Ex : 1	PROGRAM USING CLASS AND Object.
DATE: 19/06/24	

AIM:

To create a JAVA Program that have multiple user inputs & class & object concept.

PROCEDURE:

STEP 1 : Import the 'scanner' class to read input from the console

STEP 2 : Define a class to read input from the console with instance variables 'name' & 'age', a constructor to initialize these variables & a method 'display Details()' to print the student's details.

STEP 3 : Define a 'Main' class with the 'main' method.

STEP 4 : In the 'main' class with that create a 'scanner' object to read input.

STEP 5 : Use a loop to repeatedly ask for details

STEP 6 : Create 'student' object based on user inputs.

STEP 7 : After loop ends display the details.

CODE:

```
import java.util.ArrayList;
import java.util.Scanner;
class Student {
    private String name;
    private int age;
    public Student(String name, int age) {
        this.name = name;
        this.age = age; }

    public void displayDetails(){
        System.out.println("Name: "+name);
        System.out.println("Age: " + age); }
}
public class Main {
    public static void main(String[] args){
        Scanner scanner = newScanner(system.in);
        ArrayList<Student> students = newArrayList<>();
        String moveInput;
    do {
        System.out.print("Enter Student name: ");
        Stringname = Scanner.nextLine();
        System.out.print("Enter student age: ");
        int age = scanner.nextInt();
        Student student = new Student(name, age);
        Students.add(student);
        System.out.println("\n Details: "); }
        scanner.close(); }}
```

Sample Input & Output:

```
Enter student name : Alice
Enter Student age : 20
Do you want to add more?(Y/n): Y
Enter student name : Bob
Enter Student age : 22
Do you want to add more ? (Y/n) : N
Student Details:
  Name : Alice
  Age  : 20
  Name : Bob
  Age  : 22
```

RESULT:

The program is successfully accepts multiple student inputs from the user and displays it.

Ex : 2	PROGRAM USING CONSTRUCTORS
DATE : 26/06/24	

AIM:

To manage and write a code in java to use a employee management system.

PROCEDURE:

STEP 1 : Import Scanner to read input

STEP 2 : Define class 'employee' with instance variables 'name', 'salary' & method displayDetails(). to print.

STEP 3 : Define Main

STEP 4 : In the main create scanner object

STEP 5 : Use a loop to ask employee data.

STEP 6 : Create employee object based on the user input & store them in a list

STEP 7 : After loop ends display all the data.

CODE:

```
import java.util.ArrayList;
import java.util.Scanner;
class Employee {
  private String name;
  private double salary;
  public Employee(String name, double salary) {
    this.name = name;
    this.salary = salary; }
  public void displayDetails() {
    System.out.println("Name: " + name);
    System.out.println("Salary: $" + salary); } }

public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<Employee> employees = new ArrayList<>();
    String moreInput;
    do {
      System.out.print("Enter name ");
      String name = scanner.nextLine;
      System.out.print("Enter salary ");
      double salary = scanner.nextDouble();
      scanner.nextline();
      Employees employee = new Employee(name, salary);
      employees.add(employee);
      System.out.print("Do you add more: ? ");
      moreInput = scanner.nextLine(); }
      while (moreInput.equalsIgnoreCase("yes")); {
        System.out.println("In Employees:");
        for (Employee employee : employees) {
      } scanner.close();
    }
}
```

SAMPLE INPUT OUTPUT:

Enter name : John
Enter Salary : 50000
Do you want to add more : ? (Y/n) : Y
Enter name : Jane
Enter salary : 60000
Do you want to add more : (Y/n) ? : N

Employee

Name : John
Salary : 50000
Name : Jane
Salary : 60000

RESULT :

The employee management system is built and runned successfully.

EX : 3	PROGRAM USING COMMANDLINE ARGUMENTS
DATE : 10/7/24	

AIM:

To write a java program that takes command line arguments and gives their median.

PROCEDURE:

STEP 1 : Begin execution with the 'main' method of the 'MedianCalculator' class.

STEP 2 : If 'args.length' is 0, print "No numbers provided' & exit.

STEP 3 : Create an integer array 'numbers' of length 'args.length'. Convert to int and print error if it fails.

STEP 4 : Sort the numbers using Arrays.sort()

STEP 5 : Print sorted numbers.

