Basic Input/Output and Data File Handling

Learning Objectives

- Input/Output in Java
- Data File Handling
- Operations on Files
- The StringTokenizer Class
- Introducing Streams

The data entered through input devices or used in processing is temporarily stored in memory. Once we exit the program or switch off our device, this data is lost. It sometimes becomes essential to permanently store input and processed data for future reference. This is achieved by using files, which are considered the digital substitute for the paper files used to preserve physical documents.

In this chapter, we will learn about the Scanner class and the PrintStream class. Next, the concept of the StringTokenizer class will be introduced. Then, we will understand how to handle different types of files in Java including binary and text files.

INPUT/OUTPUT IN JAVA

As you know, every program needs an input to process and generate an output. Every programming language has specific keywords to accept data from the input device and to display results on the output device. Java provides Scanner and PrintStream classes to handle input and output operations, respectively. Let us learn about these classes in detail.

The Scanner Class

This is a revision tour of the Scanner class which is already discussed in Chapter 5 of this book. The Scanner class is used to read input from the keyboard for primitive data types including int, float, double, char, etc. It can also read input for non-primitive data types like String, File, InputStream(System.in) and similar classes that can execute the interfaces.

The Scanner class is defined in java.util package. So, while using the Scanner class in a program, we have to import it by using the following statement:

```
import java.util.*;
```

OR

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

Each individual piece of data read by the Scanner class is called a **token**. When data is entered through an input stream (such as the keyboard), it is not treated as a single block; instead, it is divided into smaller units called tokens.

These tokens are separated by characters known as delimiters. A delimiter is any character that marks the boundary between two tokens.

By default, the Scanner class uses whitespace as the delimiter. **Whitespace** includes characters such as space, tab (\t) and newline (\n) and these characters are generally not visible when reading input. This means that whenever a whitespace character is encountered, the input is split into separate tokens.

For example, if the input is:

Rahul 25 Delhi

Then the tokens are:

- Rahul
- 25
- Delhi

The different constructors of the Scanner class are given in the following table.

Constructor	Description
Scanner(System.in)	It reads data from the system's standard input device.
Scanner(String)	It parses a String object.
Scanner(File source)	It reads data from a specified file.

Let us use the above constructors to create an object of the Scanner class in the following way:

```
Scanner sc = new Scanner(System.in);
Scanner scan = new Scanner("Scanning a sentence separated by space");
File f = new File("STUD.txt");
Scanner inp = new Scanner(f);
```

To change or include more separators other than the default whitespace, the delimiter() method is used. Some other commonly used methods of the Scanner class are given in the following table.

Method	Description
nextInt()	It reads an integer value.
nextFloat()	It reads a float value.
nextDouble()	It reads a value of type double.
nextLong()	It reads a value of type long.
nextBoolean()	It reads a boolean value.
nextLine()	It reads a String.
next()	It reads a word.
close()	It closes a Scanner object.

Note, you have already learnt about these methods in Chapter 5 of this book.

The PrintStream Class

The PrintStream class is an important class that produces output. It converts output data of primitive type to text and writes it to the output stream. The overloaded constructors of the PrintStream class are given in the following table.

Constructor	Description
PrintStream(File file)	It creates a new print stream, without automatic line flushing, with the specified file.
PrintStream(File file, String csn)	It creates a new print stream, without automatic line flushing, with the specified file and charset.
PrintStream(OutputStream out)	It creates a new print stream.

The PrintStream class belongs to java.io package. To use this class, we need to import the java.io package by using the following statement:

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

OR

```
import java.io.PrintStream;
```

Some of the methods of the PrintStream class are given in the following table.

Method	Description
print()	It has 9 overloaded methods for printing data of different data types namely int, float, long, double, boolean, character, character array, object and String. After printing, the cursor remains in the same line.
println()	It also has 9 overloaded methods for printing data of different types as mentioned above. The println() method appends a newline character ('\n') to the output, which moves the cursor to the start of the next line.
printf()	It is used to output a formatted string to the console using various format specifiers. It includes two parameters: formatted string and arguments.

The above methods cannot be used directly with the object of the PrintStream class. Instead, we have to either use System.out.print() or System.out.println(). Let us examine the three parts separately:

- **System:** It is a final class (a class which cannot be extended or inherited) defined in java.lang package.
- **out:** It is a class variable of the PrintStream type, which is a public and static member of the System class. Since it is a static member, it is accessed using the class name; hence System.out is referenced.
- **print() or println():** These are public methods of PrintStream class. Since it is of PrintStream type, it is used to call the methods print() and println().

Input/Output Exceptions

While taking input from the user or performing file operations, errors may sometimes occur. These errors are known as **exceptions**. An exception is a problem that occurs during the execution of a program and interrupts the normal flow of the program.

For example, if a program expects an integer input but the user enters text, an InputMismatchException occurs:

```
Scanner sc = new Scanner(System.in);
int num = sc.nextInt(); // user enters "abc" → error
```

Why Do Exceptions Occur?

In input/output operations, exceptions may occur due to:

- Entering an incorrect data type (e.g., entering text instead of a number)
- Trying to access a file that does not exist
- Reaching the end of a file while reading data
- Providing input in an invalid format

Common Input/Output Exceptions

Some commonly encountered exceptions in Java related to input and output are listed below:

Exception	Description
InputMismatchException	Occurs when the type of input entered does not match the expected data type.
IOException	Occurs during general input/output operations, especially in file handling.
FileNotFoundException	Occurs when a program attempts to open a file that does not exist.
EOFException	Occurs when the end of a file is reached while reading data.

