Practical file submitted in partial fulfillment for the evaluation of

"Object Oriented Programming Lab (AIDS-252)"

Submitted By:

Student Name: Aman goel

Enrolment no: 10817711922

Branch & Section: AI-DS (B)

Submitted To:

- Ms. Shikha Jain

Index

S.No	Experiment Title	Date	Sign
1.	Getting Familiar with Eclipse: (a) Download and Install Eclipse. (b) Using Eclipse for Java.		
2.	Write a Java program to print “Hello World” to understand compilation and execution of java program.		
3.	Write a Java program demonstrating string concatenation.		
4.	Write java program demonstrating the usage of literal datatypes.		
5.	Write a Java program demonstrating the usage of arithmetic, assignment and unary operators.		
6.	Write a java program demonstrating the usage of pre order and post order operations.		
7.	Write a Java program demonstrating the usage of scanner class for user inputs.		
8.	Write a Java program to demonstrate the usage of Bitwise operators.		
9.	Write a Java program to generate random number up to 100 and print whether it is prime number or not.		
10.	(a) Design a Java program to generate first 10 terms of Fibonacci. (b) Find factorial using recursion.		

Index

Index

S.No	Experiment Title	Date	Sign

Experiment 1

Experiment 1: Getting Familiar with Eclipse:

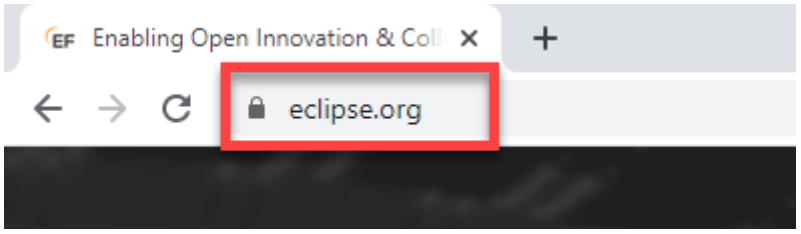
(a) Download and Install Eclipse.

(b) Using Eclipse for Java.

Eclipse Download and Installation Steps:

