Exception Handling

Exception

Is an abnormal condition that arises in a code sequence at runtime.

Java Exception is an object that describes an exceptional condition that has occured in a piece of code.

When an exceptional condition arises, an Object representing that exception is created and thrown in the method that causes the error.

The method may choose to handle it or pass it to its caller.

Caller may choose to handle it or pass it to its caller. And so on till main method.

If main is also not interested to process the exception, then it will be thrown to JRE, which uses default Exception Handler mechanism to process it.

Exception can be generated by JRE or can be manually generated.

keywords

Exception handling is managed via the following keywords

try

catch

finally

Throw

Throws

try block

Program statements that you want to monitor for exception are kept in the try block.

If an exception occured in the try block, it is thrown to its catch block.

Catch block

To process the Exception

That is write the code (may vary for each type of exception) that should get executed when an exception is raised.

A try block can follow multiple catch blocks.

Each catch block can process a specific type of exception.

In multiple catch statements it is imortant to remember that exception subclasses must come before any of their super classes.

From java 1.7 , java support multi catch blocks.

finally block

This block get executed after the completion of try block.

This block gets executed irrespective of whether the exception is raised or not, if raised wheter it is handled or not.

Useful for closing file handles and freeing up of any other resources that might have been allocated at the begining.

Every try block must be followed by at least one catch block or finally block.

try with resources

Runtime automatically closes the resources (if they are not already closed) after the excecution of the try block.

```
try( BufferedReader br=
        new FileReader("ex1.txt") )   {
          }
catch(){}
```

This works only on classes that have implemented AutoCloseable interface.

Exception Types

Throwable

Exception

ClassNotFoundException*(checked)*

IOException

InterruptedException

RuntimeException*(un checked)*

ArithmeticExcepton

NumberFormatException

ArrayIndexOutOfBoundsException

ExceptionInInitializerError

Error

Un caught Exceptions

```
Class Ex1
```

```
{
  pubic static void main(String args[])
  {
    int d=0;
    int a=40/d;
  }
}
```

Exception caught by default Exception handler of JRE.

Displays the string describing the exception, prints the stack trace from the point at which exception occured and terminates the program.

Java.lang.ArithmeticException: / by Zero

at Ex1.main (Ex1.java : 6)

Example2

```
Class Ex1
{
 static void meth1(){
   int d=0;  int a=40/d;
     }
  pubic static void main(String args[])
  {
    meth1();
  }
}
```

Exception raised in meth1(), not handled, thrown to main(), not handled, thrown to JRE.Exception caught by default Exception handler of JRE.

prints the stack trace as follows

Java.lang.ArithmeticException: / by Zeo

at Ex1.meth1 (Ex1.java : 4)

at Ex1.main (Ex1.java : 8)

Example_Porgram:

/*program with the following statements and observe the output.

```
System.out.prinrtn("Hello");
System.out.printtn("KMIT");
System.out.prinrtn(10/0);
System.out.prinrtn("Have a nice day");
System.out.prinrtn("Bye");
*/
class demo1
{
   public static void main(String args[])
   {
        System.out.println("Hello");
        System.out.println("KMIT");
        System.out.println(10/0);
        System.out.println("Have a nice day");
        System.out.println("Bye");
   }
}
```

Advantage of Handling an Exception

Stops abnormal termination.

Instead of printing the stack trace we can display user friendly messages to the user related to the exception.

Allows us to fix the error.

Example_Program:

/* to demonstrate advantage of handling exception.

program to get the following output in same order

Use exception handling

Hello

KMIT

10/0 is invalid operation

All the Best

```
Bye
*/
class demo2
{
   public static void main(String rags[])
   {
   System.out.println("Hello");
   System.out.println("KMIT");
   try
   {

      System.out.println(10/0);
   }
   catch(ArithmeticException ae)
   {
      System.out.println(ae.getMessage());
      System.out.println(ae);
   }
   System.out.println("All the Best");
   System.out.println("Bye");
   }
}
```

Example_Program:

```
//demonstration of finally block.
class demo3
{
  public static void main(String[] args)
  {
    int a=1,b=0,c;
    try
    {
```

```
    c=a/b;
   }
   catch(ArithmeticException ae)
   {
   System.out.println(" Error: denominotor cant be zero");
   }
   finally
   {
     System.out.println("This will execute");
   }
  }
}
```

Example_Program:

```
// To demonstration finally block another example
class demo4
{
  public static void main(String[] args)
  {
   int a=10,b=5,c;
   try
   {
    c=a/b;
    System.out.println("result= "+c);
   }
   catch(ArithmeticException ae)
   {
   System.out.println(" Error "+ae);
   }
   finally
   {
     System.out.println("This will execute");
```

```
    }
    }
}
```

