

JAVA EXCEPTIONS CONTINUED

WHY USE EXCEPTIONS?

- Lets see an example!..

```
readFile {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
}
```



- What happens if the file can't be opened?
- What happens if the length of the file can't be determined?
- What happens if enough memory can't be allocated?
- What happens if the read fails?
- What happens if the file can't be closed?

```
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```



```
readFile {  
  try {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
  } catch (fileOpenFailed) {  
    doSomething;  
  } catch (sizeDeterminationFailed) {  
    doSomething;  
  } catch (memoryAllocationFailed) {  
    doSomething;  
  } catch (readFailed) {  
    doSomething;  
  } catch (fileCloseFailed) {  
    doSomething;  
  }  
}
```



EXCEPTION TYPES

- Checked Exception
 - Checked exceptions are checked at compile-time (IOException, SQLException, ClassNotFoundException, InvocationTargetException)
- Unchecked Exception
 - Unchecked exceptions are not checked at compile-time rather they are checked at runtime
- Error
 - Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError

```
import java.io.*;
class ExceptionExample1 {
    public static void main(String args[])
    {
        FileInputStream fis = null;
        /*This constructor FileInputStream(File filename)
        * throws FileNotFoundException which is a checked
        * exception
        */
        fis = new FileInputStream("B:/myfile.txt");
        int k;

        /* Method read() of FileInputStream class also throws
        * a checked exception: IOException
        */
        while(( k = fis.read() ) != -1)
        {
            System.out.print((char)k);
        }

        /*The method close() closes the file input stream
        * It throws IOException*/
        fis.close();
    }
}
```

```
Command Prompt

D:\>cd myjavaprogs

D:\myjavaprogs>javac ExceptionExample1.java
ExceptionExample1.java:10: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
    fis = new FileInputStream("B:/myfile.txt");
           ^
ExceptionExample1.java:16: error: unreported exception IOException; must be caught or declared to be thrown
    while(( k = fis.read() ) != -1)
                ^
ExceptionExample1.java:23: error: unreported exception IOException; must be caught or declared to be thrown
    fis.close();
        ^
3 errors

D:\myjavaprogs>
```

```
import java.io.*;
class ExceptionExample2 {
    public static void main(String args[]) throws IOException
    {
        FileInputStream fis = null;
        /*This constructor FileInputStream(File filename)
        * throws FileNotFoundException which is a checked
        * exception
        */
        fis = new FileInputStream("myfile.txt");
        int k;

        /* Method read() of FileInputStream class also throws
        * a checked exception: IOException
        */
        while(( k = fis.read() ) != -1)
        {
            System.out.print((char)k);
        }

        /*The method close() closes the file input stream
        * It throws IOException*/
        fis.close();
    }
}
```



