INDEX

Practical - 1

Aim: To write a program to check whether two strings are equal or not.

Algorithm:

1. Take two strings as input from the user.
2. Check the length of both strings. If they are not equal, then they are not equal.
3. If both the lengths are equal, then compare each character of the strings.
4. If any character of the two strings is not equal, then they are not equal.
5. If all the characters of the two strings are equal, then they are equal.

Program:

```
import java.util.Scanner;


public class StringCompare {


   public static void main(String[] args) {


      Scanner sc = new Scanner(System.in);


      System.out.println("Enter the first string: ");
      String str1 = sc.nextLine();


      System.out.println("Enter the second string: ");
      String str2 = sc.nextLine();


      if(str1.length() != str2.length()) {
         System.out.println("Strings are not equal");
      }
```

```
        else {
            boolean isEqual = true;
            for(int i=0; i<str1.length(); i++) {
                if(str1.charAt(i) != str2.charAt(i)) {
                    isEqual = false;
                    break;
                }
            }
            if(isEqual) {
                System.out.println("Strings are equal");
            }
            else {
                System.out.println("Strings are not equal");
            }
        }
        sc.close();
    }
}
```

Output - 1:

Enter the first string:

hello

Enter the second string:

hello

Strings are equal

Output 2:

Enter the first string:

world

Enter the second string:

hello

Strings are not equal

Practical - 2

Aim: To write a program to display reverse string.

Algorithm:

1. Take a string as input from the user.
2. Find the length of the string.
3. Traverse the string from the end to the start and append each character to a new string.
4. Display the reversed string.

Program:

```
import java.util.Scanner;


public class ReverseString {


   public static void main(String[] args) {


      Scanner sc = new Scanner(System.in);


      System.out.println("Enter the string: ");
      String str = sc.nextLine();


      String reverse = "";
      for(int i=str.length()-1; i>=0; i--) {
         reverse = reverse + str.charAt(i);
      }
      System.out.println("Reversed string: " + reverse);
      sc.close();
   }
```

}

Output – 1:

Enter the string:

hello world

Reversed string: dlrow olleh

Output – 2:

Enter the string:

12345

Reversed string: 54321

Practical - 3

Aim: To write a program to find the sum of digits of a given number.

Algorithm:

1. Take an integer as input from the user.
2. Initialize a variable sum to 0.
3. Extract each digit from the number using the modulo (%) operator and add it to the sum.
4. Divide the number by 10 and continue the process until the number becomes 0.
5. Display the sum of digits.

Program:

```
import java.util.Scanner;


public class SumOfDigits {


   public static void main(String[] args) {


      Scanner sc = new Scanner(System.in);


      System.out.println("Enter the number: ");
      int num = sc.nextInt();


      int sum = 0;
      while(num > 0) {
         int digit = num % 10;
         sum = sum + digit;
         num = num / 10;
```

```
        }
        System.out.println("Sum of digits: " + sum);
        sc.close();
    }
}
```

Output:

Enter the number:

12345

Sum of digits: 15

Practical - 4

Aim: To write a program to display a multiplication table.

Algorithm:

1. Take an integer as input from the user.
2. Use nested loops to iterate from 1 to 10.
3. Multiply the outer loop variable with the inner loop variable and display the result.
4. Repeat step 3 for all values of the outer loop variable from 1 to 10.

Program:

```
import java.util.Scanner;


public class MultiplicationTable {


   public static void main(String[] args) {


      Scanner sc = new Scanner(System.in);


      System.out.println("Enter the number: ");
      int num = sc.nextInt();


      for(int i=1; i<=10; i++) {
         System.out.println(num + " x " + i + " = " + (num*i));
      }
      sc.close();
   }
}
```

Output:

Enter the number:

8

8 x 1 = 8

8 x 2 = 16

8 x 3 = 24

8 x 4 = 32

8 x 5 = 40

8 x 6 = 48

8 x 7 = 56

8 x 8 = 64

8 x 9 = 72

8 x 10 = 80

Practical - 5

Aim: To write a program to display all prime numbers between 1 to 1000.

Algorithm:

1. Use a loop to iterate from 2 to 1000.
2. For each value of the loop variable, check whether it is a prime number or not.
3. To check whether a number is prime, use another loop to iterate from 2 to the square root of the number.
4. If the number is divisible by any of the values in this loop, it is not a prime number.
5. If the number is not divisible by any of the values in the loop, it is a prime number and should be displayed.

