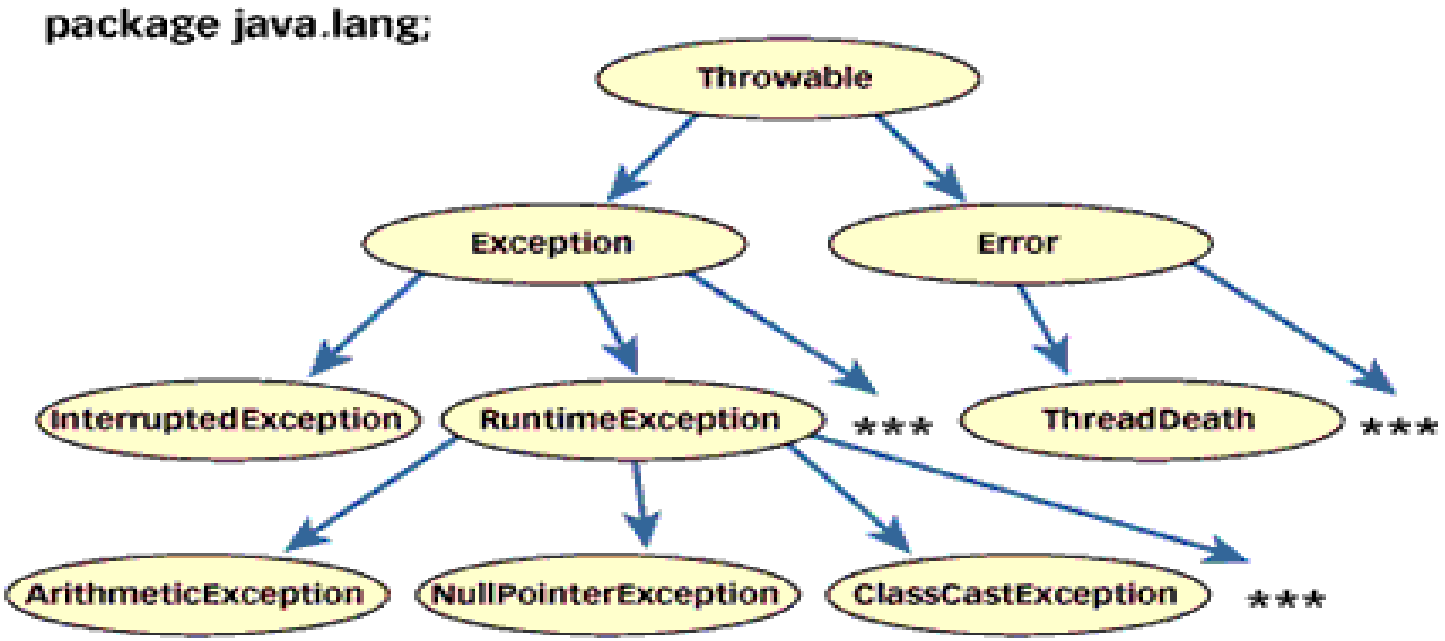
EXCEPTIONS IN JAVA

What's Exception

- *An exception is an abnormal condition that occurs at run time. For example divide by 0.*
- *During execution of a statement within any method if any exceptional condition occurs the Java Runtime Environment (JRE) i.e. java interpreter creates a suitable Exception object and throws it.*
- *Every Exception is basically an object belonging to Java's Exception class Hierarchy.*
- *Exceptions needs to be handled so that appropriate actions can be taken.*
- *Programmer can also provide exception handling code. However if there is no exception handling code present during runtime and exception occurs, then java interpreter provides default exception handler.*
- *Default Exception Handler displays the name of the exception object in string form and stops the execution of the program.*
- *However , programmer can provide exception handling code and program's execution can continue even after the occurrence of exception.*

Exception class Hierarchy

- *Every Exception type is basically an object belonging to class **Exception***
- ***Throwable** class is the root class of Exceptions.*
- ***Throwable** class has two direct subclasses named Exception, Error*



