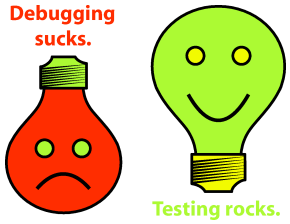


Testing on the Toilet

# Using Dependancy Injection to Avoid Singletons

May 15, 2008

**It's hard to test code that uses singletons.** Typically, the code you want to test is coupled strongly with the singleton instance. You can't control the creation of the singleton object because often it is created in a static initializer or static method. As a result, you also can't mock out the behavior of that Singleton instance.

If changing the implementation of a singleton class is not an option, but changing the **client** of a singleton is, a simple refactoring can make it easier to test. Let's say you had a method that uses a `Server` as a singleton instance:

```
public class Client {
  public int process(Params params) {
    Server server = Server.getInstance();
    Data data = server.retrieveData(params);
    ...
  }
}
```

You can refactor `Client` to use **Dependency Injection** and avoid its use of the singleton pattern altogether. You have not lost any functionality, and have also not lost the requirement that only a singleton instance of `Server` must exist. The only difference is that instead of getting the `Server` instance from the static `getInstance` method, `Client` receives it in its constructor. You have made the class easier to test!

```
public class Client {
  private final Server server;

  public Client(Server server) {
    this.server = server;
  }

  public int process(Params params) {
    Data data = this.server.retrieveData(params);
    ...
  }
}
```

When testing, you can create a mock `Server` with whatever expected behavior you need and pass it into the `Client` under test:

```
  public void testProcess() {
    Server mockServer = createMock(Server.class);
    Client c = new Client(mockServer);
    assertEquals(5, c.process(params));
  }
```

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