Unit V

5 Exception Handling and Generic Programming

5.1 : Exception and Errors

Q.1 Define the terms - Exception and errors.

☞ [SPPU : June-22, Marks 9]

Ans. : • **Exceptions : Exception** is an unusual condition that can occur in the program. This condition is caused due to run time error in the program.

- **Errors :** When any kind of serious problem occurs which could not be handled easily like **OutOfMemoryError** then an error is thrown.

Q.2 State and explain types of errors.

Ans. : • There are two types of errors -

1. **Compile time errors** 2. **Run time errors**

1. Compile time errors : • The errors that are detected by the Java compiler during the compile time are called the compile time errors.

- When compiler issues the errors it will not generate the .class file. Hence we must eliminate all the compile time errors first.
- The most commonly occurring compile time errors are -
  1. Missing semicolons.
  2. Wrong spelling of keywords and identifiers.
  3. Missing brackets of classes and methods.
  4. Use of undeclared variables.

2. Run time errors : • Sometimes the program is free from the compile time errors and creates a .class file. But it does not yield the results as per our expectations. In such situation we declare that the program has run time errors.

- The runtime errors are basically the logical errors that get caused due to the wrong logic.
- The most commonly occurring run time errors are -
  1. Accessing the array element which is out of the index.
  2. In an expression, divide by zero.
  3. Trying to store a value into an array of incompatible type.
  4. Passing the parameters that is not in the valid range.

5.2 : Built-in Exceptions and its Types

Q.3 Give the class hierarchy of exception class.

Ans. : Various exception classes get derived from **Throwable** class. This class hierarchy is given in the following figure.

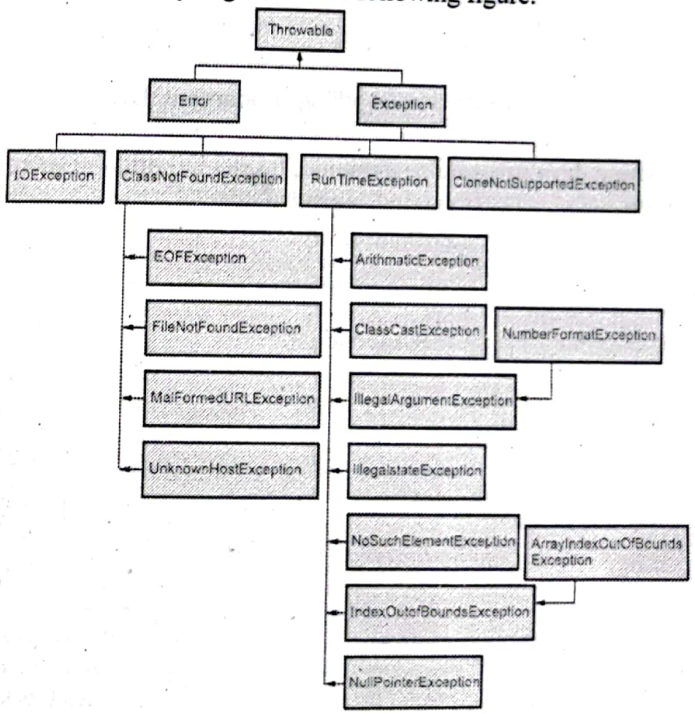

Fig. Q.3.1 Exception hierarchy

Q.4 Define checked and unchecked exceptions.

Ans. : • **Checked exception :** These types of exceptions need to be handled explicitly by the code itself either by using the **try-catch** block or by using **throws**. These exceptions are extended from the **java.lang.Exception** class.

- **For example :** IOException which should be handled using the try-catch block.
- **UnChecked exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these types of exceptions. These exceptions are extended from **java.lang.RuntimeException** class.

For example : ArrayIndexOutOfBounds, NullPointerException RunTimeException.

5.3 : Exception Handling Fundamentals

Q.5 Write a Java program to demonstrate the handling of divide by zero error using exception handling mechanism.

Ans. : Java Program[RunErrDemo.java]

```
class RunErrDemo
{
public static void main(String[] args)
{
int a,b,c;
a=10;
b=0;
try
{
c=a/b;
}
catch(ArithmeticException e)
{
System.out.println("\n Divide by zero");
}
```

Exception occurs because the element is divided by 0.