```
Command Prompt
D:\myjavaprogs>javac ExceptionExample2.java
D:\myjavaprogs>java ExceptionExample2
Good Morning
Unipi
Students of CS!
D:\myjavaprogs>_
```

UNCHECKED EXCEPTIONS EXAMPLES

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

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

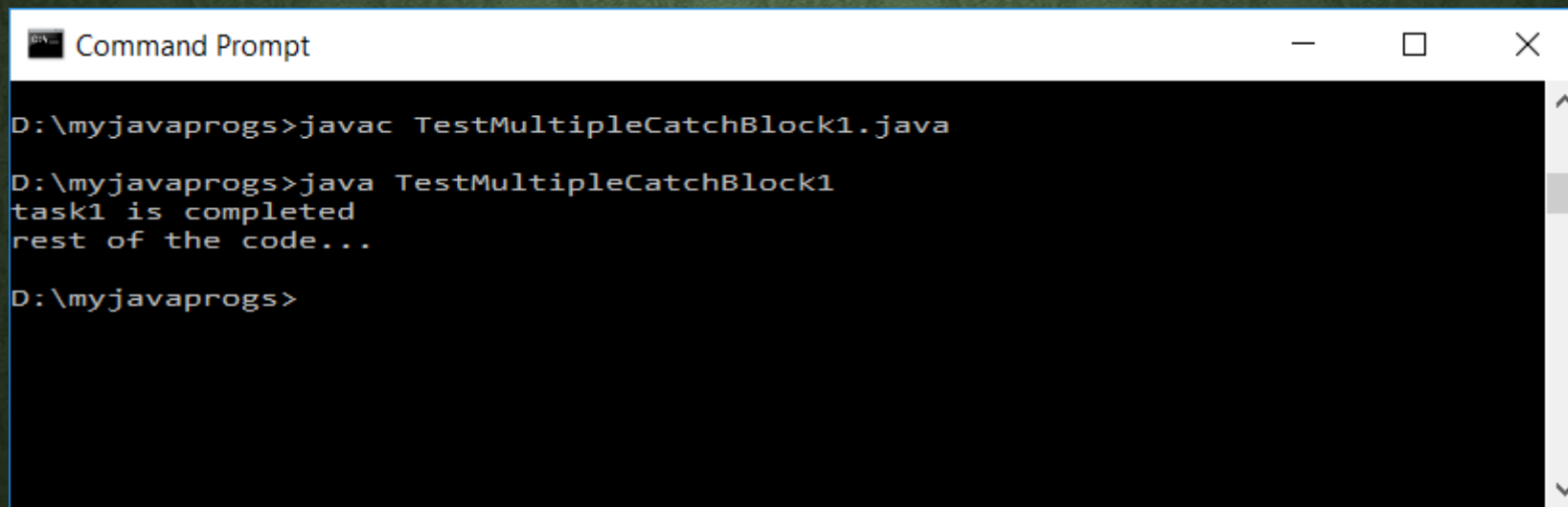
```
String s="abc";  
int i=Integer.parseInt(s);//NumberFormatException
```

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

SOME NOTES! 1/2

- At a time only one Exception is occurred and at a time only one catch block is executed
- All catch blocks must be ordered from most specific to most general i.e. catch for `ArithmeticException` must come before catch for `Exception`
- For each try block there can be zero or more catch blocks, but only one finally block
- The finally block will not be executed if program exits(either by calling `System.exit()` or by causing a fatal error that causes the process to abort)

```
class TestMultipleCatchBlock1 {
    public static void main(String args[]) {
        try {
            int a[] = new int[5];
            a[5] = 30/0;
        }
        catch (ArithmeticException e) {System.out.println("task1 is completed");}
        catch (ArrayIndexOutOfBoundsException e) {System.out.println("task 2 completed");}
        catch (Exception e) {System.out.println("common task completed");}
        System.out.println("rest of the code...");
    }
}
```



```
Command Prompt

D:\myjavaprogs>javac TestMultipleCatchBlock1.java

D:\myjavaprogs>java TestMultipleCatchBlock1
task1 is completed
rest of the code...

D:\myjavaprogs>
```

SOME NOTES! 2/2

- If you are calling a method that declares an exception, you must either catch or declare the exception

```
import java.io.*;

class TestException2{
    void m(){
        throw new IOException("device error");//checked exception
    }
    void n(){
        m();
    }
    void p(){
        try{
            n();
        }catch(Exception e){System.out.println("exception handled");}
    }
    public static void main(String args[]){
        TestException2 obj=new TestException2();
        obj.p();
        System.out.println("normal flow");
    }
}
```

```
Command Prompt
D:\myjavaprogs>javac TestException2.java
TestException2.java:5: error: unreported exception IOException; must be caught o
r declared to be thrown
        throw new IOException("device error");//checked exception
            ^
1 error

D:\myjavaprogs>
```



```
import java.io.*;

class TestException2{
    void m() throws IOException{
        throw new IOException("device error");//checked exception
    }
    void n() throws IOException{
        m();
    }
    void p(){
        try{
            n();
        }catch(Exception e){System.out.println("exception handled");}
    }
    public static void main(String args[]){
        TestException2 obj=new TestException2();
        obj.p();
        System.out.println("normal flow");
    }
}
```

```
Command Prompt
D:\myjavaprogs>javac TestException2.java
TestException2.java:5: error: unreported exception IOException; must be caught o
r declared to be thrown
    throw new IOException("device error");//checked exception
    ^
1 error

