MODULE: 3

- **Practical-1: Read a content from file: calculate number of sentences, words and characters from the file.**

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

public class OneFileReader {
   public static void main(String[] args) throws IOException {
      int i, nword=1, nline=1, nsentence=1, nchar=0;
      try{
         FileReader fr = new FileReader("D:\\Files\\one.txt");
         while((i= fr.read())!=-1){
            System.out.print((char)i);
            nchar++;
            switch((char)i){
               case ' ':
                  nword++;
                  break;
               case '.':
                  nsentence++;
                  break;
               case '\n':
                  nline++;
                  nword++;
                  break;
            }
         }

         System.out.print("\nNumber of Sentences: "+nsentence);
         System.out.print("\nNumber of Words: "+nword);
         System.out.print("\nNumber of Characters: "+(nchar-nword));
         System.out.print("\nNumber of Lines: "+nline);
```

```
            fr.close();
        }
        catch (Exception e){
            System.out.println(e);
        }
    }
}
```

Output:

```
Object Oriented Programming
in Java
Number of Sentences: 1
Number of Words: 5
Number of Characters: 31
Number of Lines: 2
```

one.txt

```
1  Object Oriented Programming
2  in Java
```

- **<u>Practical-2:</u> Read from a file convert it to uppercase and save it into another file.**

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class TwoFileWriter {
   public static void main(String[] args) throws IOException {
      int i;
      try {
         FileReader fr = new FileReader("D:\\Files\\one.txt");
         FileWriter wr = new FileWriter("D:\\Files\\two.txt");
         System.out.println("\nReading file - one.txt : ");
         while((i= fr.read())!=-1) {
            System.out.print((char)i);
            char j =Character.toUpperCase((char)i);
            wr.write(j);
         }
         wr.close();
         System.out.println("\n\nWritten file in UpperCase - two.txt : ");

                                    FileReader fwr = new FileReader("D:\\Files\\two.txt");
         while((i= fwr.read())!=-1) {
            System.out.print((char)i);
         }
         fr.close();
         fwr.close();
      }

      catch (Exception e) {
         System.out.println(e);
      }
```

```
    }
}
```

Output:

```
Reading file - one.txt :
Object Oriented Programming
in Java

Written file in UpperCase - two.txt :
OBJECT ORIENTED PROGRAMMING
IN JAVA
```

one.txt

```
1  Object Oriented Programming
2  in Java
```

two.txt

```
1  OBJECT ORIENTED PROGRAMMING
2  IN JAVA
```

- **<u>Practical-3</u>: Remove duplicate lines from a File.**

```
import java.io.*;
public class FileOperation{
    public static void main(String[] args) throws IOException {
        PrintWriter pw = new PrintWriter("output.txt");
        BufferedReader br1 = new BufferedReader(new FileReader("input.txt"));

        String line1 = br1.readLine();
        while(line1 != null){
            boolean flag = false;
            BufferedReader br2 = new BufferedReader(new FileReader("output.txt"));

            String line2 = br2.readLine();
            while(line2 != null){
                if(line1.equals(line2)){
                    flag = true;
                    break;
                }
                line2 = br2.readLine();
            }
            if(!flag){
                pw.println(line1);

                // flushing is important here
                pw.flush();
            }
            line1 = br1.readLine();
```

```
            }
            br1.close();
            pw.close();
            System.out.println("File operation performed successfully");
    }
}
```