Types of Exceptions

A. Unchecked Exceptions

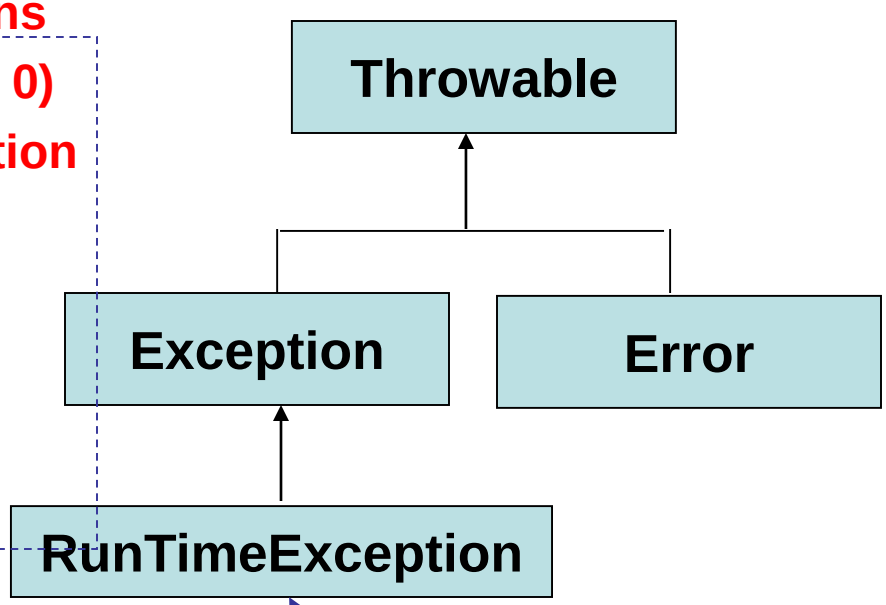
All Exceptions that extend the RuntimeException or any one of its subclass are unchecked exceptions

- ***Unchecked Exceptions are unchecked by compiler.***
- ***Whether you catch the exception or not compiler will pass the compilation process.***
- ***If Unchecked exception is caught then exception handling code will be executed and program's execution continues.***
- ***If Unchecked exception is not caught then java interpreter will provide the default handler. But in this case execution of the program will be stopped by displaying the name of the exceptions object.***

Unchecked Exceptions

Some Common Unchecked Exceptions

1. ArithmaticException (Divide By 0)
2. ArrayIndexOutOfBoundsException
3. ArrayStoreException
4. FileNotFoundException
5. NullPointerException
6. NumberFormatException
7. IllegalArumentsException



All Unchecked Exceptions directly or indirectly are sub classes of RunTimeException

Any Class belonging to RunTimeException

UncheckedExceptions Example

```
class Exceptiondemo1
{
public static void main(String arhs[])
{
int a=10;
int b= 5;
int c =5;
int x = a/(b-c); // Dynamic Initilization
System.out.println("c="+c);
int y = a/(b+c);
System.out.println("y="+y);
}
}
```

throws ArithmeticException

No Need to mention for Unchecked Exceptions

Can Throw an Exception

D:\java\bin>javac Exceptiondemo1.java << Compilation Step Pass>>

```
D:\java\bin>java Exceptiondemo1
Exception in thread "main"
java.lang.ArithmeticException: / by zero
        at Exceptiondemo1.main(Exceptiondemo1.java:8)
```

Example 2 (Unchecked Exceptions)

```
class Exceptiondemo2
{
public static void main(String args[])
{
double a= Double.parseDouble(args[0]);
}
}
```

Can throw either ArrayIndexOutOfBoundsException OR NumberFormatException

```
D:\java\bin>javac Exceptiondemo2.java
D:\java\bin>java Exceptiondemo2
Exception in thread "main"  java.lang.ArrayIndexOutOfBoundsException: 0
        at Exceptiondemo2.main(Exceptiondemo2.java:5)

D:\java\bin>java Exceptiondemo2 pankaj
Exception in thread "main" java.lang.NumberFormatException: For input
string: "pankaj“     at
sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1
2 24)   at java.lang.Double.parseDouble(Double.java:482)
        at Exceptiondemo2.main(Exceptiondemo2.java:5)
```

Put the Related/Dependent Statements in try block

```
class extest
{
public static void main(String args[])
{
try
{
int a = Integer.parseInt(args[0]);
}
catch(Exception e) {}
int b = a+10;
System.out.println("b="+b);
}
}
```

```
E:\oop>javac extest.java
extest.java:10: cannot find
symbol
symbol  : variable a
location: class extest
int b = a+10;
        ^
extest.java:10: incompatible types
found   : <nulltype>
required: int
int b = a+10;
        ^
2 errors
```

Cont…

```
class extest
{
public static void main(String args[])
{
try
{
int a =
Integer.parseInt(args[0]);
int b = a+10;
System.out.println("b="+b);
}
catch(Exception e) {}
}
}
```

Types of Exceptions

B Checked Exceptions

All Exceptions that extends the Exception or any one its subclass except RunTimeException class are checked exceptions

- **Checked Exceptions are checked by the Java compiler.**
- **There are two approaches that a user can follow to deal with checked exceptions**

