UNIT - 3

Exception handling

The dictionary meaning of exception is abnormal termination. After coriting a group of Java statements by satisfying the consults rules The program successfully complied and ready for executedn

→ There is no graunitee for program execution since run time errors may occur.

→ The errors in the run time is called exception and program pargoes to abnormal termination and restof code never execute.

Java provides another additional, feature to handle the exceptions this process is called exception handling

Java provides 5 keywords

i) try
ii) catch
iii) throw
iv) throws
v) finally

to handle the exceptions

→ try-catch (must be inside a method)

The Susceptable (suspectable) or risky code can be kept in try block, to try any run time errors. When try block identifies any errors it sends to catch block

→ Catch block contains handling code according to the recieved error.

→ catch block needs an argument of Exception class object or its child class objects.

Syntax

```
try
{
    risky or suspect code;
}
Catch (Exceptionname Obj)
{
    statements;
}
```

Java program to demonstrate exception on zero division.

```
import java.util.*;
class Pr56
{
    Public static void main(String args[])
    {
        Scanner s = new Scanner(System.in);
        // int a, b;
        System.out.println("Enter 2 numbers");
        try
        {
            int a,b;
            a = s.nextInt();
            b = s.nextInt();
            int c = a/b;
            System.out.println("The result is"+c);
        }
        catch(Exception e)
        {
            System.out.println("Exception is handled");
        }
        System.out.println("holidays decided");
        System.out.println("Eat more sweet");
    }
}
```

//exception is the super class

→ In the program above it may raises NumberFormatException or ArithmeticException these 2 execptions can be handled by its super class Exception.

→ Exception Occured or not Occured the remaining code will execute.

Write a Java program to handle ArrayIndexOutOfBoundsException.

A>
```
class A
{
    Public static void main(String args[])
    {
        int a[]= {1,2,3,4,5};
        System.Out.println(a[20]);
    }
}
```

→ If you run the above program the program goes to abnormal termination due to out of Index element. It can be handled with the following code.

```
class A
{ Public static void main(String args[])
  {
    try
    {
      int a[] = {3,4,5,6,7};
      System.out.println(a[10]);
    }
    catch (ArrayIndexOutOfBoundsException ae)
    { System.out.println("Exception is handled");
    }
```

```
System.Out.println("hi everyone");
    }
}
```

Multiple catch Blocks

In java, one "try" block may contains more than one "catch" block - Such catch blocks are called Multiple catch Blocks.

In writing the multiple catch blocks the Super class Exception must be placed at last

when try block throws an exception it checks the catch from top to bottom, when the child exceptions are not handling then it can be handled by the last Super class.

The Exception hierarchy in Java is

Object

- Throwable
  - Exception
    - RuntimeException
      - ArithmeticException
      - NullpointerException
      - NumberFormatException
      - IndexOutOfBoundsException
        - ArrayIndexOutOfBoundsException
        - StringIndexOutOfBoundsException
    - IoException
      - EOFException
      - FileNotFoundException
      - InterruptIOException
    - InterruptedException
  - Error
    - VirtualMemoryError
    - LinkageError
      - StackOverflowError
      - OutOfMemoryError

Note :- When you know the exact exception then same Exception_class in catch block otherwise use super class Exception (Throwable, Exception)

Java program to implement multiple catch blocks.

```
import java.util.*;
class Pr57
{
  Public static void main(String args[])
  {
    //Scanner s = new Scanner(String
                                  args[])
    Scanner s = new Scanner(System
                                  .in);
    try
    {
      int a[] = new int[1000000];
      for(int i=0; i<5; i++)
        {
          a[i]= s.nextInt();
        }
      System.Out.println("which element
                          index you want");
      int i = s.nextInt();
      System.out.println(a[i]);
      int a, b;
      a = s.nextInt();
      b = s.nextInt();
      int c = a/b;
      System.out.println("Result:"+c);
    }
    Catch(VirtualMemoryError ai)
```

```
System.out.println(" Memory not enough");
}
catch (ArrayIndexOutOfBoundsException v)
{ System.out.println(" Array index is out of
                        range");
}
catch (NumberFormatException ne)
{ System.out.println(" Cannot read other
                        than digits");
}
catch (ArithmeticException be)
{ System.out.println(" Division by Zero");
}
catch (Throwable b)
{ System.out.println(" All exceptions are
                        handled");
}
}
}
```

