Java Programming Lab
ETIT 353

Faculty: **Dr. Sandeep Tayal**

Name: **Syeda Reeha Quasar**

Roll No.: **14114802719**

Semester: **5**

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

VISION

To Produce "Critical thinkers of Innovative Technology"

MISSION

To provide an excellent learning environment across the computer science discipline to inculcate professional behavior, strong ethical values, innovative research capabilities and leadership abilities which enable them to become successful entrepreneurs in this globalized world.

1. To nurture an **excellent learning environment** that helps students to enhance their problem solving skills and to prepare students to be lifelong learners by offering a solid theoretical foundation with applied computing experiences and educating them about their **professional, and ethical responsibilities**.
2. To establish **Industry-Institute Interaction**, making students ready for the industrial environment and be successful in their professional lives.
3. To promote **research activities** in the emerging areas of technology convergence.
4. To build engineers who can look into technical aspects of an engineering solution thereby setting a ground for producing successful **entrepreneur.**

VISION

To nurture young minds in a learning environment of high academic value and im-bibe spiritual and ethical values with technological and management competence.

MISSION

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

Engineering Hardware - Software Symbiosis

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper un- derstanding of basic concepts and encouraging inquisitive nature.

Life - Long Learning

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

Liberalization and Globalization

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.
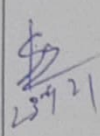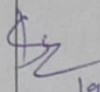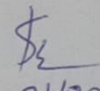
Diversification

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above at- tributes by encouraging analytical thinking.

Digitization of Learning Processes

The Institute provides seamless opportunities for innovative learning in all En- gineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics, tutorials and related tools to create a platform for multi-disciplinary approach.

INDEX

Sr. no.	Topics Covered	Experiments	Date	R1	R2	R3	R4	R5	Total Marks	Signature
1.	Basic concept of Java programming, Compilation and Execution Process, Data Types, operators, Reading user input, Strings	1. Write a program to print "Hello World" on the screen. 2. WAP that convert string to character using toString() and valueOf() . 3. WAP that convert Char to string. 4. Program to find ASCII code of a character. 5. Swapping two numbers using bitwise operator.	16-09-21						15	23-9-21
2.	Java Control statements, Command line arguments	1. WAP to check Vowel or Consonant using Switch Case 2. WAP to display first n prime numbers. 3. WAP to check whether the input year is leap or not 4. Write an application that accepts two doubles as its command line arguments, multiple these together and display the product. 5. Write an application that accepts one command line argument, display the line of reporting if number is even or odd. 6. Write an application that accepts radius of a circle as its command line argument display the area.	23-09-21						1)	30/9/21
3.	Arrays, Methods, Method Overloading	1. Write a program to find out the array index or position where sum of numbers preceding the index is equals to sum of numbers succeeding the index. 2. Write a program that creates and initializes a four-element int array. Calculate and display the average of its values. 3. WAP using Bubble sort for sorting in ascending Order.	30-09-21						15	7/10/21

INDEX

INDEX

		polymorphism and interfaces. 3. Write an application that creates an 'interface' and implements it. 4. Write an application to illustrate Interface Inheritance.										
7.	Exception Handling, Applet	1. Write an application that shows how to create a user-defined exception. 2. Create a customized exception and also make use of all the 5 exception keywords. 3. Write an Applet that displays "Hello World" (Background color-black, text color-blue and your name in the status window.) 4. Develop an analog clock using applet.	04-11-21							15		1/1/24
8.	Multithreading	1. Write a java program to show multithreaded producer and consumer application. 2. Write an application that executes two threads. One thread displays "An" every 1000 milliseconds and other displays "B" every 3000 milliseconds. Create the threads by extending the Thread class.	11-11-21							1)		1/1/21
9.	AWT Components Event Handling	1. WAP that illustrates how to process mouse click, enter, exit, press and release events. The background color changes when the mouse is entered, clicked, pressed, released or exited. 2. WAP that displays your name whenever the mouse is clicked.	18-11-21							1)		20/11/21
10.	File Handling, JDBC	1. Write a program that read from a file and write to file.	25-11-21									

INDEX

		2. Convert the content of a given file into the uppercase content of the same file. 3. JDBC (Database connectivity with MS-Access).									✓ $\frac{\$_2}{2/1/21}$
11.	Swings	1. Create runnable jar file in java. 2. Display image on a button in swing 3. Change the component color by choosing a color from Color Chooser. 4. Display the digital watch in swing tutorial. 5. Create a notepad in swing.	02-12-21								1) $\frac{\$_2}{9/1/21}$
12.	Servlet	1. Write hello world program in servlet. 2. Write a servlet which displays current system date and time? 3. Write a program that handle HTTP request. 4. Write a program that handle HTTP response 5. Create a servlet that uses Cookies to store the number of times a user has visited your servlet.	09-12-21								15 $\frac{\$_2}{9/12/21}$

LAB - 1

Java Programming Lab

Topics Covered

Basic concept of Java programming, Compilation and Execution Process,
Data Types, operators, Reading user input, Strings

Syeda Reeha Quasar

14114802719
4C7

What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.

2. Web Applications such as irctc.co.in, javatpoint.com, etc.

3. Enterprise Applications such as banking applications.

4. Mobile

5. Embedded System

6. Smart Card

7. Robotics

8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine.

Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

EXPERIMENT – 1.1

Aim:
Write a program to print "Hello World" on the screen.

Theory:
For executing any Java program, the following software or application must be properly installed.

➢ Install the JDK if you don't have installed it, download the JDK and install it.

➢ Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java

➢ Create the Java program

➢ Compile and run the Java program
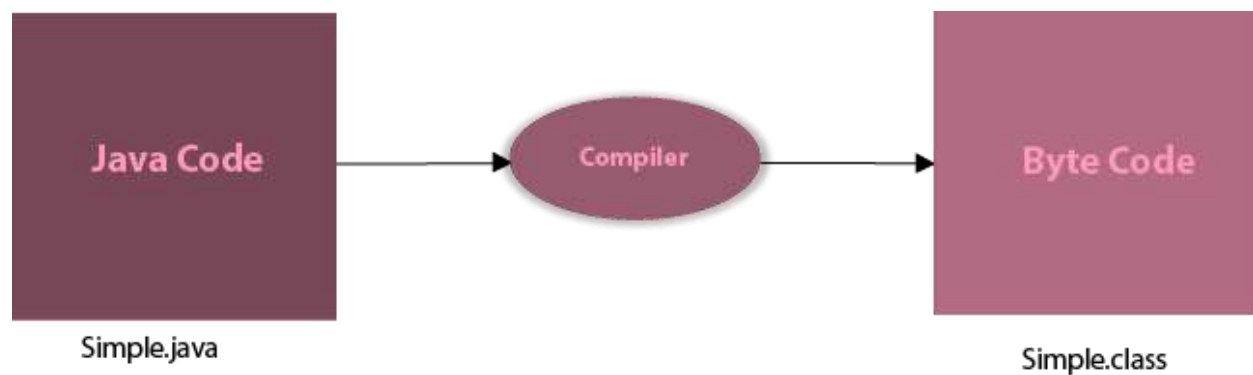
The process of Java programming can be simplified in three steps:

• Create the program by typing it into a text editor and saving it to a file – HelloWorld.java.

• Compile it by typing "javac HelloWorld.java" in the terminal window.

• Execute (or run) it by typing "java HelloWorld" in the terminal window.

Concepts:

➢ **class** keyword is used to declare a class in Java.

- ➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

- ➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

- ➢ **void** is the return type of the method. It means it doesn't return any value.

- ➢ **main** represents the starting point of the program.

- ➢ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

- ➢ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

Source Code:

```java
public class FirstProgram {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Output:

PS E:\sem 5\java> cd "e:\sem 5\java\" ; if ($?) { javac FirstProgram.java } ; if ($?) { java FirstProgram }
Hello World!

Hello World!

EXPERIMENT – 1.2

Aim:
WAP that convert string to character using toString() and valueOf() .

Theory:

Given a string, the task is to convert this string into a character array in Java.
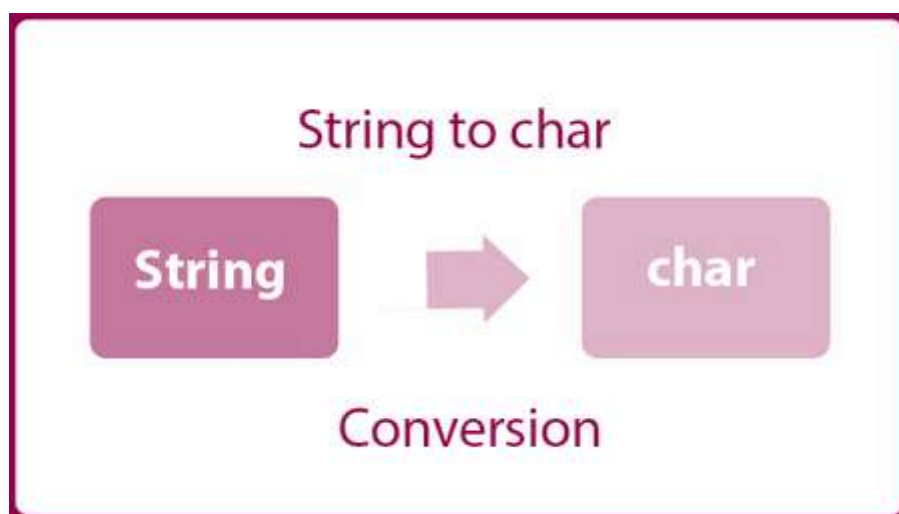
Examples:

Input: Hello World

Output: [H, e, l, l, o,, W, o, r, l, d]

Method : Naive Approach

- Step 1: Get the string.

- Step 2: Create a character array of the same length as of string.

- Step 3: Traverse over the string to copy character at the i'th index of string to i'th index in the array.

- Step 4: Return or perform the operation on the character array.

Concepts:

➢ **class** keyword is used to declare a class in Java.

➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

➢ **void** is the return type of the method. It means it doesn't return any value.

➢ **main** represents the starting point of the program.

➢ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

➢ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➢ **charAt()** method returns a single character of specified index. The signature of charAt() method is given below:
public char charAt(int index)

Source Code:

```java
public class stringToChar {
    public static void main(String args[]) {
        String s = "hello World!";
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            System.out.println("char at " + i + " index is: " + c);
        }
    }
}
```

Output:

```
PS E:\sem 5\java> cd "e:\sem 5\java\" ; if ($?) { javac stringToChar.java } ; if ($?) { java stringToChar }
char at 0 index is: h
char at 1 index is: e
char at 2 index is: l
char at 3 index is: l
char at 4 index is: o
char at 5 index is:
char at 6 index is: W
char at 7 index is: o
char at 8 index is: r
char at 9 index is: l
char at 10 index is: d
char at 11 index is: !
```

Source Code:

```java
public class stringToChar1 {
    public static void main(String args[]) {
        String s1 = "hello";
        char[] ch = s1.toCharArray();
        for (int i = 0; i < ch.length; i++) {
            System.out.println("char at " + i + " index is: " + ch[i]);
        }
    }
}
```

Output:

```
PS E:\sem 5\java> cd "e:\sem 5\java\" ; if ($?) { javac stringToChar1.java } ; if ($?) { java stringToChar1
}
char at 0 index is: h
char at 1 index is: e
char at 2 index is: l
char at 3 index is: l
char at 4 index is: o
```

EXPERIMENT – 1.3

Aim:

WAP that convert Char to string.

Theory:

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character "\0". A character array can be converted to a string and vice versa.

Methods:

1. Using copyOf() method of Arrays class

2. Using StringBuilder class

3. Using valueOf() method of String class

4. Using copyValueOf() method of String class

5. Using Collectors in Streams

Using copyOf() method of Array class

The given character can be passed into the String constructor. By default, the character array contents are copied using the Arrays.copyOf() method present in the Arrays class.

Using StringBuilder class

StringBuilder is a mutable class, therefore, the idea is to iterate through the character array and append each character at the end of the string. Finally, the string contains the string form of the characters.

Using valueOf() method of String class

This method inherently converts the character array to a format where the entire value of the characters present in the array is displayed. This method generally converts int, float, double, char, boolean, and even object to a string. Here we will achieve the goal by converting our character array to string.

Using copyValueOf() method of String class

The contents from the character array are copied subsequently modified without affecting the string to be returned, hence this method also enables us to convert the character array to string.

Using Collectors in Streams

With the introduction of streams in java8, we straight away use Collectors in streams to modify our character input array elements and later uses joining() method and return a single string and print it.

Source Code:

```java
public class charToString {
    public static void main(String args[]) {
        char ch = 'a';
        String str1 = Character.toString(ch);
        System.out.println("String after conversion is: " + str1);
    }
}
```

Output:

PS E:\sem 5\java> cd "e:\sem 5\java\" ; if ($?) { javac charToString.java } ; if ($?) { java charToString }
String after conversion is: a

String after conversion is: a

Source Code:

```
public class charToString1 {
    public static void main(String args[]) {
        char ch = 'a';
        String str = String.valueOf(ch);
        System.out.println("String after conversion is: " + str);
    }
}
```

Output:

```
PS E:\sem 5\java> java charToString1
String after conversion is: a

String after conversion is: a
```

EXPERIMENT – 1.4

Aim:

Program to find ASCII code of a character.

Theory:

ASCII acronym for American Standard Code for Information Interchange. It is a 7-bit character set contains 128 (0 to 127) characters. It represents the numerical value of a character. For example, the ASCII value of A is 65.

There are two ways to print ASCII value in Java:

- Assigning a Variable to the int Variable

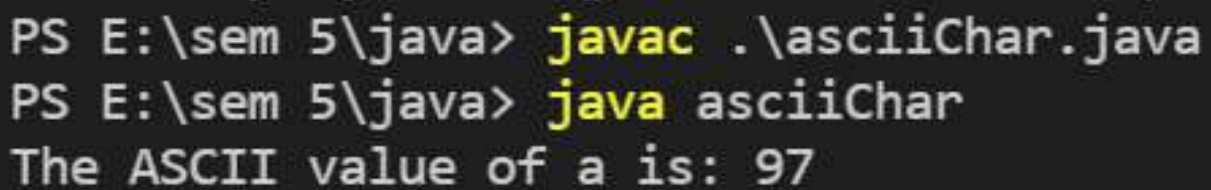- Using Type-Casting

Assigning a Variable to the int Variable

To print the ASCII value of a character, we need not use any method or class. Java internally converts the character value to an ASCII value.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174		
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

© w3resource.com

Source Code:

```java
public class asciiChar {
    public static void main(String[] args) {
        char ch1 = 'a';
        int asciivalue = ch1; //type cast to int
        System.out.println("The ASCII value of " + ch1 + " is: " + asciivalue);
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\asciiChar.java
PS E:\sem 5\java> java asciiChar
The ASCII value of a is: 97
```

```
The ASCII value of a is: 97
```

Source Code:

```java
public class asciiChar1 {
    public static void main(String[] String) {
        int ch1 = 'a';
        System.out.println("The ASCII value of a is: " + ch1);
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\asciiChar1.java
PS E:\sem 5\java> java asciiChar1
The ASCII value of a is: 97
```

```
The ASCII value of a is: 97
```

EXPERIMENT – 1.5

Aim:
Swapping two numbers using bitwise operators.

Theory:

In Java, there are many ways to swap two numbers. Generally, we use either swap() method of the Math class or use a third (temporary) variable to swap two numbers. Except these two ways, we can also swap two numbers using the bitwise operator (XOR) and using division and multiplication.

Using Bitwise Operator

Bitwise Operator: Bitwise XOR operator is used to swap two numbers. It is represented by the symbol (^). It compares bits of two operands and returns false or 0 if they are equal and returns true or 1 if they are not equal. The truth table of XOR operator is as follows:

X	Y	X ^ Y
0	0	0
0	1	1
1	0	1
1	1	0

We can use the bitwise XOR operator to swap two numbers without using the swap() method and third variable. We must follow the steps given below:

- Find the binary equivalent of given variables, say X and Y.

- Find X^Y and store it in x, i.e. X = X ^ Y.

- Again, find X^Y and store it in Y, i.e. Y = X ^ Y.

- Find X^Y and store it in X, i.e. X = X ^ Y.

- The numbers are swapped.

Now implement the above steps in an example and understand the swapping.

Example:

Swap the variables X = 5 and Y = 9 using the bitwise operator.

Solution:

Step 1: Binary equivalent of the variables X and Y are:

X = 5 = 0101 and Y = 9 = 1001

Step 2: Find X = X ^ Y.

X	0	1	0	1
Y	1	0	0	1
X = X ^ Y	1	1	0	0

Step 2: Find Y = X ^ Y.

X	1	1	0	0
Y	1	0	0	1
Y = X ^ Y	0	1	0	1

Step 3: Find X = X ^ Y.

X	1	1	0	0
Y	0	1	0	1
X = X ^ Y	1	0	0	1

We see that the variable X contains 1001 which is equivalent to 9 and Y contains 0101 which is equivalent to 5. Therefore, the variables X and Y are swapped.
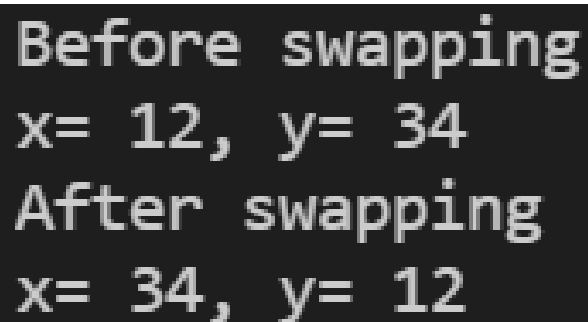

X = 9 and Y = 5

Source Code:

```java
public class swapViaBitwise {
    static void swapNumbers(int x, int y) {
        System.out.println("Before swapping");
        System.out.println("x= " + x + ", y= " + y);
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
        System.out.println("After swapping");
        System.out.println("x= " + x + ", y= " + y);
    }

    public static void main(String[] args) {
        int x = 12;
        int y = 34;
        swapNumbers(x, y);
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\swapViaBitwise.java
PS E:\sem 5\java> java swapViaBitwise
Before swapping
x= 12, y= 34
After swapping
x= 34, y= 12
```

```
Before swapping
x= 12, y= 34
After swapping
x= 34, y= 12
```
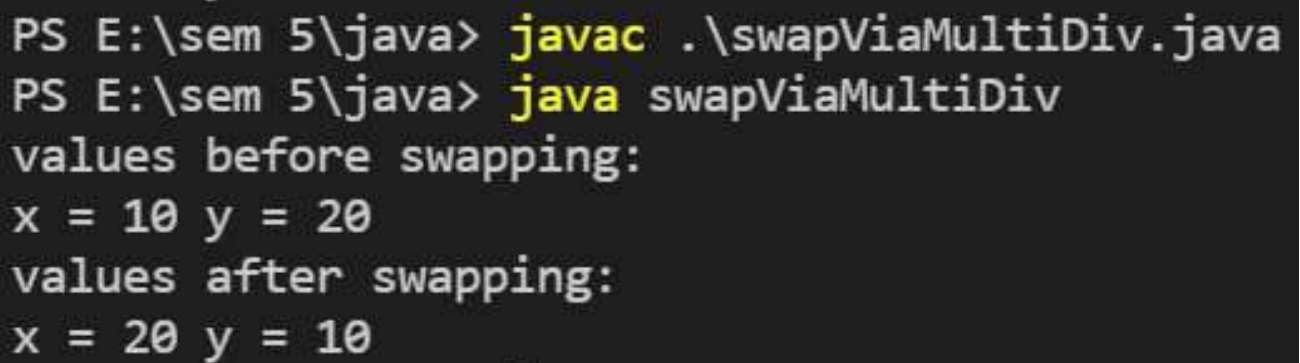
Swap – Multiplication Division

Source Code:

```
public class swapViaMultiDiv {
    public static void main(String args[]) {
        int x = 10;
        int y = 20;
        System.out.println("values before swapping:");
        System.out.println("x = " + x + " y = " + y);
        x = x * y;
        y = x / y;
        x = x / y;
        System.out.println("values after swapping:");
        System.out.println("x = " + x + " y = " + y);
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\swapViaMultiDiv.java
PS E:\sem 5\java> java swapViaMultiDiv
values before swapping:
x = 10 y = 20
values after swapping:
x = 20 y = 10
```

Viva Questions

1. What is the most important feature of Java?

Ans.

Java is a platform independent language.

2. What do you mean by platform independence?

Ans.

Platform independence means that we can write and compile the java code in one platform (e.g. Windows) and can execute the class in any other supported platform e.g. (Linux, Solaris, etc.).

3. What is JVM?

Ans.

JVM is Java Virtual Machine which is a run time environment for the compiled java class files.

4. Are JVM's platform independent?

Ans.

JVM's are not platform independent. JVM's are platform specific run time implementation provided by the vendor.

5. What is the difference between a JDK and a JVM?

Ans.

JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.

5. What is the difference between a JDK and a JVM?

LAB - 2

Java Programming Lab

Topics Covered
Java Control statements, Command line arguments

Syeda Reeha Quasar
14114802719
4C7

EXPERIMENT – 2.1

Aim:
WAP to check Vowel or Consonant using Switch Case.

Theory:

The alphabets A, E, I, O and U (smallcase and uppercase) are known as Vowels and rest of the alphabets are known as consonants. Here we will write a java program that checks whether the input character is vowel or Consonant using **Switch Case in Java**.

Example: Program to check Vowel or Consonant using Switch Case

In this program we are not using break statement with cases intentionally, so that if user enters any vowel, the program continues to execute all the subsequent cases until Case 'U' is reached and thats where we are setting up the value of a boolean variable to true. This way we can identify that the alphabet entered by user is vowel or not.

Algorithm:

1. Enter and Store User Input.

2. Duplicate the input and store it.

3. Change the Case of the duplicate for lesser comparison and uniformity.

4. Pass the input through Switch-Case construct.

5. Check for Vowel.

Print the Result accordingly.

Concepts:

- ➢ **class** keyword is used to declare a class in Java.

- ➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

- ➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to

create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

➤ **void** is the return type of the method. It means it doesn't return any value.

➤ **main** represents the starting point of the program.

➤ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

➤ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➤ **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

➤ util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

➤ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

➤ A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.

➤ The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

Source Code:

```java
import java.util.Scanner;

public class CheckVowelSwitchcase {
    public static void main(String args[]) {
        char ch;// variable declaration

        Scanner scan = new Scanner(System.in);
        // create a scanner object for input

        System.out.println("Enter the Alphabet for check vowel or consonant  ");
        ch = scan.next().charAt(0);
        ;// store the input from the user

        switch (ch) {
            // check lower case vowel letters
            case 'a':
                System.out.println(ch + " is a vowel");
                break;

            case 'e':
                System.out.println(ch + " is a vowel");
                break;

            case 'i':
                System.out.println(ch + " is a vowel");
                break;

            case 'o':
                System.out.println(ch + " is a vowel");
                break;

            case 'u':
                System.out.println(ch + " is a vowel");
                break;

            // check upper case vowel letters
            case 'A':
                System.out.println(ch + " is a vowel");
                break;

            case 'E':
                System.out.println(ch + " is a vowel");
```

```java
                break;

        case 'I':
            System.out.println(ch + " is a vowel");
            break;

        case 'O':
            System.out.println(ch + " is a vowel");
            break;

        case 'U':
            System.out.println(ch + " is a vowel");
            break;

        default:
            System.out.println(ch + " is a consonant");
            break;

        }
    }

}
```

Output:

```
PS E:\sem 5\java> javac .\CheckVowelSwitchcase.java
PS E:\sem 5\java> java CheckVowelSwitchcase
Enter the Alphabet for check vowel or consonant
r
r is a consonant
```