Program:

```
public class PrimeNumbers {


  public static void main(String[] args) {


    for(int i=2; i<=1000; i++) {
      boolean isPrime = true;
      for(int j=2; j<=Math.sqrt(i); j++) {
        if(i % j == 0) {
          isPrime = false;
          break;
        }
      }
      if(isPrime) {
        System.out.print(i + " ");
      }
```

```
        }
    }
}
```

Output:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881
883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

Practical - 6

Aim: To write a program to insert an element in an existing array.

Algorithm:

1. Declare an array with the desired length and fill it with some values.
2. Take the value to be inserted as input from the user.
3. Take the index at which the value should be inserted as input from the user.
4. Shift all the elements of the array from the given index to the end by one position to the right.
5. Insert the given value at the given index.
6. Display the modified array.

Program:

```
import java.util.Scanner;


public class InsertElement {


   public static void main(String[] args) {


      int[] arr = {1, 2, 3, 4, 5};
      int len = arr.length;


      Scanner sc = new Scanner(System.in);


      System.out.println("Enter the value to be inserted: ");
      int val = sc.nextInt();


      System.out.println("Enter the index at which the value should be inserted:
");
```

```
        int index = sc.nextInt();

        for(int i=len-1; i>=index; i--) {
            arr[i+1] = arr[i];
        }

        arr[index] = val;
        len++;

        System.out.println("Modified array:");
        for(int i=0; i<len; i++) {
            System.out.print(arr[i] + " ");
        }

        sc.close();
    }
}
```

Output:

Enter the value to be inserted:

10

Enter the index at which the value should be inserted:

2

Modified array:

1 2 10 3 4 5

Practical - 7

Aim: To write a program to sort an existing array.

Algorithm:

1. Declare an array with the desired length and fill it with some values.
2. Use a loop to compare each element with the next element and swap their positions if necessary.
3. Repeat the above step until the entire array is sorted in ascending order.
4. Display the sorted array.

Program:

```
public class SortArray {


   public static void main(String[] args) {


      int[] arr = {5, 2, 8, 3, 1, 4};
      int len = arr.length;


      for(int i=0; i<len-1; i++) {
         for(int j=i+1; j<len; j++) {
            if(arr[i] > arr[j]) {
               int temp = arr[i];
               arr[i] = arr[j];
               arr[j] = temp;
            }
         }
      }
```

```
        System.out.println("Sorted array:");
        for(int i=0; i<len; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Output:

Sorted array:

1 2 3 4 5 8

Practical - 8

Aim: To write a program to create an object for TreeSet and Stack and use all their methods.

Algorithm:

1. Create an object for TreeSet and add some elements to it.
2. Display the TreeSet using the iterator.
3. Create an object for Stack and push some elements to it.
4. Display the Stack using the iterator.
5. Pop some elements from the Stack and display it.
6. Check if the Stack is empty using the empty() method.
7. Display the size of the Stack using the size() method.
8. Use the search() method to search for an element in the Stack and display its position.
9. Use the peek() method to get the top element of the Stack without removing it.
10. Use the pop() method to remove the top element of the Stack and display it.

Program:

```
import java.util.Iterator;

import java.util.Stack;

import java.util.TreeSet;


public class CollectionExample {


   public static void main(String[] args) {


      // Creating object for TreeSet and adding elements to it

      TreeSet<String> treeSet = new TreeSet<String>();

      treeSet.add("A");
```

```
treeSet.add("B");
treeSet.add("C");
treeSet.add("D");
treeSet.add("E");

// Displaying the TreeSet using iterator
Iterator<String> treeSetIterator = treeSet.iterator();
System.out.println("TreeSet elements:");
while(treeSetIterator.hasNext()) {
    System.out.print(treeSetIterator.next() + " ");
}
System.out.println("\n");

// Creating object for Stack and pushing elements to it
Stack<Integer> stack = new Stack<Integer>();
stack.push(10);
stack.push(20);
stack.push(30);
stack.push(40);
stack.push(50);

