Testing

Towards THE bug free program

Edsger W. Dijkstra

Program testing can be used to show the presence of bug, but never to show their absence!!

I don't make
mistak

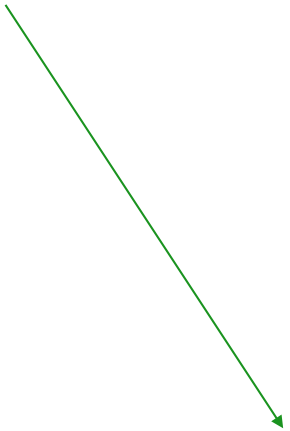
http://www.dailymail.co.uk/sciencetech/article-2531601/Windows-error-messages-let-NSA-spy-people-Crash-reports-neat-way-gaining-access-machines-claims-report.html

Why create a test suite?

- Obviously you have to test your code—right?
  - You can do ***ad hoc*** testing (running whatever tests occur to you at the moment), or
  - You can build a test suite (a thorough set of tests that can be run at any time)
- Disadvantages of a test suite
  - It's a lot of extra programming
    - True, but use of a good test framework can help quite a bit
  - You don't have time to do all that extra work
    - ***False!*** Experiments repeatedly show that test suites reduce debugging time more than the amount spent building the test suite
- Advantages of a test suite
  - Reduces total number of bugs in delivered code
  - Makes code much more maintainable and refactorable

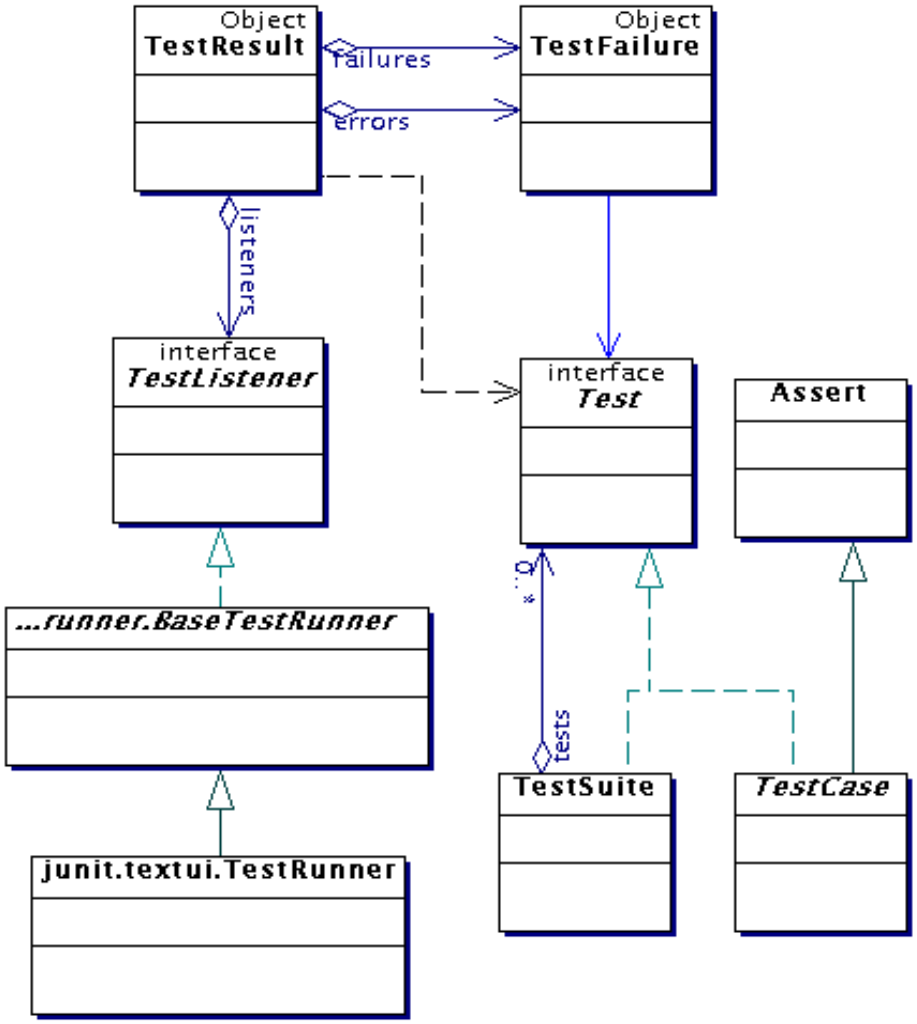
Software development enclosing the real world

- J-Unit
  - Test developing
  - Multiple tools
- Maven
  - Dependency management



Architectural overview

- JUnit test framework is a package of classes that lets you write tests for each method, then easily run thos tests
- **TestRunner** runs tests and reports **TestResult**s
- You test your class by extending abstract class ***TestCase***
- To write test cases, you nee to know and understand the **Assert** class



Writing a TestCase

- To start using JUnit, create a subclass of *TestCase*, to which you add test methods
- Here's a skeletal test class:

```
import org.junit.Test;
…
@Test
public class TestBowl {
} //Test my class Bowl
```

Writing methods in TestCase

- Pattern follows ***programming by contract*** paradigm:
  - Set up **preconditions**
  - Exercise functionality being tested
  - Check **postconditions**
- Example:

```
public void testEmptyList() {
        Bowl emptyBowl = new Bowl();
        assertEquals("Size of an empty list should be zero.",
              0, emptyList.size());
        assertTrue("An empty bowl should report empty.",
emptyBowl.isEmpty());
}
```

- Things to notice:
  - Specific method signature – public void ***test***Whatever()
    - Allows them to be found and collected automatically by JUnit
  - Coding follows pattern
  - Notice the assert-type calls…

Assert methods

- assertTrue(String ***message***, Boolean ***test***)
- assertFalse(String ***message***, Boolean ***test***)
- assertNull(String ***message***, Object ***object***)
- assertNotNull(String ***message***, Object ***object***)
- assertEquals(String ***message***, Object ***expected***, Object ***actual***) (uses equals method)
- assertSame(String ***message***, Object ***expected***, Object ***actual***) (uses == operator)
- assertNotSame(String ***message***, Object ***expected***, Object ***actual***)

Ok so dependencies

Project Dependency

- Libraries needed for build and runtime.

Project Reports

- Junit reports
- Javadoc reports
- Checkstyle reports, ….etc

Take a look to pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.0</version>
      <type>jar</type>
      <scope>test</scope>
      <optional>true</optional>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```