

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

Department of Computer Science &amp; Engineering

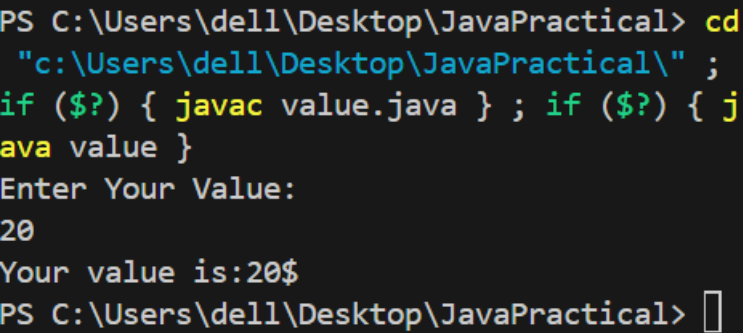
**Subject Name: Dharohar Chhaya****Semester: 3<sup>rd</sup>****Subject Code: CSE-201****Academic year: 2024-2025****Part - 1**

<b>No .</b>	<b>Aim of the Practical</b>
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>import java.util.Scanner;  public class value {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          System.out.println("Enter Your Value:");          int n = scanner.nextInt();</pre>

```
System.out.println("Your value is:"+n+"$");
```

```
}}
```

### **OUTPUT:**



```
PS C:\Users\dell\Desktop\JavaPractical> cd
"c:\Users\dell\Desktop\JavaPractical\" ;
if ($?) { javac value.java } ; if ($?) { j
ava value }
Enter Your Value:
20
Your value is:20$
PS C:\Users\dell\Desktop\JavaPractical> █
```

### **CONCLUSION:**

In this program, we created a simple banking application that stores the user's current balance in a variable. The balance is then displayed to the user using `System.out.println()`. This basic structure can be expanded to include more complex functionalities, such as deposits, withdrawals, and account management, making it a foundational step in developing a full-fledged banking application.

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 main{

public static void main(String args[]){

float distance;

int hr,min,sec;

System.out.println("Enter The Distance");

Scanner s = new Scanner(System.in);

distance =s.nextFloat();

System.out.println("Enter The time");

hr =s.nextInt();

min = s.nextInt();

sec = s.nextInt();

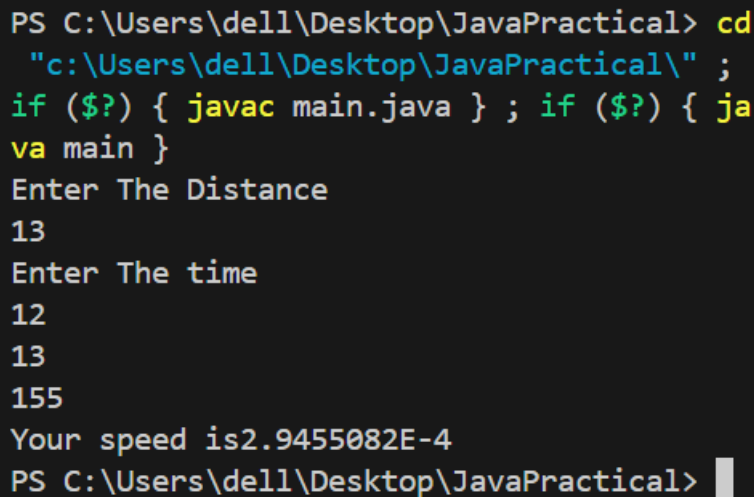
float time = (hr*3600)+(min*60)+(sec);
```

```
float speed = distance/time;

System.out.println("Your speed is"+speed);

}}
```

### OUTPUT:



```
PS C:\Users\dell\Desktop\JavaPractical> cd
"c:\Users\dell\Desktop\JavaPractical\" ;
if ($?) { javac main.java } ; if ($?) { ja
va main }
Enter The Distance
13
Enter The time
12
13
155
Your speed is2.9455082E-4
PS C:\Users\dell\Desktop\JavaPractical>
```

### CONCLUSION:

In this program, we prompt the user to input the distance traveled in meters and the time taken in hours, minutes, and seconds. The program then calculates 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 basic arithmetic operations, and display results in a user-friendly manner. Such a program can be a foundational component in applications requiring speed calculations, such as fitness trackers or transportation management systems.

4. 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.

**PROGRAM CODE :**

```
import java.util.Scanner;

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

        System.out.print("Enter the number of days: ");
        int numDays = scanner.nextInt();

        double[] expenses = new double[numDays];
        double totalExpense = 0.0;

        for (int i = 0; i < numDays; i++) {
            System.out.print("Enter expense for day " + (i + 1) + ": ");
            expenses[i] = scanner.nextDouble();
            totalExpense += expenses[i];
        }

        scanner.close();

        System.out.println("Total expenses for the month: $" + totalExpense);
    }
}
```

**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac exp.java } ; if ($?) { java exp }
Enter the number of days: 12
Enter expense for day 1: 1000
Enter expense for day 2: 200
Enter expense for day 3: 300
Enter expense for day 4: 323
Enter expense for day 5: 1221
Enter expense for day 6: 343
Enter expense for day 7: 32
Enter expense for day 8: 232
Enter expense for day 9: 232
Enter expense for day 10: 2323
Enter expense for day 11: 21
Enter expense for day 12: 132
Total expenses for the month: $6359.0
PS C:\Users\dell\Desktop\JavaPractical> 3
3
```

**CONCLUSION:**

In this program, we prompt the user to input the number of days in the month and their daily expenses. The program then calculates the total expenses by summing up the daily expenses stored in an array. This example demonstrates how to handle user input, use arrays to store data, and perform basic arithmetic operations to compute the total. Such a program can be a foundational component in a budget tracking application, helping users manage their finances effectively.

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 bill {
```

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

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

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

```
        double[] prices = {0.0, 0.0, 0.0, 0.0};
```

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

```
            System.out.print("Enter price for product code " + productCodes[i] + ": ");
```

```
            prices[i] = scanner.nextDouble();
```

```
        }
```

```
        double totalBill = 0.0;
```

```
        System.out.print("Enter the product code (1 to 4, or any other code to exit): ");
```

```
        int code = scanner.nextInt();
```

```
        while (code >= 1 && code <= 4) {
```

```
            switch (code) {
```

```
                case 1:
```

```
                    totalBill += prices[0] * 1.08;
```

```
                    break;
```

```
                case 2:
```

```
                    totalBill += prices[1] * 1.12;
```

```
        break;
    case 3:
        totalBill += prices[2] * 1.05;
        break;
    case 4:
        totalBill += prices[3] * 1.075;
        break;
    default:
        System.out.println("Invalid product code. Exiting...");
        break;
    }

    System.out.print("Enter the next product code (1 to 4, or any other code to exit): ");
    code = scanner.nextInt();
}

scanner.close();

System.out.println("Total bill: $" + totalBill);
}
}
```



**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac bill.java } ; if ($?) { java bill }
Enter price for product code 1: 120
Enter price for product code 2: 200
Enter price for product code 3: 400
Enter price for product code 4: 600
Enter the product code (1 to 4, or any other code to exit): 2
Enter the next product code (1 to 4, or any other code to exit): 3
Enter the next product code (1 to 4, or any other code to exit): 4
Enter the next product code (1 to 4, or any other code to exit): 5
Total bill: $1289.0
PS C:\Users\dell\Desktop\JavaPractical> █
```

**CONCLUSION:**

In this program, we prompt the user to input the number of items, their product codes, and prices. The program then calculates the total bill by applying the appropriate sales tax based on the product code using a switch statement. This example demonstrates how to handle user input, use arrays to store data, and perform conditional operations to compute the total bill. Such a program can be a foundational component in a point-of-sale system for an electric appliance shop.

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

        System.out.print("Enter the number of days for your exercise routine: ");
        int n = scanner.nextInt();

        System.out.println("Exercise duration for each day (in minutes):");
        for (int i = 0; i < n; i++) {
            System.out.println("Day " + (i + 1) + ": " + fibonacci(i) + " minutes");
        }

        scanner.close();
    }

    public static int fibonacci(int n) {
        if (n <= 1) {
            return n;
        }
        int a = 0, b = 1, c;
        for (int i = 2; i <= n; i++) {
            c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
}
```

**OUTPUT:**

```

PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; i
f ($?) { javac fibo.java } ; if ($?) { java fibo }
Enter the number of days for your exercise routine: 6
Exercise duration for each day (in minutes):
Day 1: 0 minutes
Day 2: 1 minutes
Day 3: 1 minutes
Day 4: 2 minutes
Day 5: 3 minutes
Day 6: 5 minutes
PS C:\Users\dell\Desktop\JavaPractical>

```

### CONCLUSION:

In this program, we prompt the user to enter the number of days for their exercise routine. The program then calculates and displays the first n terms of the Fibonacci series, representing the exercise duration for each day. This example demonstrates how to handle user input, use loops, and implement the Fibonacci sequence in Java. Such a program can be a fun and motivational way to plan an exercise routine, gradually increasing the duration based on the Fibonacci series

### PART-II Strings

7. 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;
   
front\_times('Chocolate', 2) → 'ChoCho'
   
front\_times('Chocolate', 3) → 'ChoChoCho'
   
front\_times('Abc', 3) → 'AbcAbcAbc'

**PROGRAM CODE :**

```
import java.util.Scanner;

public class string {

    public static void main(String[] args) {

        // Create a Scanner object for taking input
        from user

        Scanner scanner = new
        Scanner(System.in);

        // Prompt the user to enter a string

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

        String str = scanner.nextLine();

        // Prompt the user to enter a non-negative
        integer

        System.out.println("Enter a non-negative
        integer:");

        int n = scanner.nextInt();

        // Get the front part of the string (first 3
```

characters or whatever is available)

```
String front;
```

```
if (str.length() < 3) {
```

```
    front = str;
```

```
} else {
```

```
    front = str.substring(0, 3);
```

```
}
```

```
// Create a new string with n copies of the
```

front part

```
String result = "";
```

```
for (int i = 0; i < n; i++) {
```

```
    result += front;
```

```
}
```

```
// Output the result
```

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

```
}
```

```
}
```

**OUTPUT:**

```
1\Desktop\Set-2\" ; if ($?) { javac prac_7.java }  
; if ($?) { java prac_7 }  
Enter a string:  
purvik  
Enter a non-negative integer:  
4  
purpurpurpur  
PS C:\Users\dell\Desktop\Set-2>
```

### **CONCLUSION:**

The front\_times function takes a string and an integer n, extracts the front (first 3 characters), and returns a new string with n copies of that front. It's a simple and concise way to repeat the initial portion of a string multiple times.

8.

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

### **PROGRAM CODE :**

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

```
public class prac_8{  
    public static void main(String[] args) {  
        // Create a Scanner object for taking input  
from user  
        Scanner scanner = new  
Scanner(System.in);  
  
        // Prompt the user to enter the size of the  
array  
        System.out.println("Enter the size of the  
array:");  
        int size = scanner.nextInt();  
  
        // Create an array to store the integers  
        int[] nums = new int[size];  
  
        // Prompt the user to enter the elements of  
the array  
        System.out.println("Enter the elements of  
the array:");  
        for (int i = 0; i < size; i++) {
```

```
        nums[i] = scanner.nextInt();

    }

    // Initialize a counter for the number of 9's

    int count = 0;

    // Iterate through the array and count the
number of 9's

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

        if (nums[i] == 9) {

            count++;

        }

    }

    // Output the result

    System.out.println("Number of 9's in the
array: " + count);

}

}
```

**OUTPUT:**



```
PS C:\Users\dell\Desktop\Set-2> cd "c:\Users\dell\Desktop\Set-2\" ; if ($?) { javac prac_8.java } ;  
if ($?) { java prac_8 }  
Enter the size of the array:  
4  
Enter the elements of the array:  
1  
2  
9  
9  
Number of 9's in the array: 2  
PS C:\Users\dell\Desktop\Set-2> █
```

### **CONCLUSION:**

The array\_count9 function takes an array of integers and returns the count of occurrences of the digit 9. It's a straightforward task that involves iterating through the array and checking each element.

9.

Given a string, return a string where for every char in the original, there are two chars.

double\_char('The') → 'TThhee'

double\_char('AAbb') → 'AAAAbbbb'

double\_char('Hi-There') → 'HHii--TThheerree'

### **PROGRAM CODE :**

```
import java.util.Scanner;

public class DoubleChar {

    public static void main(String[] args) {

        // Create a Scanner object for taking input
        from user

        Scanner scanner = new
        Scanner(System.in);

        // Prompt the user to enter a string

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

        String str = scanner.nextLine();

        // Create a new string to store the result

        String result = "";

        // Iterate through each character in the
        original string

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

            // Append each character twice to the
            result string

            result += str.charAt(i);
```

```
        result += str.charAt(i);

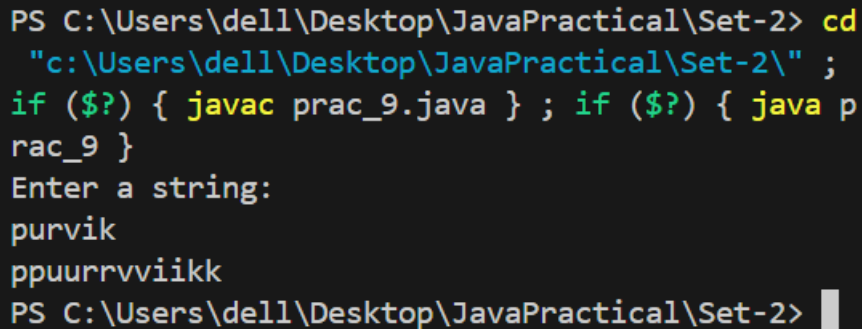
    }

    // Output the result

    System.out.println(result);

}

}
```

**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical\Set-2> cd
"c:\Users\dell\Desktop\JavaPractical\Set-2\" ;
if ($?) { javac prac_9.java } ; if ($?) { java p
rac_9 }
Enter a string:
purvik
ppuurrvviikk
PS C:\Users\dell\Desktop\JavaPractical\Set-2> █
```

10.

**CONCLUSION:**

The double\_char function takes a string and returns a new string where each character from the original string is repeated twice. It's a straightforward task that involves iterating through the characters and appending them to the output string.

Perform following functionalities of the string:

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

**PROGRAM CODE :**

```
import java.util.Scanner;

public class prac_10 {

    public static void main(String[] args) {

        Scanner scanner = new
Scanner(System.in);

        System.out.print("Enter a string: ");

        String inputString = scanner.nextLine();

        int length = inputString.length();

        System.out.println("Length of the string: "
+ length);

        String lowerCaseString =
inputString.toLowerCase();

        System.out.println("Lowercase of the
string: " + lowerCaseString);

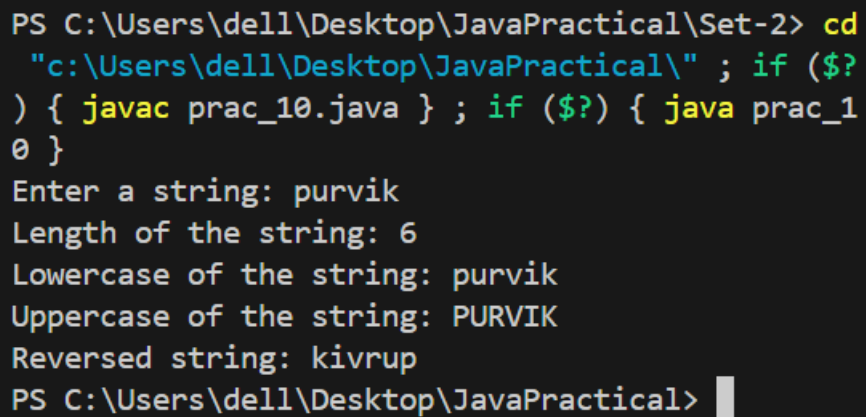
        String upperCaseString =
inputString.toUpperCase();

        System.out.println("Uppercase of the
string: " + upperCaseString);

        String reversedString = new
StringBuilder(inputString).reverse().toString();

        System.out.println("Reversed string: " +
```

```
reversedString);  
  
scanner.close();  
  
}  
  
}
```

**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical\Set-2> cd  
"c:\Users\dell\Desktop\JavaPractical\" ; if ($?)  
{ javac prac_10.java } ; if ($?) { java prac_1  
0 }  
Enter a string: purvik  
Length of the string: 6  
Lowercase of the string: purvik  
Uppercase of the string: PURVIK  
Reversed string: kivrup  
PS C:\Users\dell\Desktop\JavaPractical>
```

11.

**CONCLUSION:**

The code implementing these functionalities would involve using the appropriate methods for each task. It's a straightforward process, and you can combine these functionalities as needed for your specific use case.

Perform following Functionalities of the string:

“CHARUSAT UNIVERSITY”

- Find length
- Replace 'H' by 'FIRST LATTER OF YOUR NAME'
- Convert all character in lowercase

**PROGRAM CODE :**

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

```
public class prac_11 {  
  
    public static void main(String[] args) {  
  
        // The given string  
  
        String str = "CHARUSAT UNIVERSITY";  
  
  
        // Create a Scanner object for taking input  
from the user  
  
        Scanner scanner = new  
Scanner(System.in);  
  
  
        // Find the length of the string  
  
        int length = str.length();  
  
        System.out.println("Length of the string: "  
+ length);  
  
  
        // Prompt the user to enter their first name  
  
        System.out.println("Enter your first  
name:");  
  
        String firstName = scanner.nextLine();  
  
        char firstLetter = firstName.charAt(0);  
  
  
        // Replace 'H' with the first letter of the
```

```
user's name

    String replacedString = str.replace('H',
firstLetter);

    System.out.println("String after replacing
'H' with '" + firstLetter + "': " + replacedString);

    // Convert all characters in the string to
lowercase

    String lowercaseString =
str.toLowerCase();

    System.out.println("String in lowercase: " +
lowercaseString);

}

}
```

**OUTPUT:**

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\Java  
Practical\Set-2\" ; if ($?) { javac prac_11.java  
} ; if ($?) { java prac_11 }  
Length of the string: 19  
Enter your first name:  
purvik  
String after replacing 'H' with 'p': CpARUSAT UN  
IVERSITY  
String in lowercase: charusat university  
PS C:\Users\dell\Desktop\JavaPractical\Set-2> █
```

### **CONCLUSION:**

The code implementing these functionalities would involve using the appropriate methods for each task. It's a straightforward process, and you can adapt it to any given string.



12. 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.

**PROGRAM CODE :**

```
import java.util.Scanner;

public class Prac_12 {

    public static void main(String[] args) {

        final double conversionRate = 100.0;

        double pounds;

        if (args.length > 0) {

            // Try to parse the first command-line
            argument as a double

            try {

                pounds =
                Double.parseDouble(args[0]);

            } catch (NumberFormatException e) {

                System.out.println("Invalid command-
                line argument. Please enter a valid number.");

                return;

            }

        }
```

```
    } else {  
  
        // If no command-line arguments, read  
input interactively from the user  
  
        Scanner scanner = new  
Scanner(System.in);  
  
        System.out.print("Enter the amount in  
Pounds: ");  
  
        pounds = scanner.nextDouble();  
    }  
  
    // Convert Pounds to Rupees  
  
    double rupees = pounds * conversionRate;  
  
    // Print the result  
  
    System.out.printf("%.2f Pounds is  
equivalent to %.2f Rupees%n", pounds, rupees);  
    }  
}
```

**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical\Set-2> cd
"c:\Users\dell\Desktop\JavaPractical\set-3\" ;
if ($?) { javac Prac_12.java } ; if ($?) { java
Prac_12 }
Enter the amount in Pounds: 140
140.00 Pounds is equivalent to 14000.00 Rupees
PS C:\Users\dell\Desktop\JavaPractical\set-3> |
```

**CONCLUSION:**

In conclusion, a currency conversion tool for a travel agency that converts amounts from Pounds to Rupees using a fixed conversion rate of 1 Pound = 100 Rupees can be an efficient and user-friendly application. By supporting both command-line arguments and interactive input, it can accommodate different user preferences and scenarios, ensuring versatility and accessibility. This approach can aid travelers in quickly and accurately converting their money, enhancing their travel experience and ensuring they have the correct amount of local currency for their needs..

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 CODE :**

```
import java.util.Scanner;

public class prac_13 {
    // Instance variables
    private String firstName;
    private String lastName;
    private double monthlySalary;

    // Constructor
    public prac_13(String firstName, String
lastName, double monthlySalary) {
        this.firstName = firstName;
        this.lastName = lastName;
        // Set the monthly salary, ensuring it's not
negative
        this.monthlySalary = (monthlySalary > 0) ?
monthlySalary : 0.0;
    }

    // Getter and Setter for firstName
    public String getFirstName() {
        return firstName;
    }
}
```

```
}

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

// Getter and Setter for lastName
public String getLastName() {
    return lastName;
}

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

// Getter and Setter for monthlySalary
public double getMonthlySalary() {
    return monthlySalary;
}

public void setMonthlySalary(double
monthlySalary) {
```

```
        this.monthlySalary = (monthlySalary > 0) ?  
monthlySalary : 0.0;  
    }  
  
    // Method to calculate yearly salary  
    public double getYearlySalary() {  
        return monthlySalary * 12;  
    }  
  
    // Method to apply a raise  
    public void applyRaise(double percentage) {  
        if (percentage > 0) {  
            monthlySalary += monthlySalary *  
(percentage / 100);  
        }  
    }  
  
    // Test application  
    public static void main(String[] args) {  
        Scanner scanner = new  
Scanner(System.in);
```

```
// Input for first Employee

System.out.print("Enter first name for
Employee 1: ");

String firstName1 = scanner.nextLine();

System.out.print("Enter last name for
Employee 1: ");

String lastName1 = scanner.nextLine();

System.out.print("Enter monthly salary for
Employee 1: ");

double salary1 = scanner.nextDouble();
scanner.nextLine(); // Consume newline

// Create the first Employee object

Employee employee1 = new
Employee(firstName1, lastName1, salary1);

// Input for second Employee

System.out.print("Enter first name for
Employee 2: ");

String firstName2 = scanner.nextLine();

System.out.print("Enter last name for
Employee 2: ");
```

```
String lastName2 = scanner.nextLine();

System.out.print("Enter monthly salary for
Employee 2: ");

double salary2 = scanner.nextDouble();

// Create the second Employee object
Employee employee2 = new
Employee(firstName2, lastName2, salary2);

// Display initial yearly salaries
System.out.printf("%s %s's yearly salary:
%.2f%n", employee1.getFirstName(),
employee1.getLastName(),
employee1.getYearlySalary());

System.out.printf("%s %s's yearly salary:
%.2f%n", employee2.getFirstName(),
employee2.getLastName(),
employee2.getYearlySalary());

// Apply a 10% raise
employee1.applyRaise(10);
employee2.applyRaise(10);
```



```
// Display updated yearly salaries

System.out.printf("%s %s's yearly salary
after 10%% raise: %.2f%n",
employee1.getFirstName(),
employee1.getLastName(),
employee1.getYearlySalary());

System.out.printf("%s %s's yearly salary
after 10%% raise: %.2f%n",
employee2.getFirstName(),
employee2.getLastName(),
employee2.getYearlySalary());

// Close the scanner

scanner.close();

}

}
```

**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical\set-3> cd "
"c:\Users\dell\Desktop\JavaPractical\set-3\" ; if
($?) { javac prac_13.java } ; if ($?) { java prac
_13 }
Enter first name for Employee 1: purvik
Enter last name for Employee 1: anghan
Enter monthly salary for Employee 1: 1000000
Enter first name for Employee 2: om
Enter last name for Employee 2: barvaliya
Enter monthly salary for Employee 2: 1000000
purvik anghan's yearly salary: 12000000.00
om barvaliya's yearly salary: 12000000.00
purvik anghan's yearly salary after 10% raise: 132
00000.00
om barvaliya's yearly salary after 10% raise: 1320
0000.00
PS C:\Users\dell\Desktop\JavaPractical\set-3> █
```

14.

**CONCLUSION:**

In conclusion, the `Employee` class is designed to encapsulate an employee's first name, last name, and monthly salary, while ensuring that the salary is set to a non-negative value. By using getter and setter methods, the class maintains control over its instance variables. The `EmployeeTest` application demonstrates the functionality of the `Employee` class by creating two Employee objects, displaying their yearly salaries, applying a 10% raise, and then displaying the updated yearly salaries. This confirms the class's ability to manage salary adjustments correctly and its practical utility in handling employee data.

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 CODE :**

```
import java.util.Scanner;

public class Date {

    // Instance variables

    private int month;

    private int day;

    private int year;

    // Constructor

    public Date(int month, int day, int year) {

        this.month = month;

        this.day = day;

        this.year = year;

    }

    // Getter and Setter for month

    public int getMonth() {

        return month;

    }

    public void setMonth(int month) {
```

```
        this.month = month;
    }

    // Getter and Setter for day
    public int getDay() {
        return day;
    }

    public void setDay(int day) {
        this.day = day;
    }

    // Getter and Setter for year
    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // Method to display the date
```

```
public void displayDate() {  
    System.out.printf("%02d/%02d/%04d%n",  
month, day, year);  
}  
  
// Test application  
public static void main(String[] args) {  
    Scanner scanner = new  
Scanner(System.in);  
  
    // Input for the first date  
    System.out.print("Enter month for Date 1:  
");  
    int month1 = scanner.nextInt();  
    System.out.print("Enter day for Date 1: ");  
    int day1 = scanner.nextInt();  
    System.out.print("Enter year for Date 1: ");  
    int year1 = scanner.nextInt();  
  
    // Create the first Date object  
    Date date1 = new Date(month1, day1,  
year1);
```

```
// Input for the second date

System.out.print("Enter month for Date 2:
");

int month2 = scanner.nextInt();

System.out.print("Enter day for Date 2: ");

int day2 = scanner.nextInt();

System.out.print("Enter year for Date 2: ");

int year2 = scanner.nextInt();


// Create the second Date object

Date date2 = new Date(month2, day2,
year2);


// Display the dates

System.out.print("Date 1: ");

date1.displayDate();

System.out.print("Date 2: ");

date2.displayDate();


// Close the scanner

scanner.close();
```

}

}

**OUTPUT:**

```

PS C:\Users\dell\Desktop\JavaPractical\set-3> cd "
c:\Users\dell\Desktop\JavaPractical\set-3\" ; if (
$?) { javac Date.java } ; if ($?) { java Date }
Enter month for Date 1: 10
Enter day for Date 1: 21
Enter year for Date 1: 2024
Enter month for Date 2: 2
Enter day for Date 2: 13
Enter year for Date 2: 2025
Date 1: 10/21/2024
Date 2: 02/13/2025
PS C:\Users\dell\Desktop\JavaPractical\set-3>

```

15.

**CONCLUSION:**

The provided code outlines the creation of a `Date` class that encapsulates three pieces of date information: month, day, and year. The class includes a constructor to initialize these variables, assuming the input values are valid. It also includes getter and setter methods for each instance variable, ensuring encapsulation and flexibility in accessing and modifying the date information. Additionally, the class features a `displayDate` method, which formats the date as "month/day/year" using forward slashes. A companion test application, `DateTest`, demonstrates the practical functionality and capabilities of the `Date` class.

This structured approach ensures clear, maintainable, and easily testable code for handling

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 CODE :**

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

```
public class prac_15 {  
    // Instance variables  
  
    private double length;  
  
    private double breadth;  
  
    // Constructor  
  
    public prac_15(double length, double breadth)  
    {  
        this.length = length;  
        this.breadth = breadth;  
    }  
  
    // Method to calculate and return the area of  
the rectangle  
  
    public double returnArea() {  
        return length * breadth;  
    }  
  
    // Main method  
  
    public static void main(String[] args) {  
        Scanner scanner = new
```



```
Scanner(System.in);

// Input for length

System.out.print("Enter the length of the
rectangle: ");

double length = scanner.nextDouble();

// Input for breadth

System.out.print("Enter the breadth of the
rectangle: ");

double breadth = scanner.nextDouble();

// Create an Area object

Area rectangle = new Area(length,
breadth);

// Calculate and display the area

System.out.printf("The area of the rectangle
is: %.2f%n", rectangle.returnArea());

// Close the scanner

scanner.close();
```

}

}

**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical\set-3> cd "
c:\Users\dell\Desktop\JavaPractical\set-3\" ; if (
$?) { javac prac_15.java } ; if ($?) { java prac_1
5 }
Enter the length of the rectangle: 13
Enter the breadth of the rectangle: 45
The area of the rectangle is: 585.00
PS C:\Users\dell\Desktop\JavaPractical\set-3>
```

16.

**CONCLUSION:**

It defines a class called Area with a constructor that takes the length and breadth as parameters. The returnArea method calculates the area by multiplying the length and breadth. The user inputs the length and breadth values. An instance of the Area class is created with the provided values. Finally, the area is calculated and displayed.

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.

**PROGRAM CODE :**

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

```
public class Complex {  
    // Instance variables  
  
    private double real;  
  
    private double imaginary;  
  
    // Constructor  
  
    public Complex(double real, double  
imaginary) {  
        this.real = real;  
        this.imaginary = imaginary;  
    }  
  
    // Method to add two complex numbers  
  
    public Complex add(Complex other) {  
        double realPart = this.real + other.real;  
        double imaginaryPart = this.imaginary +  
other.imaginary;  
        return new Complex(realPart,  
imaginaryPart);  
    }  
}
```

```
// Method to subtract two complex numbers

public Complex subtract(Complex other) {

    double realPart = this.real - other.real;

    double imaginaryPart = this.imaginary -
other.imaginary;

    return new Complex(realPart,
imaginaryPart);

}

// Method to multiply two complex numbers

public Complex multiply(Complex other) {

    double realPart = this.real * other.real -
this.imaginary * other.imaginary;

    double imaginaryPart = this.real *
other.imaginary + this.imaginary * other.real;

    return new Complex(realPart,
imaginaryPart);

}

// Method to display the complex number

public void display() {

    System.out.printf("%.2f + %.2fi%n", real,
```

```
imaginary);  
  
    }  
  
    // Main method  
  
    public static void main(String[] args) {  
  
        Scanner scanner = new  
Scanner(System.in);  
  
        // Input for the first complex number  
  
        System.out.print("Enter real part of the first  
complex number: ");  
  
        double real1 = scanner.nextDouble();  
  
        System.out.print("Enter imaginary part of  
the first complex number: ");  
  
        double imaginary1 = scanner.nextDouble();  
  
        // Create the first complex number  
  
        Complex complex1 = new Complex(real1,  
imaginary1);  
  
        // Input for the second complex number  
  
        System.out.print("Enter real part of the
```

```
second complex number: ");

    double real2 = scanner.nextDouble();

    System.out.print("Enter imaginary part of
the second complex number: ");

    double imaginary2 = scanner.nextDouble();

    // Create the second complex number

    Complex complex2 = new Complex(real2,
imaginary2);

    // Perform operations

    Complex sum = complex1.add(complex2);

    Complex difference =
complex1.subtract(complex2);

    Complex product =
complex1.multiply(complex2);

    // Display results

    System.out.print("Sum: ");

    sum.display();

    System.out.print("Difference: ");

    difference.display();
```

```
System.out.print("Product: ");  
  
product.display();  
  
// Close the scanner  
scanner.close();  
}  
}
```

**OUTPUT:**

17.

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\set-3\"
; if ($?) { javac Complex.java }
; if ($?) { java Complex }
Enter real part of the first complex number: 2
Enter imaginary part of the first complex number: 3
Enter real part of the second complex number: 4
Enter imaginary part of the second complex number: 5
Sum: 6.00 + 8.00i
Difference: -2.00 + -2.00i
Product: -7.00 + 22.00i
PS C:\Users\dell\Desktop\JavaPractical\set-3>
```

### CONCLUSION:

We define a Complex class with an initializer that takes the real and imaginary parts. The add, subtract, and multiply methods perform the corresponding operations. The user inputs the real and imaginary parts of two complex numbers. We create instances of the Complex class and compute the results.

### PART-IV Inheritance, Interface, Package

**Aim:** 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 for each of the class and call 1 - method of parent class by object of parent.



**PROGRAM CODE :**

// Parent class

```
class Parent {  
    void printParent() {  
        System.out.println("This is parent class");  
    }  
}
```

// Subclass

```
class Child extends Parent {  
    void printChild() {  
        System.out.println("This is child class");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        // Creating an object of the parent class  
        Parent parentObject = new Parent();  
        // Calling the method of the parent class
```

18.

```
parentObject.printParent();

// Creating an object of the child class
Child childObject = new Child();

// Calling the method of the child class
childObject.printChild();

}

}
```

**OUTPUT:**

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_17.java } ; if ($?) { java prac_17 }

This is parent class
This is child class
PS C:\Users\dell\Desktop\JavaPractical> █
```

**CONCLUSION:**

The Parent class has a method printParent() that prints “This is parent class”. The Child class extends Parent and has an additional method printChild() that prints “This is child class”.

In the Main class, we create objects for both Parent and Child classes and call their respective methods.

Create a class named 'Member' having the following members: Data members

- 1 - Name
- 2 - Age
- 3 - Phone number

4 - Address

5 – Salary

It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherits 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.

### **PROGRAM CODE :**

```
import java.util.Scanner;

// Base class
class Member {
    String name;
    int age;
    String phoneNumber;
    String address;
    double salary;

    void printSalary() {
        System.out.println("Salary: " + salary);
    }
}
```

```
// Derived class Employee

class Employee extends Member {

    String specialization;

}


// Derived class Manager

class Manager extends Member {

    String department;

}


public class Main {

    public static void main(String[] args) {

        Scanner scanner = new
Scanner(System.in);


        // Creating an object of Employee class

        Employee employee = new Employee();

        System.out.println("Enter Employee
Details:");

        System.out.print("Name: ");

        employee.name = scanner.nextLine();

        System.out.print("Age: ");
```

```
employee.age = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Phone Number: ");

employee.phoneNumber =

scanner.nextLine();

System.out.print("Address: ");

employee.address = scanner.nextLine();

System.out.print("Salary: ");

employee.salary = scanner.nextDouble();

scanner.nextLine(); // Consume newline

System.out.print("Specialization: ");

employee.specialization =

scanner.nextLine();


// Creating an object of Manager class

Manager manager = new Manager();

System.out.println("\nEnter Manager

Details:");

System.out.print("Name: ");

manager.name = scanner.nextLine();

System.out.print("Age: ");

manager.age = scanner.nextInt();
```

```
scanner.nextLine(); // Consume newline

System.out.print("Phone Number: ");

manager.phoneNumber =

scanner.nextLine();

System.out.print("Address: ");

manager.address = scanner.nextLine();

System.out.print("Salary: ");

manager.salary = scanner.nextDouble();

scanner.nextLine(); // Consume newline

System.out.print("Department: ");

manager.department = scanner.nextLine();


// Printing details of Employee

System.out.println("\nEmployee Details:");

System.out.println("Name: " +
employee.name);

System.out.println("Age: " +
employee.age);

System.out.println("Phone Number: " +
employee.phoneNumber);

System.out.println("Address: " +
employee.address);
```

```
        employee.printSalary();

        System.out.println("Specialization: " +
employee.specialization);

        // Printing details of Manager

        System.out.println("\nManager Details:");

        System.out.println("Name: " +
manager.name);

        System.out.println("Age: " + manager.age);

        System.out.println("Phone Number: " +
manager.phoneNumber);

        System.out.println("Address: " +
manager.address);

        manager.printSalary();

        System.out.println("Department: " +
manager.department);

        scanner.close();
    }
}
```

**OUTPUT:**

19.

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_17.java } ; if ($?) { java prac_17 }

This is parent class
This is child class
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_18.java } ; if ($?) { java prac_18 }
Enter Employee Details:
Name: purvik
Age: 19
Phone Number: 3525422243
Address: home
Salary: 100000000
Specialization: cs

Enter Manager Details:
Name: om
Age: 18
Phone Number: 42342424
Address: surat
Salary: 100000000
Department: cs
```

### **CONCLUSION:**

The Member class has data members for name, age, phone number, address, and salary, along with a method printSalary(). The Employee class extends Member and adds a specialization data member. The Manager class extends Member and adds a department data member. The Main class uses a Scanner to take input from the user and assigns values to the data members of Employee and Manager objects. Finally, it prints the details of both the Employee and Manager.

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 CODE :**

```
import java.util.Scanner;

// Base class
class Rectangle {
    double length;
    double breadth;

    // Constructor to initialize length and breadth
    Rectangle(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    // Method to print the area of the rectangle
    void printArea() {
        double area = length * breadth;
        System.out.println("Area: " + area);
    }
}
```

```
// Method to print the perimeter of the
rectangle

void printPerimeter() {

    double perimeter = 2 * (length + breadth);

    System.out.println("Perimeter: " +
perimeter);

}

}

// Derived class

class Square extends Rectangle {

    // Constructor to initialize side of the square

    Square(double side) {

        super(side, side);

    }

}

public class Main {

    public static void main(String[] args) {

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

```
// Taking input for Rectangle

System.out.println("Enter the length and
breadth of the rectangle:");

double length = scanner.nextDouble();

double breadth = scanner.nextDouble();

Rectangle rectangle = new
Rectangle(length, breadth);


// Taking input for Square

System.out.println("Enter the side of the
square:");

double side = scanner.nextDouble();

Square square = new Square(side);


// Creating an array of objects

Rectangle[] shapes = {rectangle, square};


// Printing area and perimeter of each shape
for (Rectangle shape : shapes) {

    if (shape instanceof Square) {

        System.out.println("\nSquare:");

    } else {
```

20.

```
        System.out.println("\nRectangle:");

    }

    shape.printArea();

    shape.printPerimeter();

}

    scanner.close();

}

}
```

**OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_19.java } ; if ($?) { java prac_19 }
Enter the length and breadth of the rectangle:
12
19
Enter the side of the square:
1

Rectangle:
Area: 228.0
Perimeter: 62.0

Square:
Area: 1.0
Perimeter: 4.0
PS C:\Users\dell\Desktop\JavaPractical> █
```

**CONCLUSION:**

The Rectangle class has data members for length and breadth, and methods to print the area and perimeter. The Square class extends Rectangle and uses the super(s, s) constructor to

initialize the side. The Main class takes input from the user for both the rectangle and square, creates objects, and stores them in an array. It then prints the area and perimeter of each shape.

Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', '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 'Shape' and 'Rectangle' class by the object of 'Square' class.

### **PROGRAM CODE :**

```
import java.util.Scanner;

// Base class
class Shape {
    void printShape() {
        System.out.println("This is shape");
    }
}

// Derived class Rectangle
class Rectangle extends Shape {
    void printRectangle() {
        System.out.println("This is rectangular
```

```
shape");  
  
    }  
  
}  
  
// Derived class Circle  
class Circle extends Shape {  
  
    void printCircle() {  
  
        System.out.println("This is circular shape");  
  
    }  
  
}  
  
// Subclass Square of Rectangle  
class Square extends Rectangle {  
  
    void printSquare() {  
  
        System.out.println("Square is a rectangle");  
  
    }  
  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        Scanner scanner = new Scanner(System.in);
```

```
// Creating an object of Square class

Square square = new Square();

// Calling methods of Shape and Rectangle
class by the object of Square class

System.out.println("Calling methods using
Square object:");

square.printShape();

square.printRectangle();

square.printSquare();

scanner.close();

}

}
```

21. **OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_20.java } ; if ($?)  
{ java prac_20 }  
Calling methods using Square object:  
This is shape  
This is rectangular shape  
Square is a rectangle  
PS C:\Users\dell\Desktop\JavaPractical> |
```

### **CONCLUSION:**

The Shape class has a method printShape() that prints “This is shape”. The Rectangle class extends Shape and has a method printRectangle() that prints “This is rectangular shape”. The Circle class extends Shape and has a method printCircle() that prints “This is circular shape”. The Square class extends Rectangle and has a method printSquare() that prints “Square is a rectangle”. In the Main class, we create an object of the Square class and call the methods from Shape and Rectangle classes using this object.

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.

**PROGRAM CODE :**

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

```
// Base class
```

```
class Degree {
```

```
    void getDegree() {
```

```
        System.out.println("I got a degree");
```

```
    }
```

```
}
```

```
// Derived class Undergraduate
```

```
class Undergraduate extends Degree {
```

```
    @Override
```

```
    void getDegree() {
```

```
        System.out.println("I am an
```

```
Undergraduate");
```

```
    }
```

```
}
```

```
// Derived class Postgraduate
```

```
class Postgraduate extends Degree {
```

```
@Override

void getDegree() {

    System.out.println("I am a Postgraduate");

}

}

public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Creating an object of Degree class

        Degree degree = new Degree();

        // Creating an object of Undergraduate class

        Undergraduate undergraduate = new

Undergraduate();

        // Creating an object of Postgraduate class

        Postgraduate postgraduate = new

Postgraduate();

        // Calling the getDegree method for each

object

        System.out.println("Calling getDegree
```

```
method for Degree object:");

    degree.getDegree();

    System.out.println("\nCalling getDegree
method for Undergraduate object:");

    undergraduate.getDegree();

    System.out.println("\nCalling getDegree
method for Postgraduate object:");

    postgraduate.getDegree();

    scanner.close();

}

}
```

### **OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_21.java } ; if ($?) { java prac_21 }
Calling getDegree method for Degree object:
I got a degree

Calling getDegree method for Undergraduate object:
I am an Undergraduate

Calling getDegree method for Postgraduate object:
I am a Postgraduate
PS C:\Users\dell\Desktop\JavaPractical> █
```

### **CONCLUSION:**

The Degree class has a method getDegree() that prints “I godegree”The Undergraduate class extends Degree and overrides the getDegree() method to print “I am an

Undergraduate”. The Postgraduate class extends Degree and overrides the getDegree() method to print “I am a Postgraduate”. In the Main class, we create objects for each of the three classes and call their respective getDegree() methods.

Write a java that implements an interface AdvancedArithmetic which contains a method signature `int divisor_sum(int n)`. You need to write a class called `MyCalculator` which implements the interface. `divisorSum` 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 `divisor_sum` should return 12. The value of `n` will be at most 1000.

**PROGRAM CODE :**

```
import java.util.Scanner;

// Interface definition
interface AdvancedArithmetic {
    int divisor_sum(int n);
}

// Class implementing the interface
class MyCalculator implements
AdvancedArithmetic {
    @Override
    public int divisor_sum(int n) {
        int sum = 0;
```

```
        for (int i = 1; i <= n; i++) {  
            if (n % i == 0) {  
                sum += i;  
            }  
        }  
        return sum;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        // Taking input from the user  
        System.out.print("Enter an integer: ");  
        int n = scanner.nextInt();  
  
        // Creating an object of MyCalculator  
        MyCalculator myCalculator = new  
23. MyCalculator();  
  
        // Calculating and printing the sum of
```

divisors

```
        int result = myCalculator.divisor_sum(n);

        System.out.println("The sum of the divisors
of " + n + " is: " + result);

        scanner.close();

    }
}
```

### **OUTPUT:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_22.java } ; if ($?) { java prac_22 }
Enter an integer: 7
The sum of the divisors of 7 is: 8
PS C:\Users\dell\Desktop\JavaPractical> █
```

### **CONCLUSION:**

The AdvancedArithmetic interface defines the method signature `int divisor_sum(int n)`. The MyCalculator class implements the AdvancedArithmetic interface and provides the implementation for the `divisor_sum` method, which calculates the sum of all divisors of `n`. The Main class takes an integer input from the user, creates an object of MyCalculator, and prints the sum of the divisors of the input number.

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. Write a program that illustrates the significance of interface default method.

**PROGRAM CODE :**

```
import java.util.Scanner;

// Shape interface with a default method
interface Shape {

    String getColor();

    double getArea();

    default void printShapeInfo() {

        System.out.println("This is a shape with
color: " + getColor());

    }

}

// Circle class implementing Shape interface
class Circle implements Shape {

    private double radius;

    private String color;

    Circle(double radius, String color) {

        this.radius = radius;

        this.color = color;
```

```
}

@Override

public String getColor() {

    return color;

}

@Override

public double getArea() {

    return Math.PI * radius * radius;

}

@Override

public void printShapeInfo() {

    System.out.println("This is a circle with
color: " + color + " and area: " + getArea());

}

}

// Rectangle class implementing Shape interface
class Rectangle implements Shape {

    private double length;
```



```
private double width;

private String color;

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;

}

@Override

public void printShapeInfo() {
```

```
        System.out.println("This is a rectangle with  
color: " + color + " and area: " + getArea());  
    }  
}  
  
// Sign class  
class Sign {  
    private Shape shape;  
    private String text;  
  
    Sign(Shape shape, String text) {  
        this.shape = shape;  
        this.text = text;  
    }  
  
    void printSignInfo() {  
        shape.printShapeInfo();  
        System.out.println("Sign text: " + text);  
    }  
}  
  
public class Main {
```

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

    Scanner scanner = new Scanner(System.in);

    // Taking input for Circle

    System.out.println("Enter the radius and
color of the circle:");

    double radius = scanner.nextDouble();

    scanner.nextLine(); // Consume newline

    String circleColor = scanner.nextLine();

    Circle circle = new Circle(radius,
circleColor);

    // Taking input for Rectangle

    System.out.println("Enter the length, width,
and color of the rectangle:");

    double length = scanner.nextDouble();

    double width = scanner.nextDouble();

    scanner.nextLine(); // Consume newline

    String rectangleColor = scanner.nextLine();

    Rectangle rectangle = new
Rectangle(length, width, rectangleColor);
```

```
// Taking input for Sign text

System.out.println("Enter the text for the
sign:");

String signText = scanner.nextLine();

// Creating Sign objects

Sign circleSign = new Sign(circle,
signText);

Sign rectangleSign = new Sign(rectangle,
signText);

24. // Printing Sign information

System.out.println("\nCircle Sign Info:");

circleSign.printSignInfo();

System.out.println("\nRectangle Sign
Info:");

rectangleSign.printSignInfo();

scanner.close();

}

}
```

**OUTPUT:**

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_23.java } ; if ($?) { java prac_23 }

Enter the radius and color of the circle:
Enter the radius and color of the circle:
12
pink
Enter the length, width, and color of the rectangle:
12
3
black
Enter the text for the sign:
purvik

Circle Sign Info:
This is a circle with color: pink and area: 452.3893421169302
Sign text: purvik

Rectangle Sign Info:
This is a rectangle with color: black and area: 36.0
Sign text: purvik
PS C:\Users\dell\Desktop\JavaPractical> █
```

**CONCLUSION:**

The Shape interface has a default method `printShapeInfo()` that prints basic shape information. The Circle and Rectangle classes implement the Shape interface and override the `printShapeInfo()` method to provide specific information. The Sign class contains a Shape and text, and it prints the shape information along with the sign text. The Main class takes input from the user for the circle, rectangle, and sign text, creates objects, and prints the information.

Write a java program which takes two integers x & y as input, you 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.

**PROGRAM CODE :**

```
import java.util.Scanner;

public class prac_24 {

    public static void main(String[] args) {

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

```
try {  
    // Taking input for x and y  
    System.out.print("Enter the value of x: ");  
    int x = scanner.nextInt();  
  
    System.out.print("Enter the value of y: ");  
    int y = scanner.nextInt();  
  
    // Perform division  
    int result = x / y;  
    System.out.println("Result of " + x + " / "  
+ y + " = " + result);  
}  
catch (ArithmeticException e) {  
    // Handle division by zero  
    System.out.println("Error: Division by  
zero is not allowed.");  
}  
catch (Exception e) {  
    // Handle any other exceptions (like input  
mismatch)
```

25.

```
        System.out.println("Error: Invalid input.  
Please enter valid integers.");  
    }  
    finally {  
        scanner.close(); // Closing the scanner  
    }  
}  
}
```

### Output:

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_24.java } ;  
if ($?) { java prac_24 }  
Enter the value of x: 12  
Enter the value of y: 0  
Error: Division by zero is not allowed.  
PS C:\Users\dell\Desktop\JavaPractical> █
```

### CONCLUSION:

The program prompts the user to input two integers x and y. It tries to perform the division  $x/y$ . If y is zero, an `ArithmeticException` is caught, and an error message is printed. If the user enters something other than integers, an exception is caught, and the program informs the user of invalid input.

Write a Java program that throws an exception and catch it using a try-catch block.

**PROGRAM CODE :**

```
import java.util.Scanner;

public class prac_25 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        try {

            // Taking input from the user

            System.out.print("Enter a positive integer:

");

            int number = scanner.nextInt();

            // Throwing an exception if the number is
negative

            if (number < 0) {

                throw new Exception("Negative
numbers are not allowed!");

            }

            // If no exception occurs, print the input
number
```



```
        System.out.println("You entered: " +
number);

    }

    catch (Exception e) {

        // Catching the exception and displaying
an error message

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

    }

    finally {

        // Closing the scanner

        scanner.close();

    }

}

}
```

## Output:

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_25.java } ;
if ($?) { java prac_25 }
Enter a positive integer: 12
You entered: 12
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_25.java } ;
if ($?) { java prac_25 }
Enter a positive integer: 0
You entered: 0
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_25.java } ;
if ($?) { java prac_25 }
Enter a positive integer: -12
Error: Negative numbers are not allowed!
PS C:\Users\dell\Desktop\JavaPractical> 
```

**CONCLUSION:**

he user is asked to input an integer. If the user enters a negative number, the program throws an exception with a custom message. The catch block catches the exception and prints the error message. Regardless of whether an exception occurs, the finally block ensures the scanner is closed.

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).

**PROGRAM CODE :**

```
import java.util.Scanner;

// Custom exception (User-defined)
class NegativeValueException extends Exception
{
    public NegativeValueException(String
```

```
message) {  
    super(message);  
}  
}  
  
public class prac_26 {  
  
    // Method that throws a custom exception  
    public static void checkNumber(int number)  
throws NegativeValueException {  
        if (number < 0) {  
            throw new  
NegativeValueException("Negative numbers are  
not allowed.");  
        } else {  
            System.out.println("Valid number: " +  
number);  
        }  
    }  
  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);
```

27.

```
System.out.print("Enter a number: ");

int num = scanner.nextInt();

try {

    checkNumber(num);

} catch (NegativeValueException e) {

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

}

}
```

## Output:

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_26.java } ; if ($?) { java prac_26 }
Enter a number: -12
Caught Exception: Negative numbers are not allowed.
PS C:\Users\dell\Desktop\JavaPractical> █
```

## CONCLUSION:

NegativeValueException is a custom exception that is thrown when the user inputs a negative number. The throw keyword is used to explicitly throw this exception. The throws keyword in the method signature (checkNumber) indicates that this method may throw a custom exception.

## PART-VI File Handling & Streams

**AIM :** 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.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Prac_27 {
    public static void main(String[] args) {
        if (args.length == 0) {
            args = new String[]{"hello.txt"};
        }

        for (String fileName : args) {
            try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
                int lineCount = 0;
                while (reader.readLine() != null) {
                    lineCount++;
                }
                System.out.println(fileName + ": " + lineCount + " lines");
            } catch (IOException e) {
                System.err.println("Error reading file " + fileName + ": " + e.getMessage());
            }
        }
    }
}
```

28.

```
}
```

### **OUTPUT:**

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_27.java }  
; if ($?) { java Prac_27 }  
Error reading file hello.txt: hello.txt (The system cannot find the file specif  
ied)  
PS C:\Users\dell\Desktop>
```

### **CONCLUSION:**

This program counts the number of lines in a file using Java. It reads each file specified in the command-line arguments or defaults to hello.txt if no arguments are provided. The program uses `BufferedReader` to read each line and increments a counter for each line read. It handles file reading errors gracefully using a try-with-resources block. The program prints the number of lines for each file processed. This showcases efficient file handling and error

**AIM :** 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 Prac_28 {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java CharCount <file> <character>");
            return;
        }

        String fileName = args[0];
        char targetChar = args[1].charAt(0);

        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
            int charCount = 0;
            int c;
            while ((c = reader.read()) != -1) {
                if (c == targetChar) {
                    charCount++;
                }
            }
            System.out.println("The character '" + targetChar + "' appears " + charCount + " times in the file " +
                fileName);
        } catch (IOException e) {
            System.err.println("Error reading file " + fileName + ": " + e.getMessage());
        }
    }
}
```

### **OUTPUT:**

```
PS C:\Users\de11\Desktop> cd "c:\Users\de11\Desktop\" ; if ($?) { javac Prac_28
.java } ; if ($?) { java Prac_28 }
Usage: java CharCount <file> <character>
PS C:\Users\de11\Desktop> █
```

### **CONCLUSION:**

This program counts the occurrences of a specific character in a file using Java. It reads the file

character by character with `BufferedReader` and compares each character to the target character. If they match, it increments a counter. The program handles file reading errors using a try-with-resources block to ensure the reader is closed properly. It also provides usage instructions if the required command-line arguments are not provided. This showcases efficient character processing and error management in Java.

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

**PROGRAM CODE :**

```
30. import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Prac_29 {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java Prac_29 <file> <word>");
            return;
        }

        String fileName = args[0];
        String targetWord = args[1];

        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
            int wordCount = 0;
            String line;
            while ((line = reader.readLine()) != null) {
                String[] words = line.split("\\s+");
                for (String word : words) {
                    if (word.equals(targetWord)) {
                        wordCount++;
                    }
                }
            }
            System.out.println("The word '" + targetWord + "' appears " + wordCount + " times in the file " +
                fileName);
        } catch (IOException e) {
```



```

        System.err.println("Error reading file " + fileName + ": " + e.getMessage());
    }

    // Wrapper Class Example
    Integer wrapperInt = Integer.valueOf(10); // Using Integer wrapper class
    int primitiveInt = wrapperInt.intValue(); // Converting back to primitive int
    System.out.println("Wrapper Class Example: Integer value is " + wrapperInt + " and primitive int
value is " + primitiveInt);
}
}

```

### **OUTPUT:**

```

PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_29
.java } ; if ($?) { java Prac_29 }
Usage: java Prac_29 <file> <word>
PS C:\Users\dell\Desktop> 

```

### **CONCLUSION:**

This program demonstrates how to count the occurrences of a specific word in a file using Java. It reads the file line by line with `BufferedReader` and splits each line into words. It then compares each word to the target word and increments a counter if they match. The program handles file reading errors gracefully using a try-with-resources block. It also provides usage instructions if the required command-line arguments are not provided. This showcases efficient text processing and error management in Java.

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

32.

### **PROGRAM CODE :**

```

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

```

```
public class Prac_30 {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java Prac_30 <source file> <destination file>");
            return;
        }

        String sourceFile = args[0];
        String destinationFile = args[1];

        try (FileInputStream fis = new FileInputStream(sourceFile);
            FileOutputStream fos = new FileOutputStream(destinationFile)) {

            byte[] buffer = new byte[1024];
            int bytesRead;

            while ((bytesRead = fis.read(buffer)) != -1) {
                fos.write(buffer, 0, bytesRead);
            }

            System.out.println("File copied successfully from " + sourceFile + " to " + destinationFile);
        } catch (IOException e) {
            System.err.println("Error copying file: " + e.getMessage());
        }
    }
}
```

### **OUTPUT:**

```
PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_30
.java } ; if ($?) { java Prac_30 }
Usage: java Prac_30 <source file> <destination file>
```

### **CONCLUSION:**

This program demonstrates how to copy data from one file to another using byte streams in Java. It reads from a source file and writes to a destination file, creating the destination file if it does not exist. The program uses `FileInputStream` to read bytes and `FileOutputStream` to write bytes. It handles errors using a try-with-resources block to ensure streams are closed properly. The program also provides usage instructions if the required command-line arguments are not provided. This showcases efficient file handling and error management in Java.

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

## PART-VII Multithreading

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

### PROGRAM CODE :

```
import java.util.Scanner;

// Thread class by extending Thread class
class MyThread extends Thread {
    private int repeat;

    // Constructor to take user input for how many
times to display the message
    public MyThread(int repeat) {
        this.repeat = repeat;
    }

    @Override
    public void run() {
```

```
33.    for (int i = 0; i < repeat; i++) {  
        System.out.println("Hello World");  
    }  
}  
  
public class prac_32 {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.print("Enter the number of times  
to display 'Hello World': ");  
  
        int times = scanner.nextInt();  
  
        // Create and start the thread  
  
        MyThread thread = new MyThread(times);  
        thread.start();  
    }  
}
```

**Output:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_32.java } ; if ($?) { java prac_32 }
Enter the number of times to display 'Hello World': 2
Hello World
Hello World
```

### **CONCLUSION:**

The class MyThread extends the Thread class and overrides the run() method to display "Hello World" as many times as the user specifies. In the main method, the user inputs how many times they want the message displayed, and a thread is created and started using the start() method.

**Aim:** 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;

// Thread class for calculating sum of a portion of numbers
class SumThread extends Thread {
    private int start;
    private int end;
    private int partialSum;

    // Constructor to define range of numbers this thread will handle
    public SumThread(int start, int end) {
        this.start = start;
        this.end = end;
```

```

    }

    @Override
    public void run() {
        partialSum = 0;
        for (int i = start; i <= end; i++) {
            partialSum += i;
        }
    }

    // Method to return the partial sum calculated by this thread
    public int getPartialSum() {
        return partialSum;
    }
}

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

        // Input N and number of threads
        System.out.print("Enter the value of N (sum numbers from 1 to N): ");
        int N = scanner.nextInt();

        System.out.print("Enter the number of threads: ");
        int numThreads = scanner.nextInt();

        // Create an array to hold threads
        SumThread[] threads = new SumThread[numThreads];

        // Calculate the range of numbers each thread should handle
        int range = N / numThreads;
        int start = 1;

        // Create and start threads
        for (int i = 0; i < numThreads; i++) {
            int end = (i == numThreads - 1) ? N : (start + range - 1); // Last thread takes the
remaining range
            threads[i] = new SumThread(start, end);
            threads[i].start();
            start = end + 1;
        }
    }
}

```

34.

```

// Wait for all threads to finish and collect results
int totalSum = 0;
for (int i = 0; i < numThreads; i++) {
    try {
        threads[i].join(); // Wait for the thread to finish
        totalSum += threads[i].getPartialSum(); // Add each thread's partial sum to total
sum
    } catch (InterruptedException e) {
        System.out.println("Thread interrupted: " + e.getMessage());
    }
}

// Display the final result
System.out.println("The sum of numbers from 1 to " + N + " is: " + totalSum);
}
}

```

### Output:

```

PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\" ; if ($?) { javac prac_33.java } ; if ($?) { java prac_33 }
Enter the value of N (sum numbers from 1 to N): 3
Enter the number of threads: 3
The sum of numbers from 1 to 3 is: 6
PS C:\Users\dell\Desktop> 
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\" ; if ($?) { javac prac_33.java } ; if ($?) { java prac_33 }
Enter the value of N (sum numbers from 1 to N): 3
Enter the number of threads: 3
The sum of numbers from 1 to 3 is: 6
PS C:\Users\dell\Desktop> 

```

### CONCLUSION:

he program takes two inputs from the user: N, the number up to which we need to sum, and numThreads, the number of threads. SumThread Class. This class extends Thread and is responsible for calculating the sum of a specific range of numbers (from start to end). The run() method performs the summation for that thread, and getPartialSum() returns the result computed by the thread.

**Aim:** 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;
```

```
// Thread that generates a random number every 1 second
```

```
class NumberGenerator extends Thread {
```

```
    private final SharedData sharedData;
```

```
    public NumberGenerator(SharedData sharedData) {
```

```
        this.sharedData = sharedData;
```

```
    }
```

```
    @Override
```

```
    public void run() {
```

```
        Random random = new Random();
```

```
        while (true) {
```

```
            int number = random.nextInt(100); // Generate random number between 0 and 99
```

```
            sharedData.setNumber(number);
```

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

```
            try {
```

```
                Thread.sleep(1000); // Wait for 1 second
```

```
            } catch (InterruptedException e) {
```

```
                System.out.println("Number generation interrupted.");
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

```
// Thread that computes and prints the square of even numbers
```

```
class SquareCalculator extends Thread {
```

```
    private final SharedData sharedData;
```

```
    public SquareCalculator(SharedData sharedData) {
```

```
        this.sharedData = sharedData;
```

```
    }
```

```
    @Override
```



```
public void run() {
    while (true) {
        synchronized (sharedData) {
            if (sharedData.isEven()) {
                int number = sharedData.getNumber();
                System.out.println("Square of " + number + " is " + (number * number));
            }
        }
    }
}

// Thread that computes and prints the cube of odd numbers
class CubeCalculator extends Thread {
    private final SharedData sharedData;

    public CubeCalculator(SharedData sharedData) {
        this.sharedData = sharedData;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (sharedData) {
                if (!sharedData.isEven()) {
                    int number = sharedData.getNumber();
                    System.out.println("Cube of " + number + " is " + (number * number *
35. number));
                }
            }
        }
    }
}

// Shared data class to hold and manage the generated number
class SharedData {
    private int number;

    public synchronized void setNumber(int number) {
        this.number = number;
    }
}
```

```
    public synchronized int getNumber() {
        return number;
    }

    public synchronized boolean isEven() {
        return number % 2 == 0;
    }
}

public class MultiThreadedApplication {
    public static void main(String[] args) {
        SharedData sharedData = new SharedData();

        // Create and start the threads
        NumberGenerator numberGenerator = new NumberGenerator(sharedData);
        SquareCalculator squareCalculator = new SquareCalculator(sharedData);
        CubeCalculator cubeCalculator = new CubeCalculator(sharedData);

        numberGenerator.start();
        squareCalculator.start();
        cubeCalculator.start();
    }
}
```

**Output:**

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_34.java } ; if ($?) { java prac_34 }
Enter the number of random numbers to generate: 3
Generated Number: 55
Cube of 55 is: 166375
Generated Number: 43
Cube of 43 is: 79507
Generated Number: 23
Cube of 23 is: 12167
Number generation and processing complete.
PS C:\Users\dell\Desktop\JavaPractical>
```

**CONCLUSION:**

This thread generates random numbers (between 0 and 99) every second and stores them in the SharedData object. It takes N as input from the user, where N represents the number of random numbers to generate. This thread continuously checks the SharedData object. If the current number is even, it calculates and prints the square of the number. After processing, it sets the number to null to avoid repeated processing.

**Aim:** 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 :**

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

```
// Thread class to increment the value of the variable
```

```
class IncrementThread extends Thread {
```

```
    private int value;
```

```
    private int times;
```

```
    // Constructor to initialize the value and number of times to increment
```

```
    public IncrementThread(int value, int times) {
```

```
        this.value = value;
```

```
        this.times = times;
```

```
    }
```

```
    @Override
```

```
    public void run() {
```

```
        try {
```

```
            for (int i = 0; i < times; i++) {
```

```
                value++; // Increment the value by one
```

```
                System.out.println("Value after increment: " + value);
```

```
                Thread.sleep(1000); // Sleep for 1 second
```

```
            }
```

```
        } catch (InterruptedException e) {
```

```
            System.out.println("Thread interrupted: " + e.getMessage());
```

```
        }
```

```
    }
```

```
}
```

```
public class IncrementVariable {
```

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

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

```
        // Taking input from the user
```

```
        System.out.print("Enter the initial value: ");
```

```
        int initialValue = scanner.nextInt();
```

36.

```
System.out.print("Enter the number of times to increment: ");
int times = scanner.nextInt();

// Create and start the thread
IncrementThread incrementThread = new IncrementThread(initialValue, times);
incrementThread.start();

try {
    incrementThread.join(); // Wait for the thread to complete
} catch (InterruptedException e) {
    System.out.println("Main thread interrupted: " + e.getMessage());
}

System.out.println("Incrementing process completed.");
}
```

**Output:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_35.java } ; if ($?) { java pr
ac_35 }
Enter the initial value: 3
Enter the number of times to increment: 2
Value after increment: 4
Value after increment: 5
Incrementing process completed.
PS C:\Users\dell\Desktop\JavaPractical> 2
```

**CONCLUSION:**

This thread takes two inputs: the initial value of the variable and the number of times the value should be incremented. The run() method contains a loop that increments the value by one and displays the value after each increment. the Thread.sleep(1000) call makes the thread pause for 1 second after each increment.

**Aim:** 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 :**

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

```
// Custom thread class that takes the thread name and the number of times to run
```

```
class CustomThread extends Thread {
```

```
    private String threadName;
```

```
    private int times;
```

```
// Constructor to initialize the thread name and number of times to run
```

```
public CustomThread(String threadName, int times) {
```

```
    this.threadName = threadName;
```

```
    this.times = times;
```

```
}
```

```
@Override
```

```
public void run() {
```

```
    for (int i = 0; i < times; i++) {
```

```
        System.out.println(threadName + " is running.");
```

```
        try {
```

```
            Thread.sleep(500); // Sleep for half a second between each print
```

```
        } catch (InterruptedException e) {
```

```
            System.out.println(threadName + " was interrupted.");
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```
public class ThreadPriorityDemo {
```

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

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

```
// Taking input from the user for how many times each thread should run
```

```
System.out.print("Enter the number of times 'FIRST' thread should run: ");
```

```
int firstTimes = scanner.nextInt();
```

```
System.out.print("Enter the number of times 'SECOND' thread should run: ");
```

```
int secondTimes = scanner.nextInt();
```

37.

```
System.out.print("Enter the number of times 'THIRD' thread should run: ");
int thirdTimes = scanner.nextInt();

// Creating the threads
CustomThread firstThread = new CustomThread("FIRST", firstTimes);
CustomThread secondThread = new CustomThread("SECOND", secondTimes);
CustomThread thirdThread = new CustomThread("THIRD", thirdTimes);

// Setting thread priorities
firstThread.setPriority(3); // Priority of FIRST is set to 3
secondThread.setPriority(Thread.NORM_PRIORITY); // Default priority (5) for
SECOND
thirdThread.setPriority(7); // Priority of THIRD is set to 7

// Starting the threads
firstThread.start();
secondThread.start();
thirdThread.start();

try {
    // Wait for all threads to complete execution
    firstThread.join();
    secondThread.join();
    thirdThread.join();
} catch (InterruptedException e) {
    System.out.println("Main thread interrupted.");
}

System.out.println("All threads have completed execution.");
}
```

**Output:**

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_36.java } ; if ($?) { java prac_36 }
Enter the number of times 'FIRST' thread should run: 3
Enter the number of times 'SECOND' thread should run: 2
Enter the number of times 'THIRD' thread should run: 1
THIRD is running.
SECOND is running.
FIRST is running.
FIRST is running.
SECOND is running.
FIRST is running.
All threads have completed execution.
PS C:\Users\dell\Desktop\JavaPractical> █
```

**CONCLUSION:**

This class extends Thread and takes a thread name and the number of times the thread should run. The run() method prints the thread's name and then sleeps for half a second between prints. The program takes input from the user for how many times each thread (FIRST, SECOND, THIRD) should run.

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

**PROGRAM CODE :**

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

// Shared buffer class with synchronization
class SharedBuffer {
    private Queue<Integer> buffer = new LinkedList<>();
    private int capacity;

    public SharedBuffer(int capacity) {
        this.capacity = capacity;
    }

    // Method for the producer to add items to the buffer
    public synchronized void produce(int item) throws InterruptedException {
        while (buffer.size() == capacity) {
            wait(); // Wait if the buffer is full
        }
        buffer.add(item);
        System.out.println("Produced: " + item);
        notifyAll(); // Notify the consumer that an item has been produced
    }

    // Method for the consumer to take items from the buffer
    public synchronized int consume() throws InterruptedException {
```

```
        while (buffer.isEmpty()) {
            wait(); // Wait if the buffer is empty
        }
        int item = buffer.poll();
        System.out.println("Consumed: " + item);
        notifyAll(); // Notify the producer that space is available in the buffer
        return item;
    }
}

// Producer thread class
class Producer extends Thread {
    private SharedBuffer buffer;
    private int itemsToProduce;

    public Producer(SharedBuffer buffer, int itemsToProduce) {
        this.buffer = buffer;
        this.itemsToProduce = itemsToProduce;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < itemsToProduce; i++) {
                buffer.produce(i);
                Thread.sleep(500); // Simulate time taken to produce an item
            }
        } catch (InterruptedException e) {
            System.out.println("Producer interrupted.");
        }
    }
}

// Consumer thread class
class Consumer extends Thread {
    private SharedBuffer buffer;
    private int itemsToConsume;

    public Consumer(SharedBuffer buffer, int itemsToConsume) {
        this.buffer = buffer;
        this.itemsToConsume = itemsToConsume;
    }
}
```



```
@Override
public void run() {
    try {
        for (int i = 0; i < itemsToConsume; i++) {
            buffer.consume();
            Thread.sleep(1000); // Simulate time taken to consume an item
        }
    } catch (InterruptedException e) {
        System.out.println("Consumer interrupted.");
    }
}

}

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

        // Input for buffer capacity
        System.out.print("Enter the buffer capacity: ");
        int bufferCapacity = scanner.nextInt();

        // Input for the number of items to produce and consume
        System.out.print("Enter the number of items to produce: ");
        int itemsToProduce = scanner.nextInt();

        System.out.print("Enter the number of items to consume: ");
        int itemsToConsume = scanner.nextInt();

        // Create shared buffer
        SharedBuffer sharedBuffer = new SharedBuffer(bufferCapacity);

        // Create and start producer and consumer threads
        Producer producer = new Producer(sharedBuffer, itemsToProduce);
        Consumer consumer = new Consumer(sharedBuffer, itemsToConsume);

        producer.start();
        consumer.start();

        try {
            // Wait for both threads to complete execution
            producer.join();
```

```
        consumer.join();
    } catch (InterruptedException e) {
        System.out.println("Main thread interrupted.");
    }

    System.out.println("Producer and Consumer execution completed.");
}
}
```

**Output:**

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_37.java } ; if ($?) { java pr
ac_37 }
Enter the buffer capacity: 3
Enter the number of items to produce: 4
Enter the number of items to consume: 2
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Produced: 3
Producer and Consumer execution completed.
PS C:\Users\dell\Desktop\JavaPractical>
```

**CONCLUSION:**

This class represents a shared buffer with produce and consume methods.

produce(int item): Adds an item to the buffer. If the buffer is full, it waits until space is available.consume(): Removes an item from the buffer. If the buffer is empty, it waits until an item is available.notifyAll() and wait() are used to coordinate the producer and consumer thread.This thread produces items and adds them to the buffer. It simulates the production process by sleeping for 500 milliseconds between productions.

## PART-VIII Collection Framework and Generic

38. **AIM :** 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.*;

class MyStack {
    ArrayList<Object> list;

    MyStack(Object elements[]) {
        list = new ArrayList<Object>();
        for (int i = 0; i < elements.length; i++) {
            list.add(elements[i]);
        }
    }

    MyStack() {
        list = new ArrayList<Object>();
    }

    boolean isEmpty() {
        return (list.size() == 0);
    }

    Object peek() {
        return list.get(list.size() - 1);
    }

    Object pop() {
        Object ob = list.get(list.size() - 1);
        list.remove(list.size() - 1);
        return ob;
    }

    void push(Object o) {
        list.add(o);
    }
}
```

```
public class Prac_38 {  
    public static void main(String[] args) {  
        Integer arr[] = new Integer[] { 1, 2, 3, 4 };  
        MyStack s = new MyStack(arr);  
        System.out.println("Current top = " + s.peak());  
        System.out.println("Pushing 7,8,9 in the stack");  
        s.push(7);  
        s.push(8);  
        s.push(9);  
        s.pop();  
        System.out.println("Elements in the stack are: ");  
        while (!s.isEmpty()) {  
            System.out.println(s.pop());  
        }  
    }  
}
```

### **OUTPUT:**

```
PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_38  
.java } ; if ($?) { java Prac_38 }  
Current top = 4  
Pushing 7,8,9 in the stack  
Elements in the stack are:  
8  
7  
4  
3  
2  
1  
PS C:\Users\dell\Desktop> █
```

### **CONCLUSION:**

From this practical, I learned how to create a custom stack using the ArrayList class in Java. I implemented basic stack functionalities like checking if the stack is empty, getting the size, viewing the top element, and performing push and pop operations. This exercise helped me understand how to use an ArrayList to dynamically store elements and simulate a stack structure.

**AIM :** 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 :**

```
public class Prac_39 {

    public static <T extends Comparable<T>> void sortArray(T[] array) {
        int n = array.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j].compareTo(array[j + 1]) > 0) {
                    T temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) {
                break;
            }
        }
    }

    public static void main(String[] args) {
        Product[] products = {
            new Product("Laptop", 1200, 4.5),
            new Product("Phone", 800, 4.3),
            new Product("Headphones", 150, 4.7),
            new Product("Monitor", 300, 4.4)
        };

        sortArray(products);

        for (Product p : products) {
            System.out.println(p);
        }
    }
}
```

```
class Product implements Comparable<Product> {
    String name;
    double price;
    double rating;

    public Product(String name, double price, double rating) {
        this.name = name;
        this.price = price;
        this.rating = rating;
    }

    @Override
    public int compareTo(Product other) {
        return Double.compare(this.price, other.price);
    }

    @Override
    public String toString() {
        return name + " - $" + price + " - Rating: " + rating;
    }
}
```

### **OUTPUT:**

```
PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_39
.java } ; if ($?) { java Prac_39 }
Headphones - $150.0 - Rating: 4.7
Monitor - $300.0 - Rating: 4.4
Phone - $800.0 - Rating: 4.3
Laptop - $1200.0 - Rating: 4.5
PS C:\Users\dell\Desktop>
```

### **CONCLUSION:**

Through this practical, I gained insights into implementing a generic method in Java to sort arrays of objects that implement the Comparable interface. I learned how to ensure flexibility and reusability by enabling the method to sort various types of objects, such as products, customers, and orders, based on their natural ordering.

**AIM :** 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 Prac_40 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the text: ");
        String inputText = sc.nextLine();

        inputText = inputText.toLowerCase();
        HashMap<String, Integer> wordCountMap = new HashMap<>();

        StringBuilder currentWord = new StringBuilder();

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

            if (Character.isLetter(c) || Character.isDigit(c)) {
                currentWord.append(c);
            } else {
                if (currentWord.length() > 0) {
                    String word = currentWord.toString();
                    wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
                    currentWord.setLength(0);
                }
            }
        }

        if (currentWord.length() > 0) {
            String word = currentWord.toString();
            wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
        }

        TreeSet<String> sortedWords = new TreeSet<>(wordCountMap.keySet());

        System.out.println("Word occurrences:");
        for (String word : sortedWords) {
            System.out.println(word + ": " + wordCountMap.get(word));
        }

        sc.close();
    }
}
```



**OUTPUT:**

```
PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_40
.java } ; if ($?) { java Prac_40 }
Enter the text: my name is purik anghan. my id is 23dcs003.
Word occurrences:
23dcs003: 1
anghan: 1
id: 1
is: 2
my: 2
name: 1
purik: 1
PS C:\Users\dell\Desktop>
```

**CONCLUSION:**

In this practical, We learned how to use Java's Map and Set classes to count and display the occurrences of words in a given text. We implemented a method that not only counts the occurrences but also sorts the words in alphabetical order. This exercise enhanced my understanding of utilizing collections to efficiently manage and manipulate data.

**AIM :** 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.util.*;
import java.io.*;

public class Prac_41 {
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the file name you want to scan : ");
        String f = sc.nextLine();
        File file = new File(f);
        FileReader br = new FileReader(file);
        BufferedReader fr = new BufferedReader(br);
        String keywords[] = new String[] { "abstract", "assert ", "boolean", "break", "byte", "case", "catch",
"char",
        "class",
        "continue", "default", "do", "double", "else", "enum ", "extends", "final", "finally",
        "float", "for", "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" };
        HashSet<String> set = new HashSet<String>();
        for (int i = 0; i < keywords.length; ++i) {
            set.add(keywords[i]);
        }
        String st;
        int count = 0;
        while ((st = fr.readLine()) != null) {
            StringTokenizer str = new StringTokenizer(st, " +/,%<>:=&!~()");

            while (str.hasMoreTokens()) {
                String swre = str.nextToken();
                if (set.contains(swre)) {
                    count++;
                }
            }
        }
        System.out.println("Total keywords are : " + count);
        fr.close();
        sc.close();
    }
}
```

**OUTPUT:**

```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_41.java }  
; if ($?) { java Prac_41 }  
Enter the file name you want to scan : Prac_40.java  
Total keywords are : 18  
PS C:\Users\dell\Desktop>
```

**CONCLUSION:**

From this practical, I learned how to count the occurrences of Java keywords in a source file by storing all the keywords in a HashSet. By using the contains() method, I was able to check whether a word is a keyword or not. This practical improved my skills in working with Java's collection framework, particularly using sets for fast lookups.