Exception is handled using **catch** block


```
System.out.println("\n The value of a: "+a);
System.out.println("\n The value of b: "+b);
}
}
```

5.4 : Uncaught Exception

Q.6 Write short note on - uncaught exception.

Ans. :

- If there exists some code in the source program which may cause an exception and if the programmer does not handle this exception then java runtime system raises the exception.
- Following example illustrates how Java runtime system deals with an **uncaught exception.**
- When we use an index which is beyond the range of index then **ArrayIndexOutOfBoundsException** occurs. Following Java program illustrates it.

Java Program [ExceptionProg.java]

```
class ExceptionProg
{
  static void fun(int a[])
  {
  int c;
  c=a[0]/a[2];
   }
   public static void main(String args[])
   {
   int a[]={10,5};
   fun(a);
   }
}
```

5.5 : Multiple Catch Clauses

Q.7 Explain the details of multiple catch statement.

Ans. :

- It is not possible for the try block to throw a single exception always.
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.
- The **syntax** for single try and multiple catch is -

```
try
{
  ...
  ...//exception occurs
}
catch(Exception_type e)
{
  ...//exception is handled here
}
  catch(Exception_type e)
{
  ...//exception is handled here
}
  catch(Exception_type e)
{
  ...//exception is handled here
}
```

Example

Java Program[MultipleCatchDemo.java]

```
class MultipleCatchDemo
{
  public static void main (String args [ ])
```



```
  {
    int a[] = new int [3];

    try
    {
      for (int i = 1; i <=3; i++)
      {
      a[i] = i *i;
      }

      for (int i = 0; i <3; i++)
      {
        a[i] = i/i;
      }
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      System.out.println ("Array index is out of bounds");
    }
    catch (ArithmeticException e)
    {
      System.out.println ("Divide by zero error");
    }
  }
}
```

5.6 : Nested Try Statement

Q.8 What is nested try statement ? Explain.

Ans. :

- When there are chances of occurring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.
- Following is an example of nested try-catch statements -

Java Program[NestedtryDemo.java]

```
class NestedtryDemo
{
  public static void main (String[] args)
```

```
    {
try
    {
        int a = Integer.parseInt (args [0]);
        int b = Integer.parseInt (args [1]);
        int ans = 0;

      try
      {
        ans = a / b;
        System.out.println("The result is "+ans);
      }
      catch (ArithmeticException e)
      {
        System.out.println("Divide by zero");
      }
    }
    catch (NumberFormatException e)
    {
      System.out.println ("Incorrect type of data");
    }
  }
}
```

Inner try - catch

Outer try - catch

5.7 : Using Throws and Throw

Q.9 Write a Java program to demonstrate the throws keyword.
Ans. :

Java Program

```
    /* This programs shows the exception handling mechanism
using throws
    */
    class ExceptionThrows
    {
    static void fun(int a,int b) throws ArithmeticException
    {
    int c;
     try
```



```
    {
     c=a/b;
    }
    catch(ArithmeticException e)
    {


      System.out.println("Caught exception: "+e);
      }
    }

    public static void main(String args[])
    {
     int a=5;
     fun(a,0);
    }
    }
```

Q.10 Write a Java program to demonstrate the throw keyword.
Ans. :

```
class ExceptionThrow
{
  static void fun(int a,int b)
  {
    int c;
    if(b==0)
     throw new ArithmeticException("Divide By Zero!!!");
   else
        c=a/b;
  }
  public static void main(String args[])
  {
    int a=5;
    fun(a,0);
  }
}
```

5.8 : User Defined Exception using Throw

Q.11 What are the steps to develop user defined exception ? [SPPU : June-22, Marks 5]

Ans. :

- We can throw our own exceptions using the keyword **throw**.
- The syntax for throwing out own exception is -

```
throw new Throwable's subclass
```

- Here the Throwable's subclass is actually a subclass derived from the **Exception** class.

Q.12 Write a program to input name and age of a person and throws an user define exception if entered age is negative.