Output:

```
File operation performed successfully
```

- **<u>Practical-4:</u> Create a class called Student. Write a student manager program to manipulate the student information from files by using FileInputStream and FileOutputStream.**

```
import java.io.Serializable;
import java.io.*;
class Student implements Serializable{
   int rollNo;
   String name;
   public Student(int rollNo, String name){
      this.rollNo = rollNo;
      this.name = name;
   }
}

public class FourStudent {
    public static void main(String[] args) {
       try{
          Student s1 = new Student(359, "Harsh");
          Student s2 = new Student(377, "Karan");
          Student s3 = new Student(387, "Kushagara");

          FileOutputStream fout = new FileOutputStream("f1.txt");
          ObjectOutputStream oot = new ObjectOutputStream(fout);

          oot.writeObject(s1);
          oot.writeObject(s2);
          oot.writeObject(s3);
          oot.flush();
          oot.close();
          System.out.println("Done Succesfully!");

                    // Reading the file
```

```
                FileReader fr = new FileReader("D:\\Files\\f1.txt");
                int i;
                while((i= fr.read())!=-1){
            System.out.print((char)i);
                }
                fr.close();


        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Output:

```
Done Succesfully!
??♣srStudent??Le?↨??@@I♠rollNoL♦namet↕Ljava/lang/String;xp@gt♣Harshsq~@yt♣Karansq~@?t   Kushagara
```

f1.txt

```
1   ��NULENGsrNULBELStudent��Le�ETB��STXNULSTXINULACrollNoLNULEOTnametNULDCLjava/lang/String;xpNULNULSOHgtNULEMHarshsqNULANULNULNULNULSOHytNULENDKaransqNULANULNULNULNULSOH?tNUL Kushagara
```

- **<u>Practical-5</u>: Refine the student manager program to manipulate the student information from files by using the BufferedReader and BufferedWriter.**

```
import java.io.*;
import java.io.BufferedReader;


public class FiveBufferedStudent {
   public static void main(String[] args) throws IOException{


      // Buffered Write
      FileWriter writer = new FileWriter("D:\\Files\\f1.txt");
      BufferedWriter buffer = new BufferedWriter(writer);
      buffer.write("This is Java");
      buffer.close();


      System.out.println("Success");


      // Buffered Read
      FileReader fr = new FileReader("D:\\Files\\f1.txt");
      BufferedReader br = new BufferedReader(fr);
      int i;
      while((i=br.read())!=-1) {
         System.out.print((char)i);
      }
      br.close();
      fr.close();
   }
}
```

Output:

```
Success
This is Java
```

f1.txt

```
1  This is Java
```

- **<u>Practical-6</u>: Write a program to manipulate the information from files by using the Reader and Writer class. Assume suitable data.**

```
import java.io.*;
public class SixReadWrite {
   public static void main(String[] args) throws IOException {
      int data;
      try {
         Writer wr = new FileWriter("D:\\Files\\six.txt");
         Reader fr = new FileReader("D:\\Files\\six.txt");
         // Writing in file - six.txt
         wr.write("This is some text in six.txt");
         wr.close();

         while((data= fr.read())!=-1) {
            System.out.print((char)data);
         }
         fr.close();
      }
      catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

Output:

```
This is some text in six.txt
```

six.txt

```
1  This is some text in six.txt
```

- **<u>Practical-7:</u> Write a program "DivideByZero" that takes two numbers a and b as input, computes a/b, and invokes Arithmetic Exception to generate a message when the denominator is zero.**

```
import java.util.Scanner;


public class SevenDivideByZero {
   public static void main(String[] args) {
      int div=0;
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter two numbers: ");
      int n1 = sc.nextInt();
      int n2 = sc.nextInt();
      try{
         div = n1/n2;
      }

      catch(ArithmeticException e){
         System.out.println("Divide By Zero Error");
         System.out.println("n1 / n2 = "+(div));
      }
   }
}
```

Output:

```
Enter two numbers: 6
0
Divide By Zero Error
n1 / n2 = 0
```

- **<u>Practical-8</u>: Write a program to show the use of nested try statements that emphasizes the sequence of checking for catch handler statements.**

```
public class EIghtNestedTryBlock {
   public static void main(String[] args) {
      try{
         try{
            int n = 62/0;
         }
         catch(ArithmeticException e){
            System.out.println(e);
         }

         try{
            int a[] = new int[5];
            a[5]=62;
         }
         catch (ArrayIndexOutOfBoundsException e){
            System.out.println(e);
         }
      }
      catch (Exception e){
         System.out.println("Handled the Exception");
      }
   }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
```

