Exceptions

Out of Bounds

Java checks the bounds of arrays.

```
class  ArrayOutOfBounds   {
  public static void main( String args[] )  {
        int  students[] = new int[5];
        students[  10  ]  =  1;
        System.out.println( " The End" ); }
}
```

Output

```
java.lang.ArrayIndexOutOfBoundsException: 10
at ArrayOutofBounds.main(All.java:8)
```

Exception Handling

There are events which may be considered exceptional or unusual

- Error conditions: On zero divide have the program do some recovery process or give some debugging information.
- Special conditions: If a plane is to collide with another the navigation program should perform some special operation.

Detecting exceptions

```
Boolean foo( ) {
        Boolean  zeroDivideFlag = false;
        if  ( A * A - 2 * B = 0 )
             zeroDivideFlag = true;
        else result = sqrt( C )/ ( A * A - 2 * B );
        return zeroDivideFlag;
 }

     // in main
flag  =  foo( );
if ( flag == true )
     // now what?
```

Definitions

Exception

- A condition that is detected by an operation, that cannot be resolved within the local context of the operation, and must be brought to the attention of the operation's invoker.

Exception handler or handler:

- procedure or code that deals with the exception.

Raising the exception:

- The process of detecting the exception, interrupting the program and calling the exception handler.

Propagating the exception: Reporting the problem to a higher authority (the calling program) .

Handling the exception: Taking appropriate corrective actions.

Issues

- How is an exception raised?
- Where can the Handler be placed - Scope of handler?
- How is the handler for an exception found?
  - All languages find the handler dynamically
- What is executed after the handler has finished?
- What information is passed to the handler?

How is the exception raised?

- Implicitly by some error condition
- Explicitly by the programmer

Example - Raising an exception

```
class  ImplicitlyRaisedException  {
   public static void main( String[] arguments )
   {
        int  students[] = new int[5];
        students[  10  ]  =  1;
                          // Exception occurs here
   }
}
```

Example - Raising an exception

```
class  ExplicitlyRaisedException   {
  public static void main( String[] args ) {

    throw  new  ArrayIndexOutOfBoundsException();

}
}
```

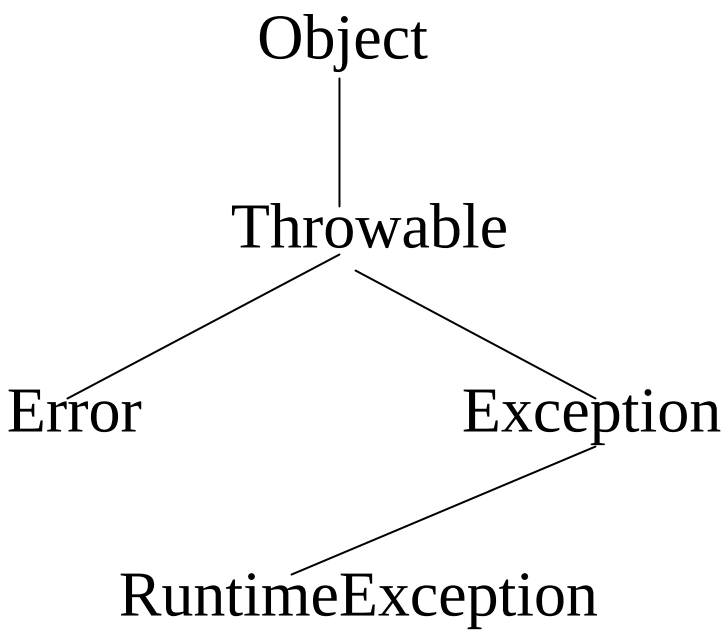
Where is the Handler placed?

```
class  ArrayOutOfBounds    {
   public static void main( String args[] )  {
        int  students[] = new int[5];
        try   {
          System.out.println(  "Start Try"  );
          students[  10  ]  =  1;
          System.out.println(  "After assignment statement"  );
         }
        //  catch is the handler
        catch (ArrayIndexOutOfBoundsException e)   {
          System.err.println( "OutOfBounds: "  +  e.getMessage());
        }
        System.out.println(  " After try " );
 }  }
```

Output

Start Try
OutOfBounds: 10
After try

Exceptions are objects



Try-Catch

try { <statements that may raise an exception>
}
catch (<exception class> <exception name>)
{
}
catch (<exception class> <exception name>)
{
}

Try-Catch

Try and catch are not separate blocks.

