CHAPTER - 7

EXCEPTION HANDLING

7. Exception:

An exception is an abnormal event that rises during the execution of a program and disturbs the normal flow of instructions i.e., exception is a run-time error.

Consider the following example:

```
class hello
{
public static void main(String[] args)
{
        int a=10,b;
        b=a/0;
        System.out.println("value="+b);
}
}
```

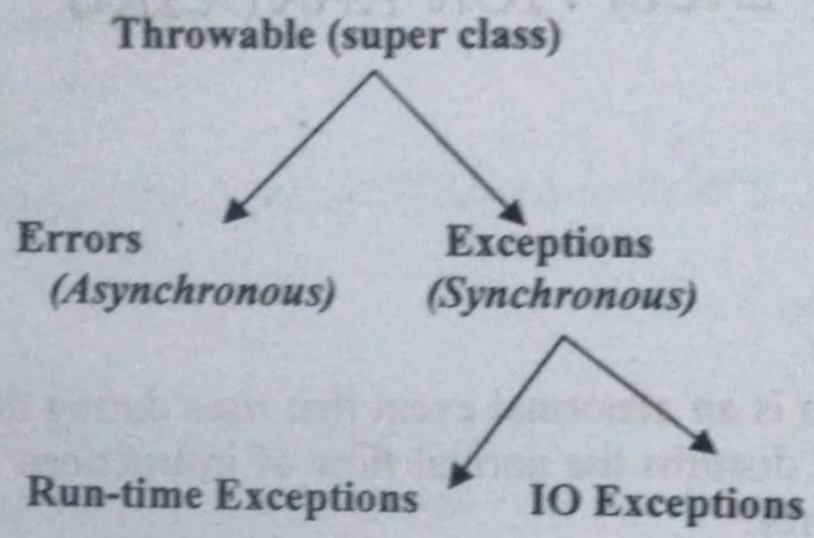
When we try to compile this program, it does not produce any errors. But when it runs the program it produces a message as "java.lang.ArithmeticException: / by zero". This exception error message is displayed and the program terminates without executing the remaining statements.

7.1. Exception Handling:

Exception Handling is designed for error processing. It enables a program to catch all types of exceptions of certain type or related type. Exception handling is designed for dealing with Synchronous errors such as an attempt to divide by zero, out of bounds exception etc., It is not designed to deal with Asynchronous errors such as mouse checks, key strokes etc.,

Exception Handling is used in situation arises the system can recover from causing the exception. The procedure is called Exception Handling and done by the Exception Handler.

All exception types are subclasses of built-in class Throwable. It is a super class. The class is available in java.lang package. It contains errors and exceptions.



Java defines several exception classes inside the java.lang package. These are all subclass of Run-time Exceptions. The following are the some of the examples of exceptions and these are unchecked exceptions because the compiler does not check to see if a method handles or throws these exceptions.

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero
ArrayIndexOutOfBoundsException	Array index is out-of-bounds
IllegalArgumentException	Illegal argument used to invoke a method
IllegalThreadStateException	Requested operation not compatible with current thread state
IndexOutOfBoundsException	Some type of index is out-of-bounds
NegativeArraySizeException	Array created with a negative size
NullPointerException	Invalid use of a null reference
NumberFormatException	Invalid conversion of a string to a numeric format
StringIndexOutOfBounds	Attempt to index outside the bounds of a string

java.lang package contains some another type of exceptions that does not handle itself. Here, the compiler checks that what is to be done when this arise, because of this checking that they are called **checked exceptions**. Examples for checked exceptions are

Exception	Meaning
ClassNotFoundException	Class not found
IllegalAccessException	Access to a class is denied
InterruptedException	One thread has been interrupted by another thread
NoSuchMethodException	A requested method does not exist

7.2. Exception-Handling Fundamentals

Java exception handling is managed through five keywords. They are 1) try 2)catch 3) throw 4) throws 5) finally. The general format of an exception-handling block is as

Syntax:
```
try
{
    // block of code to monitor for errors
}
catch (ExceptionType1 exob)
{
    // exception handler for exception type1
}
catch (ExceptionType2 exob)
{
    // exception handler for exception type2
}
.
.
finally
{
   // block of code to be executed before of try block ends
}
```

Here, ***ExceptionType*** is the type of exception that has occurred and ***exob*** is the exception object.

7.2.1 try block:

The statements that are produces exception are identified in the program and the statements are placed with in a ***try block***

Syntax:
```
try
{
    // block of code to monitor for errors
}
```

If an exception occurs within the try block, the appropriate exception handler (catch block) associated with the try block handles the exception immediately.