THE STRINGTOKENIZER CLASS

The StringTokenizer is a class defined under java.util package. It is used to split a String into tokens. As you know, tokens are separated by delimiters. Whitespace are the default delimiters, though other characters can be included. The methods of the StringTokenizer class do not distinguish among identifiers, numbers and quoted texts. Let us explain with the following example:

Suppose the String is, “Raj scored 92 in Maths, 88 in Physics.”

If space, comma (,) and fullstop (.) are defined as the delimiters, then the tokens are:

The StringTokenizer class handles tokens more efficiently than the String class, where a text is analysed character by character, to check for the delimiters. In the absence of the delimiters, it concatenates the characters to form a token. To use StringTokenizer class, we have to import the java.util package by using the following statement.

```
import java.util.*;
```
OR
```
import java.util.StringTokenizer;
```

The overloaded constructors of the StringTokenizer class are given in the following table.

Constructor	Purpose
StringTokenizer(String str)	It creates a StringTokenizer object with a specified string considering only the default delimiters.
StringTokenizer(String str, String delimiter)	It creates a StringTokenizer object with a specified string and a String of delimiters list.

Some of the methods of the StringTokenizer class are given in the following table.

Methods	Description
hasMoreTokens()	It checks for the availability of more tokens and returns true or false.
nextToken()	It returns the next token as a String from the StringTokenizer object.
nextToken(String delim)	It returns the next token based on the delimiter.
countTokens()	It returns the total number of tokens in the String.

Let us create some simple programs that will give you a better understanding of StringTokenizer class.

Program 1

Write a Java program to define a class named Words which will print words from a sentence having space, comma (,), question mark (?) and full stop (.) as delimiters. The class description is given as follows:

Data Members

string s, w : To store sentence and word respectively

Member Methods

void accept() : Accepts sentence

void print() : Using StringTokenizer class to print words from the sentence separated by spaces, comma, full stop or question mark

public static void main(String args[]) : Creates objects and calls the other methods

```
1   import java.util.*;
2   class Words
3   {
4       String s, w;
```

```
5       void accept() // accepting sentence
6       {
7           Scanner sc=new Scanner(System.in);
8           System.out.println("Enter sentence");
9           s=sc.nextLine();
10      }
11      void print()
12      {
13          //Creating object of StringTokenizer class
14          // and defining  space , . ? as delimiters
15          StringTokenizer st=new StringTokenizer(s," ,.?");
16          System.out.println("The words are:");
17          while(st.hasMoreTokens())  //Checking for words
18          {
19              w=st.nextToken();  //Extracting words
20              System.out.println(w);
21          }
22      }
23      public static void main(String args[])
24      {
25          // creating objects of Words class and calling methods
26          Words ob=new Words();
27           ob.accept();
28           ob.print();
29       }
30  }
```

The output of the preceding program is as follows:

BlueJ: Terminal Window - Java

Options

```
Enter sentence
Hello, my name is Divya
The words are:
Hello
my
name
is
Divya
```

Program 2

Write a Java program to define a class called Longest which will print the longest word in the sentence. The words are separated by space, comma (,), full stop (.) and question mark (?). The class description is given as follows:

Data Members

String sen	**:**	**To store sentence**
int len	**:**	**To store the length of the longest word**

Member Methods

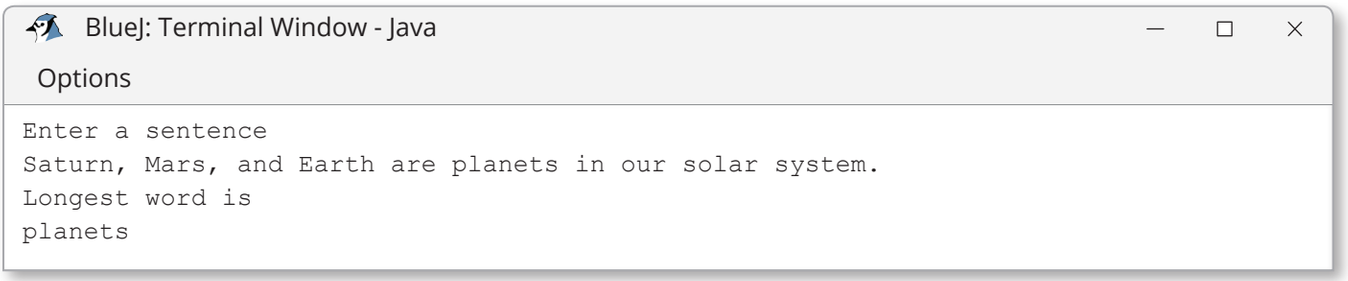
void read()	**:**	**Reads a sentence in variable sen**
void printlong()	**:**	**Prints the word with maximum length**
public static void main(String args[])	**:**	**Creates objects and calls the other methods**

```
1   import java.util.*;
2   class Longest
3   {
4       //declaring data members
5       String sen;
6       int len;
7       void read()
8       {
9           Scanner sc=new Scanner(System.in);
10          System.out.println("Enter a sentence");
11          sen=sc.nextLine();
12          len=0;
13      }
```

```
14
15      void printlong()
16      {
17          //Creating object of StringTokenizer class
18          // and defining  space , . ? as delimeters
19          StringTokenizer st=new StringTokenizer(sen," ,.?");
20          StringTokenizer st1=new StringTokenizer(sen," ,.?");
21          String w;
22          int l;
23          while(st.hasMoreTokens())  //Checking for words
24          {
25              w=st.nextToken();  //Extracting words
26              l=w.length();  // finding length of each word
27              len=Math.max(len,l); //finding word with highest length
28          }
29          // st1 object is used as we have reached the end of st object
30          System.out.println("Longest word is");
31          while(st1.hasMoreTokens()) //Checking for word
32          {
33              w=st1.nextToken();  //Extracting words
34              if(w.length()==len)  // Checking words with maximum length
35              System.out.println(w); //printing words with highest length
36          }
37      }
38
39      public static void main(String args[])
40      {
41          Longest ob=new Longest();
```

```
42              ob.read();
43              ob.printlong();
44          }
45      }
```

The output of the preceding program is as follows:



If more than one word has the highest length, which means two or more words have the same length, then the preceding program would return all the words having the highest length.