Try and catch are one construct.

Common errors:

- Use **try** without **catch**
- Use **catch** without **try**
- Put statement**s** between **try** and **catch**

How Does Exception Handling Work ?

- Execution of a thread is modelled by its call stack. Each method call corresponds to one record on the stack.
- When an exception object is thrown, the normal execution of the thread is interrupted. The exception object is propagated to higher level context until it is caught.

Context:

- an instance of a try-catch block or
- an instance of a method body.

- Execution is resumed at the catch block where the exception object is caught.

Exception Specification

- Java forces the programmer to specify the classes of *checked* exceptions that could be thrown out of a method.

<Return> <Method-name> (<Parameters>)

throws <Ex1>, <Ex2>, …, <Exk>

{ <Method-body>

}

Exception Specification Rules

- Checked: Methods are required to state if they throw these exceptions
- Unchecked : Methods are not required to state if they throw these exceptions
  - RuntimeException
  - Error

Rules for Checked Exceptions

The throws clause of a method must contain

- Any (checked) exception that
  - might be thrown inside the method, and
  - is not caught by a catch block of the method.
- Any (checked) exception that
  - appears on the **throws** clause of a method invoked by the method, and
  - is not caught by a catch block of the method.

Example

```
import java.io.*
Class ThrowsClass {
   void test() throws …
    {   …
        o.read() // throws IOException
        …
        throw new SQLException(“DB Access Error!”);
    }
}
```

Unchecked exception

Search the stack for the first enclosing try block

```
class  FindHandlerSimpleExample
   {
     public  static  void  change(  int[]  course  )
     {
       System.out.println( "Start Change"  );
       course[ 10 ]  =  1;
       System.out.println( "End Change"  );
     };
```

```
public  static  void  main( String args[] ) {
        int  students[] = new int[5];
        try  {
           System.out.println( "Start Try"  );
           change( students );
           System.out.println(  "After change"  ); }
        catch  (ArrayIndexOutOfBoundsException e)
        { System.err.println( "OutOfBounds: "  +
                                   e.getMessage());
        }
        System.out.println(  "After try "  );
      }
   }


//  Start Try
//  Start Change
//  OutOfBounds: 10
//  After try
```

Unchecked Exceptions

```
class  FindHandlerDoubleExample  {
  public  static  void  doubleChange(  int[]  course  )  {
    System.out.println( "Start Double Change"  );
    course[ 10 ]  =  1;
    System.out.println( "End Double Change"  );
  };

  public  static  void  change(  int[]  course  )  {
    System.out.println( "Start Change"  );
    doubleChange( course );
    System.out.println( "End Change"  );
  };
```

```
public  static  void  main( String args[] ) {
        int  students[] = new int[5];
        try  {
           System.out.println( "Start Try"  );
           change( students );
           System.out.println(  "After change"  ); }
        catch  (ArrayIndexOutOfBoundsException e)
        {  System.err.println( "OutOfBounds: "  +  e.getMessage());
        }
        System.out.println(  "After try "  );
     }
  }
  // Start Try
  // Start Change
  // Start Double Change
  // OutOfBounds: 10
  // After try
```

Unchecked Exceptions

```
class  FindHandlerTwoTryBlocks  {
  public  static  void  change(  int[]  course  )  {
    try
    {
      System.out.println( "Start Change"  );
      course[ 10 ]  =  1;
      System.out.println( "End Change"  );
    }
    catch  (ArrayIndexOutOfBoundsException e)
    {
      System.err.println( "Change Catch: "  +  e.getMessage());
    }
  };
```

```
public  static  void  main( String args[] )  {
        int  students[] = new int[5];
        try
        {  System.out.println( "Start Try"  );
           change( students );
           System.out.println(  "After change"  );
        }
        catch  (ArrayIndexOutOfBoundsException e)
        { System.err.println( "Main Catch: "  +  e.getMessage());
        }
        System.out.println(  "After try "  );
    }
}
```

```
Start Try
Start Change
Change Catch: 10
After change
After try
```