STEP 6 : (i) Find length of elements
(ii) If L=odd, median = middle element
(iii) If L=even, median = avg of middle elements.

STEP 7 : Print the output.

CODE:

```
import java.util.Arrays;
public class MedianCalculator{
  public static void main(String[]args){
    if (args.length==0){
      System.out.println("No numbers provided.");
      return;}
    int[] numbers: new int[args.length];
    for (int i=0; i<args.length; i++){
      try {numbers[i]= integer.parseInt(args[i]);
}catch (NumberFormatException e){
      System.out.println("Invalid input");
      return;}}
Arrays.sort(numbers);
System.out.print("Sorter numbers: ");
for(int num : numbers){
    System.out.print(num+" ");}
double median:
int length = numbers.length;
if (length % 2==0){
    median= (numbers[length/2-1]+numbers[length/2])/2;
}else{
    median = numbers[length/2];}
System.out.println("Median:"+median);}}
```

Sample Input / Output:

java Median Calculator 15 9 4 6 2

Sorted numbers: 1 2 4 5 6 9

Median : 4.5

RESULT

The program is executed successfully.

Ex: 4	PROGRAM USING VECTORS
DATE: 24/7/24	

AIM: To solve the knapsack problem using a Greedy Algorithm to maximize the total value of the items that can be put into a given capacity.

ALGORITHM:

STEP 1: Create a 2D array 'dp' where dp[i][w] represents the max value achievable with first 'i' in knapsack.

STEP 2: Set 'dp[0][w]' to 0 for all 'w'. meaning no items lead to a value of 0.

STEP 3: For each item 'i' and each capacity 'w' update 'dp[i][w]' to be the maximum of either not taking the item.

STEP 4: If current items weight is less than or equal to 'w'. update 'dp[i][w]' as 'dp[i-1][w-itemweight] + itemValue.

STEP 5: After filling the table, 'dp[n][w]' will hold the maximum value achievable.

STEP 6: Print the value of dp[n][w] as result.

CODE:

```
import java.util.*;
class Item{
    double weight, value, ratio;
    Item(double weight, double value){
        this.weight = weight;
        this.value = value;
        this.ratio = value/weight;}}

public class KnapsackGreedy{
    private static final double CAPACITY = 50.0;
    public static void main(String[] args){
        List<Item> items = Arrays.asList(
                new Item(10,60),
                new Item(20,100),
                new Item(30,120));
        double maxValue = Knapsack(items, CAPACITY);
        System.out.println("MaxValue:" + maxValue);}

private static double knapsack(List<Item> items, double
                                        capacity){
    items.sort((a,b) -> Double.compare(b.ratio, a.value));
    double totalValue = 0;
    for(Item item : items){
        if(capacity <= 0) break;
        totalValue += weightToTake * item.ratio}
        return total value;}}
```

Output:

Max value : 240.0

RESULT:

The program using vectors is successfully executed.

EX NO: 5	PROGRAM USING INTERFACE
DATE: 31/7/24	

AIM:

To calculate and display the fuel efficiency of different vehicles based on amount of petrol used & different distance traveled.

ALGORITHM:

Step 1: Define the interface & set the necessary values.

Step 2: Implement the interface: Define 'car' & 'Bike'.

Step 3: Implement Methods: Each class should implement methods to set petrol, distance.

Step 4: Create Instances: Initiate 'car' & 'Bike' objects.

Step 5: Set values: Input values for petrol & distance for each vehicle.

Step 6: Compute & print the fuel efficiency of each vehicle.

CODE

```
import java.util.*;
interface Vehicle {
    void setPetrol (double amt);
    void setDistance (double dist);
    double getfuelEfficiency (); }

class Car implements Vehicle {
    private double petrol;
    private double distance;
    @Override
    public void setPetrol (double amt) {
        this.petrol = amt; }

    @override
    public void setDistance (double dist) {
        this.distance = dist ; }

    @override
    public double getfuelEfficiency () {
      return distance / price ; } }

class Bike implements Vehicle {
    private double petrol;
    private double distance;
    public void setPetrol (double amt) {
        this.petrol = amt ; }
    public void setDistance (double dist) {
        this.distance = dist; }
    public double getfuelEfficiency () {
        return distance / petrol; } }
```

```
public class Main {
    public static void main (String [] args) {
    Vehicle car = new Car();
    Vehicle bike = new Bike();
    car. setPetrol (10);
    Car. setDistance (250);
    bike. setPetrol (5);
    bike. setDistance (150);
    System.out.println ("Car Efficiency: " + car.getFuel
                                    Efficiency + "km/l");
    System.out.println(" Bike Efficiency: " + bike.getFuel
                                    Efficiency() + "km/l"); }}
```

Output:

Car Efficiency: 25.0 km/l

Bike Efficiency: 30.0 km/l

RESULT:

The program is executed successfully.