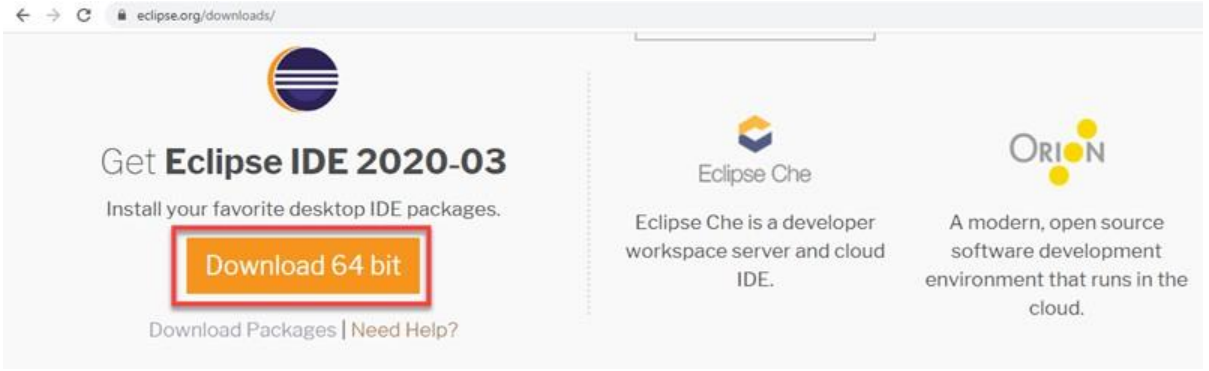
Step 1) **Installing Eclipse**

Open your browser and type https://www.eclipse.org/

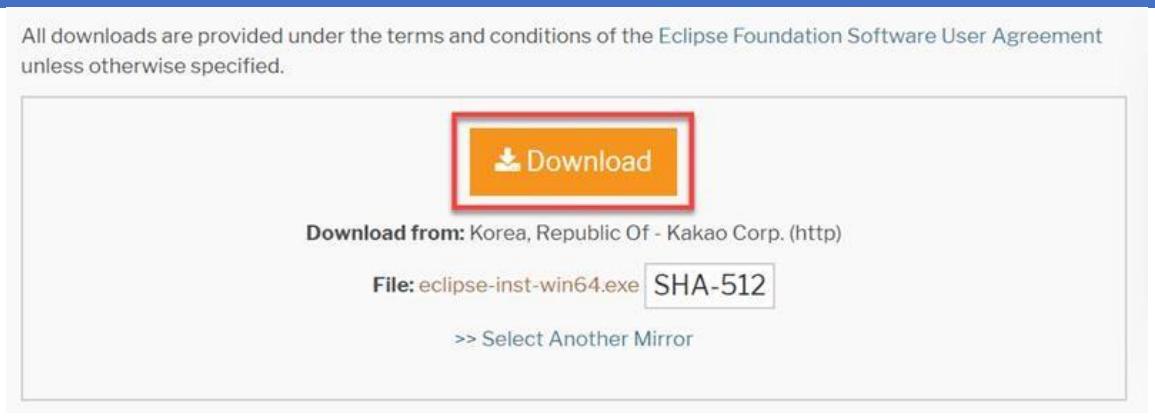


Step 2) **Click on "Download" button.**

Step 3) **Click on "Download 64 bit" button**

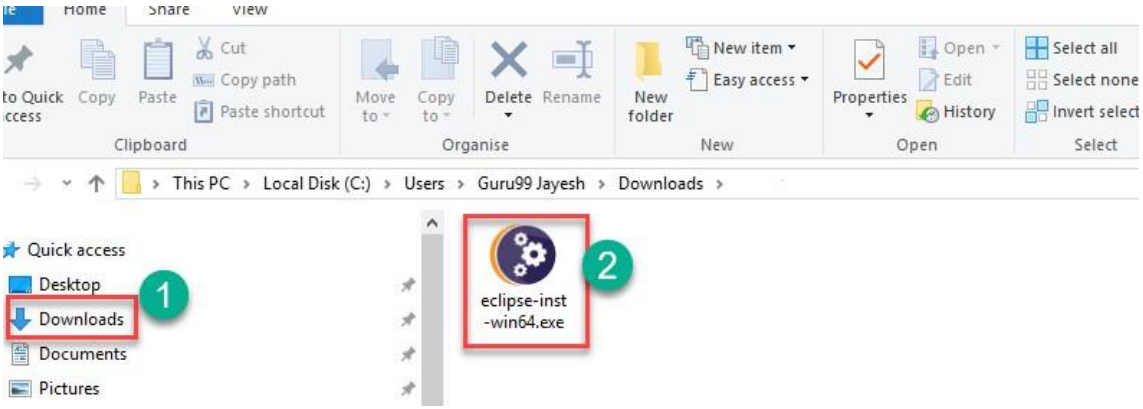


Step 4) **Click on "Download" button**

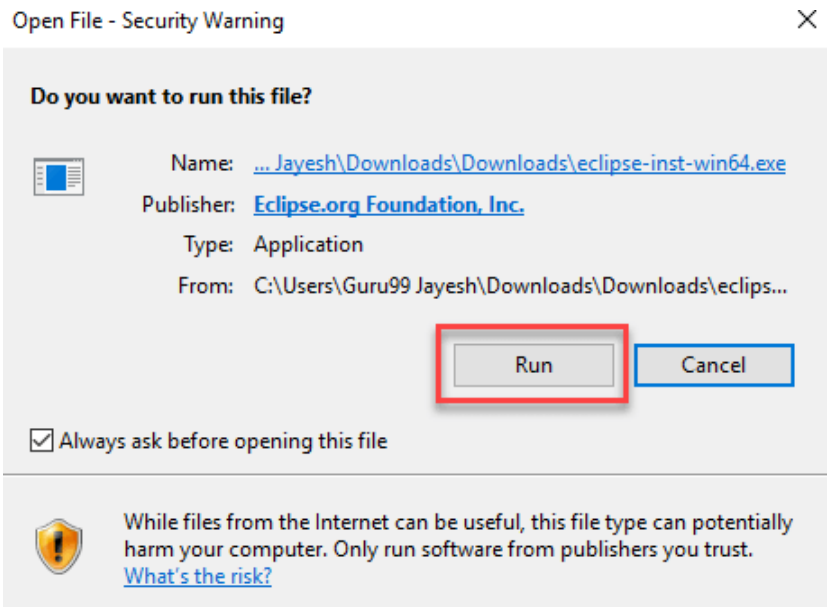


Step 4) **Install Eclipse.**

1. Click on “downloads” in Windows file explorer.
2. Click on “eclipse-inst-win64.exe” file.

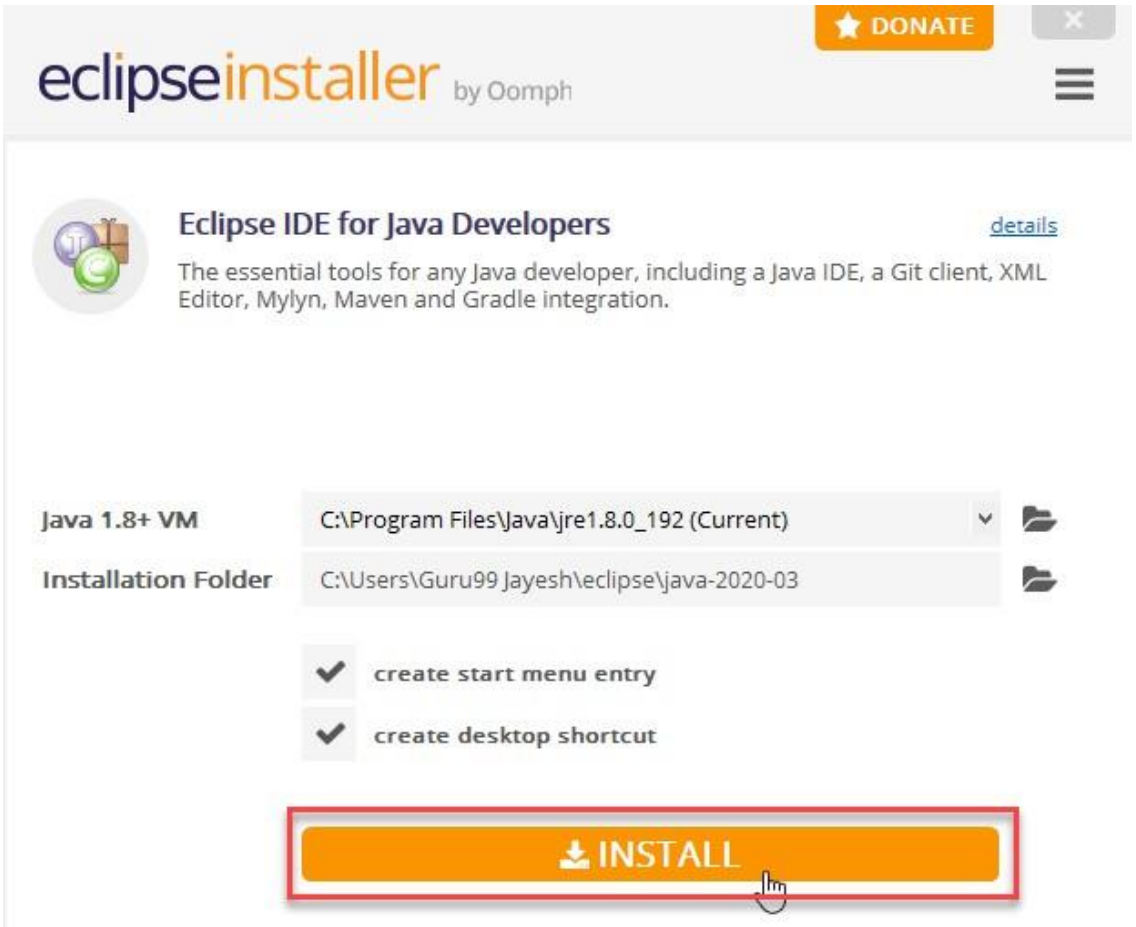


Step 5) **Click on Run button**

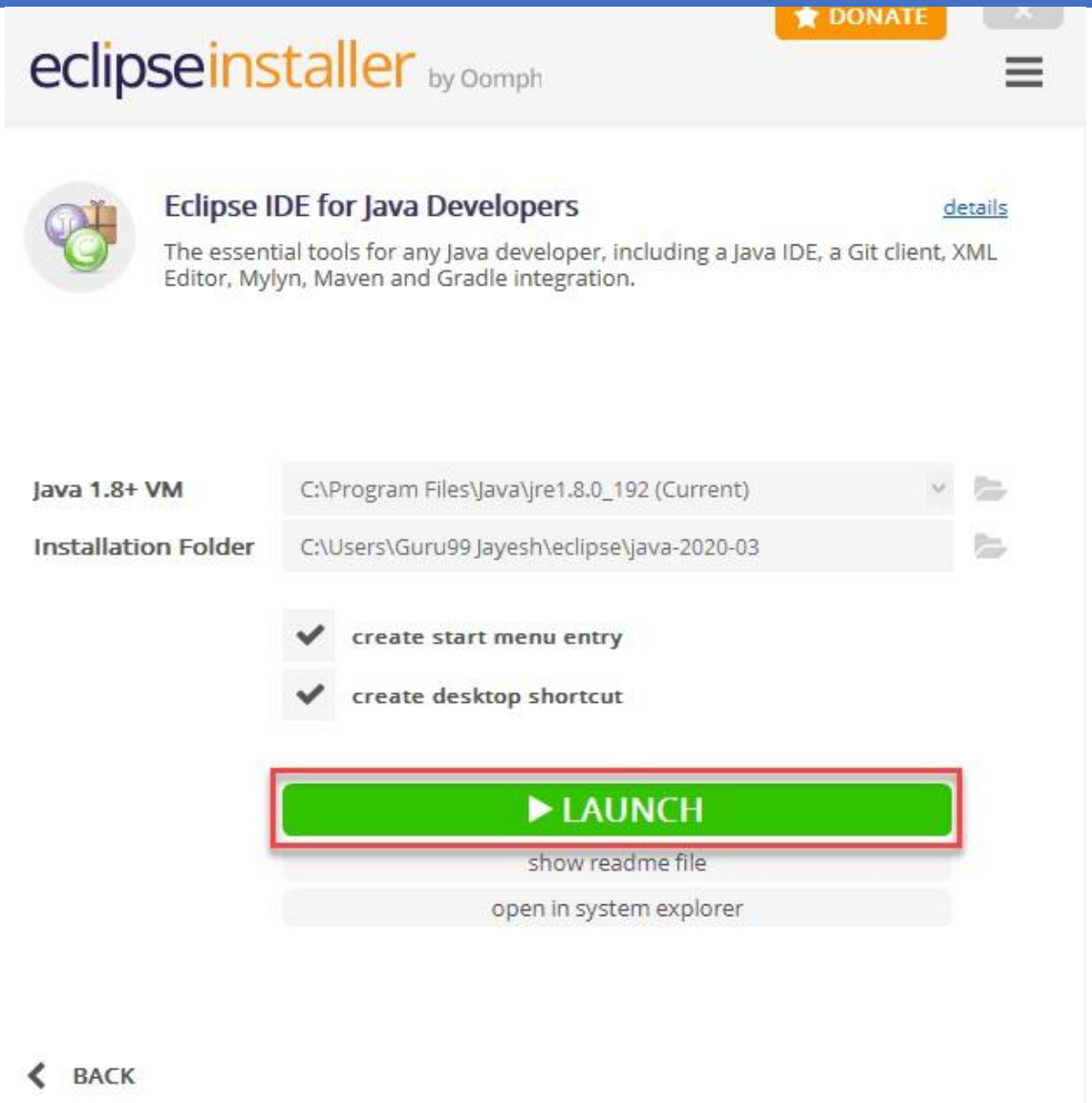


Step 6) **Click on “Eclipse IDE for Java Developers”**

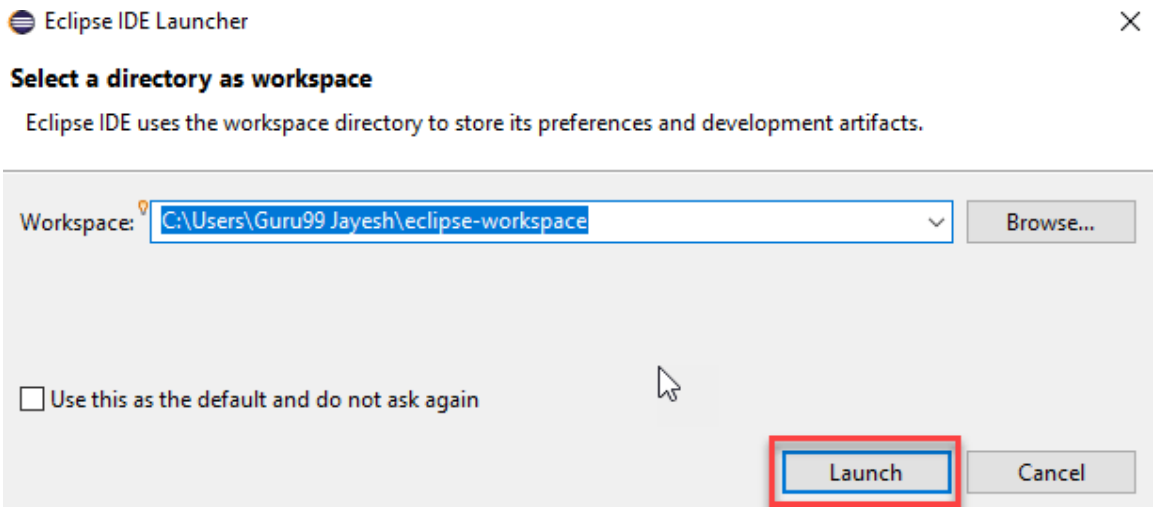
Step 7) **Click on "INSTALL" button**



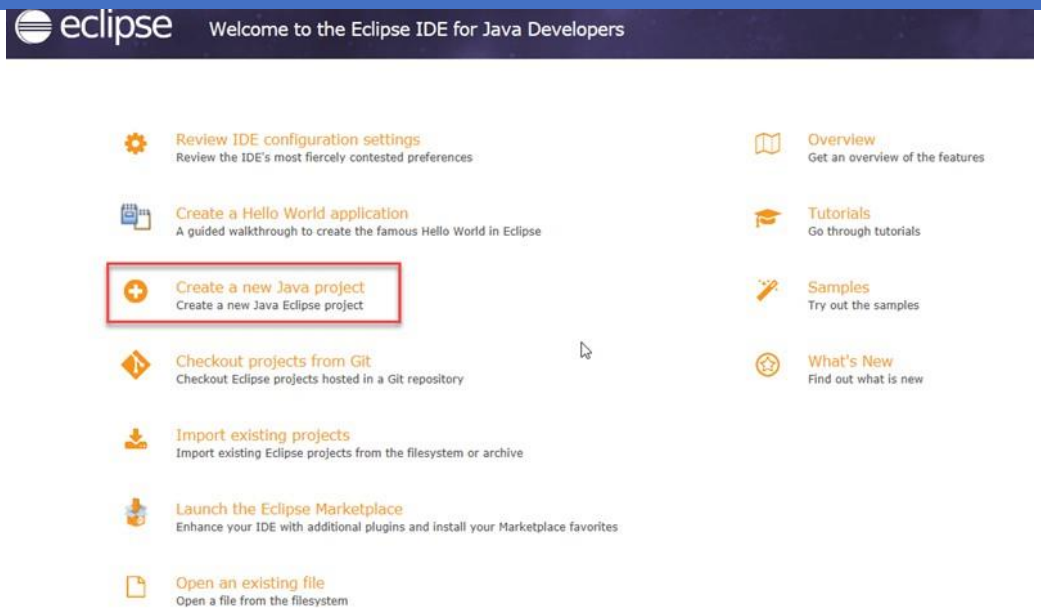
Step 8) **Click on "LAUNCH" button.**



Step 9) **Click on "Launch" button.**

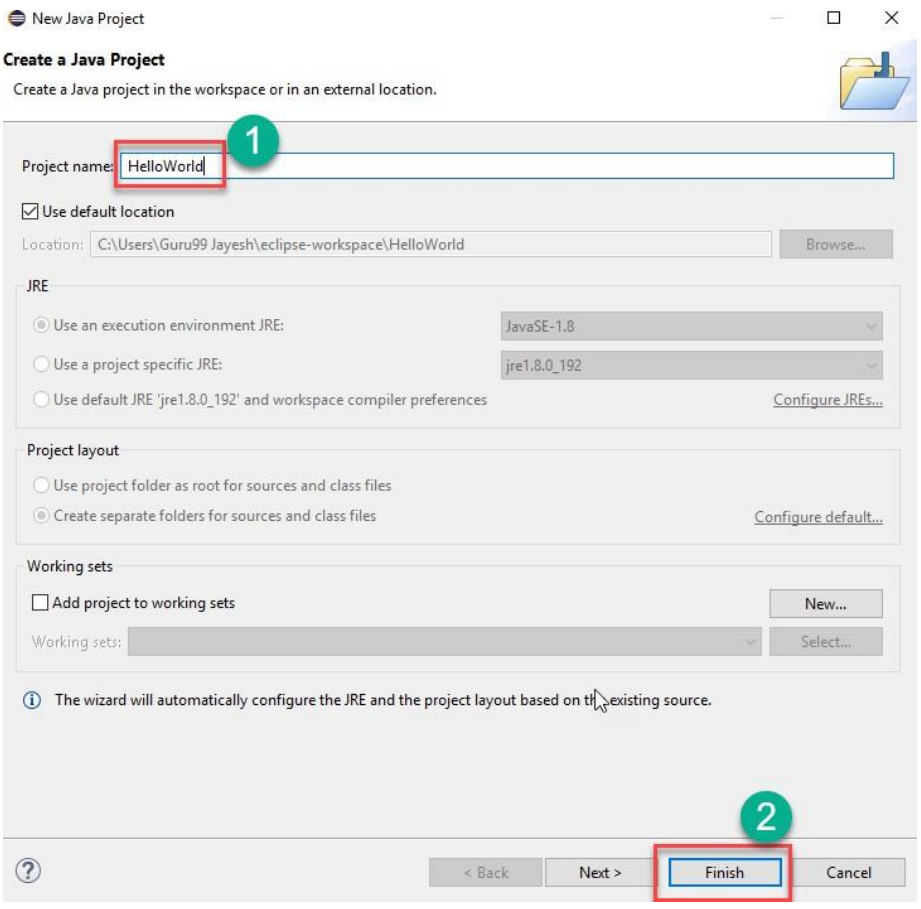


Step 10) **Click on "Create a new Java project" link.**



Step 11) **Create a new Java Project**

1. Write project name.
2. Click on “Finish button”.



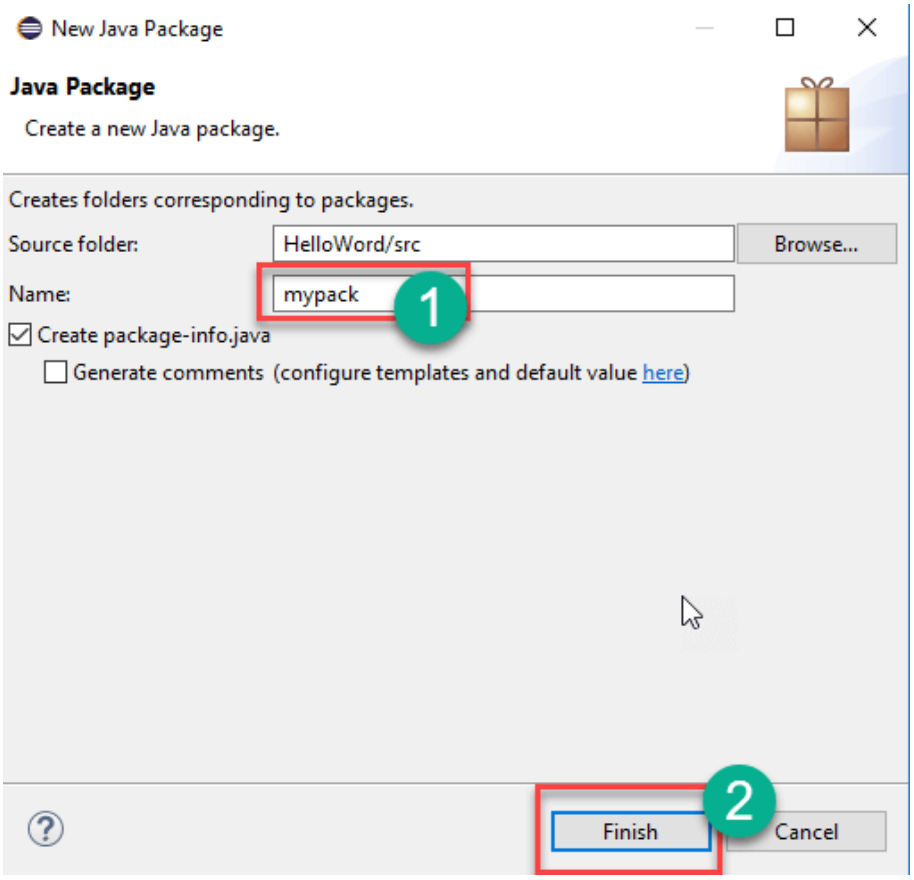
Step 12) **Create Java Package.**

1. Goto “src”.
2. Click on “New”.

3. Click on “Package”.

Step 13) **Writing package name.**

1. Write name of the package
2. Click on Finish button.

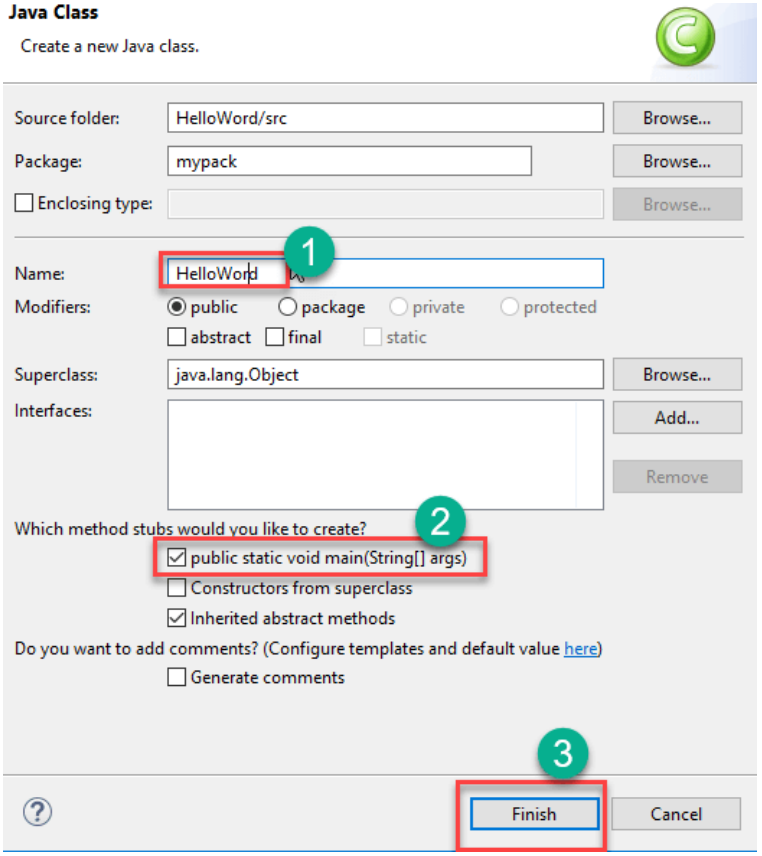


Step 14) **Creating Java Class**

1. Click on package you have created.
2. Click on “New”.
3. Click on “Class”.

Step 15) **Defining Java Class.**

1. Write class name
2. Click on “public static void main (String[] args)” checkbox.
3. Click on “Finish” button.

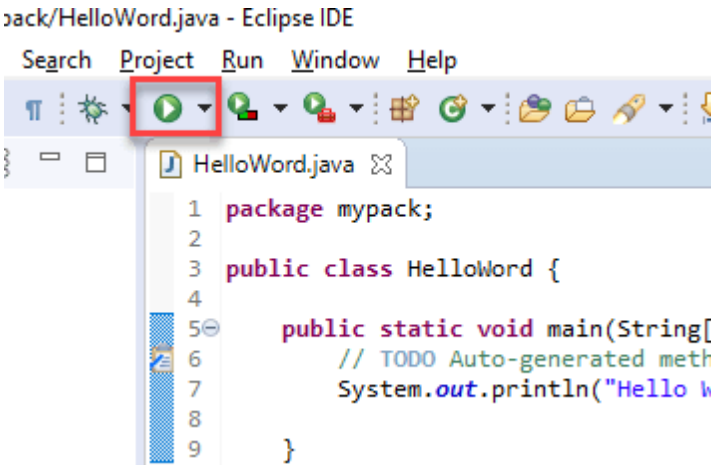