```
Enter the Alphabet for check vowel or consonant
r
r is a consonant
```

EXPERIMENT – 2.2

Aim:
WAP to display first n prime numbers.

Theory:

We have taken a while loop which acts like a counter means as soon as we get a prime number the i value will be incremented. Then we have taken a for loop for taking every number. The if condition is done so as to make sure that if the no is prime or not. Flag variable is used which signals us that if number taken is prime or not.

Concepts:

- ➢ **class** keyword is used to declare a class in Java.

- ➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

- ➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

- ➢ **void** is the return type of the method. It means it doesn't return any value.

- ➢ **main** represents the starting point of the program.

- ➢ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

- ➢ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

- ➢ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

Source Code:

```java
import java.util.*;

class primeN {
      public static void main(String args[]) {
            Scanner sc = new Scanner(System.in);
            int i, n, p, count, flag;

            System.out.println("Enter the number of prime terms you want!");
            n = sc.nextInt();
            System.out.println("First " + n + " prime numbers are :-");

            p = 2;
            i = 1;
            while (i <= n) {
                  flag = 1;
                  for (count = 2; count <= p - 1; count++) {
                        if (p % count == 0) // Will be true if p is not prime
                        {
                              flag = 0;
                              break; // Loop will terminate if p is not prime
                        }
                  }
                  if (flag == 1) {
                        System.out.print(p + " ");
                        i++;
                  }
                  p++;
            }
      }
}
```

Output:

```
PS E:\sem 5\java> javac .\primeN.java
PS E:\sem 5\java> java primeN
Enter the number of prime terms you want!
2
First 2 prime numbers are :-
2 3
PS E:\sem 5\java> java primeN
Enter the number of prime terms you want!
10
First 10 prime numbers are :-
2 3 5 7 11 13 17 19 23 29
```

```
PS E:\sem 5\java> java primeN
Enter the number of prime terms you want!
20
First 20 prime numbers are :-
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
```

EXPERIMENT – 2.3

Aim:
WAP to check whether the input year is leap or not.

Theory:

Enter any year as an input. We first whether the given year is divisible by 400 or not. If it is divisible then it is a leap year else, we check for further conditions. Now if it is divisible by 100 then it is not a leap year or else, we further divide it by 4. If it is divisible then it is a leap year else it's not.

Here is the source code of the Java Program to Find if a Given Year is a Leap Year. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.
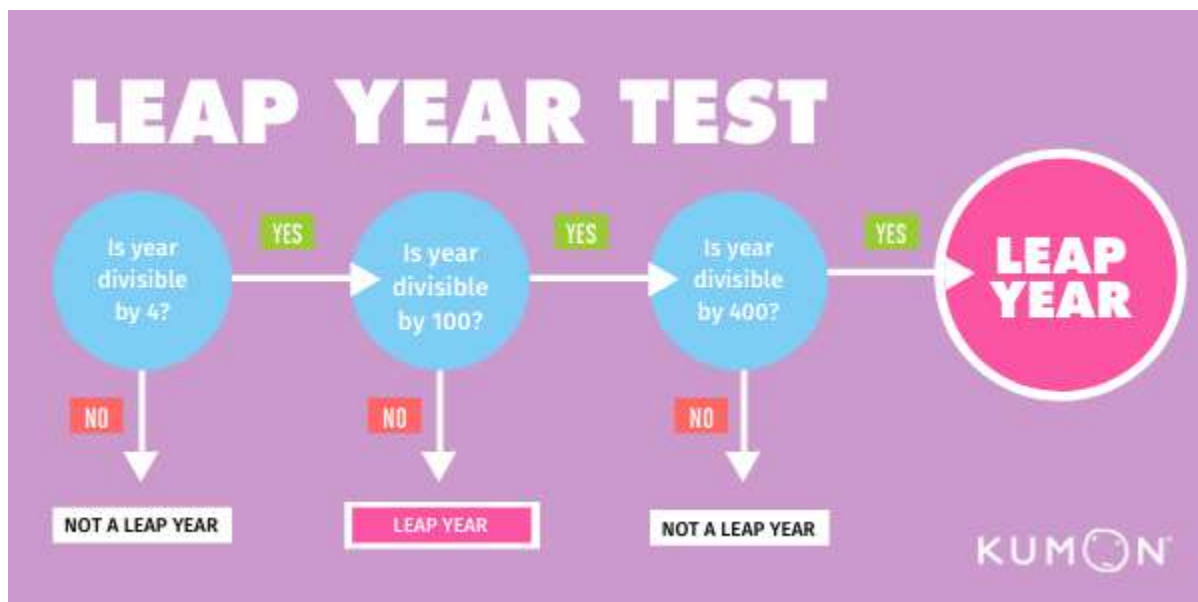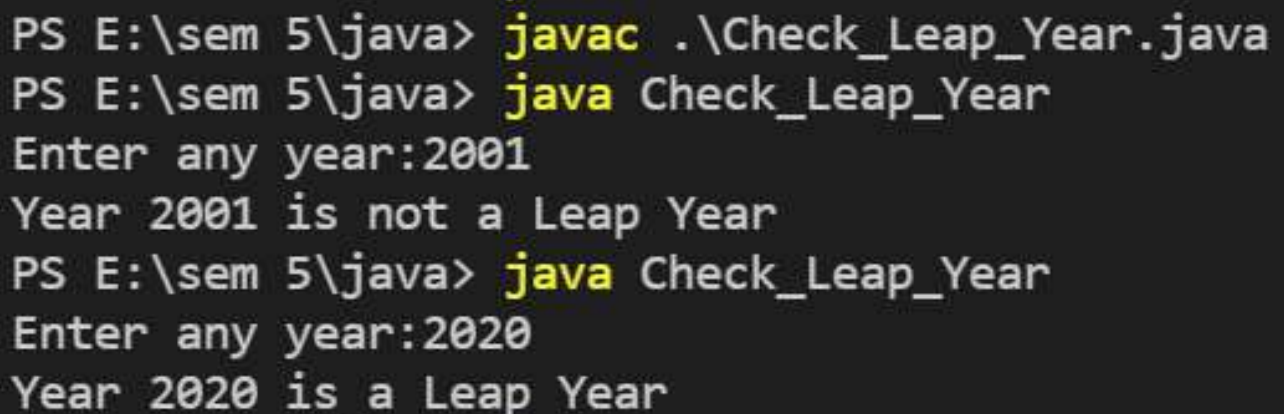

A leap year is a calendar year that contains an additional day added to keep the calendar year synchronized with the astronomical year or seasonal year.

Source Code:

```java
import java.util.Scanner;

public class Check_Leap_Year {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter any year:");
        int year = s.nextInt();
        boolean flag = false;
        if (year % 400 == 0) {
            flag = true;
        } else if (year % 100 == 0) {
            flag = false;
        } else if (year % 4 == 0) {
            flag = true;
        } else {
            flag = false;
        }
        if (flag) {
            System.out.println("Year " + year + " is a Leap Year");
        } else {
            System.out.println("Year " + year + " is not a Leap Year");
        }
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\Check_Leap_Year.java
PS E:\sem 5\java> java Check_Leap_Year
Enter any year:2001
Year 2001 is not a Leap Year
PS E:\sem 5\java> java Check_Leap_Year
Enter any year:2020
Year 2020 is a Leap Year
```

```
PS E:\sem 5\java> java Check_Leap_Year
Enter any year:2021
Year 2021 is not a Leap Year
```

```
PS E:\sem 5\java> java Check_Leap_Year
Enter any year:2020
Year 2020 is a Leap Year
```

EXPERIMENT – 2.4

Aim:

Write an application that accepts two doubles as its command line arguments, multiple these together and display the product.

Theory:

### What Are Command Line Arguments?

The command-line arguments are passed to the program at run-time. Passing command-line arguments in a Java program is quite easy. They are stored as strings in the String array passed to the args parameter of main () method in Java.

In fact, command line arguments are using just for a subject purpose and maybe it is used in advance java core system in order to build the applications, just an expectation. Since, there may be a lot of ways to execute the program.

But it is necessary that you need to know as a newbie about command line arguments.

What are command line arguments?

The console is an interface between the user and program.

When a user enters the inputs on the console using commands, we sending the input as an argument to the main method in java that's why in public static void main() we creating a string array to store values which work at executing time.
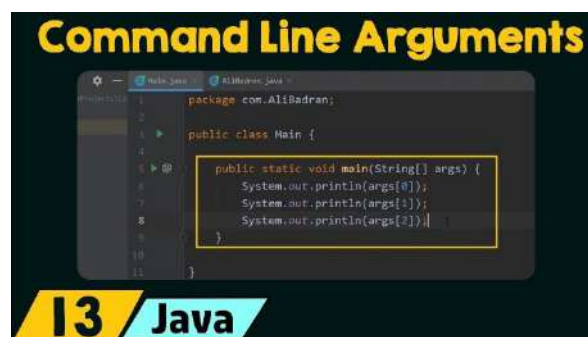
Source Code:

```
class multiply {
  public static void main(String[] args) {
    double num1 = Double.valueOf(args[0]);
    double num2 = Double.valueOf(args[1]);
    System.out.println(num1 * num2);
  }
}
```

Output:

```
PS E:\sem 5\java> javac .\multiply.java
PS E:\sem 5\java> java multiply 2 10
20.0
PS E:\sem 5\java> java multiply 1.5 3
4.5
```

```
PS E:\sem 5\java> java multiply 100 54.2
5420.0
```
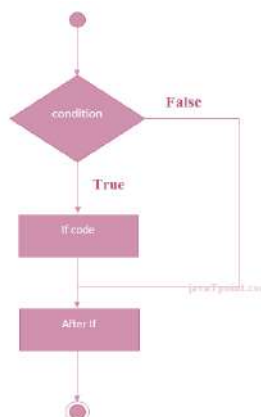
EXPERIMENT – 2.5

Aim:

Write an application that accepts one command line argument, display the line of reporting if number is even or odd.

Theory:

In this program, you'll learn to check if a number entered by an user is even or odd. This will be done using if...else statement and ternary operator in Java.

To understand this example, you should have the knowledge of the following Java programming topics:

- Java if...else Statement

- Java Scanner Class

- **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

- util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

- **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

Syeda Reeha Quasar## Source Code:

```java
public class EvenOdd {

    public static void main(String[] args) {

        int num = Integer.valueOf(args[0]);

        if (num % 2 == 0)
            System.out.println(num + " is even");
        else
            System.out.println(num + " is odd");
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\EvenOdd.java
PS E:\sem 5\java> java EvenOdd 7
7 is odd
PS E:\sem 5\java> java EvenOdd 12
12 is even
PS E:\sem 5\java> java EvenOdd 0
0 is even
```

```
PS E:\sem 5\java> java EvenOdd 923456
923456 is even
```

EXPERIMENT – 2.6

Aim:
Write an application that accepts radius of a circle as its command line argument display the area.

Theory:
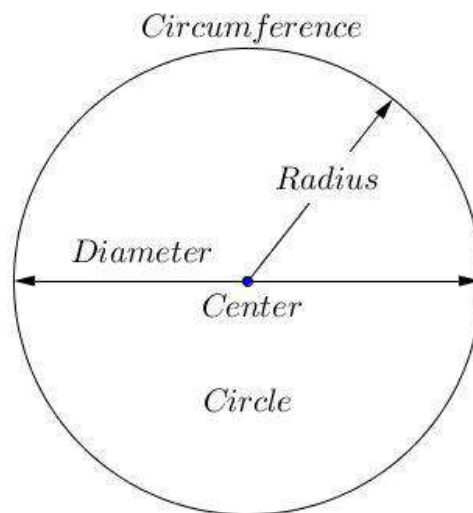
In geometry, the area enclosed by a circle of radius r is πr^2. Here the Greek letter π represents a constant, approximately equal to 3.14159, which is equal to the ratio of the circumference of any circle to its diameter.

The circumference of a circle is the linear distance around its edge.

Source Code:

```java
public class CircleArea {

    public static void main(String[] args) {

        double radius = Double.valueOf(args[0]);

        System.out.println("Area of circle with radius " + radius + " is " +
(radius * radius * 3.14));
    }
}
```
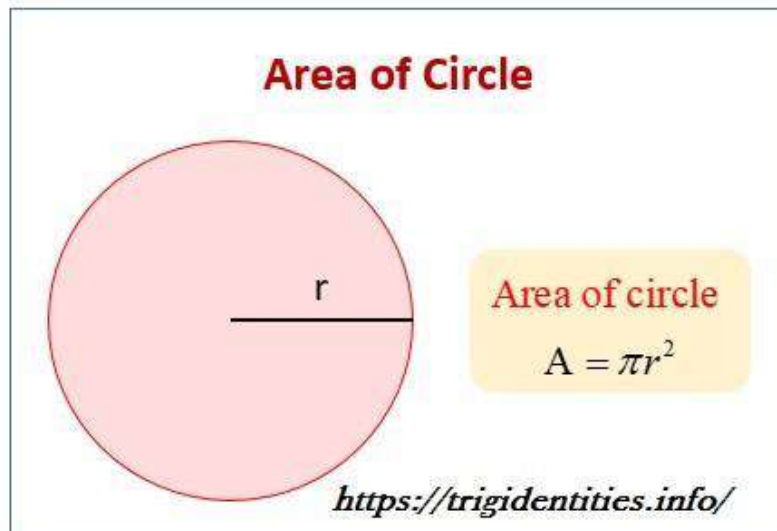
Output:

```
PS E:\sem 5\java> javac .\CircleArea.java
PS E:\sem 5\java> java CircleArea 2
Area of circle with radius 2.0 is 12.56
PS E:\sem 5\java> java CircleArea 30
Area of circle with radius 30.0 is 2826.0
```

```
PS E:\sem 5\java> java CircleArea 30.5
Area of circle with radius 30.5 is 2920.985
```

Viva Questions

1. What is a pointer and does Java support pointers?

Ans.

Pointer is a reference handle to a memory location. Improper handling of pointers leads to memory leaks and reliability issues hence Java doesn't support the usage of pointers.

2. What is the base class of all classes?

Ans.

java.lang.Object

3. Does Java support multiple inheritance?

Ans.

Java doesn't support multiple inheritance.

4. Is Java a pure object-oriented language?

Ans.

Java uses primitive data types and hence is not a pure object-oriented language.

5. Are arrays primitive data types?

Ans.

In Java, Arrays are objects.

LAB - 3

Java Programming Lab

Topics Covered
Arrays, Methods, Method Overloading

Syeda Reeha Quasar
14114802719
4C7

EXPERIMENT – 3.1

Aim:

Write a program to find out the array index or position where sum of numbers preceding the index is equals to sum of numbers succeeding the index.

Theory:

Given, an array of size n. Find an element index that divides the array into two sub-arrays with equal sums.

Examples:

Input: 1 4 2 5

Output: 2

Explanation: If 2 is the partition,   subarrays are : {1, 4} and {5}

Concepts:

> **class** keyword is used to declare a class in Java.

> **public** keyword is an access modifier that represents visibility. It means it is visible to all.

> **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

> **void** is the return type of the method. It means it doesn't return any value.

> **main** represents the starting point of the program.

> **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

> **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➢ **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

➢ util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

➢ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

➢ A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.

➢ The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

```
Input: 1 4 2 5
Output: 2
Explanation: If 2 is the partition,
        subarrays are : {1, 4} and {5}


Input: 2 3 4 1 4 5
Output: 1
Explanation: If 1 is the partition,
 Subarrays are : {2, 3, 4} and {4, 5}
```

Source Code:

```
// Program to find an element such that sum of right side element is equal to sum
of left side
public class sumFragment {

    // Finds an element in an array such that
    // left and right side sums are equal
    static int findElement(int arr[], int n) {
        // Forming prefix sum array from 0
        int[] prefixSum = new int[n];
        prefixSum[0] = arr[0];
        for (int i = 1; i < n; i++)
            prefixSum[i] = prefixSum[i - 1] + arr[i];

        // Forming suffix sum array from n-1
        int[] suffixSum = new int[n];
        suffixSum[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--)
            suffixSum[i] = suffixSum[i + 1] + arr[i];

        // Find the point where prefix and suffix
        // sums are same.
        for (int i = 1; i < n - 1; i++)
            if (prefixSum[i] == suffixSum[i])
                return i; // idex where equal

        return -1;
    }

    public static void main(String args[]) {
        int arr[] = { 1, 4, 3, 5 };
        int n = arr.length;
        System.out.println(findElement(arr, n));
    }
}
```

Output:

```
PS E:\sem 5\java> javac .\sumFragment.java
PS E:\sem 5\java> java sumFragment
2
```

```
int arr[] = { 1, 4, 2, 3, 5 };
```

```
PS E:\sem 5\java> javac .\sumFragment.java
PS E:\sem 5\java> java sumFragment
-1
```

EXPERIMENT – 3.2

Aim:

Write a program that creates and initializes a four-element int array. Calculate and display the average of its values.

Theory:

Enter size of array and then enter all the elements of that array. Now using for loop we calculate sum of elements of array and hence we divide it by number of elements in array to get average.

Here is the source code of the Java Program to Calculate Sum & Average of an Array. The Java program is successfully compiled and run on a Windows system.

Concepts:

➢ **class** keyword is used to declare a class in Java.

➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

➢ **void** is the return type of the method. It means it doesn't return any value.

➢ **main** represents the starting point of the program.

➢ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

➢ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➢ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner

class, we usually pass the predefined object System.in, which represents the standard input stream.

Source Code:

```java
import java.util.Scanner;


public class avgArray {

  public static void main(String[] args) {
    double[] arr = new double[4];
    double total = 0;

    Scanner sc = new Scanner(System.in);

    System.out.println("Enter 4 elements for which you want to calculate average:
");
    for (int i = 0; i < arr.length; i++) {
      arr[i] = sc.nextDouble();
      total = total + arr[i];
    }

    double average = total / arr.length;

    /*
     * This is used for displaying the formatted output if you give %.4f then the
     * output would have 4 digits after decimal point.
     */
    System.out.format("The average is: %.3f", average);
  }
}
```

Output:

```
PS E:\sem 5\java> javac .\avgArray.java
PS E:\sem 5\java> java avgArray
Enter 4 elements for which you want to calculate average:
12 3 4 5
The average is: 6.000
```

```
PS E:\sem 5\java> java avgArray
Enter 4 elements for which you want to calculate average:
20
40
23
87
The average is: 42.500
```
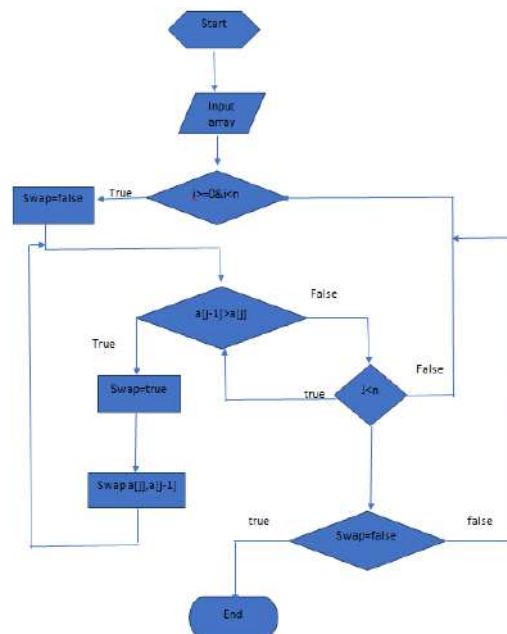
EXPERIMENT – 3.3

Aim:
WAP using Bubble sort for sorting in ascending Order.

Theory:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

i	j	0	1	2	3	4	5	6	7
i = 0	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	4				
	1	1	2	3					
i = 6	0	1	2	3					
		1	2						

Source Code:

```java
import java.util.Scanner;


public class bubbleSort {
    void sort(int[] arr, int n) {
        int temp;
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (arr[j] > arr[j + 1]) {
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
        System.out.println("Sorted Array:");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");

    }


    public static void main(String[] args) {
        bubbleSort sort = new bubbleSort();
        int n;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array ");
        n = sc.nextInt();
        var arr = new int[n];
        System.out.println("Enter the array");
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        sc.close();
        sort.sort(arr, n);
    }

}
```

Output:

```
PS E:\sem 5\java> javac .\bubbleSort.java
PS E:\sem 5\java> java bubbleSort
Enter the size of array 10
Enter the array
12 43 1 100 23 90 22 15 4 7
Sorted Array:
1 4 7 12 15 22 23 43 90 100
```

```
PS E:\sem 5\java> java bubbleSort
Enter the size of array 5
Enter the array
23
7
80
10
12
Sorted Array:
7 10 12 23 80
```

EXPERIMENT – 3.4

Aim:
Create a java program to implement stack and queue concept.

Theory:

Java Stack

The stack is a linear data structure that is used to store the collection of objects. It is based on Last-In-First-Out (LIFO). Java collection framework provides many interfaces and classes to store the collection of objects. One of them is the Stack class that provides different operations such as push, pop, search, etc.

In this section, we will discuss the Java Stack class, its methods, and implement the stack data structure in a Java program. But before moving to the Java Stack class have a quick view of how the stack works.

The stack data structure has the two most important operations that are push and pop. The push operation inserts an element into the stack and pop operation removes an element from the top of the stack. Let's see how they work on stack.
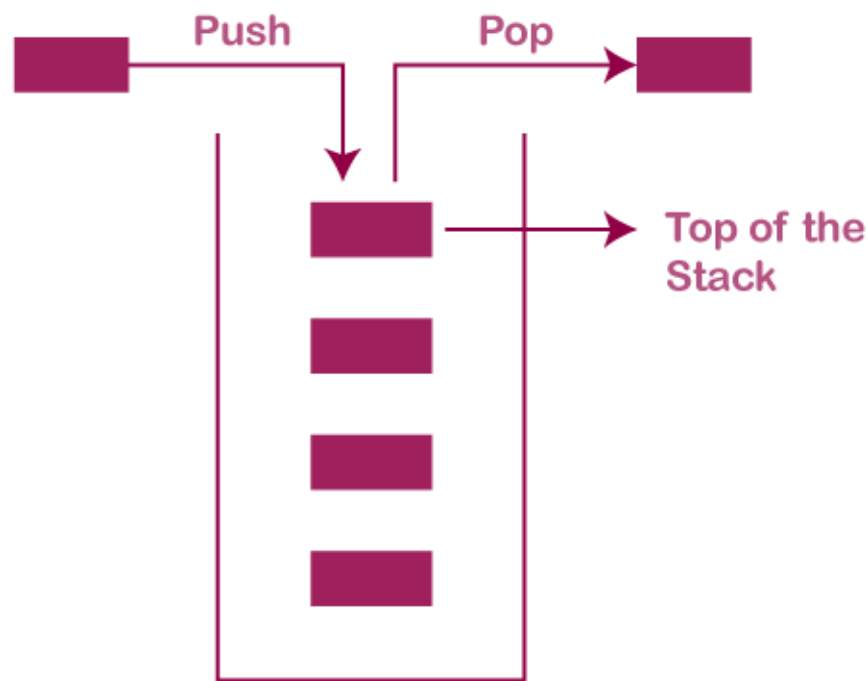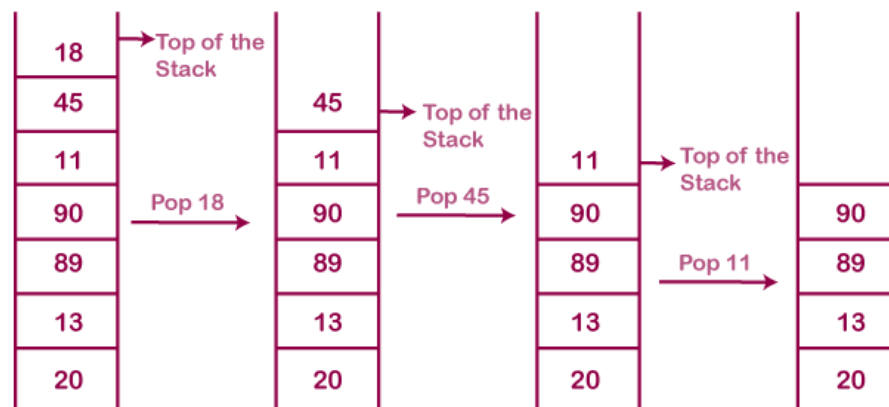
Let's push 20, 13, 89, 90, 11, 45, 18, respectively into the stack.
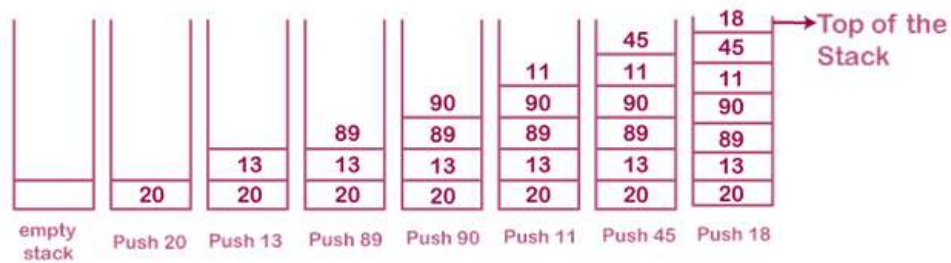
Push operation

Let's remove (pop) 18, 45, and 11 from the stack.

Pop operation

Queue Interface In Java

The Queue interface present in the java.util package and extends the Collection interface is used to hold the elements about to be processed in FIFO(First In First Out) order. It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the FIFO or the First-In-First-Out principle.

Being an interface the queue needs a concrete class for the declaration and the most common classes are the PriorityQueue and LinkedList in Java.It is to be noted that both the implementations are not thread safe. PriorityBlockingQueue is one alternative implementation if thread safe implementation is needed.

Source Code:

```java
import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class stackUsingArrayList {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        List<Integer> arrayList=new ArrayList<>();

        int val;

        System.out.println("Enter your choice: (1-Stack, 2-Queue)");

        val=sc.nextInt();

        if(val==1){

        System.out.println("Push 3 values for Stack");

        for(int i=0;i<3;i++){

            val=sc.nextInt();

            arrayList.add(val);

        }

        System.out.println("Popping element from stack");

        int valuePop=arrayList.get(arrayList.size()-1);

        arrayList.remove(arrayList.size()-1);

        System.out.println(valuePop+" popped from stack");

    }

        else{

            System.out.println("Enter 3 values for Queue");

        for(int i=0;i<3;i++){

            val=sc.nextInt();

            arrayList.add(val);
```

```
        }

        System.out.println("Removing element from queue");

        int valuePop=arrayList.get(0);

        arrayList.remove(0);

        System.out.println(valuePop+" removed from queue");

        }

        sc.close();

    }

}
```

Output:

```
Enter your choice: (1-Stack, 2-Queue)
1
Push 3 values for Stack
55
24
35
Popping element from stack
35 popped from stack
PS C:\Users\Volted User\OneDrive\Desktop\practical> []
```

```
Enter your choice: (1-Stack, 2-Queue)
2
Enter 3 values for Queue
22
41
36
Removing element from queue
22 removed from queue
PS C:\Users\Volted User\OneDrive\Desktop\practical> ▌
```

EXPERIMENT – 3.5

Aim:

Using the concept of method overloading Write method for calculating the area of triangle, circle and rectangle.

Theory:

This is a Java Program to Find Area of Square, Rectangle and Circle using Method Overloading.

We declare three methods of same name but with different number of arguments or with different data types. Now when we call these methods using objects, corresponding methods will be called as per the number of arguments or their datatypes.

Here is the source code of the Java Program to Find Area of Square, Rectangle and Circle using Method Overloading. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

In this program, we will see how to find the area of a square, rectangle, and circle using Method Overloading.

Algorithm:

1. Start

2. Declare three different classes for rectangle, square, and circle.

3. Declare two methods of the same name but with a different number of arguments or with different data types.

4. Call these methods using objects.

5. Call the corresponding methods as per the number of arguments or their data types.

6. Display the result.

7. Stop.

To understand this example, you should have the knowledge of the following Java programming topics:

- [Java if...else Statement](#)

- [Java Scanner Class](#)

- **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

- util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

- **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

Source Code:

```java
import java.util.Scanner;
public class p5 {
    public float area(int a,int b) {
        return (float)a*b;     }
    public float area(float r) {
            return (22*r*r/7); }
    public float area(float h,float b) {
                return h*b/2;        }
    public static void main(String[] args) {
        p5 ar=new p5();
        Scanner sc=new Scanner(System.in);
        int ch;
        System.out.println("Enter choice of area (1-Triangle , 2-Circle , 3-
Rectangle) ");
        ch=sc.nextInt();
```

```java
    switch(ch){
        case 1:float h,b;
            System.out.print("Enter height and base of triangle: ");
            h=sc.nextFloat();
            b=sc.nextFloat();
            System.out.println("Area of Traingle: "+ar.area(h,b));
            break;
        case 2:float r;
            System.out.print("Enter radius of circle: ");
            r=sc.nextFloat();
            System.out.println("Area of Circle: "+ar.area(r));
            break;
        case 3:int a,c;
            System.out.print("Enter length and breadth of triangle: ");
            a=sc.nextInt();
            c=sc.nextInt();
            System.out.println("Area of Rectangle: "+ar.area(a,c));
            break;
        }
    sc.close();
    }
}
```

Output:

```
Enter choice of area (1-Triangle , 2-Circle , 3-Rectangle)
1
Enter height and base of triangle: 5
3
Area of Traingle: 7.5
PS C:\Users\Volted User\OneDrive\Desktop\practical> ▮
```

```
Enter choice of area (1-Triangle , 2-Circle , 3-Rectangle)
2
Enter radius of circle: 4
Area of Circle: 50.285713
PS C:\Users\Volted User\OneDrive\Desktop\practical> ▮
```

```
Enter choice of area (1-Triangle , 2-Circle , 3-Rectangle)
3
Enter length and breadth of Rectangle: 6
3
Area of Rectangle: 18.0
PS C:\Users\Volted User\OneDrive\Desktop\practical> ▮
```

Viva Questions

1. What is difference between Path and Class path?

Ans.

Path and Class path are operating system level environment variables. Path is used define where the system can find the executables(.exe) files and class path is used to specify the location .class files.

2. What are local variables?

Ans.

Local variables are those which are declared within a block of code like methods. Local variables should be initialized before accessing them.

3. Can a class declared as private be accessed outside its package?

Ans.

Not possible.

4. Can a class be declared as protected?

Ans.

A class can't be declared as protected. only methods can be declared as protected.

5. What is the access scope of a protected method?

Ans.

A protected method can be accessed by the classes within the same package or by the subclasses of the class in any package.

LAB - 4

Java Programming Lab

Topics Covered
Classes & object Constructor

Syeda Reeha Quasar

14114802719

4C7

EXPERIMENT – 4.1

Aim:

WAP that creates a class circle with instance variables for the center and the radius. Initialize and display its variables.

Theory:

Class Definition in Java

A Class is a template for an object. Every object has a class which defines the structure of an object (that means what are various component in it, termed as member elements) its functional inter feces (called methods) which decide what messages the object will respond to and how. The general form of class definition is shown below.

Class Class Name

[extends SuperClassName]

    [implements Interface] {

[declaration of member elements]

[declaration of methods]}

Creating and Initializing Objects

Take another look at how we've been creating Circle objects:

Circle c = new Circle();

Constructor overloading in Java

In Java, we can overload constructors like methods. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

Consider the following java program, in which we have used different constructors in the class.

Instantiating a Class

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the object constructor.

Note: The phrase "instantiating a class" means the same thing as "creating an object." When you create an object, you are creating an "instance" of a class, therefore "instantiating" a class.

Concepts:

➢ **class** keyword is used to declare a class in Java.

➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

➢ **void** is the return type of the method. It means it doesn't return any value.

➢ **main** represents the starting point of the program.

➢ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

➢ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➢ **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

➢ util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

➢ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

➢ A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.

> ➢ The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

Source Code:

```java
class CircleExp1 {
    public int x;
    public int y;
    public int radius;
}

public class Lab4exp1 {
    public static void main(String args[]) {
        CircleExp1 obj = new CircleExp1();
        obj.x = 10;
        obj.y = 20;
        obj.radius = 5;
System.out.println("Co-ordinates of center : ( " + obj.x + " , " + obj.y + " )");
System.out.println("Radius : " + obj.radius );
    }
}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code> java Lab4exp1
Co-ordinates of center : ( 10 , 20 )
Radius : 5
```

EXPERIMENT – 4.2

Aim:

Modify experiment 4.1 to have a constructor in class circle to initialize its variables.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Object : It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

New Operator : The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

Instance Variables : Instance variables are declared in a class, but outside a method, constructor or any block. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.

Constructor : In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to

initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

Source Code:

```
class Circle {
    int x;
    int y;
    int radius;

    //constructor
    Circle(int i, int j, int r) {
        x = i;
        y = j;
        radius = r;
    }
}

public class Lab4exp2 {
    public static void main(String args[]) {
        Circle obj = new Circle(5, 5, 10);
System.out.println("Co-ordinates of center : ( " +   obj.x + " , " + obj.y +
" )");
            System.out.println("Radius :   : " + obj.radius);
    }
}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code> java Lab4exp2
Co-ordinates of center : ( 5 , 5 )
Radius :   : 10
```

EXPERIMENT – 4.3

Aim:

Modify experiment 4.2 to show constructor overloading.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Object : It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

New Operator : The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

Constructor : In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

Constructor Overloading : Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

Source Code:

```
class CircleExp3 {
    int x;
    int y;
    int radius;
    // constructor overloading
    // default constructor
    CircleExp3() {
        x = 0;
        y = 0;
        radius = 1;
    }
    // user input constructor
    CircleExp3(int i, int j, int r) {
        x = i;
        y = j;
        radius = r;
    }
}

public class Lab4exp3 {
    public static void main(String args[]) {
        CircleExp3 obj = new CircleExp3(1, 1, 4);
        CircleExp3 obj1 = new CircleExp3();
        System.out.println("---Constructor 1---");
        System.out.println("Co-ordinates of center : ( " + obj1.x + " , " +
obj1.y + " )");
        System.out.println("Radius :   : " + obj1.radius);
        System.out.println("---Constructor 2---");
        System.out.println("Co-ordinates of center : ( " + obj.x + " , " +
obj.y + " )");
        System.out.println("Radius :   : " + obj.radius);
    }
}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code> java Lab4exp3
---Constructor 1---
Co-ordinates of center : ( 0 , 0 )
Radius :  : 1
---Constructor 2---
Co-ordinates of center : ( 1 , 1 )
Radius :  : 4
```

```
---Constructor 1---
Co-ordinates of center : ( 0 , 0 )
Radius :  : 1
---Constructor 2---
Co-ordinates of center : ( 1 , 1 )
Radius :  : 4
```

EXPERIMENT – 4.4

Aim:
WAP to display the use of this keyword.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Object : It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

New Operator : The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

Constructor : In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

This keyword : this() reference can be used during constructor overloading to call default constructor implicitly from parameterized constructor.

Source Code:

```java
class CircleExp4 {
    int x;
    int y;
    int radius;

    //constructor
    CircleExp4(int i, int j, int r) {
        this.x = i;
        this.y = j;
        this.radius = r;
    }
}

public class Lab4exp4 {
    public static void main(String args[]) {
        CircleExp4 obj = new CircleExp4(2, 2, 5);
        System.out.println("Co-ordinates of center : ( " + obj.x + " , " + obj.y
+ " )");
        System.out.println("Radius :  : " + obj.radius);
    }
}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code> java Lab4exp4
Co-ordinates of center : ( 2 , 2 )
Radius :  : 5
```

Co-ordinates of center : (2 , 2)
Radius :   : 5

EXPERIMENT – 4.5

Aim:
Write a program that can count the number of instances created for the class.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Object : It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

New Operator : The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

Constructor : In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

Static variable : When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

Source Code:

```java
class countInstance {
    public static int count = 0;

    countInstance() {
        count++;
    }
}

public class Lab4exp5 {
    public static void main(String args[]) {
        countInstance obj1 = new countInstance();
        countInstance obj2 = new countInstance();
        countInstance obj3 = new countInstance();
        countInstance obj4 = new countInstance();
        countInstance obj5 = new countInstance();
        System.out.println("Number of instances created : " +
countInstance.count);
    }
}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code> java Lab4exp5
Number of instances created : 5
```

Number of instances created : 5

EXPERIMENT – 4.6

Aim:
Write a Java Program to get the cube of a given number using the static method.

Theory:

Functions Used :

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

- **Static method :** The static keyword is used to create methods that will exist independently of any instances created for the class. Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables. Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method.

Source Code:

```java
class getCube {
    public static int Cube(int num) {
        return (num * num * num);
    }
}

public class Lab4exp6 {
    public static void main(String args[]) {
        int num = 4;
        System.out.println("Cube is : " + getCube.Cube(num));
    }
}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code> java Lab4exp6
Cube is : 64
```

Cube is : 64

EXPERIMENT – 4.7

Aim:

WAP that describes a class person. It should have instance variables to record name, age and salary. Create a person object. Set and display its instance variables.

Theory:

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

- **Instance Variables :** Instance variables are declared in a class, but outside a method, constructor or any block. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's

state that must be present throughout the class.

- **Constructor :** In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

Source Code:

```
class Person{
    String name;
    int age;
    int salary;
    Person(String n, int a, int s){
        name = n;
        age = a;
        salary = s;
    }
}

public class Lab4exp7 {
    public static void main(String args[]) {
        Person p1 = new Person("Udit",20,200000);

System.out.println("Name is :  " + p1.name + "\nAge is :  " + p1.age + "\nSalary
is :  " + p1.salary  );
    }
}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code> java Lab4exp7
Name is :  Udit
Age is :  20
Salary is :  200000
```

```
Name: Anuj.
Age: 61
Salary: 4500000

Name: Virat.
Age: 29
Salary: 6000000

Name: Varun.
Age: 40
Salary: 4200000
```

Viva Questions

1. Is Java a pure object oriented language?

Ans.

Java uses primitive data types and hence is not a pure object oriented language.

2. What are local variables?

Ans.

Local variables are those which are declared within a block of code like methods. Local variables should be initialized before accessing them.

3. Can a class declared as private be accessed outside its package?

Ans.

Not possible.

4. Can a class be declared as protected?

Ans.

A class can't be declared as protected. only methods can be declared as protected.

5. What is the access scope of a protected method?

Ans.

A protected method can be accessed by the classes within the same package or by the subclasses of the class in any package.

6. What is the purpose of declaring a variable as final?

Ans.

A final variable's value can't be changed. final variables should be initialized before using them.

7. Can a class be declared as static?

Ans.

We can not declare top level class as static, but only inner class can be declared static.

```
public class Test

{

static class InnerClass

{

public static void InnerMethod()

{ System.out.println("Static Inner Class!");

 }

}

public static void main(String args[])

{

Test.InnerClass.InnerMethod();

}

}

//output: Static Inner Class!
```

8. When will you define a method as static?

Ans.

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

9. What are the restriction imposed on a static method or a static block of code?

Ans.

A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

LAB - 5

Java Programming Lab

Topics Covered
Inheritance , Polymorphism , Method overiding

Syeda Reeha Quasar
14114802719
4C7

EXPERIMENT – 5.1

Aim:

WAP that implements method overriding.

Theory:

Method Overriding in Java

> ➢ Understanding the problem without method overriding

> ➢ Can we override the static method

> ➢ Method overloading vs. method overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

1. Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

2. Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class

2. The method must have the same parameter as in the parent class.

3. There must be an IS-A relationship (inheritance).

Concepts:

➤ **class** keyword is used to declare a class in Java.

➤ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

➤ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

➤ **void** is the return type of the method. It means it doesn't return any value.

➤ **main** represents the starting point of the program.

➤ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

➤ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➤ **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

➤ util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

➤ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

➤ A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.

➤ The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

Source Code:

```java
class parent{

    void show(){

        System.out.println("In parent class");

    }

}
public class child extends parent{

    void show(){

        System.out.println("In child class");

    }

    public static void main(String[] args){

        child c=new child();

        c.show();

    }

}
```

Output:

```
In child class
PS C:\Users\Volted User\OneDrive\Desktop\practical> []
```

```
In child class
```

EXPERIMENT – 5.2

Aim:
WAP to illustrate simple inheritance.

Theory:

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called inheritance. The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.

Child Class:
The class that extends the features of another class is known as child class, sub class or derived class.

Parent Class:
The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.
Inheritance allows us to reuse of code, it improves reusability in your java application.
Note: The biggest advantage of Inheritance is that the code that is already present in base class need not be rewritten in the child class.

This means that the data members(instance variables) and methods of the parent class can be used in the child class as.

If you are finding it difficult to understand what is class and object then refer the guide that I have shared on object oriented programming: OOPs Concepts

Syntax: Inheritance in Java

To inherit a class we use extends keyword. Here class XYZ is child class and class ABC is parent class. The class XYZ is inheriting the properties and methods of ABC class.

class XYZ extends ABC{}

Source Code:

```
class parents{

    private String str= "Parent";

    private int age=50;

    void displayParent(){

        System.out.println("Class name: "+str);

        System.out.println("Class age: "+age);

    }

}
public class childs extends parents{

    private String str;

    private int age;

    childs(){

        str="undefined";

        age=-1;

    }

    childs(String str,int age){

        this.str=str;

        this.age=age;

    }

    void displayChild(){

        System.out.println("Class name: "+str);
```

```
        System.out.println("Class age: "+age);

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

        childs c=new childs("Child",10);

        c.displayParent();

        c.displayChild();

    }
}
```

Output:

```
Class name: Parent
Class age: 50
Class name: Child
Class age: 10
PS C:\Users\Volted User\OneDrive\Desktop\practical> []
```

```
Class name: Parent
Class age: 50
Class name: Child
Class age: 10
```

EXPERIMENT – 5.3

Aim:
WAP to illustrate multilevel inheritance.

Theory:

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

- **Constructor :** In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

- **Constructor Overloading :** Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

Multilevel inheritance in java with example

When a class extends a class, which extends anther class then this is called **multilevel inheritance**. For example class C extends class B and class B extends class A then this **type of inheritance** is known as multilevel inheritance.

Lets see this in a diagram:

Multilevel Inheritance
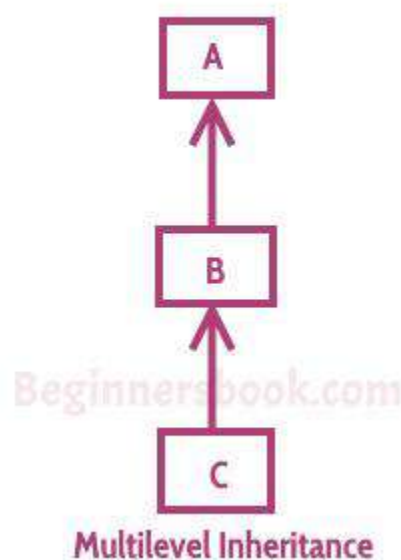
It's pretty clear with the diagram that in Multilevel inheritance there is a concept of grand parent class. If we take the example of this diagram, then class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and methods of class A along with class B that's what is called multilevel inheritance.

Source Code:

```java
class grandParent{
    private double inheritance=24000.50;
    double inheritAmount(){
        return inheritance;
    }
}
class parent extends grandParent{
    private String str= "Parent";
    private int age=49;
    void displayParent(){
        System.out.println("Class name: "+str);
        System.out.println("Class age: "+age);
    }
    double inheritAmount(){
        return super.inheritAmount()+5500;
    }
}
}
public class child extends parent{
    private String str;
    private int age;
    child(){
        str="undefined";
        age=-1;
    }
    child(String str,int age){
        this.str=str;
        this.age=age;
    }
    void displayChild(){
        System.out.println("Class name: "+str);
        System.out.println("Class age: "+age);
    }
    double inheritAmount(){
        return super.inheritAmount()+20.5;
    }
    public static void main(String[] args){
        child c=new child("Child",10);
        c.displayParent();
        c.displayChild();
        System.out.println("Total amount: "+c.inheritAmount());
    }
}
```

Output:

```
Class name: Parent
Class age: 49
Class name: Child
Class age: 10
Total amount: 29521.0
PS C:\Users\Volted User\OneDrive\Desktop\practical> []
```

```
Class name: Parent
Class age: 49
Class name: Child
Class age: 10
Total amount: 29521.0
```

EXPERIMENT – 5.4

Aim:
WAP illustrating all uses of super keywords.

Theory:

Super Keyword in Java

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.'

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.
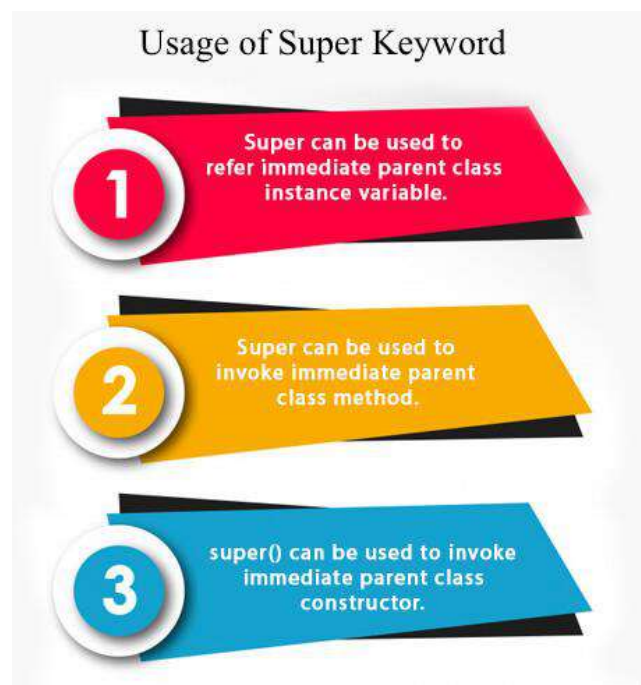
1) super is used to refer immediate parent class instance variable.

2) super can be used to invoke parent class method

3) super is used to invoke parent class constructor.

Source Code:

```
class box{
    private int a,b,c;
    box(){
        a=b=c=-1;
    }
    box(int a,int b,int c){
        this.a=a;
        this.b=b;
        this.c=c;
    }
    box(box o){
        a=o.a;
        b=o.b;
        c=o.c;
    }
    float volume(){
        return (float)a*b*c;
    }
    void shape(){
        System.out.println("It is box shape");
    }
}
class boxWeight extends box{
    private double w;
    // or we can use a for weight too, and further use super.a for superclass a
    boxWeight(){
        super();
        w=-1;
    }
    boxWeight(int a,int b,int c,double w){
        super(a,b,c);
        this.w=w;
    }
    boxWeight(boxWeight wo){
        super(wo);
```

```
            w=wo.w;
    }
    double density(){
        return w/volume();
    }
    void shape(){
        super.shape();
        System.out.println("It also have mass factor");
    }
}
public class classIntro {
    public static void main(String[] args) {
        box b1=new box();
        box b2=new box(10,15,10);
        box b3=new box(b2);
        System.out.println(b1.volume()>=0?b1.volume():"No paramteres");
        System.out.println(b2.volume()>=0?b2.volume():"No paramteres");
        System.out.println(b3.volume()>=0?b3.volume():"No paramteres");


        boxWeight b=new boxWeight(5,10,5,25.0);
        System.out.println(b.density()>=0?b.density():"No paramteres");
        b.shape();
    }

}
```

Output:

```
No paramteres
1500.0
1500.0
0.1
It is box shape
It also have mass factor
PS C:\Users\Volted User\OneDrive\Desktop\practical> 
```

Viva Questions

1. When will you define a method as static?

Ans.

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

2. What are the restriction imposed on a static method or a static block of code?

Ans.

A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

3. I want to print "Hello" even before main() is executed. How will you acheive that?

Ans.

Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the memory and even before the creation of an object. Hence it will be executed before the main() method. And it will be executed only once.

4. How-will-you-communicate-between-two-Applets

Ans.

The simplest method is to use the static variables of a shared class since there's only one instance of the class and hence only one copy of its static variables. A slightly more reliable method relies on the fact that all the applets on a given page share the same AppletContext. We obtain this applet context as follows: AppletContext ac = getAppletContext(); AppletContext provides applets with methods such as getApplet(name), getApplets(),getAudioClip, getImage, showDocument and showStatus().

5. Which classes can an applet extend?

Ans.

An applet can extend the java.applet.Applet class or the java.swing.JApplet class. The java.applet.Applet class extends the java.awt.Panel class and enables you to use the GUI tools in the AWT package. The java.swing.JApplet class is a subclass of java.applet.Applet that also enables you to use the Swing GUI tools.

6. What is the purpose of declaring a variable as final?

Ans.

A final variable's value can't be changed. final variables should be initialized before using them.

7. When will you define a method as static?

Ans.

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

LAB - 6

Java Programming Lab

Topics Covered
Inheritance, Polymorphism, Method overriding

Syeda Reeha Quasar
14114802719
4C7

EXPERIMENT – 6.1

Aim:
Create an abstract class shape. Let rectangle and triangle inherit this shape class. Add necessary functions.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Object : It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

This keyword : this() reference can be used during constructor overloading to call default constructor implicitly from parameterized constructor.

Abstract Class : A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body). **Abstraction** is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it. A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Concepts:

➢ **class** keyword is used to declare a class in Java.

➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

➢ **void** is the return type of the method. It means it doesn't return any value.

➢ **main** represents the starting point of the program.

➢ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

➢ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➢ **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

➢ util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

➢ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

➢ A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.

➢ The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

Source Code:

Shape class :

```
public abstract class Shape
{
    private double height;
    private double width;

    public void setValues(double height, double width)
    {
        this.height = height;
        this.width = width;
    }

    public double getHeight()
    {
        return height;
    }

    public double getWidth()
    {
        return width;
    }

    public abstract double getArea();
}
```

Rectangle Class :

```
public class Rectangle extends Shape
{
```

```java
    public double getArea()

    {

        return getHeight() * getWidth();

    }

}
```

Triangle Class :

```java
public class Triangle extends Shape

{

    public double getArea()

    {

        return (getHeight() * getWidth()) / 2;

    }

}
```

Main Class :

```java
public class AbstractDemo

{

    public static void main(String[] args)

    {

        Shape obj;

        //Rectangle object
        Rectangle rect = new Rectangle();
        obj = rect;
        obj.setValues(10, 20);
        System.out.println("Area of rectangle : " + obj.getArea());
```

```
        /Triangle object

        Triangle tri = new Triangle();

        obj = tri;

        obj.setValues(10,20);

        System.out.println("Area of triangle : " + obj.getArea());

    }

}
```

Output:

```
PS F:\College Stuff\3rd year\5th Sem\Java> java Lab6exp1
Area of rectangle : 200.0
Area of triangle : 100.0
```

```
Area of rectangle : 200.0
Area of triangle : 100.0
```

EXPERIMENT – 6.2

Aim:
Write a java package to show dynamic polymorphism and interfaces.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Object : It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

Interface : An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship.** It cannot be instantiated just like the abstract class. An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Polymorphism : Polymorphism in Java occurs when there are one or more classes or objects related to each other by inheritance. It is the ability of an object to take many forms. Inheritance lets users inherit attributes and methods, and polymorphism uses these methods to perform different tasks. So, the goal is communication, but the approach is different.

Dynamic Polymorphism : Dynamic polymorphism is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time. It is also known as runtime polymorphism or dynamic method dispatch. We can achieve dynamic polymorphism by using the method overriding. In this process, an overridden method is called through a reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Source Code:

```
package shapes;


interface i1 {

    void func();

}


class circle implements i1 {

    public void func() {

        System.out.println("Interface function !");

    }


    public void name() {

        System.out.println("This is a Circle ! ");
```

```
    }
}


class square extends circle {
    public void name() {
        System.out.println("TThis is a Square ! ");
    }
}


class triangle extends circle {
    public void name() {
        System.out.println("This is a Triangle ! ");
    }
}


public class lab2 {
    public static void main(String[] args) {
        circle obj1 = new circle();
        circle obj2 = new square();
        circle obj3 = new triangle();
        obj1.name();
        obj2.name();
        obj3.name();
        obj1.func();
    }
}
```

Output:

```
[Running] cd "f:\College Stuff\3rd year\5th Sem\Java\" && javac lab2.java && java lab2
This is a Circle !
TThis is a Square !
This is a Triangle !
Interface function !
```

This is a Circle !

TThis is a Square !

This is a Triangle !

Interface function !

EXPERIMENT – 6.3

Aim:
Write an application that creates an 'interface' and implements it.

Theory:

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

- **Constructor :** In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

- **Constructor Overloading :** Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

- **Interface :** An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**. It cannot be instantiated just like the abstract class.

- An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Source Code:


```java
interface Results {

    final static float pi = 3.14f;

    float areaOf(float l, float b);

}


class Rectangle implements Results {

    public float areaOf(float l, float b) {

        return (l * b);
```

```
        }
}


class Square implements Results {

    public float areaOf(float l, float b) {

        return (l * l);

    }

}


class Circle implements Results {

    public float areaOf(float r, float b) {

        return (pi * r * r);

    }

}


public class Lab6exp3 {

    public static void main(String args[]) {

        Rectangle rect = new Rectangle();

        Square square = new Square();

        Circle circle = new Circle();

        System.out.println("Area of Rectangle: " + rect.areaOf(10, 20));

        System.out.println("Are of square: " + square.areaOf(10, 20));

        System.out.println("Area of Circle: " + circle.areaOf(10, 20));

    }

}
```

Output:

```
PS C:\Users\udita\OneDrive\Desktop> java Lab6exp3
Area of Rectangle: 200.0
Are of square: 100.0
Area of Circle: 314.0
```

```
Area of Rectangle: 200.0
Are of square: 100.0
Area of Circle: 314.0
```

EXPERIMENT – 6.4

Aim:
Write an application to illustrate Interface Inheritance.

Theory:

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

- **Constructor :** In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

- **Constructor Overloading :** Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

- **Interface :** An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**. It cannot be instantiated just like the abstract class.

- An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

- **Inheritance : Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of Object Oriented programming system. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

  extends is the keyword used to inherit the properties of a class.

Source Code:

```
interface A
{
    void func_A();
}

interface B extends A
{
    void func_B();
}

class Lab6exp4 implements B
{
    @Override
    public void func_A()
    {
        System.out.println("Inside A");
    }

    @Override
    public void func_B()
    {
        System.out.println("Inside B");
    }

    public static void main (String[] args)
    {
        Lab6exp4 obj1 = new Lab6exp4();


        obj1.func_A();
        obj1.func_B();
    }
}
```

Output:

```
PS C:\Users\udita\OneDrive\Desktop> java Lab6exp4
Inside A
Inside B
```

```
Inside A
Inside B
```

Viva Questions

1. What is interface in Java?

Ans.

Interface in java is core part of Java programming language and one of the way to achieve abstraction in Java along with abstract class. Even though interface is fundamental object oriented concept; Many Java programmers thinks Interface in Java as advanced concept and refrain using interface from early in programming career. At very basic level interface in java is a keyword but same time it is an object oriented term to define contracts and abstraction , This contract is followed by any implementation of Interface in Java. Since multiple inheritance is not allowed in Java, interface is only way to implement multiple inheritance at Type level. In this Java tutorial we will see What is an interface in Java, How to use interface in Java and where to use interface in Java and some important points related to Java interface. What is an interface in Java is also a common core Java question which people asked on various programming exams and interviews.

2. What is the difference between an Interface and an Abstract class?

Ans.

An abstract class can have instance methods that implement a default behaviour. An Interface can only declare constants and instance methods, but cannot implement default behaviour and all methods are implicitly abstract. An interface has all public members and no implementation. An abstract class is a class which may have the usual flavours of class members (private, protected, etc.), but has some abstract methods.

3. I don't want my class to be inherited by any other class. What should i do?

Ans.

You should declared your class as final. But you can't define your class as final, if it is an abstract class. A class declared as final can't be extended by any other class.

4. Can an interface extends another interface in Java?

Ans.

Yes, an interface can extend another interface.

5. Can an interface implement another interface?

Ans.

No, an interface cannot implement another interface.

6. Why do we need to use inheritance?

Ans.

Inheritance is one of the main pillars of OOPs concept. Some objects share certain properties and behaviors. By using inheritance, a child class acquires all properties and behaviors of parent class.

There are the following reasons to use inheritance in java.

- We can reuse the code from the base class.
- Using inheritance, we can increase features of class or method by overriding.
- Inheritance is used to use the existing features of class.
- It is used to achieve runtime polymorphism i.e method overriding.

LAB - 7

Java Programming Lab

Topics Covered
Exception Handling,
Applet

Syeda Reeha Quasar
14114802719
4C7

EXPERIMENT – 7.1

Aim:
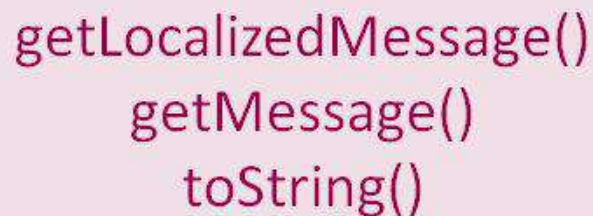
Write an application that shows how to create a user-defined exception.

Theory:

User Defined Exception or custom exception is creating your own exception class and throws that exception using 'throw' keyword. This can be done by extending the class Exception.

Exception

getLocalizedMessage()
getMessage()
toString()

There is no need to override any of the above methods available in the Exception class, in your derived class. But practically, you will require some amount of customizing as per your programming needs.

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

To understand how exception handling works in Java, you need to understand the three categories of exceptions:

- Checked exceptions: A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- Runtime exceptions: A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compliation.
- Errors: These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Concepts:

➢ **class** keyword is used to declare a class in Java.

➢ **public** keyword is an access modifier that represents visibility. It means it is visible to all.

➢ **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to

create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.

➢ **void** is the return type of the method. It means it doesn't return any value.

➢ **main** represents the starting point of the program.

➢ **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.

➢ **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

➢ **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.

➢ util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.

➢ **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.

➢ A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.

➢ The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

Source Code:

```
public class JavaException {

    public static void main(String args[]) {

        try {

            throw new MyException(2);

            // throw is used to create a new exception and throw it.

        } catch (MyException e) {

            System.out.println(e);

        }

    }

}


class MyException extends Exception {

    int a;


    MyException(int b) {

        a = b;

    }


    public String toString() {

        return ("Exception Number =  " + a);

    }

}
```

Output:

```
PS E:\sem 5\java> javac .\JavaException.java
PS E:\sem 5\java> java JavaException
Exception Number =  2
```

```
Exception Number =  2
```

EXPERIMENT – 7.2

Aim:

Create a customized exception and also make use of all the 5 exception keywords.

Theory:

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.
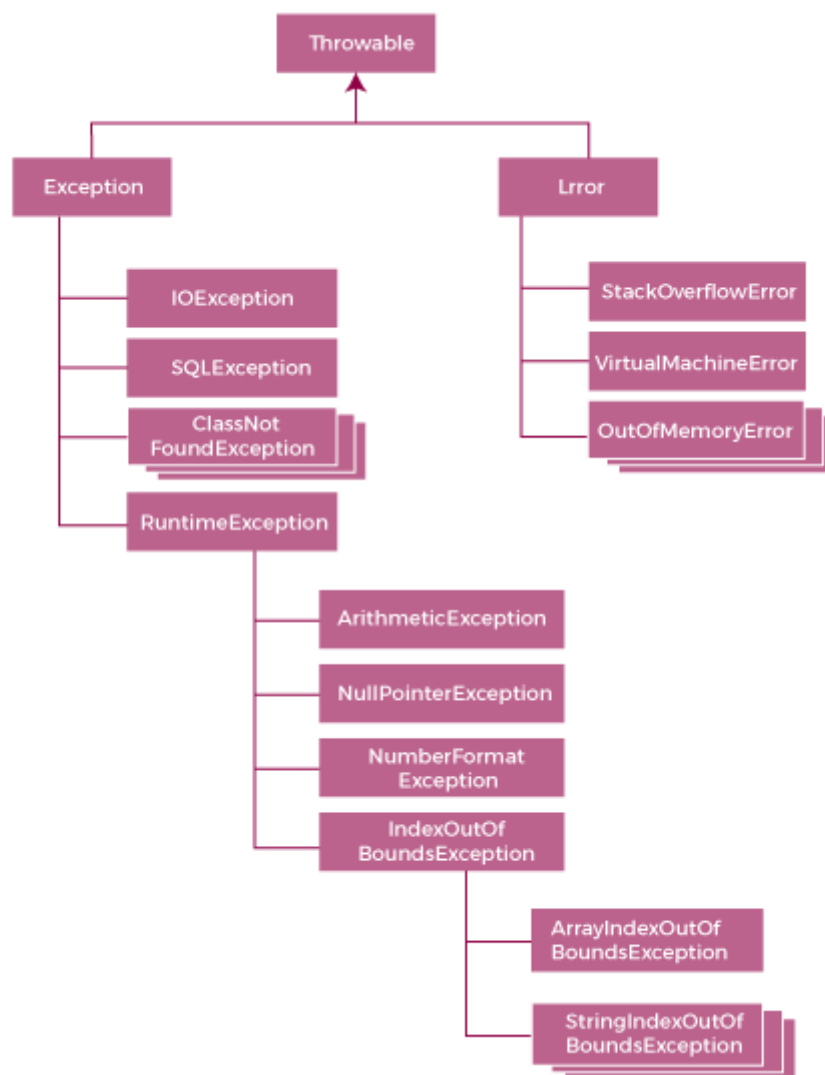
The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Source Code:

```java
import java.io.IOException;

public class exceptions {
    void divideSt() {
        try {
            throw new ArithmeticException("func Can't divide a number by 0 --
-> this is by throws");
        } catch (Exception e) {
            System.out.println("Exception handles of throw");
        }
    }

    public static void main(String args[]) throws ArithmeticException {
        exceptions obj = new exceptions();
        try {
```

```
        int data = 100 / 0;

        throw new ArithmeticException("Can't divide a number by 0 --->
this is by throws");

    } catch (ArithmeticException e) {

        System.out.println("Exception handles of throw2" + e);

        obj.divideSt();


    } catch (Exception e) {

        System.out.println("Exception handles of throw3" + e);


    } finally {

        System.out.println("finally executed");

        System.out.println("normal flow...");

    }

   }
}
```

Output:

```
PS E:\sem 5\java> javac .\exceptions.java
PS E:\sem 5\java> java exceptions
Exception handles of throw2java.lang.ArithmeticException: / by zero
Exception handles of throw
finally executed
normal flow...
```

```
Exception handles of throw2java.lang.ArithmeticException: / by zero
Exception handles of throw
finally executed
normal flow...
```

EXPERIMENT – 7.3

Aim:
Write an application that creates an 'interface' and implements it.

Theory:

All applets are subclasses of Applet. Thus, all applets must import java.applet. Applets must also import java.awt. Recall that AWT stands for the Abstract Window Toolkit. Since all applets run in a window, it is necessary to include support for that window. Applets are not executed by the console-based Java run-time interpreter. Rather, they are executed by either a Web browser or an applet viewer. The figures shown in this chapter were created with the standard applet viewer, called applet viewer, provided by the SDK. But you can use any applet viewer or browser you like. Execution of an applet does not begin at main( ). Actually, few applets even have main( ) methods. Instead, execution of an applet is started and controlled with an entirely different mechanism, which will be explained shortly. Output to your applet's window is not performed by System.out.println( ). Rather, it is handled with various AWT methods, such as drawString( ), which outputs a string to a specified X,Ylocation. Input is also handled differently than in an application. Once an applet has been compiled, it is included in an HTML file using the APPLET tag. The applet will be executed by a Java-enabled web browser when it encounters the APPLET tag within the HTML file. To view and test an applet more conveniently, simply include a comment at the head of your Java source code file that contains the APPLET tag. This way, your code is documented with the necessary HTML statements needed by your applet, and you can test the compiled applet by starting the applet viewer with your Java source code file specified as the target.

Applet An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet:

Four methods in the Applet class give you the framework on which you build any serious applet:

- init: This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- start: This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- stop: This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- destroy: This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- paint: Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

Application Conversion to Applets: It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the java program launcher) into an applet that you can embed in a web page. Here are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.

- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.
- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.
- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object.the browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

Source Code:

```java
package javaapplication1;


import javax.swing.JApplet;
import java.awt.*;


/**
 *
 * @author reeha
 */
public class NewJApplet extends JApplet {
```

```
public void init() {

    getContentPane().setBackground(Color.BLACK);

}


public void paint(Graphics g) {

    // set color to blue

    g.setColor(Color.blue);


    // print hello world

    g.drawString("Hello World...", 100, 100);


    showStatus("Syeda Reeha Quasar");

}
}
```

Output:

Applet Viewer: java...   —   ☐   ✕

Applet

Hello World...

Syeda Reeha Quasar
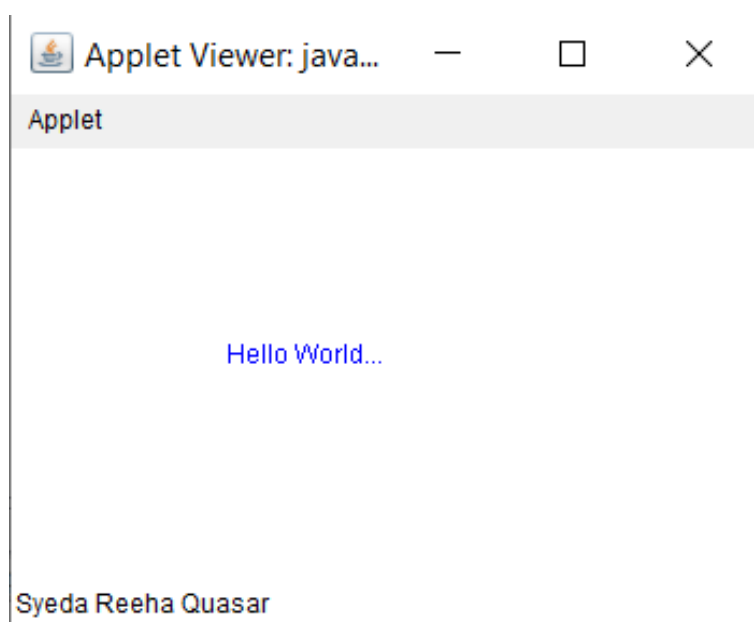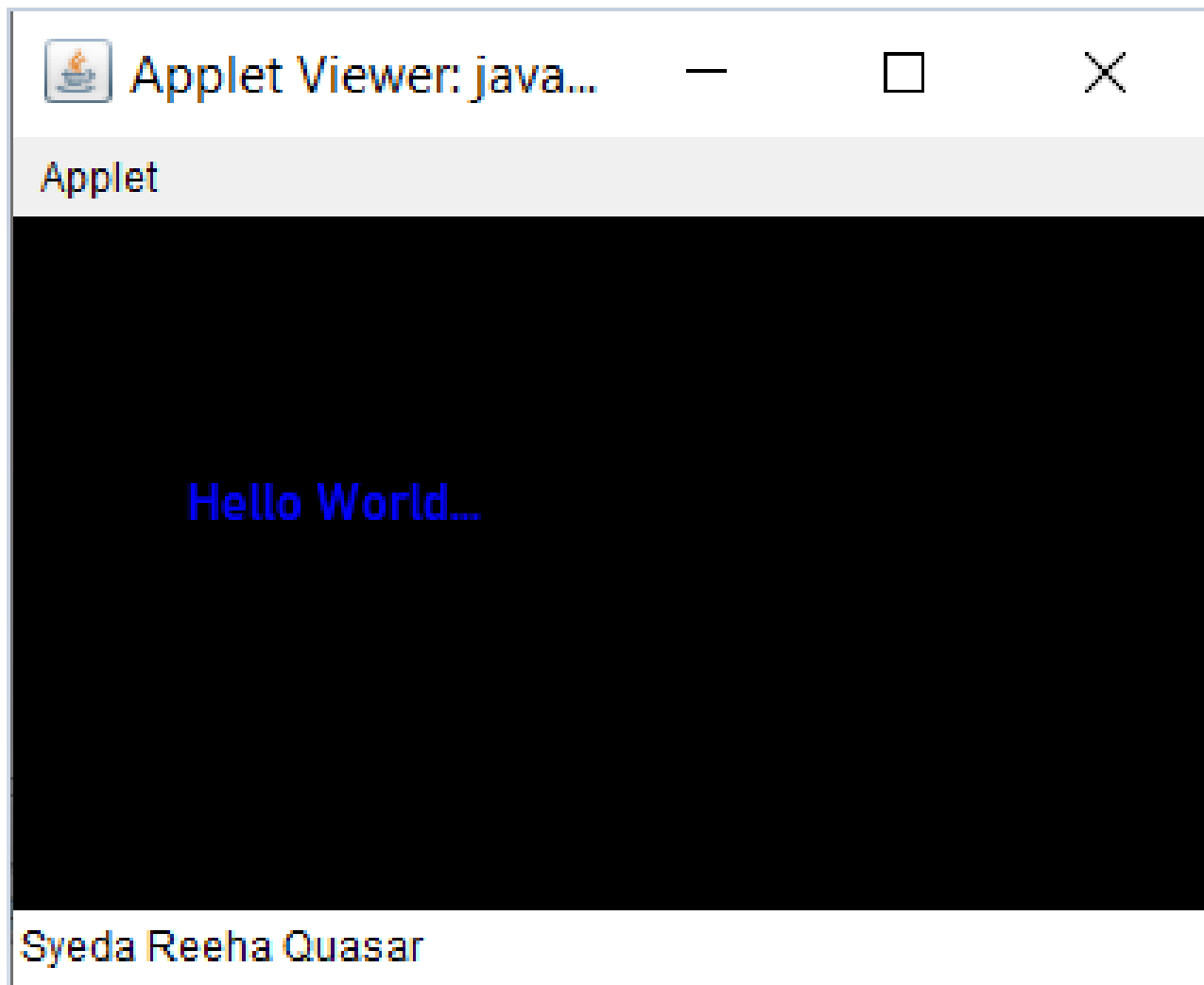
EXPERIMENT – 7.4

Aim:

Develop an analog clock using applet.

Theory:

Four methods in the Applet class give you the framework on which you build any serious applet:

- init: This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- start: This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- stop: This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- destroy: This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- paint: Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

Application Conversion to Applets: It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the java program launcher) into an applet that you can embed in a web page. Here are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.
- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.

- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.
- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object.the browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

The Applet CLASS:

Every applet is an extension of the java.applet.Applet class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context. These include methods that do the following:

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution.

The viewer may:

request information about the author, version and copyright of the applet

request a description of the parameters the applet recognizes

- initialize the applet
- destroy the applet
- start the applet's execution
- stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

Source Code:

```java
// Java program to illustrate
// analog clock using Applets

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Calendar;

public class clock extends Applet {

    @Override
    public void init()
    {
        // Applet window size & color
        this.setSize(new Dimension(800, 400));
        setBackground(new Color(50, 50, 50));
        new Thread() {
            @Override
            public void run()
            {
                while (true) {
                    repaint();
                    delayAnimation();
                }
            }
        }.start();
```