7.2.2 catch block:

The ***catch block*** is used to process the exception raised. The catch block is placed immediately after the try block.

Syntax: catch (ExceptionType exob)

```
{
    // exception handler for exception type
}
```

7.2.3 *finally block:*

This block is placed after the last catch block. This block is executed regardless of whether or not an exception is raised.

Syntax:
```
finally
{
  // block of code to be executed before of try block ends
}
```

1. The control enters into the try block and executes the statements. If an exception is raised, the exception throws and it caught by exception handler catch block. And the particular catch block statements are executed. The control never backs again to the try block. If no exceptions raised, catch blocks are not executed the control directly passed to the finally block.
2. No statements are placed between try block and catch block.

Example 1:
```
class ex11
{
    public static void main(String[] args)
    {
        try
        {
        int a=10,b;
        b=a/0;
        System.out.println("b="+b);
        }
        catch(ArithmeticException e)
        {
        System.out.println("Exception raised");
        }
        System.out.println("Quit");
    }
}
```
O/P: Exception raised
Quit

Example 2:

```
class ex12
{
    public static void main(String[] args)
    {
        int x[]=new int[10];
        try
        {
        x[20]=100;
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
        System.out.println("Exception caught");
        }
        finally
        {
        System.out.println("Not possible to print");
        System.out.println("Quit");
        }
    }
}
```

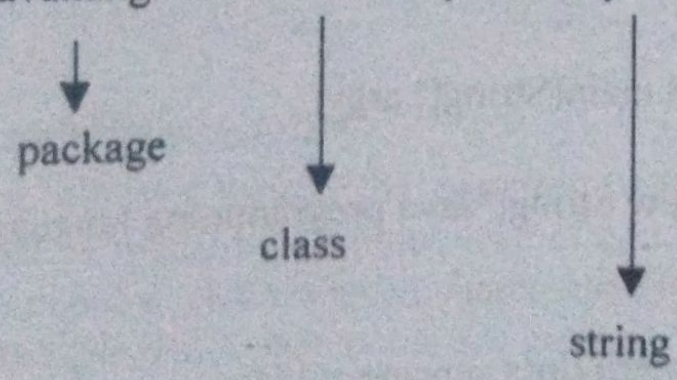
7.3 Methods of Exception Object

a) **String getMessage( ):** It returns the descriptive string stored in an exception.

Example: _ / by zero

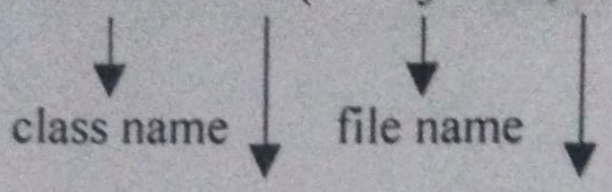
b) **String toString( ):** It returns an error message with the class name of the exception the descriptive string to be stored in the exception.

Example: java.lang.ArithmeticException: / by zero



c) **void *printStackTrace*( ):** This is the standard output error stream. It displays the error message with the class name of the exception the descriptive string to be stored in the exception along with the line number and name of the program.

Example: java.lang.ArrayIndexOutOfBoundsException
at ex12.main (ex12.java:8)



Example 1:

```
class ex12
{
    public static void main(String[] args)
    {
        int x[]=new int[10];
        try
        {
        x[20]=100;
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
        System.out.println(e.getMessage());
        System.out.println(e.toString());
        e.printStackTrace();
        }
    }
}
```

Example 2:

```
class ex12
{
    public static void main(String[] args)
    {
        String s=new String("Java programming language");
        try
        {
        System.out.println(s.substring(45));
```

```
        }
        catch(StringIndexOutOfBoundsException e)
        {
        System.out.println(e.getMessage());
        System.out.println(e.toString());
        e.printStackTrace();
        }
    }
}
```

Example 3:

```
import java.text.DecimalFormat;
class n
{
    public static void main(String[] args)
    {
        try
        {
        double x=25.6776;
        DecimalFormat d=null;
        System.out.println(d.format(x));
        }
        catch(NullPointerException e)
        {
        System.out.println(e.getMessage());
        System.out.println(e.toString());
        e.printStackTrace();
        }
    }
}
```

7.4. Multiple Catch Statements

Multiple catch clauses handle the situation where more than one exception could be raised by a single piece of code. In such situations specify two or more catch blocks, each specify different type of exception. Multiple catch statements are arranged in respective order of exceptions that araised by the try block (optional).

