Exception Handling:

Exception:

An exception is an event which disrupts the normal flow of execution of the program.

Exception Handling:

Exception handling is a mechanism to handle run-time errors.

Compile-time errors:

Errors which can be detected at compile time are known as compile-time errors.

E.g:

```
System.out.println("WELCOME")
```

1 error

Missing semicolon

Run-time errors:

Errors which occur at the run-time are known as the Run-time errors.

```
//Java program to illustrate run-time errors.
import java.util.*;
class Example
{
  public static void main(String args[]) throws Exception
  {
    int a,b, c;
    Scanner sc = new Scanner(System.in);
```

```
        System.out.println("\n\n\t ENTER THE VALUE FOR a.... ");
        a=sc.nextInt();
        System.out.println("\n\n\t ENTER THE VALUEE FOR b....");
        b=sc.nextInt();
        c=a/b;
        System.out.println("\n\n\t THE RATIO OF THE NUMBERS IS...."+c);
    }
}
```

Input / output:

```
ENTER THE VALUE FOR a.....4
ENTER THE VALUE FOR b.....0
Arithmetic Exception (Division by Zero)
```

Types of Exception:

Exceptions are classified into two types. They are

1. predefined Exceptions
2. Userdefined Exceptions

Predefined Exceptions:

The exception which are available within the java Software are known as predefined Exceptions.

E.g:

1. ArithmeticException
2. ArrayIndexOutOfBoundsException
3. ClassNotFoundException
4. FileNotFoundException
5. IOException
6. NumberFormatException

1. ArithmeticException:

It is thrown when an exceptional condition occurs in an arithmetic exception.

2. ArrayIndexOutOfBoundsException:

It is thrown when an array is accessed with an illegal index. the index may be either negative or greater than size of the array.

3. ClassNotFoundException:

It is thrown when we try to access a class whose definition is not found.

4. FileNotFoundException:

It is thrown when we try to access a file which does not exist.

5. IOException:

It is thrown when an input or output operation is failed or interrupted.

6. InterruptedException:

It is thrown when a thread which is in waiting, sleeping or doing some processing is interrupted.

7. NumberFormatException:

It is thrown when a method could not convert a String into number.

8. NullPointerException:

It is thrown when we refer to the members of a null object.

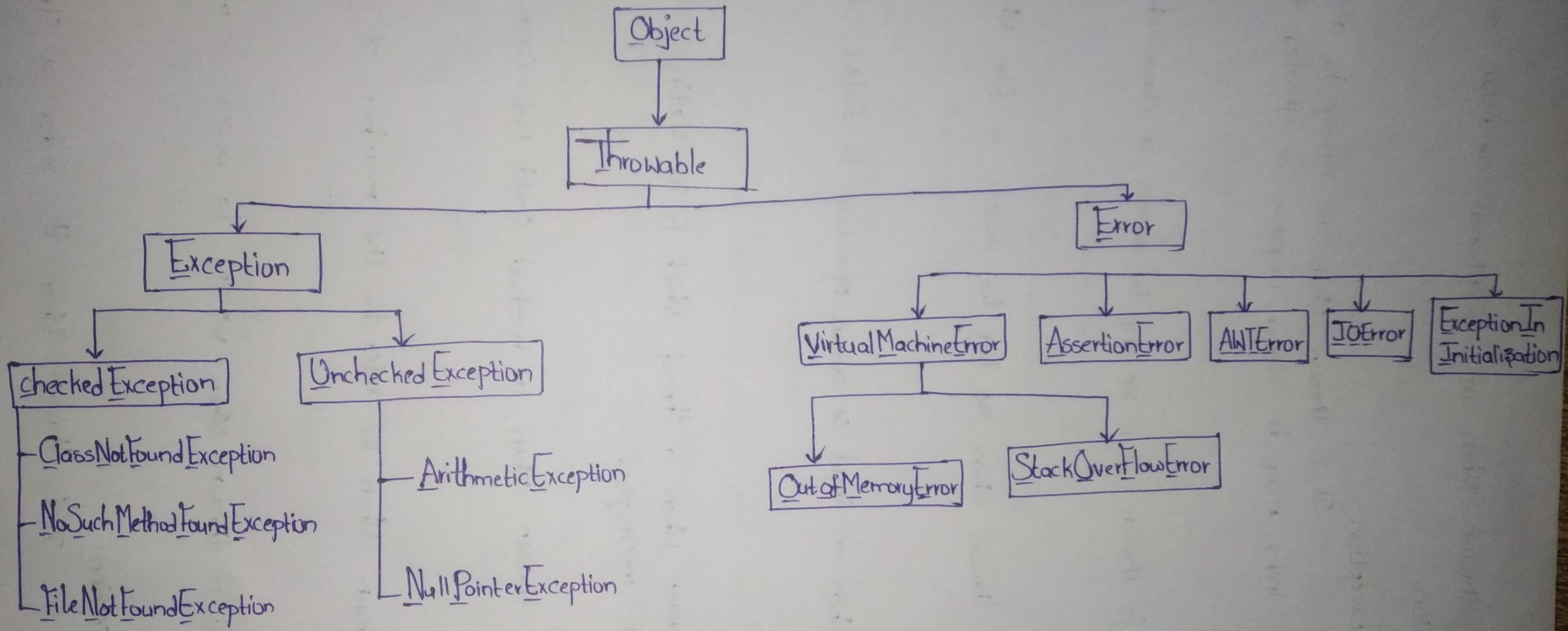
9. NoSuchMethodFoundException:

It is thrown when we try to access a method which does not exist.

10. StringIndexOutOfBoundsException:

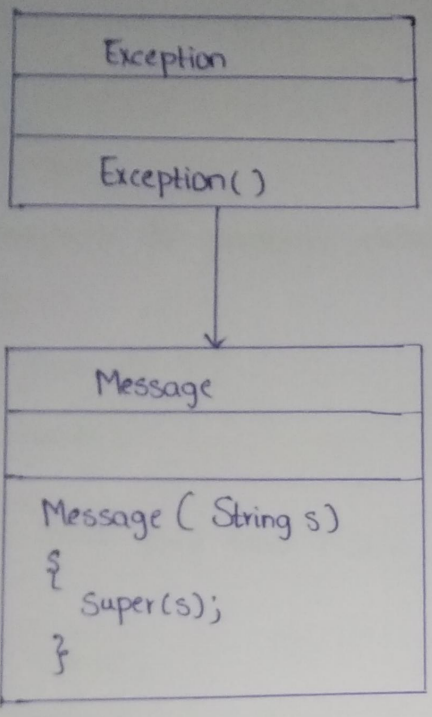
It is thrown when the index of String exceeds it's size.

Java Exception Hierarchy:



Userdefined Exceptions:

The exceptions which are created by the user are known as Userdefined Exceptions.



Exception is a predefined class and our user defined class is derived from the exception class. our user defined class has a constructor which takes a string parameter and passes it to the superclass constructor.

Exceptions can also be categorized as

1. checked Exceptions
2. Unchecked Exceptions.

checked Exception:

The exception which can be checked at compile-time are known as checked Exceptions.

Unchecked Exception:

The exception which cannot be checked at compile-time are known as Unchecked Exceptions.

Keywords associated with Exceptions:

1. try
2. catch
3. finally
4. throw
5. throws.

```
try
{
 code which contains a runtime error
}
catch (Exception e)
{
 Print the exception occured
}
finally
{
 Statement to be executed irrespective of occurence of exception
}
```

```
// Java program to perform division of two numbers without exception handling.
import java.util.*;
class Ehandle1
{
  public static void main (String args[])
  {
    int a,b,c;
    Scanner sc = new Scanner (System.in);
    System.out.println ("\n\nIt ENTER THE VALUE FOR a.....");
    a = sc.nextInt();
```

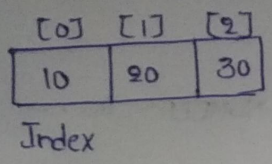
```
        System.out.println("\n\n\t ENTER THE VALUE FOR b.....");
        b=sc.nextInt();
        c=a/b;
        System.out.println("\n\n\t THE RATIO OF TWO NUMBERS IS....."+c);
    }
}
```

// Java program to perform division of two numbers using Exception handling.

```
import java.util.*;
class Ehandle2
{
  public static void main(String args[])
  {
    int a,b,c;
    Scanner sc=new Scanner(System.in);
    System.out.println("\n\n\t ENTER THE VALUE FOR a.....");
    a=sc.nextInt();
    System.out.println("\n\n\t ENTER THE VALUE FOR b.....");
    b=sc.nextInt();
    try
    {
      c=a/b;
    }
    catch(Exception e)
    {
      System.out.println(e);
    }
  } System.out.println("\n\n\t THE RATIO OF TWO NUMBERS IS....."+c);
}
```

Single try and multiple catch Statements:

Index refers to the position of the element in the array.




```
b=0;
c=a[3]/b;
```

Array Index Out Of Bounds Exception

Arithmetic Exception (Division by zero).

