**Advanced Java**

**Chapter 12 - Exceptions**

What are *preconditions* and *postconditions*?

The conditions that must be met before performing a function call

What is a *contract*?

State three traditional ways that programs use to deal with error conditions.

\* return an Error

\* Displayy an error message

\* Halt

What is an *exception*?

Simple demo: write a program that performs the following tasks (similar to Listing 14.4)

In an infinite loop

Prompt the user for two integers

If the first integer is zero, break out of the loop and exit

If the second integer is zero, throw an ArithmeticException

If both integers are nonzero,

divide the first by the second and display the quotient and remainder.

Try dividing a floating-point number by zero and displaying the result. What do you get?

Four exception key words:

try{

// a block of code that might generate an exception

}

throw( )

// throws an exception, specified in the parenthesis

catch{

// a block of code that might be invoked when one of several

// exceptions is thrown.

// Every “try” must be followed by one or more “catches”

}

finally{

// A block of code that executes when the try block exits,

// regardless of whether or not an exception was thrown.

// This is useful for clean-up code that might otherwise

// be skipped if an exception is thrown.

}

The two unchecked exception classes (and their members) are

Error

LinkageError

VirtualMachineError

RuntimeException

ArithmeticException

NullPointerException

IndexOutOfBoundsException

IllegalArgumentException

We don’t have to write try/catch blocks for code that might throw an unchecked exception. However, any code that we write that might throw a checked exception must be enclosed within a try block. Furthermore, any code that invokes Java constructors that themselves might throw checked exceptions (the culprit is usually some kind of IOException) must be enclosed in a try block (for example, attempting to open a file that’s not actually there will throw a FileNotFoundException, which is a subclass of IOException).

**Opening a File**

We do a lot of this in here, so let’s adopt some standard practices:

* If the first thing a program does is prompt for a file name, check the command-line parameters first.
* If the file doesn’t open, catch the FileNotFoundException and exit gracefully.

Demo code…

**Custom Exceptions**

Why, yes, we can write our own exception classes. More demo code…

**Homework**

Add code to your OCCCDate class constructors that validates the date as described below and throws a custom exception, InvalidOCCCDateException, if the date is invalid. The test for validity is

“Thirty days hath September, April, June, and November;

All the rest have thirty-one, except for February alone.”

The modern leap-year rule is that February has 28 days (that is, a common year) if the year is not divisible by 4 or by 100. Years that are divisible by 4 *and* 400 are leap years. The algorithm is…

**if** (*year* is not divisible by 4) **then** (it is a common year)  
**else if** (*year* is not divisible by 100) **then** (it is a leap year)  
**else if** (*year* is not divisible by 400) **then** (it is a common year)  
**else** (it is a leap year)

Now whenever you attempt to create a new OCCCDate, an exception might be thrown. Write a program to demonstrate OCCCDate (and OCCCPerson) with this added feature. Put the date-check algorithm in a boolean method. Demonstrate the use of this exception in two ways: without try (so the thrown exception ends the program) and with a try/catch block so that you can recover from the error and continue execution.

Note that it would be better for the OCCCDate constructor to throw one of the unchecked exceptions so that the user code doesn’t have to “try” to create OCCCDates. Therefore our exception class will extend IllegalArgumentException rather than just plain Exception. Remember that the reason Java gives us IllegalArgumentException so that constructors and methods can throw them, un-checked.

Your program should demonstrate every method of OCCCDate and should include ample opportunity for the user to enter dates.

Note that you submit three files: OCCCDate.java and two driver programs (one that uses try/catch and one that does not).

\*\*\*\* isValidDate(int m, int d, int y){

create Gregorian calendar with the above

then test if gregorian calendar returns the same as the original m,d,y; !!!!!

}

\*\* add super() to the constructor

Checked exception always have to have a TRY method when