Helloword.java file will be created as shown below:

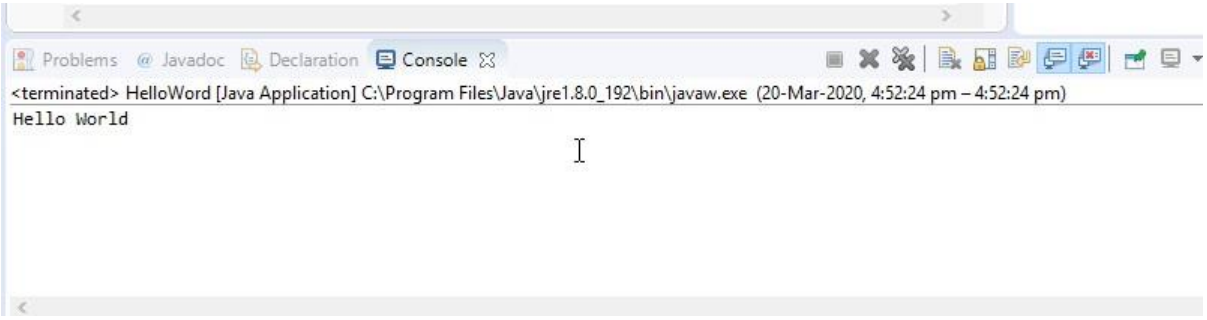
HelloWord.java

```
1  package mypack;
2
3  public class HelloWord {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          System.out.println("Hello World");
8
9      }
10
11 }
12
```

Step 16) **Click on "Run" button.**



Output will be displayed as shown below.



Learning outcome of the Experiment:

Experiment 2

Experiment 2: Write a Java program to print “Hello World” to understand compilation and execution of java program.

Theory:

Code:

```
public class helloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World!");

    }

}
```

Output:

```
Hello, World!
```

Learning outcome of the Experiment:

Experiment 3

Experiment 3: Write a Java program demonstrating string concatenation.

Theory:

Code:

```
public class concat {

    public static void main(String[] args) {
        String firstString = "This is";
        String secondString = " a concatenated string.";
        String thirdString = firstString + secondString;
        System.out.println(thirdString);
    }
}
```

Output:

```
This is a concatenated string.
```

Learning outcome of the Experiment:

Experiment 4

Experiment 4: Write java program demonstrating the usage of literal datatypes.

Theory:

Code:

```
public class literals {
    public static void main(String[] args) {
        int count = 987;
        float floatVal = 4534.99f;
        double cost = 19765.567;
        int hexaVal = 0x7e4;
        int binary = 0b11010;
        char alpha = 'p';
        String str = "Java";
        boolean boolVal = true;
        int octalVal = 067;
        String stuName = null;