// Java program to illustrate Single try and multiple catch Statements.

```
import java.util.*;
class Multiplecatch
{
  public static void main (String args[])
  {
    int a[],b,c,n;
    Scanner sc = new Scanner (System.in);
    System.out.println("\n\nIt ENTER THE SIZE OF THE ARRAY.....");
    n=sc.nextInt();
    a=new int[n];
    System.out.println("\n\nIt ENTER THE ELEMENTS OF THE ARRAY.....");
    for (i=0; i<n; i=i+1)
    {
     System.out.println("\n\nIt ENTER THE ELEMENT.....");
     try
     {
      a[i] = sc.nextInt();
     }
     catch(Exception e1)
```

```
        {
          System.out.println(e1);
        }
    }
    System.out.println("\n\nIt ENTER THE VALUE FOR b.....");
    b=sc.nextInt();
    try
    {
      c=a[n]/b;
    }
    catch (Exception e2)
    {
      System.out.println(e2);
    }
    catch (Exception e3)
    {
      System.out.println(e3);
    }
  }
}

//Java program to create a user defined exception.
import java.util.*;
class Simple extends Exception
{
  public Simple (String s)
  {
    super(s);
  }
}
```

```
class Demo
{
  public static void main (String args[])
  {
    int age;
    Scanner sc = new Scanner (System.in);
    System.out.println ("\n\nIt ENTER YOUR AGE.....");
    Age = sc.nextInt();
    if (age <18)
    {
      throw new Simple (" YOU ARE NOT ELIGIBLE FOR VOTING ");
    }
    else
    {
      System.out.println ("\n\nIt YOU ARE ELIGIBLE FOR VOTING ");
    }
  }
}
```

Streams

Stream

A Stream refers to a flow of data from a source to destination.

Stream are classified into two types.

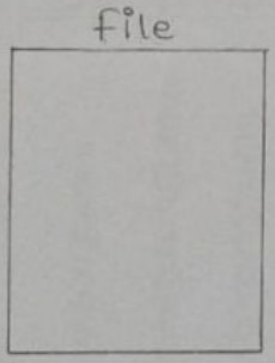
1. InputStream
2. OutputStream

InputStream

It is used to read data from a source.

OutputStream

It is used to write data to a destination.



File :

A File is a Collection of data.

Reading data from a file refers to accessing the data from the file.

Writing data to a file refers to storing the data in the file.

Depending upon the type of data being handled, streams are classified into two types.

