and circumference of the circle as shown in the examples below.

| Sample Input | Sample Output                        |
|--------------|--------------------------------------|
| 5            | Area: 78.54<br>Circumference: 31.42  |
| 12           | Area: 452.38<br>Circumference: 75.39 |

- D. Write a program that takes a floating number as an input and prints the square of the number.

| Sample Input | Sample Output |
|--------------|---------------|
| 5.0          | 25.0          |
| -4           | 16.0          |

- E. Write a program that reads two strings from the user and then prints their sum.

| Sample Input | Sample Output |
|--------------|---------------|
| "32"<br>"10" | Sum: 42       |
| "-2"<br>"12" | Sum: 10       |

## CHAPTER 6: UNDERSTANDING ERRORS

### 6.1 PREAMBLE

By this point you should have some Java coding experience and must have come across a lot of errors. Some are easy to figure out, some are difficult. In this chapter we shall learn about some common errors and how to prevent these. After completing this chapter, you will no longer have to spend hours trying to figure out what error occurred. Errors in Java can be categorized as follows:

- Syntax Errors
- Logical Errors
- Runtime Errors or Exceptions

### 6.2 SYNTAX ERRORS

Syntax error occurs while we make mistakes writing syntaxes. It is usually caused by the most trivial mistakes or lack of attention. These are the easiest to spot and Java compilers find these for you including the line numbers. This is why they are also known as compile-time errors. The program cannot generate a ".class" file. Some common syntax errors are shown below:

| Reason                           | Example                                                                                             | Error Shown                                    |
|----------------------------------|-----------------------------------------------------------------------------------------------------|------------------------------------------------|
| Semicolon or brackets missing    | <code>System.out.println("Hi")</code>                                                               | <code>;' expected</code>                       |
| Not closing first brackets       | <code>System.out.println("Hi!");</code>                                                             | <code>)' expected</code>                       |
| Not closing curly braces         | <code>class HelloWorld {<br/>    public static void main(String[] args) {<br/>        x=1; }</code> | <code>reached end of file while parsing</code> |
| Misspelled keywords              | <code>public <b>staic</b> void main(String[] args) {</code>                                         | <code>&lt;identifier&gt; expected</code>       |
| Wrong-Cased keywords             | <code><b>Public</b> static void main(String[] args) {</code>                                        | <code>&lt;identifier&gt; expected</code>       |
| Improper variable names          | <code><b>public</b>= x; //Reserved keyword</code>                                                   | <code>illegal start of expression</code>       |
|                                  | <code><b>lname</b>= 10; //variable name starts with a digit</code>                                  | <code>not a statement</code>                   |
| Missing double quotes in Strings | <code>System.out.println("Hi);</code>                                                               | <code>unclosed string literal</code>           |
| Using undeclared variables       | <code>print(x); //x not declared</code>                                                             | <code>cannot find symbol</code>                |
| Missing or excessive operators   | <code>y = 3 + * 5; //excessive operators</code>                                                     | <code>illegal start of expression</code>       |

### 6.3 LOGICAL ERRORS

Logical errors occur when the syntaxes are correct, successfully compiles, detects no errors during runtime but the code generates wrong output. Despite logical errors, your code will run, the program will generate a ".class" file, but the output will be wrong.

| Task                                             | Code                                                    | Problem                                                                                                           |
|--------------------------------------------------|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Add two integer numbers and print the summation. | <code>x=1;<br/>y=5;<br/>System.out.println(x*y);</code> | The output was supposed to be 6, but we got 5. Here the code is syntactically correct but contains logical error. |

There is no hard and fast rule for preventing logical errors. You have to crosscheck all the logics if the output does not match.

## 6.4 RUNTIME ERRORS OR EXCEPTIONS

Runtime errors occur when the program successfully compiles without any error and creates a ".class" file. However, the program does not execute properly. These errors are detected at runtime or at the time of execution of the program instead of compile time. These runtime errors are called exceptions and they terminate the program abnormally, giving an error statement. A runtime error can happen when:

| Reason           | Example                                                                                                                           | Error Shown                                                          |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| Division by Zero | <code>System.out.println(5/0);</code>                                                                                             | <code>java.lang.ArithmeticException: / by zero</code>                |
| Input Mismatch   | <code>int x = myObj.nextInt();</code> //Here an int is expected but if you enter a String instead, this will lead to an exception | <code>java.util.InputMismatchException</code>                        |
| Null Pointer     | <code>String s = null;</code><br><code>System.out.println(s.length());</code><br>// null has no length                            | <code>java.lang.NullPointerException</code>                          |
| Wrong            | <code>int x = Integer.parseInt("Hi");</code><br>//Converting a string with improper format into a numeric value                   | <code>java.lang.NumberFormatException: For input string: "Hi"</code> |

We shall learn more about Exceptions and how to handle them later down the line.

## 6.5 WORKSHEET

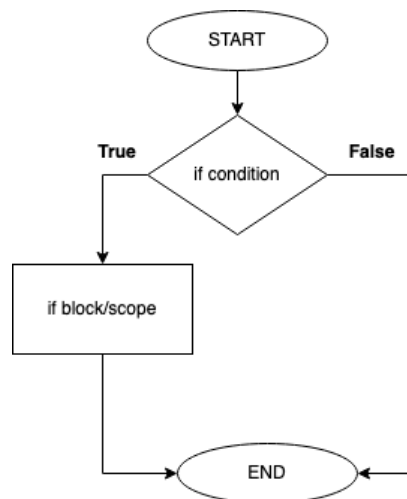
- A. Check the following code carefully and find all the possible errors and then rewrite it correctly. The output of this code should be:  
false  
15

| Given Code Lines                                         | Error Present or Not (If present specify which error) | Rewritten Code |
|----------------------------------------------------------|-------------------------------------------------------|----------------|
| <code>class A {</code>                                   |                                                       |                |
| <code>    public static void man(String[] args) {</code> |                                                       |                |
| <code>        int x= 5/0;</code>                         |                                                       |                |
| <code>        int y= 7</code>                            |                                                       |                |
| <code>        x-=1;</code>                               |                                                       |                |
| <code>        1y+=1;</code>                              |                                                       |                |
| <code>        System.out.println(y%2!=0;</code>          |                                                       |                |
| <code>        int m = Integer.valueOf("Hi");</code>      |                                                       |                |
| <code>        System.out.println(m);</code>              |                                                       |                |
| <code>    }</code>                                       |                                                       |                |

## CHAPTER 7: DECISION MAKING

### 7.1 “If” SINGLE SELECTION STATEMENT

We are now going to look at conditional statements or branching. Sometimes we need to produce outputs based on conditions, for example, if it rains today then I will take an umbrella. In this example, taking an umbrella is dependent on the condition: rain. A simple flowchart is given below:



Here, we can see that an “if” block is executed only when the if condition holds true.

A sample code structure is given below:

```

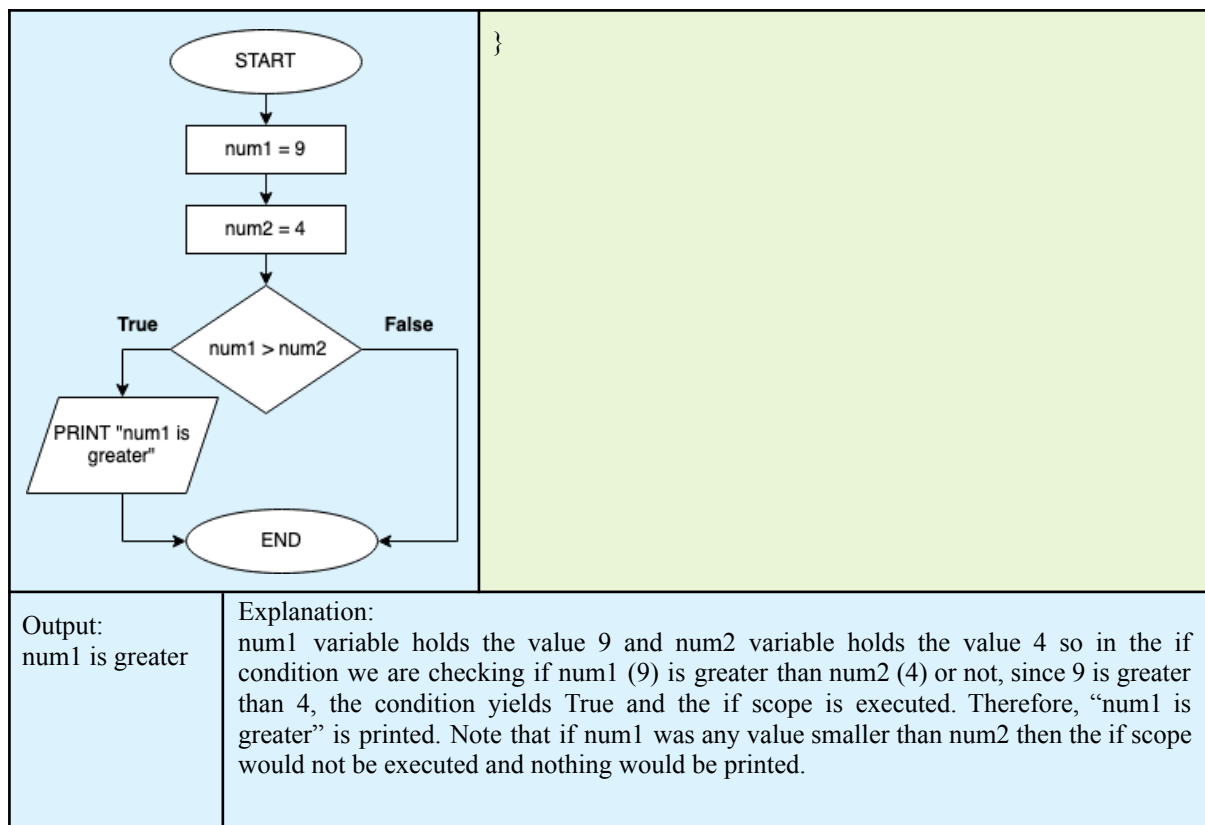
public class Sample1 {
 public static void main(String [] args){
 if (condition) {
 True
 block
 }
 }
}

```

Here, we can see that the keyword “if” is followed by a “condition” and a pair of curly braces { }. In the next line, we can see leading whitespaces followed by a block of code. A block/scope of code is a collection of lines of codes. For example, the “True block” inside the curly braces may contain multiple lines of codes which will only be executed when the “if condition” is True. The leading whitespaces are called *indentation* which separates a block/scope of code from the rest of the lines of code. Maintaining indentation is crucial in many programming languages.

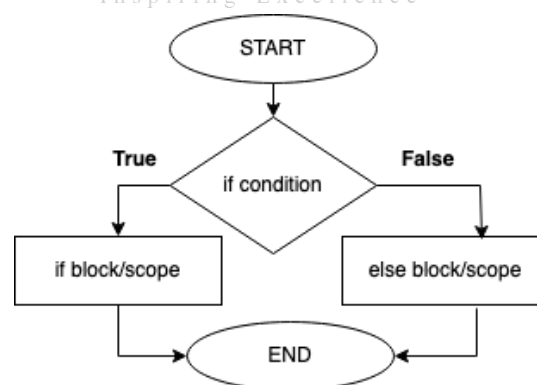
Let’s see another example. Here we want to print “num1 is greater” is the first number is greater.

| Flowchart | Code                                                                                                                                                                                                                                     |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           | <pre> public class Sample2 {     public static void main(String [] args){         int num1 = 9;         int num2 = 4;         if (num1 &gt; num2){             System.out.println("num1 is greater");         }     } }           </pre> |



## 7.2 “If - Else” DOUBLE SELECTION STATEMENT

Sometimes we might need to execute a block of code when the if condition yields False. For example, if it rains today then I will stay home, else I will go out. In this example, there are two possibilities which depend on the condition: rain. If the condition is True i.e. if it rains then the outcome is staying home but if the condition is False then the outcome will be going out. A simple flowchart is given below:



Here, we can see that an “if” block is executed only when the if condition holds true and the “else” block is executed only when the if condition holds false.

A sample code structure is given below:



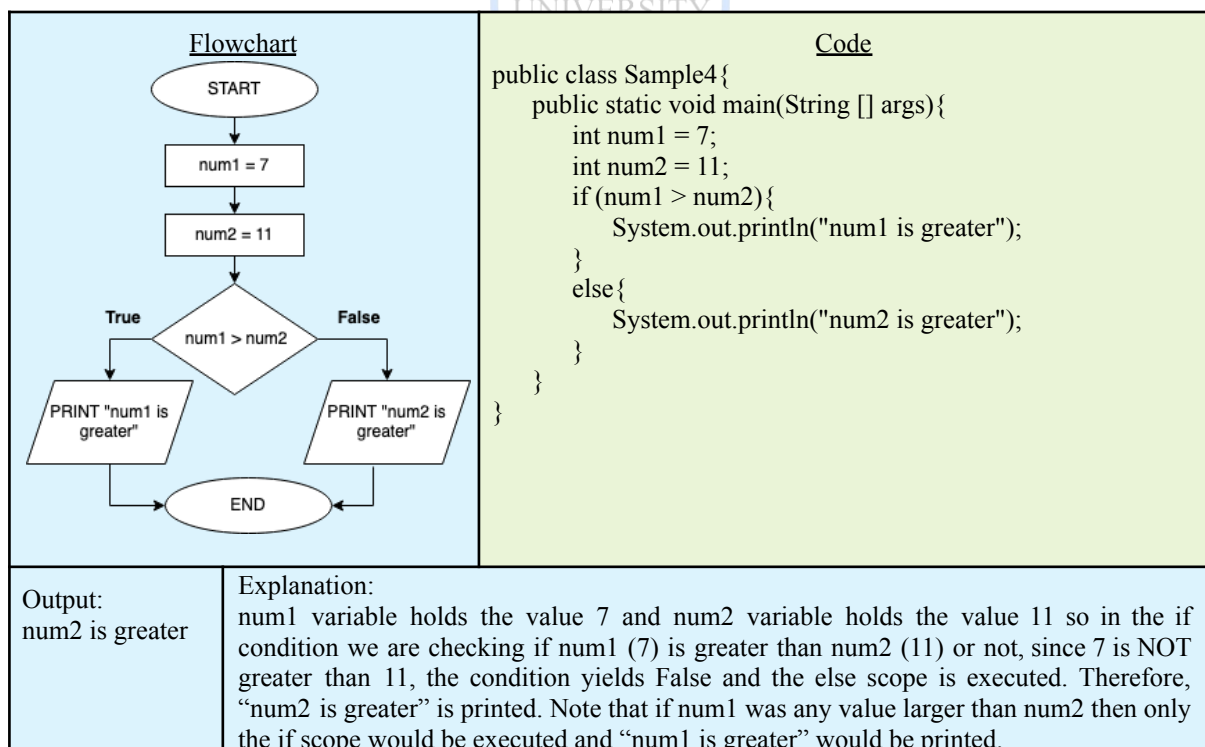
```

public class Sample3 {
 public static void main (String []
args){
 if (condition) {
 True block
 }
 else {
 False block
 }
}
}

```

Here, we can see the keyword “else” followed by a pair of curly braces { } which indicate the scope of the else block separated by an indentation. The else block of code is only executed when the if condition yields false. Notice that we do not need to write any conditional statement beside the “else” keyword.

