Units : using I/O

5.1 Console and file I/O

Input/output (I/O) is used to receive data from the user and display or store results.

Console I/O

Console I/O means reading input from the keyboard and displaying output on the screen.

↳ System.out.println() method is used to print output with a new line

↳ System.out.print() ⇒ print output without a new line.

↳ System.in → used to take input from keyboard.

↳ Scanner class is used to accept data entered by the user which is belongs to the java.util package.

Example

```
import java.util.Scanner;
public class inputexample
{
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
System.out.print("Enter your name: ");
String name = sc.nextLine();
System.out.println("Hello" + name);
```

```
}
```

output:

```
Enter your name: Rajesh
Hello Rajesh
```

file I/O

file I/O is used to store data in files and read data from files.

Java provides file input/output (I/O) - classes in the java.IO package to read and write data from files.

These classes are mainly classified into two types

Note:
input means Read
output means write

file I/O Classes

Byte Stream Classes

- fileInputStream
- BufferedInputStream
- DataInputStream
- fileOutputStream
- BufferedOutputStream
- DataoutputStream

Base Classes
- InputStream
- outputstream

Character Stream Classes

- fileReader
- fileWriter
- BufferedReader
- BufferedWriter
- printwriter

Base Classes
- Reader
- Writer

5.2 opening and closing files

Before reading or writing data, a file must be opened.

<u>opening a file</u>:- files can be opened using classes like:

→ fileInputstream
→ fileOutputStream
→ fileReader
→ filewriter

<u>Example of opening a file in java</u>

```
fileInputstream file = new fileInputStream("data.txt");
```

<u>closing a file</u>:- After finishing operations, the file should be closed to free system resources.

<u>example</u>

```
file.close();
```

closing files prevents data loss and memory leaks.

Q. Program example in java to open and close the file

```
import java.io.*;
public class fileopenclose
{ public static void main(String[] args) throws
                                        Exception
{
fileInputstream file = new fileInputstream
                        ("data.txt");

file.close();
}}
```

Q. open and close file using try-catch

```
import java.io.*;
public class fileExample
{
    public static void main(String[] args)
    {
        try
        {
            FileReader fr = new FileReader("data.txt");
            System.out.println("file opened successfully");
            fr.close();
            System.out.println("file closed successfully");
        }
        catch (Exception e)
        {
            System.out.println("Error: " + e);
        }
    }
}
```

3 3 3

5.3 Scanner class

The Scanner class is used to take input from the user. It belongs to java.util.scanner package.

Syntax to Creating Scanner object

```
Scanner sc = new Scanner(System.in);
int num = sc.nextInt();
```

methods for different datatypes

nextInt() → Reads integer
nextDouble() → Reads double
next() → Read single word
nextLine() → Reads full line.

5.4 Byte Streams and Character Streams

Java I/O streams are divided into two types:-

S.No	Byte Streams	Character Streams
①	Handles binary data (8-bit bytes)	Handles text data (16-bit characters)
②	used for images, audio, video and binary files.	used for text files (.txt)
③	Classes inherit from InputStream / OutputStream	classes inherit from Reader / writer
④	Reads / write data byte by byte.	Reads / writes data character by character.
⑤	Example Classes: fileInputStream, fileOutputstream, BufferedInputstream Buffered output stream	Example classes: fileReader, filewriter, BufferedReader, BufferedWriter
⑥	cannot directly handle unicode text	Can handle unicode text properly.
⑦	faster for binary data.	optimized for text data.

5.5 Reading and writing Byte Streams

Byte Streams read and write binary data.

Byte streams handle data byte by byte (8 bits).

Byte Streams are mainly used for binary files like images, audio, videos, PDFs, etc.

Byte streams classes are:-

↳ InputStream
↳ outputStream
↳ fileInputstream
↳ fileoutputstream

import 'java.io.*; is used to import all classes of java I/O package so that we can perform input and output ~~strea~~ operations like reading and writing files.

instead of importing java.io.fileInputstream;
java.io fileoutputstream;
we can simply import java.io.*;

program Example to write and then read from file.

```
import 'java.io.*;
class fileExample
{
  public static void main (string[] args)
  {
    try
    {
      // writing to file using Bytestream
      fileOutputstream fout = new fileOutputStream("data.txt");
      String text = "Hello java student";
      byte b[] = text.getBytes();
      fout.write(b);
      fout.close();
      System.out.println("Data written successfully.\n");
      //Reading from file
      fileInputstream fin = new fileInputstream("data.txt");
      int i;
```

```
System.out.print("Reading from file: ");
while((i = fin.read()) != -1)
 {
  System.out.print((char) i);
 }
  fin.close();
}
 catch(Exception e)
  {
   System.out.println("Error:" + e.getMessage());
  }
}}
```

output:

Data written successfully.

Reading from file: Hello Java student.