        char ch1 = '\u0021';

        System.out.println(count);
        System.out.println(floatVal);
        System.out.println(cost);
        System.out.println(hexaVal);
        System.out.println(binary);
        System.out.println(alpha);
        System.out.println(str);
        System.out.println(boolVal);
        System.out.println(octalVal);
        System.out.println(stuName);
        System.out.println(ch1);
        System.out.println("\t" + "backslash literal");
    }
}
```

Output:

```
987
4534.99
19765.567
2020
26
p
Java
true
55
null
!
        backslash literal
?
```

Learning outcome of the Experiment:

Experiment 5

Experiment 5: Write a Java program demonstrating the usage of arithmetic, assignment and unary operators.

Theory:

Code:

```
public class Arithmetic {

    public static void main(String[] args) {
        int res = 11 + 25;
        System.out.println("11 + 25 = " + res);
        int ogRes = res;

        res = res - 5;
        System.out.println(ogRes + " - 5 = " + res);
        ogRes = res;

        res = res * 7;
        System.out.println(ogRes + " * 7 = " + res);
        ogRes = res;

        res = res / 2;
        System.out.println(ogRes + " / 2 = " + res);
        ogRes = res;

        res = res % 7;
        System.out.println(ogRes + " % 2 = " + res);
        ogRes = res;

    }

}
```

Output:

```
11 + 25 = 36
36 - 5 = 31
31 * 7 = 217
217 / 2 = 108
108 % 2 = 3
```

Learning outcome of the Experiment:

Experiment 6

Experiment 6: Write a java program demonstrating the usage of pre order and post order operations.

Theory:

Code:

```
public class PrePost {
    public static void main(String[] args) {
        int i = 7;