Let’s see the previous example again. Now, we want to print “num1 is greater” if the first number is more than the second number or print “num2 is greater” otherwise.



We can use Logical Operators within if conditions as well whenever there are multiple conditions that need to be checked at a time.

Let’s see an example where we need to take an integer input from the user and determine if the number is a positive even number or not.

| Flowchart                                                                                                                                                                                                                                                                                                                                                                 |                                       |                                                                                                                                                                                                                                            | Code                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> graph TD     Start([START]) --&gt; Prompt[/PROMPT "Enter a number"/]     Prompt --&gt; Read[/READ num/]     Read --&gt; Decision{num &gt;= 0 &amp;&amp; num % 2 == 0}     Decision -- True --&gt; PrintTrue[/PRINT "positive even number"/]     Decision -- False --&gt; PrintFalse[/PRINT "Not"/]     PrintTrue --&gt; End([END])     PrintFalse --&gt; End </pre> |                                       |                                                                                                                                                                                                                                            | <pre> import java.util.Scanner; public class Sample5 {     public static void main(String [] args){         Scanner sc = new Scanner(System.in);         int num= sc.nextInt();         if (num&gt;=0 &amp;&amp; num%2==0){             System.out.println("positive even number");         }         else{             System.out.println("not");         }     } } </pre> |
| Input:<br>18                                                                                                                                                                                                                                                                                                                                                              | Output:<br>positive<br>even<br>number | Explanation:<br>The user input is 18 and in the if condition it is checked 18>=0 which is True and 18%2==0 which is also True. Since True and True yields True so the condition is True overall. Hence, "positive even number" is printed. |                                                                                                                                                                                                                                                                                                                                                                             |

### 7.3 "IF - ELSE IF - ELSE" MULTIPLE SELECTION STATEMENT

Sometimes we might need to execute different blocks of code depending on multiple conditions. These types of problems can be solved using chained conditions i.e. (if... else if... else if... else). These conditions follow a top-down approach during execution. First, the if condition is checked and if it yields False, then the next else if condition is checked and if it yields False then the next else if condition is checked and so on. If none of the else if conditions are satisfied then the final else block would be automatically executed. One if block may be followed by multiple else if blocks and a final else block. Among these several conditions, only one condition block is executed when it yields True and so the rest of the bottom conditional statements in the chain will not be checked further. A simple flowchart and sample code structure is given below:

| Flowchart                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |  |  | Code                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> graph TD     Start([START]) --&gt; Cond1{if condition}     Cond1 -- True --&gt; Block1[if block/scope]     Cond1 -- False --&gt; Cond2{else if condition}     Cond2 -- True --&gt; Block2[else if block/scope]     Cond2 -- False --&gt; Cond3{else if condition}     Cond3 -- True --&gt; Block3[else if block/scope]     Cond3 -- False --&gt; Block4[else block/scope]     Block1 --&gt; Merge(( ))     Block2 --&gt; Merge     Block3 --&gt; Merge     Block4 --&gt; Merge     Merge --&gt; End([End]) </pre> |  |  | <pre> public class Sample6 {     public static void main(String [] args){         if (condition) {             if True block         }         else if (condition) {             else if True block         }         else if (condition) {             else if True block         }         else {             else block         }     } } </pre> |

|  |   |
|--|---|
|  | } |
|  | } |

In the sample code it can be seen that conditional statements are placed after “else if” keywords. The else block will only be executed if none of the above if or else if conditions yield True.

Remember that, a single “if” block can exist on its own without the help of any “else if” block or “else” block. Also, an “if” block followed by one or multiple “else if” blocks can also exist without the help of any “else” block. But, only “else” blocks or “else if” blocks cannot stand along without any initial “if” block.

Let’s see an example. We will take the salary and work experience of an employee as an input and calculate the bonus based on their experience. If the work experience is over 8 years, then 15% bonus is awarded, if the work experience is 5 years - 8 years then 10% bonus is awarded, if the work experience is 2 years - 4 years then 5% bonus is awarded and work experience below 2 years will be awarded with 2.5% bonus.

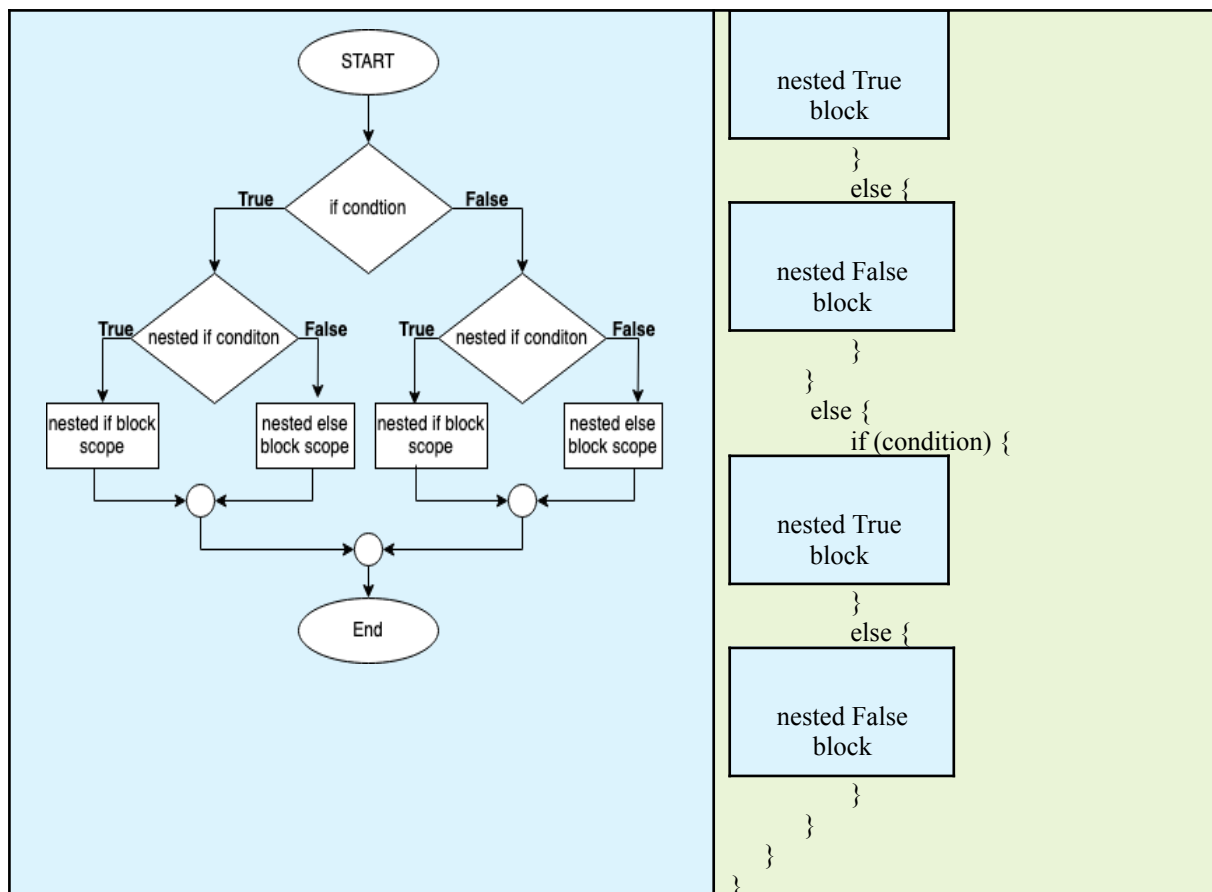
| Flowchart                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Code                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> graph TD     Start([START]) --&gt; PromptSalary[/PROMPT "enter salary"/]     PromptSalary --&gt; ReadSalary[/READ salary/]     ReadSalary --&gt; PromptExp[/PROMPT "enter experience"/]     PromptExp --&gt; ReadExp[/READ experience/]     ReadExp --&gt; Exp8{experience &gt; 8}     Exp8 -- True --&gt; Print15[/PRINT salary*0.15/]     Exp8 -- False --&gt; Exp5{experience &gt;= 5}     Exp5 -- True --&gt; Print10[/PRINT salary*0.1/]     Exp5 -- False --&gt; Exp2{experience &gt;= 2}     Exp2 -- True --&gt; Print5[/PRINT salary*0.05/]     Exp2 -- False --&gt; Print25[/PRINT salary*0.025/]     Print15 --&gt; Join(( ))     Print10 --&gt; Join     Print5 --&gt; Join     Print25 --&gt; Join     Join --&gt; End([End]) </pre> | <pre> import java.util.Scanner; public class Sample7 {     public static void main(String [] args) {         Scanner sc = new Scanner(System.in);         int salary = sc.nextInt();         int experience = sc.nextInt();         if(experience &gt; 8){             System.out.println(salary*0.15);         }         else if(experience &gt;= 5){             System.out.println(salary*0.1);         }         else if(experience &gt;= 2){             System.out.println(salary*0.05);         }         else{             System.out.println(salary*0.025);         }     } } </pre> |
| <p>Sample Input:<br/>30000<br/>3</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | <p>Output:<br/>1500.0</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <p><b>Explanation:</b> The input salary is 30000 and the experience is 3 years. So according to the code, the 2nd else if is executed only since experience is greater than or equal to 2 years and hence the output is <math>30000 \times 0.05 = 1500.0</math></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## 7.4 NESTED DECISION MAKING

Multiple if...else if...else blocks can be put inside an if block and/or an else if block and/or an else block. These blocks would be called nested blocks. The inner code blocks would only be executed if the outer conditional block yields True. In these programs, indentation is crucial to distinguish between nested code blocks.

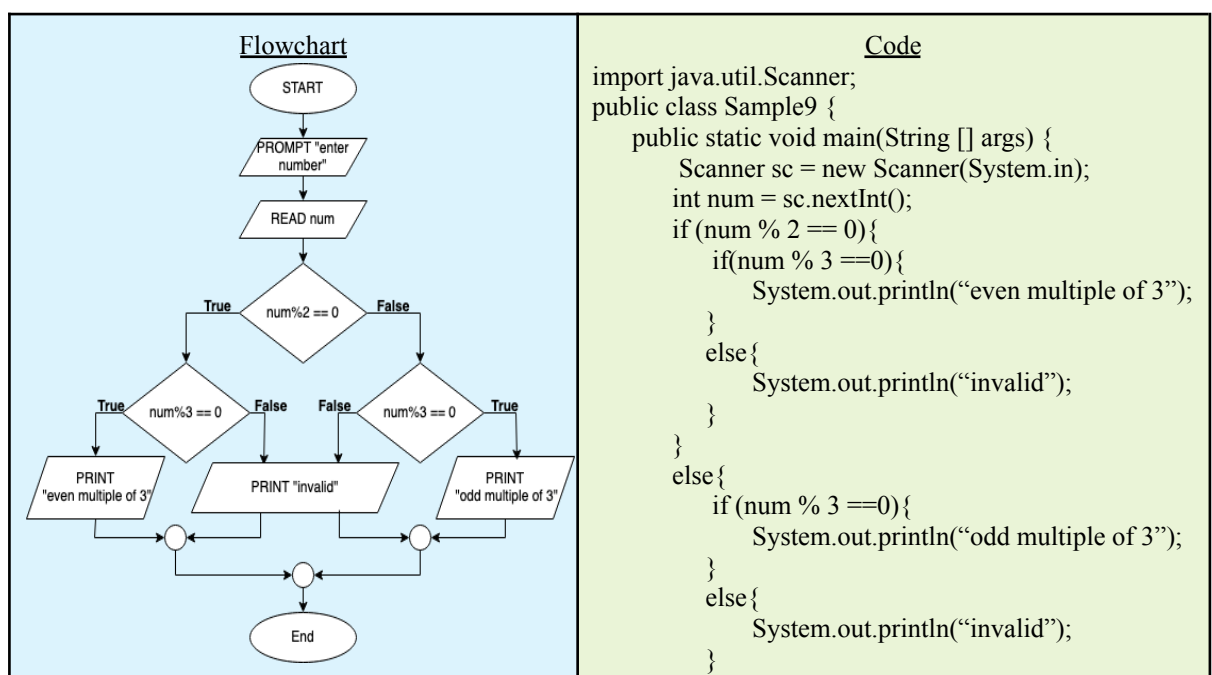
A simple flowchart and sample code structure is given below:

| Flowchart | Code                                                                                                                                    |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------|
|           | <pre> public class Sample8 {     public static void main (String [] args){         if (condition) {             if (condition) { </pre> |



In the sample code, it can be seen that the nested if/else blocks follow their own indentation. Remember, the nested blocks also follow the same convention as any basic if...else if...else block that we learned in the above sections.