```java
        }

        // Animating the applet
        private void delayAnimation()
        {
                try {

                        // Animation delay is 1000 milliseconds
                        Thread.sleep(1000);
                }
                catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        // Paint the applet
        @Override
        public void paint(Graphics g)
        {
                // Get the system time
                Calendar time = Calendar.getInstance();

                int hour = time.get(Calendar.HOUR_OF_DAY);
                int minute = time.get(Calendar.MINUTE);
                int second = time.get(Calendar.SECOND);
                    Date date = time.getTime();
                    DateFormat dateFormat = new SimpleDateFormat("yyyy-mm-dd
hh:mm:ss");

                String clkString = dateFormat.format(date);

                // 12 hour format
                if (hour > 12) {
                        hour -= 12;
                }

                // Draw clock body center at (400, 200)
                g.setColor(Color.white);
                g.fillOval(300, 100, 200, 200);

                // Labeling
                g.setColor(Color.black);
                g.drawString("12", 390, 120);
                g.drawString("9", 310, 200);
                g.drawString("6", 400, 290);
                g.drawString("3", 480, 200);
```

```java
// Declaring variables to be used
double angle;
int x, y;

// Second hand's angle in Radian
angle = Math.toRadians((15 - second) * 6);

// Position of the second hand
// with length 100 unit
x = (int)(Math.cos(angle) * 100);
y = (int)(Math.sin(angle) * 100);

// Red color second hand
g.setColor(Color.red);
g.drawLine(400, 200, 400 + x, 200 - y);

// Minute hand's angle in Radian
angle = Math.toRadians((15 - minute) * 6);

// Position of the minute hand
// with length 80 unit
x = (int)(Math.cos(angle) * 80);
y = (int)(Math.sin(angle) * 80);

// blue color Minute hand
g.setColor(Color.blue);
g.drawLine(400, 200, 400 + x, 200 - y);

// Hour hand's angle in Radian
angle = Math.toRadians((15 - (hour * 5)) * 6);

// Position of the hour hand
// with length 50 unit
x = (int)(Math.cos(angle) * 50);
y = (int)(Math.sin(angle) * 50);

// Black color hour hand
g.setColor(Color.black);
g.drawLine(400, 200, 400 + x, 200 - y);
    g.setColor(Color.yellow);
    g.drawString(clkString, 10, 10);
    showStatus("Syeda Reeha Quasar");
    }
}
```
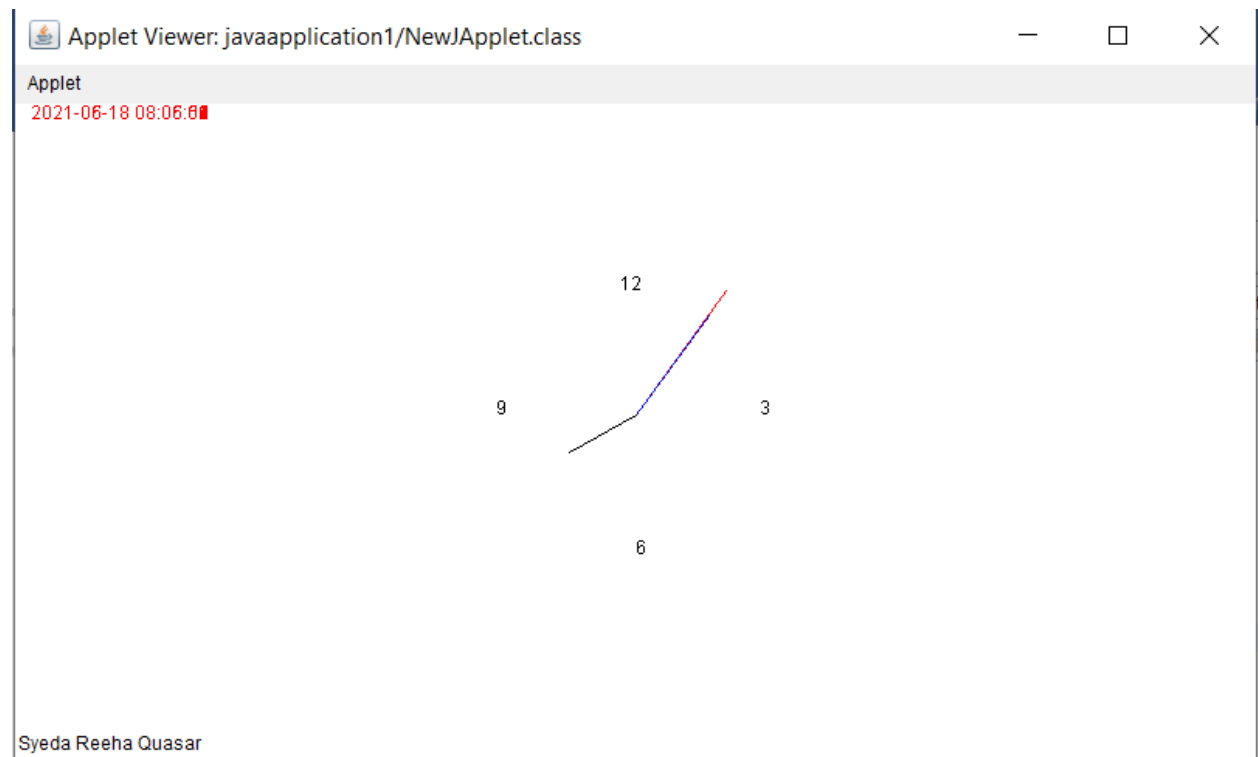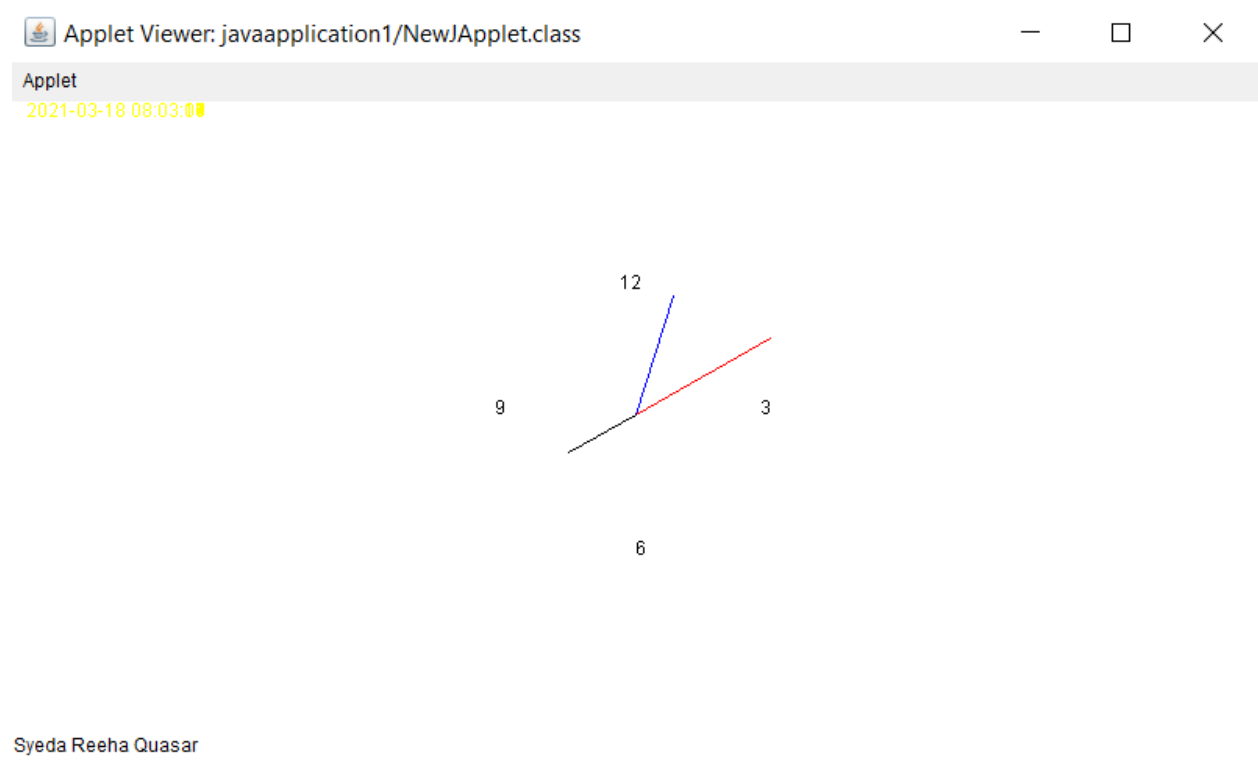
Viva Questions

1. What is the impact of declaring a method as final?

Ans.

A method declared as final can't be overridden. A sub-class can't have the same method signature with a different implementation.

2. How is final different from finally and finalize()?

Ans.

final is a modifier which can be applied to a class or a method or a variable. final class can't be inherited, final method can't be overridden and final variable can't be changed.

finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment.

finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

3. How will you communicate between two Applets?

Ans.

The simplest method is to use the static variables of a shared class since there's only one instance of the class and hence only one copy of its static variables. A slightly more reliable method relies on the fact that all the applets on a given page share the same AppletContext. We obtain this applet context as follows:

AppletContext ac = getAppletContext();

AppletContext provides applets with methods such as getApplet(name), getApplets(),getAudioClip, getImage, showDocument and showStatus().

4. Which classes can an applet extend?

Ans.

An applet can extend the java.applet.Applet class or the java.swing.JApplet class. The java.applet.Applet class extends the java.awt.Panel class and enables you to use the GUI tools in the AWT package. The java.swing.JApplet class is a subclass of java.applet.Applet that also enables you to use the Swing GUI tools.

5. How do you cause the applet GUI in the browser to be refreshed when data in it may have changed?

Ans.

You invoke the repaint() method. This method causes the paint method, which is required in any applet, to be invoked again, updating the display.

LAB - 8

Java Programming Lab

Topics Covered
Multithreading

Syeda Reeha Quasar

14114802719

4C7

EXPERIMENT – 8.1

Aim:
Write a java program to show multithreaded producer and consumer application.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Threads : Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

Multithreading : Multithreading in Java is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation, etc.

Wait() method : The wait() method is defined in object class which is the super most class in Java. This method tells the calling thread (Current thread) to give up the lock and go to sleep until some other thread enters the same monitor and calls notify() or notifyAll(). It is a final method , so we can't override it.

java.lang.Object.notifyAll() : It wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods. The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object. The awakened threads will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened threads enjoy no reliable privilege or disadvantage in being

the next thread to lock this object. This method should only be called by a thread that is the owner of this object's monitor.

Thread.sleep() : Thread.sleep() method can be used to pause the execution of current thread for specified time in milliseconds. The argument value for

milliseconds can't be negative, else it throws IllegalArgumentException

Source Code:

```java
package javaapplication1;


/**
 * @author reeha
 */


import java.lang.*;


public class ProducerConsumerTest {
    public static void main(String[] args) {
        Threadsys c = new Threadsys();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start();
        c1.start();
    }
}


class Threadsys {
    private int contents;
    private boolean available = false;
```

```java
    public synchronized int get() {

        while (available == false) {

            try {

                wait();

            } catch (InterruptedException e) {

            }

        }

        available = false;

        notifyAll();

        return contents;

    }


    public synchronized void put(int value) {

        while (available == true) {

            try {

                wait();

            } catch (InterruptedException e) {

            }

        }

        contents = value;

        available = true;

        notifyAll();

    }
}


class Consumer extends Thread {

    private Threadsys threadsys;

    private int number;
```

```java
    public Consumer(Threadsys c, int number) {

        threadsys = c;

        this.number = number;

    }


    public void run() {

        int value = 0;

        for (int i = 0; i <= 5; i++) {

            value = threadsys.get();

            System.out.println("Consumer " + this.number + " got : " + value);

        }

    }
}


class Producer extends Thread {

    private Threadsys Threadsys;

    private int number;


    public Producer(Threadsys c, int number) {

        Threadsys = c;

        this.number = number;

    }


    public void run() {

        for (int i = 0; i <= 5; i++) {

            Threadsys.put(i);

            System.out.println("Producer " + this.number + " put : " + i);

        }

    }
}
```
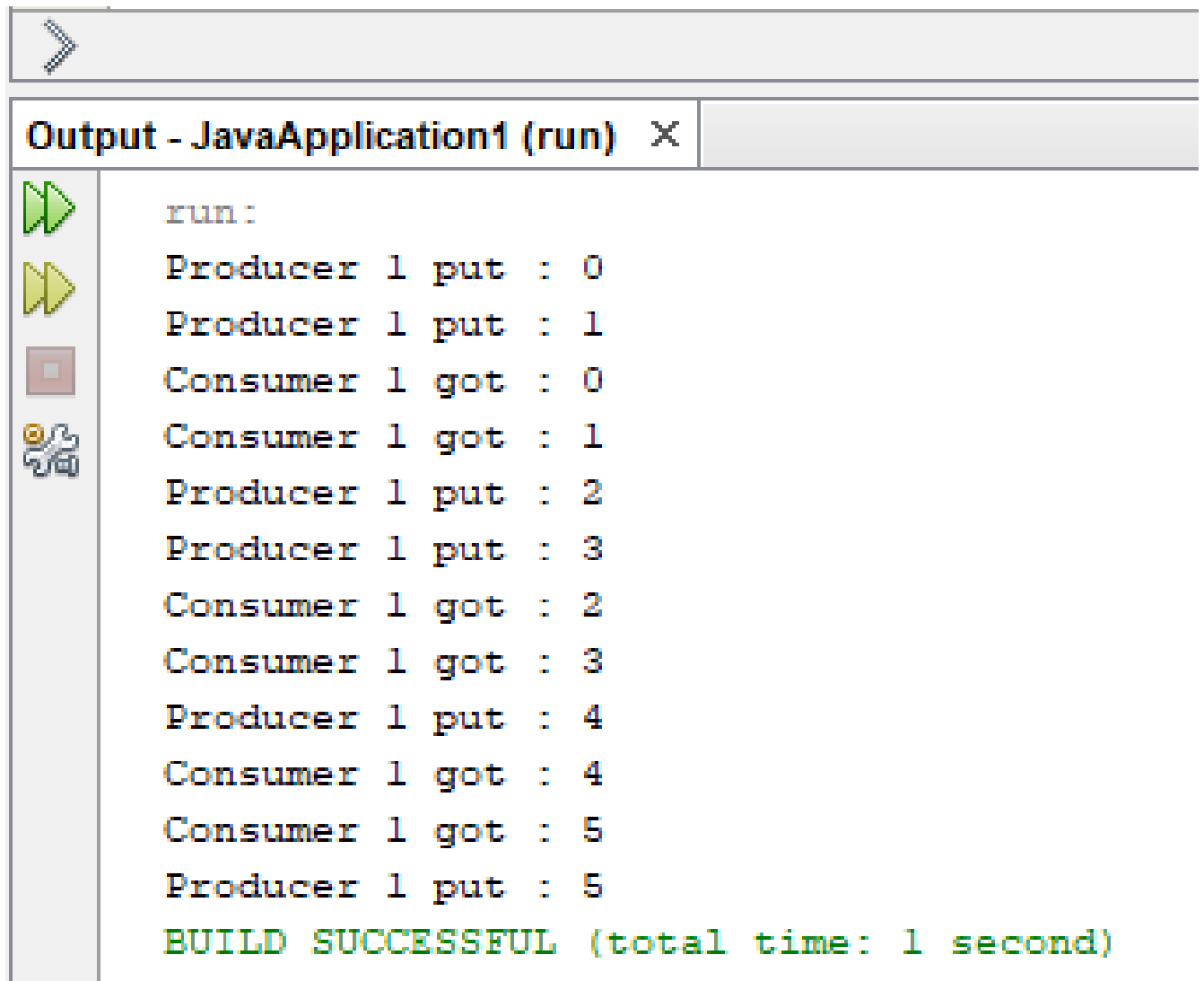
Output:

```
run:
Producer 1 put : 0
Producer 1 put : 1
Consumer 1 got : 0
Consumer 1 got : 1
Producer 1 put : 2
Producer 1 put : 3
Consumer 1 got : 2
Consumer 1 got : 3
Producer 1 put : 4
Consumer 1 got : 4
Consumer 1 got : 5
Producer 1 put : 5
BUILD SUCCESSFUL (total time: 1 second)
```

EXPERIMENT – 8.2

Aim:

Write an application that executes two threads. One thread displays "An" every 1000 milliseconds and other displays "B" every 3000 milliseconds. Create the threads by extending the Thread class.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Threads : Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

Multithreading : Multithreading in Java is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation, etc.

Thread.sleep() : Thread.sleep() method can be used to pause the execution of current thread for specified time in milliseconds. The argument value for

milliseconds can't be negative, else it throws IllegalArgumentException

Source Code:
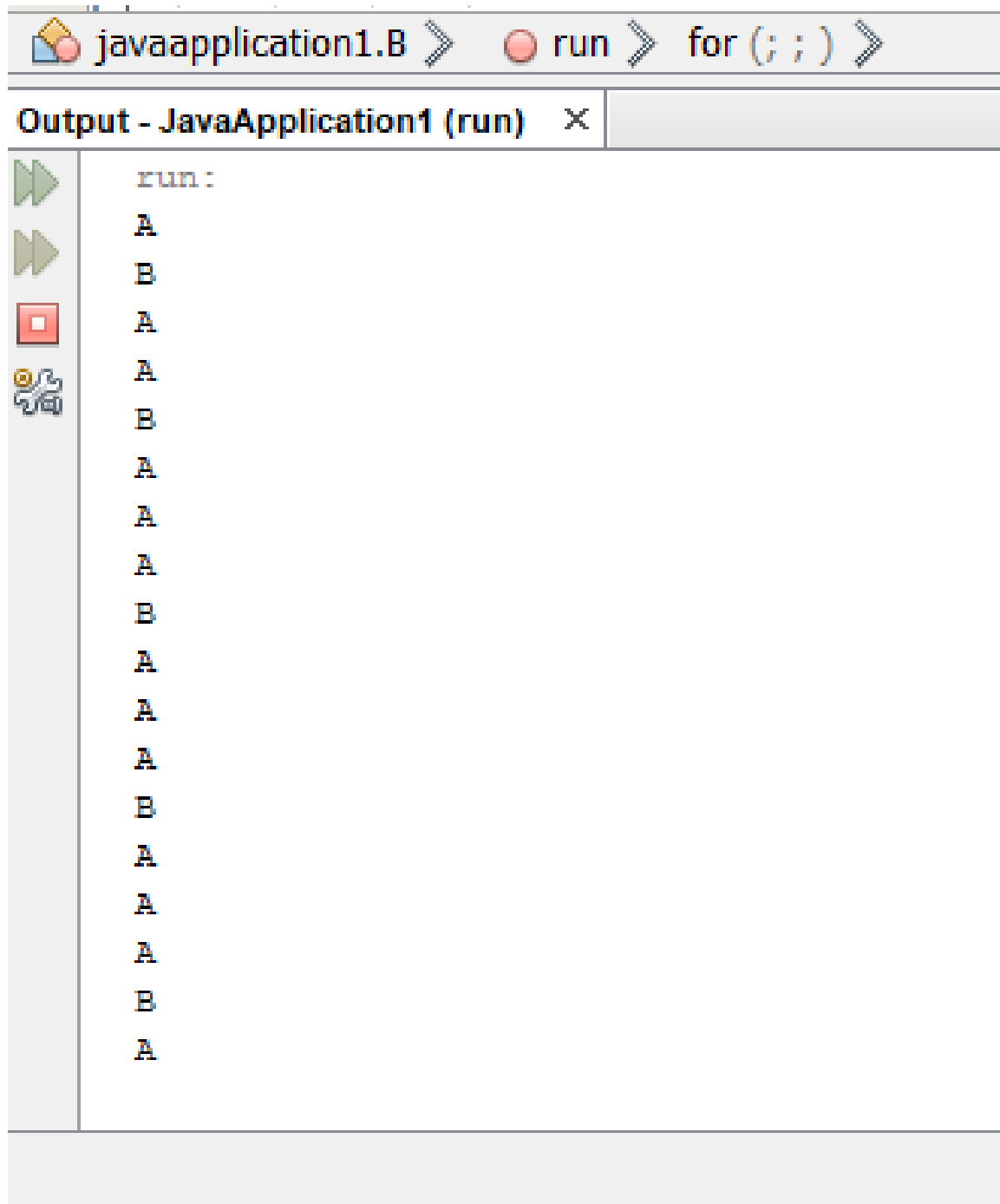
```
package javaapplication1;

/**
 *
 * @author reeha
 */

class A extends Thread {

    public void run() {
        for (;;) {
            System.out.println("A");
            try {
                Thread.currentThread().sleep(1000);
            }

            catch (InterruptedException e) {
            }
        }
    }
}

class B extends Thread {
    public void run() {
        for (;;) {
            System.out.println("B");
            try {
                Thread.currentThread().sleep(3000);
```

```
            } catch (InterruptedException e) {

            }

        }

    }

}


public class threading {

    public static void main(String args[]) {


        A obj1 = new A();

        B obj2 = new B();

        obj1.start();

        obj2.start();

    }

}
```

Output:

javaapplication1.B ❯ ● run ❯ for (; ;) ❯

Output - JavaApplication1 (run) ✕

```
run:
A
B
A
A
B
A
A
A
B
A
A
A
B
A
A
A
B
A
```

Viva Questions

1. What is multithreading?

Ans.

Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance. Its main advantages are:

- Threads share the same address space.

- The thread is lightweight.

- The cost of communication between the processes is low.

2. Differentiate between process and thread?

Ans.

There are the following differences between the process and thread.

A Program in the execution is called the process whereas; A thread is a subset of the process.

Processes are independent whereas threads are the subset of process.

Process have different address space in memory, while threads contain a shared address space.

Context switching is faster between the threads as compared to processes.

3. What is the purpose of wait() method in Java?

Ans.

The wait() method is provided by the Object class in Java. This method is used for inter-thread communication in Java. The java.lang.Object.wait() is used to pause the current thread, and wait until another thread does not call the notify() or notifyAll() method. Its syntax is given below.

public final void wait()

4. Explain different way of using thread?

Ans.

The thread could be implemented by using runnable interface or by inheriting from the Thread class. The former is more advantageous, 'cause when you are going for multiple inheritance..the only interface can help.

5. Describe synchronization in respect to multithreading.

Ans.

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchonization, it is possible for one thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to significant errors.

6. True of false: An applet can run multiple threads.

Ans.

True. The paint and update methods are always called from the AWT drawing and event handling thread. You can have your applet create additional threads, which is recommended for performing time-consuming tasks.

LAB - 9

Java Programming Lab

Topics Covered
AWT Components Event Handling

Syeda Reeha Quasar

14114802719

4C7

EXPERIMENT – 9.1

Aim:

WAP that illustrates how to process mouse click, enter, exit, press and release events. The background color changes when the mouse is entered, clicked, pressed, released or exited.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Java MouseListener Interface : The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

1. public abstract void mouseClicked(MouseEvent e);

2. public abstract void mouseEntered(MouseEvent e);

3. public abstract void mouseExited(MouseEvent e);

4. public abstract void mousePressed(MouseEvent e);

5. public abstract void mouseReleased(MouseEvent e);

Java Swing : It is a part of Java Foundation Classes (JFC) that is *used to create window- based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Java JFrame : The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like

labels, buttons, textfields are added to create a GUI . Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Source Code:

```
package javaapplication1;


import javax.swing.JApplet;

import java.awt.event.MouseListener;

import java.awt.event.MouseEvent;

import java.awt.Color;

import javax.swing.JFrame;

import java.awt.event.ActionListener;