Enter
100
java.lang.ArrayIndexOutOfBoundsException:

```
class TestExp
{ Public static void main( String args[])
  {   int arr[] = {11, 77, 88, 22};
      System.out.println (arr[100]);
  }
}
```

```
import java.util.*;
class Ex
{ Public static void main (String[] args);
  {  Scanner sc = new Scanner(System.in);
     try
     { int arr[] = {11, 77, 88, 22};
       System.out.println ("Enter which indexed you");
       int i = sc.nextInt();
       system.out.println (arr[i]);
       int a=10;
       int c=a/i;
       system.out.println (c);
     }
     catch (Exception e)
     {  System.out.println (e);
     }
  }
}
```

output

Enter

0

||

java.lang.ArithmeticException: / by zero

1) Printing the exception class object

```
try
{

}
catch(exception_class object)
{ System.out.println(object); /// description of runtime exception.
```

② tostring(); belongs exception class
belongs java.lang package

system.out.println(object.tostring());

Through tostring() we can print the description.

→ Printing the exception class object and printing the to string() produces same result

③ getMessage(); belongs exception class
belongs java.lang package

```
try
{

}
catch (exceptionname object)
{ System.out.println(object.getMessage());

}
```

```
catch (Exception e)
{System.out.println(e);   // through object
System.out.println(e.toString());   /* through toString() */
System.out.println(e.getMessage());   /* through getMessage() */
}
```

④ printStackTrace();

full details of exception
→ which class and package
→ message
→ java file name and line no. of exception occured

nested try

nest - using the same in it
nested while - using while inside a while
same for if, for

nested try
// using try block inside the try block

```
try
{   //risky code or Suspectable
    try
    {
        //risky code
    }
    catch(Prakash p)
    {
    }
catch(Hod h)   //super class
```

Outer

Inner

If an exception rises in inner try Block if it is not handled by inner catch it must hand over to outer catch

```
import java.util.*;
class NestedTry
{ Public static void main(String[]a)
  { Scanner sc = new Scanner(System.in)
    try
    {
      int arr[] = {11,22};
      System.out.println("Enter Value");
      int i = sc.nextInt();
      System.out.println(arr[i]);
      try
      {
        int a,b,c;
        System.out.println("Enter 2 Val");
        a=sc.nextInt();
        b=sc.nextInt();
        c=a/b;
        System.out.println(c);
      }
      catch(ArithmeticException e)
      {
        System.out.println(e);
      }
    }
    catch(Throwable t) //Superclass must be exception
    {
      System.out.println(t);
    }
  }
}
```

otherwise it major may not give the correct result

output

Enter value
0
11
Enter 2 Val
a=12
b=3
c=4

(2.) → Enter value
1
22
Enter 2 Val
Prakash
java.util.InputMismatchException

→ In nested try's to handle all the exceptions, use super class exception in outer catch block.

Nested try-multi catches

(1.)
```
try
{
    //risky
    {
        try
        {
            }
        Catch( )
        {
            }
        Catch( )
        {
            }
}
```

Valid

```
catch()
{

}
```

② Valid

```
try
{
    try
    {
    }
    catch()
    {
    }
}
catch()
{
    try()
    {
    }
    catch()
    {
    }
}
```

③

```
try
{
    try
    {
    }
    catch()
    {
        catch()
        {
        }
    }
}
catch()
{
}
```

Compile time error.

finally()

finally block has a group statements that statements will be executed without considering try or catch

```
try
{
    S1;
    S2;
    S3;
    S4;
    S5;
}
catch(Exception)
{
    S21;
    S22;
}
```

finally ()
{ S_{31};
S_{32};
}

→ when an exception is occured $S_1 - S_2 - S_3 - S_{21} - S_{22} - S_{31} - S_{32} - S_{33}$