Let's see an example. We want to print "even multiple of 3" if a number is an even number and a multiple of 3, "odd multiple of 3" if the number is an odd number and a multiple of 3 and "invalid" otherwise. [Note: this problem can also be solved using "and" logical operator and "if...else if...else" blocks. Try it out on your own.]



|  |                                                                                                                                                                                                                            |                              |
|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
|  | <pre>     }   } }</pre>                                                                                                                                                                                                    |                              |
|  | Input:<br>33                                                                                                                                                                                                               | Output:<br>odd multiple of 3 |
|  | Explanation:<br>The input is 33 for which $33\%2$ is not equal to 0 so it goes to the else block and then inside the else block 33 is again checked for which $33\%3$ is equal to 0 and so “odd multiple of 3” is printed. |                              |

## 7.5 SWITCH CASE CONDITIONING

Another way of handling the decision making process is called **switch statement** or **switch case conditioning**. It can be used as an alternative to writing many *if ... else if ... else* statements. Here, the syntax is more clear and easy to write & understand.

|                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b><br><pre> switch(expression){   case value1 :     // Code Statements     break; // optional   case value2 :     // Code Statements     break; // optional   .   .   .   case valueN :     // Code Statements     break; // optional   default: // optional     // Default Code Statements }</pre> | <ul style="list-style-type: none"> <li>• There can be one or multiple case values for a switch expression. The expression is evaluated once and compared with the values of each case.</li> <li>• The case values must be of switch expression type only; variables are not allowed. Also, the case values need to be unique. In case of duplicate values, it renders an error.</li> <li>• If there is a match, the associated block of code is executed. For example, if expression matches with value1, the code of case value1 is executed. Similarly, the code of case value2 is executed, if expression matches with value2 and so on.</li> <li>• The optional break statement is used inside the cases to terminate a statement sequence. If it is omitted, execution will continue on into the next case.</li> <li>• If there is no match, the code of the optional default case is executed. It can appear anywhere inside the switch block. In case, if it is not at the end, then a break statement must be kept after the default statement to omit the execution of the next case statement.</li> </ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

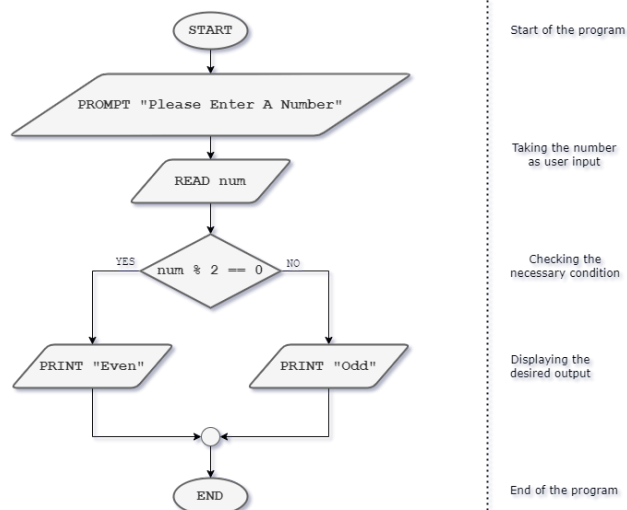
In summary, the switch statement evaluates an expression and compares it to case values of the same type. When a match is found, the associated block of code is executed. The break statement is used to exit the switch block, and the default statement is executed if there is no match.

Let's see an example.

## 7.6 FLOWCHARTS

Now, let's look at some relevant examples through flowcharts.

- ✓ **Scenario 1:** Let's design a flowchart of a program that will take a number as user input and determine whether the number is even or odd. *[Hint: If you try to divide any even number in this universe by 2, you will always end up with a remainder value 0 (zero).]*

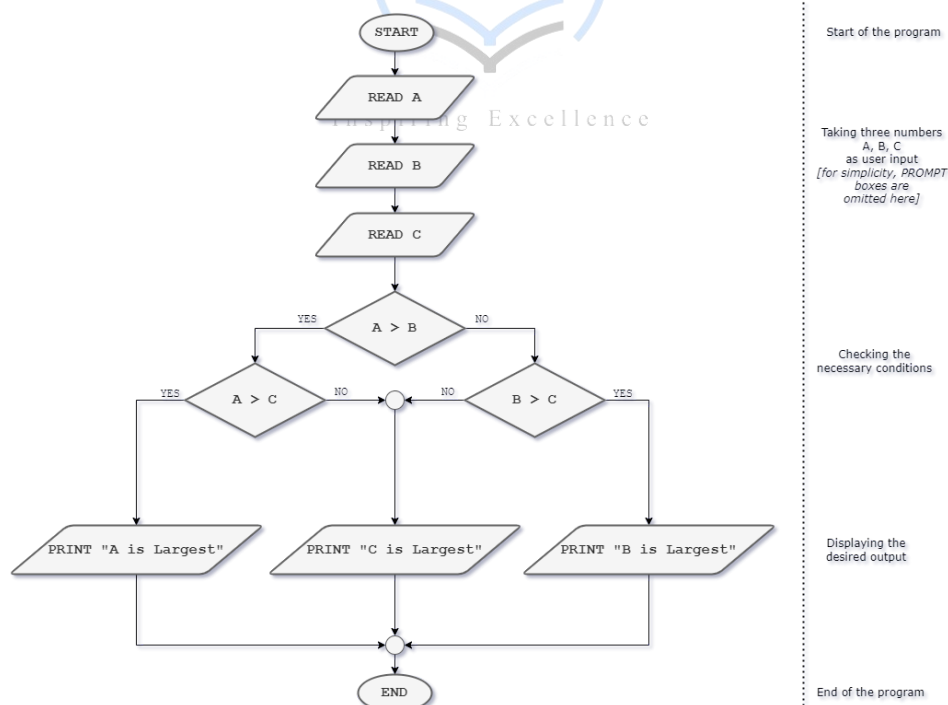


- ✓ **Scenario 2:** Let's consider three numbers A, B and C. We need to find out which number is the largest among these three and draw a flowchart for that. Let's assume, for this problem all the values of A, B & C are unique; that means there will be no duplicate values.

Before going to design the flowchart, let's break down the problem and formulate the necessary conditions first.

- If A is greater than B and A is greater than C, then definitely A will be the largest.
- If A is not greater than B (which means B is greater than A) and B is greater than C, then B will be the largest.
- If the above two conditions are not fulfilled, then obviously C will be the largest.

Now, let's look at the flowchart below.

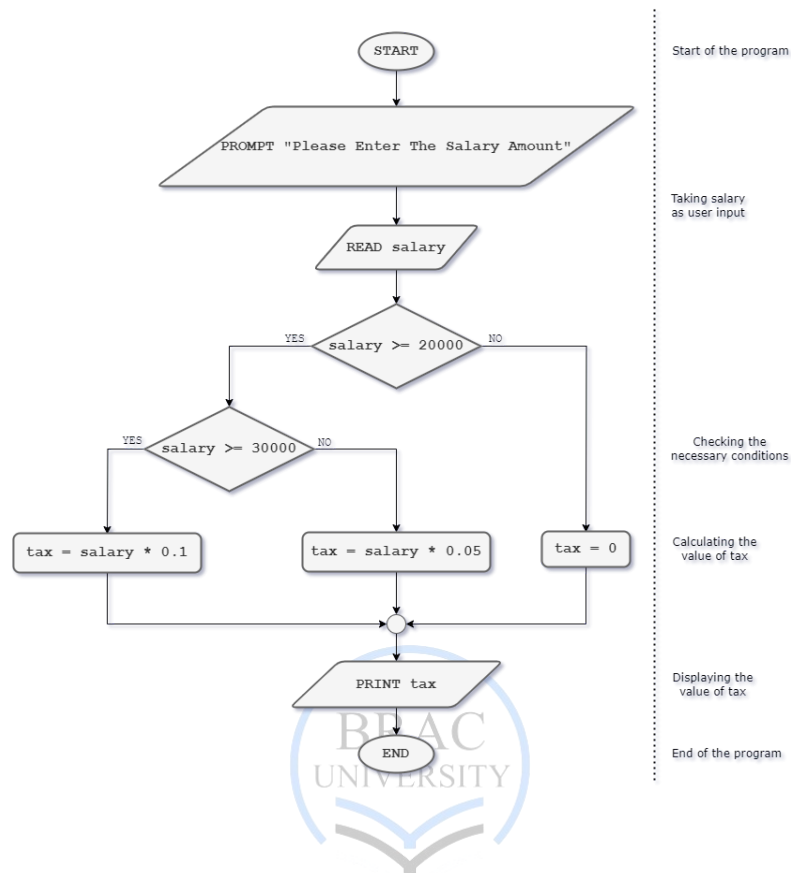


- ✓ **Scenario 3:** Let's calculate some taxes again. Suppose, you need to calculate the tax amount an employee needs to be paid based on his/her salary. Here are the requirements:

- If the salary of the employee is below 20,000 Taka, then no tax.
- If the salary is 30,000 Taka or above, then the tax will be 10% of the monthly salary.

- If the salary is 20,000 Taka or above and less than 30,000 Taka, then the tax will be 5% of the monthly salary.

Let's design the flowchart of this problem.



## 7.7 WORKSHEET

- A. Consider the following code. Suppose, the desired output should be: "A is greater than 100". What is the problem in this code? How can you overcome this problem? *[There can be multiple ways to solve the issues.]*

```
class SimpleCode {
 public static void main(String[] args) {
 int a = 120;
 if (a > 10) {
 System.out.println("A is greater than 10");
 }
 else if (a > 100){
 System.out.println("A is greater than 100");
 }
 else {
 System.out.println("A is less than 10");
 }
 }
}
```

- B. Suppose, you are dealing with 3 conditions. You are trying to implement these conditions using if → else if → else blocks. So, clearly the “else” block is missing. But the code will not give any errors. So, what do you think of omitting this “else” block? In which cases you can apply this technique?
- C. Consider the following incomplete code.
- At least how many lines of "Hello" will be printed? Explain briefly.
  - Maximum how many lines of "Hello" will be printed? Explain briefly.

```
if [condition]
 System.out.println("Hello");
if [condition]
 System.out.println("Hello");
else
 System.out.println("Hello");
if [condition]
 System.out.println("Hello");
else if [condition]
 System.out.println("Hello");
else
 System.out.println("Hello");
if [condition]
 System.out.println("Hello");
```

- D. Suppose, there are two boolean variables a and b. The value of a is set to true and the value of b is set to false. What will be the result of the following expression? Briefly explain.

```
!(a && false) && (a || !b) || (b && !a)
```

- E. Design the flowchart and also write the Java code for the following problems.
- Take a number from the user and find out if the number is positive or negative or zero.
  - Take a number as user input and calculate the absolute value of the number. *[For example, absolute value of 5 is 5 and absolute value of -5 is 5.]*
  - Find out the largest number among four numbers taken from the user.
  - Find out the second largest number / second smallest number among three numbers taken from the user. *[You are not allowed to use a loop.]*
- F. Suppose, on normal days, a worker usually works a maximum of 8 hours daily. On some special days, after 8 hours of work, overtime is counted. Suppose,
- For the first 8 hours, the worker gets 200 taka hourly.



- b. For the next 2 hours of overtime, the worker will receive 250 taka hourly.
- c. For another 2 hours of overtime, the worker will receive 300 taka hourly.

Your first task is to write a Java program that takes the daily work hour of that worker as input. Then calculate the amount the worker will receive based on the criteria mentioned above. If the daily work hour is zero or negative or greater than 12, then display "Invalid".

| Sample Input | Sample Output | Explanation                                               |
|--------------|---------------|-----------------------------------------------------------|
| 7            | 1400          | $7 \times 200 = 1400$                                     |
| 9            | 1850          | $(8 \times 200) + (1 \times 250) = 1850$                  |
| 12           | 2700          | $(8 \times 200) + (2 \times 250) + (2 \times 300) = 2700$ |
| -4           | Invalid       | Invalid case, because the input is negative.              |

- G. Let's assume, these are the grading criteria of a particular course.

| Marks Range | Grade | Comment      |
|-------------|-------|--------------|
| 80 - 100    | A     | Excellent    |
| 70 - 79     | B     | Good         |
| 60 - 69     | C     | Satisfactory |
| 50 - 59     | D     | Okay         |
| 0 - 50      | F     | Fail         |

Your task is to write a Java code where you need to take a student's obtained marks as user input. Then based on the above criteria, display the grade the student got and also display the corresponding comment from the table. If the user enters a number that is outside the valid range 0-100, then display the line "Invalid Input".

- H. Write a Java program that takes an integer number as user input and print "Yes" if the number is divisible by: *(For each subproblem, there will be separate codes)*

- a. either 3 or 5
- b. 3 and 5 both
- c. 3 but not 5
- d. 5 but not 3
- e. 3 or 5 but not both
- f. neither 3 nor 5

Print "No" otherwise.

## CHAPTER 8: REPETITIONS

### 8.1 INTRODUCING LOOPS: UNDERSTANDING WHEN TO USE REPETITION

Suppose, you are told to print the first 50 positive even numbers. It would not be smart to write the same print statement 50 times, right? Since we are doing the same task over and over, we can simply put it into a loop. Using loops will significantly make tasks less tedious, less time consuming and smaller for us. Here we reduced nearly 50 lines of code to just 4 lines using repetition which is also known as loops.

|                                                                                                               |                                                                                                      |
|---------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <pre>System.out.println(0); System.out.println(2); System.out.println(4); ..... System.out.println(98);</pre> | <pre>int num= 0; for (int i=1; i&lt;=50; i++) {     System.out.println(num);     num= num+2; }</pre> |
|---------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|

A loop structure has the following elements:

- **Initialization Expression(s):** One or more variable is responsible for the loop to keep running. Hence, initialization expressions are required.
- **Test Expression (Condition):** Every loop runs if a certain condition is fulfilled. As long as the condition satisfies, the loop must keep on running. The test expression always results in either **true** or **false**.
- **Body of the Loop:** Here the tasks that need repetitive occurrence are present.
- **Update Expression(s):** The variable responsible for the loop to run must be updated in some way to make sure the loop runs a fixed number of times and not forever. Failure to update leads to an infinite loop. Which means, once we enter the loop, we can never get out, unless the program is closed forcefully.

There are mainly three types of loops in Java:

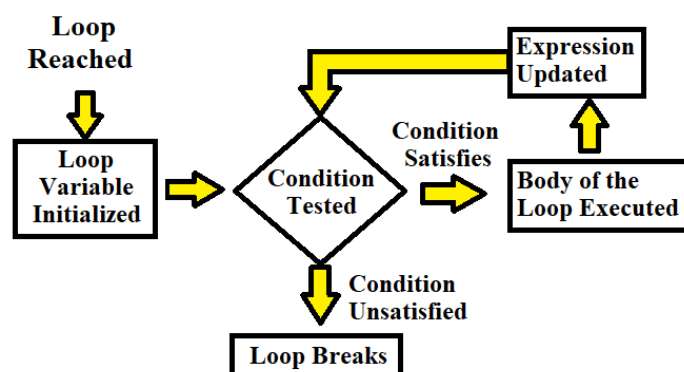
- While
- For
- Do-While

Among these three, while and for loops are entry-controlled loops and do-while is exit-controlled. We shall understand more about these right away.

### 8.2 WHILE LOOP AND FOR LOOP

While loop and for loop operate in a very similar way. Execution process and flowchart of both while loop and for loop is shown below:

1. Control reaches the while loop.
2. The Condition is tested.
3. If Condition is true, the flow enters the Body of the Loop. Goes to step 5.
4. If Condition is false, the loop breaks and the flow goes to the immediate next line after the loop.
5. The statements inside the body of the loop get executed.
6. Expression is updated.



