Exception Handling

Introduction

- An exception is a problem that arises during the **execution** of a program.
- Exception disrupts the normal flow of the program
- Abnormal termination of the program is not recommended, therefore these exceptions are to be handled
- Scenarios
  - User has entered invalid data.
  - File that needs to be opened cannot be found.
  - Network connection has been lost in the middle of communications or the JVM has run out of memory.

Advantage of Exception handling

- The core advantage of exception handling is **to maintain the normal flow of the application**.
- Suppose there are 10 statements in the program and there occurs an exception at statement 5, rest of the code will not be executed
- i.e. statement 6 to 10 will not run.
- If exception is handled, rest of the statements will be executed.
- That is why we use exception handling in java.

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exce
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Types of Exceptions

- Exceptions are caused due to
  - user error,
  - programmer error,
  - physical resources that have failed in some manner.
- **Checked exception**
  - Exception that occurs at the compile time
  - Eg; IOException, SQLException etc.
- **Unchecked exception**
  - Occurs at the time of execution, these are also called as Runtime Exceptions
  - Eg; ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.
- **Error**
  - These are not exceptions at all
  - These are problems that arise beyond the control of the user or the programmer.
  - Eg; OutOfMemoryError, VirtualMachineError, AssertionError

Exception Scenarios

1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException

```
int a=50/0;//ArithmeticException
```

2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Exception Keywords

- Java exception handling is managed via five keywords
  - Try
  - Catch
  - Throw
  - Throws
  - finally

General form of Exception handling block

```
try
{
    // block of code to monitor for errors
}
catch (ExceptionType1 exOb)
{
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{
    // exception handler for ExceptionType2
}
finally
{
    // block of code to be executed after try block ends
}
```

Checked Exception - Example

```
public class FileRead {
   public static void main(String[] args)
   {
      try
      {
         File f= new File("input.txt");
         FileInputStream fin=new FileInputStream(f);
      }
      catch(FileNotFoundException e)
      {
         System.out.println("Exception: "+e);
      }
      }
   }
```

```
run:
Exception: java.io.FileNotFoundException: input.txt (The system cannot find the file specified)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Unchecked Exceptions (Runtime Exceptions)- Example

```
public class ExceptionExample
{
   public static void main(String[] args)
    {
     int []arr={1,2,3};
      try
      {
         int d = 0;
         int a = 10 / d;
         arr[5]=9;
      }
      catch(ArrayIndexOutOfBoundsException e)
      {   System.out.println("Exception: "+e);    }

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

     }
   }
```

→ More than one exception raised by single piece of code

→Two or more **catch clauses, each catching a different** type of exception.

→When an exception is thrown, each **catch statement is inspected in order**

→The first one whose type matches that of the exception is executed.

→After one **catch** statement executes, the others are bypassed

→ Execution continues after the **try/catch** block

```
run:
Exception: java.lang.ArithmeticException: / by zero
BUILD SUCCESSFUL (total time: 0 seconds)
```

throw clause - Example

```
public class TestThrow1
{
  static void validate(int age)
 {
   if(age<18)
    throw new ArithmeticException("not valid");
   else
    System.out.println("welcome to vote");
  }
  public static void main(String args[])
  {
    validate(13);
    System.out.println("rest of the code...");
  }
}
```

throw keyword is used to throw an exception explicitly.

Output:

```
Exception in thread main java.lang.ArithmeticException:not valid
```

throws keyword - Example

```
public class FileRead
{
   public static void main(String[] args) throws IOException,
   ArithmeticException
   {
         File f= new File("input.txt");
         FileInputStream fin=new FileInputStream(f);
         int x=5/0;
   }
   }
```

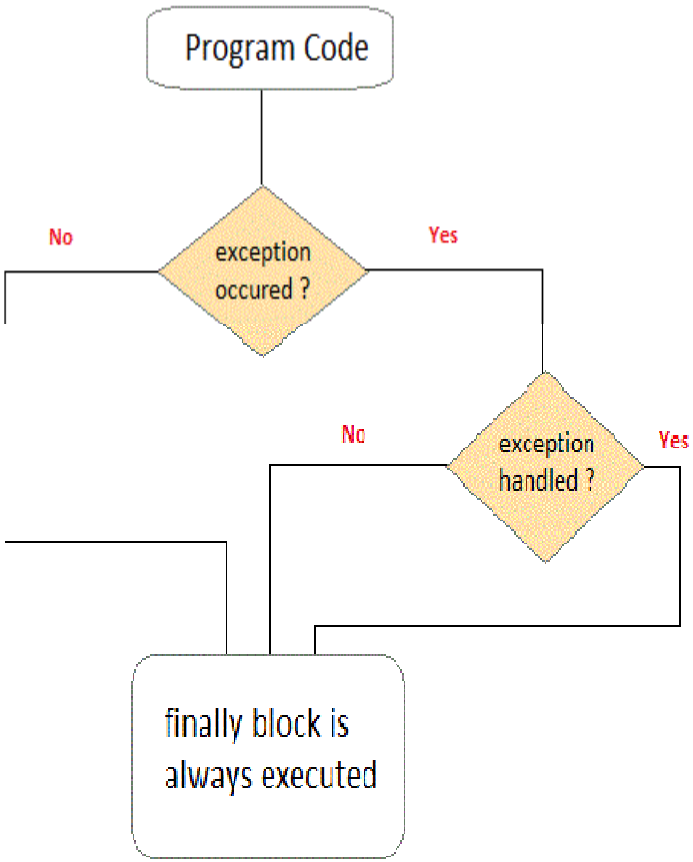
If a method is capable of causing an exception that it does not handle

include a throws clause in the method's declaration

```
run:
Exception in thread "main" java.io.FileNotFoundException: input.txt (The system cannot find the file specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:146)
        at exceptionexample.FileRead.main(FileRead.java:19)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

finally clause

- finally keyword is used to create a block of code that follows a try or catch block.
- finally block of code **always executes whether or not exception has occurred**.
- using a finally block, lets developers to write cleanup type statements



finally clause - example

```
public class ExceptionClass {

 public static void main(String[] args)
 {
  int a[]= new int[2];
  System.out.println("out of try");
  try
  {
   System.out.println("Access invalid element"+ a[3]);
  }
  finally
  {
   System.out.println("finally is always executed.");
  }
 }
}
```

```
run:
out of try
finally is always executed.
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at exceptionexample.ExceptionClass.main(ExceptionClass.java:11)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Built-in unchecked Runtime Exceptions in java.lang

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
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.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Built-in checked Runtime Exceptions in java.lang

Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

How do we create our own exception?

- Can create our own exceptions in Java.
- All exceptions must be a child of Throwable.
- If the exception to be created is checked exception need to extend the Exception class.
- If the exception to be created is a runtime exception, need to extend the RuntimeException class.