// Displaying the Stack using iterator
Iterator<Integer> stackIterator = stack.iterator();
System.out.println("Stack elements:");
while(stackIterator.hasNext()) {
    System.out.print(stackIterator.next() + " ");
}
```

```
        System.out.println("\n");

        // Popping some elements from the Stack and displaying it
        System.out.println("Popping some elements from the Stack:");
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println("\n");

        // Checking if the Stack is empty using empty() method
        System.out.println("Is the Stack empty? " + stack.empty());
        System.out.println("\n");

        // Displaying the size of the Stack using size() method
        System.out.println("Size of the Stack: " + stack.size());
        System.out.println("\n");

        // Using search() method to search for an element in the Stack and
displaying its position
        System.out.println("The position of 30 in the Stack is: " +
stack.search(30));
        System.out.println("\n");

        // Using peek() method to get the top element of the Stack without
removing it
        System.out.println("The top element of the Stack is: " + stack.peek());
        System.out.println("\n");

        // Using pop() method to remove the top element of the Stack and
displaying it
```

```
        System.out.println("The popped element from the Stack is: " +
stack.pop());
        System.out.println("\n");

        // Displaying the updated Stack using iterator
        System.out.println("Updated Stack elements:");
        stackIterator = stack.iterator();
        while(stackIterator.hasNext()) {
            System.out.print(stackIterator.next() + " ");
        }
    }
}
```

Output:

TreeSet elements:

A B C D E

Stack elements:

10 20 30

Practical – 9

Aim: To write a program to check all Math class functions.

Algorithm:

1. Use the Math class methods for various mathematical operations like addition, subtraction, multiplication, division, power, logarithm, and trigonometric functions.
2. Display the results obtained from the methods.

Program:

```
public class MathFunctions {


   public static void main(String[] args) {


      // Addition of two numbers
      int sum = Math.addExact(10, 20);
      System.out.println("Addition of 10 and 20: " + sum);


      // Subtraction of two numbers
      int difference = Math.subtractExact(50, 20);
      System.out.println("Subtraction of 50 and 20: " + difference);


      // Multiplication of two numbers
      int product = Math.multiplyExact(10, 5);
      System.out.println("Multiplication of 10 and 5: " + product);


      // Division of two numbers
      double quotient = Math.floorDiv(50, 3);
```

```
        System.out.println("Division of 50 and 3: " + quotient);

        // Power of a number
        double power = Math.pow(2, 3);
        System.out.println("Power of 2 to 3: " + power);

        // Logarithm of a number
        double logarithm = Math.log10(1000);
        System.out.println("Logarithm of 1000: " + logarithm);

        // Trigonometric functions
        double angle = Math.toRadians(30);
        double sine = Math.sin(angle);
        double cosine = Math.cos(angle);
        double tangent = Math.tan(angle);
        System.out.println("Sine of 30 degrees: " + sine);
        System.out.println("Cosine of 30 degrees: " + cosine);
        System.out.println("Tangent of 30 degrees: " + tangent);
    }

}
```

Output:

Addition of 10 and 20: 30

Subtraction of 50 and 20: 30

Multiplication of 10 and 5: 50

Division of 50 and 3: 16.0

Power of 2 to 3: 8.0

Logarithm of 1000: 3.0

Sine of 30 degrees: 0.49999999999999994

Cosine of 30 degrees: 0.8660254037844387

Tangent of 30 degrees: 0.5773502691896257

Practical – 10

Aim: To write a program to execute any Windows 95 application.

Algorithm:

1. Algorithm:
2. Use the **Runtime** class to execute the Windows 10/11 application.
3. Provide the path to the executable file of the application.
4. Use the **Process** class to execute the application.
5. Handle any exceptions that may occur during execution.

Program:

```
import java.io.IOException;


public class ExecuteApplication {


   public static void main(String[] args) {


      String applicationPath = "C:\\Windows\\System32\\calc.exe"; // Path to
calculator executable


      try {
         Process process = Runtime.getRuntime().exec(applicationPath);
         System.out.println("Application has been executed successfully!");
      } catch (IOException e) {
         System.out.println("An error occurred while executing the
application.");
         e.printStackTrace();
      }
   }
```

}

Output:

Application has been executed successfully!

Practical - 11

Aim: To write a program to find out total memory, free memory, and free memory after executing garbage collector.

Algorithm:

1. Use the **Runtime** class to obtain information about the memory usage of the Java Virtual Machine.
2. Use the **totalMemory()** method to get the total amount of memory that the JVM has allocated.
3. Use the **freeMemory()** method to get the amount of free memory in the JVM.
4. Use the **gc()** method of the **System** class to execute the garbage collector.
5. Use the **freeMemory()** method again to get the amount of free memory after garbage collection.

Program:

```
public class MemoryManagement {


   public static void main(String[] args) {


      // Total memory
      long totalMemory = Runtime.getRuntime().totalMemory();
      System.out.println("Total memory: " + totalMemory + " bytes");


      // Free memory
      long freeMemory = Runtime.getRuntime().freeMemory();
      System.out.println("Free memory: " + freeMemory + " bytes");


      // Execute garbage collector
      System.gc();
```

```
        // Free memory after garbage collection
        long freeMemoryAfterGC = Runtime.getRuntime().freeMemory();
        System.out.println("Free memory after garbage collection: " +
freeMemoryAfterGC + " bytes");
    }


}
```

Output:

Total memory: 522190848 bytes

Free memory: 515425192 bytes

Free memory after garbage collection: 521927112 bytes

Practical - 12

Aim: To write a program to copy a file to another file using Java to package classes.

Algorithm:

1. Prompt the user to enter the name of the source file and the destination file.
2. Check if the source file exists, if not, display an error message and terminate the program.
3. Check if the destination file exists, if it exists, ask the user for confirmation to overwrite the file.
4. If the user confirms, create a FileInputStream object for the source file and a FileOutputStream object for the destination file.
5. Use a buffer of size 1024 bytes to read from the source file and write to the destination file until the end of the file is reached.
6. Close the FileInputStream and FileOutputStream objects.

Program:

```
import java.io.*;


public class FileCopy {


   public static void main(String[] args) {


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


      try {
         System.out.print("Enter the source file name: ");
         String sourceFileName = br.readLine();
         File sourceFile = new File(sourceFileName);
```

```
        if (!sourceFile.exists()) {
            System.out.println("Source file not found.");
            return;
        }

        System.out.print("Enter the destination file name: ");
        String destFileName = br.readLine();
        File destFile = new File(destFileName);

        if (destFile.exists()) {
            System.out.print("The destination file already exists. Do you want to
overwrite it? (Y/N): ");
            String choice = br.readLine();

            if (!choice.equalsIgnoreCase("Y")) {
                System.out.println("File not copied.");
                return;
            }
        }

        FileInputStream fis = new FileInputStream(sourceFile);
        FileOutputStream fos = new FileOutputStream(destFile);

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

        while ((length = fis.read(buffer)) > 0) {
```

```
            fos.write(buffer, 0, length);
        }

        fis.close();
        fos.close();

        System.out.println("File copied successfully.");

    } catch (IOException e) {
        e.printStackTrace();
    }
  }

}
```

Output:

Enter the source file name: test.txt

Enter the destination file name: copy.txt

The destination file already exists. Do you want to overwrite it? (Y/N): Y

File copied successfully.

Practical - 13

Aim: To write a program to get a file name at runtime and display the number of lines and words in that file.

Algorithm:

1. Prompt the user to enter the name of the file.
2. Create a File object for the specified file name.
3. Create a FileReader object to read the contents of the file.
4. Create a BufferedReader object to read each line of the file.
5. Initialize variables to count the number of lines and words.
6. Use a loop to read each line of the file and count the number of lines and words.
7. Close the FileReader and BufferedReader objects.
8. Display the number of lines and words in the file.

Program:

```
import java.io.*;


public class FileDetails {


   public static void main(String[] args) {


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


      try {
         System.out.print("Enter the name of the file: ");
         String fileName = br.readLine();


         File file = new File(fileName);
```

```
    if (!file.exists()) {
        System.out.println("File not found.");
        return;
    }

    FileReader fr = new FileReader(file);
    BufferedReader brFile = new BufferedReader(fr);

    int lines = 0;
    int words = 0;
    String line;

    while ((line = brFile.readLine()) != null) {
        lines++;

        String[] wordsArray = line.split(" ");
        words += wordsArray.length;
    }

    fr.close();
    brFile.close();