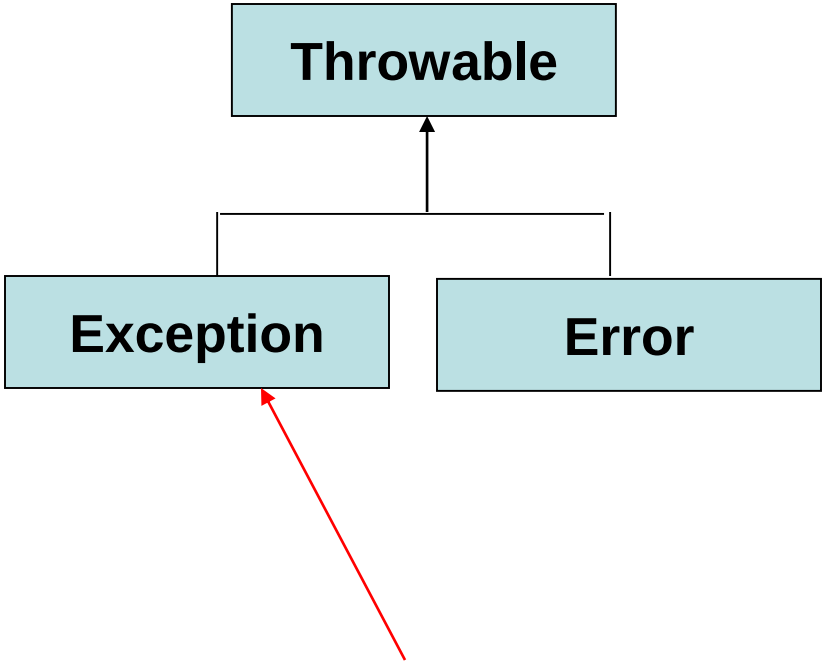
Handling Checked Exceptions

- **Inform the compiler that a method can throw an Exception.**
- **Catch the checked exception in try catch block**
- **If Checked exception is caught then exception handling code will be executed and program's execution continues.**
- **If Checked exception is not caught then java interpreter will provide the default handler. But in this case execution of the program will be stopped by displaying the name of the exceptions object.**

Checked Exceptions Examples

Some Common Checked Exceptions

1. **IOException**
2. **ClassNotFoundExceptions**
3. **InterruptedException**
4. **NoSuchMethodException**



Any Sub Class belonging to Exception

EXCEPT

RuntimeException

Checked Exceptions

```
/* Program to read two integers Display their sum */
import java.io.*;
class Exceptiondemo3
{
public static void main(String args[])
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int a = Integer.parseInt(br.readLine());

int b = Integer.parseInt(br.readLine());
System.out.println("Sum is :"+(a+b));
}
}
```

WILL THIS CODE COMPILE SUCCESSFULLY

```
Exceptiondemo3.java:9: unreported exception java.io.IOException; must be
caught or declared to be thrown
int a = Integer.parseInt(br.readLine());                    ^
Exceptiondemo3.java:10: unreported exception java.io.IOException; must be
caugh  or declared to be thrown
int b = Integer.parseInt(br.readLine());                    ^
```

Ways To Handle Checked Exceptions

```
Method 1:  << Mention thru throws clause>>
import java.io.*;
class Exceptiondemo3
{
public static void main(String args[]) throws IOException
{
BufferedReader br = new BufferedReader(new inputStreamReader(System.in));
int a = Integer.parseInt(br.readLine());
int b = Integer.parseInt(br.readLine());
System.out.println("Sum is :"+(a+b));
}
}
```

1. ***throws*** ***clause is used with methods to indicate type of Exception a method can throw***
2. ***Specifically required for Checked Exceptions [To Pass Compilation process]. It can/may be used for unchecked exceptions also.***
3. ***A method can throw as many exceptions.***

Ways To Handle Checked Exceptions cont….

```
Method 2 << Put the statements in try catch block and catch >>
import java.io.*;
class Exceptiondemo3
{
public static void main(String args[])
{
BufferedReader br = new BufferedReader(new inputStreamReader(System.in));

try {
int a = Integer.parseInt(br.readLine());
int b = Integer.parseInt(br.readLine());
System.out.println("Sum is :"+(a+b));
}
catch(IOException e) { }
}
}
```

Exception Handling

Exception Handling Requires the Following four steps

1. **Finding the problem (Identify the statements whose execution may result in Exception. Put all those statements in a *try{..}* block)**
2. **Inform that an exception is thrown (Throw the Exception) <u>*<< Note Down throw vs throws>>*</u>**
3. **Receive the exception (Catch the exception using *catch{..}* block)**
4. **Provide exception handling code in catch block.**

Exception Hadling Syntax

```
try
{
<statements that can throw exceptions>
}
catch(ExceptionType<1> e1) {….}
catch(ExceptionType<2> e2) {….}
catch(ExceptionType<3> e3) {….}
……..
catch(ExceptionType<N> e4) {….}
```

Important Points :

1. **try {} block may have one or multiple statements.**
2. **try{} block may throw a single type of Exception or multiple exceptions. But at a time it can throw only single type of exception.**
3. **There can be multiple catch() { .. } blocks associated with single try{} block.**
4. **If try{} block can throw multiple exceptions then user should catch all exceptions. (one catch block for each type of exception)**