Ex: 6	PROGRAM USING All FORMS of INHERITANCE
DATE: 7/8/24	

AIM: To demonstrate all types of inheritance (single, multilevel, hierarchial & multiple through Interface) in Java using a simple program.

PROCEDURE:

STEP 1: Define a Person class to represent single Inheritance.

STEP 2: Define an Employee class that extends Person to represent single inheritance.

STEP 3: Define a Manager class that extends Employee to represent multilevel inheritance

STEP 4: Define a circuit class that extends Person to represent hierarchial inheritance.

STEP 5: Define an Interface Department to represent multiple inheritance.

STEP 6: Implement the Department interface in Manager to represent multiple inheritance

STEP 7: Create objects of Manager and Client classes to test the inheritance.

STEP 8: Run the program to display the output showing the inheritance types in action

CODE:

```
class Person{
  string name;
  public Person(str name){
    this.name= name;}
  public void displayPerson(){
    System.out.println("Person Name: "+name);
}} class Employee extends Person{
    String employee ID;
    public Employee(String name, String employee ID){
      super(name);
      this.employee ID= employee ID;}
    public void displayEmployee(){
      System.out.println("Employee ID: "+employeeID);
}} interface Department{
    void showDepartment();}
class Manager extends Employee implements Department{
  String department;
  public Manager(String name, String employee ID,
                 string department){
    super(name, employee ID);
    this.department = department;}
  public void showDepartment(){
    System.out.println("Department: "+dept);}}

class Client extends Person{
    String cliend ID;
  public client(str name, str clientID){
    super(name);
    this.client ID= cliendID;}
  public void display client(){
    System.out.println("clientID: "+clientID);}}
```

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Manager manager = new Manager("Alice", "E123",
                                      "Sales");

        Manager.displayPerson();
        manager.displayEmployee();
        manager.showDepartment();
        Client client = new Client("Bob", "CL456");
        client.displayPerson();
        client.displayClient();
```

Output:

```
Person name : Alice
Employee ID : E123
Department  : Sales
Person name : Bob
Client ID   : CL456
```

RESULT:

The Java program is written and executed successfully.

EX: 7	PROGRAM USING STRING Class & String Buffer class.
DATE: 14/8/24	

AIM:

To demonstrate the use of the String & String Buffer class in Java for manipulating Strings.

PROCEDURE:-

Step 1: Create a StringExample class to work with the String class.

Step 2: Initialize a String with a sample value

Step 3: Demonstrate various String operations, length and comparison.

Step 4: Create a StringBuffer Example class to work with the String Buffer class.

Step 5: Initialize a String Buffer with a sample value.

Step 6: Demonstrate various StringBuffer operations like append, insert & remove.

Step 7: Create a main class StringDemo to call both String Example and StringBuffer Example functions.

Step 8: Run the program to see the differences in Immutability (String) and mutability (String Buffer).

CODE:

```
import java.util.*;
class StringExample{
  public void stringOperations(){
    String str = "Hello";
    System.out.print("Org String: " + str);
    str = str.concat("World!");
    System.out.println("Aft Concate : " + str);
    System.out.println("Length of String: " + str.length());
    System.out.print("Char at [1]: " + str.charAt(1));
    String str2 = "Hello World!";
    System.out.println("String comparison (str == str2): "
                       + str.equals(str2)); }}

class StringBufferExample{
  public void StringBuffer(){Hello");
  System.out.println("Original StringBuffer: " +sb);
  sb.append("World!");
  System.out.print("Aft Append: " + sb);
  sb.insert(6,"Beautiful");
  System.out.println("After Insertion: "+sb);
  sb.reverse();
  System.out.println("After Reversing": + sb); }}

public class StringDemo{
  public static void main(String[] args){
    string Example stringExample = new StringExample();
    stfring Example.String Operation();
    StringBuffer Example StringBufferExample = new String
                                        BufferExample();
    StringBufferExample.StringBufferOperations();
  }
}
```

Output