D:\myjavaprogs>javac TestException2.java

D:\myjavaprogs>java TestException2
exception handled
normal flow

D:\myjavaprogs>
```

HOW TO THROW EXCEPTIONS

- Regardless of what throws the exception, it's always thrown with the *throw* statement
- All methods use the *throw* statement to throw an exception
- The *throw* statement requires a single argument: a *throwable* object
- Throwable objects are instances of any subclass of the *Throwable* class
- Most programs throw and catch objects that derive from the *Exception* class

EXCEPTION HANDLING AND METHOD OVERRIDING

- If the superclass method does not declare an exception, a subclass overridden method cannot declare a checked exception
- If the superclass method does not declare an exception, a subclass overridden method can declare unchecked exception
- If the superclass method declares an exception, a subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception

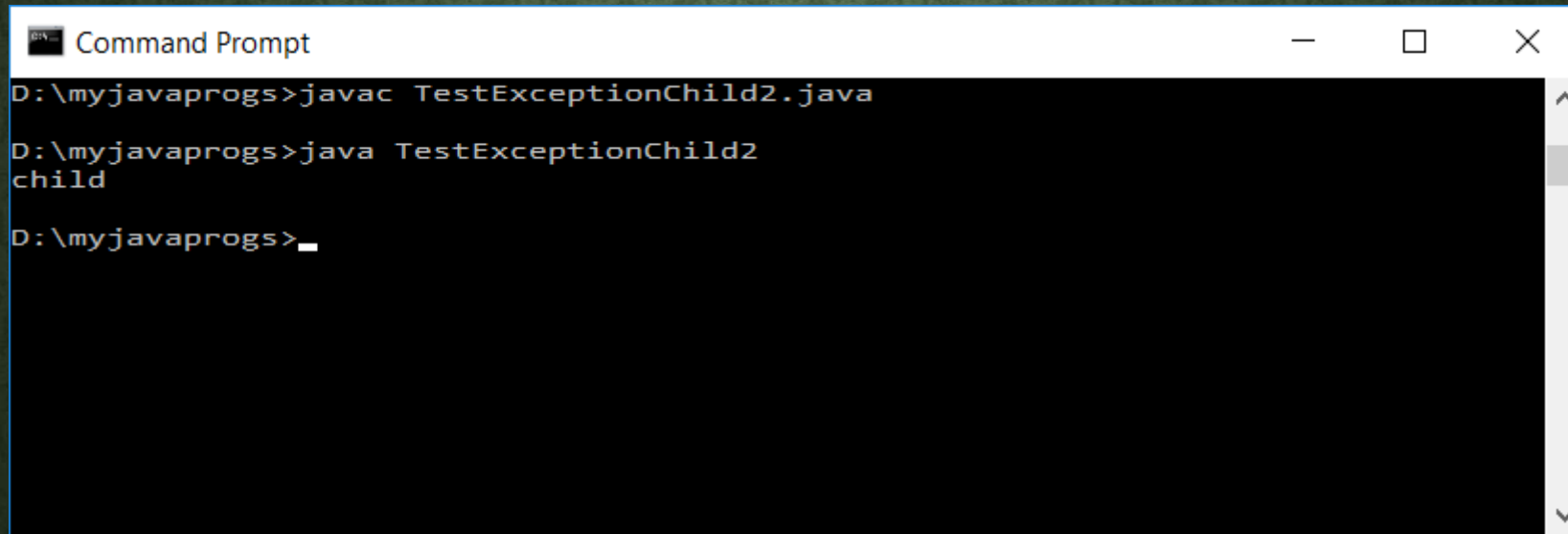
```
import java.io.*;
class Parent{
    void msg() {System.out.println("parent");}
}

class TestExceptionChild1 extends Parent{
    void msg() throws IOException{
        System.out.println("TestExceptionChild");
    }
    public static void main(String args[]){
        Parent p=new TestExceptionChild1();
        p.msg();
    }
}
```

```
Command Prompt
D:\>cd myjavaprogs
D:\myjavaprogs>javac TestExceptionChild1.java
TestExceptionChild1.java:7: error: msg() in TestExceptionChild1 cannot override
msg() in Parent
    void msg()throws IOException{
           ^
    overridden method does not throw IOException
1 error
D:\myjavaprogs>
```