Ans. :

```
import java.lang.Exception;
import java.util.Scanner;
class AgeException extends Exception
{
    AgeException(String msg)
    {
        super(msg);
    }
}
class MyExceptDemo3
{
    public static void main (String args [])
    {
        System.out.print("Enter your age: ");
        Scanner in=new Scanner(System.in);
        int age=in.nextInt();
        try
        {
          if(age<0)
            throw new AgeException("Negative Age!!!");
          else
```



```
            System.out.println("Correct Age");
        }
        catch (AgeException e)
        {
          System.out.println (e.getMessage());
        }
    }
}
```

Output

```
Enter your age: -10
Negative Age!!!
```

5.9 : What are Generics ?

Q.13 What is generic ? Explain its need.

Ans. :

- Generic is a mechanism for creating a general model in which generic methods and generic classes enable programmers to specify a single method (or a set of related methods) and single class (or a set of related classes) for performing the desired task.
- Following features show the importance of generics in Java.
  1. It saves the programmers burden of creating separate methods for handling data belonging to different data types.
  2. It allows the **code reusability**.
  3. **Compact code** can be created.

Q.14 Explain the concept of generic classes with example.

Ans. : • A generic class or interface contains one or more variables of generic data type. Following is a simple example which shows how to define the generic class.

```
public class Test<T>
{
```

```
    public Test(){val=null;}
    public Test(T val)
    {
       this.val=val;
    }
    public getVal()
    {
      return val;
    }
    public setVal()
    {
      val=newValue;
    }
    private T val;  //variable defined as of genric type
}
```

The concept of generic class or interface supports the idea of type-independency.

5.10 : Introduction to Language Specific Collection Interface

Q.15 Write a short note on - Collection framework.

Ans. :

- The standard data structures can be implemented in Java using some **library classes** and methods.
- These classes are present in the **java.util** package.
- The collection framework is comprized of **collection classes** and **collection interfaces.**
- Basically collection is a group of objects which are designed to perform certain task. These tasks are associated with alteration of data structures.
- The collection classes are the group of classes used to implement the collection interfaces.

- Various collection classes are

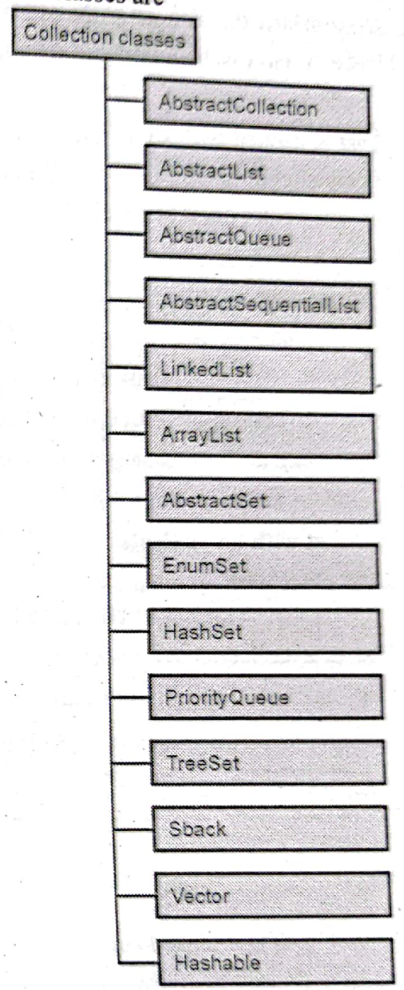

Fig. Q.15.1 Collection class hierarchy

Q.16 What is ArrayList ?

Ans. : • The **ArrayList** class implements the List interface. It is used to implement the dynamic array.

- The dynamic array means the size of array can be created as per requirement. Hence **ArrayList** is a variable length array of object references.
- Initially **ArrayList** is created with some initial size and then as per requirement we can extend the size of array. When the objects are removed then the size of array can be reduced.
- The syntax of using **ArrayList** is as given below -

```
ArrayList() ← Creates an empty list
ArrayList(Collection collection) ← Creates a list in which the
                                   collection elements are added
ArrayList(int c) ← Creates a list with specified capacity c, the
                   capacity represents the size of the
                   underlying array
```

Q.17 Explain Linked List with an example.

OR Explain linked list class with an example.

☞ **[SPPU : June-22, Marks 9]**

Ans. : • Linked List is a collection of nodes in which every node contains two fields Data and Next pointer fields.

- The link list can be graphically represented as follows -

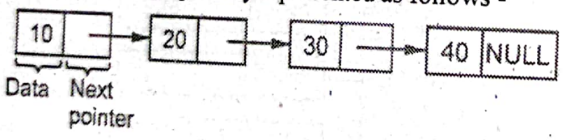

Fig. Q.17.1 Linked list

- The java.util package provides the collection class LinkedList in order to implement the List interface.
- The syntax of using this class is

```
Linked List() ← creates an empty linked list
Linked List(Collection collection) ← creates a linked list having
                                     the elements of collection
```


Programming Example : Following program makes use of various methods such as **add(), remove(), addFirst(), addLast(), removeFirst(), removeLast()** for manipulating the contents of linked list.

```
import java.io.*;
import java.util.*;
class LinkedListProg
{
  public static void main(String[] args)throws IOException
  {
    char ans='y',ch='y';
    int choice,val,position;
   String str;
    LinkedList obj=new LinkedList();
   Scanner s=new Scanner(System.in);
    do
    {
      System.out.println("\n\t\t Program for Implementing Linked
List");
      System.out.print("\n1.Create\n2.Display \n3.Insert First\n4.
Insert Last");
      System.out.print("\n5.Delete First\n6.Delete Last");
      System.out.print("\n7.Insert At any Position");
      System.out.println("\n8.Delete From any Position");
      System.out.println("Enter Your choice");
      choice=s.nextInt();
      switch(choice)
      {
        case 1:
          do
          {
            System.out.println("\n Enter the element to be inserted in
the list");
            val=s.nextInt();
            obj.add(val);
            System.out.println(" Do u want to insert more
elements?");
```

```
            str =s.next();
                ans=str.charAt(0);

            }while(ans=='y');
            break;
          case 2:
            System.out.println("\t The List elements are... "+obj);
            System.out.println("\t The size of linked list is...
"+obj.size());
            break;
          case 3: System.out.println("\n Enter the element to be
inserted in the list");
            val=s.nextInt();
            obj.addFirst(val);
            System.out.println("\n The element inserted!!!");
            break;
          case 4: System.out.println("\n Enter the element to be
inserted in the list");
            val=s.nextInt();
            obj.addLast(val);
            System.out.println("\n The element inserted!!!");
            break;
          case 5: obj.removeFirst();
            System.out.println("\n The element deleted!!!");
            break;
          case 6: obj.removeLast();
            System.out.println("\n The element deleted!!!");
            break;
          case 7:System.out.println("\n Enter the element to be inserted
in the list");
            val=s.nextInt();
            System.out.println("\n Enter the position at which the
element is to be inserted");
            position=s.nextInt();
            obj.add(position,val);
            System.out.println("\n The element inserted!!!");
            break;
```



```
          case 8: System.out.println("\n Enter the position of element to
be deleted");
            position=s.nextInt();
            obj.remove(position);
            System.out.println("\n The element deleted!!!");
            break;
        }
        System.out.println("\n Do u want to go to main menu?");
        str=s.next();
        ch=str.charAt(0);
      }while(ch=='y');
    }
}
```

Q.18 Give the difference between ArryaList and LinkedList collection.

Ans. :

Sr. No.	ArrayList	LinkedList
1.	ArrayList internally uses dynamic array to store the elements.	LinkedList internally uses doubly linked list to store the elements.
2.	It internally uses array. If any element is removed from the array, all the bits are shifted in memory.	It uses doubly linked list so no bit shifting is required in memory.
3.	Manipulation with ArrayList is slow	Manipulation with LinkedList is faster than ArrayList
4.	ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
5.	ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data.

Q.19 What is set interface ? Enlist some commonly used methods of set interface.

Ans. :

- The set interface is used to define the set of elements.
- It extends the collection interface.
- This interface defined unique elements. Hence if any duplicate elements is tried to insert in the set then the add() method returns false.
- Some commonly used functionalities in Set interface are -

Method	Description
add( )	Adds an object to the collection.
clear( )	Removes all objects from the collection.
contains( )	Returns true if a specified object is an element within the collection.
isEmpty( )	Returns true if the collection has no elements.
iterator( )	Returns an Iterator object for the collection, which may be used to retrieve an object.
remove( )	Removes a specified object from the collection.
size( )	Returns the number of elements in the collection.

END...