→ when there is no exception
$S_1 - S_2 - S_3 - S_4 - S_5 - S_{31} - S_{32} - S_{33}$

throws keyword

→ used to handle exceptions

Java has 2 ways to handle exceptions

try-catch block

→ used at implementation level or inside the method definition

eg:
```
class Apple
{
  void m1()
  {
    try
    {
      a = 10/0;
    }
    catch (Exception e)
    {
    }
  }
}
class Pr
{
  Public static void main(String[])
  {
    Apple ob = new Apple();
    ob.m1();
  }
}
```

throws keyword

→ used in the method heading and deligates the exception information to the callers

eg:
```
class Apple
{
  void m1() throws ArithmeticException
  {
    a = 10/0;
  }
}
class Pr
{
  Public static ...
  {
    Apple ob = new Apple();
    try
    {
      ob.m1();
    }
    catch (Exception e)
    {
    }
  }
}
```

when the method is not capable to handle the exception then it use throws keyword in the method heading

Syntax

```
returntype methodname(args) throws Excep1, Excep2, ... Excepn
{
    // body of the method
}
```

throws keyword demo

```
import java.util.*;
class Apple
{
    Void m1() throws ArithmeticException, ArrayIndexOutOfBoundException, InputMismatchException
    {
        Scanner s = new Scanner(System.in);
        int arr[] = {101, 201};
        int a = 32, i;
        System.Out.println("Enter i vallue");
        i = s.nextInt();  //Index 0 or 1
        System.out.println(arr[i]);
        int c = a/i;
        System.out.println("value of c"+c);
    }
}
class Pr
{ public static void main(String args[])
```

```
Apple ob=new Apple();
try
{
   ob.m1()
   System.out.println("hii");
}
catch (Exception e)
{
   System.out.println("handled in main", e);
}
}
}
```

Output

① enter i value
0
101
handled in main java.lang.ArithmeticException: / by Zero

② enter i value
Prakash
handled in main java.util.InputMismatchException

Exception Handling

try-catch --- at method Implementation level

throws -- at method heading, which delegates excep information to the Caller

when System is providing only System generated Exception and messages also we are getting default System message (try-catch is used)

In real time we should generate our own (user defined) exceptions. (throw is used)

throw: Syntax throw new Subclass_Exception ("text");

```
class Bank
{
    int bal=10;
    void withdraw(int wa)
    {
        if (wa > bal)
            System.out.println("not having");
        else
            System.out.println("take Money");
    }
}
class pr
{
    Public static void main(String args[])
    {
        Bank dcp = new Bank();
        dcp.withdraw(1300);
        System.out.println("r u want any more services");
    }
}
```

Normal flow

Output

not having

ru want any services

Note

throw new Exception("text"); // compile error due to not

```
class A
{   int b = 10;
    void withdraw (int wa)
    {   if(wa > bal)
            throw new NullPointerException(
                                You are not having);
        else
            System.out.println("take");
    }
class Pr
{   Public static void main(String args[])
    {   A dob = new A();
        try
        {   dob.withdraw(1300);
            System.Out.println("u want
                                any service");
        }
        catch (Exception e)
        {   System.out.println(e);
        }
    }
}
```

output

Creating the ~~pre~~ user defined exception classes

Exceptions are two types as checked and unchecked Exceptions.

→ At the time of compilation any exceptions are identified then those exceptions are called checked Exceptions

Eg:- Interupted Exception

→ The exceptions those are identified at run time are called unchecked exceptions

Eg:- Arithmetic Exception
ArrayIndexOutOfBounds Exception

→ when the program goes abnormal termination by using exception handling we can make the program as normal termination.

→ The predefined Exception causes an abnormal termination. //Some

→ Some times we want to make the normal flow into an abnormal termination.

→ In this process we have to create user defined Exceptions

Java ~~prov~~ provides to create our own Exception classes for checked and un checked exceptions

→ when you extending the Exception class we can handle user defined checked Exceptions.

eg: class Myexp extends Exception
{
/*checked exception*/ Myexp() // Zero-arg constructor
{
}
Myexp(String s) /*1-arg constructor*/
{ super(s);
/* calls parent class constructor */}
}

→ For unchecked exceptions we have to extend Runtime_Exception.