    System.out.println("Number of lines: " + lines);
    System.out.println("Number of words: " + words);

} catch (IOException e) {
    e.printStackTrace();
```

```
        }
    }

}
```

Output:

Enter the name of the file: test.txt

Number of lines: 3

Number of words: 8

Practical - 14

Aim: To write a program to list files in the current working directory depending upon a given pattern.

Algorithm:

1. Prompt the user to enter the pattern to be matched.
2. Create a File object for the current working directory.
3. Get a list of files in the directory using the listFiles() method.
4. Use a loop to iterate through the list of files and check if the file name matches the pattern.
5. Display the names of the files that match the pattern.

Program:

```
import java.io.*;


public class ListFiles {


   public static void main(String[] args) {


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


      try {
         System.out.print("Enter the pattern to be matched: ");
         String pattern = br.readLine();


         File directory = new File(".");
         File[] files = directory.listFiles();


         for (File file : files) {
```

```
            if (file.isFile() && file.getName().matches(pattern)) {
                System.out.println(file.getName());
            }
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
  }

}
```

Output:

Enter the pattern to be matched: *.txt

file1.txt

File2.txt

Practical - 15

Aim: To create a text field that allows only numeric value and in specified length.

Algorithm:

1. Create a JFrame object and set its size, layout and close operation.
2. Create a JPanel object and set its layout.
3. Create a JLabel object to display a message.
4. Create a JTextField object and set its size and properties to allow only numeric input.
5. Add the JLabel and JTextField to the JPanel.
6. Add the JPanel to the JFrame.
7. Display the JFrame.

Program:

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;


public class NumericTextField extends JFrame implements ActionListener {

   JPanel panel;

   JLabel label;

   JTextField textField;


   public NumericTextField() {

      setSize(400, 100);

      setLayout(new BorderLayout());

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


      panel = new JPanel(new FlowLayout());
```

```
        label = new JLabel("Enter a 6 digit number:");
        textField = new JTextField(10);
        textField.setDocument(new NumericTextFieldLimit(6));

        panel.add(label);
        panel.add(textField);

        add(panel, BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        new NumericTextField();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
    }
}

class NumericTextFieldLimit extends PlainDocument {
    private int limit;

    public NumericTextFieldLimit(int limit) {
        super();
        this.limit = limit;
```

```
    }

    public void insertString(int offset, String str, AttributeSet attr) throws
BadLocationException {
        if (str == null)
            return;

        if ((getLength() + str.length()) <= limit) {
            for (int i = 0; i < str.length(); i++) {
                if (!Character.isDigit(str.charAt(i))) {
                    return;
                }
            }
            super.insertString(offset, str, attr);
        }
    }
}
```

Output:

The above program creates a text field that allows only numeric input and limits the length to 6 digits. The user will not be able to enter any non-numeric characters or more than 6 digits. If the user tries to enter any invalid input, the program will not accept it.

Practical - 16

Aim: To create a Frame with 2 labels, at runtime display x and y command-ordinate of mouse pointer in the labels.

Algorithm:

1. Create a JFrame object and set its size, layout and close operation.
2. Create two JLabel objects and set their text.
3. Create a MouseMotionAdapter object to listen to mouse motion events.
4. Override the mouseMoved() method to update the text of the JLabels with the current mouse coordinates.
5. Add the JLabels to the JFrame.
6. Add the MouseMotionAdapter to the JFrame.
7. Display the JFrame.

Program:

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;


public class MouseCoordinates extends JFrame {

   JLabel xLabel, yLabel;


   public MouseCoordinates() {

      setSize(300, 200);

      setLayout(new FlowLayout());

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


      xLabel = new JLabel("X: ");

      yLabel = new JLabel("Y: ");
```

```
        add(xLabel);
        add(yLabel);

        addMouseMotionListener(new MouseMotionAdapter() {
            @Override
            public void mouseMoved(MouseEvent e) {
                xLabel.setText("X: " + e.getX());
                yLabel.setText("Y: " + e.getY());
            }
        });

        setVisible(true);
    }

    public static void main(String[] args) {
        new MouseCoordinates();
    }
}
```

Output:

The above program creates a JFrame with two JLabels to display the x and y coordinates of the mouse pointer. Whenever the mouse moves, the program updates the text of the JLabels with the current coordinates.