/**
 *
 * @author REEHA
 */
public class mouseEvent extends JApplet {

    public void init() {}


    public static void main(String[] args) {
        JFrame frame = new JFrame();
        final int FRAME_WIDTH = 500;
        final int FRAME_HEIGHT = 600;
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
```

```java
class MouseEnterExitListener implements MouseListener {
    public void mouseEntered(MouseEvent event) {
        System.out.println("ENTER");
        frame.getContentPane().setBackground(Color.BLUE);
    }

    public void mouseExited(MouseEvent event) {
        System.out.println("EXIT");
        frame.getContentPane().setBackground(Color.RED);
    }

    public void mouseReleased(MouseEvent event) {
        System.out.println("RELEASED");
        frame.getContentPane().setBackground(Color.GREEN);
    }

    public void mouseClicked(MouseEvent event) {
        System.out.println("CLICKED");
        frame.getContentPane().setBackground(Color.BLACK);
    }

    public void mousePressed(MouseEvent event) {
        System.out.println("PRESSED");
        frame.getContentPane().setBackground(Color.YELLOW);
    }
}
MouseListener listener = new MouseEnterExitListener();
frame.addMouseListener(listener);
    }
}
```

Output:

Window of applet with different mouse events:
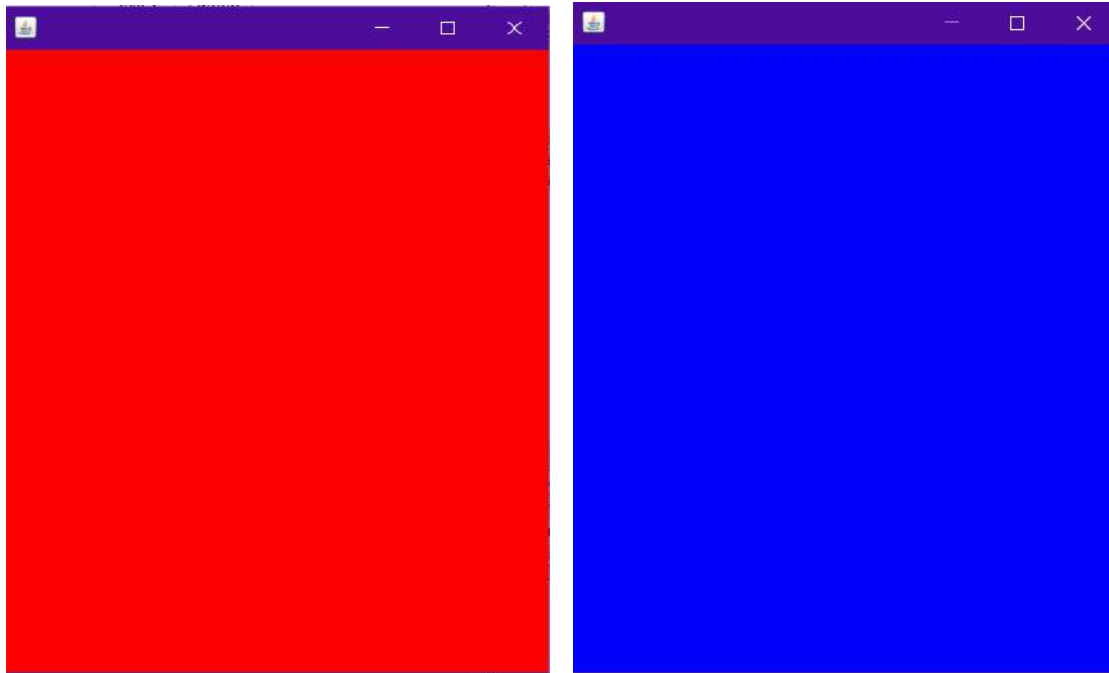
When mouse is out of the screen   When mouse in the applet window

javaapplication1.mouseEvent 〉	① main 〉 ⚫ MouseEnterExitListener 〉 ✕

```
Output - JavaApplication1 (run)  ✕
      PRESSED
      RELEASED
      CLICKED
      PRESSED
      RELEASED
      CLICKED
      EXIT
      ENTER
      PRESSED
      RELEASED
      EXIT
      ENTER
      PRESSED
      RELEASED
      CLICKED
      EXIT
      BUILD SUCCESSFUL (total time: 1 minute 52 seconds)
```

51:10   INS
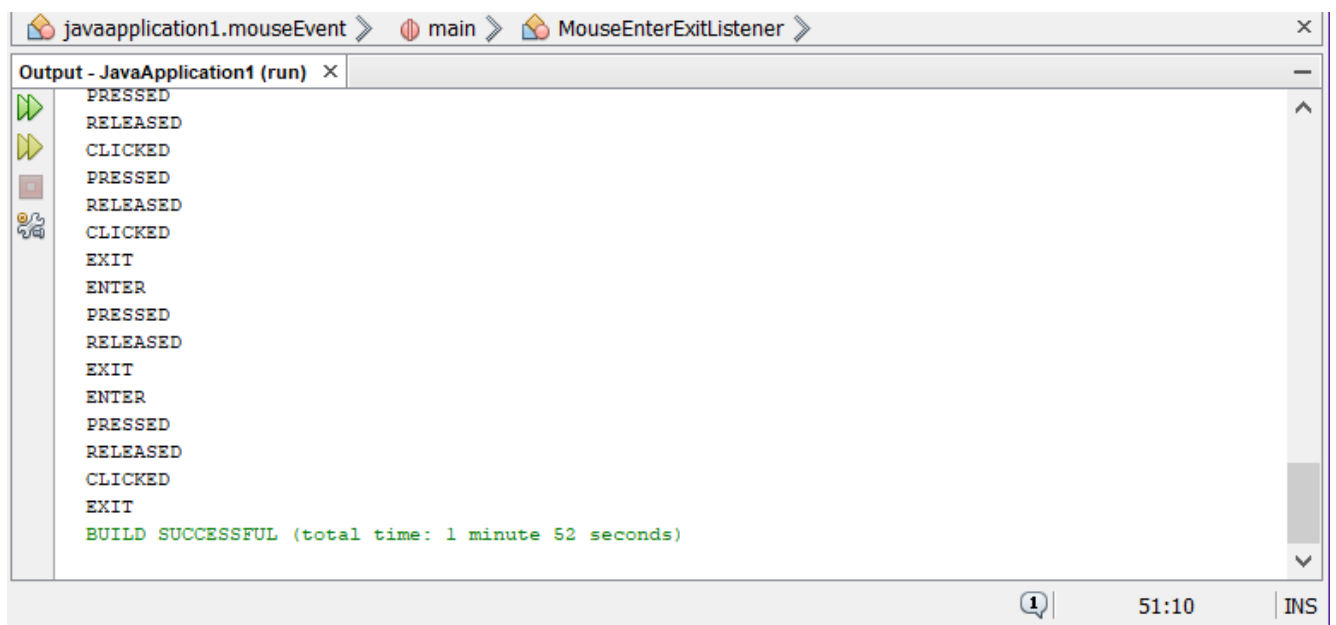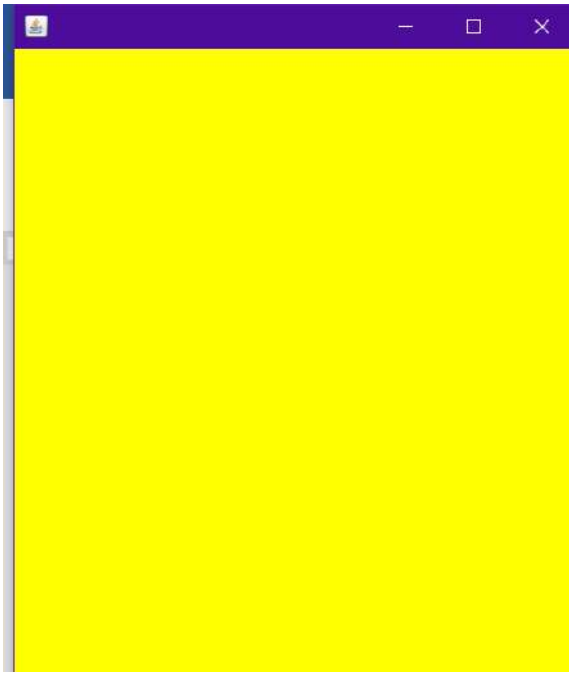
When mouse is pressed
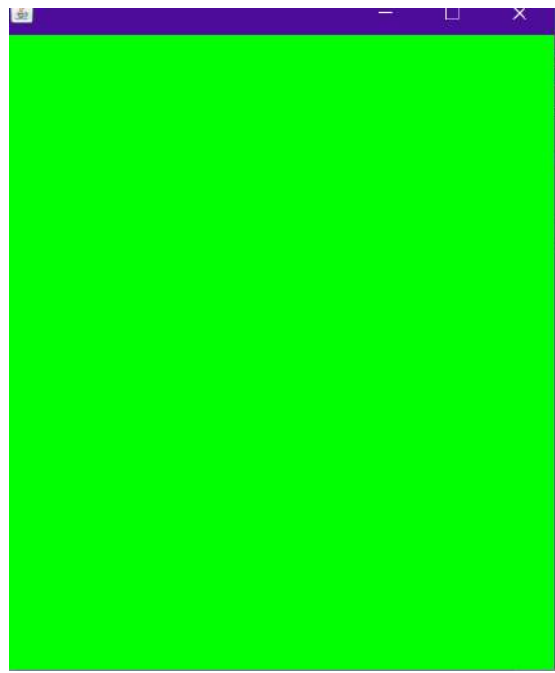
When mouse is released
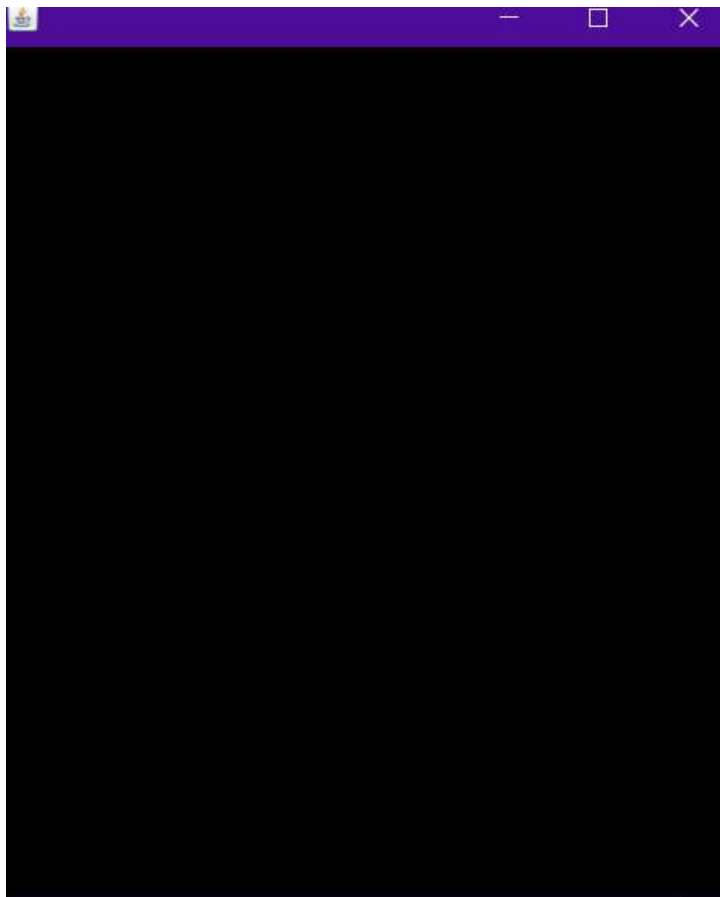
When mouse is clicked

On Mouse Entering
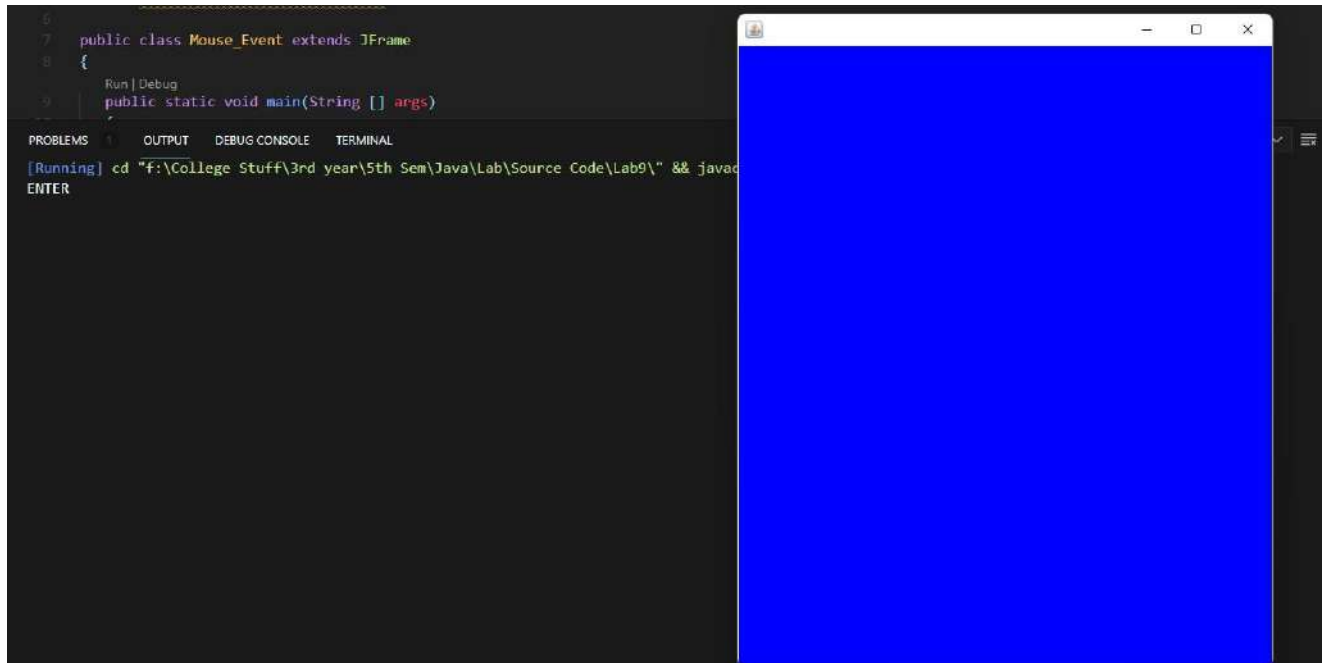
On Mouse Exiting

On Mouse Press


```
public class Mouse_Event extends JFrame
{
    Run | Debug
    public static void main(String [] args)
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

[Running] cd "f:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code\Lab9\" && javac
ENTER
EXIT
ENTER
PRESSED

On Mouse Click


```
public class Mouse_Event extends JFrame
{
    Run | Debug
    public static void main(String [] args)
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

[Running] cd "f:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code\Lab9\" && javac
ENTER
EXIT
ENTER
PRESSED
EXIT
ENTER
PRESSED
RELEASED
CLICKED
PRESSED
RELEASED
CLICKED

On Mouse Release

```
  7    public class Mouse_Event extends JFrame
  8    {
       Run | Debug
  9    public static void main(String [] args)
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

[Running] cd "f:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code\Lab9\" && javac
ENTER
EXIT
ENTER
PRESSED
EXIT
ENTER
PRESSED
RELEASED
CLICKED
PRESSED
RELEASED
CLICKED
EXIT
ENTER
PRESSED
EXIT
RELEASED

EXPERIMENT – 9.2

Aim:
WAP that displays your name whenever the mouse is clicked.

Theory:

Class : A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

Java MouseListener Interface : The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

1. public abstract void mouseClicked(MouseEvent e);

2. public abstract void mouseEntered(MouseEvent e);

3. public abstract void mouseExited(MouseEvent e);

4. public abstract void mousePressed(MouseEvent e);

5. public abstract void mouseReleased(MouseEvent e);

Java Swing : It is a part of Java Foundation Classes (JFC) that is *used to create window- based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Java JFrame : The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels,

buttons, textfields are added to create a GUI . Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Source Code:

```
package javaapplication1;


import javax.swing.JApplet;

import javax.swing.*;

import java.awt.*;

import java.awt.event.*;


/**
 *
 * @author reeha
 */
public class mouse_click implements MouseListener,ActionListener {

    static JFrame frame;
    static JTextField text;

    public static void main(String[] args) {

        frame = new JFrame("Display Name on mouse click");
        frame.setBackground(Color.white);
```

```java
        frame.setSize(500, 500);

        frame.setLayout(null);

        text = new JTextField();

        text.setBounds(0, 0, 500, 50);

        frame.add(text);

        JButton exit = new JButton("Exit");

        exit.setBounds(220, 235, 60, 30);

        frame.add(exit);

        mouse_click obj = new mouse_click();

        frame.addMouseListener(obj);

        exit.addActionListener(obj);

        frame.setVisible(true);

    }


    @Override
    public void actionPerformed(ActionEvent e) {

        frame.dispose();

    }


    @Override
    public void mouseEntered(MouseEvent e) {

        text.setText("");

    }


    @Override
    public void mouseExited(MouseEvent e) {

        text.setText("");
```

```java
    }


    @Override


    public void mouseReleased(MouseEvent e) {

        text.setText("");

    }


    @Override

    public void mousePressed(MouseEvent e) {

        text.setText("");

    }


    @Override

    public void mouseClicked(MouseEvent e) {

        text.setText("");

        text.setText("Syeda Reeha Quasar : 14114802719");

    }
}
```

Output:

No Mouse activity

Display Name on mouse click

Syeda Reeha Quasar : 14114802719
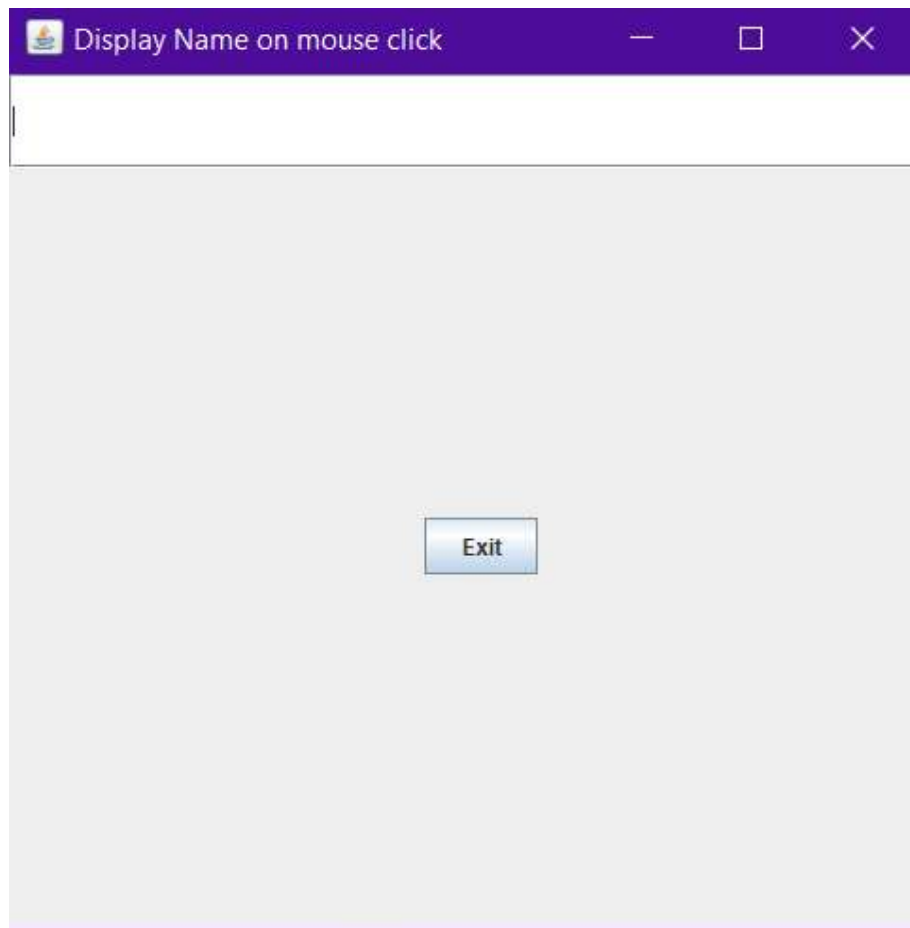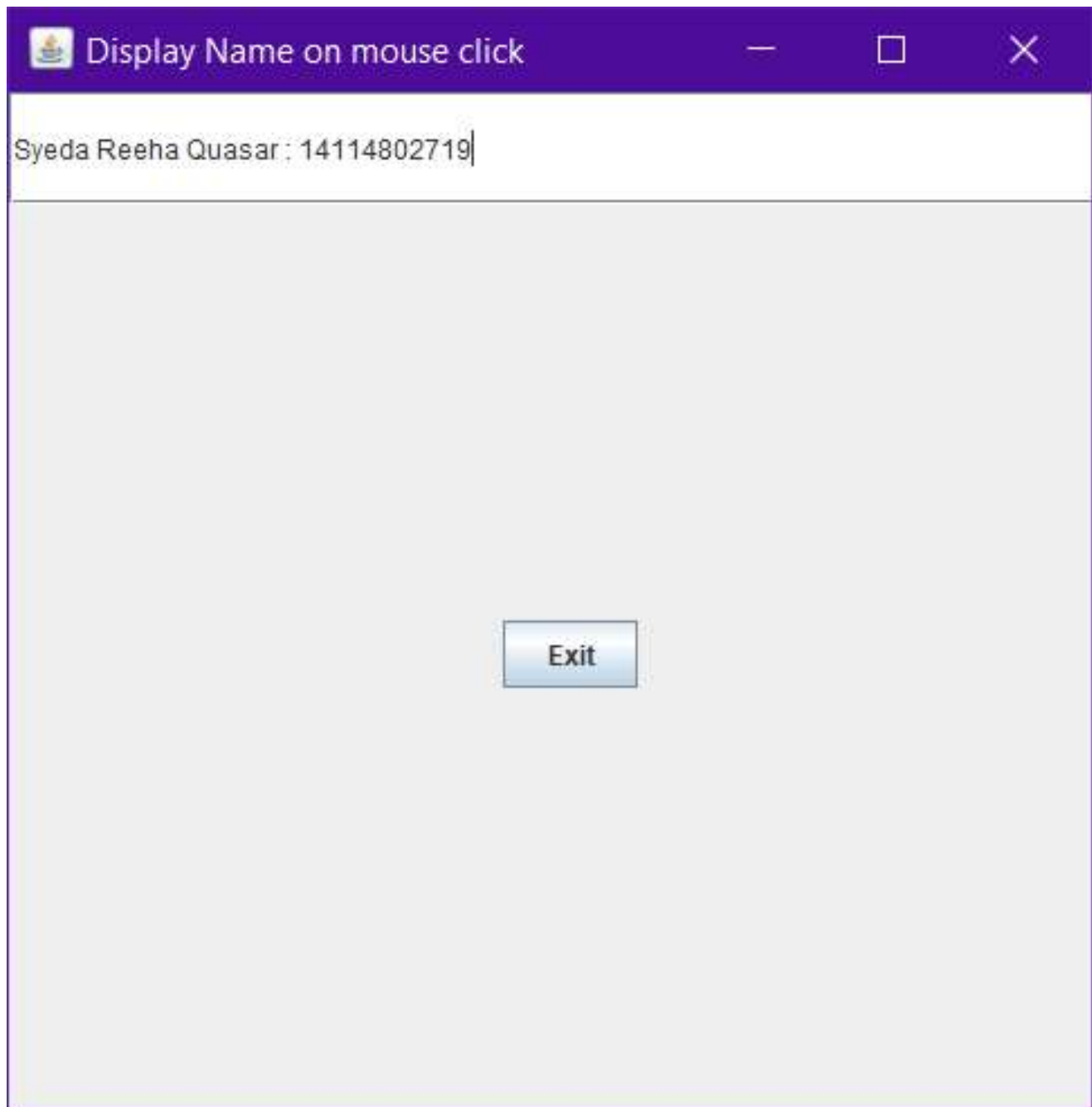
Exit

On Mouse Click

Exits as we press exit!