7. Control flows back to Step 2. Thus, the loop keeps on running until the condition in step 2 is unsatisfied.

| While Loop Structure                                                                                                                          |                                                                                                                                      |                                                                                          |                        |
|-----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|------------------------|
| <i>initialization_expression;</i><br><b>while</b> ( <i>test_expression</i> ){<br><i>body_of_the_loop</i> ;<br><i>update_expression</i> ;<br>} |                                                                                                                                      | int count= 1;<br>while (count<4) {<br>System.out.println(count);<br>count= count+1;<br>} | Output:<br>1<br>2<br>3 |
| For Loop Structure                                                                                                                            |                                                                                                                                      |                                                                                          |                        |
| for ( <i>initialization_expression</i> ;<br><i>test_expression</i> ;<br><i>update_expression</i> ){<br><i>body_of_the_loop</i> ; }            | <i>initialization_expression</i><br>for ( ; <i>test_expression</i> ; ){<br><i>body_of_the_loop</i> ;<br><i>update_expression</i> ; } | //int i = 1;<br>for (int i=1; i<4; i++) {<br>System.out.println(i);<br>//i++; }          | Output:<br>1<br>2<br>3 |

It is not necessary for the expression initialization to be inside the for loop. It can be done before the loop starts. Similarly, the expression update can also be done inside the body of the for loop.

While and for loops are entry control loops because before entering the loop the test expression always checked. Therefore, cases might also occur where we cannot enter the loop at all.

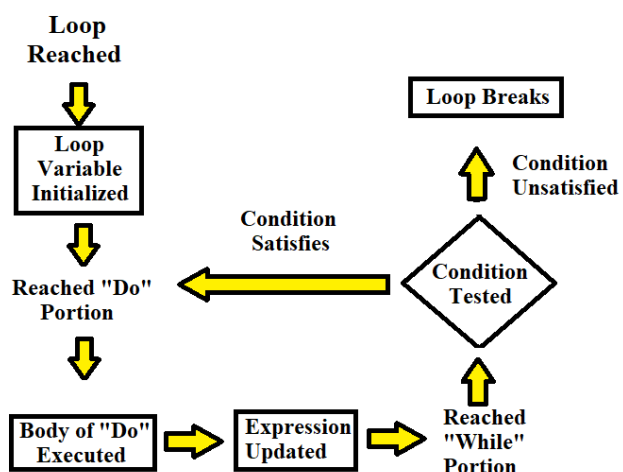
### 8.3 DO-WHILE LOOP

Do-while loop is an exit control loop because it is guaranteed for us to enter the loop. The loop will definitely run once, may or may not run more times. Therefore, the exit is controlled in do-while instead of the entry. The “Do” portion of do-while loop ensures the execution of the loop at least once. The “While” portion checks whether the loop should run further or not. If the condition in the “While” portion satisfies, the body of the “Do” portion is executed again.

Inspiring Excellence

Execution process and flowchart of do-while loop is shown below:

1. Control reaches the “Do” portion.
2. The Body of the Loop (Do) is executed.
3. Expression is updated.
4. Control reaches the “While” portion and the test condition is tested.
5. If Condition is true, the flow enters the Body of the Loop. Goes to step 2.
6. If Condition is false, the loop breaks and the flow goes to the immediate next line after the loop.



### Do-While Loop Structure

|                                                                                                                                           |                                                                         |                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <i>initialization_expression;</i><br>do {<br><i>body_of_the_loop;</i><br><i>update_expression;</i> }<br>while ( <i>test_expression</i> ); | int x = 1;<br>do {<br>System.out.println(x);<br>x++;}<br>while (x < 5); | Output:<br>1<br>2<br>3<br>4                                                                           |
|                                                                                                                                           | int x = 1;<br>do {<br>System.out.println(x);<br>x++;}<br>while (x < 1); | Output:<br>1<br><br>Here the loop runs once even though the test condition was always unsatisfactory. |

#### 8.4 INFINITE LOOP

An infinite loop runs forever because the test expression is always satisfied. Infinite loops are a common mistake made by the students when they forget to update the loop controlling variable and therefore, the loop never breaks. Some ways of creating infinite loops are shown below.

| Loop Type | Examples                                              |                                                              |                                                     |
|-----------|-------------------------------------------------------|--------------------------------------------------------------|-----------------------------------------------------|
| While     | while(true){<br>System.out.println(5); }              | int x=1;<br>while(x>0){<br>System.out.println(5);<br>x+=1; } | int x=1;<br>while(x>0){<br>System.out.println(5); } |
| For       | for (int i =1; i>0; i++){<br>System.out.println(5); } | for (int i =1; i>0;){<br>System.out.println(5);<br>}         | for ( ; ; ){<br>System.out.println(5);<br>}         |

Infinite loops can be used to solve problems that require indefinite number of iterations. You will learn more on these in Chapter 8.7.

#### 8.5 “BREAK” AND “CONTINUE” KEYWORDS

When we need to stop the loop before finishing all the iterations, we use the **break** keyword. If we want to skip a certain iteration, we use the **continue** keyword.

|                                                                                                                 |                                                                                         |                                                                                                                      |
|-----------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| int count=1;<br>while(count<5){<br>System.out.println(count);<br>count++;<br>if (count==3){<br>break;<br>}<br>} | for (int i=1; i<5; i++){<br>if (i==3){<br>continue;<br>}<br>System.out.println(i);<br>} | for (int i=1; i<10; i++){<br>if (i==3){<br>continue;<br>}<br>if (i>3){<br>break;<br>}<br>System.out.println(i);<br>} |
| Output:<br>1<br>2                                                                                               | Output:<br>1<br>2<br>4                                                                  | Output:<br>1<br>2                                                                                                    |
| Here, after the second iteration, the loop breaks.                                                              | Here, the third iteration is skipped.                                                   | Here, the third iteration is skipped and the loop breaks in the fourth iteration.                                    |

Remember that in case of nested loops (a loop inside another loop), the break or continue keywords will only apply on the loop these keywords are directly in, not on the outer loops.

## 8.6 NESTED LOOPS

Let us understand the concept of understanding nested loops using a code example.

Print the number of divisors of all the numbers ranging from 1 to 10.

Output:

```
1 has 1 divisor(s)
2 has 2 divisor(s)
3 has 2 divisor(s)
4 has 3 divisor(s)
5 has 2 divisor(s)
6 has 4 divisor(s)
7 has 2 divisor(s)
8 has 4 divisor(s)
9 has 3 divisor(s)
10 has 4 divisor(s)
```

```
class DivisorCount {
 public static void main(String[] args) {
 for (int num=1; num<=10; num++){
 int div=0;
 for (int i=1; i<=num; i++){
 if (num%i==0){
 div++;
 }
 }
 System.out.println(num+ " has "+div+" divisor(s)");
 }
 }
}
```

### Explanation:

If we had to find the number of divisors of one number, we could accomplish this with just one loop. However, in this case we are told to find the number of divisors of numbers from 1 to 10. Therefore, we have two repetitive tasks here that are dependent on each other.

- The inner loop is used to count the number of divisors
- The outer loop is used to repeat the inner loop from numbers ranging from 1-10.

Now, let us understand how **break** and **continue** keywords work in a nested loop.

```
class A {
 public static void main(String[] args) {
 for (int x=1; x<=5; x++){
 for (int i=1; i<=x; i++){
 if (x==1){
 continue;
 }
 if (x==4){
 break;
 }
 System.out.println(i);
 }
 }
 }
}
```

Output:

```
1
2
1
2
3
1
2
3
4
5
```

Here, if you notice closely, having the break or continue keywords in the inner loop affected the inner loop and not the outer loop.

## 8.7 DEFINITE VS INDEFINITE REPETITION

When we know exactly how many times the loop will run (Definite), we can solve the problem running the loop a fixed number of times. If the number of iterations is unknown or may vary (Indefinite), then we may use an infinite loop with break statement.

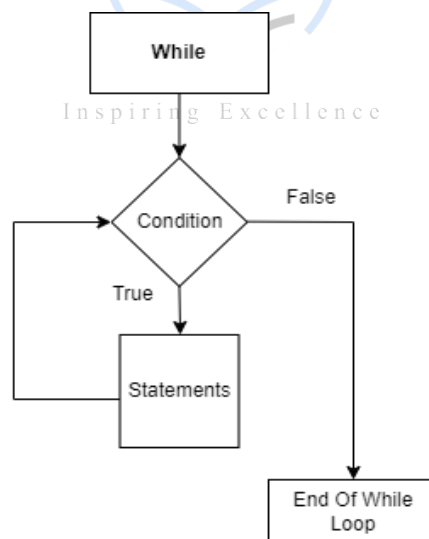
For example, suppose you have to keep on taking inputs from the user until they enter 0. Now, there is no guarantee when the user will enter 0. The user could enter 0 in the very first iteration, or even in the 100<sup>th</sup>

iteration. So, you have to run your loop indefinitely and break it when the condition is fulfilled. Here are a few examples to help you understand when to use definite and indefinite loops.

| Definite                                                                                         | Indefinite                                                                                                                                              |                                                                                                                                                                                       |              |
|--------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| Print all the even numbers within 1 to 10.                                                       | Print the first five numbers divisible by 3 within 1 to 25.                                                                                             | Keep taking user inputs until the user enters an even positive number. Then print that number.                                                                                        |              |
| <pre>for (int i=1; i&lt;=10; i++){     if (i%2==0){         System.out.println(i);     } }</pre> | <pre>int num=0; for (int i=1; i&lt;=25; i++){     if (i%3==0){         System.out.println(i);         num++; }     if (num==5){         break;} }</pre> | <pre>Scanner sc= new Scanner(System.in); while(true){     int num= sc.nextInt();     if (num%2==0 &amp;&amp; num&gt;0){         System.out.println(num);         break;     } }</pre> |              |
| Output:<br>2<br>4<br>6<br>8<br>10                                                                | Output:<br>3<br>6<br>9<br>12<br>15                                                                                                                      | Input:<br>5<br>0<br>-4<br>6                                                                                                                                                           | Output:<br>6 |

## 8.8 FLOWCHARTS

The following flowchart is what a flowchart of a loop looks like. After encountering a while loop, the condition is checked. If the condition satisfies, we enter the loop. We keep on iterating and executing the statements inside the loop as long as the condition satisfies. We break out of the loop as soon as the condition is dissatisfied.



Now we shall look at a few problem examples and their corresponding flowcharts.

---

## 8.9 WORKSHEET

### Flowchart Problems

- A. Draw the flowchart of a program that calculates the sum of all the numbers from 1 to 100 using a while loop.
- B. Draw the flowchart of a program that prompts the user to enter a password. Keep asking for the password until the user enters the correct password "abc123". Once the correct password is entered, display a message "Access granted!".
- C. Draw the flowchart of a program that prompts the user to enter a positive integer and then prints all the even numbers from 2 up to and including that number.
- D. Draw the flowchart of a program that prompts the user to enter a series of numbers. Continue reading numbers until the user enters a negative number. Then, calculate and display the average of all the positive numbers entered.

### Coding problems

- A. Write a program that asks the user to enter a series of numbers and calculates their average. Display the average to the user.
- B. Write a program that prompts the user to enter a positive integer and prints the factorial of that number.
- C. Write a program that prints the multiplication table for a given number from 1 to 10.
- D. Write a program that prints the ASCII values and characters for all uppercase letters (A-Z) using a loop.
- E. Write a program that prompts the user to enter a positive integer and checks if it is a prime number. Display an appropriate message indicating whether the number is prime or not.
- F. Write a program that prompts the user to enter a positive integer and checks if it is a perfect number. Display an appropriate message indicating whether the number is perfect or not.
- G. Write a program that prompts the user to enter a positive integer and checks if it is a palindrome number. A number is a palindrome if it reads the same forwards and backwards. Display an appropriate message indicating whether the number is a palindrome or not.

## CHAPTER 9: STRINGS

### 9.1 DECLARATION AND INITIALIZATION

Strings are sequences of characters but in Java strings are implemented as objects.

There are **two** ways of creating a string in Java:

1. Using string literal - A new string object is created by using double quotation marks like

```
String str1 = "Hello world!";
String empty_str = ""; #this is an empty
string
```

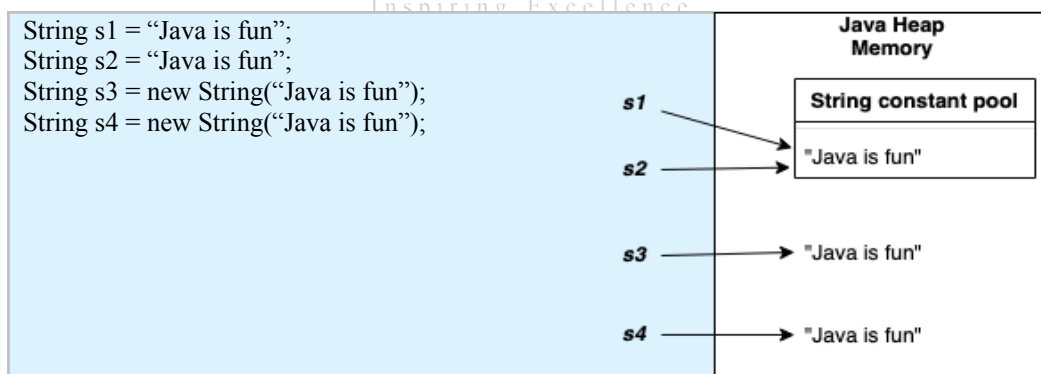
Using a string literal to create a string is more memory efficient for Java since it checks the string constant pool after a string object is created. If the string is already present in the string constant pool, then only the reference to the string is returned and if the string does not exist only then a new string object is created. You can consider the string constant pool as a special memory location.

2. Using the new keyword, i.e., using the String class constructor - A new string object is created by using the String class constructors. There are 13 constructors in the String class which can be used to create a string object. The most common ones are

```
String str2 = new String("Hello world!");
String empty_str = new String(); #this is an empty string
char [] character_array = {'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd',
'!'};
String str3 = new String(character_array);
```

Using the new keyword creates a new String object each time in the heap memory.

Let's write these 4 lines of code and see how the string objects are stored in the diagram below.



When `s1` is created, there is no String literal "Java is fun" in the string constant pool and so it is added there. When `s2` is created, the string "Java is fun" is already present in the pool and hence its reference is only returned to `s2`, and that is why `s1` and `s2` both point at the same String object. But `s3` and `s4` are created using the String constructor and so both created new String objects which reside in the heap and both have separate locations.