        System.out.println("i = " + i);
        System.out.println("++i = " + ++i);
        System.out.println("i++ = " + i++);
        System.out.println("i = " + i);

    }
}
```

Output:

```
i = 7
++i = 8
i++ = 8
i = 9
```

Learning outcome of the Experiment:

Experiment 7

Experiment 7: Write a Java program demonstrating the usage of scanner class for user inputs.

Theory:

Code:

```java
import java.util.Scanner;
public class scanf {
    public static void main(String[] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.print("Enter Name : ");
        String name = scanner.nextLine();
        System.out.print("Enter ID : ");
        String id = scanner.nextLine();
        System.out.println("Student Name = " + name);
        System.out.println("Student ID = " + id);
        scanner.close();
    }
}
```

Output:

```
Enter Name : Alex
Enter ID : 123
Student Name = Alex
Student ID = 123
```

Learning outcome of the Experiment:

Experiment 8

Experiment 8: Write a Java program to demonstrate the usage of Bitwise operators.

Theory:

Code:

```
public class bitwise {
    public static void main(String[] args) {
        int a = 5;
        int b = 7;

        System.out.println("a & b = " + (a & b));
        System.out.println("a | b = " + (a | b));
        System.out.println("a ^ b = " + (a ^ b));
        System.out.println("~a  = " + (~a));
        a &= b;
        System.out.println("a = " + a);
    }
}
```

Output:

```
a & b = 5
a | b = 7
a ^ b = 2
~a  = -6
a = 5
```

Learning outcome of the Experiment:

Experiment 9

Experiment 9: Write a Java program to generate random number up to 100 and print whether it is prime number or not.