If incase of file Not found

output: Error: data.txt (The system cannot find the file specified)

Another Example of Integer data

```
import java.io.*;
public class example{
 public static void main(String[] args)
  {
   try
    {
     fileoutputStream fout = new fileOutputStream("data1.txt");
     fout.write(65);
     fout.close();
     System.out.println("Integers written to file.");
     fileInputStream fin = new fileInputStream("data1
                                                .txt");
     int data;
```

```
System.out.println("Reading integers from file:");
 while ((data = fin.read()) != -1)
  {
    system.out.println(data);
  }
   fin.close();
}
 catch (IOException e)
   {
    system.out.println("Error:" + e.getMessage());
   }
}}
```

output:

Integers written to file.
Reading Integers from file:
65

Catch output (file Not found)
Error: ~~data1~~.txt (The system cannot find the file specified).

Example 3

Reading and writing an Image (Binary file)

```
import java.io.fileInputStream;
import java.io.fileOutputStream;
import java.io.IOException;
public class ImageExample
{
public static void main (string [] args)
  {
    try
    {
      fileInputStream fin = new fileInputStream("photo.jpg");
```

```
FileOutputStream fout = new FileOutputStream("copy_photo.jpg");
 int bytedata;
 while ((bytedata = fin.read()) != -1)
  {
    fout.write(bytedata);
  }
    fin.close();
    fout.close();
    System.out.println("Image copied successfully!");
  }
  catch (IOException e)
  {
    System.out.println("Error: " + e.getMessage());
  }
}
}
```

5.6 Reading and writing character streams

Character streams are used to read and write text data (characters). Character streams work with 16-bit unicode characters.

Character streams are in the java.io. package

Base/super classes

↳ Reader :- for reading characters from file
↳ writer :- for writing characters into file.

Reader classes	writer classes
① fileReader	① filewriter
② BufferedReader	② Buffered writer
③ InputStreamReader	③ print writer

Q. suppose the file numbers.txt contains:

10 20 30 40

Program

```
import java.io.*;
Public Class ReadNumbers
{ public static void main (string [] args)
{
try
{ fileReader fr = new fileReader ("numbers.txt");
int ch;
while (ch = fr.read ()) != -1)
{ System.out.print ((char)ch);
}
```

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

output 10 20 30

Q. Suppose a text file named data.txt with the content

my name is rajesh

Program:

```
import java.io.fileReader;
import java.io.IOException;
public class ReadName
{
  public static void main (String[] args)
  {
    try
    {
      fileReader fr = new fileReader ("data.txt");
      int ch;
      while ((ch = fr.read()) != -1)
      {
        System.out.print((char)ch);
      }
      fr.close();
    }
    Catch(IOException e)
    {
      System.out.println("Error:" + e);
    }
  }
}
```

Suppose a text file named data.txt with the content:

my name is rajesh parajuli.
I am a java teacher.

You need to write sn the file.

Program

```
import java.io.filewriter;
import java.io.IOException;
public class writeexample
{
  public static void main (string args[])
  {
    try
    {
      filewriter fw = new filewriter ("data.txt");
      fw.write("my name is Rajesh parajuli.\n");
      fw.write("I am a java teacher.\n");
      fw.close();
      System.out.println("Data written sucess
                          fully!");
    }
    catch(IOException e)
    {
      System.out.println("error:" + e);
    }
  }
}
```

333

Advantages of Character Streams

- Best for text files
- Handles unicode characters
- Easy Line-by-Line reading
- Efficient when using buffers.

BufferedReader reads text efficiently using a buffer, instead of reading one character at a time. Provides readLine() method, which reads an entire line as a string.

Example

```
import java.io.FileReader
import java.io.BufferedReader;
import java.io.IOException;
public class example
{ public static void main(String [] args)
{
try
{ FileReader fr= new FileReader("data.txt");
//wrap filereader into BufferedReader
BufferedReader br= new BufferedReader(fr);
String Line
while (Line = br.readLine()) != null)
{ System.out.println(Line);
}
br.close();
}
catch(IOException e)
{
System.out.println("Error: "+e);
}}}
```

5.7 Random Access file

A Random Access file allows reading and writing data at any position in a file. unlike sequential files, we can move the file pointer anywhere in the file. Java provides RandomAccessfile class in java.io Package. RandomAccessfile works with bytes, not with character streams. Based on DataStream (Byte).

Syntax Declaration

```
RandomAccessfile raf = new RandomAccessfile ("file.txt","rw");
```

Modes:

(i) "r" → read only
(ii) "rw" → read and write

Methods

seek(position) → move file pointer
getfilepointer() → get current position
length() → get file size
read() → read byte
write() → write byte
readInt() → read integer
writeInt() → write integer

data.txt

```
Rajesh parajuli
```

seek position at 7 and read the file in java

```
import java.io.RandomAccessfile;
import java.io.IOException;
```

```
public class seekExample
{
  public static void main(String[] args)
  {
    try
    {
      RandomAccessFile raf = new RandomAccessFile("data.txt"
                                                  "r");
      raf.seek(4);
      int ch;
      while ((ch = raf.read()) != -1)
      {
        System.out.print((char) ch);
      }
      raf.close();
    }
    catch(IOException e)
    {
      System.out.println(e);
    }
  }
}
```

output:

parajulli