### 9.2 STRING INPUT

We need to import the Scanner class in order to take a string input from the user. Let's see an example.



```
import java.util.Scanner;
class Main {
 public static void main(String[] args) {
 Scanner sc = new
Scanner(System.in);
 String user_input = sc.nextLine();
 System.out.println(user_input);

 }
}
```

Notice that we used `sc.nextLine()` to take a string input because `nextLine()` takes the input until a new line is encountered i.e. a `\n` is encountered.

### 9.3 STRING ESCAPE SEQUENCE

If we want to create strings which would incorporate single or double quotation marks then we need an escape sequence to create such strings i.e. `\`. Let's see an example:

```
String s = "I love \'Java\' very
much";
System.out.println(s);
```

The output of the above code is "I love 'Java' very much. Try the above code by replacing the single quotations with double quotes.

### 9.4 STRING LENGTH

We can find the size of a string i.e. we can find how many characters are present in a string using the **length()** method like:

```
String s = "I love Java";
System.out.println(s.length())
;
```

The output of the above code is 11 since there are 11 characters in `str5` including the spaces.

### 9.5 STRING INDEXING

Each character in a string can be accessed using its position in the string. These positions are called index numbers and are numbered from 0 to length of the string -1.

```
String s = "Programming is fun";
```

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|
|       | P | r | o | g | r | a | m | m | i | n | g  |    | i  | s  |    | f  | u  | n  |

It can be seen that the first element can be found in `index=0` and the last element can be found in `index=length-1`.

In order to access a particular element from the string, we can use the method **charAt()**. This method takes an index number as an argument and returns the particular element in that position. Let's extract the characters 'P' at `index=0`, 'n' at `index=17` and 'g' at `index 10`.

```
String s = "Programming is fun";
char first_character = s.charAt(0);
char last_character = s.charAt(s.length() - 1);
char random = s.charAt(10);
System.out.println(first_character);
System.out.println(last_character);
System.out.println(random);
```

## 9.6 CHARACTER ARRAY

We know that strings are a sequence of characters and each character has an index number defining its position in the string. A character in Java is represented using single quotation marks. A string can be converted into an array of its characters using the **toCharArray()** method like :

```
String s = "This is a string" ;
char [] char_array =
s.toCharArray();
System.out.println(char_array);
```

The above code will convert the string *s* to a character array.

## 9.7 MUTABILITY OF STRINGS

In the previous section 9.3, we saw how we can access the elements in a string. However, we cannot modify a string because in Java, strings are **immutable**. This means that we cannot modify the elements which make up the string. If we want to modify a string then a new string needs to be created which would contain the modifications, leaving the original string unchanged.

## 9.8 STRING CONCATENATION

We can connect multiple strings together by using the **+** operator between them. This process is called concatenation where a new string is created by chaining one/more strings together.

```
String s1 = "I love Java " ;
String s2 = "programming " ;
String s3 = s1 + s2 + "very
much";
System.out.println(s3);
```

The output of the above code is: *I love Java programming very much*. The output is a new string which is stored in *s3*. Let's see another example:

```
String s1 = "CSE";
int num = 110;
String output = s1 + num
System.out.println(output)
;
```

Here, notice that we are concatenating an integer to a string using the **+** operator. In this case the output is a string *CSE110*. This means that, wherever an operand of the **+** operator will be a string, the Java compiler will automatically convert the other operand to a string representation and then contact it to create a new string.

Let's see another example:

```
String semester = "Fall ";
int y1 = 20;
String output = semester + y1 +
23;
System.out.println(output);
```

Here, the output is: *Fall2023*. Note how operator precedence causes the concatenation of the string "Fall" to the integer 20 at first which creates the string "Fall20" and then this string is concatenated to the integer 23 which creates the output string "Fall2023".

Had we written: **String output = y1 + 23 + semester**, then the operator precedence would cause the integer in y1 and the integer 23 to be added using the + operator to produce an integer 43 and then this integer would be concatenated to the string "Fall" to produce a string "43Fall". Try this out yourself.

## 9.9 COMPARING STRINGS

- **equals()** - This method checks whether the contents of two strings are identical or not. It returns true if two strings contain the same elements and false otherwise. Let's see an example:

```
String s1 = "Let us code";
String s2 = "Java";
String s3 = "Let us code";
System.out.println(s1.equals(s2))
;
System.out.println(s1.equals(s3))
;
```

Here, the output of the first print statement is *false* since the elements of s1 and s2 are identical and the output of the second print statement is *true* since the elements of s1 and s3 are identical.

Note: equals() method is case sensitive and so the strings "Let us code" and "Let us Code" will yield false. However, another method **equalsIgnoreCase()** will ignore the case differences and will yield true.

- **== operator** - The == operator in Java checks whether two string objects have the same reference/location or not. Let us see an example:

```
String s1 = new String("Let us
code");
String s2 = new String("Let us
code");
System.out.println(s1.equals(s2));
System.out.println(s1 == s2);
```

Here, the output of the first print statement is *true* since the elements of s1 and s2 are identical and the output of the second print statement is *false* since s1 and s2 refer to different string objects stored in different memory locations (Refer to the heap memory where string objects are stored when created using the new keyword).

- **compareTo()** - This method compares two strings lexicographically. It returns 0 if both strings are equal with identical characters in the same positions. It returns a positive integer if the first string is lexicographically greater than the second string and it returns a negative integer if the first string is lexicographically smaller than the second string. Let's see a few examples.

```
String s1 = "Java";
String s2 = "Java";
String s3 = "Jade";
String s4 = "Lava";
System.out.println(s1.compareTo(s2))
;
System.out.println(s1.compareTo(s3))
;
System.out.println(s1.compareTo(s4))
;
```

The first output is 0 since both s1 and s2 are identical. The second output is 18 since "v" is 18 more than "d" lexicographically. The third output is -2 since "J" is 2 less than "L" lexicographically.

## 9.10 SEARCHING A CHARACTER IN A STRING

Sometimes we need to find the index number of a character in a string. There are 2 methods which help us to do this:

- **indexOf()** - This method takes a character/substring as an argument and finds the position of the first occurrence of the character/substring in the string.
- **lastIndexOf()** - This method takes a character/substring as an argument and finds the position of the last occurrence of the character/substring in the string.

Let us see an example:

```
String s1 = "random string";
System.out.println(s1.indexOf('n'));
System.out.println(s1.lastIndexOf('n'))
;
System.out.println(s1.indexOf("ring"))
;
```

The output of the first print statement is 2 since the first occurrence of the character 'n' is in index=2 and the output of the second print statement is 11 since the last occurrence of the character 'n' is in index=11. The 3rd output is 9 since the substring "ring" is found from index=9.

## 9.11 STRING SLICING

The **substring()** method helps us to slice a string in order to create a substring. If we mention only the starting position *start\_index*, then it produces a new substring starting at the *start\_index* to the end of the string. If we mention both the *start\_index* and the *end\_index*, then it produces a new substring starting at the *start\_index* and ending at the *end\_index*. Let us see an example:

```
String s1 = "Java Programming is fun";
String sub1 = s1.substring(5);
String sub2 = s1.substring(5,12);
System.out.println(sub1);
System.out.println(sub2);
```

The output of the first print statement is *Programming is fun* and the output of the second print statement is *program*. Notice that the *end\_index* is exclusive which means the substring will stop at the index before the *end\_index* and not include the character at the *end\_index* itself.

## 9.12 MODIFYING A STRING

We know that strings are immutable but we have been using different string methods to produce new strings which contain the modifications. If we want to replace a character or a substring with a different character or substring then we can use the **replace()** method. This method creates a new string which contains the replaced character or the substring. Let's see an example:

```
String s1 = "marathon";
String new_s1 = s1.replace("a", "e");
String new_s2 = s1.replace("mara",
"py");
System.out.println(new_s1);
System.out.println(new_s2);
```

The output of the first print statement is *merethon*. We can see that all the occurrences of "a" have been replaced by "e". The output of the second print statement is *python*. We can see that the substring "mara" has been replaced by the substring "py".

### 9.13 TRIMMING AND SPLITTING A STRING

- **trim()** - This method removes any leading and trailing whitespaces and returns a new string as a result. Let's see an example :

```
String s1 = " Java ";
String s2 = s1.trim();
System.out.println(s2);
```

The output of the above code is *Java* without the white spaces before and after the word "Java".

- **split()** - This method converts a string to a string array on the basis of a given parameter as delimiter. Let's see an example :

```
String s1 = "Java is fun";
String [] s1_array = s1.split("
");

String s2 = "Java,is,,fun";
String [] s2_array =
s2.split(",");
```

### 9.14 CASE CONVERSION

We can convert uppercase alphabetical characters to lowercase and vice versa using two methods.

- **toLowerCase()** - This method converts all uppercase letters to lowercase in a string and returns a new modified string.

```
String s1 = "Java Programming is
FUN";
String lower_s1 = s1.toLowerCase();
System.out.println(lower_s1);
```

The output of the above code is *java programming is fun* where all characters are presented in lowercase.

- **toUpperCase()** - This method converts all lowercase letters to uppercase in a string and returns a new modified string.

```
String s2 = "i love to CODE";
String upper_s2=
s2.toUpperCase();
```

```
System.out.println(upper_s2);
```

The output of the above code is *I LOVE TO CODE* where all characters are presented in uppercase.

## 9.15 DATA CONVERSION IN STRINGS

- **valueOf()** - This method of String class is used to convert other data types to String type.

```
int digit = 85;
float decimal = 54.643F; //F represents
float
char letter = 'A';

String s1 = String.valueOf(digit);
System.out.println(s1);

String s2 = String.valueOf(decimal);
System.out.println(s2);

String s3 = String.valueOf(letter);
System.out.println(s3);
```

The variable s1 will contain a string representation of 85 ("85"), the variable s2 will contain a string representation of 54.643 ("54.643") and the variable s3 will contain a string representation of 'A' ("A").

## 9.16 ASCII VALUES

All keyboard characters have unique Unicode values. Refer to this ASCII code table to see the unicode values corresponding to each character: <https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html>.

Sometimes we need to work with ASCII values of strings or characters. In that case we need to convert an element in a string into its corresponding Unicode value. We can use the **codePointAt()** method which takes an index of a string as a parameter and returns the Unicode value of the element in the given index.

```
String s1 = "Hello World!";
int unicode_val =
s1.codePointAt(0);
System.out.println(unicode_val);
```

The output of the above code is 72 which is the Unicode value of index=0 element, that is, "H". Try to find the Unicode value of "W" by yourself.

We can also use these Unicode values to convert uppercase letters to lowercase and vice versa. Visit the link above and check the Unicode value of uppercase "A". You'll see that it's 65. Now, check the Unicode value of lowercase "a". You'll see that it's 97. The difference between the two values is 32. That means if we add 32 to the Unicode value of "A" we will get the Unicode value of "a" and if we subtract 32 from the Unicode value of "a" we will get the unique value of "A". If we cast the Unicode value using (char) then we will get the corresponding character. This will follow for all alphabets from "A" to "Z" and from "a" to "z". Let's see an example:

```
String upper_s1 = "CODE";
int unicode_upper =
upper_s1.codePointAt(2);
int unicode_lower = unicode_upper + 32;
char lower_letter = (char)unicode_lower;
System.out.println(lower_letter);
```

The output of the above code is the character *d*. The Unicode at index=2 (uppercase 'D') of "CODE" is 68 and if we add 32 to it, we will get 100 which is the Unicode value of lowercase 'd' and hence converting the value to character using char casting gives us the letter 'd'.

---

### 9.17 CHECKING SUBSTRING EXISTENCE

- **contains()** - This method checks whether a substring is present in a string or not. It returns *true* if the substring exists in the string and *false* otherwise.

```
String s1 = "I love Java programming";
Boolean res1 = s1.contains("Java");
Boolean res2 = s1.contains("love Java
");
Boolean res3 = s1.contains("p");
Boolean res4 = s1.contains("java");
System.out.println(res1);
System.out.println(res2);
System.out.println(res3);
System.out.println(res4);
```

The output of the first three prints are *true* since “Java”, “love Java “ and “p” all these substrings exist in s1 but the output of the last print is *false* since “java” with lowercase “j” does not exist in s1.



Inspiring Excellence

### 9.18 WORKSHEET

A. Write a Java program that takes a string as input and reverses it.

Sample input 1:

Programming

Sample output 1:

gnimmargorP

B. Write a Java program that takes a string as input and counts the number of vowels (a, e, i, o, u) present in the string.

Sample input 1:

Programming

Sample output 1:

3

C. Write a Java program that takes a string as input and checks if it is a palindrome. A palindrome is a word, phrase, number, or other sequence of characters that reads the same backward as forward. The program should output "Palindrome" if the string is a palindrome and "Not a Palindrome" otherwise.

Sample input 1:

ABCDCBA

Sample output 1:

Palindrome

Sample input 2:

Nepal

Sample output 2:

Not a Palindrome

D. Write a program which takes a string as input and returns the number of characters in the string.

Sample input 1:

Programming

Sample output 1:

11

E. Input a word into a String. Print each character on a line by itself.

Sample input 1:

Program

Sample output 1:

P

r

o

g

r

a

m

F. Your task is to input a word into a String. Then print code for each character in the String using the 2<sup>nd</sup> method discussed above.

Sample input 1:

Pro

Sample output 1:

P : 80

r : 114

o : 111





G. Print the statistics of occurrence of each character on a line by itself. Assume that user will only give CAPITAL letters. So, you will have to count values of CAPITAL letters only.

Sample input 1:

BALL

Sample output 1:

A which is 65 was found 1 time(s)

B which is 66 was found 1 time(s)

L which is 76 was found 2 time(s)

H. Input a word into a String.

Print the word.

Print the word again after adding "==THE END==" at the end of the word.

Then print the word again.

Your whole program may contain the word "String" at most two times.