//class Myexp extends Excepl

class Myexp extends RuntimeException
{ Myexp()
{
}
Myexp(String s)
{ super(s);
}
}

Write a Java program to create user defined exceptions for voting process based on given age.

```
import java.util.*;
class Invalidage extends Exception
{   Invalidage (String s)
    {  Super(s);
    }
}
class Vote
{  void poling() throws Invalidage Exception
   {  Scanner sc = new Scanner(System.in);
      int a = sc.nextInt();
      if(a < 21)
         throw new Invalidage("you fool go and study well");

      else
         System.out.println("Marry and happy to all of you");

      System.out.println(" All the Best");
   }
}
```

Class Pr ex

```
{ Public static void main(String args[])
        throws Invalidage
  {
    Bam Vote d = new Vote();
        d.poling();
  }
}
```

Java program to create the user defined exceptions for the bank with draw method based on given withdra Amount.

```
import java.util.*;
class Invalidamount extends Exception
{
    Invalidamount(String s)
                → /* super(); */
    {  super(s);
    }
}
Class Bank
{   void withdraw() throws Invalida
                           mount
    {
        Scanner sc = new Scanner(System
                                 .in);
        int wda = sc.nextInt();
        if(wda<100)
            throw new Invalidamount
                        ("you fool enter
                         multiples of 100");
        else
            //throw
            System.out.println("take
                                money");
        System.out.println("thank u
                            Customer Visit again");
    }
}
```

```
class In
{ Public Static Void main(String args[]).
        throws Invalidamount
  {
    Bank d = new Bank();
      d.withdraw();
  }
}
```

Stream based Input Output

stream:- Java performs i/o operations throw streams

→ stream is an abstract which produces or consumes the information

Java has several streams all behaves same manner but different links with different devices like keyboard, console output, disk file or memory

→ Java has input streams and output streams to get and to produce data

→ Java Programs are not text based or console outputs but java programs relay on data through swings, GUI, AWT, JavaFX

→ But traditionally a java program can obtain the data from the keyboard through System.in from java.lang package

→ Similarly to display the outputs or errors to the console devices - java.lang package has System.Out and System.err (for standard error) classes

Note:- The predefined package java.lang is the default package package of any java program.

Byte and characters stream

Java program allows two types of data as binary data and character data

Byte stream:-

It produces or consumes binary data of a java program

→ In the byte stream java.io package contains byte stream classes to override all of these method in their subclasses

→ The top most two abstract classes of byte streams are InputStream and OutputStream contains read() and write() abstract methods respectively.

→ These abstract class have so many concrete sub classes as BufferedInputStream and BufferedOutputstream contains read() and write() concrete methods

→ Java has InputstreamReader class which converts binary data to character data through constructor.

Character stream

→ It produces and consumes character data with unicode (ASCII) is called character stream.

→ In this stream the topmost abstract classes are Reader and Writer contain read() and write() abstract method respectively

→ In BufferedReader class we should instance the System.in to obtain characters from the keyboard. and the constructor is BufferedReader (InputReader)

→ To convert the binary data to character data we should an object of InputStreamReader class object

Java program to read the characters from the keyboard until pressing e

```
import java.io.*;
class Readerdemo
{
  Public static void main(String args[])
                        throws IOException
  {
    char ch;
    InputStreamReader isr = new InputS
                        tream Reader(System
                        .in);

    BufferedReader br = new BufferedReader
                        (isr);

    do{
      ch=(char) br.read();
      System.out.println(ch);
    } while(ch!='e');
```

```
    }
}
```

The above program we used the read() its general form is

```
int read() throws IOException
{

}
```

readLine()

This method is used to read the string data from the console input devices and throws an IO Exception

→ The readLine() is a concrete method in BufferedReader class

Java program to allow a group of strings till pressing the "stop" string (X X X) Mid, Unit

```
import java.io.*;
class Reader
{ public static void main(String args[])
                throws IOException
  {
    string s1;
    FileInputStreamReader isr = new
                InputStreamReader(
                System.in);
    BufferedReader br = new
                BufferedReader(isr);
```

```
do
{
    s1 = br.readLine();  // reads string data
    System.out.println(s1);
} while (!s.equals("stop"));  // equales belongs to string class
}
}
```

Writing data to the console

The familiar printStream methods are print() and println().

Java has other several methods to write the data onto the console output devices.