- **<u>Practical-9:</u> Write a program to create your own exception types to handle situation specific to your application (Hint: Define a subclass of Exception which itself is a subclass of Throwable).**

```
// Use of Throw
import java.util.Scanner;
class LongLength extends Exception{
   LongLength(String str){
      super(str);
   }
}

public class NineUserDefinedException1 {
   public static void main(String[] args) throws Exception {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter 10 digit Mobile Number: ");
      try {
      long mobileNo = sc.nextLong();
      int len = (int) (Math.log10(mobileNo) + 1);
         if (!(len == 10)) {
            throw new Exception("Please enter 10 digit numbers");
         }
      }
      catch (Exception e) {
         System.out.println("Caught Exception");
         System.out.println(e);
      }
   }
}
```

Output:

```
Enter 10 digit Mobile Number: 3695924d
Caught Exception
java.util.InputMismatchException
```

// Use of Finally

```
public class NineUserDefinedException2 {
   public static void main(String[] args) {
      try {
         System.out.println("Try Block");
         int data = 65/0;
         System.out.println(data);
      }
      catch (ArithmeticException e){
         System.out.println(e);
      }
      finally {
         System.out.println("Finally gets executed");
      }
   }
}
```

Output:

```
Try Block
java.lang.ArithmeticException: / by zero
Finally gets executed
```

// Use of Throws

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

public class NineUserDefinedException3 {
   static void input() throws InputMismatchException {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter any number: ");
      int num = sc.nextInt();
      throw new InputMismatchException("Demo");
   }
   public static void main(String[] args) {
      try{
         input();
      }
      catch (InputMismatchException e){
         System.out.print("Caught ");
         System.out.println(e);
      }
   }
}
```

Output:

```
Enter any number: 6516v
Caught java.util.InputMismatchException
```

- **<u>Practical-10:</u> Write a small application in Java to develop Banking Application in which user deposits the amount Rs 1000.00 and then start withdrawing of Rs 400.00, Rs 300.00 and it throws exception "Not Sufficient Fund" when user withdraws Rs. 500 thereafter.**

```
import java.util.Scanner;


public class TenBank {
   static int Balance;
   static int exceptionCached = 0;


   // Deposit
   static void deposit(int amountDeposited){
      Balance += amountDeposited;
   }


   // Withdraw
   static void withdraw(int amountWithdrawed){
      try{
         if (amountWithdrawed > Balance){
            exceptionCached = 1;
            throw new Exception("NotSufficientFundException");
         }
         else{
            Balance -= amountWithdrawed;
         }
      }
      catch (Exception e){
         System.out.println(e);
      }
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
```

```
        int answer = 1;
        int amountWithdraw;
        int amountDeposited;

        try{
            System.out.print("Enter amount to be deposited: ");
            amountDeposited = sc.nextInt();
            deposit(amountDeposited);

            while (answer==1 && exceptionCached==0){
                System.out.print("Enter amount to withdraw: ");
                amountWithdraw = sc.nextInt();
                withdraw(amountWithdraw);
                if (exceptionCached==0){
                    System.out.print("Do you want to withdraw more money???\n1-Yes\n0-
No\nEnter: ");
                    answer = sc.nextInt();
                }
            }
            System.out.println("Balance: " + Balance);

        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output

```
Enter amount to be deposited: 1000
Enter amount to withdraw: 400
Do you want to withdraw more money???
1-Yes
0-No
Enter: 1
Enter amount to withdraw: 300
Do you want to withdraw more money???
1-Yes
0-No
Enter: 1
Enter amount to withdraw: 500
java.lang.Exception: NotSufficientFundException
Balance: 300
```