Sample input 1:

Programming

Sample output 1:

Programming

Programming==THE END==

Programming

I. Answer the following theoretical questions:

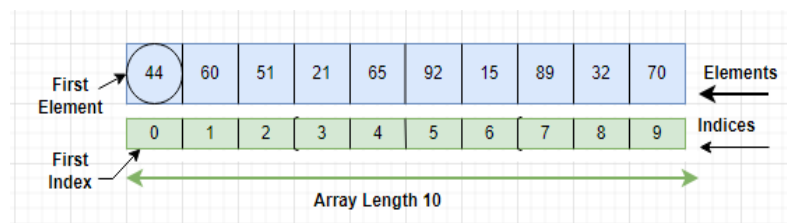
- What is string immutability in Java? Explain why strings are immutable and discuss the advantages and disadvantages of immutability.
- Discuss the significance of the "equals()" method versus the "==" operator when comparing strings in Java.
- What is the purpose of the "substring()" method in Java? Explain how it can be used to extract substrings from a larger string.
- Describe the process of converting a string to a numeric value in Java. Explain the use of the "parseInt()" and "valueOf()" methods for this purpose.
- What are some common methods available in the String class in Java? Provide examples of how these methods can be used.

## CHAPTER 10: ARRAYS

### 10.1 PROPERTIES

Information storage is an important task for programmers to perform in today's technologically advanced world. For later use, each and every item of information must be stored in the memory location. We already know what a variable is and how it stores data. The same way, data is stored using an array. However, Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. It is a data storage where we store a fixed set of similar elements. For example, if we want to store the names of 40 students then we can create an array of the string type that can store 40 names.

Any data type, including primitive types like integer, double, and Boolean as well as object types like String, can be stored in a Java array. Arrays can also be multi-dimensional, meaning that they can have multiple rows and columns. An array's length is assigned when it initially gets constructed. After creation, its length is fixed and cannot be changed. Arrays in Java are index-based, the first element of the array is stored at the 0th index, the 2nd element is stored on the 1st index and so on.



### 10.2 DECLARATION, CREATION AND INITIALIZATION

Making an array in a Java program involves three distinct steps:

- Declare the array name.
- Create the array.
- Initialize the array values.

To create an array, an array variable of the desired data type must be declared first. The following is the general form for declaring an array:

```
data_type variable_name [] ;
```

Here, the data type determines what type of elements will be stored in the array. For example:

```
int quiz_marks [] ;
```

In the above declaration `quiz_marks` is the name of the array. And this array will store only the integer types of elements. Although this declaration establishes the fact that `quiz_marks` is an array variable, no array actually exists.

To link `quiz_marks` with an actual, physical array of integers, we must allocate one actual array using the “new” keyword and assign it to `quiz_marks`. “new” is a special operator that allocates memory for arrays.

```
quiz_marks = new int [5] ;
```

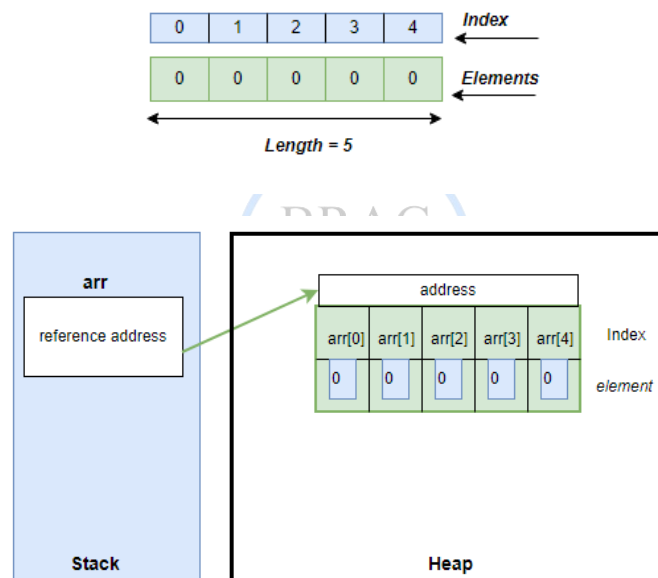
Now, we have finally created the actual array. Here, 5 defines the length of the array. That means this array can store five elements.

<sup>data type of each element</sup>  
`int [ ]`
<sup>name of the array</sup>  
`total_marks;` } Declaration  
  
`total_marks = new int [5];` } Creation  
<sup>Array length/ total number of elements</sup>

This was the descriptive process for declaration and creation of an array. There is another short way for declaring and creating an array. Instead of using two separate lines of codes we can simply use one line for declaration and creation of an array.

**`int [ ] total_marks = new int [5];`**

Here, we have created an integer array, which can store 5 elements. Initially when we create an integer array, all the elements of the array remain 0.



Stack is a temporary memory. after using new, now the array will be inserted into an actual data storage memory called heap. Also, the name of the array only refers to the address where the actual array is stored.

Initial elements of the array change according to the data type. According to different data type array initial elements:

| Data Type | Default Values                         |
|-----------|----------------------------------------|
| Byte      | 0                                      |
| Short     | 0                                      |
| Int       | 0                                      |
| Long      | 0                                      |
| Float     | 0.0                                    |
| Double    | 0.0                                    |
| Boolean   | false                                  |
| Char      | '\u0000' which is the Unicode for null |
| String    | null                                   |

|        |      |
|--------|------|
| Object | null |
|--------|------|

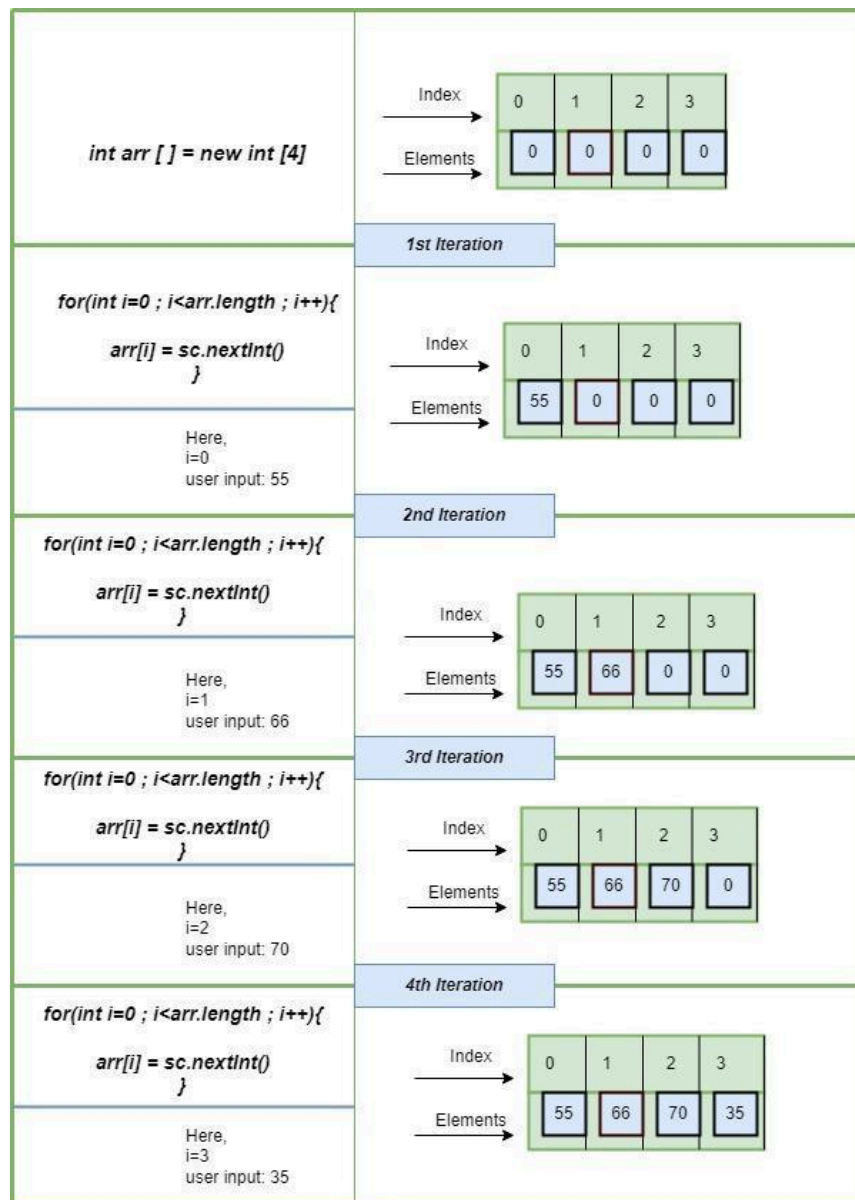
Array declaration and creation examples with different data types:

|                                              |                                                                                                                                                                                    |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>int [ ] a1 = new int [5];</code>       | <p>Diagram illustrating the creation of an <code>int</code> array of length 5. The array contains five elements, all initialized to 0. The indices are 0, 1, 2, 3, and 4.</p>      |
| <code>double [ ] a1 = new double [5];</code> | <p>Diagram illustrating the creation of a <code>double</code> array of length 5. The array contains five elements, all initialized to 0.0. The indices are 0, 1, 2, 3, and 4.</p>  |
| <code>float [ ] a1 = new float [5];</code>   | <p>Diagram illustrating the creation of a <code>float</code> array of length 5. The array contains five elements, all initialized to 0.0. The indices are 0, 1, 2, 3, and 4.</p>   |
| <code>String [ ] a1 = new String [5];</code> | <p>Diagram illustrating the creation of a <code>String</code> array of length 5. The array contains five elements, all initialized to null. The indices are 0, 1, 2, 3, and 4.</p> |

To sum up, there are two steps involved in constructing an array. First, you must create a variable with the specified data type. Second, you must use the “new” keyword to allocate the memory required to store the array and then assign that memory to the array variable. After that default elements will be assigned automatically to the array according to the data type.

Now, we need to store our actual elements. There are multiple ways to assign values into an empty array. The general code for inserting any element into any specific index of an array is:

**array\_name [index\_number] = element**



Here, at first, we declared the array with length 4. We declared the data type integer that is why the initial array is loaded with all 0's. After that we initiated the for loop, where we stated  $i=0$ . Because the first index of an array is 0. The loop will stop when we reach the last index of the array. For that reason we have to know the length of the array. In Java, the array length is the number of elements that an array can hold. There is no predefined method to obtain the length of an array. We can find the array length in Java by using the array attribute length. We use this attribute with the array name. In the above example if we write:

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

Our program will give us 4 as an output. So, the last index of an array is  $\text{array.length}-1$ .

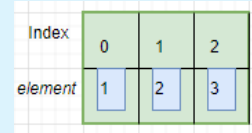
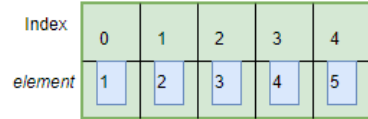
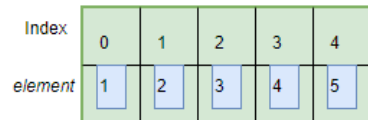
In the above diagram, the value of "i" becomes: 0,1,2,3.

So, the first index is 0 and the last index is 3 which is the length - 1. Inside the loop we are taking inputs from the user and inserting that input into the array index.

```
arr[i] = sc.nextInt();
```

In this line, arr is the name of the array, i is the index number. `sc.nextInt()` is taking inputs from the user. Here, `nextInt` only takes the integer inputs

In this way we can take inputs from the user and create an array. There are various ways to initialize the array values. Some of them are given below:

| Process                           | Description                                                                                                                                                                                                                                                                                                                           | Code                                                                                                                                                                                                       |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Dynamically allocating one by one | Here, a is the name of the array. a[0] means the first index of that array. so in the first index we insert the element 1 and so on                                                                                                                                                                                                   | <pre>int a [] = new int [3]; a[0] = 1; a[1] = 2; a[2] = 3;</pre> <p>Created array:</p>                                  |
| At the time of declaration        | We can also insert the elements at the time of array creation. here at first we declared the data type which is int, then a is the name of the array. And then on the right side of the equal sign inside curly braces we write the elements we want to insert.                                                                       | <pre>int a[] = { 1, 2, 3, 4, 5 };</pre> <p>Created array:</p>                                                           |
| Using loop                        | Using for or while loop we can also insert values into an array. Here, the loop starts from 0, and it will continue until it doesn't reach the last index of the array. The length of the array is 5. the value of i will be 0,1,2,3,4.                                                                                               | <pre>int a [] = new int [5]; for (int i = 0; i &lt; a.length; i++) {     a[i] = i + 1; }</pre> <p>Created array:</p>  |
| Taking User Input                 | We can insert elements into an array by taking user inputs.<br>a[i] = sc.nextInt();<br>In this line, a is the name of the array, i is the index number. sc.nextInt() is taking inputs from the user. Here, nextInt only takes the integer inputs.                                                                                     | <pre>import java.util.Scanner; Scanner sc= new Scanner(System.in); int a[] = new int [5]; for (int i = 0; i &lt; 5 ; i++) {     a[i] = sc.nextInt(); }</pre>                                               |
| Random value generator            | Java has a special class called Random. First we need to import that class. After that we create an object of that class just like a scanner.<br>rd.nextInt(10). Here, rd is the object name, nextInt will generate only integer values and 10 defines the range. Any random numbers between 0 to 10 will be inserted into the array. | <pre>import java.util.Random; Random rd = new Random(); int a[] = new int [5]; for (int i = 0; i &lt; 5 ; i++) {     a[i] = rd.nextInt(10); }</pre>                                                        |

In this way, we can declare, create and initialize an array in java. For better efficiency practice different approaches with different data types for array initialization.

### 10.3 ARRAY ITERATION

Arrays are used to store homogeneous elements, meaning the same type of elements. Till now we have already learned what an array is and how to declare, initialize an array. Throughout this process we stored our necessary elements into the array. Our task is not only to store the values but also we need to work with the values. Here

comes the iteration part where we need to access the elements from the array. Iterating over an array means accessing each element of the array one by one.

```
String student_name [] = {"David", "Jon", "Sam", "Elsa", "Anne"};
System.out.println(name[0]);
```

Output:  
David

Suppose we have an array named `student_name`. Where we stored the name of those students who were the top scorers in the midterm examination. Now you guys want to know the name of the student who got the highest marks and became first.

```
System.out.println(name[0]);
```

Index 0 contains the highest scorer. So inside the print statement if we write the array name with index 0. then the output will show the highest scorer's name. Here, we just printed the name. We can also store elements as separate variables from the array.

```
String student_name [] = {"David", "Jon", "Sam", "Elsa", "Anne"};
top_scorer = student_name [0]
System.out.println(top_scorer);
```

Output:  
David

Here, we created a new variable called `top_scorer` and copied an element from the array inside the new variable. The element will remain the same in the array, we just copied it and stored that copy inside a new variable.

|                            |                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------|
| Print the element          | <code>System.out.println (array_name [ index_number ] );</code>                                              |
| Copy and print the element | <code>Variable_name = array_name [index_number];</code><br><code>System.out.println (Variable_name );</code> |

Sometimes, we need to iterate the whole array instead of a specific index. There are multiple ways to iterate over a whole array. Mostly we use a loop for that process. The first index of an array is 0 so we initialize for loop variable `i = 0`. Then `i` will continue incrementing by 1 value until it doesn't reach the last index which is `array length - 1`. Inside the loop we write the print statement for printing the array.

