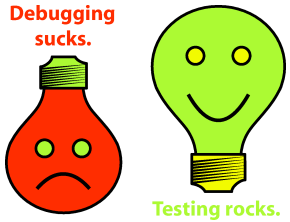

Testing on the Toilet

Defeat "Static Cling"

June 26, 2008

You're pair programming and, as many brilliant people are apt to do, talking out loud. "I'll make a mock, inject it, and rerun the test. It should pa- ...D'OH" Your partner notices the exception "*ConnectionFactory not initialized*". "What?" she says, "Something is using the database? Dang, and this was supposed to be a small test."

Upon inspection you find that your class is calling a **static** method on some other class. You've got **Static Cling**! If you're (ab)using a data persistence layer that generates code which relies on static methods, *and weren't careful*, your code might look something like this:

```
public class MyObject {
  public int doSomething(int id) {
    return TheirEntity.selectById(id).getSomething();
  }
```

As a result, you can't call doSomething without calling **`TheirEntity`**'s static method. This code is **hard to test** because static methods are **impossible to mock** in Java.

So, how do you get rid of this form of Static Cling and get that small test to pass? You can use a technique sometimes known as the **Repository Pattern,** a form of Abstract Factory. Create an interface and an implementation with the unmockable static method calls:

```
interface TheirEntityRepository {
  TheirEntity selectById(int id);
  // other static methods on TheirEntity can be represented here too
}
public class TheirEntityStaticRepository implements TheirEntityRepository {
  public TheirEntity selectById(int id) { // method not static
    return TheirEntity.selectById(id);    // calls static method
  }
```

Next, inject a **`TheirEntityRepository`** into **`MyObject`** and use it instead of calls to the static method:

```
public class MyObject {
  private final TheirEntityRepository repository;
  public MyEntity(TheirEntityRepository arg) { this.repository = arg; }

  public int doSomething(int id) {
    return repository.selectById(id).getSomething();
  }
```

You can do this even if you don't have access to source code for **`TheirEntity`**, since you're not changing the source itself, but merely encapsulating its static methods in an injectable interface. The techniques shown here generalize to the case where a static method acts as a Factory of objects.

Now you can inject different implementations of the repository for different tests, such as "never finds anything," "always throws an exception," "only returns a **`TheirEntity`** if the id is a prime," and so forth. These kinds of tests would've been impossible before this refactoring.

More information, discussion, and archives:
http://googletesting.blogspot.com