DATA FILE HANDLING

Till now, we have used variables to store values. These values get lost when a program's execution is stopped. Sometimes, we face a situation in which we need to store a value or several values permanently for future reference. For example, when an employee joins a particular organisation, the organisation collects data about the employee like name, father's name, mother's name, date of birth, gender, etc. and stores this data in the organisation's information system permanently. Similarly, organisation has collected data of thousands of employees. So, to handle large amounts of data, Java provides the concept of files. The process of handling files is called file handling.

A file is a named location that can be used to store related data permanently in a secondary storage device. Files are stored in an organised manner in directories. There are different types of files and formats available on computers as described in the following table.

File Type	Description
Image files	These files store images or graphics. Image files can be of different types namely .jpeg, .gif, .png, .tif, .svg, etc.
Document files	These files are used to create, edit and print documents. There are different types of document files like .doc, .pdf, .txt, etc.
Video files	These files are stored in file formats like .mp4, .avi, .mov, .flv, etc.
Audio files	These files are used to store songs, lectures, etc. in different audio file formats like .mp3, .wav, etc.
Database files	These files store related records in an organised manner. There are various types of database files like .mdb, .csv, .xls, .sql, .idx, etc.
Program files	These files are used to store code written in different programming languages. There are different types of program files like .java, .bas, .cpp, .py, etc.

Data files are the files that contain data in the form of records. A record is further divided into fields. Data files are non-executable files that can be created, read or viewed. In some data files, data is stored in the form of text but some other data files store data in a more organised manner as records.

The different subdivisions are:

- **Field:** It is the smallest unit of data that can be accessed by a user. It is identified by a unique name called field name.
- **Record:** It is a collection of related fields.
- **File:** Files (data files) are a collection of related records.

For example, a sample data file "Application.dat" that stores the details of candidates applying for an interview for the post of a programmer is given below.

Fields

Application_Number	Applicant_Name	Mobile_No	Mail_ID
P100101	Ajay Nanda	9876501210	ajay.n@gmail.com
P100102	Geeta Suresh	7665743781	geeta.pune@yahoomail.com
P100103	Akash Singh	7453523544	akash.b2002@rediffmail.com

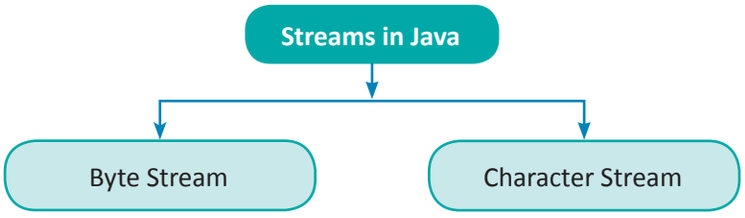
Record

In Java, mainly two types of files are used to store data which are as follows:

- **Text Files:** The files that store data in the form of human-readable text are known as text files. These files contain data as a sequence of characters in the form of ASCII or UNICODE characters. A text file is identified by .txt extension. In this type of file, each line is terminated by an End of Line (EOL) character.
- **Binary Files:** Binary files are not in human-readable form. These are machine-readable files. A binary file stores data in the form of 0s and 1s. There is no need to put a comma, space or an end-of-line character in the binary files.

INTRODUCING STREAMS

In Java, a sequence of data is called a stream. In other words, a stream can be defined as a path along which the data travels from an input device to a program and from a program to an output device. It is a communication medium through which the data is stored and retrieved. The stream used to read data from various input devices is known as the input stream. The stream used to write data to various output devices is known as the output stream. These streams are used to perform input/output operations on files. The java.io package provides all the classes to perform these operations. Streams in Java are of two types:

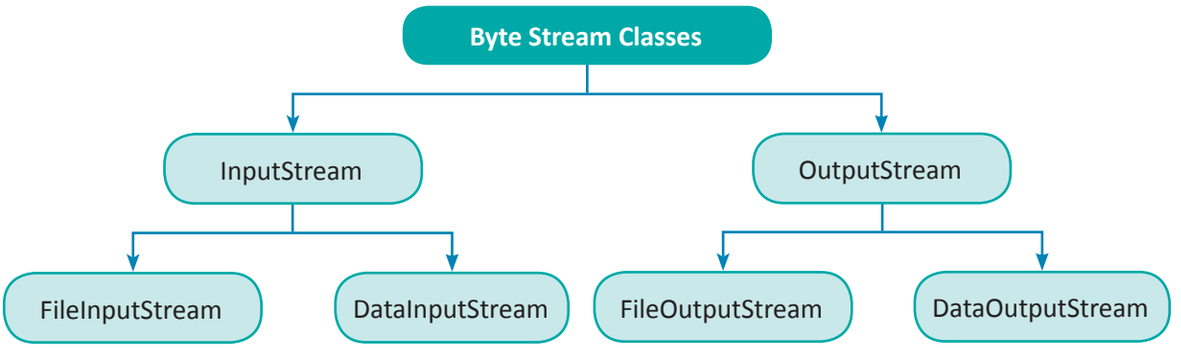


Let us learn about these in detail.

Byte Stream

Byte-oriented input/output is performed on binary files that store data in machine-readable form. Byte stream class handles data in bytes. Java has two abstract classes which are the superclasses of all other classes used to read or write a stream of bytes.