```
Original String : Hello
After Concatenation : Hello World!
Length of string : 12
Character at index 1 : e
String comparison (str == str2) = true

Original StringBuffer : Hello
After Append : Hello World!
After Insertion : Hello Beautiful World!
After Reversing : !dlroW lufituaeB olleH
```

RESULT:

The program in JAVA is written and executed succesfully

Ex:8	Program Using Exception Handling.
DATE: 21/8/24	

AIM:

To determine exception handling in Java by handling different types of exception using try, catch, finally & throw.

PROCEDURE:

Step 1: Create a Division Example class to perform division of two numbers.

Step 2: Implements a divideNumbers method that accepts two integers and handles possible exceptions.

Step 3: Use try & catch blocks to handle ArithmeticException (division by zero).

Step 4: Use another try-catch to handle NumberFormat Exception (invalid input format).

Step 5: Use the finally block to ensure certain code runs regardless of exceptions.

Step 6: Implement a throw statement to manually throw errors detected.

Step 7: Create a main class to test division & exception handling.

Step 8: Run the program with different inputs to trigger exceptions.

CODE:

```
import java.util.Scanner;
class DivisionExample {
  public void divideNumbers(String input1, String
                                    input2){
    try {
      int num1 = Integer.parseInt(input1);
      int num2 = Integer.parseInt(input2);
      if(num2 == 0){
        throw new ArithmeticException("By zero")
      ;};
      int result = num1/num2;
      System.out.println("Result "+result);}
    catch(NumberFormatException e){
      System.out.println("Invalid");}
    Catch(ArithmeticException e){
      System.out.println("Error: "+e.getMessage());}

    finally{
      System.out.println("Execution completed.");}}}
public class ExceptionDemo{
  public static void main(String[] args){
    Scanner scanner = new Scanner(System.in);
    DivisionExample divisionExample = new DivisionExample();
    System.out.println("Enter first number: ");
    String input1 = scanner.nextLine();
    System.out.println("Enter second number: ");
    String input2 = scanner.nextLine();
    divisionExample.divideNumbers(input1, input2);

  }
}
```

Output

```
Enter first number
10
Enter second number:
2
Result :5
Execution complete

Enter first number
10
Enter second number:
0
Error: By zero is not allowed

Enter first number
abc
Enter second number
2
Error: Invalid number format
```

RESULT:

The program is written and executed Succesfully.

Ex: 9	Implementing Thread based Applications
DATE: 28/8/24	

AIM: To implement a multithreaded application in Java.

PROCEDURE:

Step 1: Create a class FileDownload that extends thread.

Step 2: Override the run method to simulate downloading files.

Step 3: Use the sleep method to simulate the time taken for each file download.

Step 4: Create multiple threads to download multiple files simultaneously.

Step 5: Use start() to initiate the download thread.

Step 6: Display the status of each download (start, complete).

Step 7: Use join() to ensure all threads complete execution before proceeding.

Step 8: Run the program and observe concurrent downloads.

CODE:

```
class FileDownloader extends Thread {
    private String fileName;
    public FileDownloader(String fileName){
        this.fileName = fileName; }
    public void run(){
        System.out.println("Starting download:"
                                + fileName);
        try {
            Thread.sleep((long)(Math.random()*5000)); }
        catch(InterruptedException e){
            System.out.println(fileName + "download
                                Interrupted"); }
        System.out.println("Completed download: "
                                + fileName);

} }
public class DownloadManager{
    public static void main(str[] args){
        String[] files = {"file1.pdf", "file2.jpg"};
    Thread[] downloaders = new Thread[files.length];
    for(int i=0; i< files.length; i++){
        downloaders[i] = new FileDownloader(files[i]);
        downloaders[i].start();
    for(Thread downloader : downloaders){
        try {
            downloader.join(); }
        catch(InterruptedException e){
            System.out.println("Download Interrupted."); } }