In java.lang package, System class contains write()

Its general form

```
void write(byte val)
{
```

eg:- int a=12;

System.out.write(a)

```
}
```

O/P 00001100

Write a Java program to initialize one integer and one character variable & display them by using write method.

```
class A
{
  Public static void main(String args[])
  {
    int a = 12;          /*integer initialization*/
    System.out.write(a);
    char s = '@';        /* character initialization
    System.out.write(s);
  }
}
```

/* using System class to print using write() */

PrintWriter class

This is the concrete class of OutputStream abstract class. it contains several methods to display the data on the console device

→ The print() and println() are also available in this class, we can access them by using PrintWriter class

→ It also contains several constructors (overloading constructor) we use two-argument constructor.

```
PrintWriter(OutputStream, boolean flushing on)
{

}
```

Java program to display a string, an integer value and a double value through printwriter class.

```
import java.io.*;
class Writerof
{ Public static void main(String
        args[]) throws IOException
  {
    PrintWriter ob = new PrintWriter
                (System.out, true);
      ob.println("hii everyone");
    int x = 9;
      ob.println("the x value is"+x);
    double d = 9.5;
      ob.println("the d value is"+d);
  }
}
```

Write a Java program to read two integers (without using Scanner class), and display their product by using Printwriter class

A.) import java.ut io.*;

```
class Readerin
{ Public static void main(String args[])
        throws IOException
  {
    InputStreamReader isr =
        new InputStreamReader(
                System.in);
```

```
BufferedReader br = new BufferedReader
                    (isr);
int a = Integer.parseInt(br.readline
int b = Integer.parseInt(br.readline
PrintWriter pr = new PrintWriter(
                System.out, true);
  pr.println("The product of
              2 values :" + (a*b));
  }
}
```

Files

File is a collection of data has the operations like opening, reading, writing and closing.

→ Java has several file streams to read the data from the file and to write the data to the file the basic file classes are FileInputStream and FileOutputStream.

→ This 2 classes has concrete methods read() and write() respectively.

Reading data from a file.

→ read() is used to read the data from the file and converts it into integer.

→ write() is used to write the first 8 bits of data to the file.

→ The FileInputStream class with read() method throws FileNotFoundException

Similarly the FileOutputStream write() throws same Exception.

All the above said classes belongs to java.io package

To Java program to read the data from the existing file

{ Select the notepad and type some [illegible] finally save the file with

name abc.txt in java working directory

```
}
import java.io.*;
class A
{
  Public static Void main(String args[])
          throws FileNotFoundException
  {
    FileInputStream fis = new FileInputStream
                        ("abc.txt");
    Int x;
    x = fis.read();   //reads 1st 8 bits
                        of data (ASCII
                        Value)
    System.out.println(x);
    System.out.println((char)x);
  }
}
```

Writing the data to a file

Java has an OutputStream ~~Out~~ abstract class with several methods. one of the method is write();

→ The ~~file~~ FileOutputStream is the concrete class with a concrete write() and makes direct link with file.

→ This class has overloaded constructors to open the file in ~~right~~ write mode or append mode

The constructors are (FileOutStream)

```
File Output Stream (String filename)
  {

  }
```

This constructor opens the specified file name in write mode. If the file doesn't exist it creates the new file.

→ If the file exist the contents are aqised and updates with the new data

```
FileOutputStream (String s, boolean append)
  {

  }
```

Java program to write the data to a file.

```
import java.io.*;
class Abc
{ Public static Void main(String [])
        throws FileNotFoundException
  {
    FileOutputStream fos = new
                FileOutputStream ("bcd.txt");
    int a;
    a=8;
    fos.write(a);
```

```
fos.close();
    }
}
```

D:\501>javac Abc.java
D:\501>java Abc
D:\501>

Stores in file

Additional

The disadvantages of read() and write() is they are only applicable for byte data.

→ Java has another I/O classes as DataInputStream and DataOutputStream for reading and writing primitive data of a file.

→ DataInputStream class is the Sub class to FilterInputStream and also implements DataInput interface.

→ The available methods are

readByte()
readChar()
readInt()
readUTF() (uniform text format)
readDouble()

→ DataOutputStream class is also the Sub class to FilterInputStream and also implements DataInput interface.