- **<u>Practical-11</u>: Write a program to handle ArrayIndexOutOfBounds exception for binary search.**public class ElevenArrayIndexOutOfBound {

```
   public static void main(String[] args) {
      int arr[] = {1,2,3,4,5};
      try{
         System.out.println(arr[5]);
      }
      catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Array Index Out Of Bound Exception");
      }
   }
}
```

Output:

```
Array Index Out Of Bound Exception
```

]

- **Practical-12: Write a Java Program that demonstrates thread class and few methods.**

```
public class Main extends Thread{
   public static void main(String[] args) {
      Main t1 = new Main();
      t1.start();
      Thread t2 = currentThread();

      System.out.println("Id: "+currentThread().getId());
      System.out.println("Name: "+currentThread().getName());
      System.out.println("Priority: "+currentThread().getPriority());
      t2.setName("Current Thread");
      System.out.println("Name: "+currentThread().getName());
   }
}
```

Output:

```
Id: 1
Name: main
Priority: 5
Name: Current Thread
```

- **Practical-13: Write a program to demonstrate thread example by implementing runnable interface.**

```
// Taking in TestThread
class testThread implements Runnable{
   public void run() {
      System.out.println("Running the Thread from different class");
   }
}

public class Main{
   public static void main(String[] args) {
      testThread t1 = new testThread();
      Thread t2 = new Thread(t1);
      t2.start();
   }
}
```

Output:

```
Running the Thread from different class
```

- **Practical-14: Write a program to demonstrate priorities among multiple threads.**

```
import java.lang.*;
public class FourteenPriorityThread extends Thread {
   public void run(){
      System.out.println("Inside the run method");
   }
   public static void main(String[] args) {
      FourteenPriorityThread t1 = new FourteenPriorityThread();
      FourteenPriorityThread t2 = new FourteenPriorityThread();
      FourteenPriorityThread t3 = new FourteenPriorityThread();

      System.out.println("Priority of Thread t1: " + t1.getPriority());
      System.out.println("Priority of Thread t2: " + t2.getPriority());
      System.out.println("Priority of Thread t2: " + t3.getPriority());

      t1.setPriority(8);
      t2.setPriority(5);
      t3.setPriority(2);

      System.out.println("Priority of Thread t1: " + t1.getPriority());
      System.out.println("Priority of Thread t2: " + t2.getPriority());
      System.out.println("Priority of Thread t2: " + t3.getPriority());

      Thread.currentThread().setPriority(10);

      System.out.println("\Priority of the main thread: " +
Thread.currentThread().getPriority());
   }
}
```

Output:

```
Priority of Thread t1: 5
Priority of Thread t2: 5
Priority of Thread t2: 5
Priority of Thread t1: 8
Priority of Thread t2: 5
Priority of Thread t2: 2

Priority of the main thread: 10
```

- **Practical-15: Write a program to demonstrate multithread communication by implementing synchronization among threads (Hint: you can implement a simple producer and consumer problem).**

```
class BookTheatreSeat{
   int totalSeats = 10;
   void bookSeat(int seats){
      synchronized(this){
         if (seats <= totalSeats) {
            System.out.println("Seats booked succesfully");
            totalSeats -= seats;
            System.out.println("Seats available: " + totalSeats);
         }
         else {
            System.out.println("Sorry! " + seats + " seats cannot be booked");
            System.out.println("Seats available: " + totalSeats);
         }
      }
   }
}



class BookMovie extends Thread{
   static BookTheatreSeat bts;
   int seats;
   public void run(){
      bts.bookSeat(seats);
   }

   public static void main(String[] args) {
      bts = new BookTheatreSeat();

      BookMovie m1 = new BookMovie();
```

```
        m1.seats = 7;
        m1.start();

        BookMovie m2 = new BookMovie();
        m2.seats = 6;
        m2.start();
    }
}
```

Output:

```
Seats booked succesfully
Seats available: 3
Sorry! 6 seats cannot be booked
Seats available: 3
```