Example:

```
class cat
{
        public static void main(String[] args)
        {
        try
        {
        int a=args.length;
        System.out.println("a="+a);
        int b=42/a;
        int c[]={1};
        c[42]=99;
        }
        catch(ArithmeticException e)
        {
        System.out.println(e.getMessage());
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
        System.out.println(e.getMessage());
        }
        }
}
```

Note: A subclass must come before their super classes in a series of catch statements. Exception class is the super class of all exceptions.

Example:

```
class cat1
{
      public static void main(String[] args)
      {
      try
      {
      int a=0;
      int b=42/a;
      }
     catch(ArithmeticException e)
     {
```

```
System.out.println(e.getMessage());
}
catch(Exception e)
{
System.out.println(e.getMessage());
}
}
}
```

7.5 Nested try Statements

A try block is placed inside the block of another try block is termed as Nested try block statements. If any error statement is in outer try block it goes to the corresponding outer catch block. If any error statement is in inner try block first go to the inner catch block. If it is not the corresponding exception next goes to the outer catch, which is also not corresponding exception then terminated.

Example:

```
class ntry
{
    public static void main(String[] args)
    {
        try
        {
        int a=args.length;
        int b=42/a;
        System.out.println("a="+a);
        try
        {
        if(a==1)
        a=a/(a-a);
        if(a==2)
        {
        int c[]={1};
        c[42]=99;
        }
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
```

```
            System.out.println("Array Exception:"+e);
            }
            }
            catch(ArithmeticException e)
            {
            System.out.println("Arithemetic exception:"+e);
            }
        }
}
```

Example:

```
class ntry1
{
    public static void main(String[] args)
    {
            try
            {
            int a=args.length;
            int b=42/a;
            System.out.println("b="+b);
            }
            catch(ArithmeticException e)
            {
            System.out.println("Arithemetic exception:"+e);
            }
            finally
            {
            System.out.println("final block");
            }
            System.out.println("after try block");
    }
}
```

7.6 Throw statement

All previous exceptions are catching that are thrown by the java run time system. It is also possible to create a program that throws an exception explicitly, using the "throw" statement. The general format of throw statement is

Syntax: throw throwable-instance;

Here, throwable-instance must be an object type of Throwable class or subclass of Throwable. There are two ways to obtain a Throwable object.

1. Using a parameter into a catch clause
2. Creating one with the new operator

Example:

```
class ntry2
{
    public static void main(String[] args)
    {
        try
        {
        throw new ArithmeticException("hello");
        }
        catch(ArithmeticException e)
        {
        System.out.println(e.getMessage());
        }
    }
}
```

O/P: hello

Example:

```
class t
{
    public t()
    {
    try
    {
    throw new NullPointerException("demo");
    }
    catch(NullPointerException e)
    {
    System.out.println("caught");
    throw e;   //rethrow the exception
```

```
        }
        }
}
class ntry2
{
        public static void main(String[] args)
        {
                try
                {
                t obj=new t();
                }
                catch(NullPointerException e)
                {
                System.out.println("Recaught");
                }
        }
}
```

O/P: caught
Recaught

7.7 Throws statement

If a method is capable of causing an exception that it doesn't handle, it must specify the behavior to the callers of the method can guard themselves against that exception. This can be done by throws statement in the method declaration.

A **throws** statement lists the types of exceptions that a method might throw. This is necessary for all exceptions except that those of type **Error** or RuntimeException, or any of their subclasses. All other exceptions that a method can throw must be declared in the **throws** statement. Otherwise, it reports compile time error. The general format of **throws** statement is as

Syntax:
```
type method-name(parameter-list) throws exception-list
{
    // method body
}
```

Here, exception-list is a comma-separated list of the exceptions that a method can throw.

Example:

```
class t
{
    public t()throws NullPointerException
    {
    System.out.println("caught");
    throw new NullPointerException("demo");
    }
}
class ntry3
{
    public static void main(String[] args)
    {
        try
        {
        t obj=new t();
        }
        catch(NullPointerException e)
        {
        System.out.println("Recaught");
        }
    }
}
```

O/P: caught
Recaught

7.8 User-defined Exceptions

It is also possible to create our own exceptions types to handle situations specific to our application. Such exceptions are called User-defined Exceptions. User-defined exceptions are created by extending **Exception** class. The **throw** and **throws** keywords are used while implementing user-defined exceptions.