These two classes are as follows:



The InputStream Class

The InputStream is an abstract class which means it cannot be instantiated. Various classes inherit the InputStream class and override its methods. Let us learn about two main classes: FileInputStream and DataInputStream which inherit the InputStream class. These classes are used to read input from a standard input device like keyboard, mouse, memory and any other program.

The FileInputStream Class

The FileInputStream class has methods to read bytes from a file. It is meant for reading streams of raw bytes such as image data. The following table shows the description of the commonly used methods of the FileInputStream class.

Method	Description
close()	It closes the current file input stream.
read()	It reads a byte of data from this input stream.

The DataInputStream Class

The DataInputStream class has methods to read primitive data types from an underlying input stream. It is not directly inherited from the InputStream class. It is inherited from the another input stream class. The following table shows the description of the commonly used methods of the DataInputStream class.

Method	Description
read()	It reads the number of bytes from the input stream.
readBoolean()	It reads one input byte and returns true if that byte is nonzero, false if that byte is zero.
readByte()	It reads and returns one input byte.
readChar()	It reads two input bytes and returns a char value.
readDouble()	It reads eight input bytes and returns a double value.
readFloat()	It reads four input bytes and returns a float value.
readInt()	It reads four input bytes and returns an int value.
readLong()	It reads eight input bytes and returns a long value.
readShort()	It reads two input bytes and returns a short value.
readUTF()	It reads from the stream in a representation of a Unicode character string.

The OutputStream Class

The OutputStream class is also abstract in nature. So, we cannot create its objects. But the classes derived from it can write a stream of bytes. Let us learn about two main classes named FileOutputStream and DataOutputStream which inherit the OutputStream class.

The FileOutputStream Class

The FileOutputStream class has methods to write bytes to a file. The most commonly used methods of the FileOutputStream class are listed in the following table.

Method	Description
close()	It closes the current file output stream and releases any system resources associated with this stream.
write()	It writes the specified byte to the file output stream. There are two overloaded methods for write() method.

The DataOutputStream Class

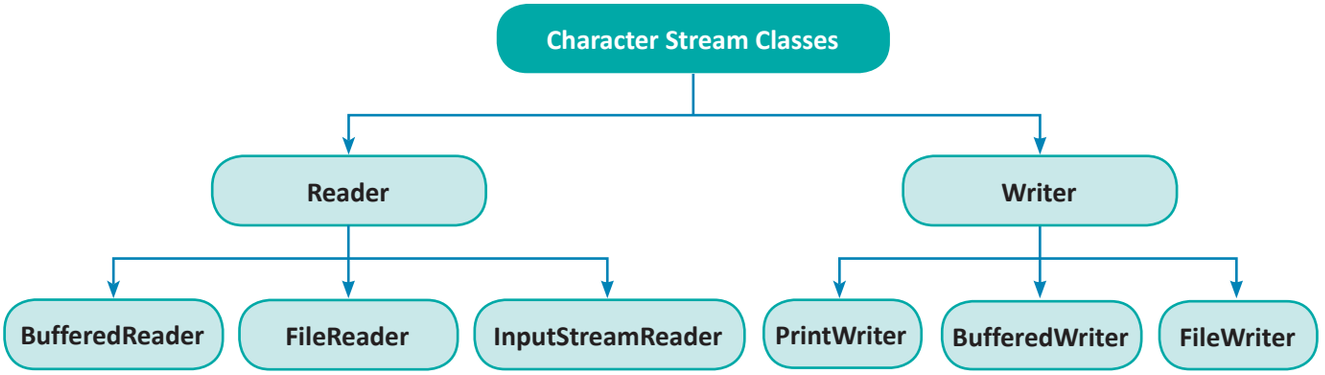
The DataOutputStream class has methods to write primitive data types to a file. The most commonly used methods of the DataOutputStream class are listed in the following table:

Method	Purpose
writeByte()	It writes a byte to an output stream as a 1-byte value.
writeBytes()	It writes a string to an output stream as a sequence of bytes.
writeChar()	It writes a char to an output stream as a 2-byte value.
writeChars()	It writes a string to an output stream as a sequence of characters.
writeFloat()	It writes float-type data to an output stream.
writeInt()	It writes integer data to an output stream.
writeDouble()	It writes double-type data to an output stream.
writeLong()	It writes long-type data to an output stream.
writeShort()	It writes a short to the underlying output stream as two bytes, high byte first.
writeUTF()	It writes String data to an output stream.

Character Stream

The character stream classes are used to create text files. These classes can handle 16-bit Unicode characters. They are more advanced than byte stream classes as the byte stream classes can only handle 8-bit bytes and are not compatible with the Unicode characters.

The input and output operations are handled by abstract classes named Reader and Writer respectively belonging to java.io package. However, these classes cannot be directly used in the program. Some other classes which are inherited from these classes are used to perform different read/write operations. The following diagram shows the different classes inherited from the Reader and Writer classes.



Let us learn about them in detail.

Reader Class

The Reader class is abstract in nature which means it cannot be instantiated but it has subclasses that override its methods and can read 16-bit characters from InputStream.

Some important classes that are inherited from the Reader class are described in the following table.

Class	Description
BufferedReader	It has methods to read characters from the buffer.
FileReader	It has methods to read characters from the file.
InputStreamReader	It has methods to convert byte streams to character streams.

Writer Class

The Writer class is also an abstract base class whose methods are implemented by its derived classes. Some of the derived classes of the Writer class are described in the following table.

Class	Description
BufferedWriter	It provides methods to write characters to the buffer.
FileWriter	It provides methods to write characters to the file.
PrintWriter	It enables to print the formatted representation of objects to the text-output stream. PrintWriter class has two overloaded methods: • **print()**: Writes data on the same line. • **println()**: Adds a new line character after writing data.