Theory:

Code:

```
import java.util.Random;

public class prime {
    public static void main(String[] args) {
        Random random=new Random();
        int n=random.nextInt(100);
        System.out.println("random number is "+n);
        boolean isPrime = true;
        for(int i=2;i<=Math.sqrt(n);i++) {
            if(n%i==0) {
                isPrime=false;
                break;
            }
        }

        if(isPrime==true) {
            System.out.println("number is prime");
        } else {
            System.out.println("number is not prime");
        }
    }
}
```

Output:

```
random number is 97
number is prime
```

Learning outcome of the Experiment:

Experiment 10

Experiment 10: (a) Design a Java program to generate first 10 terms of Fibonacci.
(b) Find factorial using recursion.

Theory:

(a) Generating First 10 terms of Fibonacci:

Code:

```
public class fibo {
    static int n1=0,n2=1,n3=0;
    static void printFibonacci(int count) {
        if(count>0) {
            n3=n1+n2;
            n1=n2;
            n2=n3;
            System.out.println(n3);
            printFibonacci(count-1);
        }
    }

    public static void main(String[] args) {
        int count=10;
        System.out.println(n1);
        System.out.println(n2);
        printFibonacci(count-2);

    }

}
```

Output:

```
0
1
1
2
3
5
8
13
21
34
```

(b) Finding Factorial using Recursion:

Code:

```java
import java.util.Scanner;
public class Factorial {
    public static void main(String[] args) {
        Scanner input= new Scanner(System.in);
        System.out.print("Enter the number:");
        int num= input.nextInt();
        System.out.println("Factorial of "+ num +" is "+ fact(num));
    }
    static int fact(int num){
        if (num==1 || num==0){
            return 1;
        }
        return num*fact(num-1);
    }
}
```

Output:

```
Enter the number:5
Factorial of 5 is 120
```

Learning outcome of the Experiment:

Experiment 11

Experiment 11: Design a Java program to find the average sum of array of N numbers entered by user.

Theory:

Code:

```
import java.util.Scanner;
public class SumOfArray {
    public static void main(String[] args) {
        Scanner sin= new Scanner(System.in);
        System.out.print("Enter Number of Elements : ");
        int t=sin.nextInt();
        int[] arr= new int[t];
        for (int i = 0; i < t; i++) {
            System.out.print("Enter Element " + (i+1) + " : ");
            arr[i]=sin.nextInt();
        }
        System.out.println("Sum: "+sum(arr)+"\nAverage: "+average(arr));
    }
    static int sum(int[] arr){
        int res=0;
        for (int x:arr) {
            res+=x;
        }
        return res;
    }
    static int average(int[] arr){
        return sum(arr)/arr.length;
    }
}
```

Output:

```
Enter Number of Elements : 5
Enter Element 1 : 5
Enter Element 2 : 10
Enter Element 3 : 15
Enter Element 4 : 20
Enter Element 5 : 25
Sum: 75
Average: 15
```

Learning outcome of the Experiment:

Experiment 12

Experiment 12: Design a Java program to implement classes and objects

(a) Using default constructor.

(b) Using parametrized constructor.

(c) Using Copy constructor.

Theory:

Code:

```
public class Constructor {
    int x;
    static class DefaultConst{
        int first, second;
    }
    Constructor(int x){
        this.x=x; // Parameter
    }
    Constructor(Constructor old){
        this(old.x); // Clone
    }

    public static void main(String[] args) {
        DefaultConst defaultCon = new DefaultConst();
        Constructor ParamCon = new Constructor(5);
        Constructor CloneCon = new Constructor(ParamCon);
        System.out.print("Default : ");
        System.out.println(defaultCon.first+" "+defaultCon.second);
        System.out.print("Parameterized : ");
        ParamCon.display();
        System.out.print("Copy : ");
        CloneCon.display();
    }
    void display() {
        System.out.println(this.x);
    }
}
```

Output:

```
Default : 0 0
Parameterized : 5
Copy : 5
```

Learning outcome of the Experiment:

Experiment 13

Experiment 13: Create a class and find out the area and perimeter of rectangle.

Theory:

Code:

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

        System.out.print("Enter length:");
        float len= input.nextFloat();

        System.out.print("Enter breadth:");
        float brd= input.nextFloat();

        System.out.println("Area = " + area(len, brd));
        System.out.println("Perimeter = " + perimeter(len, brd));
    }

    static float area(float x, float y) {
        return x * y;
    }
    static float perimeter(float x, float y) {
        return 2*(x+y);

    }

}
```

Output:

```
Enter length:2.5
Enter breadth:2
Area = 5.0
Perimeter = 9.0 _
```

Learning outcome of the Experiment:

Experiment 14

Experiment 14:
Create a class circle with instance variable radius and member function
(a) area (b) circumference (c) display
Write a test application named circletest that demonstrate class circled capabilities.

Theory:

Code:

(i) circle.java

```
public class circle {
    double areaCalculated;
    double circumferenceCalculated;
    double area(double x) {
        return Math.PI * x * x;
    }
    double circumference(double x) {
        return 2 * Math.PI * x ;
    }
    public void display() {
        System.out.println("Area = " + areaCalculated);
        System.out.println("Circumference = " + circumferenceCalculated);
    }

}
```

(ii) circletest.java

```java
import java.util.Scanner;

public class circletest {
    public static void main(String[] args) {
        circle obj1= new circle();

        Scanner input= new Scanner(System.in);
        System.out.print("Enter the Radius:");
        double rad= input.nextDouble();

        System.out.println("Calculating area...");
        obj1.areaCalculated = obj1.area(rad);

        System.out.println("Calculating circumference...");
        obj1.circumferenceCalculated = obj1.circumference(rad);

        System.out.println("Displaying area and circumference...");
        obj1.display();
    }
}
```

Output:

```
Enter the Radius:5
Calculating area...
Calculating circumference...
Displaying area and circumference...
Area = 78.53981633974483
Circumference = 31.41592653589793
```

Learning outcome of the Experiment:

Experiment 15

Experiment 15: Design a class that perform string operation (equal, reverse and changeCase).

Theory:

Code:

```
import java.util.Scanner;

public class string {
    public static void main(String[] args) {
        Scanner input= new Scanner(System.in);
        System.out.print("Enter String 1 : ");
        String str1= input.nextLine();

        System.out.print("Enter String 2 : ");
        String str2= input.nextLine();

        if (isEqual(str1, str2)) {
            System.out.println("Entered Strings are Equal.");
            System.out.println("String reversed : "+reveString(str1));
            System.out.println("String after changing case : "+changeCase(str1));

        } else {
            System.out.println("Entered Strings are not Equal.");
            System.out.println("String 1 reversed : "+reveString(str1));
            System.out.println("String 2 reversed : "+reveString(str2));
            System.out.println("String 1 after changing case : "+changeCase(str1));
            System.out.println("String 2 after changing case : "+changeCase(str2));
        }
    }

    static boolean isEqual(String x, String y) {
        return x.equals(y);
    }

  static String reveString(String x) {
        char[] charArray = x.toCharArray();
```

