UNIT IV I/O, GENERICS, STRING HANDLING

I/O Basics - Reading and writing console I/O - Reading and writing Files. Generics: Generic Programming - Generic classes - Generic Methods - Bounded Types - Restrictions & Limitations. Strings: Basic string class, methods & Buffer class.

1. IO BASICS:

Java I/O is used to process the i/p and Produce the o/p based on i/p.

Streams:

Based on type of operation

1. System.in
2. system.out
3. system.err

Depending on type of file

1. Byte Stream
2. character Stream.

2. READING AND WRITING CONSOLE I/O:

Reading console input:

Method 1: Reading input using Buffered Reader, Inputstream Reader, System.in.

Syntax: Buffered_Reader obj_name = new BufferedReader(new InputstreamReader (system.in));

Program:

```
Class Readchar
{   Public static void main(String args[]) throws
                                        IOException
    {
        int count = 0;
        char ch;
        System.out.println("\n Enter 5 characters");
        while (count<5)
        {
            ch = (char) obj.read();
            System.out.println(ch);
            count++;
        }
    }
}
```

O/P:

```
Enter 5 characters
hello
h
e
l
l
o
```

Method 2: Reading console input using scanner and system.in

Pgm:

```
Public class Example
{
    PSVm( )
```

```
{
    Scanner sc = new scanner (string [] args)
    System.out.print ("Name?");
    String name = sc.next();
    S.o.pln(+name);
}
}
```

O/P

Name?
Ddurga

Writing console output:

The simple method used for writing the o/p on the console is write().

Syntax: System.out.write(int b)

---

3. READING AND WRITING FILES:

For Reading files in Java, we have

1. BufferedReader:

BufferedReader in = new BufferedReader (Reader in, int size);

2. FileReader class:

```
Public class ReadingFile
{
    PSVM
    {
        FileReader fileread = new FileReader ("c:\\Users\\test.txt");
        int i;
        While ((i = fileread.read()) != -1)
        { S.o.pln((char) i);
    }
}
```

3. Scanner class:

A scanner can parse the primitive types.

Ex:

```
import java.io.File;
import java.util.Scanner;
Public class Main
{
    psvm
    {
     try
      {
          File file = new File (pathname = "C:\\test.txt");
          Scanner sc = new Scanner(file);
          while (sc.hasNextline())
          {
              S.O.pln (Sc.hasNextLine());
          }
      }
     catch (Exception e)
     { S.O.pln ("Error"); }
    }
```

Writing text File in Java:

1. Java BufferedWriter class

Public class BufferedWriter extends Writer.

2. Java FileWriter class

```
import java.io.File;
import java.io.FileWriter;
public class Main
{
    psvm()
    {
```

```
try
{
    FileWriter fw = new FileWriter(Pathname:"C:\\test.txt");
    fw.write(str:"Hello");
    fw.close();
    S.O.pln("succesfully written");
}
catch(Exception e)
{ S.O.pln("error"); }
}
}
```

O/P
Hello

---

4. GENERIC PROGRAMMING:

Generic is a mechanism for creating a general model in which generic methods and generic classes enable prgmrs to specify a single method & single class for performing desired task.

Generic class:

TO create object of generic class, we use the following syntax:

UDCN<type> obj = new UDCN<type>()

UDCN - User defined class name.

type - any one of valid data types.

Example:

```
Class Test<T>
{
    T obj;
    Test(T obj)
    {
        this.obj = obj;
    }
    Public T getobject()
    {
        return this.obj;
    }
}
class gsample
{
    Psvm()
    {
        Test<Integer> iobj = new Test<Integer>(15);
        S.o'pln(iobj.getobject());
        Test<String> sobj = new Test<String>("Hello");
        S.o.pln(sobj.getobject());
    }
}
```

O/P

15
Hello

Generic Method:

It allows programmer to write a generalised method for the methods of different data types.

Example:

```
class Test
{
    static <T> void genericDisplay (T element)
    {
        S.O.pln (element.getclass().getName() + "=" + element);
    }
    PSVM()
    {
        genericDisplay(11);
        genericDisplay("hello");
        genericDisplay(1.0);
    }
}
```

O/P

```
java.lang.Integer = 11
java.lang.String = hello
java.lang.Double = 1.0
```

Bounded Types:

Using bounded types, we can make the object of generic class to have data of specific derived types. To restrict the type that can be used as argumen

Syntax: <T extends Superclass>

Example:

```
class Generic class <T extends Number>
{
    Public void display()
    {
        S.O.pln ("Bounded");
    }
}
```

```
class Main
{
    PSVM()
    {
        Genericclass<String> obj = new Genericclass<>();
    }
}
```

RESTRICTIONS AND LIMITATIONS:

→ Runtime execution is possible only if it is used along with raw type.

→ Primitive type parameter is not allowed.

→ Generic class throw & catch not allowed.

→ Instantiation of generic parameter T is not allowed. new T() // Error new T[10].

→ Parameterized types are not allowed.

Eg: new Stack<string>[10];

---

5. STRINGS

→ String is a collection of characters.

→ Two ways to create strings in Java

1. String S = "Hello"
2. String S = new String ("Hello");

Methods:

S.length() - Gives length of string S1.

S1.concat(S2) - concatenation of string S1 and S2.

S1.equals(S2) - If S1 and S2 equal, it returns True.

S1.charAt(position) - Returns character present at index position.

String toLowerCase - returns String in lowercase

Example:

```
import java.util.*;
class Main
{
    psvm()
    {
        String s = "welcome";
        S.O.pln (" String length = " + S.length());
        S.O.pln (" character at 3rd pos = " + S.charAt(3));
        S.O.pln ("Substring" + S.substring(3));
        String S1 = "Home";
        S.O.pln ("concatenated String = " + S.concat(S1));
        Boolean out = "hello".equals("hello");
        S.O.pln ("checking Equality" + out);
        Out = "hello".equalsIgnoreCase("HELLO");
        S.O.pln ("checking Equality " + out);
        S.O.pln ("changing to uppercase" + S.toUpperCase());
    }
}
```

O/P

String length = 7
character at 3rd pos = c
substring come
concatenated String = welcomeHome
checking Equality true
checking Equality true
changing to UPPER case WELCOME

STRING BUFFER CLASS:

StringBuffer class is used to create mutable string. ie) it can be changed.

Syntax:

```
StringBuffer <obj_Name> = new StringBuffer();
```

Example:

```
class Main
{
  Psvm()
  {
    StringBuffer sb = new StringBuffer("Hello");
    sb.append("Java");
    S.opln(sb);
    sb.insert(6, "welcome");
    S.o.pln(sb);
    sb.replace(9, 13, "done");
    S.o.pln(sb);
    sb.delete(0,13);
    S.o.pln(sb);
    sb.reverse();
    S.o.pln(sb);
  }
}
```

O/P:

```
Hello Java
Hello welcomeJava
Hello weldoneJava
Java
avaJ
```