



**Devang Patel Institute of  
Advance Technology and Research**  
(A Constitue Institute of CHARUSAT)

## Certificate

*This is to certify that*

*Mr./Mrs.* Shubh Chintalkumar Patel.

*of* BCSE-2 *Class,*

*ID. No.* 23DCS091 *has satisfactorily completed*

*his/ her term work in* CSE201- Java Programming *for*

*the ending in* October- 2024 / 2025  
November

*Date :* 17-10-2024

  
*Sign. of Faculty*

  
*Head of Department*

**CHAROTAR UNIVERSITY OF SCIENCE TECHNOLOGY**  
**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

Department of Computer Science & Engineering

•

**Subject Name: java programming**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024**

**Part - 1**

No .	Aim of the Practical
1.	<p>Demonstration of installation steps of Java, Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE, or BlueJ and Console Programming.</p> <p><b><u>THEORY :</u></b></p> <ul style="list-style-type: none"> <li>• <b>Class:</b> <ol style="list-style-type: none"> <li>1. A blueprint for creating objects (a particular data structure), providing initial values for state (member variables or fields), and implementations of behavior (member functions or methods).</li> </ol> </li> <li>• <b>Object:</b> <ol style="list-style-type: none"> <li>1. An instance of a class. When a class is defined, no memory is allocated until an object of that class is created.</li> </ol> </li> <li>• <b>Inheritance:</b> <ol style="list-style-type: none"> <li>1. Mechanism in which one class acquires the property of another class. It provides the idea of reusability.</li> </ol> </li> <li>• <b>Polymorphism:</b> <ol style="list-style-type: none"> <li>1. The ability to present the same interface for differing underlying forms (data types).</li> </ol> </li> </ul>

- **Encapsulation:**

1. Wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.

- **Abstraction:**

1. The concept of hiding the complex implementation details and showing only the necessary features of the object.

## COMPARISON OF JAVA

- **C++:**

1. Java is platform-independent while C++ is platform-dependent.
2. Java does not support multiple inheritance directly due to ambiguity issues, while C++ supports it.
3. Java has automatic garbage collection, whereas C++ requires manual memory management.

- **Python:**

1. Java is statically typed while Python is dynamically typed.
2. Python is generally slower than Java due to its interpreted nature, while Java is compiled to bytecode which is faster.
3. Python syntax is simpler and more concise compared to Java.

- **C#:**

1. Both are similar in syntax and usage, but C# is mainly used for Windows applications.
2. Java has a broader range of platforms compared to C# which is more integrated with Windows OS.
3. Both have automatic garbage collection and similar object-oriented principles.

## Introduction to JDK, JRE, JVM, Javadoc, Command Line Argument

- **JDK (Java Development Kit):**

- The JDK is a software development kit used to develop Java applications. It includes the JRE and development tools such as the Java compiler and debugger.

- **JRE (Java Runtime Environment):**

	<ul style="list-style-type: none"><li>○ The JRE provides the libraries, Java Virtual Machine (JVM), and other components to run applications written in Java. It does not include development tools like compilers and debuggers.</li><li>• <b>JVM (Java Virtual Machine):</b><ul style="list-style-type: none"><li>○ The JVM is a virtual machine that enables a computer to run Java programs. It converts Java bytecode into machine code.</li></ul></li><li>• <b>Javadoc:</b><ul style="list-style-type: none"><li>○ Javadoc is a tool that generates API documentation in HTML format from Java source code.</li></ul></li><li>• <b>Command Line Argument:</b><ul style="list-style-type: none"><li>○ Command line arguments are passed to the main() method of a Java program and can be accessed via the <code>args</code> parameter.</li></ul></li></ul> <p><b><u>PROGRAM CODE :</u></b></p> <pre>public class demo {     public static void main(String[] args){         System.out.println("hello world");         System.out.println("23dcs091 Shubh Patel");     } }</pre> <p><b><u>OUTPUT:</u></b></p> <pre>hello world 23dcs091 Shubh Patel</pre> <p><b><u>CONCLUSION:</u></b></p> <p>This is a basic program to display Hello, World!.</p>
2.	<p>Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's say the current balance is \$20. Write a java program to store this balance in a variable and then display it to the user.</p> <p><b><u>PROGRAM CODE :</u></b></p> <pre>class practical2 {</pre>

```
public static void main(String[] args)
{
    double currentbalance = 20.0;
    System.out.println("Current Balance is: $" + currentbalance);
    System.out.println("23dcs091 Shubh Patel");
}
}
```

**OUTPUT:**

```
Current Balance is: $20.0
23dcs091 Shubh Patel
```

**CONCLUSION:**

In this example, we created a basic Java program that simulates a banking application by storing a user's account balance in a variable and then displaying it. This demonstrates the fundamental concept of variable storage and output in Java, which is essential for developing more complex banking systems.

3. Write a program to take the user for a distance (in meters) and the time taken (as three numbers: hours, minutes, seconds), and display the speed, in meters per second, kilometers per hour and miles per hour (hint: 1 mile = 1609 meters).

**PROGRAM CODE :**

```
import java.util.Scanner;
public class practical3
{
    public static void main(String[] args)
    {
        Scanner obj= new Scanner(System.in);

        System.out.println("Enter Distance(in meters):");
        double distance= obj.nextDouble();
```

```
System.out.println("Enter timetaken in hours:");
int hours= obj.nextInt();

System.out.println("Enter timetaken in minutes:");
int minutes= obj.nextInt();

System.out.println("Enter timetaken in seconds:");
int seconds= obj.nextInt();

double totaltime= hours*3600 + minutes*60 + seconds;

double speedms= distance/totaltime;

double speedkmh= speedms*3.6;

double speedmh= speedkmh/1.609;

System.out.println("Speed in meters per seconds is:" + speedms);

System.out.println("Speed in kilometers per hour is:" + speedkmh);

System.out.println("Speed in miles per hour is:" + speedmh);

System.out.println("23dcs091 Shubh Patel");
}
}
```

**OUTPUT:**

```
Enter Distance(in meters):
600
Enter timetaken in hours:
5
Enter timetaken in minutes:
30
Enter timetaken in seconds:
45
Speed in meters per seconds is:0.030234315948601664
Speed in kilometers per hour is:0.108843537414966
Speed in miles per hour is:0.0676466982069397
23dcs091 Shubh Patel
```

	<p><b><u>CONCLUSION:</u></b></p> <p>In this program, we used Java's Scanner class to take user input for distance in meters and time in hours, minutes, and seconds. We then converted the time to total seconds to simplify the calculations. The program calculates and displays the speed in three different units: meters per second, kilometers per hour, and miles per hour. This example demonstrates how to handle user input, perform unit conversions, and output results in Java.</p>
4.	<p>Imagine you are developing a budget tracking application. You need to calculate the total expenses for the month. Users will input their daily expenses, and the program should compute the sum of these expenses. Write a Java program to calculate the sum of elements in an array representing daily expenses.</p> <p><b><u>PROGRAM CODE :</u></b></p> <pre>import java.util.Scanner;  public class practical4 {     public static void main(String[] args)     {         Scanner obj= new Scanner(System.in);          System.out.println("Enter number of days in month:");         int numdays= obj.nextInt();          double[] dailyexpenses= new double[numdays];         double totalexpenses=0;          for(int i=0; i&lt;numdays; i++)         {             System.out.println("Enter expense for day" + (i+1) + ":");             dailyexpenses[i]= obj.nextInt();             totalexpenses += dailyexpenses[i];         }          System.out.println("Total Expense of month is:" + totalexpenses);         System.out.println("23dcs091 Shubh Patel");     } }</pre>

**OUTPUT:**

```
Enter number of days in month:
8
Enter expense for day1:
200
Enter expense for day2:
300
Enter expense for day3:
500
Enter expense for day4:
100
Enter expense for day5:
300
Enter expense for day6:
200
Enter expense for day7:
400
Enter expense for day8:
300
Total Expense of month is:2300.0
23dcs091 Shubh Patel
```

**CONCLUSION:**

This Java program provides a simple yet effective way to track and calculate monthly expenses based on daily inputs from the user. It demonstrates how to use arrays to store data, take user input through a loop, and perform a summation of array elements.

5. An electric appliance shop assigns code 1 to motor, 2 to fan, 3 to tube and 4 for wires. All other items have code 5 or more. While selling the goods, a sales tax of 8% to motor, 12% to fan, 5% to tube light, 7.5% to wires and 3% for all other items is charged. A list containing the product code and price in two different arrays. Write a java program using switch statement to prepare the bill.

