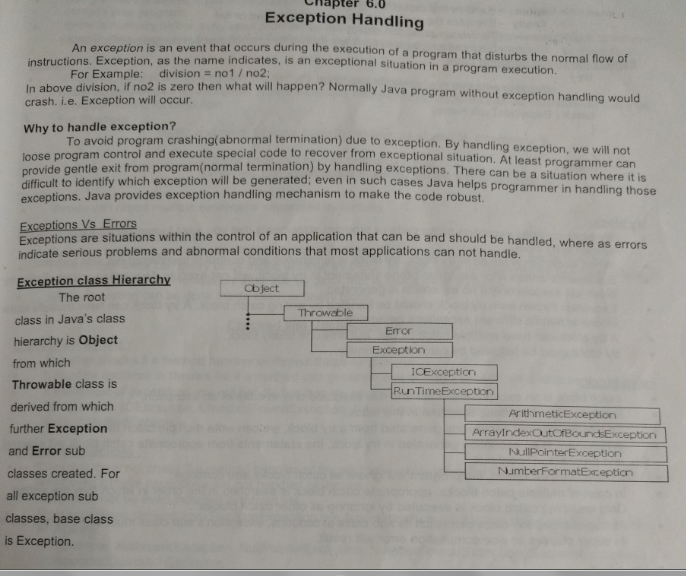
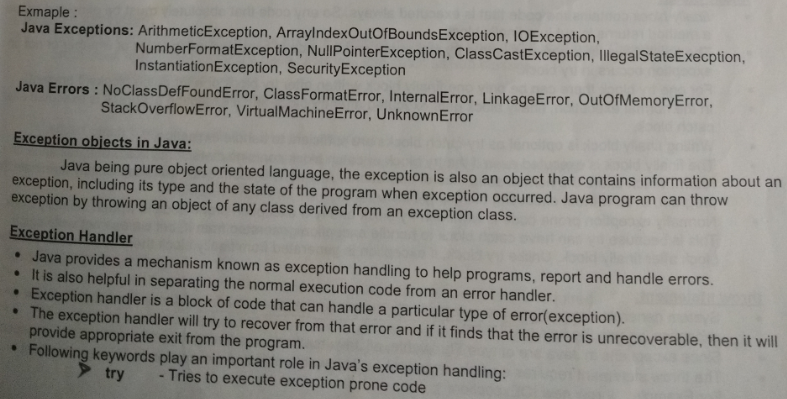
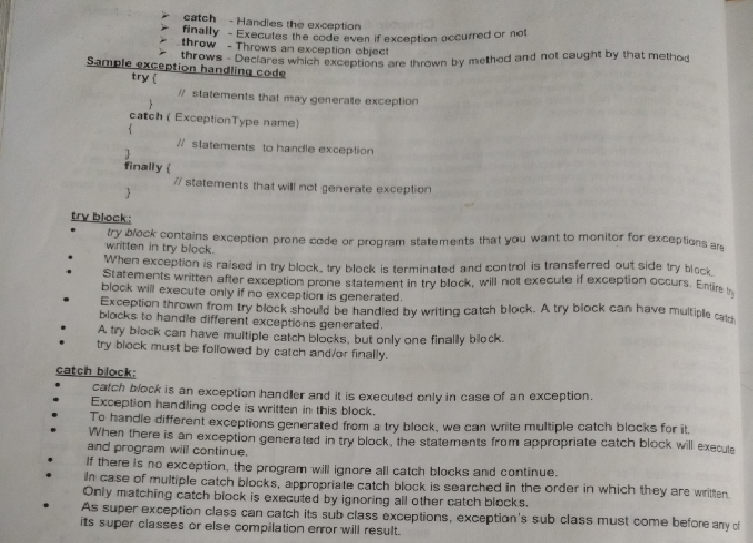
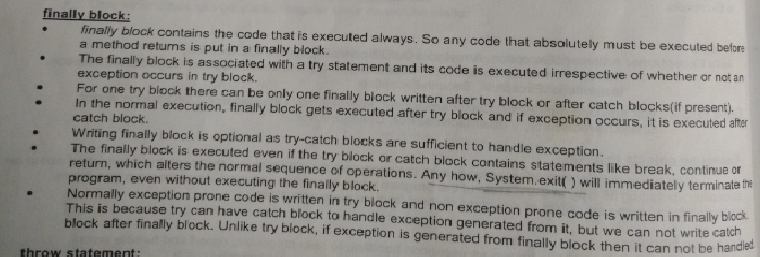
The **exception handling in java** is one of the powerful *mechanisms to handle the* ***runtime*** *exceptions* so that normal flow of the application can be maintained. All exception and error types are sub classes of class **Throwable.**