OPERATIONS ON FILES

Java allows us to perform various types of operations such as creating, writing, reading, appending and searching data in text files. Let us learn about them in detail.

Creating and Writing Data to a Text File

To create and write data to a text file, the following steps are to be followed:

Step 1: Create an object of the FileWriter class and connect it with the physical file on the disk. If the file does not exist, then it is created by default. If it is already present on the disk then its previous contents are overwritten.

```
FileWriter <fileobject> = new FileWriter("File name");
```

Step 2: Connect the FileWriter object with the BufferedWriter object. Buffers are temporary storage areas where the output stream is first stored. Data in the buffer is generally transferred to the file when the buffer is full or during the closing of the stream objects.

```
BufferedWriter <bufferobject> = new BufferedWriter(fileobject);
```

Step 3: Now, link the BufferedWriter object with the PrintWriter object for writing characters to the text file.

```
PrintWriter <printobject> = new PrintWriter(bufferobject);
```

Step 4: Write the text onto the file by using print() or println() method.

```
<printobject>.print(text) or <printobject>.println(text);
```

Step 5: Lastly, close all the stream objects using the close() method.

Appending Data to a Text File

Appending data to a text file is similar to writing data. The only difference is that the file is opened in append mode by passing true as a parameter to the FileWriter object. In this mode, new data is added at the end of the file without overwriting the existing contents.

```
FileWriter <fileobject> = new FileWriter("File name", true);
BufferedWriter <bufferobject> = new BufferedWriter(fileobject);
PrintWriter <printobject> = new PrintWriter(bufferobject);
<printobject>.println(text);
<printobject>.close();
```

Reading and Searching Data from a Text File

To read and search data from a text file, the FileReader class is used along with BufferedReader. The text is read line by line using the readLine() method. This method returns null when the end of the file is reached. While reading, each line can also be compared with the required data. If the required data is found, it is displayed.

```
FileReader <fileobject> = new FileReader("File name");
BufferedReader <bufferobject> = new BufferedReader(fileobject);
String <variable>;
while((<variable> = <bufferobject>.readLine()) != null)
{
    System.out.println(<variable>); // for reading
    if(<variable>.contains("search text")) // for searching
    {
        System.out.println(<variable>);
    }
}
<bufferobject>.close();
```

Program 3

Write a Java program to define a class called Flowers to perform read and write operations on a text file "Flower.txt". The detail of the class is given as follows:

Data Members

String fname : To store names of flowers

Member Methods

void createFile() : Accepts names of flowers and stores them in a text file "Flower.txt"

void readFile() : Reads text file "Flower.txt" and prints names of those flowers that start with vowels

public static void main(String args[]) : Creates object and executes other methods

```
1   import java.io.*;
2   import java.util.*;
3   class Flowers
4   {
5       String fname; // data member
6       void createFile() throws IOException
7       {
8           // declaring the classes required to create file
9           // parameter true opens file in append mode
10          FileWriter fw=new FileWriter("Flower.txt");
11          BufferedWriter bw=new BufferedWriter(fw);
12          PrintWriter pw=new PrintWriter(bw);
13          Scanner sc=new Scanner(System.in);
14          char ch='y';
15          while(ch=='y' || ch=='Y')   // loop to store multiple records
16          {
17              System.out.println("Enter the name of the flower");
18              // entering data using Scanner class
19              fname=sc.next();
20              // writing data to buffer using specific write methods of data type
21              pw.println(fname);
22              System.out.print("Continue y/n");
23              ch=sc.next().charAt(0);
```

```
24              }
25              // closing the objects transfers data in buffer to file permanently
26              pw.close();
27              bw.close();
28              fw.close();
29          }
30          void readFile() throws IOException
31          {
32              // declaring the classes required to read file
33              FileReader fr=new FileReader("Flower.txt");
34              BufferedReader br=new BufferedReader(fr);
35              // Checking for end of file
36              while((fname=br.readLine())!=null)
37              {
38                  char c=fname.charAt(0);  // Extracting the first character
39                  if("AEIOUaeiou".indexOf(c)!= -1)    // Checking for vowels
40                  System.out.println(fname);
41              } // end of while
42              // closing stream objects
43              br.close();
44              fr.close();
45          }
46          // main() method to call other methods
47          public static void main(String args[]) throws IOException
48          {
49              Flowers ob = new Flowers();
50              ob.createFile();
51              ob.readFile();
52          } // end of main
53      } // end of class
```

The output of the preceding program is as follows:

Program 4

Write a Java program to define a class named Poems to perform append and searching operations on a text file "Poem.txt". The details of the class are as follows:

Data Members

String pline	**:**	**To store lines of a poem**
Member Methods		
void appendFile()	**:**	**Accepts five lines of a poem and appends them to a text file "Poem.txt"**
void searchWord()	**:**	**Reads text file "Poem.txt" and counts how many times the word "divine" is present in the lines**
public static void main(String args[])	**:**	**Creates object and executes other methods**

```
1   import java.io.*;
2   import java.util.*;
3   class Poems
4   {
5       String pline; // data member
6       void appendFile() throws IOException
7       {
8           FileWriter fw=new FileWriter("Poem.txt",true);
9           BufferedWriter bw= new BufferedWriter(fw);
10          PrintWriter pw = new PrintWriter(bw);
```

```
11          Scanner sc=new Scanner(System.in);
12          System.out.println("Enter 5 lines of a poem:");
13          for(int i=1;i<=5;i++)   // loop to store multiple records
14          {
15              pline=sc.nextLine();
16              // writing data to buffer using specific  write methods of data type
17              pw.println(pline);
18              sc=new Scanner(System.in);
19          }
20          // closing the objects transfers data in buffer to file permanently
21          pw.close();
22          bw.close();
23          fw.close();
24      }
25      void searchWord() throws IOException    {
26          int count=0;
27          // declaring the classes required to read file
28          FileReader fr=new FileReader("Poem.txt");
29          BufferedReader br=new BufferedReader(fr);
30          StringTokenizer st;
31          // Checking for end of file
32          while((pline=br.readLine())!=null)
33          {
34              st=new StringTokenizer(pline," .?!,");
35              // creating object of StringTokenizer class
36              while(st.hasMoreTokens())
37              {
38                  String w = st.nextToken();      // Extracting words form the file
39                   if(w.equalsIgnoreCase("divine"))     // Checking for divine
40                       count++;
```

```
41                  }
42              } // end of while
43              System.out.println("Divine occurs "+count+" times");
44              // closing stream objects
45              br.close();
46              fr.close();
47          }
48          // main() method to call other methods
49          public static void main(String args[]) throws IOException
50          {
51              Poems ob = new Poems();
52              ob.appendFile();
53              ob.searchWord();
54          } // end of main
55      }  // end of class
```