**PROGRAM CODE :**

```
import java.util.Scanner;

public class practical5
{
    public static void main(String[] args)
    {
```



```
Scanner obj= new Scanner(System.in);

System.out.println("Enter number of items you wanna purchase:");
int numpur= obj.nextInt();

int[] code= new int[numpur];

double[] price= new double[numpur];

System.out.println("Code 1:motor, Code 2:fan, Code 3:tube, Code 4:wires,
Code 5:other products");

for(int i=0; i<numpur; i++)
{
System.out.println("Enter code of product:" + (i+1) );
code[i]= obj.nextInt();

System.out.println("Enter price of product:" + (i+1) );
price[i]= obj.nextDouble();
}

double totalprice=0;

for(int i=0;i<numpur;i++)
{
double taxrate = 0;
switch(code[i])
{
case 1:
taxrate = 0.08;
break;
case 2:
taxrate = 0.12;
break;
case 3:
taxrate = 0.05;
break;
case 4:
taxrate = 0.075;
break;
default:
taxrate = 0.03;
}

double prodprice = price[i];
```

```
double taxprice = price[i]*taxrate;
double prodtpprice = prodprice + taxprice;

System.out.println("PRODUCT" + (i+1) + ":");
System.out.println("Product price:" + prodprice);
System.out.println("Tax price:" + taxprice);
System.out.println("Product price including taxes:" + prodtpprice);

totalprice += prodtpprice;
}

System.out.println("Your Bill is:" + totalprice);
System.out.println("23dcs091 Shubh Patel");
}
}
```

**OUTPUT:**

```
Enter number of items you wanna purchase:
3
Code 1:motor, Code 2:fan, Code 3:tube, Code 4:wires, Code 5:other products
Enter code of product:1
1
Enter price of product:1
300
Enter code of product:2
3
Enter price of product:2
200
Enter code of product:3
2
Enter price of product:3
400
PRODUCT1:
Product price:300.0
Tax price:24.0
Product price including taxes):324.0
PRODUCT2:
Product price:200.0
Tax price:10.0
Product price including taxes):210.0
PRODUCT3:
Product price:400.0
Tax price:48.0
Product price including taxes):448.0
Your Bill is:982.0
23dcs091 Shubh Patel
```

**CONCLUSION:**

This practical exercise reinforced the fundamentals of array handling, control structures, and basic arithmetic operations in Java

6. Create a Java program that prompts the user to enter the number of days (n) for which they want to generate their exercise routine. The program should then calculate and display the first n terms of the Fibonacci series, representing the exercise duration for each day.

**PROGRAM CODE :**

```
import java.util.Scanner;

public class practical6
{
    public static void main(String[] args)
    {
        Scanner obj= new Scanner(System.in);

        System.out.println("Enter the term till which you wanna run the series:");
        int n= obj.nextInt();

        int fterm = 0;
        int sterm = 1;

        for(int i=0;i<n;i++)
        {
            int nextterm = fterm + sterm;
            System.out.println( fterm + "+" + sterm + "=" + nextterm );
            fterm = sterm;
            sterm = nextterm;
        }
        System.out.println("23dcs091 Shubh Patel");
    }
}
```

**OUTPUT:**

```
Enter the term till which you wanna run the series:
13
0+1=1
1+1=2
1+2=3
2+3=5
3+5=8
5+8=13
8+13=21
13+21=34
21+34=55
34+55=89
55+89=144
89+144=233
144+233=377
23dcs091 Shubh Patel
```

	<p><b><u>CONCLUSION:</u></b></p> <p>This program provides an easy way to generate an exercise routine based on the Fibonacci series, which is often used to progressively increase workout durations. By following this routine, you can ensure a gradual increase in your exercise time, which can help improve endurance and overall fitness. The Fibonacci series starts with 0 and 1, and each subsequent term is the sum of the previous two terms. This pattern creates a naturally progressive routine, making it a useful tool for designing exercise plans.</p>
--	--

## SET - 2

No .	Aim of the Practical
7.	<p>Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;</p> <p>front_times('Chocolate', 2) → 'ChoCho'</p> <p>front_times('Chocolate', 3) → 'ChoChoCho'</p> <p>front_times('Abc', 3) → 'AbcAbcAbc'</p> <p><b><u>PROGRAM CODE:</u></b></p>

```
import java.util.Scanner;

public class practical7
{
    public static void main(String[] args)
    {
        Scanner obj = new Scanner(System.in);

        System.out.println("Enter a string: ");
        String str = obj.nextLine();

        System.out.println("Enter a number: ");
        int n = obj.nextInt();

        String result= fronttimes(str,n);
        System.out.println("RESULT= " + result);
        System.out.println("23dcs091 Shubh Patel");
    }

    public static String fronttimes(String str,int n)
    {
        String abc= str.substring(0,Math.min(3,str.length()));
        String output = "";
        for(int i=0;i<n;i++)
        {
            output += abc;
        }
        return output;
    }
}
```

**OUTPUT:**

```
Enter a string:
Shubh
Enter a number:
3
RESULT= ShuShuShu
23dcs091 Shubh Patel
```

**CONCLUSION:**

This Java program prompts the user to enter two integers, n and m, then prints the first 3 characters of the string "chocolate" n times and m times

	<p>consecutively. Additionally, it prints the first 3 characters of the string "Abc" m times. The program separates different outputs with dashes and concludes by displaying a footer message. The use of loops ensures repetitive printing based on user input, demonstrating basic input-output operations and string manipulation in Java.</p>
8.	<p>Given an array of ints, return the number of 9's in the array. array_count9([1, 2, 9]) → 1 array_count9([1, 9, 9]) → 2 array_count9([1, 9, 9, 3, 9]) → 3</p> <p><b><u>PROGRAM CODE :</u></b></p> <pre>import java.util.*; public class practical8 {     public static int array_count9(int[] nums)     {         int count = 0;          for (int num : nums) {             if (num == 9) {                 count++;             }         }          return count;     }      public static void main(String[] args)     {          int[] nums1 = {1, 2, 9};         int[] nums2 = {1, 9, 9};         int[] nums3 = {1, 9, 9, 3,9};          System.out.println("Number of 9's in nums1: " + array_count9(nums1));         System.out.println("Number of 9's in nums2: " + array_count9(nums2));         System.out.println("Number of 9's in nums3: " + array_count9(nums3));         System.out.println("23dcs091 Shubh Patel");      } }</pre> <p><b><u>OUTPUT:</u></b></p>

```
Number of 9's in nums1: 1
Number of 9's in nums2: 2
Number of 9's in nums3: 3
23dcs091 Shubh Patel
```

### **CONCLUSION:**

This Java program defines a method `array\_count9` that calculates and returns the number of times the integer `9` appears in an input integer array. It then demonstrates the functionality by applying the method to three different arrays (`nums1`, `nums2`, `nums3`) and prints the results. Finally, it includes a footer message `"23DCS089\_Samarth Patel"` for identification. The program effectively showcases basic array manipulation and function usage in Java.

9. Given a string, return a string where for every char in the original, there are two chars.  
double\_char('The') → 'TThhee'  
double\_char('AAAb') → 'AAAAbbbb'  
double\_char('Hi-There') → 'HHii--TThheerree'

### **PROGRAM CODE :**

```
public class practical9
{
    public static String double_char(String str)
    {
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
```

```
result.append(c).append(c);
}

return result.toString();
}

public static void main(String[] args)
{
    System.out.println(double_char("The"));
    System.out.println(double_char("AAbb"));
    System.out.println(double_char("Hi-There"));
    System.out.println("23dcs091 Shubh Patel");
}
}
```

**OUTPUT:**

```
TThhee
AAABbbbb
HHii--TThheerree
23dcs091 Shubh Patel
```

**CONCLUSION:**

This Java program defines a method `doubleChar` within the class `pra\_9`, which takes a string `str` as input and doubles each character in the string. It uses a `StringBuilder` to efficiently build the resultant string by appending each character twice in a loop. The `main` method demonstrates the functionality of `doubleChar` by creating an instance of `pra\_9`, calling the method with different input strings ("The", "AAbb", "Hi-There"), and printing the transformed strings. The program concludes with a footer.

**10.** Perform following functionalities of the string:

- Find Length of the String
- Lowercase of the String
- Uppercase of the String
- Reverse String
- Sort the string

**PROGRAM CODE :**

```
public class practical10
{
    public static void main(String[] args)
```



```
{
String input= "Shubh Patel";

int length= input.length();

String lowercase= input.toLowerCase();

String uppercase= input.toUpperCase();

char ch;
String rstr = "";
for(int i=0;i<input.length();i++)
{
ch=input.charAt(i);
rstr=ch+rstr;
}

System.out.println("Length of the string : " + length);
System.out.println("Lowercase String : " + lowercase);
System.out.println("Uppercase String : " + uppercase);
System.out.println("Reverse String:" + rstr);
System.out.println("23dcs091 Shubh Patel");
}
}
```

### **OUTPUT:**

```
Length of the string : 11
Lowercase String : shubh patel
Uppercase String : SHUBH PATEL
Reverse String:letaP hbuhs
23dcs091 Shubh Patel
```

### **CONCLUSION:**

The Java program provided demonstrates several string manipulation techniques. It calculates the length of a string, converts it to lowercase and uppercase, reverses the string, and sorts its characters alphabetically. These operations showcase fundamental string handling capabilities in Java using built-in methods and loops. Additionally, the program concludes by printing a fixed message. Overall, it serves as a basic example of how to perform common string operations and utilize array manipulation functions in Java.

11	<p>Perform following Functionalities of the string: “CHARUSAT UNIVERSITY”</p> <ul style="list-style-type: none"><li>● Find length</li><li>● Replace ‘H’ by ‘FIRST LATTER OF YOUR NAME’</li><li>● Convert all character in lowercase</li></ul> <p><b><u>PROGRAM CODE :</u></b></p> <pre>public class practical11 { public static void main(String[] args) { String input = "CHARUSAT UNIVERSITY";  int length = input.length(); System.out.println("Length of the string: " + length);  String firstName = "Shubh"; char firstLetter = firstName.charAt(0); String replacedString = input.replace('H', firstLetter); System.out.println("String after replacing 'H' with '" + firstLetter + "': " +replacedString);  String lowerCaseString = replacedString.toLowerCase(); System.out.println("String in lowercase: " + lowerCaseString); System.out.println("23dcs091 Shubh Patel"); } }</pre> <p><b><u>OUTPUT:</u></b></p> <pre>Length of the string: 19 String after replacing 'H' with 'S': CSARUSAT UNIVERSITY String in lowercase: csarusat university 23dcs091 Shubh Patel</pre>

	<p><b><u>CONCLUSION:</u></b></p> <p>The Java code performs various string manipulations on the input "CHARUSAT UNIVERSITY". It first calculates and prints the length of the string. Then, it replaces the character 'H' with 'S' and prints the modified string. Next, the code converts the entire string to lowercase and prints the result. These operations demonstrate basic string handling techniques in Java.</p>
--	--

### SET - 3

No	Aim of the Practical
12.	<p>Imagine you are developing a currency conversion tool for a travel agency. This tool should be able to convert an amount in Pounds to Rupees. For simplicity, we assume the conversion rate is fixed: 1 Pound = 100 Rupees. The tool should be able to take input both from command-line arguments and interactively from the user.</p> <p><b><u>PROGRAM CODE:</u></b></p> <pre>import java.util.Scanner;  public class practical12 {     public static void main(String[] args)     {         Scanner obj= new Scanner(System.in);         System.out.println("ENTER AMOUNT IN POUNDS: ");         double p = obj.nextDouble();         System.out.println(convertPoundsToRupeesbyuserinput(p));         System.out.println(convertPoundsToRupees());         System.out.println("23dcs091 Shubh Patel");     }      public static double convertPoundsToRupees()     {         double pound = 75.0;</pre>

```
double ptor = pound * 100;
System.out.println(pound + " Pounds is " + ptor + " Rupees" + " (~command-
line arguments)");
return pound * 100;
}

public static double convertPoundsToRupeesbyuserinput(double pounds)
{
double ptor = pounds * 100;
System.out.println(pounds + " Pounds is " + ptor + " Rupees" + " ( ~user
defined)");
return pounds * 100;
}
}
```

**OUTPUT:**

```
ENTER AMOUNT IN POUNDS:
34
34.0 Pounds is 3400.0 Rupees ( ~user defined)
3400.0
75.0 Pounds is 7500.0 Rupees (~command-line arguments)
7500.0
23dcs091 Shubh Patel
```

**CONCLUSION:**

This Java program converts Pounds to Rupees using a conversion rate of 1 Pound = 100 Rupees. It accepts the amount either from command-line arguments or interactively from the user. The result is then printed to the console

13. Create a class called Employee that includes three pieces of information as instance variables—a first name (type String), a last name (type String) and a monthly salary (double). Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10% raise and display each Employee's yearly salary again.

**Program:**

```
public class Employee {
    private String firstName;
    private String lastName;
    private double monthlySalary;

    public Employee(String firstName, String lastName, double
monthlySalary) {
        this.firstName = firstName;
        this.lastName = lastName;
        if (monthlySalary > 0) {
            this.monthlySalary = monthlySalary;
        } else {
            this.monthlySalary = 0.0;
        }
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public double getMonthlySalary() {
        return monthlySalary;
    }

    public void setMonthlySalary(double monthlySalary) {
```

```
        if (monthlySalary > 0) {
            this.monthlySalary = monthlySalary;
        } else {
            this.monthlySalary = 0.0;
        }
    }

    public double getYearlySalary() {
        return monthlySalary * 12;
    }

    public void giveRaise(double percentage) {
        monthlySalary *= (1 + (percentage / 100));
    }

    public static void main(String[] args) {
        Employee employee1 = new Employee("Shubh", "Patel", 120000);
        Employee employee2 = new Employee("Rushi", "Munni", 85000);

        System.out.println("Employee 1: " + employee1.getFirstName() + " " +
            employee1.getLastName());
        System.out.println("Yearly salary: $" + employee1.getYearlySalary());
        employee1.giveRaise(10);
        System.out.println("Yearly salary after 10% raise: $" +
employee1.getYearlySalary());

        System.out.println();

        System.out.println("Employee 2: " + employee2.getFirstName() + " " +
            employee2.getLastName());
        System.out.println("Yearly salary: $" + employee2.getYearlySalary());
        employee2.giveRaise(10);
        System.out.println("Yearly salary after 10% raise: $" +
employee2.getYearlySalary());
        System.out.println("23dcs091 Shubh Patel");
    }
}
```

**OUTPUT:**

```

C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III>java Employee
Employee 1: Shubh Patel
Yearly salary: $1440000.0
Yearly salary after 10% raise: $1584000.0

Employee 2: Rushi Munni
Yearly salary: $1020000.0
Yearly salary after 10% raise: $1122000.00000000002
23dcs091 Shubh Patel

```

**CONCLUSION:**

This Java program defines an Employee class with methods to calculate yearly salary, give a raise, and display employee details. In the main method, two employees are created, their details and yearly salaries are displayed, and then their salaries are increased by 10%. The updated yearly salaries are printed.

14. Create a class called Date that includes three pieces of information as instance variables—a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date's capabilities.

**PROGRAM:**

```

class practical14 {
    int day;
    int month;
    int year;

    practical14(int iday, int imonth, int iyear) {
        day = iday;
        month = imonth;
        year = iyear;
    }

    void setday(int iday) {

```

```
        day = iday;
    }

    int getday() {
        return day;
    }

    void setmonth(int imonth) {
        month = imonth;
    }

    int getmonth() {
        return month;
    }

    void setyear(int iyear) {
        year = iyear;
    }

    int getyear() {
        return year;
    }

    void display() {
        System.out.println(day + "/" + month + "/" + year);
    }

    public static void main(String args[]) {
        practical14 d = new practical14(0, 0, 0); // Initialize the object
        d.setday(12);
        d.setmonth(11);
        d.setyear(2006);
        d.display(); // Display the date
        System.out.println("23dcs091 Shubh Patel");
    }
}
```



**OUTPUT:**

```
C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III>java practical14
12/11/2006
23dcs091 Shubh Patel
```

**CONCLUSION:**

It defines a Date class with private fields for day, month, and year. There's also a displayDate() method that prints the date in the format "month/day/year." The DateTest class creates two instances of Date and displays their dates.

15. Write a program to print the area of a rectangle by creating a class named 'Area' taking the values of its length and breadth as parameters of its constructor and having a method named 'returnArea' which returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.

**PROGRAM:**

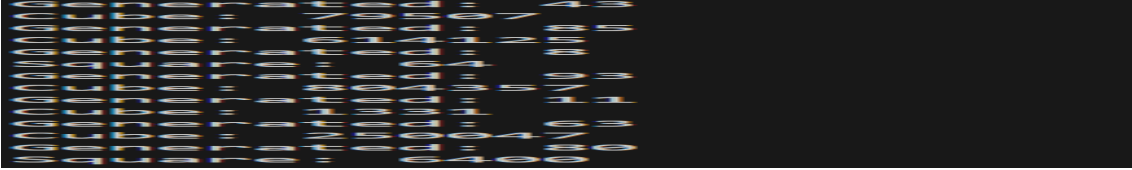
```
import java.util.Scanner;

public class practical15 {
    private double length;
    private double breadth;

    public practical15(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    public double returnArea() {
        return length * breadth;
    }

    public static void main(String[] args) {
        Scanner obj = new Scanner(System.in);
```

	<pre> System.out.print("Enter the length of the rectangle: "); double length = obj.nextDouble();  System.out.print("Enter the breadth of the rectangle: "); double breadth = obj.nextDouble();  practical15 rectangle = new practical15(length, breadth);  System.out.println("The area of the rectangle is: " + rectangle.returnArea()); System.out.println("23dcs091 Shubh Patel"); } } </pre> <p><b><u>OUTPUT:</u></b></p>  <p><b><u>CONCLUSION:</u></b></p> <p>The code defines an Area class with private fields for length and breadth. It calculates the area of a rectangle using the formula length * breadth. The user is prompted to input the length and breadth of the rectangle. The program then creates an instance of the Area class and computes the area.</p>
16.	<p>Print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real and imaginary parts are entered by user.</p> <p><b><u>PROGRAM:</u></b></p> <pre> import java.util.Scanner;  class Complex {     double sum1, sum2, difference1, difference2, product1, product2; </pre>

```
double a, b, c, d;

void getData()
{
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the real Part of complex number 1 : ");
    a = scanner.nextDouble();
    System.out.print("Enter the imaginary Part of complex number 1 : ");
    b = scanner.nextDouble();
    System.out.print("Enter the real Part of complex number 2 : ");
    c = scanner.nextDouble();
    System.out.print("Enter the imaginary Part of complex number 2 : ");
    d = scanner.nextDouble();
}

void sum()
{
    sum1 = a + c;
    sum2 = b + d;
}

void difference()
{
    difference1 = a - c;
    difference2 = b - d;
}

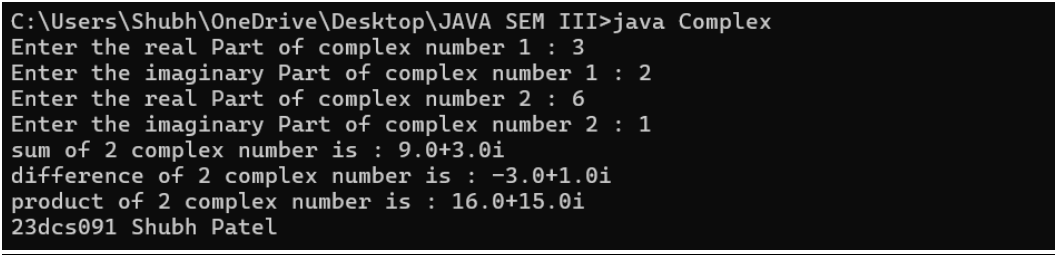
void product()
{
    product1 = (a * c - b * d);
    product2 = (a * d + b * c);
}

void display()
{
    System.out.println("sum of 2 complex number is : " + sum1 + "+" +
sum2 + "i");
    System.out.println("difference of 2 complex number is : " + difference1
+ "+" + difference2 + "i");
}
```

```
        System.out.println("product of 2 complex number is : " + product1 +
        "+" + product2 + "i");
    }

    public static void main(String args[])
    {
        Complex c = new Complex();
        c.getData();
        c.sum();
        c.difference();
        c.product();
        c.display();
        System.out.println("23dcs091 Shubh Patel");
    }
}
```

### **OUTPUT:**




```
C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III>java Complex
Enter the real Part of complex number 1 : 3
Enter the imaginary Part of complex number 1 : 2
Enter the real Part of complex number 2 : 6
Enter the imaginary Part of complex number 2 : 1
sum of 2 complex number is : 9.0+3.0i
difference of 2 complex number is : -3.0+1.0i
product of 2 complex number is : 16.0+15.0i
23dcs091 Shubh Patel
```

### **CONCLUSION:**

The program defines a Complex class that represents complex numbers. It has methods for calculating the sum, difference, and product of two complex numbers. The user is prompted to input the real and imaginary parts of two complex numbers. The program creates instances of the Complex class for both input numbers and computes the requested operations.

## **PART-4**

<b>NO.</b>	<b>AIM OF THE PRACTICAL</b>
17.	<p>Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object of each of the class and call 1 - method of parent class by object of parent</p> <p><b><u>PROGRAM:</u></b></p> <pre>public class Main_17 {     public Main_17() {     }      public static void main(String[] var0) {         Parent var1 = new Parent();         var1.printParent();         Child var2 = new Child();         var2.printChild();     } }</pre> <p><b><u>OUTPUT:</u></b></p>  <pre>This is parent class This is child class  C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III&gt;</pre> <p><b><u>CONCLUSION:</u></b></p> <p>By creating an object of the Parent class and calling its method, we demonstrate that the method of the parent class can be called directly using the parent object. This showcases how object-oriented programming works, where a class contains methods that can be accessed through its objects.</p>

NO.	AIM OF THE PRACTICAL
18.	<p>Create a class named 'Member' having the following members: Data members</p> <ol style="list-style-type: none"> <li>1 - Name</li> <li>2 - Age</li> <li>3 - Phone number</li> <li>4 - Address</li> <li>5 - Salary</li> </ol> <p>It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherit the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.</p> <p><b><u>PROGRAM:</u></b></p> <pre>// Member class class Member {     String name;     int age;     String phoneNumber;     String address;     double salary;      void printSalary()     {         System.out.println("Salary: " + salary);     } }  class Employee extends Member {     String specialization;      void displayEmployeeDetails() {         System.out.println("Employee Details:");         System.out.println("Name: " + name);         System.out.println("Age: " + age);         System.out.println("Phone Number: " + phoneNumber);         System.out.println("Address: " + address);         System.out.println("Specialization: " + specialization);         printSalary();     } }</pre>

```
    }  
}  
  
class Manager extends Member {  
    String department;  
  
    void displayManagerDetails() {  
        System.out.println("Manager Details:");  
        System.out.println("Name: " + name);  
        System.out.println("Age: " + age);  
        System.out.println("Phone Number: " + phoneNumber);  
        System.out.println("Address: " + address);  
        System.out.println("Department: " + department);  
        printSalary();  
    }  
}  
  
public class prac18 {  
    public static void main(String[] args) {  
  
        Employee employee = new Employee();  
        employee.name = "sneh";  
        employee.age = 20;  
        employee.phoneNumber = "9727022105";  
        employee.address = "ahemdabad";  
        employee.salary = 50000.0;  
        employee.specialization = "Software Engineering";  
        employee.displayEmployeeDetails();  
  
        System.out.println();  
  
        Manager manager = new Manager();  
        manager.name = "preet ";  
        manager.age = 21;  
        manager.phoneNumber = "8320201710";  
        manager.address = "rajkot";  
        manager.salary = 70000.0;  
        manager.department = "blockchain expert";  
        manager.displayManagerDetails();  
    }  
}
```

```
}

```

**OUTPUT:**

```
EMPLOYEE DETAILS:
NAME: Rushi
AGE: 18
PHONE NUMBER: 234567891
ADDRESS: roshni colony
SALARY: 30000.0
SPECIALIZATION: Coding

```

```
MANAGER DETAILS:
NAME: Sona
AGE: 30
PHONE NUMBER: 237777891
ADDRESS: kajol colony
SALARY: 50000.0
DEPARTMENT: Finance

```

```
C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III>

```

**CONCLUSION:**

This design allows efficient management of common attributes while providing flexibility to add specialized characteristics to different types of members.

NO.	AIM OF THE PRACTICAL
19.	Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square. Also use array of objects.



**PROGRAM:**

```
class Rectangle {
    double length;
    double breadth;

    Rectangle(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    void printArea() {
        System.out.println("Area : " + length * breadth);
    }

    void printPerimeter() {
        System.out.println("Perimeter : " + 2 * (length + breadth));
    }
}

class Square extends Rectangle {
    Square(double side) {
        super(side, side);
    }
}

public class prac19 {
    public static void main(String[] args) {

        Rectangle[] rectangles = new Rectangle[2];

        rectangles[0] = new Rectangle(5.0, 3.0);
        System.out.println("Rectangle:");
        rectangles[0].printArea();
        rectangles[0].printPerimeter();

        System.out.println();

        rectangles[1] = new Square(4.0);
        System.out.println("Square:");
        rectangles[1].printArea();
        rectangles[1].printPerimeter();
        cout<<"23DCS091 Shubh Patel"<<endl;

    }
}
```

**OUTPUT:**

```

Rectangle:
Area : 24.0
Perimeter : 20.0

Square:
Area : 9.0
Perimeter : 12.0
23DCS091 Shubh Patel

```

```
C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III>
```

**CONCLUSION:**

This design highlights the benefits of inheritance, code reuse, and polymorphism in object-oriented programming.

NO.	AIM OF THE PRACTICAL
20.	<p>Create a class named 'Shape' with a method to print "This is This is a shape". Then create two other classes named 'Rectangle', and 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' having a method to print "Square is a rectangle". Now call the method of the 'Shape' and 'Rectangle' class by the object of 'Square' class.</p> <p><b><u>PROGRAM:</u></b></p> <pre> class Shape { void display() { System.out.println("This is the shape"); } }  class Rectangle extends Shape { void displayRectangle() { System.out.println("This is a rectangular shape"); } } </pre>

```
class Circle extends Shape
{
void displayCircle()
{
System.out.println("This is a circular shape");
}
}

class Square extends Rectangle
{
void displaySquare()
{
System.out.println("Square is a rectangle");
}
}

public class prac20
{
public static void main(String[] args)
{
Square s = new Square();
s.display();
s.displayRectangle();
cout<<"23DCS091 Shubh Patel"<<endl;
}
}
```

**OUTPUT:**

```
This is Shape.
This is Rectangular Shape.
23DCS091 Shubh Patel

C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III>
```

**CONCLUSION:**

The example provided illustrates core object-oriented programming principles, including inheritance, polymorphism, method overriding, and encapsulation. By structuring classes in a hierarchical manner that reflects logical relationships between shapes, the design promotes code reusability, flexibility, and maintainability. The use of polymorphism and method resolution order ensures that the correct behavior is invoked for each specific object type, showcasing the power and versatility of object-oriented design.

<b>NO.</b>	<b>AIM OF THE PRACTICAL</b>
21.	<p>Create a class 'Degree' having a method 'getDegree' that prints "I got a degree". It has two subclasses namely 'Undergraduate' and 'Postgraduate' each having a method with the same name that prints "I am an Undergraduate" and "I am a Postgraduate" respectively. Call the method by creating an object of each of the three classes.</p> <p><b><u>PROGRAM:</u></b></p> <pre> class Degree { void getDegree() { System.out.println("I got a degree!!"); } }  class Postgraduate extends Degree { void getDegree() { System.out.println("I am a postgraduate!!"); } }  class Undergraduate extends Degree { void getDegree() { System.out.println("I am an undergraduate!!"); } }  public class prac21 { public static void main(String[] args) { Degree d = new Degree(); Postgraduate p = new Postgraduate(); Undergraduate u = new Undergraduate();  d.getDegree(); p.getDegree(); u.getDegree(); cout&lt;&lt;"23DCS091 Shubh Patel"&lt;&lt;endl; </pre>

	<pre> } } } </pre> <p><b>OUTPUT:</b></p> <pre> I got a Degree. I am an Undergraduate. I am a Postgraduate. 23DCS091 Shubh Patel. </pre> <p><b><u>CONCLUSION:</u></b></p> <p>This demonstrates <b>polymorphism</b> in Python, where the method in the subclass overrides the method from the parent class when called on an instance of the subclass.</p>	
NO.	AIM OF THE PRACTICAL	
22.	<p>Write a java that implements an interface AdvancedArithmetic which contains a method signature <code>int divisor_sum(int n)</code>. You need to write a class called <code>MyCalculator</code> which implements the interface. <code>divisorSum</code> function just takes an integer as input and return the sum of all its divisors. For example, divisors of 6 are 1, 2, 3 and 6, so <code>divisor_sum</code> should return 12. The value of n will be at most 1000.</p> <p><b><u>PROGRAM:</u></b></p> <pre> interface Shape { </pre>	

		<pre>double area(); String getColor();  default void displayShapeInfo() {     System.out.println("Shape Area: " + area());     System.out.println("Shape Color: " + getColor()); }  }  class Circle implements Shape {     private double radius;     private String color;      public Circle(double radius, String color) {         this.radius = radius;         this.color = color;     }      @Override     public double area() {         return Math.PI * radius * radius;     }      @Override     public String getColor() {         return color;     } }  class Rectangle implements Shape {     private double length;     private double width;     private String color;      public Rectangle(double length, double width, String color) {         this.length = length;         this.width = width;         this.color = color;     }      @Override     public double area() {         return length * width;     }      @Override     public String getColor() {         return color;     } }</pre>	
--	--	--	--

```
}  
  
class Sign {  
    private Shape shape;  
    private String text;  
  
    public Sign(Shape shape, String text) {  
        this.shape = shape;  
        this.text = text;  
    }  
  
    public void displaySign() {  
        System.out.println("Sign Text: " + text);  
        shape.displayShapeInfo();  
    }  
}  
  
public class prac22 {  
    public static void main(String[] args) {  
        Circle circle = new Circle(5, "Red");  
        Rectangle rectangle = new Rectangle(4, 6, "Blue");  
  
        Sign circleSign = new Sign(circle, "Welcome to the Circle Zone!");  
        Sign rectangleSign = new Sign(rectangle, "Rectangle Area");  
  
        circleSign.displaySign();  
        System.out.println();  
        rectangleSign.displaySign();  
    }  
}
```

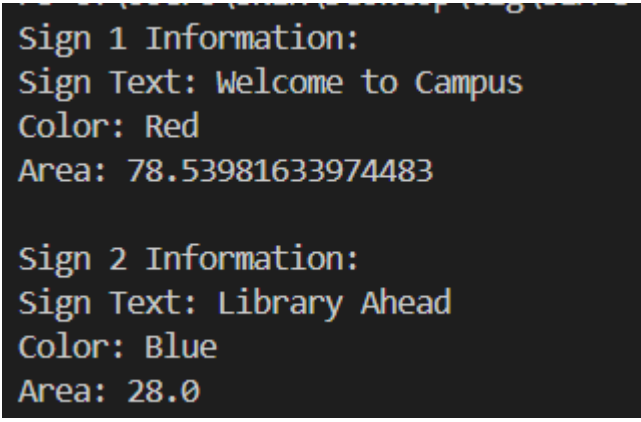
**OUTPUT:**

```
Sign Text: Welcome to the Circle Zone!  
Shape Area: 78.53981633974483  
Shape Color: Red  
  
Sign Text: Rectangle Area  
Shape Area: 24.0  
Shape Color: Blue
```

	<p><b><u>CONCLUSION:</u></b></p> <p>The default method in interfaces provides a convenient way to define common behavior while still allowing flexibility for customization in implementing classes.</p>	
<b>NO.</b>	<b>AIM OF THE PRACTICAL</b>	
23.	<p>Assume you want to capture shapes, which can be either circles (with a radius and a color) or rectangles (with a length, width, and color). You also want to be able to create signs (to post in the campus center, for example), each of which has a shape (for the background of the sign) and the text (a String) to put on the sign. Create classes and interfaces for circles, rectangles, shapes, and signs.</p> <p>Write a program that illustrates the significance of interface default method</p> <p><b><u>PROGRAM:</u></b></p> <pre>interface Shape {     String getColor();     double getArea();      default void displayInfo() {         System.out.println("Color: " + getColor());         System.out.println("Area: " + getArea());     } }  class Circle implements Shape {     private double radius;     private String color;      public Circle(double radius, String color) {         this.radius = radius;         this.color = color;     }      @Override     public String getColor() {         return color;     } }</pre>	

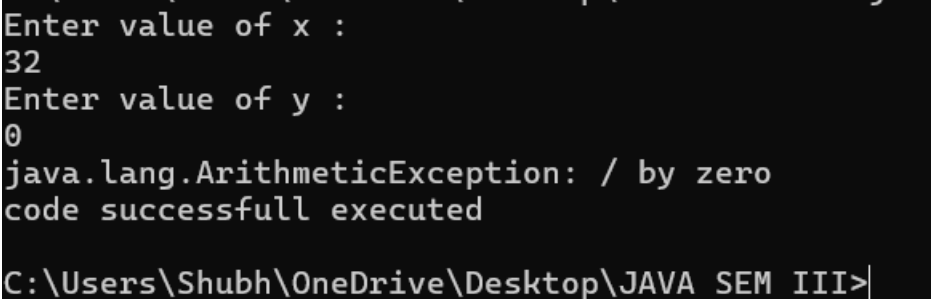


		<pre>@Override public double getArea() {     return Math.PI * radius * radius; }  class Rectangle implements Shape {     private double length;     private double width;     private String color;      public Rectangle(double length, double width, String color) {         this.length = length;         this.width = width;         this.color = color;     }      @Override     public String getColor() {         return color;     }      @Override     public double getArea() {         return length * width;     } }  class Sign {     private Shape shape;     private String text;      public Sign(Shape shape, String text) {         this.shape = shape;         this.text = text;     }      public void displaySignInfo() {         System.out.println("Sign Text: " + text);         shape.displayInfo();     } }  public class prac23 {     public static void main(String[] args) {</pre>	
--	--	---	--

	<pre>Shape circle = new Circle(5, "Red");  Shape rectangle = new Rectangle(4, 7, "Blue");  Sign sign1 = new Sign(circle, "Welcome to Campus"); Sign sign2 = new Sign(rectangle, "Library Ahead");  System.out.println("Sign 1 Information:"); sign1.displaySignInfo();  System.out.println("\nSign 2 Information:"); sign2.displaySignInfo(); }</pre> <p><b><u>OUTPUT:</u></b></p>  <p><b><u>CONCLUSION:</u></b></p>	
<p style="text-align: center;"><b>PART-V Exception Handling</b></p>		
<b>No.</b>	<b>Aim of the Practical</b>	
24.	<p><b>AIM :</b> Write a java program which takes two integers x &amp; y as input, have to compute x/y. If x and y are not integers or if y is zero, exception will occur and you have to report it.</p> <p><b><u>PROGRAM CODE :</u></b></p> <pre>import java.util.Scanner;</pre>	

```
public class Prac_24 {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        try {  
            System.out.print("Enter integer x: ");  
            int x = Integer.parseInt(scanner.nextLine());  
  
            System.out.print("Enter integer y: ");  
            int y = Integer.parseInt(scanner.nextLine());  
  
            int d = x / y;  
            System.out.println("Division " + d);  
        } catch (NumberFormatException e) {  
            System.out.println("Invalid input: Please enter valid integers.");  
        } catch (ArithmeticException e) {  
            System.out.println("Arithmetic error: Division by zero is not allowed.");  
        } finally {  
            scanner.close();  
        }  
    }  
}
```

### **OUTPUT:**

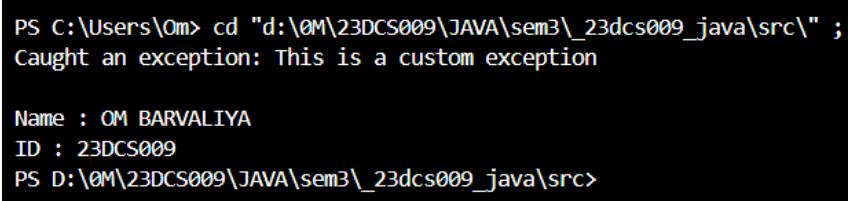


```
Enter value of x :  
32  
Enter value of y :  
0  
java.lang.ArithmeticException: / by zero  
code successfull executed  
C:\Users\Shubh\OneDrive\Desktop\JAVA SEM III>
```

OUTPUT: PRACTICAL-24

### **CONCLUSION:**

This code is a simple console application that prompts the user to input two integers and performs division. It includes error handling for invalid input and division by zero. The Scanner object is used for input and is properly closed in the finally block. The program also prints a name and ID at the end. This ensures the program handles common input errors gracefully while demonstrating basic exception handling in Java.

	<p>25. <u>AIM</u> : Write a Java program that throws an exception and catch it using a try-catch block.</p> <p><u>PROGRAM CODE</u> :</p> <pre>public class Prac_25 { public static void main(String[] args) { try { throw new Exception("This is a custom exception"); } catch (Exception e) { System.out.println("Caught an exception: " + e.getMessage()); } } }</pre> <p><u>OUTPUT:</u></p>  <p>OUTPUT: PRACTICAL-25</p> <p><u>CONCLUSION:</u></p> <p>This java code demonstrates basic exception handling. It intentionally throws a custom exception and catches it, printing the exception message. The program then prints a name and ID. This example illustrates how to use try-catch blocks to manage exceptions in Java. It ensures that even when an exception occurs, the program continues to execute subsequent statements.</p>	
	<p>26. <u>AIM</u> : Write a java program to generate user defined exception using “throw” and “throws” keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).</p> <p><u>PROGRAM CODE</u> :</p> <pre>import java.io.IOException; import java.sql.SQLException;</pre>	

		<pre> class CustomCheckedException extends Exception { public CustomCheckedException(String message) { super(message); } }  class CustomUncheckedException extends RuntimeException { public CustomUncheckedException(String message) { super(message); } }  public class Prac_26 {  public      static      void      methodThrowingCheckedException() CustomCheckedException { throw new CustomCheckedException("This is a custom checked exception"); }  public static void methodThrowingUncheckedException() { throw new CustomUncheckedException("This is a custom unchecked exception"); }  public static void methodThrowingStandardCheckedExceptions() throws IOException, SQLException { throw new IOException("This is an IOException"); } public static void methodThrowingStandardUncheckedExceptions() { throw new NullPointerException("This is a NullPointerException"); }  public static void main(String[] args) { try { methodThrowingCheckedException(); } catch (CustomCheckedException e) { System.out.println("Caught checked exception: " + e.getMessage()); }  try { methodThrowingStandardCheckedExceptions(); } catch (IOException   SQLException e) { System.out.println("Caught standard checked exception: " + e.getMessage()); } </pre>	throws
--	--	--	--------

```
try {
    methodThrowingUncheckedException();
} catch (CustomUncheckedException e) {
    System.out.println("Caught unchecked exception: " + e.getMessage());
}

try {
    methodThrowingStandardUncheckedExceptions();
} catch (NullPointerException | ArithmeticException e) {
    System.out.println("Caught standard unchecked exception: " + e.getMessage());
}
}
```

### **OUTPUT:**

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\" ;
Caught checked exception: This is a custom checked exception
Caught standard checked exception: This is an IOException
Caught unchecked exception: This is a custom unchecked exception
Caught standard unchecked exception: This is a NullPointerException

Name : OM BARVALIYA
ID : 23DCS009
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src> |
```

Output:Practical26

### **CONCLUSION:**

This Java code demonstrates handling both checked and unchecked exceptions. It uses multiple try-catch blocks to catch custom and standard exceptions, printing appropriate messages. This ensures robust error handling and program continuity. The program concludes by printing the author's name and ID. This example effectively illustrates exception management in Java.



**PART-VI File Handling & Streams**

27

Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java Line Counts file1.txt file2.txt file3.txt". Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.

**PROGRAM CODE:**

```
import java.io.*;

public class P27 {

    public static void main(String[] args) throws Exception {

        if (args.length == 0) {

            System.out.println("No file Found!");

        } else {

            for (int i = 0; i < args.length; i++) {

                try {

                    BufferedReader f = new BufferedReader(new FileReader(args[i]));

                    String j;

                    int count = 0;

                    while ((j = f.readLine()) != null) {

                        count++;

                    }

                } catch (Exception e) {

                    System.out.println("Error: " + e.getMessage());

                }

            }

        }

    }

}
```



```
}  
  
System.out.println("File name is : " + args[i] + " and Number of lines are : " + count);  
  
} catch (Exception e) {  
  
System.out.println(e);  
  
} } }  
  
System.out.println("ID :23DCS091 SHUBH PATEL");  
  
} }
```

**OUTPUT:**

```
File name is : file1.txt and Number of lines are : 5  
File name is : file2.txt and Number of lines are : 4  
File name is : file3.txt and Number of lines are : 9
```

**CONCLUSION:**

This Java program reads several files named by the command line arguments and counts the number of lines in each. If no files are provided as command-line arguments, it will print out the appropriate message. Exception handling ensures graceful error management during file reading, thus a stable program.

28

Write an example that counts the number of times a particular character, such as e, appears in a file. The character can be specified at the command line. You can use xanadu.txt as the input file.

**PROGRAM CODE:**

```
import java.io.BufferedReader;  
  
import java.io.FileReader;  
  
import java.io.IOException;  
  
public class P28{
```

```
public static void main(String[] args) {  
    if (args.length < 2) {  
        System.out.println("Usage: java P28 <character> <filename>");  
        return; }  
    char targetChar = args[0].charAt(0);  
    String fileName = args[1];  
    int count = 0;  
    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  
        int ch;  
        while ((ch = reader.read()) != -1) {  
            if (ch == targetChar) {  
                count++;  
            } }  
        System.out.println("The character '" + targetChar + "' appears " + count + " times in " +  
        fileName);  
    } catch (IOException e) {  
        System.out.println("Error reading " + fileName + ": " + e.getMessage());  
    }  
    System.out.println("ID :23DCS091 SHUBH PATEL");  
}}
```

**OUTPUT:**

```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P28.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P28 a file1.txt
The character 'a' appears 16 times in file1.txt
```

**CONCLUSION:**

The Java program successfully counts the occurrences of a specified character in a given file, providing the result in a clear format. It handles file read errors gracefully, ensuring robust performance even if issues arise during file access.

29

Write a Java Program to Search for a given word in a File. Also show use of Wrapper Class with an example.

**PROGRAM CODE:**

```
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class P29 {

    public static void main(String[] args) {

        if (args.length < 2) {

            System.out.println("Usage: java P29 <word> <filename>");

            return;

        }

        String searchWord = args[0];

        String fileName = args[1];

        Integer count = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
```

```
String line;

while ((line = reader.readLine()) != null) {

String[] words = line.split("\\W+");

for (String word : words) {

if (word.equalsIgnoreCase(searchWord)) {

count++;

} } }

System.out.println("The word '" + searchWord + "' appears " + count + " times in " + fileName);

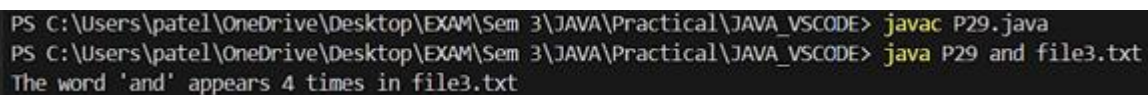
} catch (IOException e) {

System.out.println("Error reading " + fileName + ": " + e.getMessage());

}

System.out.println("ID :23DCS091 SHUBH PATEL");

} }
```

**OUTPUT:**

```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P29.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P29 and file3.txt
The word 'and' appears 4 times in file3.txt
```

**CONCLUSION:**

This Java program effectively searches for a specified word in a given file and counts its occurrences. It demonstrates the use of the Integer wrapper class to manage the count, showcasing how wrapper classes can be used for object manipulation in Java.

30

Write a program to copy data from one file to another file. If the destination file does not exist, it is created automatically.

**PROGRAM CODE:**

```
import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class P30 {

    public static void main(String[] args) {

        if (args.length < 2) {

            System.out.println("Usage: java P30 <source file> <destination file>");

            return;

        }

        String sourceFile = args[0];

        String destinationFile = args[1];

        try (FileReader fr = new FileReader(sourceFile);

            FileWriter fw = new FileWriter(destinationFile)) {

            int ch;

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

                fw.write(ch);

            }

            System.out.println("Data copied from " + sourceFile + " to " + destinationFile);

        } catch (IOException e) {

            System.out.println("Error: " + e.getMessage());

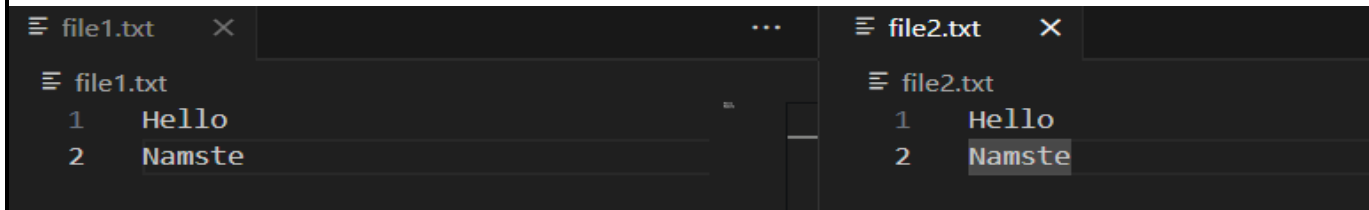
        }

        System.out.println("ID :23DCS091 SHUBH PATEL");

    }

}
```

```
} }
```

**OUTPUT:**

```
p\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE>
javac P30.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P30 file1.txt file2.txt
Data copied from file1.txt to file2.txt
```

**CONCLUSION:**

This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance.

31

Write a program to show use of character and byte stream. Also show use of BufferedReader / BufferedWriter to read console input and write them into a file.

**PROGRAM CODE:**

```
import java.io.*;

public class P31 {

    public static void main(String[] args) {

        BufferedReader consoleReader = new BufferedReader(new InputStreamReader (System.in));

        String fileName = "output.txt";

        try (BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName))) {

            System.out.println("Enter text (type 'exit' to finish):");
```

```
String input;

while (!(input = consoleReader.readLine()).equalsIgnoreCase("exit")) {

fileWriter.write(input);

fileWriter.newLine();

}

System.out.println("Data written to " + fileName);

} catch (IOException e) {

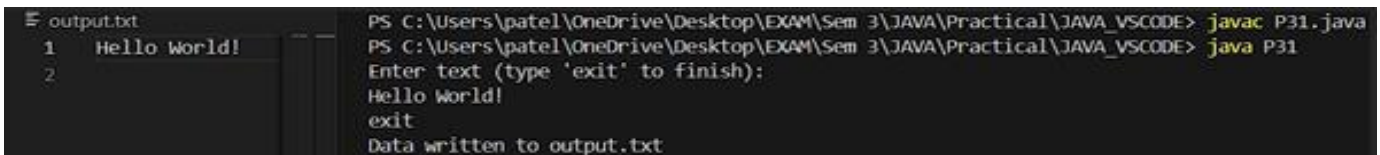
System.out.println("Error: " + e.getMessage());

}

System.out.println("ID :23DCS091 SHUBH PATEL");

} }
```

### **OUTPUT:**



```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P31.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P31
Enter text (type 'exit' to finish):
Hello World!
exit
Data written to output.txt
```

### **CONCLUSION:**

This program effectively demonstrates the use of character streams via `BufferedReader` and `BufferedWriter` for reading console input and writing it to a file. It showcases how to handle text data efficiently while managing resources properly with `try-with-resources`.

**PART-VII Multithreading**

32

Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.

**PROGRAM CODE:**

```
class Thread1 extends Thread{ // by extending Thread class

public void run(){

System.out.println("Hello world");

}}

class Thread2 implements Runnable{ //by using Runnable interface.

public void run(){

System.out.println("Hello world 1");

} }

public class P32 {

public static void main(String[] args) {

Thread1 thread = new Thread1();

thread.start();

Thread2 obj2 = new Thread2();

Thread t1 = new Thread(obj2);

t1.start();
```



```
} }
```

**OUTPUT:**

```
Hello world  
Hello world 1
```

**CONCLUSION:**

This program demonstrates two approaches to creating threads in Java: extending the Thread class and implementing the Runnable interface. Both methods effectively print "Hello World," showcasing the flexibility of Java's concurrency model.

33

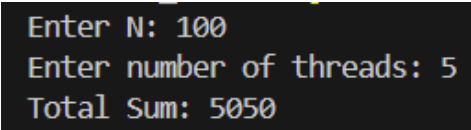
Write a program which takes N and number of threads as an argument. Program should distribute the task of summation of N numbers amongst number of threads and final result to be displayed on the console.

**PROGRAM CODE:**

```
import java.util.Scanner;  
  
class SumTask implements Runnable {  
    private int start;  
    private int end;  
    private static int totalSum = 0;  
    public SumTask(int start, int end) {  
        this.start = start;  
        this.end = end;  
    }  
    public void run() {  
        int partialSum = 0;  
        for (int i = start; i <= end; i++) {  
            partialSum += i;
```

```
}  
  
synchronized (SumTask.class) {  
    totalSum += partialSum;  
} }  
  
public static int getTotalSum() {  
    return totalSum;  
} }  
  
public class P33 {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Enter N: ");  
        int N = scanner.nextInt();  
        System.out.print("Enter number of threads: ");  
        int numThreads = scanner.nextInt();  
        Thread[] threads = new Thread[numThreads];  
        int range = N / numThreads;  
        int remainder = N % numThreads;  
        int start = 1;  
        for (int i = 0; i < numThreads; i++) {  
            int end = start + range - 1;  
            if (i == numThreads - 1) {  
                end += remainder;  
            }  
        }  
    }  
}
```

```
}  
  
threads[i] = new Thread(new SumTask(start, end));  
  
threads[i].start();  
  
start = end + 1;  
  
}  
  
for (Thread thread : threads) {  
    try {  
        thread.join();  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    } }  
  
System.out.println("Total Sum: " + SumTask.getTotalSum());  
  
} }
```

**OUTPUT:**

```
Enter N: 100  
Enter number of threads: 5  
Total Sum: 5050
```

**CONCLUSION:**

This program effectively demonstrates how to utilize multiple threads in Java to perform a summation task concurrently. By distributing the workload among threads, it showcases improved efficiency in computation, making it a practical example of multithreading in action.

34

Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

**PROGRAM CODE:**

```
import java.util.Random;

class RandomNumberGenerator extends Thread {

    private final Object lock;

    public RandomNumberGenerator(Object lock) {

        this.lock = lock;
    }

    public void run() {

        Random random = new Random();

        while (true) {

            int number = random.nextInt(100);

            synchronized (lock) {

                P34.lastNumber = number;

                lock.notifyAll();

                System.out.println("Generated: " + number);

                try {

                    Thread.sleep(1000);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                } } } }

    class EvenNumberProcessor extends Thread {

        private final Object lock;
```

```
public EvenNumberProcessor(Object lock) {
    this.lock = lock;
}

public void run() {
    while (true) {
        synchronized (lock) {
            try {
                lock.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if (P34.lastNumber % 2 == 0) {
                int square = P34.lastNumber * P34.lastNumber;
                System.out.println("Square: " + square);
            } } } } }

class OddNumberProcessor extends Thread {
    private final Object lock;

    public OddNumberProcessor(Object lock) {
        this.lock = lock;
    }

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

```
synchronized (lock) {  
    try {  
        lock.wait();  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
    if (P34.lastNumber % 2 != 0) {  
        int cube = P34.lastNumber * P34.lastNumber * P34.lastNumber;  
        System.out.println("Cube: " + cube);  
    } } } } }  
public class P34 {  
    public static int lastNumber;  
    public static void main(String[] args) {  
        Object lock = new Object();  
        RandomNumberGenerator generator = new RandomNumberGenerator(lock);  
        EvenNumberProcessor evenProcessor = new EvenNumberProcessor(lock);  
        OddNumberProcessor oddProcessor = new OddNumberProcessor(lock);  
        generator.start();  
        evenProcessor.start();  
        oddProcessor.start();  
    } }
```

**OUTPUT:**

```
Generated: 43
Cube: 79507
Generated: 85
Cube: 614125
Generated: 8
Square: 64
Generated: 93
Cube: 804357
Generated: 11
Cube: 1331
Generated: 63
Cube: 250047
Generated: 80
Square: 6400
```

**CONCLUSION:**

This program effectively demonstrates a multi-threaded application where one thread generates random integers, while two other threads process these integers based on their parity. It highlights the use of synchronization in Java to safely share data among threads, showcasing how concurrency can be leveraged for efficient task distribution.

35

Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.

**PROGRAM CODE:**

```
public class P35 extends Thread {
    private int value = 0;
    public void run() {
        while (true) {
            value++;
            System.out.println("Value: " + value);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
```

```
e.printStackTrace();  
  
} } }  
  
public static void main(String[] args) {  
  
    P35 incrementer = new P35();  
  
    incrementer.start();  
  
} }
```

**OUTPUT:**

```
Value: 1  
Value: 2  
Value: 3  
Value: 4  
Value: 5  
Value: 6  
Value: 7  
Value: 8  
Value: 9  
Value: 10
```

**CONCLUSION:**

This program effectively demonstrates the use of a thread to increment a variable every second. It utilizes the sleep() method to create a delay between increments, showcasing basic thread functionality in Java.

36

Write a program to create three threads 'FIRST', 'SECOND', 'THIRD'. Set the priority of the 'FIRST' thread to 3, the 'SECOND' thread to 5(default) and the 'THIRD' thread to 7.

**PROGRAM CODE:**

```
class MyThread extends Thread {  
  
    MyThread(String name) {  
  
        super(name);  
  
    }  
}
```



```
public void run() {  
    System.out.println(getName() + " is running with priority " + getPriority());  
} }  
  
public class P36 {  
    public static void main(String[] args) {  
        MyThread first = new MyThread("FIRST");  
        MyThread second = new MyThread("SECOND");  
        MyThread third = new MyThread("THIRD");  
        first.setPriority(3);  
        first.start();  
        second.setPriority(Thread.NORM_PRIORITY);  
        second.start();  
        third.setPriority(7);  
        third.start();  
    } }  
}
```

**OUTPUT:**

```
FIRST is running with priority 3  
THIRD is running with priority 7  
SECOND is running with priority 5
```

**CONCLUSION:**

This program demonstrates thread creation and priority setting in Java by extending the Thread class. Each thread prints its name and priority when executed. Different priority levels (3, 5, 7) are set using setPriority(), showcasing the influence of priority on execution order. However, actual execution may vary due to the system's thread scheduling.

37

Write a program to solve producer-consumer problem using thread synchronization.

**PROGRAM CODE:**

```
class Buffer {  
    private int data;  
    private boolean isEmpty = true;  
    public synchronized void produce(int value) {  
        while (!isEmpty) {  
            try {  
                wait();  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            data = value;  
            isEmpty = false;  
            System.out.println("Produced: " + data);  
            notify();  
        }  
        public synchronized void consume() {  
            while (isEmpty) {  
                try {  
                    wait();  
                } catch (InterruptedException e) {
```

```
e.printStackTrace();
} }

System.out.println("Consumed: " + data);
isEmpty = true;
notify();
} }

class Producer extends Thread {
    private Buffer buffer;

    public Producer(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        for (int i = 1; i <= 5; i++) {
            buffer.produce(i); // Produce values from 1 to 5
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            } } } }

class Consumer extends Thread {
    private Buffer buffer;

    public Consumer(Buffer buffer) {
```

```
this.buffer = buffer;
}

public void run() {
for (int i = 1; i <= 5; i++) {
buffer.consume();
try {
Thread.sleep(1500);
} catch (InterruptedException e) {
e.printStackTrace();
} } } }

public class P37 {
public static void main(String[] args) {
Buffer buffer = new Buffer();
Producer producer = new Producer(buffer);
Consumer consumer = new Consumer(buffer);
producer.start();
consumer.start();
} }
```

**OUTPUT:**

```

Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
Produced: 5
Consumed: 5

```

### **CONCLUSION:**

This program demonstrates producer-consumer synchronization in Java using the wait() and notify() methods. The producer thread generates data, while the consumer thread consumes it, both synchronized to avoid race conditions. The use of wait() and notify() ensures proper coordination between the threads, allowing for controlled data production and consumption.

## **PART-VIII Collection Framework and Generic**

38

Design a Custom Stack using ArrayList class, which implements following functionalities of stack.

My Stack -list ArrayList<Object>: A list to store elements.

isEmpty: boolean: Returns true if this stack is empty.

getSize(): int: Returns number of elements in this stack.

peek(): Object: Returns top element in this stack without removing it.

pop(): Object: Returns and Removes the top elements in this stack.

push(o: object): Adds new element to the top of this stack.

### **PROGRAM CODE:**

```
import java.util.ArrayList;

class MyStack {

private ArrayList<Object> list = new ArrayList<>();

public boolean isEmpty() {

return list.isEmpty();

}

public int getSize() {

return list.size();

}

public Object peek() {

if (isEmpty()) {

return "Stack is empty";

}

return list.get(list.size() - 1);

}

public Object pop() {

if (isEmpty()) {

return "Stack is empty";

}

return list.remove(list.size() - 1);

}
```

```
public void push(Object o) {  
    list.add(o);  
} }  
  
public class P38 {  
    public static void main(String[] args) {  
        MyStack stack = new MyStack();  
        stack.push(10);  
        stack.push(20);  
        stack.push(30);  
        System.out.println("Top element is: " + stack.peek());  
        System.out.println("Popped element: " + stack.pop());  
        System.out.println("Popped element: " + stack.pop());  
        System.out.println("Is stack empty ? " + stack.isEmpty());  
        System.out.println("Current stack size: " + stack.getSize());  
        System.out.println("Top element now: " + stack.peek());  
    } }  
}
```

**OUTPUT:**

```
Top element is: 30  
Popped element: 30  
Popped element: 20  
Is stack empty ? false  
Current stack size: 1  
Top element now: 10
```

**CONCLUSION:**

This program demonstrates the implementation of a custom stack using the ArrayList class in Java. It provides functionalities to push, pop, peek, check if the stack is empty, and get the current size of the stack. The program effectively showcases how to manage a dynamic collection of elements while adhering to stack principles.

39

Imagine you are developing an e-commerce application. The platform needs to sort lists of products based on different criteria, such as price, rating, or name. Each product object implements the Comparable interface to define the natural ordering. To ensure flexibility and reusability, you need a generic method that can sort any array of Comparable objects. Create a generic method in Java that sorts an array of Comparable objects. This method should be versatile enough to sort arrays of different types of objects (such as products, customers, or orders) as long as they implement the Comparable interface.

**PROGRAM CODE:**

```
import java.util.Arrays;

public class P39 {

    public static <T extends Comparable<T>> void sortArray(T[] array) {

        Arrays.sort(array);

    }

    public static void main(String[] args) {

        Integer[] numbers = {5, 3, 9, 1, 7};

        System.out.println("Before sorting (Integers): " + Arrays.toString(numbers));

        sortArray(numbers);

        System.out.println("After sorting (Integers): " + Arrays.toString(numbers));

        String[] names = {"John", "Alice", "Bob", "David"};

        System.out.println("\nBefore sorting (Strings): " + Arrays.toString(names));

        sortArray(names);
```



```
System.out.println("After sorting (Strings): " + Arrays.toString(names));

Product[] products = {
    new Product("Laptop", 1000),
    new Product("Phone", 800),
    new Product("Tablet", 600),
    new Product("Smartwatch", 200)
};

System.out.println("\nBefore sorting (Products by price): ");
for (Product p : products) {
    System.out.println(p);
}

sortArray(products);

System.out.println("\nAfter sorting (Products by price): ");
for (Product p : products) {
    System.out.println(p);
} } }

class Product implements Comparable<Product> {
    private String name;
    private int price;
    public Product(String name, int price) {
        this.name = name;
        this.price = price;
    }
}
```

```
}  
  
@Override  
public int compareTo(Product other) {  
    return this.price - other.price;  
}  
  
@Override  
public String toString() {  
    return name + ": $" + price;  
} }
```

### **OUTPUT:**

```
Before sorting (Integers): [5, 3, 9, 1, 7]  
After sorting (Integers): [1, 3, 5, 7, 9]  
  
Before sorting (Strings): [John, Alice, Bob, David]  
After sorting (Strings): [Alice, Bob, David, John]  
  
Before sorting (Products by price):  
Laptop: $1000  
Phone: $800  
Tablet: $600  
Smartwatch: $200  
  
After sorting (Products by price):  
Smartwatch: $200  
Tablet: $600  
Phone: $800  
Laptop: $1000
```

### **CONCLUSION:**

This program demonstrates the use of generics in Java to create a versatile sorting method for arrays of different types. By implementing the Comparable interface in the Product class, it enables sorting of custom objects based on specific criteria, such as price. The output shows the effective sorting of integers, strings, and products, highlighting the flexibility and reusability of the generic sorting method.

40

Write a program that counts the occurrences of words in a text and displays the words and their occurrences in alphabetical order of the words. Using Map and Set Classes.

**PROGRAM CODE:**

```
import java.util.*;

public class P40 {

    public static void main(String[] args) {

        Map<String, Integer> wordMap = new TreeMap<>();

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter a text:");

        String text = scanner.nextLine();

        String[] words = text.toLowerCase().split("\\W+");

        for (String word : words) {

            if (!word.isEmpty()) {

                wordMap.put(word, wordMap.getOrDefault(word, 0) + 1);

            }

        }

        System.out.println("\nWord Occurrences (in alphabetical order):");

        Set<Map.Entry<String, Integer>> entrySet = wordMap.entrySet();

        for (Map.Entry<String, Integer> entry : entrySet) {

            System.out.println(entry.getKey() + ": " + entry.getValue());

        }

    }

}
```

**OUTPUT:**

```

Enter a text:
This is java Program.

Word Occurrences (in alphabetical order):
is: 1
java: 1
program: 1
this: 1

```

### **CONCLUSION:**

This program demonstrates how to count and display the occurrences of words in a given text using Java's Map and Set classes. The words are stored in a TreeMap, ensuring that they are presented in alphabetical order. The use of `getOrDefault()` simplifies the counting process, showcasing efficient word frequency analysis.

41

Write a code which counts the number of the keywords in a Java source file. Store all the keywords in a HashSet and use the `contains()` method to test if a word is in the keyword set.

### **PROGRAM CODE:**

```

import java.io.*;

import java.util.*;

public class P41 {

private static final HashSet<String> keywords = new HashSet<>();

static {

String[] keywordArray = {

"abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class",

"const", "continue", "default", "do", "double", "else", "enum", "extends", "final",

"finally", "float", "for", "goto", "if", "implements", "import", "instanceof", "int",

"interface", "long", "native", "new", "package", "private", "protected", "public",

"return", "short", "static", "strictfp", "super", "switch", "synchronized", "this",

"throw", "throws", "transient", "try", "void", "volatile", "while"

```

```
};  
for (String keyword : keywordArray) {  
    keywords.add(keyword);  
} }  
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    System.out.print("Enter the path of the Java source file: ");  
    String filePath = scanner.nextLine();  
    try {  
        File file = new File(filePath);  
        Scanner fileScanner = new Scanner(file);  
        int keywordCount = 0;  
        while (fileScanner.hasNext()) {  
            String word = fileScanner.next();  
            if (keywords.contains(word)) {  
                keywordCount++;  
            }  
        }  
        System.out.println("Number of Java keywords in the file: " + keywordCount);  
        fileScanner.close();  
    } catch (FileNotFoundException e) {  
        System.out.println("File not found: " + filePath);  
    } } }
```

**OUTPUT:**

```
Enter the path of the Java source file: P40.java  
Number of Java keywords in the file: 11
```

**CONCLUSION:**

This program demonstrates the use of a HashSet to efficiently count Java keywords in a source file. By reading each word from the file and checking for its presence in the set of keywords, it showcases how to utilize collections for rapid lookups. The result is the total number of keywords, providing a simple yet effective tool for analyzing Java code.