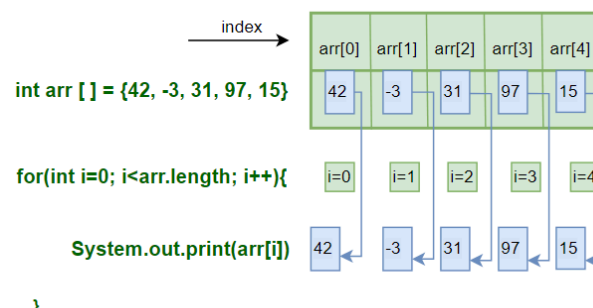
```
int arr [] = {42,-3,31,97,15};

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

**Output:**

42  
-3  
31  
97  
15

Here, loop variable `i` works as the index value and `i` iterate over the array.



We can iterate an array using not only for loop but also while loop and do while loops. As per our convenience and requirements we can use any of the iteration processes.

|          |                                                                                                                                                 |                                       |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
| For Loop | loop starts from 0, runs until it reaches the last index which is array length-1. <code>i</code> increments by 1. iterate over the whole array. | <code>int a [ ] = {5,6,9,8,6};</code> |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|

|               |                                                                                                                                                                                                                         |                                                                                                                      |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
|               | inside the loop, print statement prints the elements.                                                                                                                                                                   | <pre>for (int i = 0; i &lt; a.length; i++) {     System.out.println(a[i]); }</pre>                                   |
| While Loop    | <p>loop starts from 0, runs until it reaches the last index which is array length-1.</p> <p>inside the loop, print statement prints the elements and i increments by 1. Iterate over the whole array.</p>               | <pre>int a [] = {5,6,9,8,6}; int i = 0; while (i &lt; a.length) {     System.out.println(a[i]);     i++; }</pre>     |
| Do While Loop | The initial variable i starts from 0. inside the do block we write the statements to be executed which is printing elements of the array. Then we increment i by 1. Then using while we set the value to stop the loop. | <pre>int a [] = {5,6,9,8,6}; int i = 0; do {     System.out.println(a[i]);     i++; } while (i &lt; a.length);</pre> |

#### 10.4 OPERATIONS ON AN ARRAY

Several different operations can be performed on an array. Some examples are given below. For all the examples, we shall use this array:

```
int arr [] = {55, 66, 7, -2, 9};
```

| Operation | Description                                                                                                                                                         | Code                                                                                                                                             | Output                   |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| Length    | Counts total number of elements in an array.                                                                                                                        | <pre>System.out.println(arr.length);</pre>                                                                                                       | 5                        |
| Traverse  | Traversing through all the elements in the array one by one. Can be used to perform tasks that require visiting all the elements such as printing all the elements. | <pre>for (int i = 0; i &lt; arr.length; i++) {     System.out.println(arr[i]); }</pre>                                                           | 55<br>66<br>7<br>-2<br>9 |
| Insertion | Adds an element at the given index.                                                                                                                                 | <pre>int x [] = new int [5]; x [1] = 10; System.out.println(x[1]); for (int i = 0; i &lt; x.length; i++) {     System.out.println(x[i]); }</pre> | 0<br>10<br>0<br>0<br>0   |
| Deletion  | Deletes an element at the given index.                                                                                                                              | <pre>arr [1]= 0; for (int i = 0; i &lt; arr.length; i++) {     System.out.println(arr[i]); }</pre>                                               | 55<br>0<br>7<br>-2<br>9  |
| Update    | Updates an element at the given index.                                                                                                                              | <pre>arr [1]= 65; for (int i = 0; i &lt; arr.length; i++) {     System.out.println(arr[i]); }</pre>                                              | 55<br>65<br>7<br>-2<br>9 |



|                  |                                                                                                                                                                                                                             |                                                                                                                                                                                              |                         |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| Copying an array | Create another array with the same length and datatype, and copy all the elements to the new array one by one.<br><br>The memory addresses of both arrays are different and changing one array does not affect the another. | <pre>int arr2 [] = new int [arr.length]; for (int i = 0; i &lt; arr.length; i++){     arr2[i]= arr[i]; } for (int i = 0; i &lt; arr2.length; i++) {     System.out.println(arr2[i]); }</pre> | 55<br>0<br>7<br>-2<br>9 |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|

## 10.5 MULTIDIMENSIONAL ARRAYS

The arrays we have looked at so far were all one-dimensional arrays (values are only in a row). When we want to represent data in a tabular form (rows and columns), we use multidimensional arrays. These are basically arrays of arrays, i.e. an array inside another array. The basic format for creating such arrays is-

**data\_type**[1st dimension][2nd dimension]...[Nth dimension] **array\_name** = new **data\_type**[size1][size2]....[sizeN];

For example:

```
int[][][] newArray = new int[10][20][30]; //This is a multidimensional array.
```