Catching an Exception

```
class Exceptiondemo1
{
public static void main(String arhs[])
{
int a=10;
int b= 5;
int c =5;
try
{
int x = a/(b-c);
System.out.println("c="+c);
}
catch(ArithmeticException e)
{
System.out.println(e.toString());
}
int y = a/(b+c);
System.out.println("y="+y);
}
}
```

D:\java\bin>java Exceptiondemo1
java.lang.ArithmeticException: / by zero
y=1

Catching Multiple Exceptions

```
class Exceptiondemo4
{
public static void main(String args[])
{
int a[]= {5,10};
try
{
int b= Integer.parseInt(args[0]);
int x = a[b]/(b-a[1]);
System.out.println("x="+x);
}
```

```
catch(ArithmeticException e)
{
System.out.println(e.toString());
}
catch(NumberFormatException e)
{
System.out.println(e.toString());
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println(e.toString());
}
```

This Statement is outside catch block and will be executed in any case

```
System.out.println("Hello This is Exception Test");
} // End of main() method
}// End of class Exceptiondemo4
```

OUTPUT

What will be o/p if you execute it like

1. **Java Exceptiondemo4** ← **NO COMMAND LINE ARGUMENTS**
2. **java Exceptiondemo4 1** ← **COMMAND LINE ARGUMENTS PASSED AS 1**
3. **3. java exceptiondemo4 oop** ← **COMMAND LINE ARGUMENT PASSED oop**

Nested Try Statements

- **Try{ } statements can be nested. One try block may contain another try block**
- **In case of nested try blocks, context of that exception is pushed onto stack.**
- **Inner try block may/or may not have catch statements associated with it.**
- **If an exception is thrown from inner try block then first inner catch statements are matched (if present) . If no match is found then outer try block are matched. If there also no match found then default handler will be invoked.**
- **However, if outer try block throws the exception then only outer try blocks are matched.**

Nested try blocks

A typical Syntax

```
try
{
Statement  A;
Statement B;
      try
      {
       Statement  C;
       Statement  D;
      }
     catch(CException e) {  ....  }
     catch(DException e) {  ....  }
}
catch(AException e) {  ....  }
catch(BException e) {  ....  }
```

```
try
{
Statement  A;
Statement B;
      try
      {
       Statement  C;
       Statement  D;
      }
}
catch(AException e) {  ....  }
catch(BException e) {  ....  }
catch(CException e) {  ....  }
catch(DException e) {  ....  }
```

Nested try blocks cont…

```
try
{
Statement  A;
Statement B;
        try
        {
         Statement  C;
         Statement  D;
        }
       catch(CException e) {  ….  }
       catch(DException e) {  ….  }
}
catch(AException e) {  ….  }
catch(BException e) {  ….  }
catch(CException e) {  ….  }
catch(DException e) {  ….  }
```

Nested try statements Example

```
class nestedtry
{
public static void main(String args[])
{
int a[] = { 2,5,6};    // { a[0] = 2, a[1] = 5, a[2] = 6}
try  // outer try
{
   int b = Integer.parseInt(args[0]);
    try  // inner try
    {
     int c[] = { 4,5,6};          // { c[0] = 4, c[1] = 5, c[2] = 6}
     int d  = c[b]/(c[b]-4);
    } // End of inner try
    catch(ArrayIndexOutOfBoundsException e)
    {
    System.out.println("Exception : "+ e.toString());
    System.out.println("By Inner try");
    }
    catch(ArithmeticException e)
    {
    System.out.println("Exception : "+ e.toString());
    System.out.println("By Inner try");
    }
 } // End of outer try
```

```
// Catch Blocks for outer try
catch(ArrayIndexOutOfBoundsException e)
    {
    System.out.println("Exception : "+ e.toString());
    System.out.println("By Outr try");
    }
catch(NumberFormatException e)
    {
    System.out.println("Exception : "+ e.toString());
    System.out.println("By Outer try");
    }
} // End of main
} // End of class
```

```
D:\java\bin>java nestedtry
Exception : java.lang.ArrayIndexOutOfBoundsException: 0
By Outer try

D:\java\bin>java nestedtry 4
Exception : java.lang.ArrayIndexOutOfBoundsException: 4
By Inner try

D:\java\bin>java nestedtry 0
Exception : java.lang.ArithmeticException: / by zero
By Inner try
```

Writing Your Own Exceptions

- **Programmers Can write their own Exception classes apart from java's library Exceptions.**
- **Programmer can write either checked Exception OR Unchecked Exception.**
- **To make a checked exception , make your exception class a subclass of Exception OR any one of its subclass EXCEPT RunTimeException.**

```
class AException extends Exception { …}      → Checked Exception
class BException extends IOException { ..} → Checked Exception
```

- **To make a Unchecked exception , make your exception class a subclass of RunTimeException OR any one of its subclass .**

```
class XException extends RunTimeException { … }
class YException extends AritmeticException { … }
class ZException extends ArrayIndexOutOfException { … }
class ZException extends IndexOutOfBoundsException { … }
```

