CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: JAVA PROGRAMMING
Semester: 3
Subject Code: CSE201
Academic year: 2024 - 25

PART-I

No.	Data Types, Variables, String, Control Statements, Operators, Arrays
1.	Demonstration of installation steps of Java , Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE, or BlueJ and Console Programming.

1. Installation of Java

Steps to install Java Development Kit (JDK):

➢ Download JDK:

- Go to the Oracle JDK download page: [Oracle JDK Downloads] (https://www.oracle.com/java/technologies/javase-downloads.html).
- Select the appropriate JDK version for your operating system (Windows, macOS,Linux).
- Download the installer package (.exe for Windows, .dmg for macOS, .tar.gz for Linux).

➢ Install JDK:

- Windows: Double-click the downloaded .exe file and follow the installation instructions.
- macOS: Double-click the downloaded .dmg file, then drag and drop the JDK package icon to the Applications folder.
- Linux: Extract the downloaded .tar.gz file to a directory and follow the instructions in the

README file for installation.

- ➢ Set JAVA_HOME (Optional):

- Windows: Set the JAVA_HOME environment variable to the JDK installation directory.
- macOS/Linux: Add the JDK bin directory to your PATH and set JAVA_HOME in your shell profile (e.g., ~/.bash_profile, ~/.bashrc).

- ➢ Verify Installation:

- Open a terminal or command prompt.
- Type `java -version` and `javac -version` to verify that Java runtime and compiler are installed correctly.

2. Introduction to Object-Oriented Concepts

Object-oriented programming (OOP) revolves around the concept of objects, which are instances of classes. Key principles include:

- Classes and Objects: Classes define the blueprint for objects.
- Encapsulation: Bundling data (attributes) and methods (functions) that operate on the data within a single unit (class).
- Inheritance: Mechanism where a new class (derived or child class) is created from an existing class (base or parent class).
- Polymorphism: Ability of different objects to be treated as instances of the same class through method overriding and overloading.

3. Comparison of Java with Other Object-Oriented Programming Languages

Java is often compared with languages like C++, C#, and Python in terms of syntax, features, and application domains. Key points of comparison include:

- Syntax: Java has a C-style syntax with similarities to C++.
- Memory Management: Java uses automatic garbage collection, unlike C++ which requires manual memory management.
- Platform Independence: Java programs are compiled into bytecode, which can run on any JVM, making it platform-independent.

- Libraries: Java has a rich standard library (Java API) comparable to those in C++ and C#.
- Community and Ecosystem: Java has a large developer community and extensive third-party libraries and frameworks.

4. Introduction to JDK, JRE, JVM, Javadoc, Command Line Arguments

- JDK (Java Development Kit): Includes tools for developing and running Java programs, including JRE and development tools such as javac (Java compiler).
- JRE (Java Runtime Environment): Includes JVM (Java Virtual Machine) and libraries required to run Java applications, but does not include development tools.
- JVM (Java Virtual Machine): Executes Java bytecode and provides a runtime environment for Java programs.
- Javadoc: Tool for generating API documentation from Java source code comments.
- Command Line Arguments: Parameters passed to a Java program when it is invoked from the command line.

5. Introduction to Eclipse or NetBeans IDE (Integrated Development Environment)

- Eclipse : A widely used open-source IDE for Java development, also supports other programming languages through plugins. Features include code editing, debugging, and version control integration.
- NetBeans: Another popular open-source IDE primarily for Java development, with features similar to Eclipse.

6. Introduction to BlueJ and Console Programming

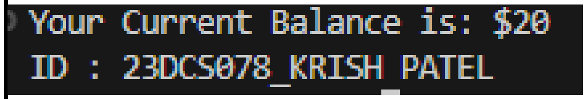
- BlueJ : A lightweight IDE specifically designed for teaching and learning Java programming, providing a simplified interface and visualization tools for object-oriented concepts.
- Console Programming : Refers to writing Java programs that interact with users via text-based input and output through the console (command line interface).

2. 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.

PROGRAM CODE:

```
public class P2{

public static void main(String args[]) {

int balance=20;

System.out.println("Your Current Balance is: $"+balance);

System.out.println("ID : 23DCS078_KRISH PATEL");

} }
```

OUTPUT:



```
Your Current Balance is: $20
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

The following Java program reads a user's current balance and displays it. This program uses variable declaration for simple variables and concatenation for output. The program illustrates one of the effective ways to handle simple variables and format the output of this information in Java. This is a pretty simple example and therefore good for understanding the most basic I/O operations in Java.

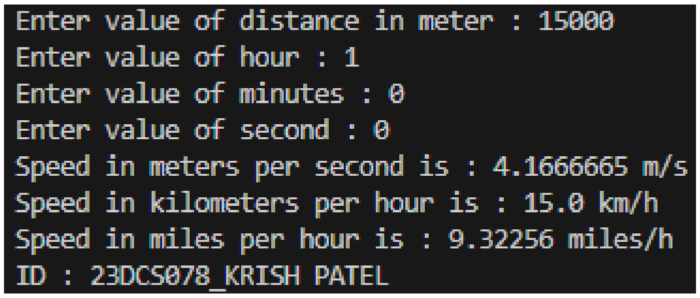
3. Write a program to take the user for a distance (in meters) and the time taken (as three numbers: hours, minutes, seconds), and display the speed, in meters per second, kilometers per hour and miles per hour (hint:1 mile = 1609 meters).

PROGRAM CODE:

```
import java.util.*;

public class P3 {

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

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

System.out.print("Enter value of distance in meter : ");

float meter = sc.nextFloat();

System.out.print("Enter value of hour : ");

float hour = sc.nextFloat();

System.out.print("Enter value of minutes : ");

float minutes = sc.nextFloat();

System.out.print("Enter value of second : ");

float second = sc.nextFloat();

float total_minutes = (hour * 60) + minutes + (second / 60);

float kilometers = meter / 1000;

float miles = meter / 1609;

float total_second = total_minutes * 60;

float total_hour = total_minutes / 60;

System.out.println("Speed in meters per second is : " + meter / total_second + " m/s");

System.out.println("Speed in kilometers per hour is : " + kilometers / total_hour + "
km/h");

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

System.out.println("ID : 23DCS078_KRISH PATEL");

sc.close();

} }
```

OUTPUT:



```
Enter value of distance in meter : 15000
Enter value of hour : 1
Enter value of minutes : 0
Enter value of second : 0
Speed in meters per second is : 4.1666665 m/s
Speed in kilometers per hour is : 15.0 km/h
Speed in miles per hour is : 9.32256 miles/h
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

The Java program below calculates and outputs the speed in meters per second, kilometers per hour, and miles per hour based on the user's input for distance and time. In this case, the example shows input through the Scanner class, arithmetic operators about how to perform the calculations, and formatting of output. This example is very effective in explaining the basics of input handling and conversion, calculation aspects in Java.

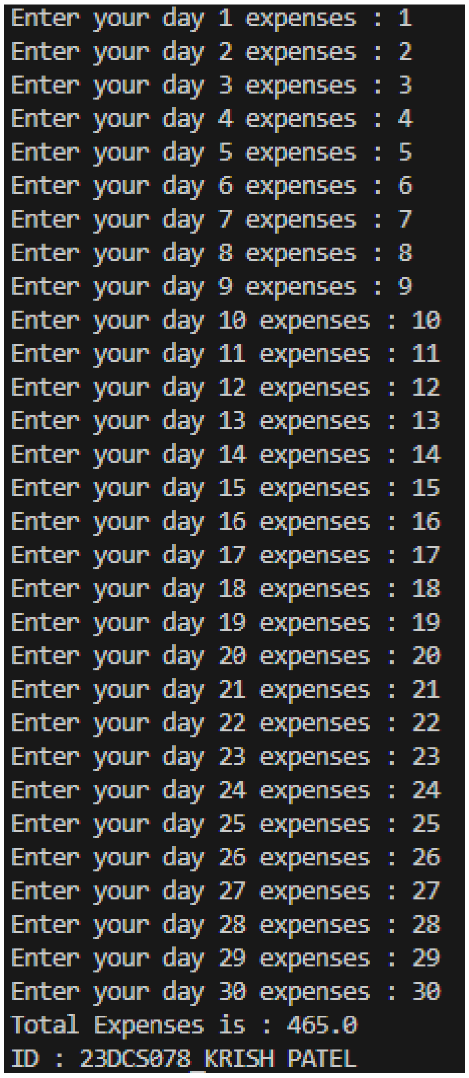
4.

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.*;

public class P4 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

float Expenses[] = new float[31];

float total_expenses = 0;

for(int i = 1 ; i <= 30 ; i++) {

System.out.print("Enter your day " + i + " expenses : ");
Expenses[i] = sc.nextFloat();

total_expenses += Expenses[i];

}

System.out.println("Total Expenses is : " + total_expenses);

System.out.println("ID : 23DCS078_KRISH PATEL");

sc.close();

} }
```

OUTPUT:



```
Enter your day 1 expenses : 1
Enter your day 2 expenses : 2
Enter your day 3 expenses : 3
Enter your day 4 expenses : 4
Enter your day 5 expenses : 5
Enter your day 6 expenses : 6
Enter your day 7 expenses : 7
Enter your day 8 expenses : 8
Enter your day 9 expenses : 9
Enter your day 10 expenses : 10
Enter your day 11 expenses : 11
Enter your day 12 expenses : 12
Enter your day 13 expenses : 13
Enter your day 14 expenses : 14
Enter your day 15 expenses : 15
Enter your day 16 expenses : 16
Enter your day 17 expenses : 17
Enter your day 18 expenses : 18
Enter your day 19 expenses : 19
Enter your day 20 expenses : 20
Enter your day 21 expenses : 21
Enter your day 22 expenses : 22
Enter your day 23 expenses : 23
Enter your day 24 expenses : 24
Enter your day 25 expenses : 25
Enter your day 26 expenses : 26
Enter your day 27 expenses : 27
Enter your day 28 expenses : 28
Enter your day 29 expenses : 29
Enter your day 30 expenses : 30
Total Expenses is : 465.0
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This Java program reads from a user the daily expenses over a period of 30 days and calculates the total expenses. The program illustrates the use of arrays to store daily expenses and simple loops for their summation. The example shows basic array manipulation and user input handling in Java.

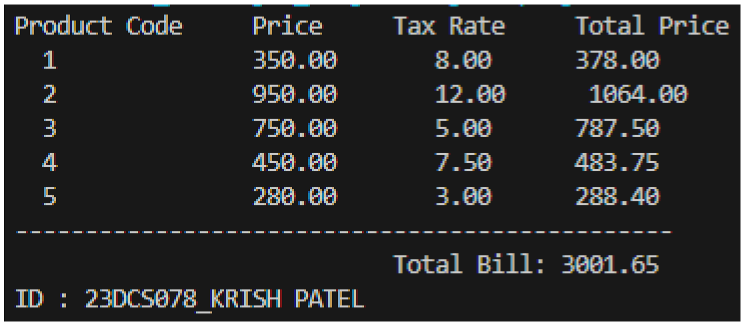
5.

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:

```
public class P5 {

public static void main(String[] args) {

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

double[] prices = {350.0, 950.0, 750.0, 450.0, 280.0};

double totalBill = 0;

System.out.printf("Product Code     Price     Tax Rate     Total Price\n");

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

int productCode = productCodes[i];

double price = prices[i];

double taxRate;

switch (productCode) {

case 1:

taxRate = 0.08; // 8% tax for motor

break;
case 2:
taxRate = 0.12; // 12% tax for fan
break;
case 3:
taxRate = 0.05; // 5% tax for tube light
break;
case 4:
taxRate = 0.075; // 7.5% tax for wires
break;
default:
taxRate = 0.03; // 3% tax for all other items
break;
}
double taxAmount = price * taxRate;
double totalPrice = price + taxAmount;
totalBill += totalPrice;
System.out.printf("  %d             %.2f       %.2f      %.2f\n",productCode,price,taxRate
*100,totalPrice);
}
System.out.println("--------------------------------------------");
System.out.printf("                          Total Bill: %.2f\n", totalBill);
System.out.println("ID : 23DCS078_KRISH PATEL");

}

}
```

OUTPUT:



```
Product Code      Price      Tax Rate      Total Price
  1               350.00       8.00        378.00
  2               950.00       12.00        1064.00
  3               750.00       5.00        787.50
  4               450.00       7.50        483.75
  5               280.00       3.00        288.40
------------------------------------------------
                           Total Bill: 3001.65
ID : 23DCS078_KRISH PATEL
```

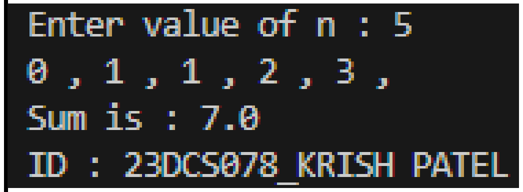
CONCLUSION:
This Java program will calculate the total bill for a myriad of products, applying different tax rates depending on the product codes. It covers arrays, loops, and switch statements to run multiple conditions and arithmetic operations. The example will project effectively how one can do basic control structures and format output in Java.

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

PROGRAM CODE:

```
import java.util.*;

public class P6 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter value of n : ");

int n = sc.nextInt();
double a = 0;

double b = 1;

double  sum = a + b;

System.out.print(String.format("%.0f" + " , " + "%.0f" + " , ", a , b ));

for(int i = 0 ; i < n - 2 ; i++) {

double c = a + b;

a = b;

b = c;

System.out.print(String.format("%.0f" + " , ", c));

sum += c;

}

System.out.println("\nSum is : " + sum);

System.out.println("ID : 23DCS078_KRISH PATEL");

sc.close();

} }
```

OUTPUT:



```
Enter value of n : 5
0 , 1 , 1 , 2 , 3 ,
Sum is : 7.0
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This Java program generates the Fibonacci sequence up to n terms and calculates its sum. The program demonstrates an example of using loops for iterative calculations and input through the Scanner class with formatted output. In this example, one gets a very good illustration of how basic sequence generation and summation go in Java.

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'

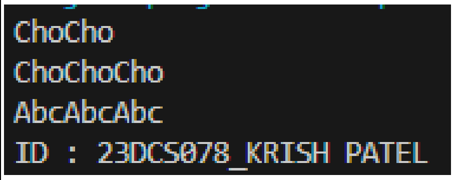
front_times('Chocolate', 3) → 'ChoChoCho'

front_times('Abc', 3) → 'AbcAbcAbc'

PROGRAM CODE:

```
public class P7 {

public static String front_times(String str , int n) {

String s4 = "";

String s5 = "";

int length = str.length();

if(str.length() > 3) {

s4 = str.substring(0, 3);

}

else {

s4 = str.substring(0,length);

}

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

s5 += s4;

}

return s5;
}
public static void main(String[] args) {
String s1 = "Chocolate";
String s2 = "Chocolate";
String s3 = "Abc";
front_times(s1, 2);
System.out.println(  front_times(s1, 2));
front_times(s2, 3);
System.out.println(  front_times(s2, 3));
front_times(s3, 3);
System.out.println(  front_times(s3, 3));
System.out.println("ID : 23DCS078_KRISH PATEL");
} }
```

OUTPUT:



```
ChoCho
ChoChoCho
AbcAbcAbc
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

The Java program below defines a method that repeats the first three characters of the input string n times and then prints the results for several inputs. The example illustrates string manipulation extracting a substring and concatenation with basic control structures. In this way, it will demonstrate efficiently the manipulation of strings and the use of methods in Java.

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

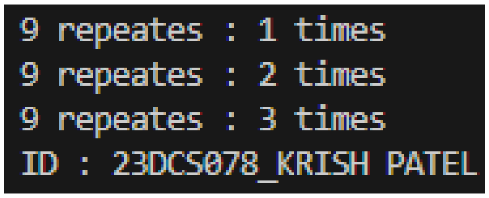
array_count9([1, 9, 9, 3, 9]) → 3

PROGRAM CODE:

```
public class P8 {

public static int array_count(int arr[],int count) {

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

if(arr[i] == 9) {

count ++;

} }

return count;

}

public static void main(String[] args) {

int arr1[] = {1,2,9};

int arr2[] = {1,9,9};

int arr3[] = {1,9,9,3,9};

System.out.println("9 repeates : " +array_count(arr1,0) + " times");

System.out.println("9 repeates : " + array_count(arr2, 0)+ " times");

System.out.println("9 repeates : " +array_count(arr3, 0)+ " times");

System.out.println("ID : 23DCS078_KRISH PATEL");
```

} }

OUTPUT:



```
9 repeates : 1 times
9 repeates : 2 times
9 repeates : 3 times
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:
This Java program will count the occurrences of the number 9 in different integer arrays and print out results. This program is an example of basic operations on arrays and counting values handled in Java.

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

double_char('The') → 'TThhee'

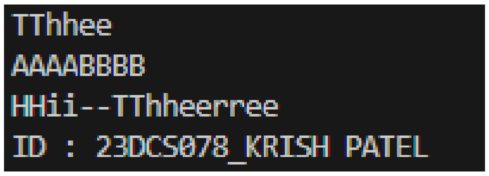
double_char('AAbb') → 'AAAAbbbb'

double_char('Hi-There') → 'HHii--TThheerree'

PROGRAM CODE:

```
public class P9 {

public static void main(String[] args) {

System.out.println(double_char("The"));

System.out.println(double_char("AABB"));

System.out.println(double_char("Hi-There"));

System.out.println("ID : 23DCS078_KRISH PATEL");

}

public static String double_char(String str){

String result="";
char ch=0;

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

ch=str.charAt(i);

result+=ch;

result+=ch;

}

return result;

} }
```

OUTPUT:



```
TThhee
AAAABBBB
HHii--TThheerree
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This is a simple Java program that doubles every character in a string and then prints the results after transformation. In this example, you will see how to manipulate strings at a very basic level: access to a character and concatenation in a loop. The example will then demonstrate how, in Java, a string might be modified and processed to bring about certain formatting.

10. Perform following functionalities of the string:

● Find Length of the String

● Lowercase of the String

● Uppercase of the String

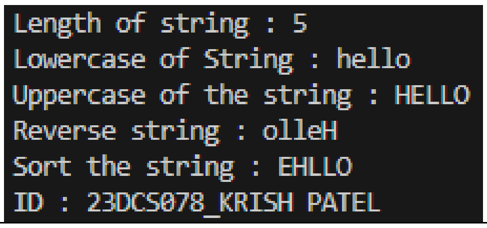
● Reverse String

● Sort the string

PROGRAM CODE:

```
import java.util.Arrays;

public class P10 {

public static void main(String[] args) {

String str="Hello";

StringBuilder str1=new StringBuilder();

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

System.out.println("Lowercase of String : "+str.toLowerCase());

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

str1.append(str);

System.out.println("Reverse string : "+str1.reverse());

if(!str.isEmpty()){

String tqr=str.toUpperCase();

char tempArray[] = tqr.toCharArray();

Arrays.sort(tempArray);

System.out.println("Sort the string : "+new String(tempArray));

System.out.println("ID : 23DCS078_KRISH PATEL");

 } } }
```

OUTPUT:



```
Length of string : 5
Lowercase of String : hello
Uppercase of the string : HELLO
Reverse string : olleH
Sort the string : EHLLO
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This Java program demonstrates many string manipulations dealing with length calculation, case conversion, reversal, and sorting. It uses StringBuilder effectively for the reverse and Arrays.sort for sorting characters. The following example shows very essential techniques of string processing using Java's in-built functions to handle string data.

11. Perform following Functionalities of the string: “CHARUSAT UNIVERSITY”

● Find length

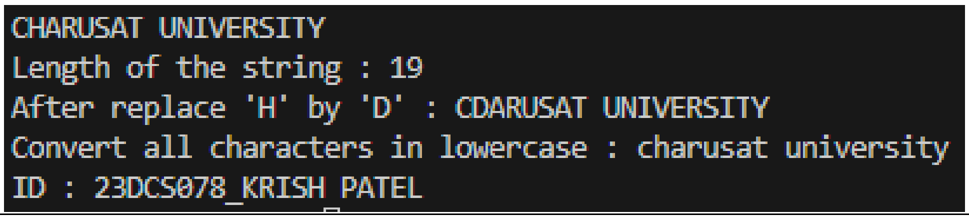
● Replace ‘H’ by ‘FIRST LATTER OF YOUR NAME’

● Convert all character in lowercase

PROGRAM CODE:

```
public class P11 {

public static void main(String[] args) {

String  str="CHARUSAT UNIVERSITY";

System.out.println(str);

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

System.out.println("After replace 'H' by 'D' : "+str.replace('H', 'D'));

System.out.println("Convert all characters in lowercase : "+str.toLowerCase());

System.out.println("ID : 23DCS078_KRISH PATEL");

} }
```

OUTPUT:



```
CHARUSAT UNIVERSITY
Length of the string : 19
After replace 'H' by 'D' : CDARUSAT UNIVERSITY
Convert all characters in lowercase : charusat university
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This Java program below demonstrates some of the basic string manipulation techniques like calculating the length of a string, replacing characters in a string, and case change of characters. The example clearly explains all the basic operations on strings and provides an excellent example of processing string data under Java. This is a very simple and descriptive code that may help in understanding the primary string methods in Java.

Supplementary Experiment:

Write a Java program to count and print all duplicates in the input string.

Sample Output: The given string is: resource

The duplicate characters and counts are:

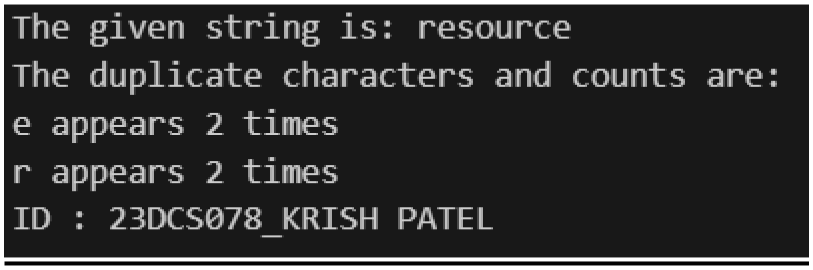
e appears 2 times

r appears 2 times

PROGRAM CODE:

```
public class DuplicateCharacters {

public static void main(String[] args) {

String input = "resource";

System.out.println("The given string is: " + input);

findDuplicates(input);

System.out.println("ID : 23DCS078_KRISH PATEL");

}

public static void findDuplicates(String str) {

int[] charCount = new int[256]; // ASCII character set size

// Count the occurrences of each character
for (int i = 0; i < str.length(); i++) {

charCount[str.charAt(i)]++;

}

System.out.println("The duplicate characters and counts are:");

// Print the characters that appear more than once

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

if (charCount[i] > 1) {

System.out.println((char) i + " appears " + charCount[i] + " times");

} } } }
```

OUTPUT:



```
The given string is: resource
The duplicate characters and counts are:
e appears 2 times
r appears 2 times
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This Java program is a simple and efficient way to identify and print all duplicate characters in a given string. The design of this solution is based on an array and works in linear time; hence, it is very efficient and simple. This solution nicely demonstrates the power of simple programming ideas for solving common problems.

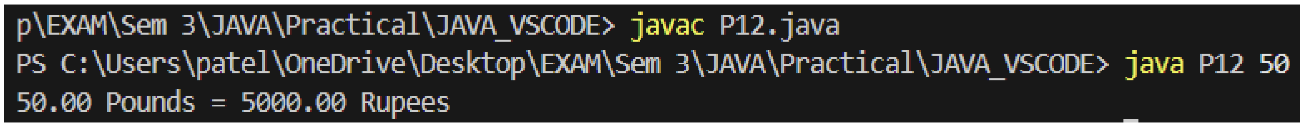
PART-III Object Oriented Programming: Classes, Methods, Constructors

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.*;
public class P12 {
private static final double CONVERSION_RATE = 100.0;
public static void main(String[] args) {
if (args.length > 0) {
double pounds = Double.parseDouble(args[0]);
double rupees = convertToRupees(pounds);
System.out.printf("%.2f Pounds = %.2f Rupees\n", pounds, rupees);
} else {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter amount in Pounds: ");
double pounds = scanner.nextDouble();
double rupees = convertToRupees(pounds);
System.out.printf("%.2f Pounds = %.2f Rupees\n", pounds, rupees);
System.out.println("ID : 23DCS078_KRISH PATEL");
} }
private static double convertToRupees(double pounds) {

return pounds * CONVERSION_RATE;

} }
```

OUTPUT:



```
p\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P12.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P12 50
50.00 Pounds = 5000.00 Rupees
```

CONCLUSION:

This program defines a CurrencyConverter class for the conversion of an amount in pounds to rupees using a predefined conversion rate. The conversion may take a place with either the amount passed as a command-line argument or entered interactively through the console. It then displays the amount that will be converted with formatted output.

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.

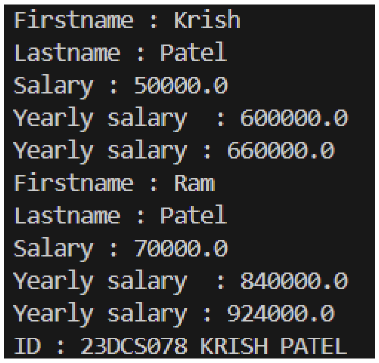
PROGRAM CODE:

```
class employee {

String firstname;

String lastname;

double salary;

double yearlysalary;

employee(String a , String b , double c) {
```

```
firstname = a;

lastname = b;

if(salary < 0) {

salary = 0.0;

}

salary = c;

}

public void display()  {

System.out.println("Firstname : " + firstname  + "\n" + "Lastname : " + lastname + "\n"
+"Salary : " + salary);

}

public void yearlysalary() {

yearlysalary =  salary * 12;

System.out.println("Yearly salary  : " + yearlysalary);

}

public void displaysalary() {

System.out.println("Yearly salary : " + yearlysalary);

}

public void raisesalary() {

yearlysalary = yearlysalary + (yearlysalary * 10)/100;

} }

public class P13 {
```

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

employee e1 = new employee("Krish","Patel",50000);

e1.display();

e1.yearlysalary();

e1.raisesalary();

e1.displaysalary();

employee e2 = new employee("Ram","Patel",70000);

e2.display();

e2.yearlysalary();

e2.raisesalary();

e2.displaysalary();

System.out.println("ID : 23DCS078_KRISH PATEL");

}

}
```

OUTPUT:



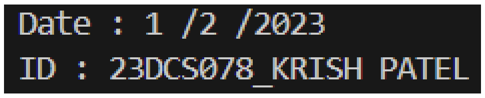
```
Firstname : Krish
Lastname : Patel
Salary : 50000.0
Yearly salary  : 600000.0
Yearly salary : 660000.0
Firstname : Ram
Lastname : Patel
Salary : 70000.0
Yearly salary  : 840000.0
Yearly salary : 924000.0
ID : 23DCS078_KRISH PATEL
```

CONCLUSION: This code defines a class of employees to hold details of the employees and determine salaries. The code increases salaries by 10%. The main method creates two instances of the employee, then displays their information and applies a salary raise.

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

PROGRAM CODE:

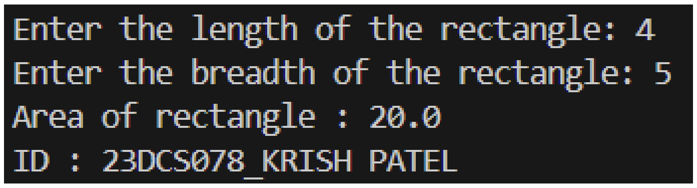
```
class Date{
int day;
int month;
int year;
Date(int a , int b , int c) {
month = a;
day =  b;
year = c;
}
public void displayDate() {
System.out.println("Date : " +  month + " /" + day + " /" + year);
} }
public class P14 {
public static void main(String[] args) {
Date d1 = new Date(1, 2, 2023);
d1.displayDate();
```

	System.out.println("ID : 23DCS078_KRISH PATEL"); } } **OUTPUT:**  Date : 1 /2 /2023 ID : 23DCS078_KRISH PATEL **CONCLUSION:** This code defines a class to store and display a date in the format month/day/year. Then, a Date object is created in the main method, after which the date of the object is displayed.

15.	Write a program to print the area of a rectangle by creating a class named 'Area' taking the values of its length and breadth as parameters of its constructor and having a method named 'returnArea' which returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.
	
	PROGRAM CODE:
	
	import java.util.*;
	
	class Area {
	
	float length;
	
	float breadth;
	
	Area(float a , float b) {
	
	length = a;
	
	breadth = b;
	
	}
	
	float returnArea() {
	
	return length * breadth;
	
	} }

```
public class P15 {

public static void main(String[] args) {

float a,b;

Scanner sc = new Scanner(System.in);

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

a = sc.nextFloat();

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

b = sc.nextFloat();

Area rectangle = new Area(a,b);

System.out.println("Area of rectangle : " + rectangle.returnArea());

System.out.println("ID : 23DCS078_KRISH PATEL");

sc.close();

} }
```

OUTPUT:



```
Enter the length of the rectangle: 4
Enter the breadth of the rectangle: 5
Area of rectangle : 20.0
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This code defines a class, Area, that will calculate an area given a rectangle of length and breadth. In the main method, it obtains user input for dimensions of a rectangle, creates an Area object, calculates it, and prints it. The program illustrates one of the ways in which constructors, methods, and the handling of user input may be employed in Java. Also shown is the proper way to close the Scanner object after its use.

16. 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:

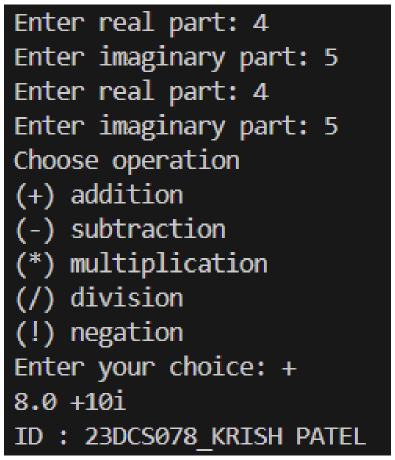
```
import java.text.DecimalFormat;

import java.util.Scanner;

class Complex {

float real;

float imag;

static final DecimalFormat df = new DecimalFormat("+#;-#");

Scanner sc = new Scanner(System.in);

void getData() {

System.out.print("Enter real part: ");

real = sc.nextFloat();

System.out.print("Enter imaginary part: ");

imag = sc.nextFloat();

}

Complex add(Complex b) {

Complex temp = new Complex();

temp.real = this.real + b.real;

temp.imag = this.imag + b.imag;

return temp;
```

```
}

Complex subtract(Complex b) {

Complex temp = new Complex();

temp.real = this.real - b.real;

temp.imag = this.imag - b.imag;

return temp;

}

Complex multiply(Complex b) {

Complex temp = new Complex();

temp.real = (this.real * b.real) - (this.imag * b.imag);

temp.imag = (this.real * b.imag) + (this.imag * b.real);

return temp;

}

Complex divide(Complex b) {

Complex temp = new Complex();

float denominator = (b.real * b.real) + (b.imag * b.imag);

temp.real = ((this.real * b.real) + (this.imag * b.imag)) / denominator;

temp.imag = ((this.imag * b.real) - (this.real * b.imag)) / denominator;

return temp;

}

Complex negate() {
```

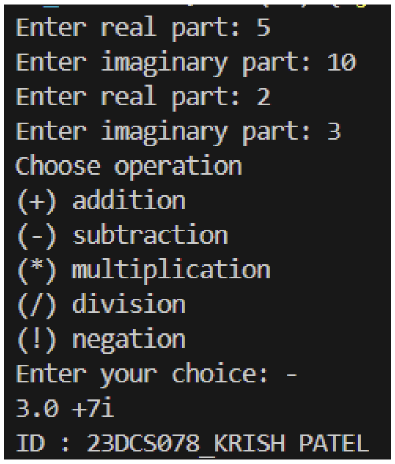
```
Complex temp = new Complex();
temp.real = -this.real;
temp.imag = -this.imag;
return temp;
}
void setData() {
System.out.println(real + " " + df.format(imag) + "i");
} }
public class P16 {
public static void main(String[] args) {
Complex obj1 = new Complex();
Complex obj2 = new Complex();
Complex obj3 = new Complex();
Scanner sc = new Scanner(System.in);
obj1.getData();
obj2.getData();
System.out.println("Choose operation ");
System.out.println("(+) addition ");
System.out.println("(-) subtraction ");
System.out.println("(*) multiplication ");
System.out.println("(/) division ");
System.out.println("(!) negation ");

System.out.print("Enter your choice: ");

char choice = sc.next().charAt(0);

switch (choice) {

case '+':

obj3 = obj1.add(obj2);

obj3.setData();

break;

case '-':

obj3 = obj1.subtract(obj2);

obj3.setData();

break;

case '*':

obj3 = obj1.multiply(obj2);

obj3.setData();

break;

case '/':

obj3 = obj1.divide(obj2);

obj3.setData();

break;

case '!':
```

```
obj3 = obj1.negate();

obj3.setData();

obj3 = obj2.negate();

obj3.setData();

break;

default:

System.out.println("Enter a valid operator.");

}

System.out.println("ID : 23DCS078_KRISH PATEL");

sc.close();

} }
```

OUTPUT:



```
Enter real part: 4
Enter imaginary part: 5
Enter real part: 4
Enter imaginary part: 5
Choose operation
(+) addition
(-) subtraction
(*) multiplication
(/) division
(!) negation
Enter your choice: +
8.0 +10i
ID : 23DCS078_KRISH PATEL
```



```
Enter real part: 5
Enter imaginary part: 10
Enter real part: 2
Enter imaginary part: 3
Choose operation
(+) addition
(-) subtraction
(*) multiplication
(/) division
(!) negation
Enter your choice: -
3.0 +7i
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

This code defines a Complex class for addition, subtraction, multiplication, division, and negation of complex numbers. In the main method of your program, read two complex numbers from the user and allow the user to select an operation to perform on those two numbers. Then output the result of the operation using the setData method.

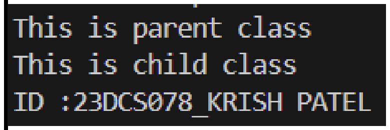
PART-IV Inheritance, Interface, Package

17 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:

```
class parent {
void display()  {
System.out.println("This is parent class");
} }
class child extends parent {
void childdisplay() {
System.out.println("This is child class");
} }
public class P17 {
public static void main(String[] args) {
parent p =  new parent();
p.display();
child c = new child();
c.childdisplay();
System.out.println("ID :23DCS078_KRISH PATEL");
} }
```

OUTPUT:



```
This is parent class
This is child class
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

This Java code will exemplify elementary inheritance. Here, the child class is inherited from the parent class. The method display() of the parent class is accessible to the objects of the child class.

18 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:

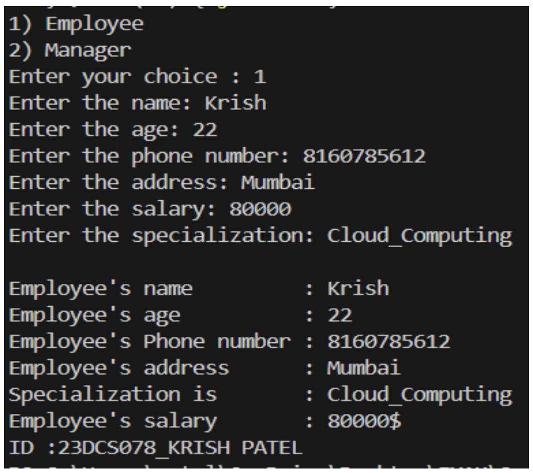
```
import java.util.Scanner;

class Member {

String name;

int age;

long phone_number;

String address;

int salary;

void getData(Scanner sc) {

System.out.print("Enter the name: ");
name = sc.next();
System.out.print("Enter the age: ");
age = sc.nextInt();
System.out.print("Enter the phone number: ");
phone_number = sc.nextLong();
sc.nextLine();
System.out.print("Enter the address: ");
address = sc.nextLine();
System.out.print("Enter the salary: ");
salary = sc.nextInt();
}
void putData() {
System.out.println("Employee's name         : " + name);
System.out.println("Employee's age          : " + age);
System.out.println("Employee's Phone number : " + phone_number);
System.out.println("Employee's address      : " + address);
}
void printSalary() {
System.out.println("Employee's salary       : " + salary + "$" );
} }
class Employee extends Member {
```

```
String specialization;

@Override

void getData(Scanner sc) {

super.getData(sc);

System.out.print("Enter the specialization: ");

specialization = sc.next();

}

@Override

void putData() {

super.putData();

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

} }

class Manager extends Member {

String department;

@Override

void getData(Scanner sc) {

super.getData(sc);

System.out.print("Enter the department: ");

department = sc.next();

}

@Override
```

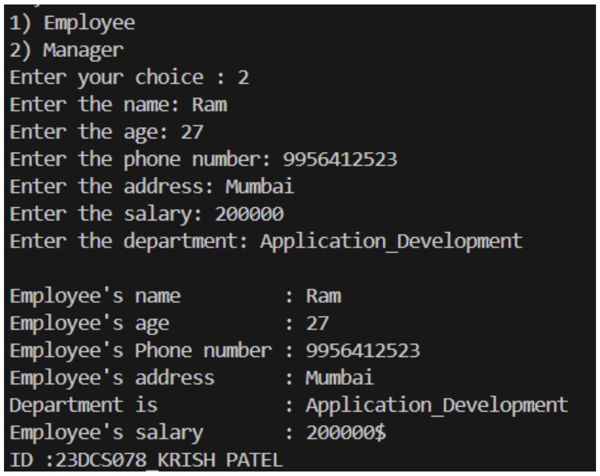
```
void putData() {

super.putData();

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

} }

public class P18 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("1) Employee \n2) Manager");

System.out.print("Enter your choice : ");

int n = sc.nextInt();

switch (n) {

case 1:

Employee e1 = new Employee();

e1.getData(sc);

System.out.println();

e1.putData();

e1.printSalary();

break;

case 2:

Manager m1 = new Manager();

m1.getData(sc);
```

```
System.out.println();
m1.putData();
m1.printSalary();
break;
default:
System.out.println("Invalid choice.");
break;
}
System.out.println("ID :23DCS078_KRISH PATEL");
sc.close();
} }
```

OUTPUT:



```
1) Employee
2) Manager
Enter your choice : 1
Enter the name: Krish
Enter the age: 22
Enter the phone number: 8160785612
Enter the address: Mumbai
Enter the salary: 80000
Enter the specialization: Cloud_Computing

Employee's name         : Krish
Employee's age          : 22
Employee's Phone number : 8160785612
Employee's address      : Mumbai
Specialization is       : Cloud_Computing
Employee's salary       : 80000$
ID :23DCS078_KRISH PATEL
```



```
1) Employee
2) Manager
Enter your choice : 2
Enter the name: Ram
Enter the age: 27
Enter the phone number: 9956412523
Enter the address: Mumbai
Enter the salary: 200000
Enter the department: Application_Development

Employee's name         : Ram
Employee's age          : 27
Employee's Phone number : 9956412523
Employee's address      : Mumbai
Department is           : Application_Development
Employee's salary       : 200000$
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

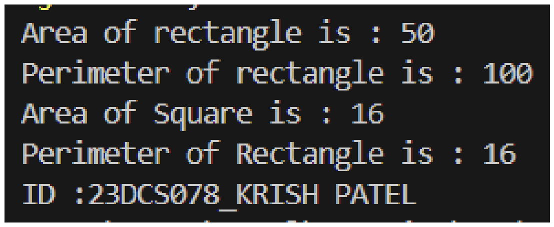
This code shows Java inheritance, in which an Employee and a Manager class inherit the basic Member class and all the features of that base class. The information that would be recorded and displayed in this case is characteristic of employees and managers specifically—a specialization of information to show department information.

19 Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square. Also use array of objects.

PROGRAM CODE:

```
class Rectangle {
int length;
int breadth;
public Rectangle (int length , int breadth) {
this.length = length;
this.breadth = breadth;
}
void printArea() {
System.out.println("Area of rectangle is : " + length*breadth);
}
void printPerimeter() {
System.out.println("Perimeter of rectangle is : " + 2*(length*breadth));
} }
class Square extends Rectangle {
public Square(int side)  {
super(side,side);
}
@Override
void printArea() {
System.out.println("Area of Square is : " + length*length);
}
@Override
void printPerimeter() {
System.out.println("Perimeter of Rectangle is : " + 4*(length));
} }
public class P19 {
public static void main(String[] args) {
Rectangle[] shapes = new Rectangle[2];
shapes[0] = new Rectangle(5, 10);
shapes[1] = new Square(4);
shapes[0].printArea();
shapes[0].printPerimeter();
shapes[1].printArea();
shapes[1].printPerimeter();
System.out.println("ID :23DCS078_KRISH PATEL");
}
}
```

OUTPUT:



```
Area of rectangle is : 50
Perimeter of rectangle is : 100
Area of Square is : 16
Perimeter of Rectangle is : 16
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

This Java program illustrates inheritance with method overriding. There is a Square class extending a Rectangle class. It contains methods to compute the areas and perimeters for both a rectangle and a square.

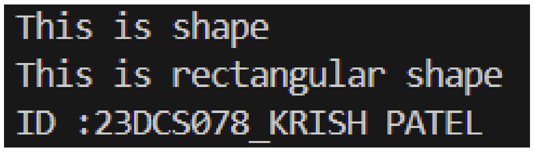
20 Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' 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:

```
class Shape {

void printShape() {

System.out.println("This is shape");

} }

class Rectangle extends Shape {

void printRectangle() {

System.out.println("This is rectangular shape");

} }

class Circle extends Shape {

void printCircle() {
```

```
System.out.println("This is circular shape");
} }
class Square extends Rectangle {
void printSquare() {
System.out.println("Square is a rectangle");
} }
public class P20 {
public static void main(String[] args) {
Square square = new Square();
square.printShape();
square.printRectangle();
System.out.println("ID :23DCS078_KRISH PATEL");
} }
```

OUTPUT:



```
This is shape
This is rectangular shape
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

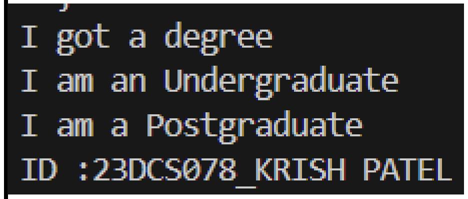
This Java code illustrates multilevel inheritance. In this case, class Square inherits from Rectangle itself inheriting from Shape. The instance of Square can access methods of both its parent and grandparent classes. That means through inheritance, a class can support and expand functionality from several levels up its hierarchy. The program also prints a message indicating the relationship between squares and rectangles.

21 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:

```
class Degree {
void getDegree() {
System.out.println("I got a degree");
} }
class Undergraduate extends Degree {
@Override
void getDegree() {
System.out.println("I am an Undergraduate");
} }
class Postgraduate extends Degree {
@Override
void getDegree() {
System.out.println("I am a Postgraduate");
} }
public class P21 {
public static void main(String[] args) {
Degree degree = new Degree();
degree.getDegree();

Undergraduate undergraduate = new Undergraduate();

undergraduate.getDegree();

Postgraduate postgraduate = new Postgraduate();

postgraduate.getDegree();

System.out.println("ID :23DCS078_KRISH PATEL");

} }
```

OUTPUT:



```
I got a degree
I am an Undergraduate
I am a Postgraduate
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

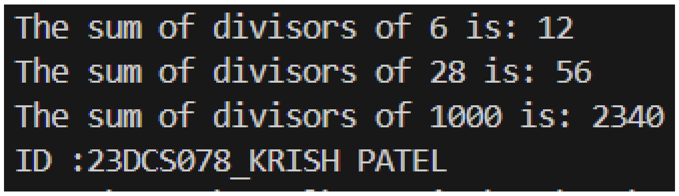
This Java code is an example of how method overriding is done. In the example, there are two classes, Undergraduate and Postgraduate, which override the getDegree() method of the Degree class to print class-specific messages. Now, the program is creating objects of each of the derived classes and then calling their respective getDegree() methods.

22 Write a java that implements an interface AdvancedArithmetic which contains amethod signature int divisor_sum(int n). You need to write a class calledMyCalculator 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:

```
interface AdvancedArithmetic {

int divisor_sum(int n);
}
class MyCalculator implements AdvancedArithmetic {
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 P22 {
public static void main(String[] args) {
MyCalculator myCalculator = new MyCalculator();
int n = 6;
System.out.println("The sum of divisors of " + n + " is: " + myCalculator.divisor_sum(n));
n = 28;
System.out.println("The sum of divisors of " + n + " is: " + myCalculator.divisor_sum(n));
n = 1000;
System.out.println("The sum of divisors of " + n + " is: " + myCalculator.divisor_sum(n));
System.out.println("ID :23DCS078_KRISH PATEL");
} }
```

OUTPUT:



```
The sum of divisors of 6 is: 12
The sum of divisors of 28 is: 56
The sum of divisors of 1000 is: 2340
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

This is an example of how you would use an interface in Java, whereby class MyCalculator implements the AdvancedArithmetic interface. I define a divisor_sum method that calculates and returns the sum of all divisors of any given integer. Later on, I call and print the sum of divisors for several numbers to demonstrate how practical interfaces are in Java.

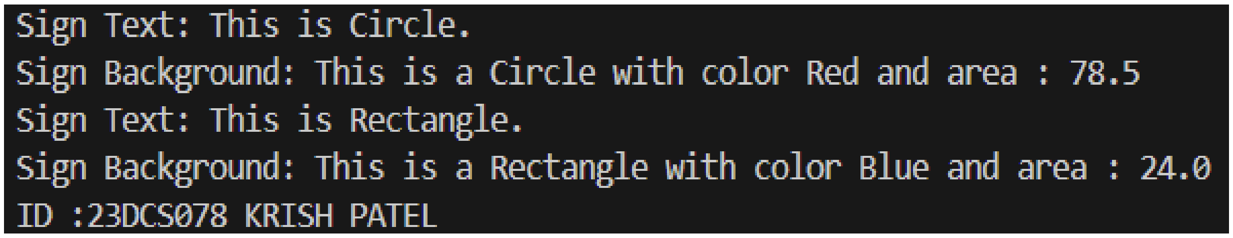
23 Assume you want to capture shapes, which can be either circles (with a radius and a color) or rectangles (with a length, width, and color). You also want to be able to create signs (to post in the campus center, for example), each of which has a shape (for the background of the sign) and the text (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:

```
interface Shape {
String getColor();
double getArea();
String getShape();
default String get
Description() {
return "This is a "+ getShape() + " with color " + getColor() + " and area : " + getArea();
} }
class Circle implements Shape {
```

```
double radius;

String color;

public Circle(double radius, String color) {

this.radius = radius;

this.color = color;

}

public String getColor() {

return color;

}

public double getArea() {

return 3.14 * radius * radius;

}

public String getShape() {

return "Circle";

} }

class Rectangle implements Shape {

double length;

double width;

String color;

public Rectangle(double length, double width, String color) {

this.length = length;
this.width = width;
this.color = color;
}
public String getColor() {
return color;
}
public double getArea() {
return length * width;
}
public  String getShape() {
return "Rectangle";
} }
class Sign {
Shape shape;
String text;
public Sign(Shape shape, String text) {
this.shape = shape;
this.text = text;
}
public void display() {
System.out.println("Sign Text: " + text);
System.out.println("Sign Background: " + shape.getDescription());

} }

public class P23{

public static void main(String[] args) {

Circle circle = new Circle(5.0, "Red");

Rectangle rectangle = new Rectangle(4.0, 6.0, "Blue");

Sign circleSign = new Sign(circle, "This is Circle.");

Sign rectangleSign = new Sign(rectangle, "This is Rectangle.");

circleSign.display();

rectangleSign.display();

System.out.println("ID :23DCS078_KRISH PATEL");

} }
```

OUTPUT:



```
Sign Text: This is Circle.
Sign Background: This is a Circle with color Red and area : 78.5
Sign Text: This is Rectangle.
Sign Background: This is a Rectangle with color Blue and area : 24.0
ID :23DCS078 KRISH PATEL
```

CONCLUSION:

The next code defines the use of interfaces and default methods in Java. In this example, the Shape interface is implemented by Circle and Rectangle classes. Both provide an implementation to getColor, getShape and getArea, but a default implementation to getDescription combines these to describe the shape. A Sign class then uses these shapes to display signs with specified text and background details.

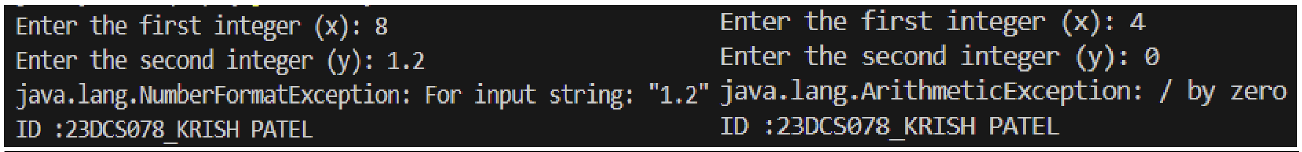
PART-V Exception Handling

24 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 P24 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

try {

System.out.print("Enter the first integer (x): ");

int x = Integer.parseInt(sc.nextLine()); // Parse integer input

System.out.print("Enter the second integer (y): ");

int y = Integer.parseInt(sc.nextLine()); // Parse integer input

int result = x / y;

System.out.println("Result of " + x + " / " + y + " = " + result);

}

catch (NumberFormatException e) {

System.out.println(e);

}

catch (ArithmeticException e) {

System.out.println(e);

}
System.out.println("ID :23DCS078_KRISH PATEL");
sc.close();
} }
```

OUTPUT:



```
Enter the first integer (x): 8
Enter the second integer (y): 1.2
java.lang.NumberFormatException: For input string: "1.2"
ID :23DCS078_KRISH PATEL
```

```
Enter the first integer (x): 4
Enter the second integer (y): 0
java.lang.ArithmeticException: / by zero
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

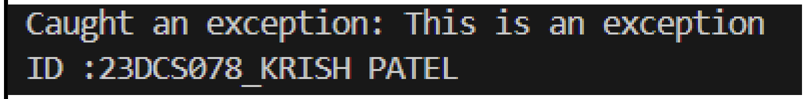
This program handles user input for two integers and performs division while using exception handling to catch invalid input (`NumberFormatException`) and division by zero (`ArithmeticException`). It ensures the program doesn't crash due to errors and prints the result or the exception message. Finally, it closes the scanner and displays the student ID.

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

PROGRAM CODE:

```
public class P25 {
public static void main(String[] args) {
try {
throw new Exception("This is an exception");
} catch (Exception e) {
System.out.println("Caught an exception: " + e.getMessage());
System.out.println("ID :23DCS078_KRISH PATEL");
}
}}
```

OUTPUT:



```
Caught an exception: This is an exception
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

In this program an exception is deliberately created by use of the keyword throw and then all lines after this statement in the catch block catch it. Here is how the Java programming language is made useful in handling runtime errors.

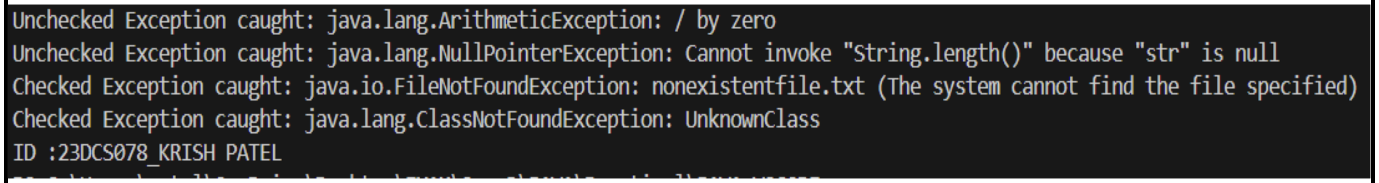
26 Write a java program to generate user defined exception using “throw” and “throws” keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).

PROGRAM CODE:

```
import java.io.*;

public class P26 {

public static void readFile(String filePath) throws IOException {

FileReader file = new FileReader(filePath);

BufferedReader br = new BufferedReader(file);

System.out.println(br.readLine());

br.close();

}

public static void main(String[] args) {

try {

int a = 10 / 0;

} catch (ArithmeticException e) {
```

```
System.out.println("Unchecked Exception caught: " + e);

}

try {

String str = null;

System.out.println(str.length());

} catch (NullPointerException e) {

System.out.println("Unchecked Exception caught: " + e);

}

try {

readFile("nonexistentfile.txt");

} catch (IOException e) {

System.out.println("Checked Exception caught: " + e);

}

try {

Class.forName("UnknownClass");

} catch (ClassNotFoundException e) {

System.out.println("Checked Exception caught: " + e);

}

System.out.println("ID :23DCS078_KRISH PATEL");

} }
```

OUTPUT:



```
Unchecked Exception caught: java.lang.ArithmeticException: / by zero
Unchecked Exception caught: java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is null
Checked Exception caught: java.io.FileNotFoundException: nonexistentfile.txt (The system cannot find the file specified)
Checked Exception caught: java.lang.ClassNotFoundException: UnknownClass
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

This Java program manages unchecked as well as checked exceptions. Arithmetic, null pointer file not found, and class not found exceptions are caught using the try-catch blocks, therefore the program runs quite seamlessly. Exception handling is quite important in building robust applications that can handle errors with grace.

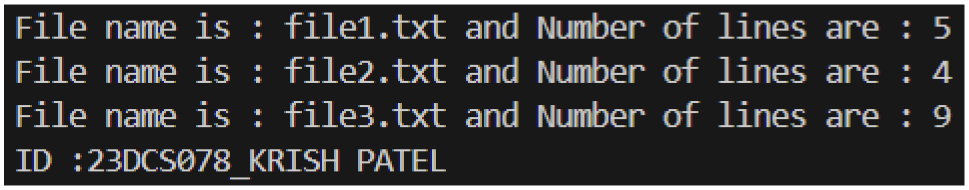
PART-VI File Handling & Streams

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

PROGRAM CODE:

```
import java.io.*;

public class P27 {

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

if (args.length == 0) {

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

} else {

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

try {

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

String j;

int count = 0;

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

count++;

}

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

System.out.println(e);

} } }

System.out.println("ID :23DCS078_KRISH PATEL");

} }
```

OUTPUT:



```
File name is : file1.txt and Number of lines are : 5
File name is : file2.txt and Number of lines are : 4
File name is : file3.txt and Number of lines are : 9
ID :23DCS078_KRISH PATEL
```

CONCLUSION:

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

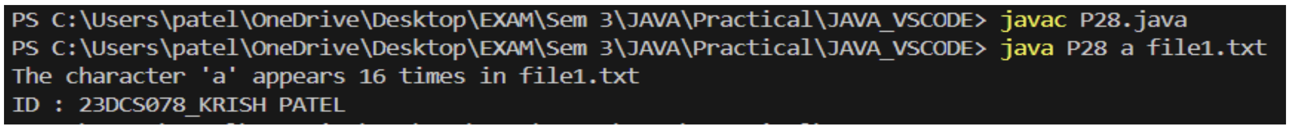
28 Write an example that counts the number of times a particular character, such as e, appears in a file. The character can be specified at the command line. You can use xanadu.txt as the input file.

PROGRAM CODE:

```
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class P28{

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

```
if (args.length < 2) {
System.out.println("Usage: java P28 <character> <filename>");
return; }
char targetChar = args[0].charAt(0);
String fileName = args[1];
int count = 0;
try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
int ch;
while ((ch = reader.read()) != -1) {
if (ch == targetChar) {
count++;
} }
System.out.println("The character '" + targetChar + "' appears " + count + " times in " + fileName);
} catch (IOException e) {
System.out.println("Error reading " + fileName + ": " + e.getMessage());
}
System.out.println("ID : 23DCS078_KRISH PATEL");
}}
```

OUTPUT:



```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P28.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P28 a file1.txt
The character 'a' appears 16 times in file1.txt
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

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

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

PROGRAM CODE:

```
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class P29 {

public static void main(String[] args) {

if (args.length < 2) {

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

return;

}

String searchWord = args[0];

String fileName = args[1];

Integer count = 0;

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

String line;

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

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

for (String word : words) {

if (word.equalsIgnoreCase(searchWord)) {

count++;

} } }

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

} catch (IOException e) {

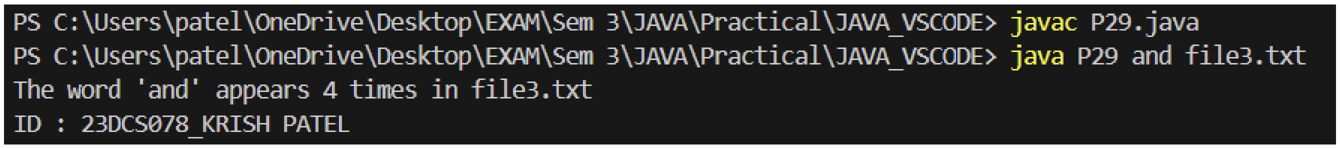
System.out.println("Error reading " + fileName + ": " + e.getMessage());

}

System.out.println("ID : 23DCS078_KRISH PATEL");

} }

OUTPUT:



```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P29.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P29 and file3.txt
The word 'and' appears 4 times in file3.txt
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

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

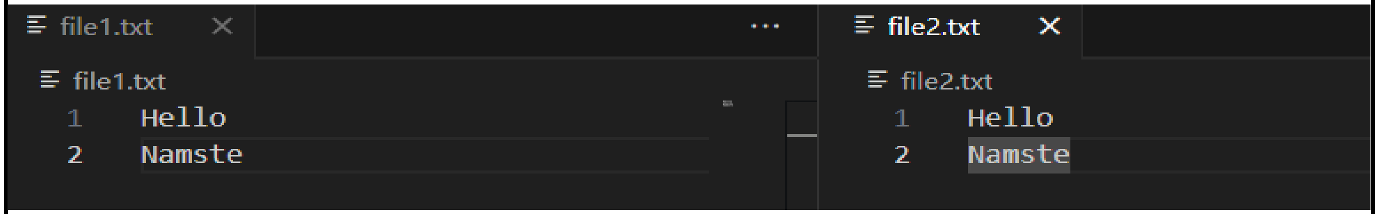
30 Write a program to copy data from one file to another file. If the destination file does not exist, it is created automatically.

PROGRAM CODE:

import java.io.FileReader;

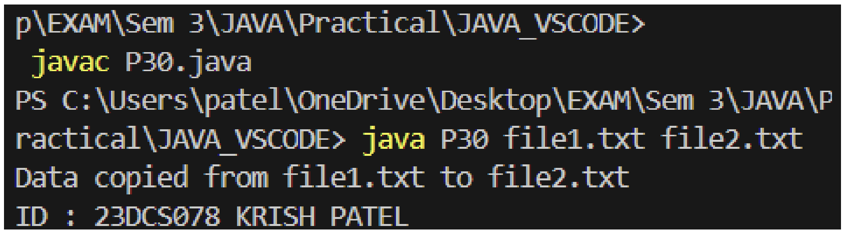
```
import java.io.FileWriter;
import java.io.IOException;
public class P30 {
public static void main(String[] args) {
if (args.length < 2) {
System.out.println("Usage: java P30 <source file> <destination file>");
return;
}
String sourceFile = args[0];
String destinationFile = args[1];
try (FileReader fr = new FileReader(sourceFile);
FileWriter fw = new FileWriter(destinationFile)) {
int ch;
while ((ch = fr.read()) != -1) {
fw.write(ch); }
System.out.println("Data copied from " + sourceFile + " to " + destinationFile);
} catch (IOException e) {
System.out.println("Error: " + e.getMessage());
}
System.out.println("ID : 23DCS078_KRISH PATEL");
} }
```

OUTPUT:



```
file1.txt
1   Hello
2   Namste
```

```
file2.txt
1   Hello
2   Namste
```



```
p\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE>
 javac P30.java
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\P
ractical\JAVA_VSCODE> java P30 file1.txt file2.txt
Data copied from file1.txt to file2.txt
ID : 23DCS078 KRISH PATEL
```

CONCLUSION:

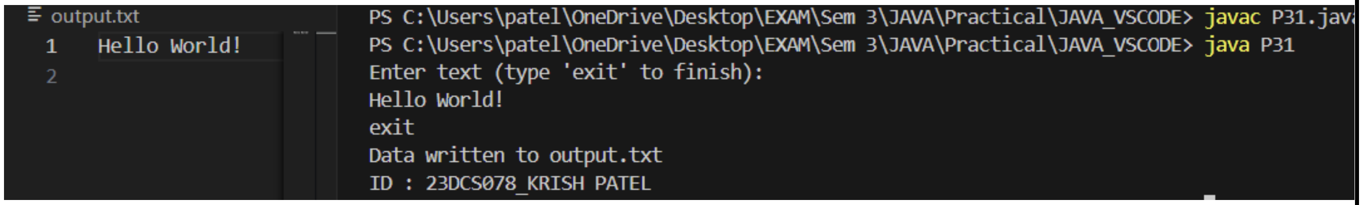
This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance.

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

PROGRAM CODE:

```
import java.io.*;

public class P31 {

public static void main(String[] args) {

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

String fileName = "output.txt";

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

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

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

fileWriter.write(input);

fileWriter.newLine();

}

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

} catch (IOException e) {

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

}

System.out.println("ID : 23DCS078_KRISH PATEL");

} }
```

OUTPUT:



```
output.txt
Hello World!

```

```
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> javac P31.jav
PS C:\Users\patel\OneDrive\Desktop\EXAM\Sem 3\JAVA\Practical\JAVA_VSCODE> java P31
Enter text (type 'exit' to finish):
Hello World!
exit
Data written to output.txt
ID : 23DCS078_KRISH PATEL
```

CONCLUSION:

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

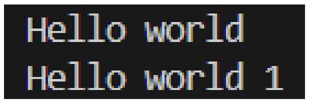
PART-VII Multithreading

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

PROGRAM CODE:

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

public void run(){

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

}}

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

public void run(){

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

} }

public class P32 {

public static void main(String[] args) {

Thread1 thread = new Thread1();

thread.start();

Thread2 obj2 = new Thread2();

Thread t1 = new Thread(obj2);

t1.start();

} }
```

OUTPUT:



```
Hello world
Hello world 1
```

CONCLUSION:

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

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

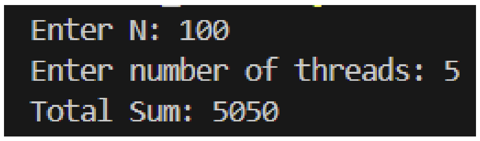
PROGRAM CODE:

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

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

```
}
for (Thread thread : threads) {
try {
thread.join();
} catch (InterruptedException e) {
e.printStackTrace();
} }
System.out.println("Total Sum: " + SumTask.getTotalSum());
} }
```

OUTPUT:



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

CONCLUSION:

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

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

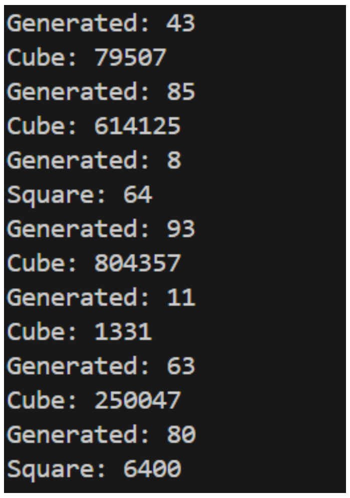
PROGRAM CODE:

```
import java.util.Random;
class RandomNumberGenerator extends Thread {
private final Object lock;
```

```
public RandomNumberGenerator(Object lock) {
this.lock = lock;
}
public void run() {
Random random = new Random();
while (true) {
int number = random.nextInt(100);
synchronized (lock) {
P34.lastNumber = number;
lock.notifyAll();
System.out.println("Generated: " + number);
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
} } } } }
class EvenNumberProcessor extends Thread {
private final Object lock;
public EvenNumberProcessor(Object lock) {
this.lock = lock;
}
public void run() {
while (true) {
```

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

OUTPUT:



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

CONCLUSION:

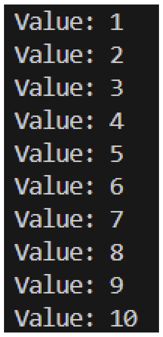
This program effectively demonstrates a multi-threaded application where one thread generates random integers, while two other threads process these integers based on their parity. It highlights the use of synchronization in Java to safely share data among threads, showcasing how concurrency can be leveraged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

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

PROGRAM CODE:

```
public class P35 extends Thread {

private int value = 0;

public void run() {

while (true) {

value++;

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

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

} } }

public static void main(String[] args) {

P35 incrementer = new P35();

incrementer.start();

}

}
```

OUTPUT:



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

CONCLUSION:

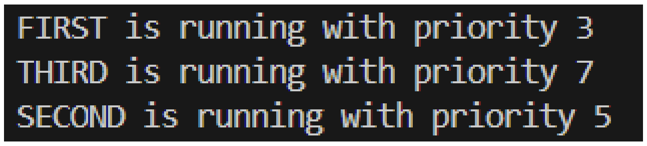
This program effectively demonstrates the use of a thread to increment a variable every second. It utilizes the sleep() method to create a delay between increments, showcasing basic thread functionality in Java.

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

PROGRAM CODE:

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

first.start();

second.setPriority(Thread.NORM_PRIORITY);

second.start();

third.setPriority(7);

third.start();

} }
```

OUTPUT:



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

CONCLUSION:

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

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

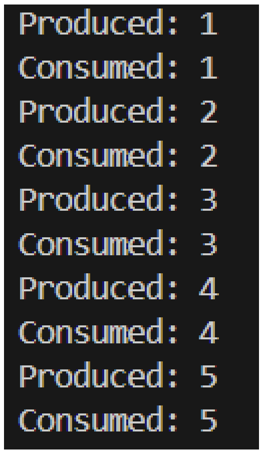
PROGRAM CODE:

```
class Buffer {

private int data;

private boolean isEmpty = true;

public synchronized void produce(int value) {

while (!isEmpty) {

try {

wait();
} catch (InterruptedException e) {
e.printStackTrace();
} }
data = value;
isEmpty = false;
System.out.println("Produced: " + data);
notify();
}
public synchronized void consume() {
while (isEmpty) {
try {
wait();
} catch (InterruptedException e) {
e.printStackTrace();
} }
System.out.println("Consumed: " + data);
isEmpty = true;
notify();
} }
class Producer extends Thread {
private Buffer buffer;
public Producer(Buffer buffer) {
this.buffer = buffer;
```

```
}
public void run() {
for (int i = 1; i <= 5; i++) {
buffer.produce(i);  // Produce values from 1 to 5
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
} } } }
class Consumer extends Thread {
private Buffer buffer;
public Consumer(Buffer buffer) {
this.buffer = buffer;
}
public void run() {
for (int i = 1; i <= 5; i++) {
buffer.consume();
try {
Thread.sleep(1500);
} catch (InterruptedException e) {
e.printStackTrace();
} } } }
public class P37 {
```

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

Buffer buffer = new Buffer();

Producer producer = new Producer(buffer);

Consumer consumer = new Consumer(buffer);

producer.start();

consumer.start();

} }
```

OUTPUT:



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

CONCLUSION:

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

PART-VIII Collection Framework and Generic

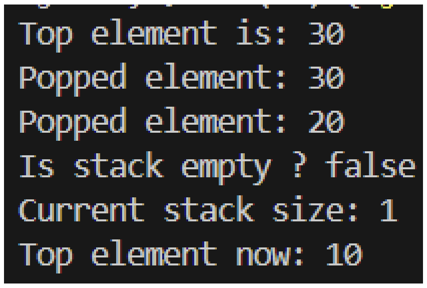
38 Design a Custom Stack using ArrayList class, which implements following functionalities of stack.
My Stack -list ArrayList<Object>: A list to store elements.
isEmpty: boolean: Returns true if this stack is empty.
getSize(): int: Returns number of elements in this stack.
peek(): Object: Returns top element in this stack without removing it.
pop(): Object: Returns and Removes the top elements in this stack.
push(o: object): Adds new element to the top of this stack.

PROGRAM CODE:

```
import java.util.ArrayList;

class MyStack {

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

public boolean isEmpty() {

return list.isEmpty();

}

public int getSize() {

return list.size();

}

public Object peek() {

if (isEmpty()) {

return "Stack is empty";

}

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

```
}
public Object pop() {
if (isEmpty()) {
return "Stack is empty";
}
return list.remove(list.size() - 1);
}
public void push(Object o) {
list.add(o);
} }
public class P38 {
public static void main(String[] args) {
MyStack stack = new MyStack();
stack.push(10);
stack.push(20);
stack.push(30);
System.out.println("Top element is: " + stack.peek());
System.out.println("Popped element: " + stack.pop());
System.out.println("Popped element: " + stack.pop());
System.out.println("Is stack empty ? " + stack.isEmpty());
System.out.println("Current stack size: " + stack.getSize());
System.out.println("Top element now: " + stack.peek());

} }
```

OUTPUT:



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

CONCLUSION:

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

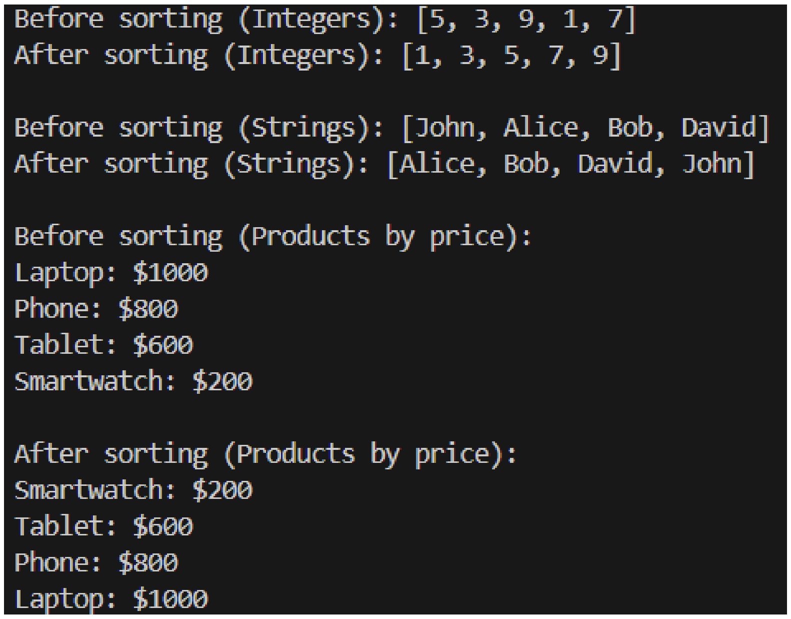
39 Imagine you are developing an e-commerce application. The platform needs to sort lists of products based on different criteria, such as price, rating, or name. Each product object implements the Comparable interface to define the natural ordering. To ensure flexibility and reusability, you need a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

PROGRAM CODE:

```
import java.util.Arrays;

public class P39 {

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

Arrays.sort(array);

}

public static void main(String[] args) {

Integer[] numbers = {5, 3, 9, 1, 7};
System.out.println("Before sorting (Integers): " + Arrays.toString(numbers));
sortArray(numbers);
System.out.println("After sorting (Integers): " + Arrays.toString(numbers));
String[] names = {"John", "Alice", "Bob", "David"};
System.out.println("\nBefore sorting (Strings): " + Arrays.toString(names));
sortArray(names);
System.out.println("After sorting (Strings): " + Arrays.toString(names));
Product[] products = {
new Product("Laptop", 1000),
new Product("Phone", 800),
new Product("Tablet", 600),
new Product("Smartwatch", 200)
};
System.out.println("\nBefore sorting (Products by price): ");
for (Product p : products) {
System.out.println(p);
}
sortArray(products);
System.out.println("\nAfter sorting (Products by price): ");
for (Product p : products) {
System.out.println(p);
} } }
class Product implements Comparable<Product> {
```

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

OUTPUT:



```
Before sorting (Integers): [5, 3, 9, 1, 7]
After sorting (Integers): [1, 3, 5, 7, 9]

Before sorting (Strings): [John, Alice, Bob, David]
After sorting (Strings): [Alice, Bob, David, John]

Before sorting (Products by price):
Laptop: $1000
Phone: $800
Tablet: $600
Smartwatch: $200

After sorting (Products by price):
Smartwatch: $200
Tablet: $600
Phone: $800
Laptop: $1000
```

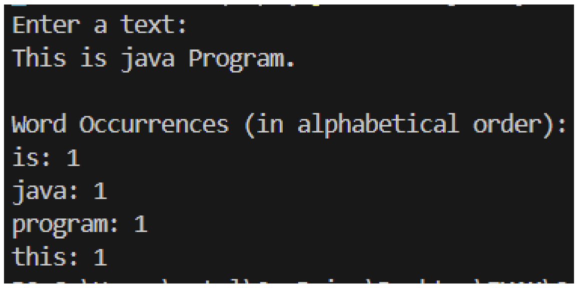
CONCLUSION:

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

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

PROGRAM CODE:

```
import java.util.*;

public class P40 {

public static void main(String[] args) {

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

Scanner scanner = new Scanner(System.in);

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

String text = scanner.nextLine();

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

for (String word : words) {

if (!word.isEmpty()) {

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

} }

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

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

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

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

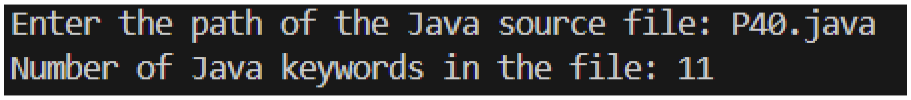
	} } } **OUTPUT:**  Enter a text: This is java Program. Word Occurrences (in alphabetical order): is: 1 java: 1 program: 1 this: 1 **CONCLUSION:** This program demonstrates how to count and display the occurrences of words in a given text using Java's Map and Set classes. The words are stored in a TreeMap, ensuring that they are presented in alphabetical order. The use of getOrDefault() simplifies the counting process, showcasing efficient word frequency analysis.

41	Write a code which counts the number of the keywords in a Java source file. Store all the keywords in a HashSet and use the contains () method to test if a word is in the keyword set.
	
	 PROGRAM CODE:
	
	 import java.io.*;
	
	 import java.util.*;
	
	 public class P41 {
	
	 private static final HashSet<String> keywords = new HashSet<>();
	
	 static {
	
	 String[] keywordArray = {
	
	 "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class",
	
	 "const", "continue", "default", "do", "double", "else", "enum", "extends", "final",
	
	 "finally", "float", "for", "goto", "if", "implements", "import", "instanceof", "int",
	
	 "interface", "long", "native", "new", "package", "private", "protected", "public",

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

} } }

OUTPUT:



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

CONCLUSION:

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