The output of the preceding program is as follows:

BlueJ: Terminal Window - Java

Options

```
Enter 5 lines of a poem:
The divine breeze
The blooming trees
The divine sound of humming bees
I am lost in nature
Divine bliss.
Divine occurs 3 times
```

Let's Revisit

- The Scanner class is defined in java.util package.
- Each individual piece of data read by the Scanner class is called a token.
- StringTokenizer is a class which is used to split a String into tokens.
- The StringTokenizer class handles tokens more efficiently than the String class.
- Data files are the files that contain data in the form of records.
- Java allows us to perform various types of operations such as creating, writing, reading, appending and searching data in text files.

MIND DRILL

Solved Questions

A. Tick (✓) the correct option.

1. Identify the Java package that includes the Scanner class.
   a. java.io
   b. java.util
   c. java.net
   d. java.lang
2. Which of the following methods is used to scan int type data in the Scanner class?
   a. nextInteger()
   b. NextInt()
   c. nextInt()
   d. nextUTF
3. Select the characters that separate tokens.
   a. Delimiters
   b. Carriage return
   c. Escape sequence characters
   d. New line
4. Choose the character that the println() method adds after printing.
   a. null
   b. '\0'
   c. '\r'
   d. '\n'
5. Which of the following is the return type of the countTokens() method in StringTokenizer class?
   a. boolean
   b. int
   c. String
   d. char

Answers

1. b 2. c 3. a 4. d 5. b

B. Fill in the blanks.

1. ..... class has the print() method to write data to a text file.
2. ..... method is used in the Scanner class to input boolean data.
3. System is a final ..... defined in java.lang package.
4. ..... method of StringTokenizer class returns the next token.
5. .mp4, .mov, .flv are files of ..... type.

Answers

1. PrintWriter 2. nextBoolean() 3. class 4. nextToken()
5. Video

C. Answer the following questions:

1. Write the different methods of the Scanner class to scan input.

Ans. The different methods of Scanner class are:

Method	Purpose
nextInt()	It reads an integer value.
nextFloat()	It reads a float value.
nextDouble()	It reads a value of type double.
nextLong()	It reads a value of type long.
nextBoolean()	It reads a boolean value.
nextLine()	It reads a string.
next()	It reads a word.

2. Write the steps to read data from a text file.

Ans. To read data from a text file, the following steps are to be followed:

Step 1: Create an object of FileReader class and connect it with the physical file on the disk.

```
FileReader <fileobject> = new FileReader("File name");
```

Step 2: Connect the FileReader object with the BufferedReader object. Output buffers are temporary storage areas that store a chunk of data read from the file. When the buffer is full, it is emptied and the next chunk of data is stored.

```
BufferedReader <bufferobject> = new BufferedReader (fileobject);
```

Step 3: The text is read by invoking the method readLine() using BufferedReader object and stored in a String variable. This object call should be inside the loop body which will return null, if the file does not exist or if the end of file is reached.

```
String <variable> = <bufferobject>.readLine();
```

Step 4: Lastly, close all the stream objects using the close() method.

3. Define the purpose of the StringTokenizer class in Java. List its constructors and their purposes.

Ans. The StringTokenizer is a class defined under java.util package. It is used to split a String into tokens. The overloaded constructors of the StringTokenizer class are given in the following table.

Constructor	Purpose
StringTokenizer(String str)	It creates a StringTokenizer object with a specified string considering only the default delimiters.
StringTokenizer(String str, String delimiter)	It creates a StringTokenizer object with a specified string and a String of delimiters list.

4. Explain the three parts of System.out.println().

Ans.
- **System:** It is a final class (a class that does not allow other classes to inherit its features) defined in java.lang package.
- **out:** It is a class variable of PrintStream type, which is a public static member of the System class. Since it is a static member, it is accessed using the class name, hence System.out is referenced.
- **print() or println():** These are public methods of PrintStream class. Since out is of PrintStream type, it is used to call the methods print() and println().

5. Describe the purpose of the Scanner class with an example.

Ans. The Scanner class is used to read input from the keyboard for primitive data types including int, float, double, char, etc. It can also read input for non-primitive data types like String, File, InputStream(System.in) and similar classes that can execute the interfaces.

The Scanner class is defined in java.util package.

For example,

```
import java.util.Scanner;  class Main
{
  public static void main(String[] args)
  {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();   System.out.println("Username is: " + userName);
  }
}
```

D. Higher Order Thinking Skills (HOTS)

1. In a text-processing application, you need to extract meaningful tokens from a sentence that contains both words and numbers. Given that spaces, commas and full stops are the delimiters, Which class can you use to efficiently extract the tokens? What would the resulting tokens be for the string "Raj scored 92 in Maths, 88 in Physics"?

