Error Handling in Java

• Run time errors
  ✦ It is difficult to recover gracefully from run-time errors that occur in the middle of a program.
  ✦ At the point where the problem occurs, there isn’t enough information in that context to resolve the problem.
  ✦ So that function/method hands off the problem out to a higher context where someone is qualified to make the proper decision.
  ✦ There is no need to check for errors at multiple places (after each call to access a file, for instance). The code to handle a given error can be put in a single location in the code (the exception handler).

• If the error can be resolved in the immediate context where it occurs, it is NOT called an exception.

Exception semantics - 1

• When an error occurs inside a method, the method creates an exception object.
  ✦ could be in a library method or a user-defined method

• The exception object contains information about the error, including:
  ✦ the type of the exception and
  ✦ the state of the program when the error occurred (the call stack)

• Creating an exception and reporting it to the runtime system is called throwing an exception.

Exception semantics - 2

• When a method throws an exception,
  ✦ the current path of execution is interrupted, and
  ✦ the runtime system attempts to find an appropriate place to continue executing the program.

• The runtime system searches the call stack for an appropriate exception handler
  ✦ the call stack: the list of methods that have been called and are waiting for the current method to return.
  ✦ A calls B that calls C that calls D: The call stack contains A, B, C and D with D on the top.
Exception semantics - 3

- The runtime system is looking for a previous method call that is embedded in a block that has an exception handler associated with it.
  - It starts at the top of the call stack and goes down (in reverse order in which the methods were called)

- The runtime system is searching for an appropriate exception handler
  - An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler
  - “Matching” is the same as is used for function calls, a thrown exception matches any superclass of its type.

Exception semantics - 4

- The first exception handler encountered that matches the exception is said to catch the exception.

- If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, the runtime system terminates the program.

Exception syntax: how to throw an exception

- To throw an exception, use the keyword throw.
- To create an exception, use the appropriate constructor.

```java
if (t==null)
    throw new NullPointerException();
```

- This will cause the method to be exited.
- Exception classes can be found in the API website: see java.lang.Exception

Exception syntax: how to catch an exception

- To catch an exception, use the try-catch block.
- Surround the code that might generate an exception in the try
- Make an exception handler for every exception type you want to catch.

```java
try {
    // Code that might generate exceptions
} catch(Type1 id1) {
    // Handle exceptions of Type1
} catch(Type2 id2) {
    // Handle exceptions of Type2
} catch(Type3 id3) {
    // Handle exceptions of Type3
}
// etc...
```
Exception syntax: how to catch an exception

- Each catch clause is like a little method that takes one argument of a particular type.
- The parameter (id1, id2, and so on) can be used inside the handler, just like a method argument.
- If the handler catches an exception, its catch block is executed, and the flow of control proceeds to the next statement after the try/catch.
  ✦ only the first matching catch clause is executed.

The exception specification: being civil

- In Java, you’re (strongly!) encouraged to inform the client programmer, who calls your method, of the exceptions that might be thrown from your method.
  ✦ Then the caller can know exactly what catch clauses to write to catch all potential exceptions.
- The exception specification states which exceptions are thrown by a method.

```java
void f() throws TooBig, TooSmall, DivZero { //...

✦ Catch or specify requirement: If the method generates exceptions, it must handle them or specify them in the signature.
  ✦ Otherwise it’s a compiler error.
```

Catch or Specify: example

```java
// Note: This class won’t compile by design!
import java.io.*;
import java.util.Vector;
public class ListOfNumbers {
  private Vector<Integer> vector;
  private static final int SIZE = 10;
  public ListOfNumbers () {
    vector = new Vector<Integer>(SIZE);
    for (int i = 0; i < SIZE; i++) {
      vector.addElement(new Integer(i));
    }
  }
  public void writeList() {
    PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
    for (int i = 0; i < SIZE; i++) {
      out.println("Value at: " + i + ": " + vector.elementAt(i));
    }
    out.close();
  }
}
```

Catch or Specify: solution 1

```java
// Note: This class won’t compile by design!
import java.io.*;
import java.util.Vector;
public class ListOfNumbers {
  private Vector<Integer> vector;
  private static final int SIZE = 10;
  public ListOfNumbers () {
    vector = new Vector<Integer>(SIZE);
    for (int i = 0; i < SIZE; i++) {
      vector.addElement(new Integer(i));
    }
  }
  public void writeList() throws IOException {
    PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
    for (int i = 0; i < SIZE; i++) {
      out.println("Value at: " + i + ": " + vector.elementAt(i));
    }
    out.close();
  }
}
```
public void writeList() {
    PrintWriter out = null;
    try {
        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + vector.elementAt(i));
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (out != null) {
            out.close();
        }
    }
}

The finally block

- The finally block is an additional block you can add to the try catch.
- The finally block ALWAYS executes when the try block exits
  - Whether it exited through an exception handler or just normal termination.