```
int[][] numbers = { {1, 2, 3, 4}, {5, 6, 7} }; //This is also a multidimensional array.
```

To access or modify values of a multidimensional array, we can use array indexing, just like previous examples, but instead of just one index, we use multiple indexes to point to the location of a certain value. For example **numbers[1][2]** represents 7, as [1] means the index of the outer array, and [2] is the index of the inner array.

| Example Code                                                                                                                                                                                                                                                                                                | Output |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| <pre>//To access values. public class Example1 {     public static void main(String[] args) {         int[][] numbers = { {1, 2, 3, 4}, {5, 6, 7} };         System.out.println(numbers[0][1]);     } }</pre>                                                                                               | 2      |
| <pre>//To modify values. public class Example2 {     public static void main(String[] args) {         int[][] numbers = { {1, 2, 3, 4}, {5, 6, 7} };         System.out.println(numbers[0][0]); //Prints 1.         numbers[0][0] = 9;         System.out.println(numbers[0][0]); //Prints 9.     } }</pre> | 1<br>9 |

## 10.6 WORKSHEET

- A. Answer the following theoretical conceptual questions:
1. Multi-Dimensional Arrays: Discuss the concept of multi-dimensional arrays, such as 2D arrays or matrices. Explain how they are implemented and provide examples of their applications.
  2. Array Indexing: Explain the concept of array indexing and how it is used to access elements in an array. Discuss the starting index convention (0-based or 1-based) in different programming languages and its implications.
  3. Array Memory Allocation: Describe how arrays are stored in memory. Discuss the contiguous allocation of elements in memory and the impact of this allocation on array access and performance.
  4. Static vs. Dynamic Arrays: Compare static and dynamic arrays. Explain the difference between fixed-size arrays and arrays with dynamic memory allocation. Discuss the advantages and limitations of each approach.
  5. Array Resizing: Discuss the challenges associated with resizing arrays, especially when they are implemented with static memory allocation. Explain how dynamic arrays address this issue by dynamically resizing the array as needed.
  6. Array Copying and References: Explain how arrays are copied or referenced in different programming languages. Discuss the implications of shallow copying versus deep copying for multidimensional arrays or arrays containing mutable objects.

- B. Given an array of integers, write a program to find the maximum element in the array.

Given Array 1  
 {5, 2, 8, 3, 9}  
 Sample Output 2  
 9



- C. Write a function that takes an array of integers as input and returns the sum of all the elements.

Given Array 1  
 {5, 2, 8, 3, 9}  
 Sample Output 1  
 27

Inspiring Excellence

- D. Write a program that takes an array and a target element as input and prints the index of the target element in the array. If not found in the array, print "Not Found".

Given Array 1  
 {5, 2, 8, 3, 9}  
 8  
 Sample Output 1  
 Target element is at index: 2

Given Array 2  
 {5, 2, 8, 3, 9}  
 18  
 Sample Output 2  
 Not Found

- E. Given an array of integers, write a program that prints the occurrence of each element in the array.

Given Array 1  
 {5, 3, 5}  
 Sample Output 1  
 5 is repeated 2 time(s)  
 3 is repeated 1 time(s)

- F. Given two arrays, write a program that prints an array containing the common elements in both arrays.

Given Arrays

{5, 2, 8, 3, 9}

{6, 3, 5, 11, 10}

Sample Output 1

{3, 5}

- G. Given an array with duplicate elements, write a program that removes the duplicates and prints a new array with unique elements only.

Given Array 1

{5, 2, 8, 3, 2}

Sample Output 1

{5, 2, 8, 3}

- H. Write a program that takes an array as input and returns a new array with the elements reversed.

Given Array 1

{1, 2, 3, 4, 5}

Sample Output 1

{5, 4, 3, 2, 1}



Inspiring Excellence

## CHAPTER 11: ARRAY SORTING

Arranging data in an ordered sequence is called "sorting". This order can be done in two ways: ascending and descending.

- Ascending (or increasing) order: arranged from the smallest to the largest element.
- Descending (or decreasing/non-increasing) order: arranged from the largest to the smallest element.

Java has a built-in `Arrays.sort()` method for sorting arrays. Let's see that first.

| Code                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Output                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| <pre>import java.util.Arrays; // importing Arrays class public class BuiltInSortAscending{      public static void main(String[] args){         int[] arr = { 10, -5, 7, 45, 27 };          // calling the built-in sorting method on the created array         Arrays.sort(arr);          // displaying the elements of the sorted array         for (int elem = 0; elem &lt; arr.length; elem++){             System.out.println(arr[elem]);         }     } }</pre> | <pre>-5 7 10 27 45</pre> |

The above code uses the Java built-in method to sort an array in ascending order. Now, if you just add another step and reverse the sorted array, you will get the sorted array in descending order.

But this is pretty straight forward. As a Computer Science student, you need to know the underlying concepts behind array sorting. You need to understand and visualize different sorting techniques. There are a lot of sorting algorithms used in several domains. In this course, we will learn three such common sorting algorithms.

Inspiring Excellence

### 11.1 SELECTION SORT

The first technique in our list is called Selection Sort. Here, in the ascending order, at first we need to find the minimum value of the given array and swap (exchange) it with the first element of the array. Then, we need to find the second minimum from the rest of the array elements (starting from the 2nd index) and swap it with the 2nd element of the array. Then, we need to find the 3rd minimum from the rest of the array elements (starting from the 3rd index) and swap it with the 3rd element of the array. We need to continue this process until all the elements of the array have moved to their proper position.

At each iteration, the array is partitioned into two sections. **The left section is sorted and the right section is being processed.** Each iteration, we move the partition one step to the right, until the entire array elements have been processed.

Let's consider the following array.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | 40 | 2 | 27 | -7 | 14 |
| index   | 0  | 1 | 2  | 3  | 4  |

Now, let's visualize how Selection Sort works. We will sort the array in ascending order.

**Iteration 1:** At first, we will consider the element in index 0 as the minimum value which is 40 in this example. Then, we will search from index 1 to index (length-1) to find an element less than 40.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | 40 | 2 | 27 | -7 | 14 |
| index   | 0  | 1 | 2  | 3  | 4  |

Here, we have found the lowest possible value less than 40 in index 3 which is -7. We will swap the values of these two indexes.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 27 | 40 | 14 |
| index   | 0  | 1 | 2  | 3  | 4  |

After swapping, we can see that -7 is in the correct sorted position.

**Iteration 2:** Here, we will consider the element in index 1 as the minimum value which is 2. Then, we will search from index 2 to index (length-1) to find an element less than 2.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 27 | 40 | 14 |
| index   | 0  | 1 | 2  | 3  | 4  |

But there are no values which are less than 2 from index 2 to 4; which means 2 is already in the correct sorted position. So, no swapping will be needed here.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 27 | 40 | 14 |
| index   | 0  | 1 | 2  | 3  | 4  |

Position of element 2 will be unchanged here.

**Iteration 3:** You guessed it right. Now, we will look for the value smaller than 27 from index 3 to 4.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 27 | 40 | 14 |
| index   | 0  | 1 | 2  | 3  | 4  |

The value less than 27 is 14 which is in index 4. So, we will swap the values of index 2 and 4.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 14 | 40 | 27 |
| index   | 0  | 1 | 2  | 3  | 4  |

**Iteration 4:** Similarly, the value of index 3 and 4 will be swapped since 27 is less than 40.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 14 | 27 | 40 |
| index   | 0  | 1 | 2  | 3  | 4  |

**Iteration 5:** This step is kind of useless and unnecessary. Because all the remaining elements are sorted from left to right. So, the element in the last index of the array will definitely be in the correct sorted position. So, the position of 40 will be unchanged.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 14 | 27 | 40 |
| index   | 0  | 1 | 2  | 3  | 4  |

You can see our array is now successfully sorted in the ascending order using Selection Sort. For descending order sort, just find the maximum value instead of the minimum value on each iteration.

Here, for each iteration, we only consider the right section of the partition for finding the minimum value as the left side is already sorted. Lastly, in the last step, only one element remains which is the minimum. So sorting is unnecessary in that case. So, for Selection Sort, (length-1) times iteration is needed.

Now, let's look at the code.

| Code                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Output                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|
| <pre> public class SelectionSortAscending{      public static void main(String[] args){         int[] arr = { 40, 2, 27, -7, 14 };          // implementing Selection Sort on the created array          for (int i = 0; i &lt; arr.length-1; i++){             int min_idx = i;             // finding out the index containing minimum value from i+1 to last index             for (int j = i+1; j &lt; arr.length; j++){                 if (arr[j] &lt; arr[min_idx]){                     min_idx = j;                 }             }             // swapping the minimum index value with the current index value             int temp = arr[min_idx];             arr[min_idx] = arr[i];             arr[i] = temp;         }          // displaying the elements of the sorted array         for (int idx = 0; idx &lt; arr.length; idx++){             System.out.println(arr[idx]);         }     } } </pre> | <pre> -7 2 14 27 40 </pre> |

Inspiring Excellence

## 11.2 BUBBLE SORT

Now, we will learn the Bubble Sort technique. For Bubble sort, in each iteration, adjacent elements are compared. If the adjacent elements are in the wrong order, then they are swapped. It has been named "Bubble sort" because of the way smaller or larger elements "bubble" to the top (Left side) of the array. As the number of iterations increases, the number of comparisons decreases.

Let's consider the same array as before.

| element | 40 | 2 | 27 | -7 | 14 |
|---------|----|---|----|----|----|
| index   | 0  | 1 | 2  | 3  | 4  |

Let's visualize how this algorithm works. Here, we will sort the array in ascending order too.

**Iteration 1:** We will compare between two adjacent index elements here. If the first element is greater than the second element, then we will swap between two index values. For example, we will compare the index 0 value with index 1 value, which means we will compare between 40 and 2. Since, 2 is less than 40, so these two values will be swapped. Now, 40 is in index 1. We will compare 40 with the index 2 element which is 27. Again, these two values will be swapped and 40 will move to index 3. Similarly, we need to do the same process for the entire array in this iteration.

|         |    |   |    |    |    |                       |
|---------|----|---|----|----|----|-----------------------|
| element | 40 | 2 | 27 | -7 | 14 | 40 > 2<br>swap places |
| index   | 0  | 1 | 2  | 3  | 4  |                       |

|         |   |    |    |    |    |                        |
|---------|---|----|----|----|----|------------------------|
| element | 2 | 40 | 27 | -7 | 14 | 40 > 27<br>swap places |
| index   | 0 | 1  | 2  | 3  | 4  |                        |

|         |   |    |    |    |    |                        |
|---------|---|----|----|----|----|------------------------|
| element | 2 | 27 | 40 | -7 | 14 | 40 > -7<br>swap places |
| index   | 0 | 1  | 2  | 3  | 4  |                        |

|         |   |    |    |    |    |                        |
|---------|---|----|----|----|----|------------------------|
| element | 2 | 27 | -7 | 40 | 14 | 40 > 14<br>swap places |
| index   | 0 | 1  | 2  | 3  | 4  |                        |

After these 4 comparisons, the array looks like this:

|         |   |    |    |    |    |
|---------|---|----|----|----|----|
| element | 2 | 27 | -7 | 14 | 40 |
| index   | 0 | 1  | 2  | 3  | 4  |

You can see after iteration 1, the largest element 40 moved to the last index position. So, 40 is now in the sorted position and we do not need to check for 40 again in the upcoming iterations.

**Iteration 2:** We will follow the similar procedure for the remaining unsorted array elements.

|         |   |    |    |    |    |                   |
|---------|---|----|----|----|----|-------------------|
| element | 2 | 27 | -7 | 14 | 40 | 2 < 27<br>no swap |
| index   | 0 | 1  | 2  | 3  | 4  |                   |

|         |   |    |    |    |    |                        |
|---------|---|----|----|----|----|------------------------|
| element | 2 | 27 | -7 | 14 | 40 | 27 > -7<br>swap places |
| index   | 0 | 1  | 2  | 3  | 4  |                        |

|         |   |    |    |    |    |                        |
|---------|---|----|----|----|----|------------------------|
| element | 2 | -7 | 27 | 14 | 40 | 27 > 14<br>swap places |
| index   | 0 | 1  | 2  | 3  | 4  |                        |

After these 3 comparisons, the array looks like below:

|         |   |    |    |    |    |
|---------|---|----|----|----|----|
| element | 2 | -7 | 14 | 27 | 40 |
| index   | 0 | 1  | 2  | 3  | 4  |

**Iteration 3:** You get the idea now. We will now perform 2 comparisons.

|         |   |    |    |    |    |                       |
|---------|---|----|----|----|----|-----------------------|
| element | 2 | -7 | 14 | 27 | 40 | 2 > -7<br>swap places |
| index   | 0 | 1  | 2  | 3  | 4  |                       |

|         |    |   |    |    |    |                   |
|---------|----|---|----|----|----|-------------------|
| element | -7 | 2 | 14 | 27 | 40 | 2 < 14<br>no swap |
| index   | 0  | 1 | 2  | 3  | 4  |                   |

After the iteration, now the array looks like below:

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 14 | 27 | 40 |
| index   | 0  | 1 | 2  | 3  | 4  |

We can see the array is already sorted. But the algorithm will run for another iteration.

**Iteration 4:** The algorithm will compare between -7 and 2. But there will be no swap because -7 is less than 2.

|         |    |   |    |    |    |                   |
|---------|----|---|----|----|----|-------------------|
| element | -7 | 2 | 14 | 27 | 40 | 2 < 14<br>no swap |
| index   | 0  | 1 | 2  | 3  | 4  |                   |

So, -7 and 2 are in the correct sorted position. So, after the 4th iteration, we get the sorted array.

|         |    |   |    |    |    |
|---------|----|---|----|----|----|
| element | -7 | 2 | 14 | 27 | 40 |
| index   | 0  | 1 | 2  | 3  | 4  |

Similar to Selection Sort, for Bubble Sort, (length-1) times iteration is needed. Another thing to notice is that here in each iteration, the number of comparisons is reduced by one. For example, in our example, in the 1st iteration, there were 4 comparisons. In the 2nd iteration, there were 3 comparisons and so on.

For the descending order sort, we just need to compare the two adjacent index values and if the later element is greater than its immediate previous element, then the two array elements will be swapped.

Now, let's look at the code.



| Code                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Output                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|
| <pre> public class BubbleSortAscending{      public static void main(String[] args){         int[] arr = { 40, 2, 27, -7, 14 };          // implementing Bubble Sort on the created array          for (int i = 0; i &lt; arr.length-1; i++){             // comparing between the two adjacent index elements in a loop             for (int j = 0; j &lt; arr.length-i-1; j++){                 if (arr[j+1] &lt; arr[j]){                     // swapping the values                     int temp = arr[j];                     arr[j] = arr[j+1];                     arr[j+1] = temp;                 }             }         }          // displaying the elements of the sorted array         for (int idx = 0; idx &lt; arr.length; idx++){             System.out.println(arr[idx]);         }     } } </pre> | <pre> -7 2 14 27 40 </pre> |

### 11.3 INSERTION SORT

The last sorting technique we will learn in this chapter is called Insertion Sort. It has a similarity with sorting playing cards by hand. The first card in the card game is expected to be sorted, and afterwards we choose an unsorted card. If the unsorted card chosen is less than the first, it will be placed on the left side; otherwise, it will be placed on the right side. Similarly, all unsorted cards are taken and placed exactly where they belong. So, in this technique, the array effectively breaks down itself into two sections: sorted and unsorted. We select a value from the unsorted section one by one and place that value in the correct position in the sorted section.

Let's visualize the working procedures of this technique using the same array used before.

| element | 40 | 2 | 27 | -7 | 14 |
|---------|----|---|----|----|----|
| index   | 0  | 1 | 2  | 3  | 4  |

Initially we assume that the first index element is already sorted. So, as mentioned earlier, the array can be split into two parts like below:

|         | <i>sorted</i> | <i>unsorted</i> |    |    |    |
|---------|---------------|-----------------|----|----|----|
| element | 40            | 2               | 27 | -7 | 14 |
| index   | 0             | 1               | 2  | 3  | 4  |

**Iteration 1:** We consider the value of index 1 as key. So, key = 2 in this case. Here, key means the value of the current array element which will be inserted in the correct position in the sorted section. Here, the value of index 0 is 40, which is greater than the key value 2. So, 40 will be moved to index 1 and the value of key will be inserted to index 0.

|         | sorted | unsorted |    |    |    | key = 2                    |
|---------|--------|----------|----|----|----|----------------------------|
| element | 40     | 2        | 27 | -7 | 14 | 40 > key<br>shift 40 right |
| index   | 0      | 1        | 2  | 3  | 4  |                            |

|         |    |    |    |    |    | key = 2                  |
|---------|----|----|----|----|----|--------------------------|
| element | 40 | 40 | 27 | -7 | 14 | insert key in<br>index 0 |
| index   | 0  | 1  | 2  | 3  | 4  |                          |

After inserting the key in index 0, the array now looks like this.

|         | sorted |    | unsorted |    |    |
|---------|--------|----|----------|----|----|
| element | 2      | 40 | 27       | -7 | 14 |
| index   | 0      | 1  | 2        | 3  | 4  |

**Iteration 2:** The value of key will be the value of index 2 now. So, key = 27. We need to follow the same procedure as before.

|         | sorted |    | unsorted |    |    | key = 27                   |
|---------|--------|----|----------|----|----|----------------------------|
| element | 2      | 40 | 27       | -7 | 14 | 40 > key<br>shift 40 right |
| index   | 0      | 1  | 2        | 3  | 4  |                            |

|         |   |    |    |    |    | key = 27                            |
|---------|---|----|----|----|----|-------------------------------------|
| element | 2 | 40 | 40 | -7 | 14 | 2 < key<br>insert key in<br>index 1 |
| index   | 0 | 1  | 2  | 3  | 4  |                                     |

After inserting the key in index 1, the array looks like below:

|         | sorted |    |    | unsorted |    |
|---------|--------|----|----|----------|----|
| element | 2      | 27 | 40 | -7       | 14 |
| index   | 0      | 1  | 2  | 3        | 4  |

**Iteration 3:** The value of the key will be -7 now. So, key = -7.

|         | sorted |    |    | unsorted |    | key = -7                   |
|---------|--------|----|----|----------|----|----------------------------|
| element | 2      | 27 | 40 | -7       | 14 | 40 > key<br>shift 40 right |
| index   | 0      | 1  | 2  | 3        | 4  |                            |
|         |        |    |    |          |    | key = -7                   |
| element | 2      | 27 | 40 | 40       | 14 | 27 > key<br>shift 27 right |

|         |    |    |    |    |    |                          |
|---------|----|----|----|----|----|--------------------------|
| index   | 0  | 1  | 2  | 3  | 4  |                          |
|         |    |    |    |    |    | key = -7                 |
| element | 2  | 27 | 27 | 40 | 14 | 2 > key<br>shift 2 right |
| index   | 0  | 1  | 2  | 3  | 4  |                          |
|         |    |    |    |    |    | key = -7                 |
| element | -7 | 2  | 27 | 40 | 14 | insert key in<br>index 0 |
| index   | 0  | 1  | 2  | 3  | 4  |                          |

After inserting the key, the array will look like this:

|         |               |   |    |    |                 |
|---------|---------------|---|----|----|-----------------|
|         | <i>sorted</i> |   |    |    | <i>unsorted</i> |
| element | -7            | 2 | 27 | 40 | 14              |
| index   | 0             | 1 | 2  | 3  | 4               |

**Iteration 4:** We now have to insert 14 in the correct position in this step. Here, key = 14.

|         |               |   |    |    |                 |                            |
|---------|---------------|---|----|----|-----------------|----------------------------|
|         | <i>sorted</i> |   |    |    | <i>unsorted</i> | key = 14                   |
| element | -7            | 2 | 27 | 40 | 14              | 40 > key<br>shift 40 right |
| index   | 0             | 1 | 2  | 3  | 4               |                            |
|         |               |   |    |    |                 | key = 14                   |
| element | -7            | 2 | 27 | 40 | 40              | 27 > key<br>shift 27 right |
| index   | 0             | 1 | 2  | 3  | 4               |                            |

|         |    |   |    |    |    |                                     |
|---------|----|---|----|----|----|-------------------------------------|
|         |    |   |    |    |    | key = 14                            |
| element | -7 | 2 | 27 | 27 | 40 | 2 < key<br>insert key in<br>index 2 |
| index   | 0  | 1 | 2  | 3  | 4  |                                     |

After inserting the key in index 2, we now have the full array sorted.

|         |               |   |    |    |    |
|---------|---------------|---|----|----|----|
|         | <i>sorted</i> |   |    |    |    |
| element | -7            | 2 | 14 | 27 | 40 |
| index   | 0             | 1 | 2  | 3  | 4  |

That's how the Insertion sort algorithm works. Here, we initially assume the index 0 value is already in the sorted position. So, we look from index 1 to the last index of the array. So, (length-1) times iteration is needed here too. We consider each index value as the key and insert that key into the appropriate position in the sorted part of the array.

For the descending order sort, we will shift the current index value if it is less than the key and insert the key in the appropriate position in each step afterwards.

Now, let us look into the code.

| Code                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Output                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|
| <pre> public class InsertionSortAscending{      public static void main(String[] args){         int[] arr = { 40, 2, 27, -7, 14 };          // implementing Insertion Sort on the created array          for (int i = 1; i &lt; arr.length; i++){             int key = arr[i]; // the element which will be inserted             int j = i - 1;             /* shift elements of index 0 to (i-1) which are greater             than key to one index forward of the current index */             while (j &gt;= 0 &amp;&amp; arr[j] &gt; key){                 arr[j + 1] = arr[j];                 j = j - 1;             }             arr[j + 1] = key; // inserting the key in the suitable place         }          // displaying the elements of the sorted array         for (int idx = 0; idx &lt; arr.length; idx++){             System.out.println(arr[idx]);         }     } } </pre> | <pre> -7 2 14 27 40 </pre> |

## 11.4 WORKSHEET

- A. Using Java built-in Arrays.sort() method, sort an array in descending order. You cannot use any other built-in method to complete this task. [Hint: Modify the ascending order sort code and add another step using a loop.]
- B. Implement the descending order sort for the following sorting techniques:
  1. Selection Sort,
  2. Bubble Sort,
  3. Insertion Sort.
- A. We have learnt that Bubble sort needs (length - 1) times iteration. But sometimes even before completing all the iterations, the array might get sorted and the code runs for some extra redundant iterations. In that case, modify your ascending/descending order Bubble sort code so that it can handle this issue and stop immediately without running for these redundant iterations. [Hint: Utilize the concept of Flag variable.]
- B. Let's say you have an array of n elements. You only need the first k sorted elements (where  $k < n$ ) of the array. Which sorting algorithm will be efficient in this case? Share your opinion briefly.
- C. Suppose, you have an array whose elements are nearly sorted (for example, 90% of the elements are already in the correct sorting position). To sort the remaining values faster, which sorting technique will you use? Explain shortly.
- D. We have seen in both Selection Sort and Bubble Sort; we need to perform swapping. If we want to minimize the overall swap operations, which sorting technique would be better? Explain briefly.
- E. Write a Java program that takes a number T as user input where T denotes the total number of students. Then the user needs to take the name and corresponding marks of T students as input. Display the names of the students based on their marks from highest to lowest.

| Sample Input                                                                                                                                                                                                                                                                                       | Sample Output                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|
| 5<br>Clark<br>56<br>Bruce<br>51<br>Diana<br>76<br>Barry<br>91<br>Lois<br>63                                                                                                                                                                                                                        | Barry<br>Diana<br>Lois<br>Clark<br>Bruce |
| <b>Explanation:</b><br>In this sample example, T=5. So, the user needs to enter 5 students' names and marks one by one as input. From the sample, Clark got 56, Bruce got 51, and so on. In the output, the student names are displayed based on the marks they have received in descending order. |                                          |