}
}
```

Output

Starting download: file1.pdf
Starting download: file2.jpg
Completed download: file1.pdf
Completed download: file2.jpg
All downloads completed.

RESULT:

The above java program is written and executed sucessfully.

EX: 10	PROGRAM USING PACKAGES
DATE: 4/9/24	

AIM: To use packages in a java program for calculating & display purpose.

PROCEDURE:

Step 1: Create a package named calculation & define a class calculator within it.

Step 2: Implement methods in Calculator for basic arithmetic operations (addition, sub, mul, div).

Step 3: Create another package named display & define a class DisplayResult within it.

Step 4: The DisplayResult class will print the results of operation.

Step 5: Import the calculation & display packages into the main program.

Step 6: Use the calculator class to perform operations & the DisplayResult class to display results.

Step 7: Compile & execute the program using packages.

Step 8: Ensure the program follows the correct package structure.

CODE:

Package: Calculation

```
package calculation;
public class Calculator {
    public int add (int a, int b)
      { return a+b; }
    public int sub (int a, int b)
      { return a-b; }
    public int mul (int a, int b)
      { return a*b; }
    public double div (int a, int b)
      { return a/b; }
```

Package: Display

```
package display;
public class DisplayResult {
    public void showResult (String operation,
                            double result) {
    System.out.println(" The result of " + operation
                + " is : " + result );
}
}
```

Main App.java

```
import calculation.Calculator;
import display.DisplayResult;
public class MainApp {
    public static void main(String[] args) {
        calculator.Calculator calculator = new Calculator();
        DisplayResult display = new DisplayResult();
        int a = 10;
        int b = 5;
        int sum = calculator.add(a,b); display.showResult(sum);
        int diff = calculator.sub(a,b); display.showResult(diff);
        int mul = calculator.mul(a,b); display.showResult(mul);
        int div = Calculator.div(a,b); display.showResult(div);
    }
}
```

Output:

```
The result of Addition is : 15
The result of sub      is : 5
The result of mul      is : 50
The result of div      is : 2.0
```

RESULT:

The program is written and executed successfully.

EX: 11	PROGRAM USING FILES
DATE: 11/9/24	

AIM:

To determine file handling in Java by making a simple program to write and read a file.

PROCEDURE:

Step 1 : Import necessary classes for file handling

Step 2 : Create a class File Operations.

Step 3 : Implement a method to write user input to a text file.

Step 4 : Implement a method to write user read.

Step 5: Use scanner to take input from the user.

Step 6: Call the write method to save input & the read method to display the content.

Step 7 : Compile and run the program.

Step 8 : Ensure the program handles exception.

CODE:

```
import java.io.*;
import java.util.Scanner;
public class FileOperations {
    Public void writeToFile(String data){
      try(BufferedWriter writer= newBufferedWriter
      (new FileWriter("output.txt",true)
      writer.write(data);
      writer.newLine();
      System.out.println("Data written to file.");
      catch(IOException e){
        System.out.println("Error" + e.getMessage());}

    }
public void readfromfile(){
  try(BufferedReader reader=newBuffered Reader
      (new FileReader("Output.txt")))

  {
    String line;
    System.out.println("File contents:");
    while(line.reader.readLine()!=null){
    System.out.println(line);}}

  catch(IOException e){
      System.out.println("Error reading from file:
                          +e.getMessage();}

}
}
```

```
public Static void main (String[] args) {
    File Operations fileOps = new file Operations ();
    Scanner scanner : new scanner (system. in);
    System. out. print (" Enter data ");
    String input data : Scanner. nextLine ();
    fileOps. writeToFile (input Data);
    fileOps. read FromFile ();
    Scanner. close ();
```

Output:

```
Enter data to write: Hello
Data written to file
File contents:
Hello
```

RESULT:

The Java program is written and executed successfully.

Ex: 12	Working With Colors and Fonts Using Applets
DATE 18/9/24	

AIM: To create a simple java applet that demonstrates the use of colors and fonts

PROCEDURE:

Step 1 : Import the necessary applet and graphics packages.

Step 2 : Create a class that extends Applet.

Step 3 : Override the paint method to draw text using various colors and fonts.

Step 4 : Use setColor to apply different colors to text.

Step 5 : Use setFont to change the font style, size and type.

Step 6 : Call drawString to display text with the specified styles.

Step 7 : Compile & run the applet to see the output.

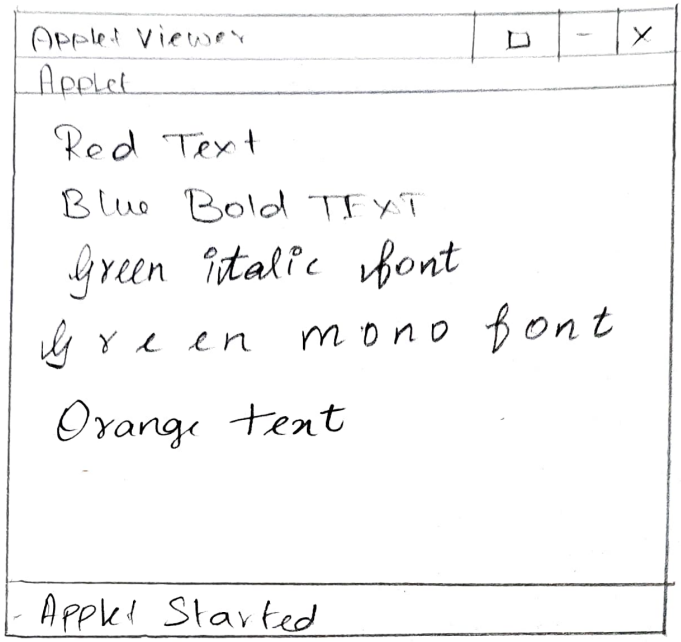
CODE:

```
import java.applet.Applet;
import java.awt.*;
public class ColorFontApplet extends Applet {

    @Override
    public void paint(Graphics g) {
        g.setColor(Color.RED);
        g.drawString("This is RED text!", 20, 40);
        Font boldFont = new Font("Serif", Font.BOLD, 20);
        g.setFont(boldFont);
        g.setColor(Color.BLUE);
        g.drawString("This is blue bold text!", 20,
                                                80);

        Font italicFont = new Font("Monospaced", Font
                                    .ITALIC, 18);

        g.setFont(italicFont);
        g.setColor(Color.GREEN);
        g.drawString("This is green italic text!",
                                    20, 120));

        Font dialogFont = new Font("Dialog",
                                    Font.PLAIN, 16);

        g.setFont(dialogFont);
        g.setColor(Color.ORANGE);
        g.drawString("This is organge text
                        with Dialog font!", 20, 160);

    }
}
```

HTML

```
<html>
<body>
<h2> Applet </h2>
<applet code: "ColorFontApplet.class  width: "400"
        height: "200" >

</applet>
</body>
</html>
```

Output:



RESULT:

The java applet program is written and executed successfully.

EX: 13	Parameter passing techniques using applets
DATE: 25/9/24	

AIM: To demonstrate parameter passing techniques to Java applets

PROCEDURE:

Step 1: Create a simple Java applet that accepts parameter from HTML.

Step 2: Use the <param> tag within the HTML to pass values.

Step 3: Override the init() method in the applet to retrieve these parameters using the getParameter() method.

Step 4: Use Graphics to display the values passed as parameters.

Step 5: Compile the applet code and create the HTML file that passes parameter.

Step 6: Test the applet using an applet viewer or compatible browser.

Step 7: Verify that the parameters passed via the HTML file are displayed in the applet.

Step 8: Ensure proper handling of missing or invalid parameters.

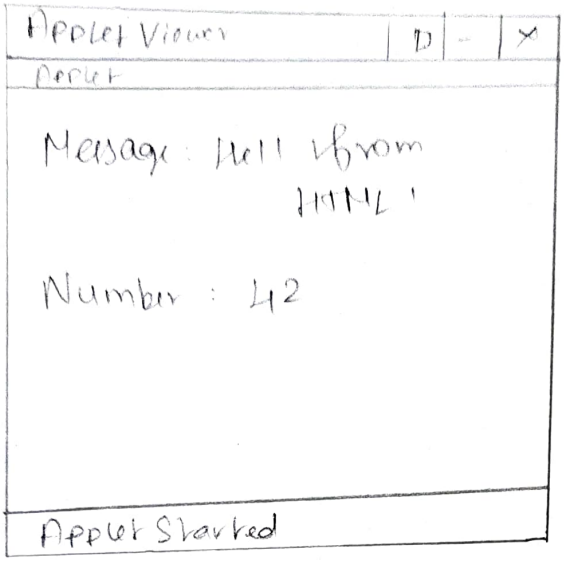
CODE:

```
import java.applet.Applet;
import java.awt.*;
public class ParameterApplet extends Applet {
    String message;
    int number;
    @Override
    public void init(){
        message = getParameter("message");
        if (message ==null) {
            message = "No message passed!";
        }
    Try {
        number = Integer.ParseInt(getParameter
                                  ("number"));
} catch (NumberFormatException e) {
    number = 0; }}
 @Override
    public void paint(Graffics g) {
        g.drawString("Message:" + message,20,40);
        g.drawString("Number:" +number,20,60);
}
}
```

HTML

```
<html>
<body>
<h2> Applet Code </h2>
<applet code:" ParameterApplet.class ">
    <param name:"message" value:" Hell from
                                HTML!" >
    <param name:"number" value:"42" >

</applet>
</body>
</html>
```

Output:



RESULT:

The passing parameters program is written & executed successfully.

EX: 14	Drawing Various Shapes Using graphical Statements
DATE: 25/9/24	

AIM: To draw various shapes using graphical statements in a Java applet.

PROCEDURE:

Step 1: Create a simple applet that extends the applet class.

Step 2: Override the paint() method to draw different shapes like rect, oval, line.

Step 3: Use Graphics class methods like drawRect(), drawOval(), etc. to render shapes

Step 4: Set different colors for each shape using setColor() method

Step 5: Compile the applet and run it

Step 6: Display the shapes in a visually clear manner.

Step 7: Handle any potential errors

Step 8: Test the applet using an applet viewer.

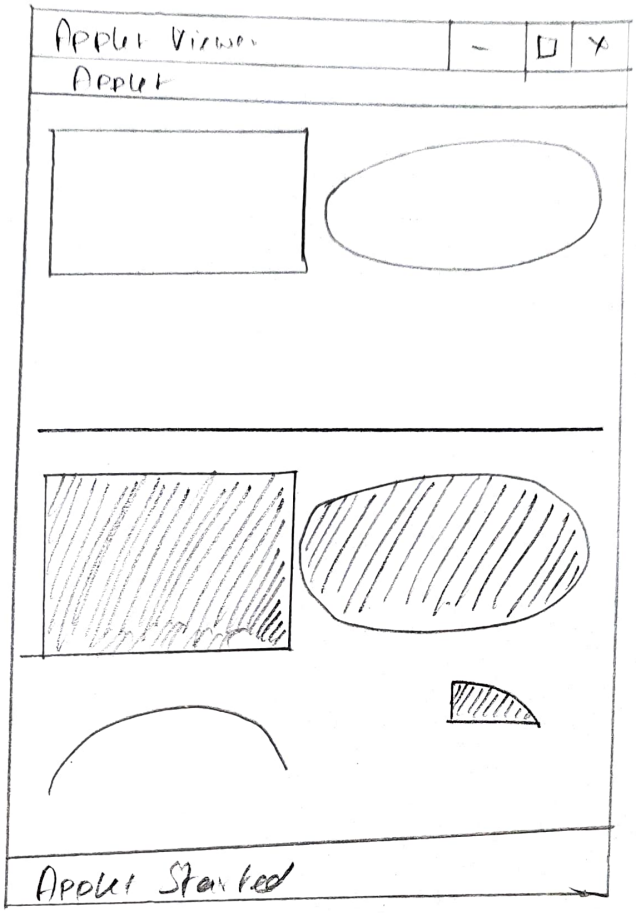
CODE:

```
import java.applet.Applet;
import java.awt.*;
public class ShapeDrawingApplet extends
                                    Applet {
  @Override
  public void paint(Graphics g){
      g.setColor(Color.RED);
      g.drawRect(20,20,100,50);
      g.setColor(Color.BLUE);
      g.setColor(Color.GREEN);
      g.drawLine(20,100,250,100);
      g.setColor(Color.ORANGE);
      g.fillRect(20,120,100,50);
      g.fillOval(150,120,100,50);
      g.setColor(Color.CYAN);
      g.drawArc(20,200,100,50,0,180);
      g.setColor(Color.MAGENTA);
      g.fillArc(150,200,100,50,0,90);

  }
}
```

HTML

```
<html>
<body>
<applet code = "ShapeDrawingApplet.class">
  Your PC do not support it
</applet>
</body>
</html>
```

Output:



RESULT:

The program using java applet is written and executed successfully