```
        int left = 0;
        int right = charArray.length - 1;

        while (left < right) {
            char temp = charArray[left];
            charArray[left] = charArray[right];
            charArray[right] = temp;

            left++;
            right--;
        }
        return new String(charArray);
    }
    static String changeCase(String x) {
        char[] charArray = x.toCharArray();
        for (int i = 0; i < charArray.length; i++) {
            if (Character.isLowerCase(charArray[i])) {
                charArray[i] = Character.toUpperCase(charArray[i]);
            } else if (Character.isUpperCase(charArray[i])) {
                charArray[i] = Character.toLowerCase(charArray[i]);
            }
        }
        return new String(charArray);
    }
}
```

Output:

```
Enter String 1 : Hello, World!
Enter String 2 : Hello world.
Entered Strings are not Equal.
String 1 reversed : !dlroW ,olleH
String 2 reversed : .dlrow olleH
String 1 after changing case : hELLO, wORLD!
String 2 after changing case : hELLO WORLD.
```

Learning outcome of the Experiment:

Experiment 16

Write a Java program to demonstrate passing objectas parameters. (b)Write a Java program to demonstrate the difference between call by value and call by reference.

Theory:

Code:

a)
```
class MyClass {
    int value;

    MyClass(int value) {
        this.value = value;
    }
}

public class ObjectParameterDemo {
    static void modifyObject(MyClass obj) {
        obj.value *= 2;
```

```
    }
      public static void main(String[] args) {
        MyClass obj = new MyClass(10);
        System.out.println("Before modification: " + obj.value);
        modifyObject(obj);
        System.out.println("After modification: " + obj.value);
    }
}
```

b)
```
class Test {
    int value;

    Test(int value) {
        this.value = value;
    }
}

public class CallByValueReferenceDemo {
    static void modifyPrimitive(int num) {
        num *= 2;
        System.out.println("Inside modifyPrimitive: " + num);
    }
    static void modifyObject(Test obj) {
        obj.value *= 2;
        System.out.println("Inside modifyObject: " + obj.value);
    }
    public static void main(String[] args) {
        int num = 10;
        System.out.println("Before modifyPrimitive: " + num);
        modifyPrimitive(num);
        System.out.println("After modifyPrimitive: " + num);
```

```
        Test obj = new Test(10);
        System.out.println("Before modifyObject: " + obj.value);
        modifyObject(obj);
        System.out.println("After modifyObject: " + obj.value);
    }
}
```

Output:

a)

```
Before modification: 10
After modification: 20
```

b)

```
Before modifyPrimitive: 10
Inside modifyPrimitive: 20
After modifyPrimitive: 10
Before modifyObject: 10
Inside modifyObject: 20
After modifyObject: 20
```

Learning outcome of the Experiment:

Experiment 17

Write a Java program to implement inheritance. Define a class Box with the following instance variables: width, height, and depth, all of type float. Create a new class BoxWeight that extends Box to include weight as an instance variable. Write an application that tests the functionalities of both these classes.

Theory:

Code:

```
class Box {
    float width;
    float height;
    float depth;
    Box(float width, float height, float depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
    }
    float volume() {
        return width * height * depth;
    }
}
class BoxWeight extends Box {
    float weight;

    // Constructor
```

```
    BoxWeight(float width, float height, float depth, float weight) {
        super(width, height, depth);
        this.weight = weight;
    }
}

public class InheritanceDemo {
    public static void main(String[] args) {
        // Create a Box object
        Box box = new Box(5, 3, 2);
        System.out.println("Volume of the box: " + box.volume());

        // Create a BoxWeight object
        BoxWeight boxWeight = new BoxWeight(3, 2, 1, 10);
        System.out.println("Volume of the box weight: " + boxWeight.volume());
        System.out.println("Weight of the box: " + boxWeight.weight);
    }
}
```

Output:

```
Volume of the box: 30.0
Volume of the box weight: 6.0
Weight of the box: 10.0
```

Learning outcome of the Experiment:

Experiment 18

Implement the following Java programs to demonstrate the concept of exception handling using keywords try, catch, finally, throw, and throws wherever required. (a) Write a Java program using switch to demonstrate the usage of try/catch block for the following handling exceptions:

Case 1: ArithmeticException

Case 2: IndexOutOfBoundsException

Case 3: NullPointerException

Case 4: NumberFormatException (b) Write a Java program to demonstrate how unreachable code is created and compile-time error occurs when superclass exception occurs prior to subclass exception in a series of catch statements. (c) Write a Java program that shows how to catch and handle number format and division by zero exceptions in programs that use input dialog boxes and/or text fields.

Theory:

Code:

a)
```
import java.util.Scanner;


public class ExceptionHandlingDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        int choice = scanner.nextInt();

        try {
            switch (choice) {
                case 1:
```

```
                int result = 10 / 0; // Arithmetic Exception
                break;
            case 2:
                int[] arr = new int[3];
                int value = arr[5]; // Index Out of Bounds Exception
                break;
            case 3:
                String str = null;
                int length = str.length(); // Null Pointer Exception
                break;
            case 4:
                int num = Integer.parseInt("abc"); // Number Format Exception
                break;
            default:
                System.out.println("Invalid choice");
        }
    } catch (ArithmeticException e) {
        System.out.println("ArithmeticException occurred: " + e.getMessage());
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("IndexOutOfBoundsException occurred: " + e.getMessage());
    } catch (NullPointerException e) {
        System.out.println("NullPointerException occurred: " + e.getMessage());
    } catch (NumberFormatException e) {
        System.out.println("NumberFormatException occurred: " + e.getMessage());
    }
    scanner.close();
  }
}
```

b) `import java.io.IOException;`

```
public class UnreachableCodeDemo {
    public static void main(String[] args) {
        try {
            // Code that may throw IOException
            throw new IOException();
        } catch (Exception e) {
            System.out.println("Exception occurred: " + e.getMessage());
        } catch (IOException e) { // This catch block will never be reached
            System.out.println("IOException occurred: " + e.getMessage());
        }
    }
}
```

c)
```
import java.util.Scanner;
public class InputExceptionHandlingDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Enter a number: ");
            String input = scanner.nextLine();
            int number = Integer.parseInt(input);
            int result = 10 / number;
            System.out.println("Result: " + result);
        } catch (NumberFormatException e) {
            System.out.println("Number Format Exception: Invalid input");
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception: Division by zero");
        }
        scanner.close();
    }
}
```

Output:

a)

```
Enter a number:
3
NullPointerException occurred: Cannot invoke "String.length()" because "<local6>" is
    null
```

```
Enter a number:
2
IndexOutOfBoundsException occurred: Index 5 out of bounds for length 3
```

b)

```
ERROR!
/tmp/PIYC4qHdLb/UnreachableCodeDemo.java:10: error: exception IOException has already
    been caught
        } catch (IOException e) { // This catch block will never be reached
          ^
1 error

=== Code Exited With Errors ===
```

c)

```
Enter a number: 0
Arithmetic Exception: Division by zero
```

Learning outcome of the Experiment:

Experiment 19

Demonstrate the use of final key word with data member, function and class.

Theory:

Code:

```
public class FinalKeywordDemo {
    final int finalData = 100; // Final data member

    final void finalMethod() { // Final method
        System.out.println("This is a final method.");
    }

    public static void main(String[] args) {
        FinalKeywordDemo obj = new FinalKeywordDemo();
        System.out.println("Final data: " + obj.finalData);
        obj.finalMethod();
    }
}

final class FinalClass { // Final class
    // Final class cannot be subclassed
}
```

Output:

```
Final data: 100
This is a final method.

