July 14, 2010

Exception Handling

There are lots of events that can cause runtime errors in your program. Some are avoidable, others are not, but you will get blamed for all of them if they happen.

- Dividing by zero: 1 / 0
- Array index out of bounds: a[-1]
- Accessing a file which cannot be found: new File("this_file_does_not_exist")
- Null References: Scanner scan = null; scan.nextInt();
- Reading an integer using characters.

Java provides a syntax built into the language for handling problems in various ways. There are three ways to handle problems:

- Ignore it. Do nothing. Maybe the problem will never arise. And if it does, "Garbage in, garbage out.". When a Java program encounters an exception that isn't handled, it will abort the program and print a "Stack Trace" to the screen. This is handy if you are debugging, because it will report all of the method calls from the start of the program to the current method and which line number called which method.
- Pass the problem off to something else in Java.

```
void myMethod() throws Exception {
	// Statements
	// Some bad condition happens which the programmer knows is a problem
	if ( /* bad condition */ ) {
		throw new Exception("Bad Condition Happened");
	}
}
```

- Attempt to catch these problems as they happen using "try" blocks to detect a problem and "catch" blocks to execute when a problem does happen.

```
void myMethod() {
	try {
		// Statements
		// Some bad condition happens which the programmer knows is a problem
		if ( /* bad condition */ ) {
			throw new Exception("Bad Condition Happened");
	}
	catch (Exception e) {
		// Handle the problem here.
	}
}
```

In the class "Scanner", there is a tool for reading a number called "nextInt()". This method has an exception which is triggered when the user types anything which isn't a number called "InputMismatchException". Here's how to handle it using try/catch blocks.

```
Scanner scan = new Scanner(System.in);
int n = 0;
while (true) {
        try {
                n = scan.nextInt();
                break;
        }
        catch (InputMismatchException e) {
                System.out.println("Didn't type a number.");
                scan.nextLine();
        }
}
System.out.println("You typed number "+n);
```

To better illustrate the exception handling traits in Java, this good example is found in the book:

```
// From "Java Program" by Malik, 1-4188-3540-4
// Chapter 12, Page 773
//

import java.io.*;

public class PrintStackTraceExample1 {
        public static void main(String[] args) {
                try {
                        methodA();
                }
                catch (Exception e) {
                        System.out.println(e.toString() + " caught in main");
                        e.printStackTrace();
                }
        }

        public static void methodA() throws Exception { methodB(); }

        public static void methodB() throws Exception { methodC(); }

        public static void methodC() throws Exception {
                throw new Exception("Exception generaged in method C");
        }
}
```

Making your own exceptions is simply making a new class and extending Exception. Make the class name the name of the exception and make the constructor accept a String parameter and then call “super” on that parameter.

```
class MyOwnException extends Exception {
    MyOwnException(String s) {
        super(s);
    }
}
```

Using your exception class like you would any other exception error.

```
void myMethodWhichDoesStuff() throws MyOwnException {
    // Statements
    if ( /* bad condition */ )
        throw new MyOwnError("My Own Error Exception");
}
```