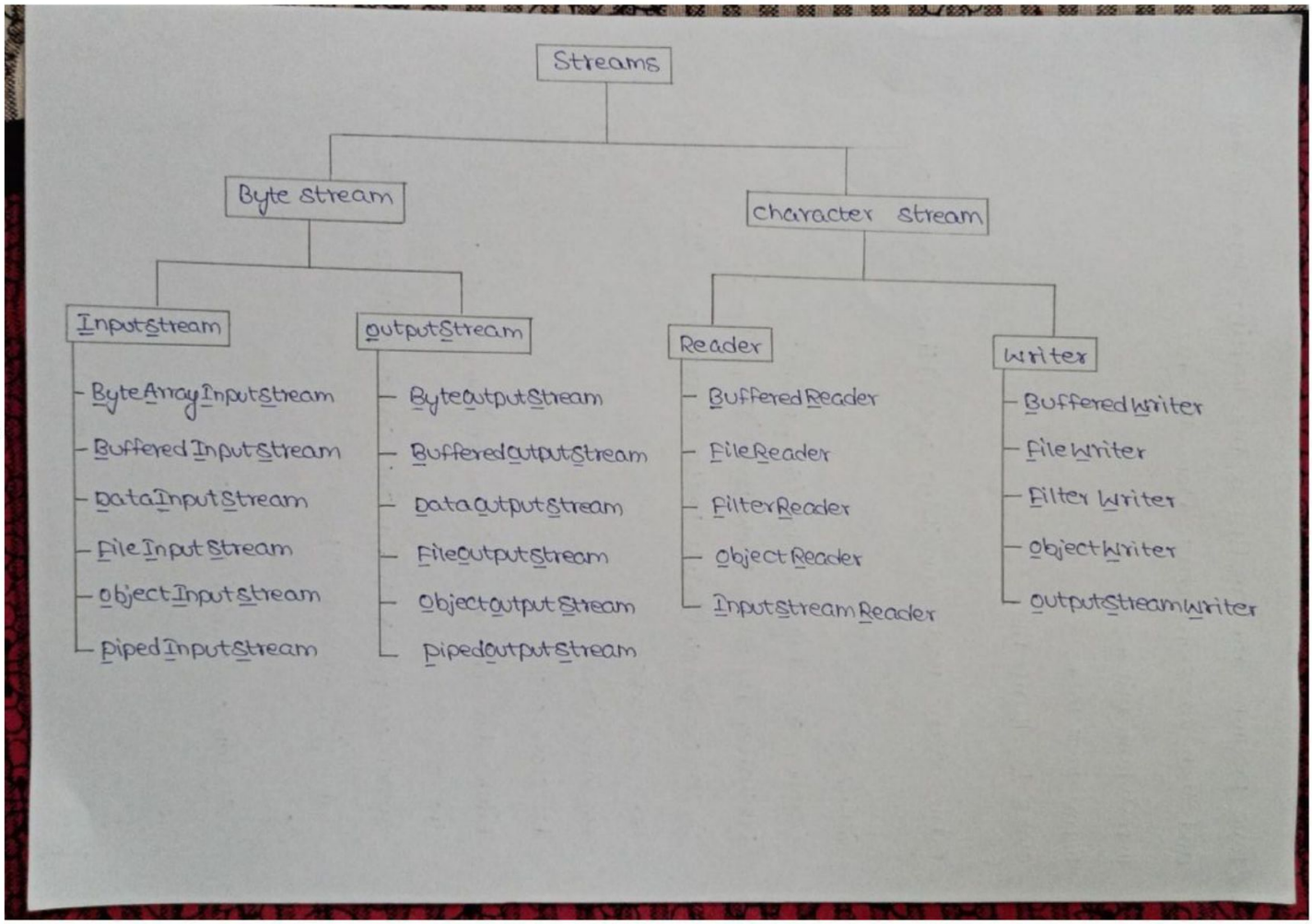
1. Byte Stream
2. Character Stream.

Byte Stream

When an I/O Stream handle data in the form of 8-bit bytes then it is known as Byte Stream.

Character Stream

When an I/O Stream handles data in the form of 16-bit bytes unicode characters then it is known as character stream.


Streams
Byte stream
character stream
InputStream
OutputStream
Reader
Writer
ByteArrayInputStream
BufferedInputStream
DataInputStream
FileInputStream
ObjectInputStream
PipedInputStream
ByteOutputStream
BufferedOutputStream
DataOutputStream
FileOutputStream
ObjectOutputStream
PipedOutputStream
BufferedReader
FileReader
FilterReader
ObjectReader
InputStreamReader
BufferedWriter
FileWriter
FilterWriter
ObjectWriter
OutputStreamWriter

```
//Java program for creating a file, writing data to file
and then accessing data from the file.

import java.util.*;
import java.io.*;
class Readwritefile
{
public static void main(String args[])throws Exception
{
 String str;
 Scanner sc = new Scanner(System.in);
 FileWriter fw = new FileWriter("Simple.txt");
 System.out.println("\n\nIt ENTER A STRING....");
 str = sc.next();
 fw.write(str);
 fw.close();
 FileReader fr = new FileReader("Simple.txt");
 int ch;
 System.out.println("\n\nIt THE CONTENTS OF FILE ARE...:");
 while((ch=fr.read())!=-1)
 {
  System.out.print((char)ch);
 }
}
}
```

```
//Java program to display information about given file.
import java.util.*;
import java.io.*;
class FileInfo
{
public static void main (String args[])throws Exception
{
 String s;
 Scanner sc = new Scanner (System.in);
 System.out.println("\n\nIt ENTER THE FILE NAME....");
 s = sc.nextInt();
 File f = new file (s);
 if (f.exists())
 {
  System.out.print("\n\nIt FILE EXISTS");
 }
 else
 {
 System.out.print("\n\nIt FILE DOES NOT EXISTS");
 }
 if (f.canRead())
 {
 System.out.print("\n\nIt FILE IS READABLE");
 }
```

```
else
{
 System.out.print("\n\nIt FILE IS NOT READABLE");
}
if(f.canWrite())
{
 System.out.print("\n\nIt FILE IS WRITABLE");
 }
 else
 {
  System.out.print("\n\nIt FILE IS NOT WRITABLE");
 }
  System.out.print("\n\nIt THE LENGTH OF THE FILE IS...."+ f.length());
  }
}
```

counting the number of characters, words and lines in the given text file.

