Exceptions

Exceptions are:

- Flexible way of dealing with errors
- Separate out error-handling code
- Propagation up the call stack Grouping types of errors

Exceptions are like errors. Instead of hardcoding an error into the function, we can "throw" an exception in which we can assign another block of code to deal with it. An example of throwing an exception is

```
Public void setNumLeg(int newNumLegs){
        if (newNumLegs < 0) {
                throw new IllegalArgumentException();
        }
        else {
                this.numLegs = newNumLegs;
        }
```

Catching exceptions

With functions that have exceptions, we use try blocks in our code. In the following block of code, the program will try to execute the block of code in the try block and, if it "catches" an exception, execute the block of code in the catch block, with the exception as an object e.

```
Canine fido = new Canine();
try{
        code
}
catch (Exception e){
        code
}
```

Multiple catch clauses

When you have multiple catch clauses in your try block, you may want to create multiple catch conditions, starting with the **MOST SPECIFIC**. This is because Java will execute the first block that meets its conditon. For example:

```
catch (IOexception){
        System.out.println("File not found.");
}
catch (Exception e){
        System.out.println("Exception that is not IOexception.");
```

In the code block above, when Java encounters an exception, it will proceed to the exception catch blocks, checking the topmost condition and executing **ONLY THE FIRST** catch block that fulfills it's condition. In this case, IOexceptions will print "File not found." Any other exceptions will print "Exception that is not IOexception."

After the catch block

After the catch block, the program will continue on the program, **NOT the try block that caught the exception**. To create a block of code that will execute regardless of whether an exception is caught or not, use the `finally` block.

```
try {
        code that will run until it finishes or an exception is caught
}
catch (Exception e){
        code that will run if an exception is caught
}
```

```
finally {
        code that will execute regardless of caught exception or not
}
```

In summary, if the try block encounters an exception, it will leave the try block, execute the catch block that corresponds to it's exception, and then move on. The finally block is code that will execute after the try or catch block.

Debugging

Exceptions are great in debugging. We can use `System.out.println(e.getMessage())` to print the message to the system. However, we must associate a message with the exception before we use this. You can do this by using a string argument inside the exception like so: `throw new Exception("Get good")`.

What happens when I don't catch an exception

The program will just terminate if we don't catch exceptions to handle them because the program does not know how to deal with the exception thrown. It will return an error.