Throwing Unchecked Exception

1. **Create an InvalidBOXException which will be thrown by the constructor of the BOX class whenever an attempt will be made to create an invalid BOX object. (Any Dimension = 0 or < 0).**
2. **Create an InvalidTriangleException which will be thrown whenever an attempt will be made to create an invalid Triangle object. (In Triangle sum of two sides must be > third side).**

EXAMPLE 1:

```
class InvalidBOXException extends RuntimeException
{
InvalidBOXException(String msg)
{
super(msg);
System.out.println("An attempt is made to create an Invalid BOx object ");
}
}
class BOX
{
private double length;
private double width;
private double height;
BOX(double l, double w, double h) throws InvalidBOXException
{
if( l <=0 || w <= 0 || h <= 0)
throw new InvalidBOXException("Invalid BOX Object creation");
length = l;
width = w;
height = h;
}
```

Optional as InvalidBOXException is Unchecked

```
   double getLength() { return length; }
   double getWidth()  { return width;  }
   double getHeight() { return height; }

   double Area()   { return 2*(length*width + width*height + height*length); }
   double Volume() { return length*width*height ; }
   }
class exceptiontest1
{
public static void  main(String args[])
{
BOX b1 = new BOX(0,0,0);
BOX b2 = new BOX(10,4,5);
System.out.println("Area of b2:"+b2.Area());
}
}
```

```
D:\java\bin>java exceptiontest1
An attempt is made to create an Invalid BOx object
Exception in thread "main" InvalidBOXException: Inavlid BOX Object
creation
        at BOX.<init>(exceptiontest1.java:18)
        at exceptiontest1.main(exceptiontest1.java:35)
```

Change of main method No 1

```
class exceptiontest1
{
public static void  main(String args[])
{
try {
BOX b1 = new BOX(0,0,0);
System.out.println("Area of b1"+b1.Area());
}
// catch(InvalidBOXException e) {   }
catch(Exception e) { };
try {
BOX b2 = new BOX(10,4,5);
System.out.println("Area of b2:"+b2.Area());
}
// catch(InvalidBOXException e) {    }
catch(Exception e) {};
}}
```

If these statements are out side try block?

```
D:\java\bin>java exceptiontest1
An attempt is made to create an Invalid BOx object
Area of b2:220.0
```

Change of Main Method No 2

```
class exceptiontest1
{
public static void  main(String args[])
{
BOX b1;
System.out.println(b1.Area());
}
}
```

<Compile Time Error>

```
D:\java\bin>javac exceptiontest1.java
exceptiontest1.java:36: variable b1
might not have been initialized
System.out.println(b1.Area());
                   ^
1 error
```

```
class exceptiontest1
{
public static void  main(String args[])
{
BOX b1 = null;
System.out.println(b1.Area());

}
}
```

<RUNTIME Error>

```
D:\java\bin>java exceptiontest1
Exception in thread "main"
java.lang.NullPointerException
        at
exceptiontest1.main(exceptiontest1.java:
36)
```

Checked Exceptions

- **Make your exception class extends Exception class or any one of its subclass except RumtimeException.**
- **Checked Exceptions needs to either caught or informed by use of *throws* clause**
- **Note down that *throw* clause is used to throw the exception where as *throws* clause is used to inform that an exception is thrown by the method.**
- **Throw clause is used inside method body where as throws clause is used with first line of the method.**
- **Throws clause can be used to inform both type of exceptions. But in case a method is throwing a unchecked exception then it is not compulsory to inform.**
- **In case a method is throwing a checked Exception, then it has either to caught the exception or informs by using throws clause or it can do both.**

EXAMPLE 1:

```
class InvalidBOXException extends Exception
{
InvalidBOXException(String msg)
{
super(msg);
System.out.println("An attempt is made to create an Invalid BOx object ");
}
}
class BOX
{
private double length;
private double width;
private double height;
BOX(double l, double w, double h)
{
if( l <=0 || w <= 0 || h <= 0)
throw new InvalidBOXException("Inavlid BOX Object creation");
length = l;
width = w;
height = h;
}
```

Checked Exception

Any Method or constructor which throws an checked Type Exception must inform it thru throws clause

```
double getLength() { return length; }
double getWidth()  { return width;  }
double getHeight() { return height; }

double Area()   { return 2*(length*width + width*height + height*length); }
double Volume() { return length*width*height ; }
}
class exceptiontest1
{
public static void  main(String args[])
{
BOX b1 = new BOX(0,0,0);
BOX b2 = new BOX(10,4,5);
System.out.println("Area of b2:"+b2.Area());
}
}
```

```
D:\java\bin>javac exceptiontest1.java  < Compile Time Error>
exceptiontest1.java:18: unreported exception InvalidBOXException; must be
caught  or declared to be thrown throw new InvalidBOXException("Inavlid
BOX Object creation");
^
1 error
```