Example_Program:

```
/*Program to demonstrate ArrayIndexOutOFBoundsException*/
class demo5
{
    public static void main(String args[])
    {
        int a[]={10,20,30,40,50};
        System.out.println(a[0]);
        System.out.println(a[1]);
        try
        {
            System.out.println(a[5]);
        }
        catch(ArrayIndexOutOfBoundsException ae)
        {
          System.out.println("Invalid index");
        }
        System.out.println(a[2]);
        System.out.println(a[3]);
        System.out.println(a[4]);
    }
}
```

Example_Program:

```
/*program to demonstrate NullPointerException for Scanner class object
*/
import java.util.*;
class demo6
{
```

```
    public static void main(String args[])
    {
        Scanner s=null;
        try
        {
            int a=s.nextInt();
        }
        catch(NullPointerException ne)
        {
            System.out.println(ne);
        }
    System.out.println("End of the program");
    }
}
```

Multiple Catch blocks

Each catch block is to process the Exception

That is write the code (may vary for each type of exception) that should get executed when an exception is raised.

A try block can follow multiple catch blocks.

Each catch block can process a specific type of exception.

In multiple catch statements it is imortant to remember that exception subclasses must come before any of their super classes.

From java 1.7 , java support multi catch blocks.

Multi-catch block

Allows two or more exceptions to be caught by the same catch clause.

Used when two or more catch blocks has the same code.

Each multi catch parameter is implicitly final. So it can't be assigned a new value.

Example

```
  int a=10,b=0;  int vals[]={1,2,3};
    try{
```

```
    int result=aa/b; vals[10]=19;
        }
  catch(ArithmeticException |
   ArrayIndexOutOfBoundsException e)
  {
   s.o.p("Exception caught:" + e);
  }
```

Example_Program:

```
//To demonstrate multiple catch blocks
class demo7
{
  public static void main(String[] args)
  {
    int a[]={10,20,30,40,50,60};
    try
    {
//      int c=a[10]/0;
        int c=a[1]/0;
      System.out.println("result= "+c);
    }
    catch(ArithmeticException ae)
    {
    System.out.println(" Error "+ae);
    }
    catch(ArrayIndexOutOfBoundsException ab)
    {
     System.out.println("Error : "+ab);
    }
    finally
    {
```

```
      System.out.println("this will execute");
    }
  }
}
```

Example_Program:

```
//To demonstrate multi-catch block
class demo8
{
  public static void main(String[] args)
  {
    int a[]={10,20,30,40,50,60};
    try
    {
      int c=a[10]/0;
    //    int c=a[1]/0;
      System.out.println("result= "+c);
    }
    catch(ArithmeticException | ArrayIndexOutOfBoundsException ae)
    {
     System.out.println("Error : "+ae);
    }
    finally
    {
      System.out.println("this will execute");
    }
  }
}
```

Example_Program:

```
//To demonstrate catch all catch block
class demo9
{
```

```
  public static void main(String[] args)
  {
    int a[]={10,20,30,40,50,60};
    try
    {
      int x=Integer.parseInt("5a");
      int c=a[10]/0;
      System.out.println("result= "+c);
    }
    catch(ArithmeticException | ArrayIndexOutOfBoundsException ae)
    {
     System.out.println("Error : "+ae);
    }
    catch(Exception e)
    {
      System.out.println("Error:"+e);
    }
    finally
    {
      System.out.println("this will execute");
    }
  }
}
```

Exceptions with Nested try blocks

Example_Program:

```
import java.util.*;
class demo10
{
public static void main(String args[])
{
```

```
  Scanner s=new Scanner(System.in);
  System.out.println("enter the number");
  int a=s.nextInt();
  try
  {
    int b=10/a;
    System.out.println(b);
    try
    {
      if(a==1) a=a/(a-a);
      else
      {
        int c[]={1,2,3};
        c[4]=4;
      }
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
     System.out.println("Invalid index");
    }
  }
  catch(ArithmeticException e)
  {
    System.out.println("Divide be zero Error");
  }
}
}
```

Exceptions with Nested method calls

Example_Program:

```
import java.util.*;
class demo11
```

```
{
static void meth1(int a)
{
    try
     {
       if(a==1) a=a/(a-a);
       else
       {
         int c[]={1,2,3};
         c[4]=4;
       }
     }
     catch(ArrayIndexOutOfBoundsException e)
     {
      System.out.println("Invalid index");
     }
}
public static void main(String args[])
{
  Scanner s=new Scanner(System.in);
  System.out.println("enter the number");
  int a=s.nextInt();
  try
  {
     int b=10/a;
     System.out.println(b);
     meth1(a);
  }
  catch(ArithmeticException e)
  {
    System.out.println("Divide be zero Error");
```

```
  }
}
}
```

Throw

To Manually throw an exception object.