proverb.txt

```
BE BRAVE AND FACE THE DIFFICULT SITUATIONS

TIME IS PRECIOUS.
```

Creating a text file

1. Open the Notepad
2. Type the Content
3. Save the file with txt extension.

String s; ⟹ It is used to store the name of the file

int cc, wc, lc;

cc → charactercount ⟹ It is used to store the total number of characters in the file.

wc → wordcount ⟹ It is used to store the total number of words in the given file.

lc → linecount ⟹ It is used to store the total number of lines in the given file.

cc = 0;

wc = 0;

lc = 0;

String str; ⟹ It is used to store single line of a file

String words[] ⟹ It is used to store the words in a single line.

BufferedReader br ⟹ It is used to point to the file through FileReaders object.

FileReader → It is used to point the file.

//Java program to print the number of characters, words and lines in a file.

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

```
Class Count
{
public static void main(String args[])throws Exception
{
 String s, str;
 int cc, wc, lc;
 Scanner sc = new Scanner(System.in);
 System.out.print("\n\nIt ENTER THE FILE NAME....");
 s= sc.next();
 BufferedReader br = new BufferedReader (new FileReader(s));
 cc=0;
 wc=0;
 lc=0;
 While ((str = br.readLine()) != null)
 {
  lc = lc+1;
  String words[] = str. Split (" ");
  wc = wc + words. length;
  for(String x : words)
  {
   cc = cc + x. length();
  }
 }
System.out.print("\n\nIt THE NUMBER OF CHARACTER IN THE
                                    FILE IS...."+cc);
```

```
System.out.print("\n\nIt THE NUMBER OF WORDS IN THE FILE
                                        IS...."+WC);
System.out.print("\n\nIt THE NUMBER OF LINES IN THE FILE
                                        IS...."+ LC);
 }
}
```

```
// Java program to display the contents of a file along
  with its line number
  import java.io.*;
  import java.util.*;
  Class Line
  {
   public static void main (String args[]) throws Exception
   {
    String s;
    Scanner sc = new Scanner (System.in);
    System.out.print("\n\nIt ENTER THE FILE NAME...");
    S= SC.next();
    LineNumberReader lnr = new LineNumberReader (new
                                        FileReader(s));
    String Str;
    While((Str = lnr.readLine()) != null)
    {
```

```
System.out.print("\n\nIt LINE_" + lnr.getLineNumber() + " " + str);
}
}
}
```

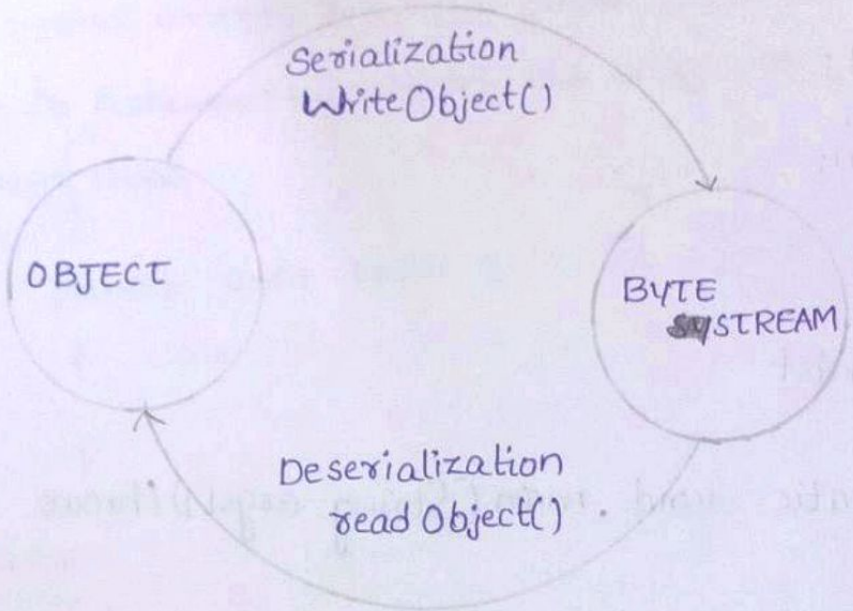
Serialization and Deserialization

Serialization

1. The process of converting an object into byte stream is called Serialization.

Deserialization

The process of Converting byte stream into object is called Deserialization.