→ The available methods are

~~read~~	writeByte()	writeInt()
	writeChar()	writeFloat()
	writeUTF()	writeDouble()
		writeBoolean()

writing Primitive data Values to the file.

```
import java.io.*;
class Alldata
{
  Public static Void main (String [] args)
                throws IOException
  {
/* opens file */  FileOutputStream fos = new FileOutput
                Stream ("dcp.txt");
/* Not capable of opening file so, fos is used */  DataOutputStream dos = new
                DataOutputStream (fos);
link.  byte b = 1;
       short s = 7;
       int a = 6;
       float f = 8.4;
       double d = 7.42;
       long l = 7273;
       char c = '@';
       boolean bi = true;
       string si = "All";
       // dos.write (b);
       // dos.short (s);
       dos.writeByte (b);
       dos.writeShort (s);
       dos.writeInt (a);
       dos.writeFloat (f);
       dos.writeDouble (d);
       dos.writeLong (l);
       dos.writeChar (c);
```

```
dos.BowriteBoolean(bi);
dos.writeUTF(si);
fos.close();
}
```

Random Access file : (* * *) exam Sem

Random Access File

Input, output stream are in same channel so, specify modes

with existring we can read or writes the data in sequential manner there is no possibility to skip unwanted data.

→ Java has Random Access File class with seek() we can sets the file pointer position from current to desired.

The Syntax is

object.seek(int);

→ The starting position indicates '0'.

→ to know the last position of the file we have to use available() which returns an integer

→ when the current position is at x, the desired position is less than means backward direction.

→ when the current position is at x, the desired position is greater than means forward direction.

```
import java.io.*;
//class Reader
class Randomdemo
{
    Public static void main (String
                    args[]) throws
                        Exception
    {
/* 2-arg  RandomAccessFile raf = new
Constructor*/   RandomAccessFile ("abc.txt",
                    /* mode*/ "rw");
        String s = "Java is a good language".
        raf.writeUTF(s);
        raf.seek(9); // sets the filepointer
                        position from 0 to 9
//good langu*  String s1 = raf.readUTF();
        System.out.println(s1);
    }
}
```

Console class

Java needs some Security for the data for that io package contains Console class with no constructor and the methods readLine() and readPassword()

→ readPassword() doesnot displays the entered characters since "echo" is off

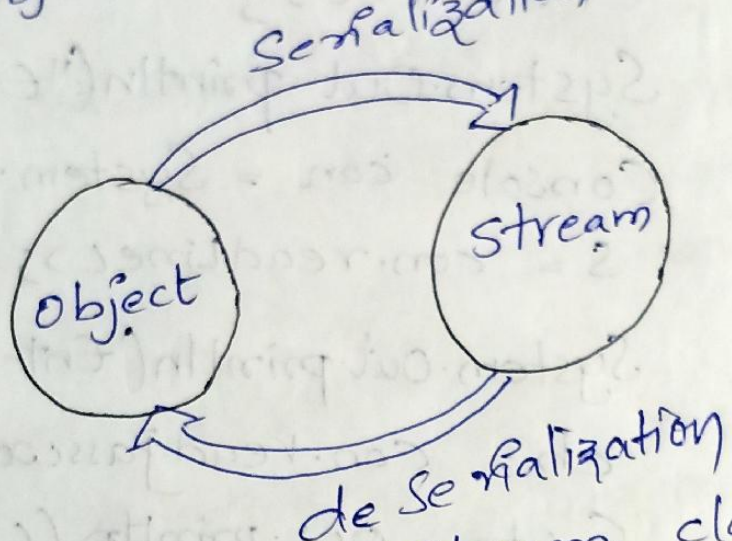
```
import java.io.*
class Consoledemo
{ public static void main(String args[])
                    throws Exception
  {
      String s;
      char ch[];
      System.Out.println("Enter Your Id");
      Console con = System.console();
      s = con.readLine(); //enter data from Keyboard
      System.Out.println("Enter passord");
      ch = con.readPassword();
      System.Out.println(ch);
//object code will print so, typecast
      String ps = String.ValueOf(ch)
      System.Out.println("password "+ps
      con.printf("Password is %s\n",ps)
  }
}
```

Serialization

Serialization is a process of converting object data into stream data

de-Serialization

The process of converting streamdata into object data



Java has ObjectInputStream class with readObject() to ~~send~~ get the objects data from the file

Java has ObjectOutputStream class with writeObject() to ~~get~~ send the object data to the file.