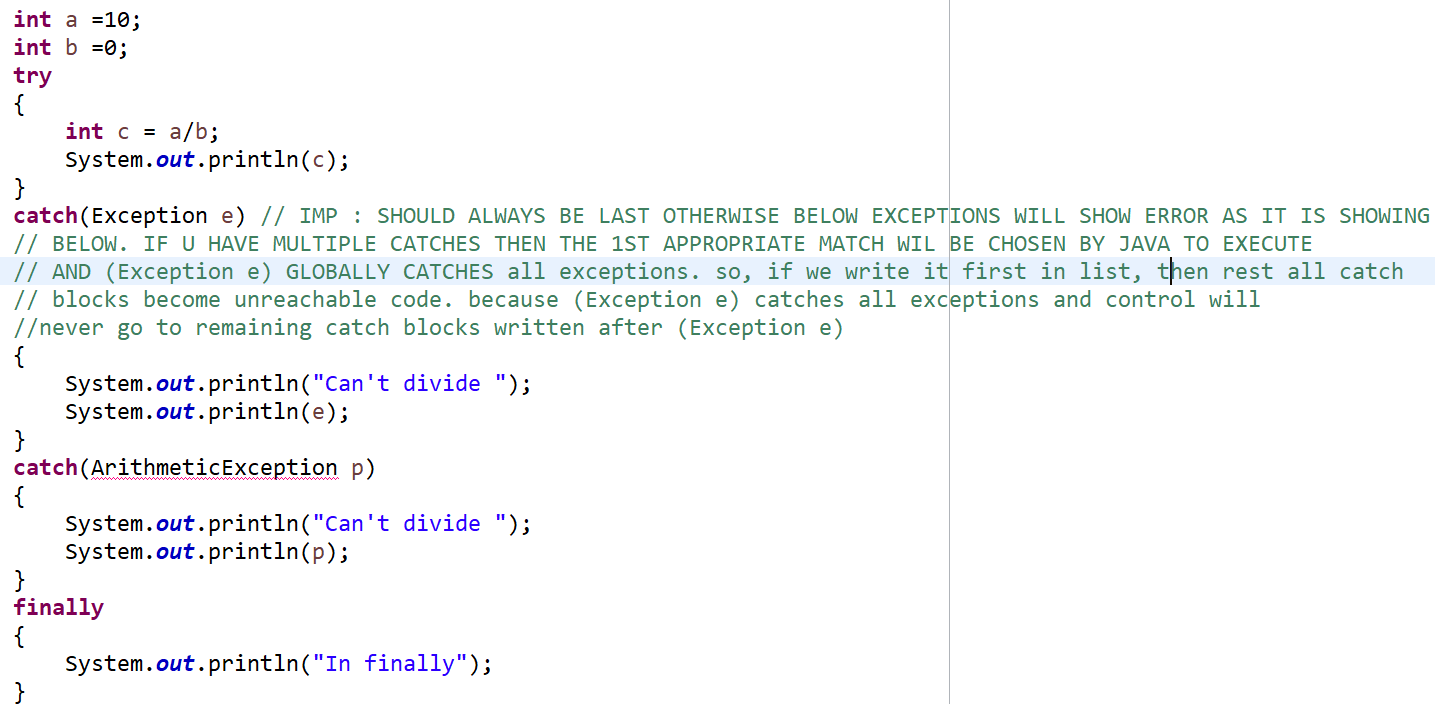




try must have either catch OR finally. One out of these 2 are compulsory. We cant have only finally without try.



IMP :



Throw and Throws :

Throw : The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.

e.g. : class ThrowExcep

{

    static void fun()

    {

        try

        {

            throw **new** NullPointerException("demo");

        }

        catch(NullPointerException e)

        {

            System.out.println("Caught inside fun().");

            throw e; // rethrowing the exception

        }

    }

}

Throws :

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions.

class tst

{

public void abc()throws InterruptedException, ArrayIndexOutOfBoundsException

    {

        System.out.println("Hello all");

    }

}

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The **caller** to this method must handle the exception using a try-catch block.