Ans. You can use StringTokenizer class to efficiently extract tokens by specifying delimiters like spaces, commas and full stops. It breaks the given string into smaller tokens based on these delimiters.

For example,

```
import java.util.StringTokenizer;
public class Main
{
    public static void main(String[] args)
   {
        String str = "Raj scored 92 in Maths, 88 in Physics";
        StringTokenizer st = new StringTokenizer(str, " ,.");
        while (st.hasMoreTokens())
        {
            System.out.println(st.nextToken());
        }
    }
}
```

2. Aditya has been assigned to create a program that can store employee data, such as name, gender and date of birth. However, he is struggling to store information for all the employees of the company at this scale. How should the data be saved for future reference in an organised manner?

Ans. Aditya should save the employee data in a text file using Java's file handling classes like FileWriter or PrintWriter. This allows data to be stored persistently on disk in an organized manner, such as in CSV format (comma-separated values), where each employee's details are stored on a separate line.

For example,

```
import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.IOException;
public class EmployeeData {
    public static void main(String[] args) throws IOException {
        PrintWriter writer = new PrintWriter(new FileWriter("employees.txt"));
        writer.println("Name, Gender, DOB");
        writer.println("Raj, Male, 15-08-1990");
        writer.println("Priya, Female, 22-03-1995");
        writer.println("Amit, Male, 10-11-1988");
        writer.close();

        System.out.println("Employee data saved successfully!");
    }
}
```

E. Assertion and reasoning questions.

The following questions consist of two statements – Assertion (A) and Reason (R). Answer these questions by selecting the appropriate option given below:

a. Both A and R are true and R is the correct explanation of A.

b. Both A and R are true but R is not the correct explanation of A.

c. A is true but R is false.

d. A is false but R is true.

1. Assertion (A): File handling is used to store data permanently for future reference.

   Reason (R): A file is a named location that stores related data in secondary storage devices and is organized in directories.

Ans. a. Both A and R are true and R is the correct explanation of A.

2. Assertion (A): A record is the smallest unit of data in file handling that can be accessed.

   Reason (R): A field is the smallest unit of data that can be accessed and it is identified by a unique field name.

Ans. d. A is false but R is true.

3. Assertion (A): A binary file stores data in human-readable form using ASCII or UNICODE characters.
   Reason (R): Binary files are machine-readable files that store data in 0s and 1s, not in human-readable form.

Ans. d. A is false but R is true.

F. Case study-based question.

Rohan is developing a program that stores student marks in a text file named "marks.txt". The file contains data such as roll number, student name and marks for various subjects. The program uses FileReader and BufferedReader to read the data.

Based on the given case, answer the following questions:

1. Which class is used to read text data from the "marks.txt" file?
   a. FileOutputStream
   b. FileReader
   c. DataOutputStream
   d. BufferedReader
2. What is the primary purpose of using BufferedReader in the program?
   a. To read binary data
   b. To handle file errors
   c. To write data to the file
   d. To read text data
3. If the "marks.txt" file does not exist, what will happen when the program tries to read from it?
   a. FileNotFoundException
   b. NullPointerException
   c. EOFException
   d. ArithmeticException
4. Which method in BufferedReader would you use to read a line of text from the "marks.txt" file?
   a. readLine()
   b. read()
   c. readInt()
   d. readDouble()

Answers

1. b 2. d 3. a 4. a

G. Identify the error in the given codes and rewrite the correct code:

1.
```
import java.util.Scanner;
import java.io.PrintStream;
class IOExample {
    public static void Main(String[] args) {
        Scanner sc = new Scanner(System.in);
        PrintStream ps = System.out;
        ps.print("Enter your name: ");
        String name = sc.nextLine();
        ps.print("Enter your age: ");
        int age = sc.nextInt();
        ps.println("Hello " + name + ", you are " + age + " years old.");
        sc.close();
    }
}
```

Ans.
```
import  java.util.Scanner;
import java.io.PrintStream;
class IOExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        PrintStream ps = System.out;
        ps.print("Enter your name: ");
        String name = sc.nextLine();
        ps.print("Enter your age: ");
        int age = sc.nextInt();
```

```
            ps.println("Hello " + name + ", you are " + age + " years old.");
            sc.close();
        }
    }
```

2.
```
class TokenizerExample {
    public static void main(String[] args) {
        String sentence = "Java programming is fun and educational.";
        StringTokenizer st = new StringTokenizer(sentence, " ");
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
}
```

Ans.
```
import java.util.StringTokenizer;
class TokenizerExample {
    public static void main(String[] args) {
        String sentence = "Java programming is fun and educational.";
        StringTokenizer st = new StringTokenizer(sentence, " ");
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
}
```

3.
```
import java.io.*;
import java.util.Scanner;
class StudentData
{
    public static void main(String[] args)
    {
        FileData file = new FileData("studentData.txt");
        Scanner sc = new Scanner(System.in);
        try
        {
            FileWriter fw = new FileWriter(file, true);
            BufferedWriter bw = new BufferedWriter(fw);
            System.out.print("Enter student name: ");
            String name = sc.nextLine();
            System.out.print("Enter student marks: ");
            double marks = sc.nextDouble();
            bw.write("Student Name: " + name + ", Marks: " + marks);
            bw.newLine();
            bw.close();
            fw.close();
        }
        catch (IOException e)
        {
            System.out.println("An error occurred: " + e.getMessage());
        }
```

```
        }
    }
```