Making Serialization class

At the time of defining the class when it is implements Serializable interface then we are ready to and fro object data of the file

eg:

```
class kusuma implements Serializable
{ int a, b, c;
}
```

Java program to send class object information to the file.

```
import java.io.*;
class Student implements Serializable
{ int Sno;
  String Sname;
}
class Serialdemo
{ Public Static Void main(String args[])
                throws Exception
  {
    FileOutputStream fos = new
                FileOutputStream("abc.txt");
    ObjectOutputStream os = new
                ObjectOutputstream(fos);
    Student dcp = new Student();
      dcp.Sno = 420;
      dcp.Sname = "takkaridonga";
      os.writeObject(dcp);
      fos.close();
  }
}
```

Java program for getting object data from the file.

```
class student implements Serializable
{
    int sno;
    String sname;
}
class Deserialdemo
{ Public static void main (String[] args) throw Exception
  {FileInputStream fis = new FileInputStream ("abc.txt");
   ObjectInputStream ois = new ObjectInputStream (fis);
   student pcd = (student) os.readObject (dcp);
   System.Out.println ( pcd.sno);
   System.Out.println (pcd.Sname);
   fis.close();
  }
}
```

Enumerations

Like classes and interfaces Java has Enumerations to create final static members (constants) and also concrete methods

→ Enumerations are used for Data Safe.

→ After Compilation JVM creates .classfile for every enumeration

→ The enumeration members are constants, it is better to create them in upper case

Syntax

```
enum enumname
    { MEMBER1, M2, M3 ... Mn;   // Semicolon are optional
    }
```

eg:

```
enum Direction
    {
        SOUTH, EAST, NORTH, WEST
    }
```

Program to demonstrate the enumerations

```
import java.util.*;
enum Week
    {
        T, W, TH, F, SA;
```

```
class Pr
{
  Public static void main(String args)
  {
      Week w = Week.S;
      System.out.println(w); //S
//0     System.out.println(w.ordinal());
                              Index
      w = Week.TH;
//4   System.out.println(w.ordinal())
```

Enumerations have 2 methods called.
Values() and ValueOf()
ValueOf() it returns the name of the member which assigns to enumeration variable or object
Values() it returns an array list with all the enumeration members

```
enum Dir
{
  NORTH, SOUTH, EAST, WEST;
}
class Pr
{
  Public Static Void main(String args[])
  {
      Dir d = Dir.NORTH;
//NORTH System.out.println(d);
```

```
System.out.println(d.valueOf());
 for(Dir i : Dir.Values())
   {  System.Out.println(i);
   }
 }
}
```

Java program to fix the Values of enumerated members.

```
import java.util.*;
enum  Bike
  {  HERO(99875), HONDA(34521), BONDA(4225),
        KTM(3125), JAVA(12312), VADA(14325)

     Private int price;
       Bike(int p)
       {  Price = p;
       }
       int getprice()
          {  return(price);
          }
     }
class Pr
{ Public Static Voidmain(String args[])
    {  Bike b = Bike.HONDA;
```

```
// only One member System.out.println("Honda bike"+b.getprice());
        for (Bile i : b.values())
        {
// All the members System.Out.Println(i+"price"+i.getprice());
        }
    }
}
```

Auto Boxing

from JDK 1.5 java has an additional feature Auto Boxing the primitive data to the primitive class object

Without Boxing the assigning of integer value to Integer class object is

```
Integer a = new Integer(100);
// a value is 100
// a is an object
```

with autoboxing the above steps can be written as

```
Integer i = 10;
```

Auto unBoxing:-

The process of assigning primitive class object data to primitive variable is called auto unboxing

```
Integer iob = 100; //auto boxing
int a = iob; // auto unboxing.

class A
{ Public static void main(String args[])
  {
    Integer iob = 100; // auto boxing
    int a = iob;  // unboxing.
    System.out.println (iob+" "+a);
  }
}
```

Generics (util package)

It is the second version of collections to create an arraylist to only specific datatype.

Generics doesn't need any typecasting

The errors are identified at compile time. It is good to handle rather than handling the Exception.

Generics program

```
class Pr
{
  Public static void main(String args[])
  {
    ArrayList<String> al = new ArrayList<
                           String>();

    al.add("happy");
    al.add("hello");
    al.add(10); // compilation error
    System.out.println(al.get(0));
```

// happy

no need of type casting

```
}
```