Syntax:

```
  throw    throwable_instance.
```

The object thowable_instance must be an instance of Throwable class type.

When you manually throw an exception object from the try block ,it checks for the matching catch block among the catch blocks which follows the try block.

If there is a match, the exception is caught by that catch block and it can process that exception(optional).

After processing, if needed the method can rethrow the exception object to its caller to give an oppurtunity to the caller to know about this exception and to process.

This caller can rethrow the exception object to its caller. And so on.

If ther is no matching catch block, the exception object is thrown to its caller.

If the caller also does not handle that exception, it will be thrown to its caller and so on till the main method.

If main method also do not handle it, it will be thrown to JRE.

At a time we can throw only one Exception object.

If we need to report more that one exception, we need to chain the exceptions with their cause exceptions.

Example_Program:

```
// To demonstrate throw keyword
class demo12
{
 static void meth1()
 {
 try
 {
   int x=12/0;
```

```
 }
 catch(ArithmeticException e)
 {
  System.out.println("caught inside method:"+e);
  throw e;
 }
 }
 public static void main(String args[])
 {
  try
  {
   meth1();
  }
  catch(ArithmeticException e)
  {
   System.out.println("Recaught:"+e);
  }
 }
}
```

Example_Program:

```
// To demonstrate throw keyword- another example
class demo13
{
 static void meth1()
 {
 try
 {
  throw new NullPointerException("demo");
 }
 catch(NullPointerException e)
 {
```

```
  System.out.println("caught inside method:"+e);
  throw e;
 }
 }
 public static void main(String args[])
 {
  try
  {
   meth1();
  }
  catch(NullPointerException e)
  {
   System.out.println("Recaught:"+e);
  }
 }
}
```

Throws

If a method is capable of causing an exception that it does not handle, it must specify this behaviour so that callers of this method can guard themselves against that exception.

Not necessary for exceptions that are subclasses of RuntimeException class.

thows clause for an overriden method

The overridden method can declare the same exceptions as that of its super class.

The overridden method can be defined as not throwing any exceptions.

Can add any new RuntimeExceptons (un checked).

Can't and any new Checked Exceptions.

Can narrow down to the Exception subclasses declared in the parent class.

Can't specify super classes of the chekced exceptions declared in the parent class.

final rethrow /more precise rethrow

```
Class Ex1{
public static meth1(int x) throws IOException,InterruptedException {
try{
if(x==0) throw new IOException();
else throw new InterruptedException();
}
catch(Exception e){
throw e;
}}

public static void main(String args[])
{
    try
   {
     meth1(0);
    }
    catch(IOException e){}
    catch(InterruptedException e){}
} }
```

Example Program:

```
// To demonstrate throws keyword
import java.io.*;
class demo14
{
 static void meth1() throws FileNotFoundException
 {
  FileInputStream fis=null;
  fis=new FileInputStream("ex1.dat");
 }
 public static void main(String args[])
 {
  try
  {
   meth1();
  }
  catch(FileNotFoundException e)
```

```
    {
      System.out.println("plz check your filename");
    }
  }
}
```

Example_Program:

```
// To demonstrate final rethrow

import java.io.*;

class demo15
{
  static void meth1(int x) throws     IOException,InterruptedException
  {
   try
   {
      if(x == 0)
        throw new IOException();
      else
        throw new InterruptedException();
   }
   catch(Exception e)
   {
     throw e;
   }

 }
  public static void main(String args[])
  {
    try
    {
      meth1(0);
    }
    catch(IOException | InterruptedException e)
    {
     System.out.println(e);
    }
  }
}
```

User defined exceptions

we can create our own Exceptions by extending from the class Exception.

Example_Program:

```
// User defined exceptions
import java.util.*;
class Vote extends Exception
{
  Vote(String str)
  {
   super(str);
  }
}

class demo16
{
  public static void meth1(int age) throws Vote
  {
      if(age<18)
         throw new Vote("Not eligible to vote");
      System.out.println("Eligible to vote");
  }
 public static void main(String[] args)
  {
    Scanner s=new Scanner(System.in);
    System.out.println("enter your age:");
    int age=s.nextInt();
    try
    {
     meth1(age);
    }
   catch(Vote v)
   {
    System.out.println("Error  "+v);
   }
 }
}
```