```
import java.io.*;
class Parent{
    void msg() {System.out.println("parent");}
}

class TestExceptionChild2 extends Parent{
    void msg() throws ArithmeticException{
        System.out.println("child");
    }
    public static void main(String args[]){
        Parent p=new TestExceptionChild2();
        p.msg();
    }
}
```



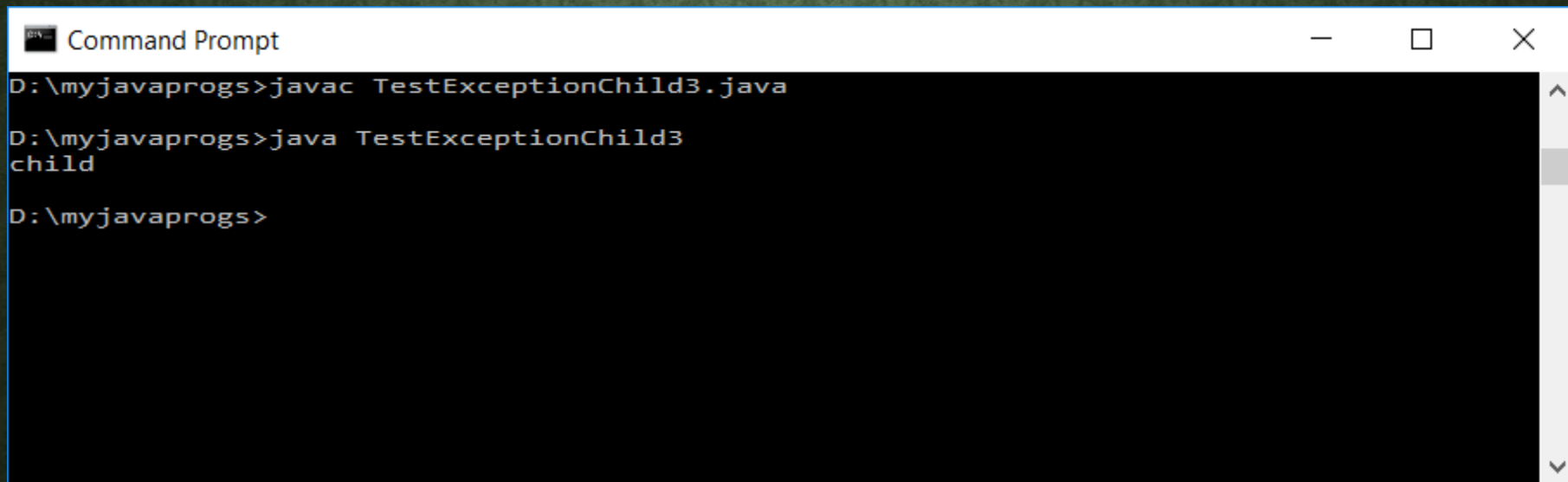
```
Command Prompt
D:\myjavaprogs>javac TestExceptionChild2.java
D:\myjavaprogs>java TestExceptionChild2
child
D:\myjavaprogs>_
```



```
import java.io.*;
class Parent{
    void msg() throws Exception{System.out.println("parent");}
}

class TestExceptionChild3 extends Parent{
    void msg() throws ArithmeticException{System.out.println("child");}

    public static void main(String args[]){
        Parent p=new TestExceptionChild3();
        try{
            p.msg();
        }catch(Exception e){}
    }
}
```



```
Command Prompt
D:\myjavaprogs>javac TestExceptionChild3.java
D:\myjavaprogs>java TestExceptionChild3
child
D:\myjavaprogs>
```

THE TRY-WITH-RESOURCES STATEMENT

- The *try-with-resources* statement is a try statement that declares one or more resources
- A resource is an object that must be closed after the program is finished with it
- The *try-with-resources* statement ensures that each resource is closed at the end of the statement
- Any object that implements *java.lang.AutoCloseable*, which includes all objects which implement *java.io.Closeable*, can be used as a resource

