Writing Exceptions to the Log

Not the Right Way

This is not a good way to handle an exception:

```
} catch(Exception e) {
}
```

An exception occurred, but we ignored it. Don't do this. Please.

This isn't very good either (although, to be fair, it is better than a kick in the head):

```
} catch(Exception e) {
  e.printStackTrace();
}
```

In Vivo/Vitro the stack trace is printed to catalina.out instead of vivo.all.log. In the Vivo Harvester it is printed to standard out (System.out). It has no timestamp and no source information, so we can't correlate it with other messages in the log. Were any other messages produced by the same request? We'll never know.

Declaring a Logger

In Vivo and Vitro, we use Apache Commons Logging. Create a logger in your Java code with a couple of imports and a static variable:

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class MyClass {
    private static final Log log = LogFactory.getLog(MyClass.class);
    ...
```

In the Vivo Harvester, we use Simple Logging Facade 4 Java. Create a logger in your Java code much like ACL:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyClass {
    private static Logger log = LoggerFactory.getLogger(MyClass.class);
    ...
```

Bad, Better, Good

So, if this isn't good, how can we improve on it?

```
} catch(Exception e) {
}
```

This is better. We're still ignoring it, but we could stop ignoring it just by raising the logging level:

```
} catch(Exception e) {
  log.debug(e, e);
}
```

This is better still. Here is a clue as to why we're ignoring the exception.

```
} catch(Exception e) {
  // This happens if the model data is bad - it's not important
  log.debug(e, e);
}
```

What if we do want to write the exception to the log? What's the right way to do it?

Not like this, for reasons mentioned earlier:

```
} catch(Exception e) {
  e.printStackTrace();
}
```

This is better:

```
} catch(Exception e) {
  log.error(e, e);
}
```

If you have an idea of why a certain exception might be occurring, this would be the best:

```
} catch(IllegalStateException e) {
  log.error("One of the flay-rods has gone out of skew.", e);
} catch(Exception e) {
  log.error(e, e);
}
```

But alas, sometimes no useful message occurs to us.

Whoops

Unlike some other logging frameworks (Log4J, for example) Apache Commons Logging won't check to see whether your first argument is an exception. Instead, it just converts it to a String and prints it to the log.

So, this probably doesn't do what you wanted:

```
} catch(Exception e) {
  log.error(e);
}
```

It logs the class of the exception, and the message in the exception, but it doesn't write the stack trace. That's why this is better:

```
} catch(Exception e) {
  log.error(e, e);
}
```

This way, the Exception class and it's message are written to the log twice, but that's a small price to pay – at least you get the stack trace in the log as well.

And this is best:

```
  } catch(ExpectedTypeAException e) {
    log.error("Some informative message explaining why TypeA might occur", e);
  } catch(ExpectedTypeBException e) {
    log.error("Some informative message explaining why TypeB might occur", e);
  } catch(Exception e) {
    log.error("Some informative message explaining that an unexpected error occurred", e);
  }
```

Because you get to provide more information, you don't write anything twice, and you do get the stack trace.