=== Code Execution Successful ===
```

Learning outcome

Experiment 20

Write a Java program to demonstrate the concept of abstract classes and interfaces

Theory:

Code:

```
// Abstract class
  abstract class Shape {
      // Abstract method
      abstract double area();
   // Concrete method
      void display() {
          System.out.println("This is a shape.");
      }}
  // Interface
  interface Drawable {
      void draw();
  }
  // Concrete class implementing the abstract class and interface
  class Circle extends Shape implements Drawable {
      double radius;
  Circle(double radius) {
          this.radius = radius;
      }


      // Implementing the abstract method from the abstract class
```

```
    double area() {
        return Math.PI * radius * radius;
    }
// Implementing the method from the interface
    public void draw() {
        System.out.println("Drawing a circle.");
    }
}
public class AbstractInterfaceDemo {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        System.out.println("Area of the circle: " + circle.area());
        circle.display();
        circle.draw();
    }
}
```

Output:

```
Area of the circle: 78.53981633974483
This is a shape.
Drawing a circle.
```

Learning outcome of the Experiment:

Experiment 21

Write a Java program to demonstrate the usage of following Collections:

List: ArrayList, Set , Map

Theory:

Code:

```
import java.util.*;


public class CollectionsDemo {
    public static void main(String[] args) {
        // ArrayList (List)
        List<String> arrayList = new ArrayList<>();
        arrayList.add("Apple");
        arrayList.add("Banana");
        arrayList.add("Orange");
        System.out.println("ArrayList:");
        for (String fruit : arrayList) {
            System.out.println(fruit);
        }
 // HashSet (Set)
        Set<String> hashSet = new HashSet<>();
        hashSet.add("Apple");
        hashSet.add("Banana");
        hashSet.add("Orange");
        System.out.println("\nHashSet:");
        for (String fruit : hashSet) {
```

```
            System.out.println(fruit);
        }
    // HashMap (Map)
        Map<Integer, String> hashMap = new HashMap<>();
        hashMap.put(1, "Apple");
        hashMap.put(2, "Banana");
        hashMap.put(3, "Orange");
        System.out.println("\nHashMap:");
        for (Map.Entry<Integer, String> entry : hashMap.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

Output:

```
ArrayList:
Apple
Banana
Orange

HashSet:
Apple
Orange
Banana

HashMap:
1: Apple
2: Banana
3: Orange
```

Learning outcome of the Experiment:

Experiment 22

Design a program to demonstrate multi-threading using Thread Class.

Theory:

Code:

```
class MyThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName() + " is executing: " + i);
            try {
                Thread.sleep(1000); // Sleep for 1 second
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}
public class MultiThreadingDemo {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();

        // Start threads
        t1.start();
        t2.start();
```

```
    }
  }
```

Output:

```
Thread-1 is executing: 1
Thread-0 is executing: 1
Thread-1 is executing: 2
Thread-0 is executing: 2
Thread-1 is executing: 3
Thread-0 is executing: 3
Thread-1 is executing: 4
Thread-0 is executing: 4
Thread-1 is executing: 5
Thread-0 is executing: 5

=== Code Execution Successful ===
```

Learning outcome of the Experiment:

Experiment 23

Design a program to create game ‘Tic Tac Toe’.

Theory:

Code:

```
import java.util.Scanner;

public class TicTacToe {
    // Constants representing players
    private static final char PLAYER_X = 'X';
    private static final char PLAYER_O = 'O';

    // Game board
    private char[][] board;
    // Current player
    private char currentPlayer;

    // Constructor
    public TicTacToe() {
        board = new char[3][3];
        currentPlayer = PLAYER_X;
        initializeBoard();
    }

    // Initialize the board with empty spaces
```

```
private void initializeBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = ' ';
        }
    }
}

// Display the board
public void displayBoard() {
    System.out.println(" ---------- ");
    for (int i = 0; i < 3; i++) {
        System.out.print("| ");
        for (int j = 0; j < 3; j++) {
            System.out.print(board[i][j] + " | ");
        }
        System.out.println("\n ----------");
    }
}

// Switch players
private void switchPlayer() {
    currentPlayer = (currentPlayer == PLAYER_X) ? PLAYER_O : PLAYER_X;
}

// Check if the board is full
private boolean isBoardFull() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
```

```
                return false; // Board is not full
            }
        }
    }
    return true; // Board is full
}
// Check if a player has won
private boolean hasWon(char player) {
    // Check rows and columns
    for (int i = 0; i < 3; i++) {
        if (board[i][0] == player && board[i][1] == player && board[i][2] == player) {
            return true; // Row i
        }
        if (board[0][i] == player && board[1][i] == player && board[2][i] == player) {
            return true; // Column i
        }
    }
    // Check diagonals
    if (board[0][0] == player && board[1][1] == player && board[2][2] == player) {
        return true; // Main diagonal
    }
    if (board[0][2] == player && board[1][1] == player && board[2][0] == player) {
        return true; // Secondary diagonal
    }
    return false; // No win
}
// Main game logic
public void play() {
    Scanner scanner = new Scanner(System.in);
    int row, col;
```

```
        System.out.println("Welcome to Tic Tac Toe!");
        while (true) {
            // Display current board
            displayBoard();
            // Prompt current player for their move
            System.out.println("Player " + currentPlayer + ", enter your move (row[1-3] col[1-
3]):");
            row = scanner.nextInt() - 1;
            col = scanner.nextInt() - 1;

            // Check if the move is valid
            if (row < 0 || row >= 3 || col < 0 || col >= 3 || board[row][col] != ' ') {
                System.out.println("Invalid move. Try again.");
                continue;
            }
            // Update board with player's move
            board[row][col] = currentPlayer;
 // Check if the current player has won
            if (hasWon(currentPlayer)) {
                displayBoard();
                System.out.println("Congratulations! Player " + currentPlayer + " wins!");
                break;
            }
            // Check if the board is full (draw)
            if (isBoardFull()) {
                displayBoard();
                System.out.println("It's a draw!");
                break;
            }
// Switch players
            switchPlayer();
```

```
        }
    scanner.close();
    }
    public static void main(String[] args) {
        TicTacToe game = new TicTacToe();
        game.play();
    }
}
```

Output:

```
Welcome to Tic Tac Toe!
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Player X, enter your move (row[1-3] col[1-3]):
1 1
-------------
| X |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Player O, enter your move (row[1-3] col[1-3]):
1 2
-------------
| X | O |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
```

Learning outcome of the Experiment: