raisoni EDUCATION

Experiment - 01

Aim:
To write a Java Program that calculates the sum and average of a set of numbers using loops.

Objective:
To develop a Java program that calculates the sum and average of a set of numbers efficiently, allowing the user to input the total number of elements and the respective values. This will demonstrate the use of loops for iterative processing and basic arithmetic operations.

Theory:
In Java, loops like for, while, and do-while are used for repetitive operations. The program will utilize a for loop to iterate through the user inputs. The sum of the numbers is accumulated in a variable, and the average is calculated by dividing the sum by the count of numbers (n). Input handling and result display are achieved using the scanner class and System out for output.

Algorithm:
1. Start the program.
2. Input the number of elements (n) from the user.
3. Initialize.
   i) sum = 0
   ii) average = 0.

raisoni
EDUCATION

Code.

```
import java.util.Scanner;
public class SumAverages
    public static void main (String args[])
  {
    Scanner scanner= new scanner(System.in);
    System.out.println("Enter the number of
      elements");
    int n = scanner.nextint();
    int sum=0,
    system.out.println("Enter"+n" num");
      for (int i=0, i<n, i++)
       {
       int num= scanner.neatInt();
         sum+ num;
  }
  double average= (double) sum/5;
  System.out.println("sum="+sum);
system.out.println("Average= "+Averay);
   scanner.close();
  }
}
```

Output:

Enter the number OF elecm-ent is
Enter 5 numbers
10, 20, 30, 40, 50
Sum=150
Avg - 30.0.

5. After the loop, calculate average = sum / n.
6. Output the sum and average.
7. End the program.

Conclusion:

This program successfully calculates the sum and average of a set of numbers entered by the user using a loop. By employing loops, the program efficiently processes multiple inputs, performs calculations, and outputs the results. The logic is dynamic, allowing any number of inputs as specified by the user.

raisoni EDUCATION

Assignment -02

Title:
Basic calculator using switch-case in Java.

Objective:
To implement a simple calculator program in Java using the switch-case control statement that performs basic arithmetic operations based on user input.

Theory:
Switch-case statement:
The switch statement in Java allows the execution of different blocks of code based on the value of an expression. The case lables define the possible values, and the break statement ensures that only one case is executed.

Calculator Operations:
A calculator takes two operands and an operator to perform a computation. Common operations include:
1. Addition (+)
2. Substraction (-)
3. Multiplication (*)
4. Division (/)

Error Handling:
Division by zero is checked to prevent runtime errors.
Invalid operators are handled using the default.

Code:

```
import java.util.scanner;
  public class calculator{
  public static void main (String args[]){
    scanner scanner = new scanner(system.in);

System.out.println ("Enter first number");
  double num1 = scanner.nextdouble ();
  so
system.out.println("Enter first number);
  double num1 = scanner.next double();

System.out.println("Choose operator+,-,*
  char operator = scanner.next()
  char+; double result;

  switch (operator){
   case'+';
   result = num1 + num2.
System.out.println (Result" + result);
        break;
       case'-';
      result = num1 - num2;
system.out.println ("+" result);
       break.
       case '*';
       result = num1 * num2;
System.out.println (" + result);
            break;
```

```
System.out.println("Error! Division
    by zero is not allowed");
                }
                break;
            default:
System.out.println("Invalid operator
    please choose +, -, * or /");
                }
            scanner close();
        }
    }
```

Output: first no. = 10
enter second no. = 5
choose operation = +, -, *, /
result is 0.

Algorithm:

1. Start the program.
2. Input two numbers and an operator.
3. Use a switch statement to handle operations based on the operator.

   + : Perform addition
   - : Perform substraction
   * : Perform multiplication.
   / : Check if the second number is not zero, then perform division.
   default: Print an error message for invalid operators.
4. Output the result of the operation.
5. End the program.

Conclusion:

The program demonstrates the use of the switch-case statement in Java to implement a basic calculator. It efficiently handles different arithmetic operations and includes error checking for invalid inputs, such as unsupported operators or division by zero. This approach highlights the importance of control flow structures in Java programming.

raisoni EDUCATION

Assignment -08.

Title:
Develop a program to create and initialize a class representing a student with attributes such as name, roll number, and marks.

Objective:
To create a class representing a student in Java and initialize it with attributes such as name, roll number, and marks. The program will also demonstrate how to instantiate objects and display the student's information.

Theory:
A class in Java serves as a blueprint for creating objects. Each object created from a class will have its own set of attributes and methods. In this case, we will create a student class with three attribute name, roll number, and marks. These attributes will be initialized using a constructor, and their values will be displayed through a method.
We will use the constructor __init__ to initialize these attributes when a new instance of the class is created.

Algorithm:
1. Define a class named student.
2. Declare instance variables for the student's name, roll number, and marks.

3. Create a constructor to initialize the values of these attributes.
4. Define a method to display the student's information.
5. In the main method, create an instance of the student class and initialize the student's detail.
6. Call the method to display the student's detail.

Explanation:
- Class Definition: We define a class named student which contains the attributes name, rollNumber, and marks as instance variable
- Constructor: The constructor is used to initialize these attributes when a student object is created. The constructor parameters take values that will be assigned to the instance variables.
- Method to Display Details: The displayDetails() method is used to print out the student's name, roll number, and marks
- Main Method: The main method creates a student object, initializes it with specific values and calls the displayDetails() method to print the student's information.

Code:

```
import java.util.scanner;
   class student {
    string name;
    int roll number;
    double marks;
   student (string name, int roll number
     double marks) {
     this.name: name;
     this.roll number = roll number;
      this.marks = marks;
 }
 void display-Details() {
  System.out.println("\n student Details)
System.out.println ("Name:" +name);
System.out.println ("Roll no": + Roll no.);
system.out.println ("Marks" +marks);
  }
}
public class student Demo {
    public static void main (String args[])
{ Scanner s = new Scan.in);
System.out.println ("Enter student
 name");
  String Name = scanner.nextline();

System.out.println ("Enter Roll no:");
 int roll number = scanner.nextInt();
System.out println ("Enter marks");
   double marks = scanner.nextdouble;
```

```
rollno, marks);
    student.displayDetails();
        scanner.close();
    }
}
```

Example Output:

Enter student name: John Doe
enter ROLL No. 101
Enter marks: 85.5

Conclusion:

The program illustrates how to define a class in Java, initialize its attributes through a constructor, and display the information using a method. This approach helps manage related data and provides clear structure in object-oriented programming.

raisoni EDUCATION

Assignment - 04

Title:
Write a program to demonstrate method overloading and the use of this keyword.

Objective:
1. Demonstrate method overloading, where multiple methods with the same name but different parameter lists are defined.
2. Demonstrate the usage of this keyword, which can be used to refer to the current instance of the class or to differentiate between class attributes and parameters when they have the same name

Theory:
Method overloading
1. Method overlodding is a feature in Java where you can define multiple methods with the same name but with different signatures
2. Java determines which method to call based on the number and type of arguments passed.

this keyword.

This keyword is a reference variable that refers to the current instance of the class. It can be used to.
1. Access instance variable when the parameter names are the same as the instance variable names.

2. Call another constructor of the class.
3. Refer to the current object.

Algorithm:
1. Define a class named calculator.
2. Define overloaded methods to perform addition, with each method accepting different numbers or types of parameters
3. Use the this keyword to differentiate between instance variables and constructor parameters where necessary.
4. In the main method, create an object of the calculator class and call the overloaded methods.

Explanation:
- Class Definition: The class calculator has instance variables number1 and number2 to store integer values.
- Constructor: The constructor accepts two integers and initializes the instance variables. The this keyword is used to distinguish between the instance variables and the constructor parameters with the same name.
- Method Overloading:
1. The add method is overloaded in three ways:
i-) The first add method takes two integers and returns their sum.
ii-) The second add method takes three integers and returns their sum.
iii-) The third add method takes two doubles.

raisoni
EDUCATION

```
code: class student {
        String name;
        int roll number;
        double marks;

  student (String name, int roll no.
   double marks) {
   this.name = name;
   this.roll no. = roll no.;
   this.marks = marks;
  void display () {
   system.out.println (" \n student detail");
System.out.println (" Roll no" + this.roll );
System.out.println (" Marks: " this.marks);
    }
void display (String message) {
   system.out println (" \n " + message);
system.out.println (" Name: " + this.name
   + " Roll no.: " + this.
    roll no. + " Marks");
  }
  void display (boolean showMarks)
{
system.out.println (" \n student details");
System.out.println (" Roll no. " + this.
      roll no.);
        if (showMarks) {
```

```
    }
}
}

public class methodOverloadingDemo{
  public static void main(String args[])
    Student student = new Student("Alice",
      102, 88.5);
  student.display();
    student.display("Student Record");
      student.display(false);
    }
}
```

Output Student Details
Name: Alice
Roll no. = 102
mark = 88.5

Student Record:
Name: Alice, roll no. = 102
mark = 88.5.

2. This demonstrates method overloading as the method name add is used multiple times but with different parameters.

Method to Display Addition: The method displa Addition uses the add method to display the sum of the instance variables number1 & number 2.

Main Method: The main method.

1. Creates an instance of the calculator class.
2. Demonstrates method overloading by calling add with different types and numbers of arguments.
3. Displays the sum of the instance vari--ables using the display Addition method

conclusion:

This program demonstrates how method overloading allows you to define multiple methods with the same name but different parameter lists, and how the this keyword is used to refer to the current instance of the class. Method overloading enhances the flexibility of a program, and this helps manage instance variables and method parameters efficiently.

raisoni EDUCATION

Experiment - 05

Title:
Implement a Java program to demonstrate single and multilevel inheritance.

Objective:
To understand and implement the concepts of single inheritance and multilevel inheritance in Java by creating a program that demonstrates these principles.

Theory:
Inheritance is a fundamental concept of OOP but that allows a class to derive properties and behaviours from another class.

Types of Inheritance in Java:

Single Inheritance:
1. A subclass inherits from a single superclass
2. Example: Class B extends Class A.

Multilevel Inheritance:
1. A subclass inherits from another subclass. Forming a chain.
2. Example: Class c extends Class B, and Class B extends class A.

Key Concepts:
- extends keyword is used to implement inheritance.
- The child class has access to public and protected members of the parent class.

raisoni
EDUCATION

Code:

```
class Animal {
    void eat (){
    System.out.println("This animal
    eats food");
    }
}
class Dog extends Animal {
    void bark(){
    system.out.println("The dog
    bark");
    }
}
class puppy extends dog {
    void weep ()
    system.out.println("The puppy weeps");
    }
}
public class Inheritance Demo {
    public static void main (String args
    [] ) {
    puppy puppy = new puppy ();
    puppy.eat();
    puppy.bark();
    puppy.weep ();
```

Output: This animal eats Food
The dog breaks
The puppy weep.

raisoni
EDUCATION

Algorithm:

1. Define a Base class with attributes & methods.
2. Create a Derived class that extends the base class
3. create another class that extends the derived class to demonstrate Multilevel Inheritance.
4. In the main method, create objects and call the methods of each class to show inheritance in action.

Explanation:

Class Animal (Base class):
- Defines a method eat()

class Mammal (Derived From Animal - single Inheritance).
- Inherits eat() from Animal.
- Defines bark.

Main Method (InheritanceDemo):

- Create an instance of Dog.
- Calls methods from all three levels to demostrate inheritance.

Conclusion:
- Single and multilevel inheritance allows reusability of code.
- The subclass can access properties and methods

of its parent class.
- Java does not support multiple inheritance to avoid ambiguity.
- This example successfully demonstrates both single and multilevel inheritance to avoid ambiguity.
- This example successfully demonstrates both single and multilevel inheritance in Java.

raisoni
EDUCATION

```
Code: Interface shape{
  double calculateAreas();
}
  class circle implements shapes{
      double radius;
    circle (double, radius){
    this. radius = radius;
  }
public double calculatearea Areas(){
  return math. PI * radius * radius
    }
}
class rectangle implements shape{
   double length, breadth;
   Rectangle (double length, double
   breadth){
       this.length = length;
       this. breadth = breadth;
  }
  public double calculate Areas{
      return length * breadth;
    }
}
public class InterfaceDemo{
  public static void main (String
   args[]){
   shape circle. new circle (5);
System. out. println ("Circle Area:"
+ Circle:calculate Area());

System. out println ("Rectangle.
area:" + rectangle. calculate Areas);
  }
}
```

Experiment -06

Title:
Program to calculate Area of Different Shapes using Interfaces in Java

Objective:
To implement a Java program that demonstrates the use of interfaces to calculate the area of different shapes such as Circle, Rectangle, and Triangle.

Theory:
In Java, an interface is a reference type that acts as a contract for classes. It can contain abstract methods, and any class implementing the interface must provide concrete implementations of these methods. This approach promotes polymorphism and code reusability. In this lab, we will create an interface Shape with a method calculateArea(). The classes Circle, Rectangle, and Triangle will implement this interface and provide their specific implementations for calculating areas.

Algorithm:
1. Start.
2. Define an interface Shape with an abstract method calculateArea().
3. Create classes Circle, Rectangle, and Triangle that implement the Shape interface.
4. In each class, define the required parameters

5. Implement the calculateArea() method in each class using the appropriate formula:
   i-) Circle: Area = π * radius* radius.
   ii-) Rectangle: Area = length * width.
   iii-) Triangle: Area = 0.5 * base * height.
6. In the Main class, create objects for each shape and initialize them with specific values.
7. Call the calculateArea() method for each object and display the results.
8. End.

Explanation:
1. The Shape interface defines the calculateArea method.
2. Each shape class implements the Shape interface and provides its specific formula to calculate the area.
3. In the Main class, objects of each shape are created, and their areas are calculated and displayed.

Conclusion:
This lab demonstrated how to use interfaces in Java to achieve polymorphism. By implementing a common interface, different classes provided their unique implementations while maintaining a consistent method signature.

raisoni EDUCATION

Experiment No.-07

Write a program to demonstrate exception handling on division by zero and invalid input

Aim:

To Illustrate the concept of exception handling in Java by creating a program that performs divisions and handles potential 'Arithmetic exception' due to division by zero, and 'Input m is match exception (due to invalid I/P).

Objective:

1) demonstrates how to use 'try-catch' blocks to handle exceptions.
2) Specifying handle the 'Arithmetic exception that occurs when dividing by zero.
3) Specifically handle the Input mismatch exception, that occurs when the user provides non-integers input.
4) Ensure the program does not demonstrate abruptly when those exception occurs but instead provides informative error messages.
5) Allow the user reply input after an m invalid input exception.

Theory:

Exception handling is a mechanism in Java that allows you to deal with runtime errors in a gragful manner without exception handling when an exception occurs, the

raisoni EDUCATION

program terminates abruptly, exception handling provides a way to ~~cat~~ catch these errors also execute specific code to recevers from t[illegible] inform the use about the issue.

Code:

```
import java.util.scanner
public class exception handling
public static void main (string () args);
scanner sc=new manner (system in);
by d
system. out. print ("Enternumrator:")
int a = scan text Int ();
system. out. print=) ("Enter denominator);
int b = scan text Int ();
int result = a/b;
s.o.p('Result:" & reset;
y catch (Arithmetic exception)e
s.op. ("Error, cannot device by
zero")
y catch (Java. uti). Input m is match
exceptmehc);
sop ("Error: invalid input; please,
print).
integers only");
y Finally;
sc.close);
system-out. printIn (: program Finish
, cal. ");
}
}.
```

Output :-
Enter numerator : 10
Enter denominator = 2
Result : 5
program finished

Key concept:
- Exception: An event that occurs during the execution of a program that develop the normal flow of instructions.
- try block: A block of code where in exception might occur
- catch block: A block e.g code that is expression of a specific type of exception occurs within the corresponding 'try' block.

Conclusion:
This Java program effectively demonstration how to use exception handling to manage potential runtime errors. writing divisions.

raisoni EDUCATION

Experiment No-08

create a program to simulate a banking system with exception handling for invalid transactions.

Aim: To develop a banking system simulation in Java with proper exception handling for invalid transactions.

Theory:
A banking system requires proper errors handling to prevent invalid transactions like withdrawing more than the balance on entering in Java ensure such cases are managed smoothly

Objective:
① To simulate a basic banking system.
② To implement exception handling for invalid withdrawals and deposits.

Code implementation:

```
class InsufficientFundsException {
public
InsufficientFundsException(String args)
{
  super(message);
  }
}
  class BankAccount {
private double balance;
public bankAccount(double balance)
```

```
public bank account (double balance) {
    this.balance= balance}
  }.
public void deposit (double amount)
illegal Argumeat exception {
    if (amount <=0) {;
            throvo ho;
  Illegal Argument exception ('idepast
amount must be +ve");
  balance amount.

public void withdraw
  throws
  InSufficient funds exception ("Insufficien
    balance");
    balance =amount;

public class bankingsystem {
  public static void main = = string
  {
    by {
    Bank account  account = new bank
    Account (1000);
     account. deposit (1000);
     account. withdraw (2000 ;}
      catch (exception) {
        S.op (" Error tage (1);
          }

    }
  },
```

raisoni
EDUCATION

Output
deposited : 500.0
error: Insufficient balance.

Conclusion:
This experiment successfully demonstrate exception handling u's a banking system It ensure deposits are posture and prevents out withdrawing the reliability of financial transactions.

raisoni EDUCATION

Experiment No. 9

Write an applet to display a welcome message and change the background color dynamically

Aim:
To create a Java applet that displays a welcome message acts changes the background color dynamically

Theory: Java applets interactive GUI based programs that run in a web browser or an applet viewer background color dynamically based on user intraction

Objectives
1. To develop a simple Java applet.
2. To dynamically change the background color.

Code:

```
import Java.applet.Aplet
import Java.awt.*;
import Java.awt.event

Public class welcome Applet extends
Applet implement actions listener
 Button change color button
color current color = Color.white;

 Public void unit() {
change.color Button = new Button
```

```
("change Background color")
End (change colorButton);

change color button add Action lister
(this);}

public void pon't ( graphics g ) {
set background ( current color);
g.set color (color. block);
g. draw string( "welcome to Java
applet programming)"

public void
action performed (Action Event e) {
current color = new
color ((int) (math.random)() * 255)
(int) (math.random() * 255) (int)
(math.random) () * 255));
repaint ();
}
}.
```

Output:

- Display "Welcome to Java applet programming"
- Backcolour changes dynamically on click button.

- Conclusion:

This experimeld successfully implements an interactive JAVA applet that displays a message and dynamically changes background color enhancing user interaction

raisoni EDUCATION

Experiment No. -10.

create a GUI based calculator using JAVA swing components like button and text fields.

Aim: To develop a simple GUI based calculator using, JAVA swing components buttons and text fields.

Theory: Swing is a JAVA GUI framework that provides components like button, text fields and like JButton, JtextField, and Jframe to build interactive applicat-ons.

Objective:
1) To create a basic calculator with GUI
2) To implement event handling for performing arithmetic operation.

Code:

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

Public class calculatotas extend
 Jframe implements Action lister{
  Jtext field display;
  double num1, num2, result;
     error operation
  public calculator() {
  display = new J-TextField();
```

```
display setbound (30, 40, 280, 30)
add (display);

string (]button = {"7" "8" "1" "9", "1", "4"
"5", "6", "7", "*", "1", "2", "3", "8", "0",
"c", "p", "8"};

Jpanel panel = new Japent ();
panel set layout (new and layot
(new and layot (4,4,10,10));
panel.set bound (30, 100, 280, 200);
for (string text; bottau);
}
add (panel);

set size (310, 400);
set layout (null);
set visible (true);

set default close operator (Jframe.NEI-ON-
CLOSE); }

Public void
action performed (Action event) {
string command = c. get Action();
if command chart At(0) >=
command chart as (0) <= 'g') {

display: set Text (display .get Text ()
command);
```

```
switch (operator){
case: '+': result(num1+num2: break;
case: '-', result =num1-num2; breake;
case: '*' : result =num1 * num2 : break;
case: '/': result = num1 / num2 : breake;

display.set Pset (" "+resit);
 } ase
   num1 = Double . parse -bouble
   (display.get text (/);

   operator = command. char At (0);
     display'set text (" ");}
}
public static void main (string args[])
          new calculator ();
     }
}
```

Output: A simple calculator GUI allows basic a-Atrmetic operator.

Conclucion:
This experiment successFully demonstrate GUI programming using JAVA. The calculator performs basic arithmetic operation and handle uses interaction.

© Amit

raisoni EDUCATION

Experiment NO-11.

Write a program to demonstrate the single design pattern and explain its real world use case.

Aim: To implement a singleton design pattern to JAVA and explain if tractical applications.

Theory: The singleton design pattern assures that a class has only one instance and provides a global point of access get. It is commonly used in database connections logging and configuration setting.

Objective:

1) To understand the suggestion design pattern.
2) To implement in JAVA and demonstra its usage.

Code:

```
class singleton {
   private static singleton {
    instance;
   private singleton () {
   }
   public static singletone
    get instance () {
      if instance = = null () {
   instance = new singleton ();
    }    return instanse ;    }
```

```
return instance;
}
  public void Message (String args[]){
  }
}
public class singleton Demo{
  public static void main (String args[]){
  singleton obj1 = singleton.getInstance();
  singleton obj2 = singleton.getInstance();

  obj1 = show message();
  S.O.P.("Are object1 and object2 the
  same instance" + (obj1.iF = obj2));
  }
```

Output : singleton instance accesseed and obj.1 and obj2 the same instance true

Conclusion: This experiment successfully demonstrated the singleton design pattern ensuring the only one instance of a close is created and stored. The pattern is widely used in logging catching and databox correction.

raisoni EDUCATION

Experiment No-12

Develop a program to creats multiple threads For tasks. Like a File reading and mathematical computations ensuring thread recognition.

Aim: To create a multithreaded JAVA program for performing file reading and mathematical computation ensuring proper thread synchornisation

Theory: Multi-threading allows concurrent execution of multiple task improving performance, synchonig action ensures that shared resources are accessed safely is multi thread environment.

Objective:
1) Tounderstand multithreading in JAVA
2) To implement syhorornized threads for file reading and computations.

code:

```
class File reaction Task extends threade
  public void sum () {
   S.O.P. ("Reading file ... ");
    try {
     Thread . sleep (2000);
    } catch.
class Math computing Task extend Threade
   public void run () {
      synchronized (this) {
```

```
S.O.P.("Performing" mathematical computatio
...");
 int sum=0;
for(int i=1; i<=100; i++)
 {
 sum+=1;
 }
SOP ("computation result:" sum);
      }
    }
}
```

```
Public class, Multithread Demo {
 Public static void main (String
 args[]){
File Reader Task File task = new
 File Reader Task();
 File Task .start();
  main Task.start()
 }
```

Output: Performing mathematical computation. computation result: 5050

Conclusion:

Thus experiment successfully demonstrates the creation e.g multiple threads for concurrent execution task. The use of synchronization ensures data consistantly with accessing shared.