EXAMPLE 1:

```
class InvalidBOXException extends Exception
{
InvalidBOXException(String msg)
{
super(msg);
System.out.println("An attempt is made to create an Invalid BOx object ");
}
}
class BOX
{
private double length;
private double width;
private double height;
BOX(double l, double w, double h) throws InvalidBOXException
{
if( l <=0 || w <= 0 || h <= 0)
throw new InvalidBOXException("Inavlid BOX Object creation");
length = l;
width = w;
height = h;
}
```

```
    double getLength() { return length; }
    double getWidth()  { return width;  }
    double getHeight() { return height; }

    double Area()   { return 2*(length*width + width*height + height*length); }
    double Volume() { return length*width*height ; }
    }
class exceptiontest1
{
public static void  main(String args[]) throws InvalidBOXException
{
BOX b1 = new BOX(0,0,0);
BOX b2 = new BOX(10,4,5);
System.out.println("Area of b2:"+b2.Area());
}
}
```

```
D:\java\bin>java exceptiontest1
An attempt is made to create an Invalid BOx object
Exception in thread "main" InvalidBOXException: Inavlid BOX Object
creation
        at BOX.<init>(exceptiontest1.java:18)
        at exceptiontest1.main(exceptiontest1.java:36)
```

Use of finally Clause

- finally statement can be used to handle an exception that is not caught by previous statements.
- finally block may be added immediately after try block or after the last catch block.
- finally block in general used to perform house keeping operations such as closing files or releasing system resources.
- Finally block when present is guaranteed to execute regardless of whether an exception is thrown or not.
- If you want then finally block can be used to handle any exception generated within a try block.

finally clause Syntax

```
try
{
 ……….
……….
……….
}
finally
{
 ……..
……..
…….
}
```

```
try
{
 ……..
……..
……..
}
catch(…….)
{  ……. }
catch(……..)
{  …….  }
…..
…..
finally
{
    ……..
    ……..
}
```

Example(finally clause)

```
class ex10
{
public static void main(String args[])
{
int a=10;
int b = 20;
try
{
 int b1=Integer.parseInt(args[0]);
 int x = a/(a-b1);
     try
     {
     int y = b/(b-b1);
     }
     finally
     {
     System.out.println("Inner Block executed");
     }
}
```

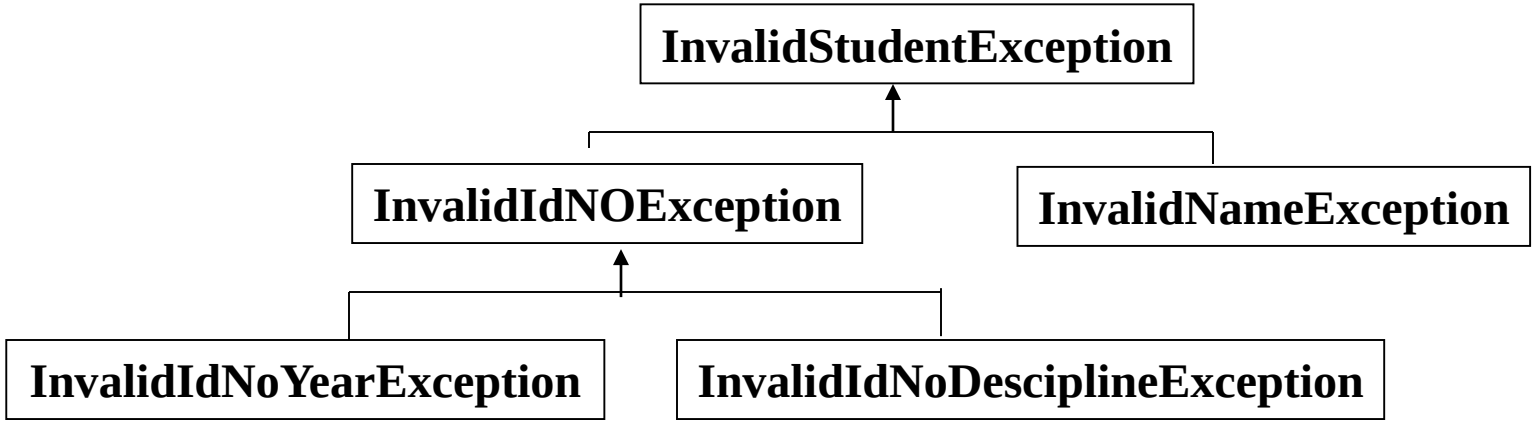
```
finally
{
    System.out.println("Outer Block executed");
}
}
}
```

Output

```
D:\java\bin>java ex10
Outer Block executed
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
0        at ex10.main(ex10.java:9)

D:\java\bin>java ex10 45
Inner Block executed
Outer Block executed

D:\java\bin>java ex10 10
Outer Block executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ex10.main(ex10.java:10)

D:\java\bin>java ex10 20
Inner Block executed
Outer Block executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ex10.main(ex10.java:13)
```