Example:

```
class own extends Exception
{
      own(String msg)
      {
      super(msg);
      }
}
class Test1
{
      public static void main(String[] args)
      {
            try
            {
            int mark=Integer.parseInt(args[0]);
            if(mark>100)
            {
            throw new own("Greater than 100");
            }
            System.out.println("Marks="+mark);
            }
            catch(own e)
            {
            System.out.println("Exception caught");
            System.out.println(e.getMessage());
            }
            finally
            {
            System.out.println("completed");
            }
      }
}
```

O/P: Exception caught
Greater than 100
completed

7.9 Example on exception on handling

EXAMPLE :: 1

```
class ExcDemo0 {
 public static void main(String args[]) {
  int a = 0;
  int b = 42 /a;
 }
}
```

OUTPUT:

```
C:\ >javac ExcDemoO.java
C:\ >java ExcDemoO
Exception in thread "main" java.lang.noclassdeffounderror:ExcDemo
```

EXAMPLE :: 2

```
class ExcDemo1 {
 static void subroutine() {
  int a = 0;
  int b = 10 / a;
 }
 public static void main(String args[]) {
  ExcDemo1.subroutine();
 }
}
```

OUTPUT:

```
C:\vijay>javac ExcDemo1.java
C:\vijay>java ExcDemo1
Exception in thread "main" java.lang.ArithmeticException:/by zero
        at ExcDemo1.subroutine< ExcDemo1.java:4>
        at ExcDemo1.main< ExcDemo1.java:7>
```

EXAMPLE :: 3

```
class Exc2 {
 public static void main(String args[]) {
  int  a,b;

  try { // monitor a block of code.
   d = 0;
   a = 42 / d;
   System.out.println("This will not be printed.");
  } catch (ArithmeticException e) { // catch divide-by-zero error
   System.out.println("Division by zero.");
  }
  System.out.println("After catch statement.");
 }
}
```

OUTPUT:

```
C:\ >javac Exc2.java
C:\ >java Exc2
Division by zero
After each statement
C:\vijay>
```

EXAMPLE :: 4

```
// This program creates a custom exception type.
class MyExc extends Exception {
 private int d;

 MyExc(int a) {
  d = a;
 }

 public String toString() {
  return "MyExc[" + d + "]";
 }
}
```



```
class ExceptionDemo {
 static void compute(int a) throws MyExc {
  System.out.println("Called compute(" + a + ")");
  if(a > 10)
   throw new MyExce(a);
  System.out.println("Normal exit");
 }
 public static void main(String args[]) {
  try {
   compute(1);
   compute(20);
  } catch (MyExc e) {
   System.out.println("Caught " + e);
  }
 }
}
```

OUTPUT:

```
C:\ >javac ExceptionDemo.java
C:\ >java ExceptionDemo
Called compute<1>
Called compute<20>
Normal exit
```

EXAMPLE :: 5

```
// Demonstrate finally.
class FinallyDemo {
 // Through an exception out of the method.
 static void A() {
  try {
   System.out.println("inside A");
   throw new RuntimeException("demo");
  } finally {
   System.out.println("procA's finally");
  }
 }
```

```
// Return from within a try block.
static void B() {
 try {
  System.out.println("inside B");
  return;
 } finally {
  System.out.println("B's finally");
 }
}
// Execute a try block normally.
static void C() {
 try {
  System.out.println("inside C");
 } finally {
  System.out.println("C's finally");
 }
}

public static void main(String args[]) {
 try {
  A();
 } catch (Exception e) {
  System.out.println("Exception caught");
 }
 B();
 C();
}
```

OUTPUT:

```
C:\ >javac FinallyDemo.java
C:\ >java FinallyDemo
Inside A
procA's finally
Exeption caught
Inside B
B's finally
Inside c
C's finally
```


EXAMPLE :: 6

```
// Handle an exception and move on.
import java.util.Random;
class HandleErrorDemo {
 public static void main(String args[]) {
  int a=0, b=0, c=0;
  Random r = new Random();
  for(int i=0; i<32000; i++) {
   try {
    b = r.nextInt();
    c = r.nextInt();
    a = 12345 / (b/c);
   } catch (ArithmeticException e) {
    System.out.println("Division by zero.");
    a = 0; // set a to zero and continue
   }
   System.out.println("a: " + a);
  }
 }
}
```

OUTPUT:

```
C:\ >javac HandleErrorDemo.java
C:\ >java HandleErrorDemo
a:0
a:12345
Division by zero
a:0
Division by zero
a:0
Division by zero
a:0
a:-4115
a:-12345
a:12345
```