ObjectOutputStream ⟹ FileOutputStream
writeObject()

ObjectInputStream ⟹ FileInputStream
readObject()

```
// Java program to create a student class which
Supports Serialization

import java.io.*;
class Student implements Serializable
{
int rollno;
String name;
float marks;
public Student (int r, String s, float m)
{
roll no= r;
 name = s;
marks = m;
 }
}
class Persist
{
public static void main (String args[])throws Exception
{
Student S1= new Student (1000, "Kiran", 75);
ObjectOutputStream OOS= new ObjectOutputStream(new FileOutput-
                                    Stream ("sample.txt");

OOS.WriteObject(S1);
}
}
```

// Java program to create a student class which supports deserialization.

```
import java.io.*;
class Deperist
{
public static void main(String args[]) throws Exception
{
ObjectInputStream OIF = new ObjectInputStream(new
                        FileInputStream("Sample.txt");
Student S1 = Student + (OIS) OIS.readObject();
System.out.println("\n\nIt "Roll no"= +S1.rollno + "name"= +Sr.name +
                   "marks"= +S1.marks);
}
}
```

Boxing

The process of converting primitive data types into its equivalent Wrapper class Object is called boxing.

unBoxing

The process of converting Wrapper class Object into equivalent primitive data type is called unboxing.

primitive data type	Wrapper class
int	Integer
float	Float
double	Double
char	character
boolean	Boolean.

* Java program to illustrate boxing

```
class Boxing
{
public static void main(String args[]) throws Exception
{
int a=5;
Integer x=new Integer(a);
System.out.println("\n\nIt x="+a);
}
}
```

Java program to illustrate unboxing.

```
class Unboxing
{
public static void main(String args[])throws Exception
{
int a;
Integer x= new Integer(50);
a=x;
System.out.println("\n\nIt a="+a);
}
}
```

Enumeration

An enumeration is a list of named constants.

An enumeration defines a class type.

An enumeration is created using enum keyword.

An enumeration can have constructors, methods and instance variables.

Advantages

* Enumeration improves type safety
* Enumeration can be easily used in switch.
* Enumeration can be easily traversed.

E.g.

```
// Java program to create an enumeration.

class Edemo
{
  public enum Season { SPRING, SUMMER, WINTER,
                       AUTUMN }

  public static void main(String args[])
  {
    for (Season s: Season.values())
    {
      System.out.println(s);
    }
  }
```

Java compiler internally adds values(), valueOf() and ordinal() methods within the enum at compile time.

values()

The values() method returns an array containing all the values of enum.

valueOf()

The valueOf() method returns the value of given constant enum.

ordinal()

The ordinal() method returns the index of the enum value.

An enumeration can be created within the class or outside of the class.

Generics

Generics allows us to create a class, interface and method that can be used with different types of data.

Generics does not work with primitive datatypes.

Generic class

A class that can be used with any type of data is known as Generic class.

E.g.

```
// Java program to create a Generic class.

class   GenericsClass <T>
{
    private  T   data;

    public  GenericsClass(T  data)
    {
        this.data = data;
    }

    public  T  getData( )
    {
        return this.data;
    }
}
```

```
class GCdemo
{
 Public static void main(String args[])
 {
  GenericsClass<Integer> x = new GenericsClass<>(5);
  System.out.print("\n\n\t GENERIC CLASS RETURNS-"+
                   x.getData());
  GenericsClass<String> y = new GenericsClass<>
                   ("JAVA");
  System.out.print("\n\n\t GENERIC CLASS RETURNS-"+
                   y.getData());
 }
}
```

output

GENERIC CLASS RETURNS - 5

GENERIC CLASS RETURNS - JAVA.

Generic Method

A method that can be used with any type of data is known as Generic Method.

E.g.

```
// Java program to create a Generic Method

class Demo
{
  public <T> void genericsMethod(T data)
  {
    System.out.println("GENERIC METHOD");
    System.out.println("DATA PASSED : "+ data);
  }
}

class GMdemo
{
  public static void main(String args[])
  {
    Demo d = new Demo();
    d.<String>genericsMethod("JAVA");
    d.<Integer>genericsMethod(25);
  }
}
```

Output

```
GENERIC METHOD
DATA PASSED : JAVA
GENERIC METHOD
DATA PASSED : 25
```

Advantages of Generics

1. code Reusability

With the help of generics, we can write code that will work with different types of data.

2. Compile-time checking

The type parameter of generics provides information about the type of data used in the generics code.

3. Used with Collections

The collections framework uses the concept of generics in java.