Ch 12 - Exceptions and Files

12.1 Handling Exceptions

`Exception` is a class. Exceptions are objects. Cool. Actually `Exception` is-a `Throwable`. There are many classes that extend `Exception`, like `IOException`. The `Exception` class hierarchy is quite large. (page 735 and 749)

Use a try-catch block:

```
try {
   // code
}
catch( ExceptionType parameter) {
   // exception-handling code
}
```

Details:

- You can print an exception or get a "detail message string" for a `Throwable` at the method `getMessage()`
- Catch multiple exceptions in a `try` block, by adding `catch` blocks one after another
- A `finally` block can appear after the try and all catch blocks. "The statements in a finally block execute whether an exception occurs or not." Example: close a file.
- An `Exception` that you don't ever catch is caught by Java's default exception handler. It will print a stack trace of method calls. If you're fond of this kind of message, you can print this using the `Throwable` method `printStackTrace()`.

Two flavors of exceptions:

➔ **unchecked** - those inherited from the `Error` class or `RuntimeException` class in Java. These are usually critical errors.

➔ **checked** - exceptions usually handled in your code

Your methods must handle any checked exceptions by either: a) try-catch block, or b) declaring that the method `throws` the exception. Like this:

```
public void example( int x) throws FileNotFoundException { …  }
```

12.2 Throwing exceptions

You can throw your own exceptions. Format is:

```
throw new ExceptionType( messageString);
```

You can create your own exceptions just like a regular, old class. Example:

```
MyException extends Exception
```

12.3 Advanced Files

Three subsections: binary files, random access files, and serialization

Binary Files

A file that contains raw binary data is known as a binary file (page 761)
Numbers we see as Unicode chars can be smushed into binary form to save space.

Write binary files using 2 classes:

- `FileOutputStream` - very low level, write bytes or raw data
- `DataOutputStream`. - higher level, write data types (int, float, etc) in binary

Usually wrap DOS around FOS, like this:

```
DataOutputStream dos =
      new DataOutputStream( new FileOutputStream( "test.dat"));
```

Reading mirrors writing with `FileInputStream` and `DataInputStream`.

Random Access Files

A random access file allows a program to read data from any location within the file. (page 761)

- Use `RandomAccessFile` class: docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html
- Data is still written sequentially.
- Data can be read in any order. Your call to seek() sets the file pointer to that byte location. That's where the next read occurs. Method signature: `void seek( long pos)`

Serialization

Object serialization is the process of converting an object to a series of bytes and saving them to a file. Deserialization is the process of reconstructing a serialized object. (page 761)

Steps:

- Make sure your class `implements Serializable`.
- Write your serializable objects using class `ObjectOutputStream`. The method of interest is `writeObject( Object obj)`.
- Read your serializable objects using class `ObjectInputStream`. The method here is (you guessed it) `Object readObject()`.