Ans.

```
import java.io.*;
import java.util.Scanner;
class StudentData
{
    public static void main(String[] args)
    {
        String file = "studentData.txt";
        Scanner sc = new Scanner(System.in);
        try
        {
            FileWriter fw = new FileWriter(file, true);
            BufferedWriter bw = new BufferedWriter(fw);
            System.out.print("Enter student name: ");
            String name = sc.nextLine();
            System.out.print("Enter student marks: ");
            double marks = sc.nextDouble();
            bw.write("Student Name: " + name + ", Marks: " + marks);
            bw.newLine();
            bw.close();
            fw.close();
        }
        catch (IOException e)
        {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

Unsolved Questions

A. Tick (✓) the correct option.

1. StringTokenizer class belongs to the ..... package in Java.
   a. java.io
   b. java.util
   c. java.net
   d. java.lang
2. Which of the following methods is used to scan double type data in the Scanner class?
   a. nextDouble()
   b. NextDouble()
   c. nextdouble()
   d. nextUTF()
3. Which of the following is the default delimiter for StringTokenizer class?
   a. Null
   b. White space
   c. Escape sequence characters
   d. New line
4. In print() method, after printing cursor remains in the ..... line.
   a. different
   b. front
   c. same
   d. back
5. Which of the following is the return type of hasMoreTokens() method in StringTokenizer class?
   a. boolean
   b. int
   c. String
   d. char

B. Fill in the blanks:

1. The ..... method is used in the Scanner class to input long type data.
2. The ..... method is used in StringTokenizer class to count the number of tokens.
3. Sorting of records can be done in ..... and ..... order.
4. A ..... parameter opens a file in append mode.
5. ..... class handles data in bytes.

C. Answer the following questions:

1. Define the following methods of the StringTokenizer class:
   a. hasMoreTokens()
   b. nextToken()
   c. countTokens()
2. Define exceptions. List some common reasons why they occur.
3. Write the steps to write data to a text file.
4. List the commonly used methods of the DataInputStream class and explain the purpose of each method.
5. Differentiate between a field and a record in a data file.

D. Higher Order Thinking Skills (HOTS)

1. How would you decide when to use a text file versus a binary file for storing employee data?
2. Advik is tasked to write a Java program that reads a list of employee names and IDs from a file, but it throws an exception due to incorrect data formatting in the input file. How would he handle the input/output exception that arises when the program encounters invalid data?

E. Case study-based question.

Ravi is developing a program that takes user input for student grades and stores them in a file. While reading the data from the file, the program encounters various Input/Output exceptions. For example, if the user enters an incorrect data type (like entering text when an integer is expected), an InputMismatchException occurs. Similarly, if the file is missing or has incorrect data formatting, exceptions such as IOException or FileNotFoundException are raised. To ensure the program continues running smoothly, Ravi decides to implement exception handling using try-catch blocks.

Based on the given case, answer the following questions:

1. Which of the following exceptions occurs when the user enters a non-integer value (like text) when an integer is expected?
   a. IOException
   b. FileNotFoundException
   c. InputMismatchException
   d. ArithmeticException
2. What is the primary purpose of using try-catch blocks in the program when handling input/output exceptions?
   a. To prevent the program from reading data
   b. To handle exceptions
   c. To close all open files
   d. To displaying messages
3. If the program attempts to open a file that does not exist, which exception will be thrown?
   a. EOFException
   b. IOException
   c. InputMismatchException
   d. FileNotFoundException
4. When an EOFException occurs, what does it indicate about the program's file reading operation?
   a. When the end of a file is reached
   b. The file contains invalid data.
   c. The program has successfully read the entire file.
   d. The file is not accessible due to permission issues.

F. Find the output of the following programs:

1.
```
import java.util.Scanner;
public class InputExceptionHandling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = "abc";
        try
        {
            System.out.print("Enter an integer: ");
```

```
            int num = Integer.parseInt(input);
            System.out.println("You entered: " + num);
        }
       catch (NumberFormatException e)
       {
        System.out.println("Invalid input!");
        } finally {
            sc.close();
        }
    }
}
```

2.
```
import java.util.StringTokenizer;
public class StringTokenizerExample
{
    public static void main(String[] args)
    {
        String str = "Aarav, Priya, Rahul. Neha, Rohan.";
        StringTokenizer st = new StringTokenizer(str, " ,.");
        System.out.println("Tokens extracted from the sentence:");
        while (st.hasMoreTokens())
       {
            System.out.println(st.nextToken());
        }
    }
}
```

3.
```
import java.io.*;
public class DataFileHandling {
    public static void main(String args[]) {
        try {
            FileWriter fw = new FileWriter("employeeData.txt");
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write("Employee ID: 1001, Name: John, Department: HR\n");
            bw.write("Employee ID: 1002, Name: Sarah, Department: IT\n");
            bw.close();
            fw.close();
            System.out.println("Data written successfully!");
        } catch (IOException e) {
            System.out.println("An error occurred while writing.");
        }
    }
}
```

G. Write the java program for the following:

1. Define a class Bank having the following class description:

Data Members

AccountNo	:	To store account number of customer type integer
Name	:	To store account holders name of String type
Principal	:	To store the amount in the bank
Pan	:	To store the Pan number of the customer

Member Methods

void createFile()	:	Creates a binary file BANK.txt having the above fields
void printRec()	:	Print the details of the customers
void withdraw()	:	Updates balance when the amount is withdrawn from the account
void deposit()	:	Updates balance when the amount is deposited to the account
static void main()	:	Writes a menu-driven program to perform the following file operations

2. Define a class VoteFinder having the following class description:

Data Members

FileName	:	To store the name of the file of String type
SearchWord	:	To store the word to search for of String type

Member Methods

VoteFinder(String fileName, String searchWord)	:	Constructor to initialize the file name and the word to search
void readFile()	:	Reads the file line by line and displays lines containing the search word (case-insensitive search)
void displayLines(String line)	:	Prints the line that contains the search word
public static void main(String args[])	:	Creates an object of VoteFinder to search for the word "vote" in the file "Elections. txt"