Unchecked Exceptions

What happens if there is no try block?

```
class  NoHandler  {
   public  static  void  change(  int[]  course  )
   {  System.out.println( "Start Change"  );
      course[ 10 ]  =  1;
      System.out.println( "End Change"  ); };
   public  static  void  main( String args[] )    {
      int  students[] = new int[5];
      System.out.println( "Start Try"  );
      change( students );
      System.out.println(  "After change"  );  }
}
```

```
Start Try
Start Change
java.lang.ArrayIndexOutOfBoundsException: 10
at test.main(classNotes.java)
```

Checked exceptions

Check Exceptions must be either:

- handled in the method they are thrown or
- the method in which they are thrown must state they will throw the exception

Example

```
import java.io.*;
class  CheckedExceptionDeclaredInMethod  {
    public  static  void  readSomething() throws IOException
      {
        System.out.println( "Please type an integer: " );
        int  Typed;
        DataInputStream  input =
              new DataInputStream (new FileInputStream(“Data_in”));
        Typed = input.readInt();   // throws IOException
      };
```

```
public  static  void  main( String args[] ) {
        try
        {
          System.out.println( "Start Try"  );
          readSomething();
          System.out.println(  "After readSomething"  );
        }
        catch  (IOException e)
        {
          System.err.println( "IO error: "  +  e.getMessage());
        }
        System.out.println(  "After try "  );
    }
  }
```

Checked exceptions

```
class  CheckedExceptionHandledInMethod   {
  public  static  void  readSomething()   {
        try  {
          System.out.println( "Please type an integer: " );
          int  Typed;
          DataInputStream  input =
            new DataInputStream (new FileInputStream(“Data_in”));
          Typed = input.readInt();   // throws IOException
          }
        catch  (IOException e)
        { System.err.println( "IO error: "  +  e.getMessage()); }
      };

   public  static  void  main( String args[] )
    {    System.out.println( "Start Try"  );
         readSomething();
         System.out.println(  "After readSomething"  );
    }
}
```

Checked exceptions

```
class  IfYouDontDeclareInMethod  {
       public  static  void  readSomething()   {
          System.out.println( "Please type an integer: " );
          int  Typed;
          DataInputStream  input =
                new DataInputStream (new FileInputStream("Data_in"));
          Typed = input.readInt();   // throws IOException
      };
      public  static  void  main( String args[] ) {
          try   {
            System.out.println( "Start Try"  );
            readSomething();
            System.out.println(  "After readSomething"  );
          }
          catch  (IOException e)
          {  System.err.println( "IO error: "  +  e.getMessage()); }
          System.out.println(  "After try "  );
       }
    }
// Compile Error
```

Re-throwing an exception

```
class  RethrowingException  {
  public  static  void  readSomething() throws
  { try     {
       System.out.println( "Please type an integer: " );
       int  Typed;
       DataInputStream  input =
        new DataInputStream (new FileInputStream(“Data_in”));
       Typed = input.readInt();   // throws IOException
    }
    catch  (IOException e)
    { System.err.println( "IO error: "  +  e.getMessage());
       throw e;
    }
  };
```

```
public  static  void  main( String args[] ) {
      try
      {
        readSomething();
      }
      catch  (IOException e)
      {
        System.err.println( "Error: "  +  e.getMessage());
      }
      System.out.println(  "After try "  );
   }
}
```