Viva Questions

1. Difference between Swing and Awt?

Ans.

AWT are heavy-weight componenets. Swings are light-weight components. Hence swing works faster than AWT.

2. What Are The Different Types Of Controls In Awt?

Ans.

The AWT supports the following types of controls:

Labels, Pushbuttons, Checkboxes, Choice lists, Lists, Scroll bars, Text componentsThese controls are subclasses of component.

3. How will you communicate between two Applets?

Ans.

The simplest method is to use the static variables of a shared class since there's only one instance of the class and hence only one copy of its static variables. A slightly more reliable method relies on the fact that all the applets on a given page share the same AppletContext. We obtain this applet context as follows:

AppletContext ac = getAppletContext();

AppletContext provides applets with methods such as getApplet(name), getApplets(),getAudioClip, getImage, showDocument and showStatus().

4. Which classes can an applet extend?

Ans.

An applet can extend the java.applet.Applet class or the java.swing.JApplet class. The java.applet.Applet class extends the java.awt.Panel class and enables you to use

the GUI tools in the AWT package. The java.swing.JApplet class is a subclass of java.applet.Applet that also enables you to use the Swing GUI tools.

5. What Are The Benefits Of Swing Over Awt?

Ans.

- Swing components are light weight.

- We can have a pluggable look and feel feature which shows us how they appear in other platforms.

- We can add images to Swing components. We have toolbars and tooltips in Swing.

LAB - 10

Java Programming Lab

Topics Covered

File Handling, JDBC

Syeda Reeha Quasar

14114802719

4C7

EXPERIMENT – 10.1

Aim:
Write a program that read from a file and write to file.

Theory:

FileReader : This class inherits from the InputStreamReader class. FileReader is used for reading streams of characters. This class has several constructors to create required objects. Following is the list of constructors provided by the FileReader class.

FileWriter : FileWriter is meant for writing streams of characters. For writing streams of raw bytes, consider using a FileOutputStream. FileWriter creates the output file if it is not present already.

.read() method : The read() method of Reader Class in Java is used to read a single character from the stream.

Write() method : The read() method of Reader Class in Java is used to read a single character from the stream.

Source Code:
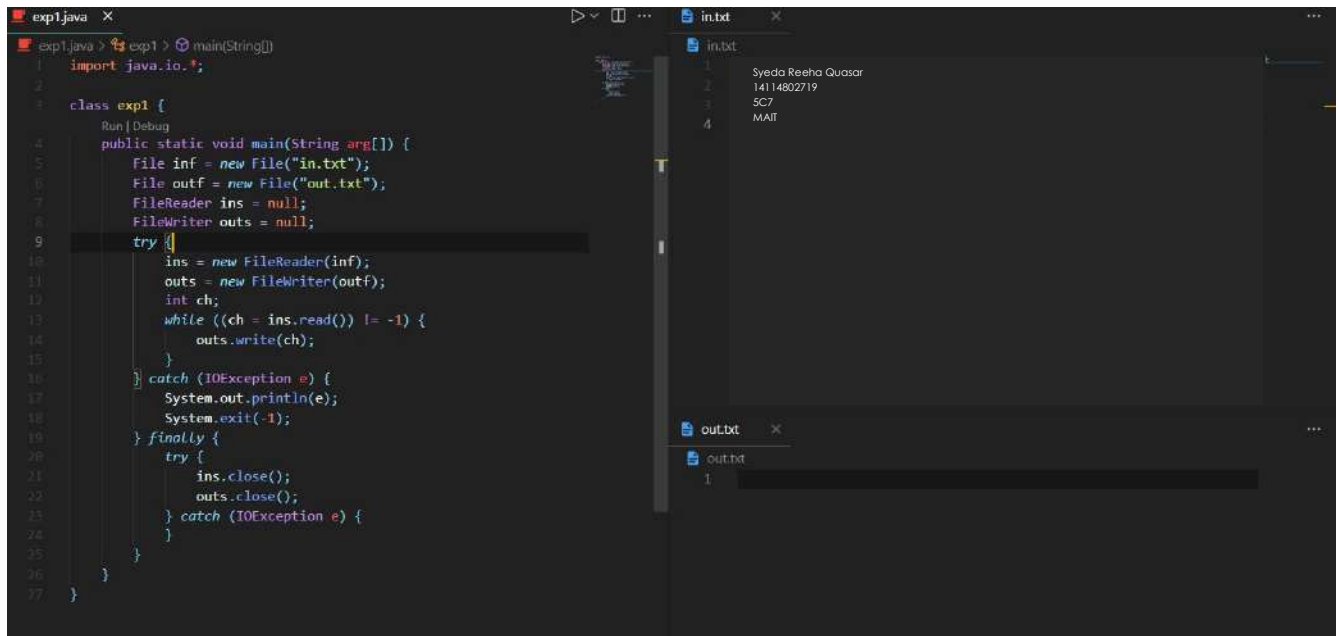
```java
package javaapplication1;

import java.io.*;


/**
 *
 * @author reeha
 */
```

```java
public class IOFile {

    public static void main(String arg[]) {

        File inf = new File("in.txt");

        File outf = new File("out.txt");

        FileReader ins = null;

        FileWriter outs = null;

        try {

            ins = new FileReader(inf);

            outs = new FileWriter(outf);

            int ch;

            while ((ch = ins.read()) != -1) {

                outs.write(ch);

            }

        } catch (IOException e) {

            System.out.println(e);

            System.exit(-1);

        } finally {

            try {

                ins.close();

                outs.close();

            } catch (IOException e) {

            }

        }

    }

}
```

Output:

Before :

After :

EXPERIMENT – 10.2

Aim:
Convert the content of a given file into the uppercase content of the same file.

Theory:

FileReader : This class inherits from the InputStreamReader class. FileReader is used for reading streams of characters. This class has several constructors to create required objects. Following is the list of constructors provided by the FileReader class.

FileWriter : FileWriter is meant for writing streams of characters. For writing streams of raw bytes, consider using a FileOutputStream. FileWriter creates the output file if it is not present already.

Buffered Reader : Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

Buffered Writer : Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.

Source Code:

```
package javaapplication1;

/**
 *
 * @author reeha
 */

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Files;
import java.nio.file.Path;

public class exp2 {
    public static void main(String[] arguments) {
        String sourceName = "in.txt";
        try {
            Path source = FileSystems.getDefault().getPath(sourceName);
            Path temp = FileSystems.getDefault().getPath("tmp_" +
sourceName);
            FileReader fr = new FileReader(source.toFile());

            BufferedReader in = new BufferedReader(fr);
            FileWriter fw = new FileWriter(temp.toFile());
            BufferedWriter out = new BufferedWriter(fw);
```
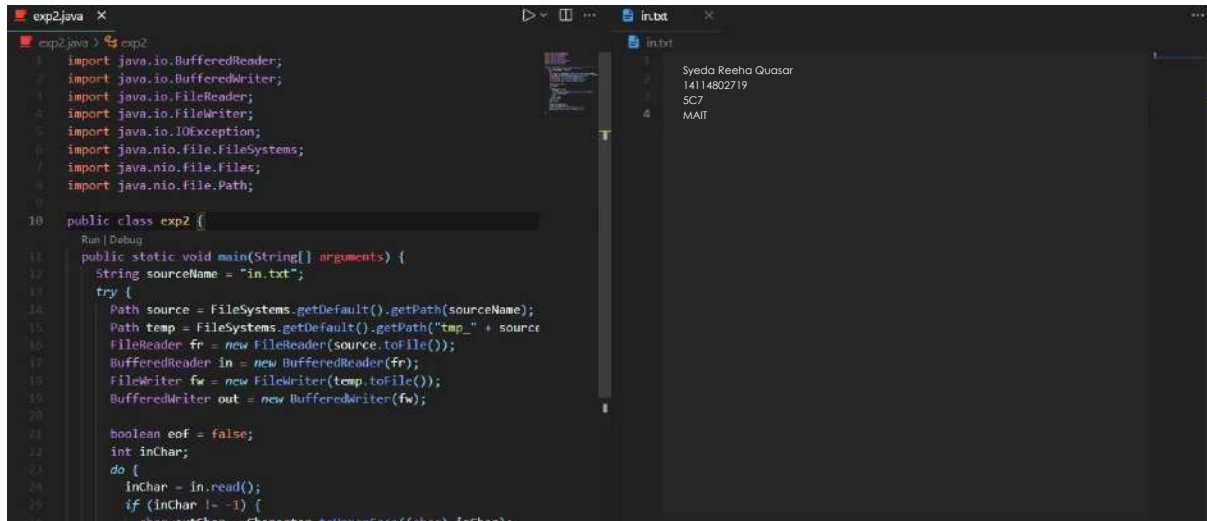
```java
            boolean eof = false;

            int inChar;

            do {

                inChar = in.read();

                if (inChar != -1) {

                    char outChar = Character.toUpperCase((char) inChar);

                    out.write(outChar);

                } else

                    eof = true;

            } while (!eof);

            in.close();

            out.close();


            Files.delete(source);

            Files.move(temp, source);

        } catch (IOException | SecurityException se) {

            System.out.println("Error" + se.toString());

        }

    }

}
```
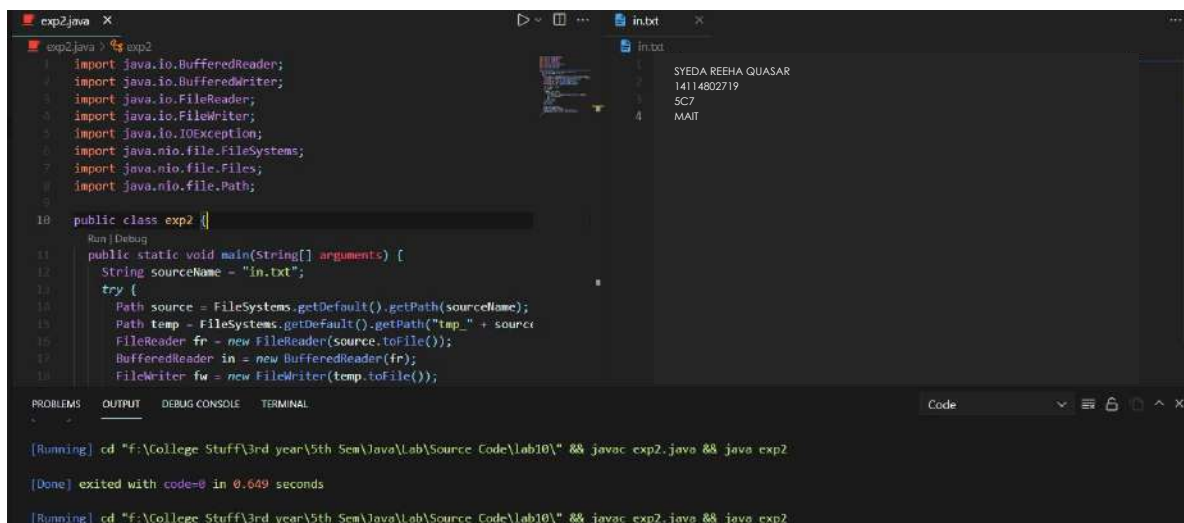
Output:

Before :

After :

EXPERIMENT – 10.3

Aim:
JDBC (Database connectivity with MS-Access)

Theory:

 FileReader : This class inherits from the InputStreamReader class. FileReader is used for reading streams of characters. This class has several constructors to create required objects. Following is the list of constructors provided by the FileReader class.

FileWriter : FileWriter is meant for writing streams of characters. For writing streams of raw bytes, consider using a FileOutputStream. FileWriter creates the output file if it is not present already.

Buffered Reader : Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

Buffered Writer : Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.

Source Code:

```
package javaapplication1;


/**
 *
 * @author reeha
 */
```

```java
import java.sql.*;

public class exp3 {
    public static void main(String[] args) {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection conn = DriverManager.getConnection("jdbc:odbc:Test");
            Statement st = conn.createStatement();
            String sql = "Select * from Table1";
            ResultSet rs = st.executeQuery(sql);
            while (rs.next()) {
                System.out.println("\n" + rs.getString(1) + "\t" +
rs.getString(2) + "\t" + rs.getString(3) + "\t"
                        + rs.getString(4));
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
[Running] cd "f:\College Stuff\3rd year\5th Sem\Java\Lab\Source Code\lab10\" && javac exp3.java && jav
sun.jdbc.odbc.JdbcOdbcDriver
ID: 1, Name: Udit, Age: 22, Branch: cse
ID: 2, Name: Pratyaksh, Age: 21, Branch: ece
ID: 3, Name: Ankit, Age: 22, Branch: mae
ID: 3, Name: Ahmad, Age: 20, Branch: it
```

Viva Questions

1. What is JDBC in Java?

Ans.

JDBC(Java Database Connectivity) is a Java API, which is helpful in interaction with the database to retrieve, manipulate and process the data using SQL. It will make use of JDBC drivers for connecting with the database. By using JDBC, we can access tabular data stored in various types of relational databases such as Oracle, MySQL, MS Access, etc.

2. What is a stream and its types and classes?

Ans.

A Stream is an abstraction that either produces or consumes information. There are two types of Streams :

Byte Streams: Provide a convenient means for handling input and output of bytes. Character Streams: Provide a convenient means for handling input & output of characters.

Byte Streams classes: Are defined by using two abstract classes, namely InputStream and OutputStream.

Character Streams classes: Are defined by using two abstract classes, namely Reader and Writer.

3. What is the difference between InputStream and OutputStream in Java?

Ans.

InputStream is used to read data from sources like File, Socket, or Console, while OutputStream is used to write data into a destination like a File, Socket, or Console.

4. What is ODBC Bridge driver ?

Ans.

In this, the JDBC–ODBC bridge acts as an interface between the client and database server. When a user uses a Java application to send requests to the database using JDBC– ODBC bridge, it converts the JDBC API into ODBC API and then sends it to the database. When the result is received from the database, it is sent to ODBC API and then to JDBC API.It is platform-dependent because it uses ODBC which depends on the native library of the operating system. In this, JDBC–ODBC driver should be installed in every client system and database must support for ODBC driver. It is easier to use but it gives low performance because it involves the conversion of JDBC method calls to the ODBC method calls

5. Describe synchronization in respect to multithreading.

Ans.

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchonization, it is possible for one thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to significant errors.

LAB - 11

Java Programming Lab

Topics Covered
Swings

Syeda Reeha Quasar
14114802719
4C7

EXPERIMENT – 11.1

Aim:
Create runnable jar file in java.

Theory:

Java swing : Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

JFrame : The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Source Code:

```java
import javax.swing.*;


public class IOFile {

    IOFile() {

        JFrame f = new JFrame("My Jar File");


        JButton b = new JButton("Click Me!");

        b.setBounds(100, 150, 100, 40);

        JLabel l1 = new JLabel("Syeda Reeha Quasar : 14114802719");

        l1.setBounds(100, 50, 170, 100);
```

```
        f.add(b);

        f.add(l1);

        f.setSize(300, 400);

        f.setLayout(null);

        f.setVisible(true);


        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }


    public static void main(String[] args) {

        new IOFile();

    }
}
```

Output:

EXPERIMENT – 11.2

Aim:

Display image on a button in swing.

Theory:

Java swing : Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

JFrame : The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method

Jbutton : The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Source Code:

```
import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.SwingUtilities;

import javax.swing.WindowConstants;
```

```java
import java.awt.FlowLayout;

public class exp2 extends JFrame {
    public exp2() {
        initComponents();
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new exp2().setVisible(true));
    }

    private void initComponents() {

        setTitle("Image inside Button");
        setSize(500, 500);

        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(new FlowLayout(FlowLayout.CENTER));

        JButton helloButton = new JButton("Hello", new ImageIcon(

                this.getClass().getResource("/images/hello.png")));
        helloButton.setMnemonic('H');

        getContentPane().add(helloButton);
    }
}
```

Output:
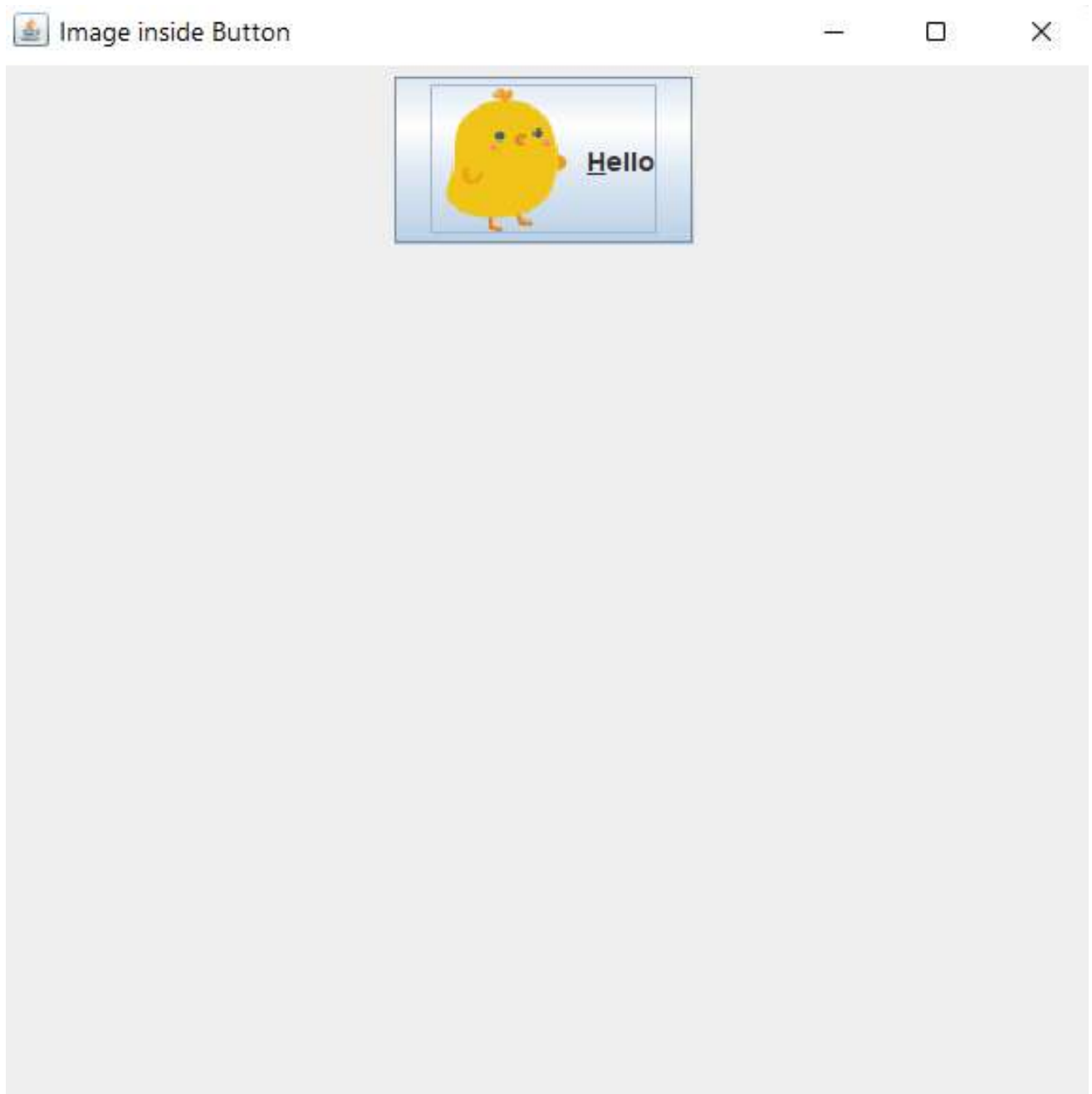
EXPERIMENT – 11.3

Aim:
Change the component color by choosing a color from ColorChooser.

Theory:

Java swing : Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

JFrame : The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method

Jbutton : The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JColorChooser : The JColorChooser class is used to create a color chooser dialog box so that user can select any color. It inherits JComponent class.

Source Code:

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
```

```java
public class IOFile extends
        JFrame implements ActionListener {
    JButton b = new JButton("color");
    protected JLabel label;
    Container c = getContentPane();

    IOFile() {
        label = new JLabel("Syeda Reeha Quasar : 14114802719",
JLabel.CENTER);
        label.setForeground(Color.BLACK);
        label.setBackground(Color.WHITE);
        label.setOpaque(true);
        label.setFont(new Font("SansSerif", Font.BOLD, 25));

        c.setLayout(new FlowLayout());
        b.addActionListener(this);
        c.add(label);
        c.add(b);
    }

    public void actionPerformed(ActionEvent e) {
        Color initialcolor = Color.RED;
        Color color = JColorChooser.showDialog(this, "Select a color",
initialcolor);
        label.setForeground(color);
    }

    public static void main(String[] args) {
        IOFile ch = new IOFile();
        ch.setSize(400, 400);
        ch.setVisible(true);
        ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Output:
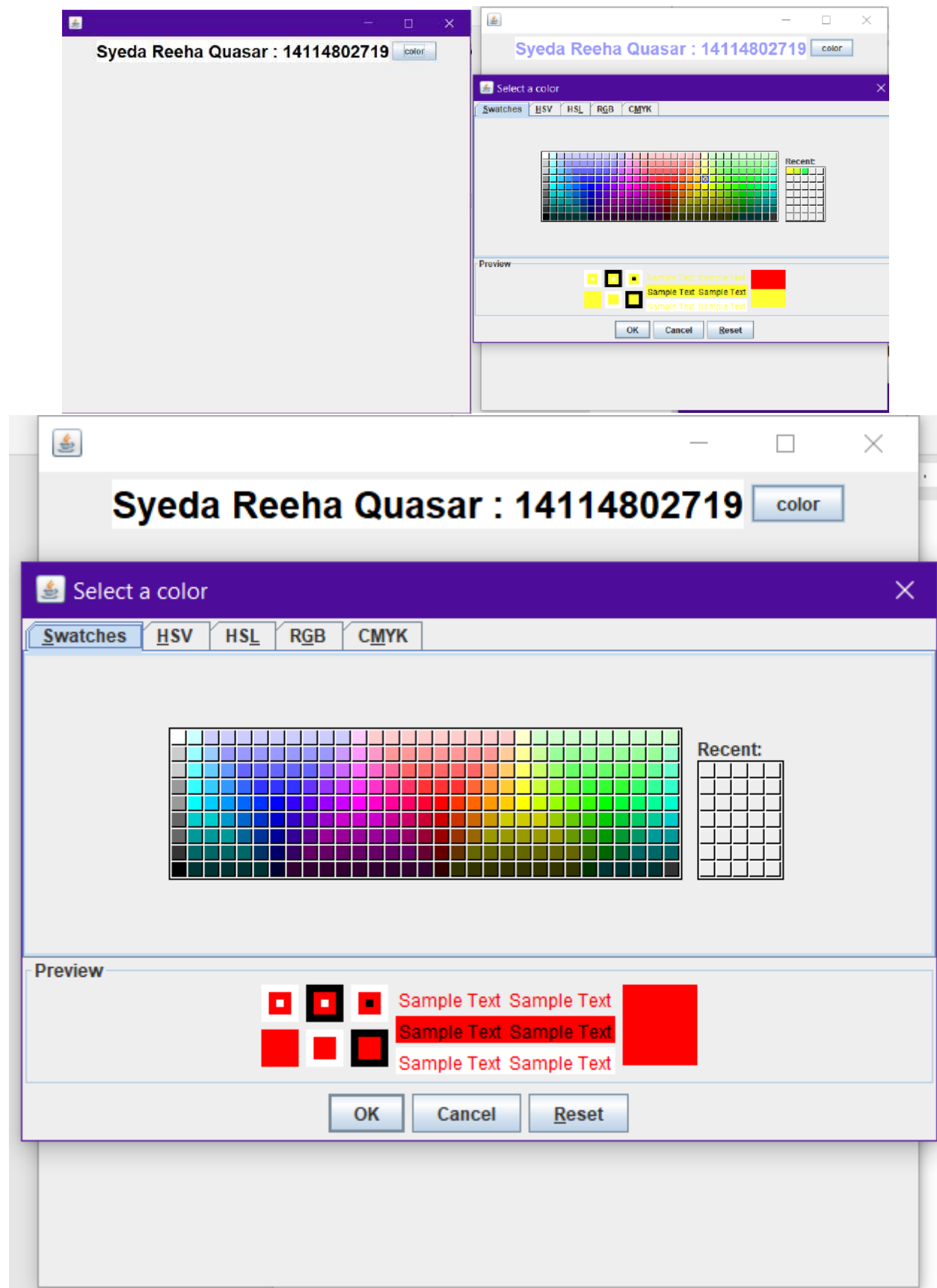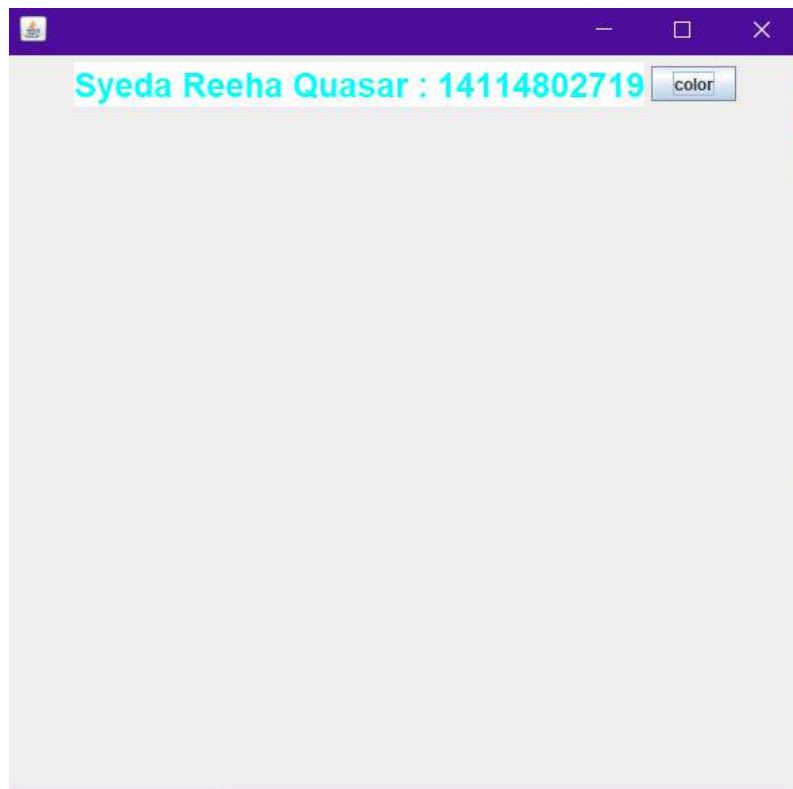
EXPERIMENT – 11.4

Aim:
Display the digital watch in swing tutorial.