EXAMPLE :: 7

```
/* Try statements can be implicitly nested via
   calls to methods. */
class MethNestTry Demo
{
 static void nesttry(int a)
 {
   try { // nested try block

     /* If one command line arg is used,  then an divide-by-zero exception
        will be generated by the following code. */
     if(a==1) a = a/(a-a); // division by zero

     /* If two command line args are used  then generate an out-of-bounds
exception. */
if(a==2) {
      int c[] = { 1 };
      c[42] = 99; // generate an out-of-bounds exception
     }
   } catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("Array index out-of-bounds: " + e);
   }
 }
 public static void main(String args[]) {
  try {
   int a = args.length;
   /* If no command line args are present, the following statement will
generate
      a divide-by-zero exception. */
      int b = 42 / a;
      System.out.println("a = " + a);
   nesttry(a);
  } catch(ArithmeticException e) {
   System.out.println("Divide by 0: " + e);
  }
 }
}
```

OUTPUT:

```
C:\ >javac MethNestTryDemo.java
C:\ >java MethNestTryDemo
```

EXAMPLE :: 8

```
// Demonstrate multiple catch statements.
class MultiCatchDemo {
 public static void main(String args[]) {
  try {
   int x = args.length;
   System.out.println("a = " + x);
   int y = 42 / x;
   int z[] = { 1 };
   z[42] = 99;
  } catch(ArithmeticException e) {
   System.out.println("Divide by 0: " + e);
  } catch(ArrayIndexOutOfBoundsException e) {
   System.out.println("Array index oob: " + e);
  }
  System.out.println("After try/catch blocks.");
 }
}
```

OUTPUT:

```
C:\ >javac MultiCatchDemo.java
C:\ >java MultiCatchDemo
a=0
Divide by 0:java.lang.ArthimeticException:/by zero
After try/catch blocks
```

EXAMPLE :: 9

```
/ An example nested try statements.
class NestTryDemo{
 public static void main(String args[]) {
  try {
   int x = args.length;
```

```
    /* If no command line args are present, the following statement will
  generate
       a divide-by-zero exception. */
  int y = 42 /x;
  System.out.println("x = " +x);
     try { // nested try block
      /* If one command line arg is used, then an divide-by-zero exception
        will be generated by the following code. */
      if(x==1)
  x = x/(x-x); // division by zero

      /* If two command line args are used then generate an out-of-bounds
  exception. */
     if(x==2) {
       int z[] = { 1 };
       z[42] = 99; // generate an out-of-bounds exception
      }
     } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Array index out-of-bounds: " + e);
     }
    } catch(ArithmeticException e) {
     System.out.println("Divide by 0: " + e);
    }
  }
}
```

OUTPUT:

```
C:\ >javac NestTryDemo.java
C:\ >java NestTryDemo
Divide by 0:java.lang.ArithemeticException:/by zero
```

EXAMPLE :: 10

```
/* This program contains an error. A subclass must come before its
superclass in
 a series of catch statements. If not, unreachable code will be created and a
 compile-time error will result.
*/
```

```
class SuperSubCatchDemo {
 public static void main(String args[]) {
  try {
   int x = 0;
   int y = 42 /xa;
  } catch(Exception e) {
   System.out.println("Generic Exception catch.");
  }
  /* This catch is never reached because
    ArithmeticException is a subclass of Exception. */
  catch(ArithmeticException e) { // ERROR - unreachable
   System.out.println("This is never reached.");
  }
 }
}
```

OUTPUT:

```
C:\ >javac SuperSubCatchDemo.java
C:\ >java SuperSubCatchDemo
```

This program contains an error. A subclass must come before its superclass in a series of catch statements. If not, unreachable code will be created and a compile-time error will result.

EXAMPLE :: 11

```
// Demonstrate throw.
class ThrowDemo {
 static void demoproc() {
  try {
   throw new NullPointerException("demo");
  } catch(NullPointerException e) {
   System.out.println("Caught inside demoproc.");
   throw e; // re-throw the exception
  }
 }
 public static void main(String args[]) {
  try {
   demoproc();
  } catch(NullPointerException e) {
```

```
    System.out.println("Recaught: " + e);
   }
  }
}
```

OUTPUT:

```
C:\ >javac ThrowDemo.java
C:\ >java ThrowDemo
Caught inside demoproc
Recaught:java.lang.NullpointerException:Demo
```

EXAMPLE :: 12

```
// This is now correct.
class ThrowsDemo1 {
  static void throwOne() throws IllegalAccessException {
    System.out.println("Inside throwOne.");
    throw new IllegalAccessException("demo");
  }
  public static void main(String args[]) {
    try {
      throwOne();
    } catch (IllegalAccessException e) {
      System.out.println("Caught " + e);
    }
  }
}
```

OUTPUT:

```
C:\ >javac ThrowsDemo1.java
C:\ >java ThrowsDemo1
Inside throwone
Caught java.lang.illlegalAccessException:demo
```