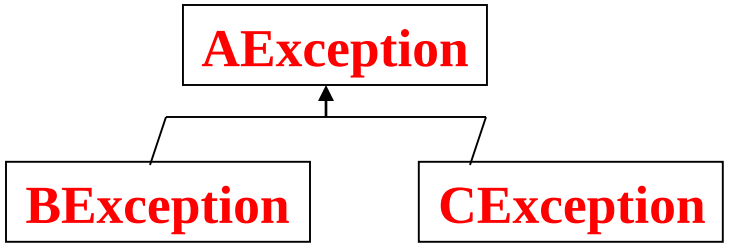
Creating Hierarchy of Exceptions

1. **We can create our own tree of exception classes.**
2. **All Exceptions classes in tree are either checked or unchecked depending upon whether the super class is checked or unchecked.**



Example

```
class AException extends RuntimeException{}
class BException extends AException{}
class CException extends AException{}

class ex11
{
public static void main(String args[])
{
try
{
int a=10;
}
catch(AException e) {}
catch(BException e) {}
catch(CException e) {}
}
}
```

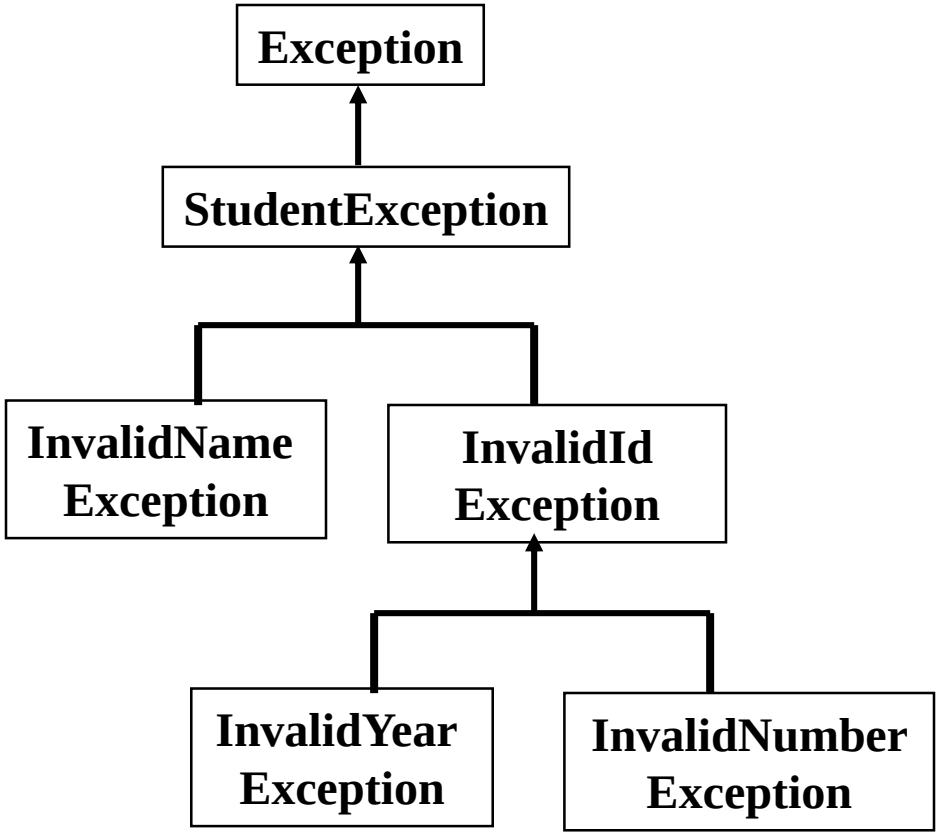


Catch sub class Exceptions First then super class Exceptions

```
D:\java\bin>javac ex11.java
ex11.java:14: exception BException has already been caught
catch(BException e) {}
^
ex11.java:15: exception CException has already been caught
catch(CException e) {}
^
2 errors
```