Theory:

Java swing : Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Thread : Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

JFrame : The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method

Jbutton : The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Source Code:

```
import javax.swing.*;
import java.awt.*;
import java.text.*;
import java.util.*;
```

```java
public class IOFile implements Runnable {
    JFrame f;
    Thread t = null;
    int hours = 0, minutes = 0, seconds = 0;
    String timeString = "";
    JButton b;
    JLabel l1;

    IOFile() {
        f = new JFrame();

        t = new Thread(this);
        t.start();
        b = new JButton();
        b.setBounds(100, 100, 100, 50);
        l1 = new JLabel("Reeha : 14114802719");
        l1.setBounds(100, 10, 170, 100);

        f.add(l1);
        f.add(b);
        f.setSize(300, 400);
        f.setLayout(null);
        f.setVisible(true);
    }

    public void run() {
        try {
            while (true) {

                Calendar cal = Calendar.getInstance();
                hours = cal.get(Calendar.HOUR_OF_DAY);
                if (hours > 12)
                    hours -= 12;
                minutes = cal.get(Calendar.MINUTE);
                seconds = cal.get(Calendar.SECOND);

                SimpleDateFormat formatter = new
SimpleDateFormat("hh:mm:ss");
                Date date = cal.getTime();
                timeString = formatter.format(date);
                printTime();
                t.sleep(1000);
            }
        } catch (Exception e) {
```

```
        }
    }

    public void printTime() {
        b.setText(timeString);
    }

    public static void main(String[] args) {
        new IOFile();

    }
}
```

Output:

EXPERIMENT – 11.5

Aim:
Create a notepad in swing.

Theory:

Java swing : Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

JFrame : The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method

Jbutton : The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Source Code:

```
import java.awt.*;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;
import java.awt.event.*;
import java.io.File;
import java.io.PrintWriter;
import java.util.Scanner;
```

```java
import javax.swing.*;

public class Notepad extends JFrame {

    private static final long serialVersionUID = 1L;
    JFrame frame;
    JMenuBar menuBar;
    JMenu file;
    JMenu edit;
    JMenuItem open, newFile, save, exit;
    JMenuItem undo, paste, selectAll;
    JMenu format;
    JMenu help;
    JFileChooser fileChooser;
    JTextArea textArea;
    Clipboard clip;

    Notepad() {
        frame = new JFrame("Notepad Application");
        file = new JMenu("File");
        edit = new JMenu("Edit");
        format = new JMenu("Format");
        help = new JMenu("Help");

        newFile = new JMenuItem("New");
        open = new JMenuItem("Open");
        save = new JMenuItem("Save");
        exit = new JMenuItem("Exit");
        undo = new JMenuItem("Undo                Ctrl+Z");
        paste = new JMenuItem("Paste               Ctrl+V");
        selectAll = new JMenuItem("Select All      Ctrl+A ");
        textArea = new JTextArea();
        fileChooser = new JFileChooser();
        menuBar = new JMenuBar();

        frame.setLayout(new BorderLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.add(textArea);
        file.add(open);
        file.add(newFile);
        file.add(save);
        file.add(exit);
        edit.add(undo);
```

```
            edit.add(paste);
            edit.add(selectAll);
            menuBar.add(file);
            menuBar.add(edit);
            menuBar.add(format);
            menuBar.add(help);

            frame.setJMenuBar(menuBar);

            OpenListener openL = new OpenListener();
            NewListener NewL = new NewListener();
            SaveListener saveL = new SaveListener();
            ExitListener exitL = new ExitListener();
            open.addActionListener(openL);
            newFile.addActionListener(NewL);
            save.addActionListener(saveL);
            exit.addActionListener(exitL);
            // UndoListener UndoL = new UndoListener();
            PasteListener pasteL = new PasteListener();
            // EditListener EditL = new EditListener();
            // SelectListener SelectL = new SelectListener();
            // undo.addActionListener(UndoL);
            // paste.addActionListener(EditL);
            // selectAll.addActionListener(SelectL);
            frame.setSize(800, 600);
            frame.setVisible(true);
        }

    class OpenListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (JFileChooser.APPROVE_OPTION == fileChooser.showOpenDialog(frame))
{
                File file = fileChooser.getSelectedFile();
                textArea.setText("");
                Scanner in = null;
                try {
                    in = new Scanner(file);
                    while (in.hasNext()) {
                        String line = in.nextLine();
                        textArea.append(line + "\n");
                    }
                } catch (Exception ex) {
                    ex.printStackTrace();
                } finally {
                    in.close();
```

```java
                }
            }
        }
    }

    class SaveListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (JFileChooser.APPROVE_OPTION == fileChooser.showSaveDialog(frame))
{
                File file = fileChooser.getSelectedFile();
                PrintWriter out = null;
                try {
                    out = new PrintWriter(file);
                    String output = textArea.getText();
                    System.out.println(output);
                    out.println(output);
                } catch (Exception ex) {
                    ex.printStackTrace();
                } finally {
                    try {
                        out.flush();
                    } catch (Exception ex1) {

                    }
                    try {
                        out.close();
                    } catch (Exception ex1) {

                    }
                }
            }
        }
    }

    class NewListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            textArea.setText("");
            // frame.add(newFile);
            // textArea.(newFile+"\n");

        }
    }

    class ExitListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
```

```
                    System.exit(0);
            }
        }

    class PasteListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            Transferable cliptran = clip.getContents(Notepad.this);
            try {
                String sel = (String)
cliptran.getTransferData(DataFlavor.stringFlavor);
                textArea.replaceRange(sel, textArea.getSelectionStart(),
textArea.getSelectionEnd());
            } catch (Exception exc) {
                System.out.println("not string flavour");
            }

        }
    }

    public static void main(String args[]) {
        Notepad n = new Notepad();
    }
}
```

Output:

Notepad Application   —   ☐   ✕

File   Edit   Format   Help

Syeda Reeha Quasar

14114802719

Notepad Application   —   ☐   ✕

File   Edit   Format   Help

Syeda Reeha Quasar

14114802719

Save   ✕

Save In: ☐ Documents   ▼

☐ Adobe   ☐ Rockstar Games
☐ Aura Profiles   ☐ Sound recordings
☐ Custom Office Templates ☐ xgen
☐ My Games   ☐ Udit's Notebook.url
☐ NetBeansProjects
☐ OneNote Notebooks
☐ Python Scripts

File Name: [   ]
Files of Type: All Files   ▼

                                    Save   Cancel
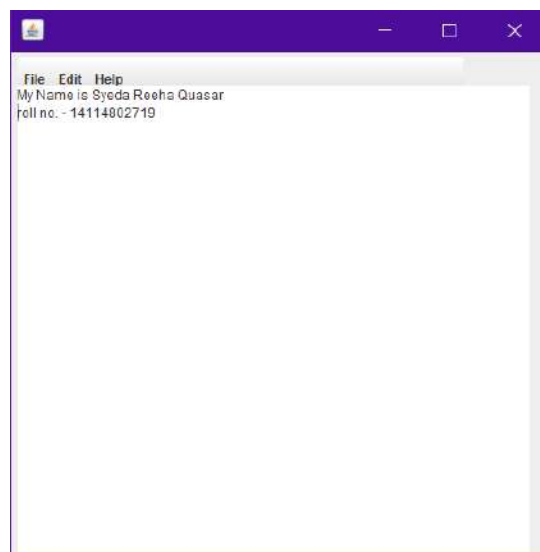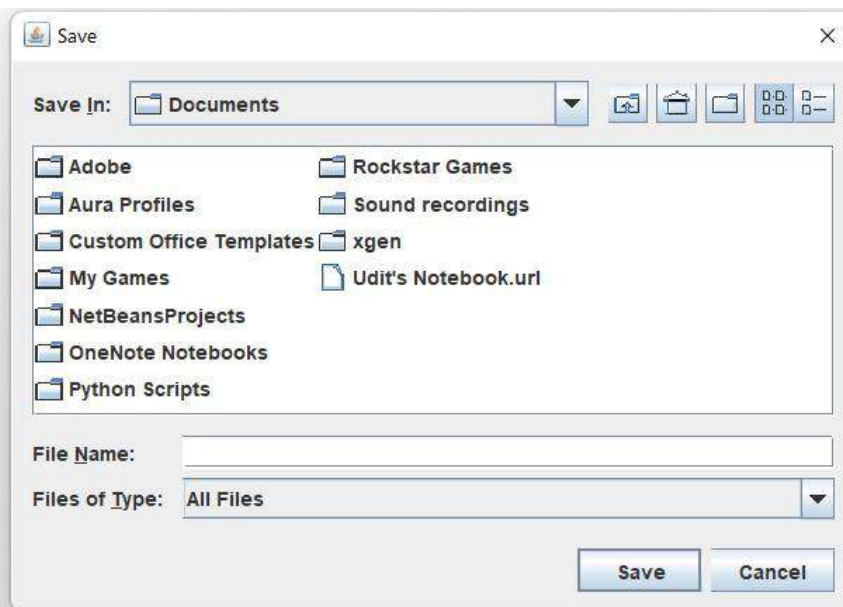
Viva Questions

1. What is Java Swing?

Ans.

It is a part of JFC (Java Foundation Classess) that is used to create window-based applications.

Java Swing components are platform independent and lightweight .

2. What are the methods of component class in Java Swing?

Ans.

There are four types of methods of component class are:

- public void add(Component c)
- public void setSize(int width, int height)
- public void setLayout(LayoutManager m)
- public void setVisible(boolean b)

3. How many ways to create a frame in Java Swing ? Ans.

There are two ways to create a frame:

- By Association(creating the object of Frame class)
- By Inheritance(extending Frame class)

4. What are differences between Swing and AWT?

Ans.

There is couple of differences between swing and AWT.

- AWT component are considered to be heavyweight while Swing component are lightweights.

- Swing has plug gable look and feel.

- AWT is platform dependent same GUI will look different platform while Swing is developed in Java and is platform dependent.

5. True of false: An applet can run multiple threads.

Ans.

True. The paint and update methods are always called from the AWT drawing and event handling thread. You can have your applet create additional threads, which is recommended for performing time-consuming tasks.

LAB - 12

Java Programming Lab

Topics Covered
Servlet

Syeda Reeha Quasar
14114802719
4C7

EXPERIMENT – 12.1

Aim:

Write hello world program in servlet.

Theory:

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows –

$ javac HelloWorld.java

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable.

If everything goes fine, above compilation would produce **HelloWorld.class** file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

Source Code:

```java
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

    private String message;

    public void init() throws ServletException {
        // Do required initialization
        message = "Hello World";
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }

    public void destroy() {
        // do nothing.
    }
```

}

Output:

EXPERIMENT – 12.2

Aim:
Write a servlet which displays current system date and time.

Theory:

The most important advantage of using Servlet is that we can use all the methods available in core java. The Date class is available in java.util package.

A **servlet** is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

This chapter focuses on writing servlets that generate responses to HTTP requests.

Properties of Servlets are as follows:

- Servlets work on the server-side.

- Servlets are capable of handling complex requests obtained from the webserver.

Servlet Architecture is can be depicted from the image itself as provided below as follows:
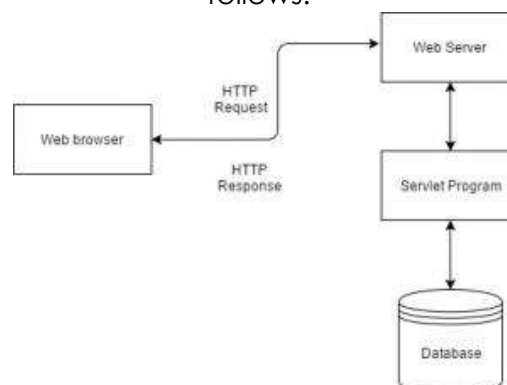
Execution of Servlets basically involves six basic steps:

1. The clients send the request to the webserver.

2. The web server receives the request.

3. The web server passes the request to the corresponding servlet.

4. The servlet processes the request and generates the response in the form of output.

5. The servlet sends the response back to the webserver.

6. The web server sends the response back to the client and the client browser displays it on the screen.

Source Code:

```java
import java.io.*;
import javax.servlet.*;

public class DateSrv extends GenericServlet
{
    //implement service()
    public void service(ServletRequest req, ServletResponse res) throws
IOException, ServletException
    {
        //set response content type
        res.setContentType("text/html");
        //get stream obj
        PrintWriter pw = res.getWriter();
        //write req processing logic
        java.util.Date date = new java.util.Date();
        pw.println("<h2>"+"Current Date & Time: "
+date.toString()+"</h2>");
        //close stream object
        pw.close();
    }
}
```

Output:

EXPERIMENT – 12.3

Aim:
Write a program that handle HTTP request.

Theory:

The most important advantage of using Servlet is that we can use all the methods available in core java. The Date class is available in java.util package.

A **servlet** is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

The doGet () method is invoked by server through service () method to handle a HTTP GET request. This method also handles HTTP HEAD request automatically as HEAD request is nothing but a GET request having no body in the code for response and only includes request header fields. To understand the working of doGet () method, let us consider a sample program to define a servlet for handling the HTTP GET request.

The HTTP client sends the request to the server in the form of request message which includes following information:

- o   The Request-line
- o   The analysis of source IP address, proxy and port
- o   The analysis of destination IP address, protocol, port and host
- o   The Requested URI (Uniform Resource Identifier)
- o   The Request method and Content
- o   The User-Agent header
- o   The Connection control header

o   The Cache control header

Source Code:

```java
import java.net.*;
import java.io.*;

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
                new InputStreamReader(
                        yc.getInputStream()));
        String inputLine;

        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Output:

EXPERIMENT – 12.4

Aim:

Write a program that handle HTTP response.

Theory:

Methods to Set HTTP Response Header

There are following methods which can be used to set HTTP response header in your servlet program. These methods are available with *HttpServletResponse* object.

Sr.No.	Method & Description
1	**String encodeRedirectURL(String url)** Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
2	**String encodeURL(String url)** Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
3	**boolean containsHeader(String name)** Returns a Boolean indicating whether the named response header has already been set.
4	**boolean isCommitted()** Returns a Boolean indicating if the response has been committed.
5	**void addCookie(Cookie cookie)** Adds the specified cookie to the response.
6	**void addDateHeader(String name, long date)**

		Adds a response header with the given name and date-value.
7		**void addHeader(String name, String value)** Adds a response header with the given name and value.
8		**void addIntHeader(String name, int value)** Adds a response header with the given name and integer value.
9		**void flushBuffer()** Forces any content in the buffer to be written to the client.
10		**void reset()** Clears any data that exists in the buffer as well as the status code and headers.
11		**void resetBuffer()** Clears the content of the underlying buffer in the response without clearing headers or status code.
12		**void sendError(int sc)** Sends an error response to the client using the specified status code and clearing the buffer.
13		**void sendError(int sc, String msg)** Sends an error response to the client using the specified status.
14		**void sendRedirect(String location)** Sends a temporary redirect response to the client using the specified redirect location URL.
15		**void setBufferSize(int size)** Sets the preferred buffer size for the body of the response.

16	**void setCharacterEncoding(String charset)** Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8.
17	**void setContentLength(int len)** Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
18	**void setContentType(String type)** Sets the content type of the response being sent to the client, if the response has not been committed yet.
19	**void setDateHeader(String name, long date)** Sets a response header with the given name and date-value.
20	**void setHeader(String name, String value)** Sets a response header with the given name and value.
21	**void setIntHeader(String name, int value)** Sets a response header with the given name and integer value
22	**void setLocale(Locale loc)** Sets the locale of the response, if the response has not been committed yet.
23	**void setStatus(int sc)** Sets the status code for this response

we would use **setIntHeader()** method to set **Refresh** header.

Source Code:

```java
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class refresh extends HttpServlet {

    // Method to handle GET method request.
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // Set refresh, autoload time as 5 seconds
        response.setIntHeader("refresh", 5);

        // Set response content type
        response.setContentType("text/html");

        // Get current time
        Calendar calendar = new GregorianCalendar();
        String am_pm;
        int hour = calendar.get(Calendar.HOUR);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);

        if (calendar.get(Calendar.AM_PM) == 0)
            am_pm = "AM";
        else
            am_pm = "PM";

        String CT = hour + ":" + minute + ":" + second + " " + am_pm;

        PrintWriter out = response.getWriter();
        String title = "Auto refresh Header Setting";
        String docType = "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

        out.println(docType +
                "<html>\n" +
```

```
                    "<head><title>" + title + "</title></head>\n" +
                    "<body bgcolor = \"#f0f0f0\">\n" +
                    "<h1 align = \"center\">" + title + "</h1>\n" +
                    "<p>Current Time is: " + CT + "</p>\n");
    }

    // Method to handle POST method request.
    public void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        doGet(request, response);
    }
}
```

Output:

EXPERIMENT – 12.5

Aim:
Create a servlet that uses Cookies to store the number of times a user has visited your servlet.

Theory:

Many times you would be interested in knowing total number of hits on a particular page of your website. It is very simple to count these hits using a servlet because the life cycle of a servlet is controlled by the container in which it runs.

Following are the steps to be taken to implement a simple page hit counter which is based on Servlet Life Cycle –

- Initialize a global variable in init() method.

- Increase global variable every time either doGet() or doPost() method is called.

- If required, you can use a database table to store the value of global variable in destroy() method. This value can be read inside init() method when servlet would be initialized next time. This step is optional.

- If you want to count only unique page hits with-in a session then you can use isNew() method to check if same page already have been hit with-in that session. This step is optional.

- You can display value of the global counter to show total number of hits on your web site. This step is also optional.

Here I'm assuming that the web container will not be restarted. If it is restarted or servlet destroyed, the hit counter will be reset.

```xml
<servlet>
    <servlet-name>PageHitCounter</servlet-name>
    <servlet-class>PageHitCounter</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>PageHitCounter</servlet-name>
    <url-pattern>/PageHitCounter</url-pattern>
</servlet-mapping>
....
```

Source Code:

```
import java.io.*;
import java.sql.Date;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PageHitCounter extends HttpServlet {

    private int hitCount;

    public void init() {
        // Reset hit counter.
        hitCount = 0;
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");

        // This method executes whenever the servlet is hit
        // increment hitCount
        hitCount++;
        PrintWriter out = response.getWriter();
        String title = "Total Number of Hits";
        String docType = "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

        out.println(docType +
            "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor = \"#f0f0f0\">\n" +
                    "<h1 align = \"center\">" + title + "</h1>\n" +
                    "<h2 align = \"center\">" + hitCount + "</h2>\n" +
                "</body>
            </html>"
        );
    }

    public void destroy() {
        // This is optional step but if you like you
```

```
        // can write hitCount value in your database.
    }
}
```

Output:

http://localhost:8080/HelloWorld - Windows Internet Explorer

http://localhost:8080/HelloWorld

File   Edit   View   Favorites   Tools   Help

http://localhost:8080/HelloWorld

Total Number of Hits

6

Hit Counter for a Website:

Viva Questions
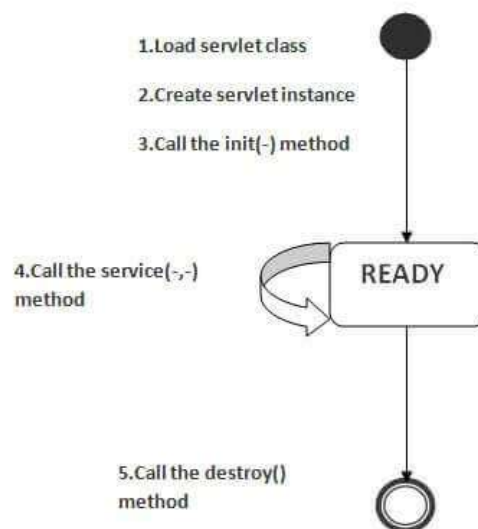
1. How many objects of a servlet is created?

Ans.

Only one object at the time of first request by servlet or web container.

2. What is the life-cycle of a servlet?

Ans.

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.

3. What are the life-cycle methods for a servlet?

Ans.

Method	Description
public void init(ServletConfig config)	It is invoked only once when first request comes for the servlet. It is used to initialize the servlet.
public void service(ServletRequest request,ServletResponse)throws ServletException,IOException	It is invoked at each request.The service() method is used to service the request.
public void destroy()	It is invoked only once when servlet is unloaded.

4. Who is responsible to create the object of servlet?

Ans.

The web container or servlet container.

5. When servlet object is created?

Ans.

At the time of first request.