Multiple Exceptions

```
class  MultipleExceptions {
       public static void main( String args[] )       {
          int  students[] = new int[5];
          int  classSize  =  0;
          try          {
            System.out.println( "Start Try"  );
            int  classAverage  =  10 / classSize;
            System.out.println( "After class average"  );
            students[  10  ]  =  1;
            System.out.println(  "After array statement"  ); }
          catch  (ArrayIndexOutOfBoundsException e)
            { System.err.println( "OutOfBounds: "  +  e.getMessage());}
          catch  (  ArithmeticException  e  )
            {  System.err.println( " Math Error"  +  e.getMessage() );
            }
          System.out.println(  " After try " );
          } }
```

// Start Try

// Math Error/ by zero

// After try

One Size Fits All Exception

```
class  MultipleExceptions  {
       public static void main( String args[] )  {
            int  students[] = new int[5];
            int  classSize  =  0;
        try  {   System.out.println( "Start Try"  );
                  students[  10  ]  =  1;
                  System.out.println(  "After array statement"  );
                  int  classAverage  =  10 / classSize; }
          catch  ( Exception  e )    // Not a good idea
            { System.err.println( "Exception:"  +  e.getMessage());
               e.printStackTrace();
            }
          }
       }

    // Start Try
    // Exception:10
    // java.lang.ArrayIndexOutOfBoundsException: 10
    // at MultipleExceptions.main(All.java:11)
```

Beware of Order

```
Class  MultipleExceptions  {
       public static void main( String args[] ) {
          int  students[] = new int[5];
          int  classSize  =  0;
          try  {  System.out.println( "Start Try"  );
                  students[  10  ]  =  1;
                  System.out.println(  "After array statement"  );
                  int  classAverage  =  10 / classSize;
          }
          catch  ( Exception  e )
          {  System.err.println( "Exception:"  +  e.getMessage());
             e.printStackTrace();
          }
          catch  (ArrayIndexOutOfBoundsException e)
          {  System.err.println( "OutOfBounds: "  +  e.getMessage());
          }
      }
}
//  Compile Error, second catch can not be reached.
```

Final Block - For Clean Up

```
class  FinalBlockWithExceptionCalled  {
       public static void main( String args[] ) {
          int  students[] = new int[5];
          try  {  System.out.println( "Start Try"  );
                    students[  10  ]  =  1;
                    System.out.println(  "After array statement"  ); }
          catch  (ArrayIndexOutOfBoundsException e)
           { System.err.println( "OutOfBounds: "  +  e.getMessage());}
          finally {  System.out.println(  "In Final Block"  ); }
          System.out.println(  " After try " );
       }
    }
// OutOfBounds: 10
// In Final Block
// After try
```

Final Block - Always Called

```
class  FinalBlockExceptionNotCalled  {
       public static void main( String args[] )
       { int  students[] = new int[5];
         try  {
            System.out.println( "Start Try"  );
            students[  2  ]  =  1;
            System.out.println(  "After array statement"  );
         }
         catch  (ArrayIndexOutOfBoundsException e)
         { System.err.println( "OutOfBounds: "  +  e.getMessage()); }
         finally   { System.out.println(  "In Final Block"  ); }
         System.out.println(  " After try "  );
       }
    }
//Start Try
//After array statement
//In Final Block
//After try
```

Can't Escape Finally

```
class  ExceptionFunction  {
     public  static  boolean  change(  int[]  course  )   {
       try  { System.out.println( "Start change"  );
              course[ 10 ]  =  1;
              System.out.println( "End Change"  ); }
       catch  (ArrayIndexOutOfBoundsException e)
       { System.err.println( "OutOfBounds: " );
         return  true;
       }
       finally
       { System.out.println(  "In Final Block"  ); }
       return  false;
     };
```

```
public  static  void  main( String args[] )
      {
        int  students[] = new int[5];
        System.out.println( "Main"  );
        System.out.println(  change( students )  );
        System.out.println(  "After Change"  );
      }
   }



// Main                  //  In Final Block
// Start Change          //  true
// OutOfBounds:          //  After Change
```

Finally is Master

A finally block is entered with a reason, such as end of the method, an exception was thrown

If the finally block creates it own reason to leave (throws an exception, executes a return or break) then the original reason is forgotten