Exception Example

```
class Student
{
private String name;
private String idno;
```



```
class StudentException extends Exception
{
StudentException(String msg)
{
super(msg);
System.out.println(msg);
}
}
class InvalidNameException extends StudentException
{
InvalidNameException(String msg)
{
super(msg);
System.out.println(msg);
}
}
class InvalidIdException extends StudentException
{
InvalidIdException(String msg)
{
super(msg);
System.out.println(msg);
}}
```

```
class InvalidYearException extends StudentException
{
InvalidYearException(String msg)
{
super(msg);
System.out.println(msg);
}
}
class InvalidNumberException extends StudentException
{
InvalidNumberException(String msg)
{
super(msg);
System.out.println(msg);
}
}
```

```
class Student
{
private String name;
private String idno;
private boolean containsAlphabetsOnly(String str)
{
        for(int i=0;i<str.length();i++)
        {
        int j = str.charAt(i);
        if(j < 65) return false;
        if(j > 125) return false;
        if(j > 91 && j < 96) return false;
         }
        return true;
}
```

```
Student(String name,String idno) throws StudentException
{
if(!containsAlphabetsOnly(name))
throw new InvalidNameException("Invalid Name");

int a = Integer.parseInt(idno.substring(0,4));
if( a < 1995 || a > 2007)
throw new InvalidYearException("Invalid Id Year");

int b = Integer.parseInt(idno.substring(8));
if( b <= 0 || b > 999)
throw new InvalidNumberException("Invalid Student Number");

this.name = name;
this.idno = idno;
}
}// End of student class
```

```
class exceptiontest
{
public static void main(String args[]) throws StudentException
{
Student std1 = new Student("123","sttts");
}
}
```

```
E:\oop>java studentexception
Exception in thread "main" java.lang.NoClassDefFoundError:
studentexception (wrong name: StudentException)at
java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(Unknown Source)
        at java.security.SecureClassLoader.defineClass(Unknown
Source)at java.net.URLClassLoader.defineClass(Unknown Source)
```

Sample Example 1

```
class Sample
{
public static void main(String args[])
{
        try
        {
                int a = 10;
        }
        catch(Exception e) {}
} // End of main()
}//End of class Sample
```

NO ERROR

Example 2

```
import java.io.*;
class Sample
{
public static void main(String args[])
{
        try
        {
                int a = 10;
        }
        catch(IOException e) {}
}// End of main()
}//End of class sample
```

```
Sample.java:10: exception
java.io.IOException is never thrown in body
of corresponding try statement
catch(IOException e) {}
^
1 error
```

Example 3

```
import java.io.*;
class Sample
{
public static void main(String args[])
{
        try
        {
                int a = 10;
        }
        catch(Exception e) {}
        catch(RuntimeException e) {}
}
}//End of class
```

```
D:\java\bin>javac Sample.java
Sample.java:11: exception
java.lang.RuntimeException has already
been caught
catch(RuntimeException e) {}
^
1 error
```

Example 4

```
class ExceptionDemo4
{
public static void main(String args[]) throws Exception
{
int a= 10;
int b = a + 10;
System.out.println("a="+a+"b="+b);
}
}
```

```
E:\Java Programs>javac ExceptionDemo4.java
E:\Java Programs>java ExceptionDemo4
a=10b=20
```

Example 5

```
class ExceptionDemo4
{
public static void main(String args[]) throws RuntimeException
{
int a= 10;
int b = a + 10;
System.out.println("a="+a+"b="+b);
}
}
```

```
E:\Java Programs>javac ExceptionDemo4.java
E:\Java Programs>java ExceptionDemo4
a=10b=20
```

Example 6

```
import java.io.*;
class ExceptionDemo4
{
public static void main(String args[]) throws IOException
{
int a= 10;
int b = a + 10;
System.out.println("a="+a+"b="+b);
}
}
```

```
E:\Java Programs>javac ExceptionDemo4.java
E:\Java Programs>java ExceptionDemo4
a=10b=20
```

Example 7

```
class A
{
public void display() throws Exception
{
System.out.println("Hello");
}// End of display()
}// End of class A

class B extends A
{
public void display() throws RuntimeException
{
System.out.println("Hi");
}// End of display()
}// End of class B
```

display() method is overridden in sub class B.

A's display throws Exception

B's display throws RuntimeException

NO ERROR IN CODE.

COMPILES SUCESSFULLY

Example 8

```
class A
{
public void display() throws RuntimeException
{
System.out.println("Hello");
}// End of display()
}// End of class A

class B extends A
{
public void display() throws Exception
{
System.out.println("Hi");
}// End of display()
}// End of class B
```

display() method is overridden in sub class B.

A's display throws RuntimeException

B's display throws Exception

```
E:\Java Programs>javac AB.java
AB.java:10: display() in B cannot
override display() in A; overridden
method does not throw
java.lang.Exception
public void display() throws
Exception
          ^
1 error
```

Example 9

```
import java.io.*;
class A
{
public void display() throws RuntimeException
{
System.out.println("Hello");
}// End of display()
}// End of class A

class B extends A
{
public void display() throws IOException
{
System.out.println("Hi");
}// End of display()
}// End of class B
```

display() method is overridden in sub class B

```
E:\Java Programs>javac AB.java
AB.java:10: display() in B cannot override
display() in A; overridden method does not
throw java.io.IOException
public void display() throws IOException
```

Example 10

```
import java.io.*;
class A
{
public void display() throws IOException
{
System.out.println("Hello");
}// End of display()
}// End of class A

class B extends A
{
public void display() throws RuntimeException
{
System.out.println("Hi");
}// End of display()
}// End of class B
```

display() method is overridden in sub class B

NO ERROR IN CODE. COMPILES SUCESSFULLY