**User defined exception :**

In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom** exceptions.

class MyException extends Exception{

String str1;

/\* below is Constructor of custom exception class

\* here I am copying the message that we are passing while

\* throwing the exception to a string and then displaying

\* that string along with the message.

\*/

MyException(String str2) {

str1=str2;

}

public String toString(){

return ("MyException Occurred: "+str1) ;

}

}

class Example1{

public static void main(String args[]){

try{

System.out.println("Starting of try block");

// I'm throwing the custom exception using throw

throw new MyException("This is My error Message");

}

catch(MyException exp){

System.out.println("Catch Block") ;

System.out.println(exp) ;

}

}

}

**Output:**

Starting of try block

Catch Block

MyException Occurred: This is My error Message

**Exception and Error** : Exceptions and errors both are subclasses of Throwable class. Error refers to an illegal operation performed by user due to which program work abnormally, Programming errors often remain undetected until the program is compiled or executed. It mostly occurs runtime. We can’t handle it with try catch. It is of 3 types:

* Compile-time
* Run-time
* Logical

Examples of errors :

|  |  |
| --- | --- |
| **InternalError** | Indicating the occurrence of an unexpected internal error in the JVM. |
| **UnknownError** | In case a serious exception that is unknown has occurred in the JVM. |
| **OutOfMemoryError** | In case JVM cannot allocate an object as it is out of memory, such error is thrown that says no more memory could be made available |

**Checked and Un-Checked exceptions :**

**Unchecked exceptions** are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error.

**Checked exceptions** are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using [try-catch block](https://beginnersbook.com/2013/04/try-catch-in-java/) or it should declare the exception using [throws keyword](https://beginnersbook.com/2013/04/java-throws/), otherwise the program will give a compilation error.

**List of Common Checked Exceptions in Java**

Common checked exceptions defined in the java.lang package:

* ReflectiveOperationException
  + ClassNotFoundException
  + InstantiationException
  + IllegalAccessException
  + InvocationTargetException
  + NoSuchFieldException
  + NoSuchMethodException
* CloneNotSupportedException
* InterruptedException

Common checked exceptions defined in the java.io package:

* IOException
  + EOFException
  + FileNotFoundException
  + InterruptedIOException
  + UnsupportedEncodingException
  + UTFDataFormatException
  + ObjectStreamException
* InvalidClassException
* InvalidObjectException
* NotSerializableException
* StreamCorruptedException
* WriteAbortedException

Common checked exceptions defined in the java.net package (almost are subtypes of IOException):

* SocketException
  + BindException
  + ConnectException
* HttpRetryException
* MalformedURLException
* ProtocolException
* UnknownHostException
* UnknownServiceException

Common checked exceptions defined in the java.sql package:

* SQLException
  + BatchUpdateException
  + SQLClientInfoException
  + SQLNonTransientException
* SQLDataException
* SQLFeatureNotSupportedException
* SQLIntegrityConstraintViolationException
* SQLSyntaxErrorException
  + SQLTransientException
* SQLTimeoutException
* SQLTransactionRollbackException
* SQLTransientConnectionException
  + SQLRecoverableException
  + SQLWarning

**4. List of Common Unchecked Exceptions in Java**

Common unchecked exceptions in the java.lang package:

* ArithmeticException
* IndexOutOfBoundsException
  + ArrayIndexOutOfBoundsException
  + StringIndexOutOfBoundsException
* ArrayStoreException
* ClassCastException
* EnumConstantNotPresentException
* IllegalArgumentException
  + IllegalThreadStateException
  + NumberFormatException
* IllegalMonitorStateException
* IllegalStateException
* NegativeArraySizeException
* NullPointerException
* SecurityException
* TypeNotPresentException
* UnsupportedOperationException

Common unchecked exceptions in the java.util package:

* ConcurrentModificationException
* EmptyStackException
* NoSuchElementException
  + InputMismatchException
* MissingResourceException

 **If the superclass method does not declare an exception**

* If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.

 **If the superclass method declares an exception**

* If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.