Example

```
class  FinalExample  {
      public static void explainThis() {
        int  students[] = new int[5];
        try  { System.out.println( "Start Try"  );
                students[  10  ]  =  1; }
        finally {
          System.out.println(  "In Final Block"  );
          return; }
       }
```

```
public  static  void  main( String  args[]  )  {
        try  {   explainThis(); }
        catch  (ArrayIndexOutOfBoundsException missed)
          {
            System.out.println(  "In ArrayIndexOutOfBoundsException"  );
          }
        System.out.println(  "After try in main"  );
        }
}

// Start Try
// In Final Block
//After try in main
```

Exceptions inside Exceptions

```
class  ExceptionFunction  {
     public  static  boolean  change(  int[]  course  )  {
       try  {  System.out.println( "Start change"  );
                 course[ 10 ]  =  1; }
       catch  (ArrayIndexOutOfBoundsException e) {
         course[ 10 ]  =  1;
         System.err.println( "OutOfBounds: " );
         return  true; }
       finally   { System.out.println(  "In Final Block"  ); }
       return  false;
     };
```

```
public  static  void  main( String args[] ) {
        try  {
           int  students[] = new int[5];
           System.out.println( "Main"  );
           System.out.println(  change( students )  );
           System.out.println(  "After Change"  );
        }
        catch  (ArrayIndexOutOfBoundsException e) {
           System.out.println( "Over Here" );
        }
     }
  }

  // Main
  // Start change
  // In Final Block
  // Over Here
```

Information passed to the handler

- Two pieces of information are standar
  - Message
  - Stack Info
- When you define your own exceptions you can add more data if needed

Modifying Stack Trace

```
class XXX {
  void origin() throws Exception {
     throw new Exception ("Thrown by "); }
  void processor() throws Exception {
     try { origin(); }
     catch (Exception e)
     { System.out.println(e.getMessage());
       throw e.fillInStackTrace(); }
  }
  public static void main(String[] args) {
     try { processor();}
     catch (Exception e)
     { System.out.println(e.getMessage());
       throw e.fillInStackTrace(); }
  }
}
```

```
class  FindHandlerSimpleExample  {
      public  static  void  change(  int[]  course  )  {
        course[ 10 ]  =  1;
      };

      public  static  void  main( String args[] ) {
        int  students[] = new int[5];
        try  {
          change( students );
        }
        catch  (ArrayIndexOutOfBoundsException e)  {
          System.err.println( "Message: "  +  e.getMessage()  );
          e.printStackTrace();
        }
      }
    }
```

```
Message: 10
java.lang.ArrayIndexOutOfBoundsException: 10
at test.change(All.java:7)
at test.main(All.java:13)
```

Exceptions and Inheritance

A method that overrides another method may not be declared to throw more checked exceptions than the overridden method.

Example.

- Let class Child be a subclass of class Parent
- Let foo be a method of Parent that is overridden by Child
- If Child.foo has a throws clause that mentions any checked exceptions, then Parent.foo must have a throws clause.
- For every checked exception listed in Child.foo, that same exception class or one of its superclasses must occur in the throws clause of Parent.foo

```
class Parent
{
        public void foo() throws IOException {};
        public void bar() {};
}

class Child extends Parent
{
        public void foo() {};          // OK
        public void bar() throws IOException{};
                                       // Compile time error
}
```

User Defined Exceptions

```
class  BoringLecture  extends  Exception
   public  BoringLecture ()  {
                super();
         }

   public  BoringLecture ( String msg )
         {  super(  msg ); }
    }
```

```
class  NewExceptionTest  {
    public static void dullSpeaker() throws BoringLecture {
      // Code can go here
      throw  new  BoringLecture (  "Change Topics"  );
    }

    public  static  void  main(  String  args[]  )   {
      try  { dullSpeaker(); }
      catch  (  BoringLecture  e  )
      { System.err.println( "Time to "  +  e.getMessage() );
      }
    }
  }
```



```
// Time to Change Topics
```