```
import java.io.*;

class TestTry1{

    private static void printFile() throws IOException {
        InputStream input = null;
        try {
            input = new FileInputStream("file.txt");
            int data = input.read();
            while(data != -1){
                System.out.print((char) data);
                data = input.read();
            }
        } finally {
            if(input != null){
                input.close();
            }
        }
    }

    public static void main(String args[]){
        try{
            printFile();
        }catch(Exception m){System.out.println("Exception occurred: "+m);}
        System.out.println("rest of the code...");
    }
}
```

```
Command Prompt
D:\myjavaprogs>javac TestTry1.java
D:\myjavaprogs>java TestTry1
Exception occurred: java.io.FileNotFoundException: file.txt (The system cannot find the f
ile specified)
rest of the code...
D:\myjavaprogs>
```

```
import java.io.*;

class TestTry2{

    private static void printFileJava7() throws IOException {

        try(FileInputStream input = new FileInputStream("file.txt")) {

            int data = input.read();
            while(data != -1){
                System.out.print((char) data);
                data = input.read();
            }
        }
    }

    public static void main(String args[]){
        try{
            printFileJava7();
        }catch(Exception m){System.out.println("Exception occurred: "+m);}
        System.out.println("rest of the code...");
    }
}
```

```
Command Prompt
D:\myjavaprogs>javac TestTry2.java
D:\myjavaprogs>java TestTry2
Exception occurred: java.io.FileNotFoundException: file.txt (The system cannot find the f
ile specified)
rest of the code...
D:\myjavaprogs>_
```

CATCH MULTIPLE EXCEPTIONS JAVA 7

```
try {  
  
    // execute code that may throw 1 of the 3 exceptions below.  
  
} catch(SQLException | IOException e) {  
    logger.log(e);  
  
} catch(Exception e) {  
    logger.severe(e);  
}
```


CUSTOM EXCEPTIONS

- User-defined exceptions
- Extend `Throwable` or `Exception`

```
import java.io.*;

class InvalidAgeException extends Exception{
    InvalidAgeException(String s){
        super(s);
    }
}

class TestCustomException1{

    static void validate(int age) throws InvalidAgeException{
        if(age<18)
            throw new InvalidAgeException("not valid age");
        else
            System.out.println("welcome to voting system");
    }

    public static void main(String args[]){
        try{
            validate(13);
        }catch(Exception m){System.out.println("Exception occurred: "+m);}

        System.out.println("rest of the code...");
    }
}
```

```
Command Prompt
D:\myjavaprogs>javac TestCustomException1.java
D:\myjavaprogs>java TestCustomException1
Exception occurred: InvalidAgeException: not valid age
rest of the code...
D:\myjavaprogs>_
```



IS THE FOLLOWING CODE LEGAL?

```
try {  
  
} finally {  
  
}
```

ANSWER

- Yes, it's legal — and very useful A try statement does not have to have a catch block if it has a finally block

WHAT IS WRONG WITH USING THIS TYPE OF EXCEPTION HANDLER?

```
catch (Exception e) {  
  
}
```

ANSWER

- This handler catches exceptions of type Exception; therefore, it catches any exception. This can be a poor implementation because you are losing valuable information about the type of exception being thrown and making your code less efficient

**IS THERE ANYTHING WRONG WITH THIS
EXCEPTION HANDLER AS WRITTEN?
WILL THIS CODE COMPILE?**

```
try {  
  
} catch (Exception e) {  
  
} catch (ArithmeticException a) {  
  
}
```

ANSWER

- This first handler catches exceptions of type `Exception`; therefore, it catches any exception, including `ArithmeticException`. The second handler could never be reached. This code will not compile

MATCH EACH SITUATION IN THE FIRST LIST WITH AN ITEM IN THE SECOND LIST

- `int[] A; A[0] = 0;`
- The JVM starts running your program, but the JVM can't find the Java platform classes. (The Java platform classes reside in `classes.zip` or `rt.jar`.)
- A program is reading a stream and reaches the end of stream marker.
- Before closing the stream and after reaching the end of stream marker, a program tries to read the stream again.

- ❖ `__error`
- ❖ `__checked exception`
- ❖ `__compile error`
- ❖ `__no exception`