More on Exceptions

Ch 8.3
- throws

```java
public int divide(int a, int b) throws ArithmeticException {
    return a/b;
}
```

- finally

```java
try{
    dangerousMethod();
}
catch(Exception e){
    System.out.println(e.toString());
}
finally{
    System.out.println("Ending program");
    System.exit(0);
}
```
So far we have used exceptions to handle runtime errors using `try` and `catch` blocks.

Exceptions sometimes need to be passed to other parts of code that can handle it better.

This is because other parts of code might have more information or need to act accordingly.
throws
To show that your method might throw an exception, you put `throws` in the method head.

```java
public int divide(int a, int b) throws ArithmeticException {
    return a/b;
}
```

This allows you to put the try and catch blocks not inside “divide” but where the method was called from.

(See: OldTryCatch.java and Throws.java)
If an exception is thrown, the method will terminate

You can chain throws in this manner through multiple methods

If all the methods (including main()) chain the throw, you will crash with angry red text

(See: ChainThrow.java)
In addition to `try` and `catch` blocks, you can also put a `finally` block after the catch block.

```java
catch (ArithmeticException e)
{
    System.out.println("Error divide by zero");
}
finally
{
    System.out.println("I will always run");
}
```
The code inside `finally` is always run (exception or not)

Is this different than code after the `try` block?

Answer: the only difference happens when an exception happens that isn't caught
- `finally` code will run
- normal code will not run

(See: Finally.java)
Exceptions are classes

Every file you have made has been a class

It should come as no surprise that Exceptions are simply a type of class as well (they simply store some information)

Similar to classes, you might need to import an exception in order to be able to use it

(See: ImportException.java)
Exceptions in inheritance

When you override methods that you inherited, you can catch on less exceptions *(more restrictive)*

Remember child's overridden method must be able to “fit into” the parent's method

This means you...

1. Remove exceptions
2. Replace exceptions by a child (descendant)

(See: InheritException and InheritExceptionParent.java)
Throw non-exceptions

You can actually throw more than exceptions (although normally only exceptions are used)

(See: CatchError.java)
Checked vs. Unchecked

Checked exceptions need to either..
1. Have a try and catch block (or the code will not compile)
2. Throw the exception

Unchecked exceptions...
1. Do not need to be thrown (but still can be caught)
2. Will compile without being caught
(See: UncheckedExceptions.java)
Checked vs. Unchecked

= Checked exceptions
= Unchecked
Exceptions

Exceptions are our first encounter with event-driven programming.

When an exception happens, we can bounce around to different places in the code at runtime based on what the user inputs.

We will see more event-driven programming in shortly...
Nested try-catch

It is possible to nest try-catch, but it is generally frowned upon.

You should either make a method for the inside try-catch, or simply catch at the outside (remove inside try).