Unit Testing of Java Applications for BlackBerry

Danny Preussler,
Development Team Lead, Cortado. A Division of Thinprint

B09

BlackBerry DEVELOPER CONFERENCE

08 | BUILDING FOR SUCCESS

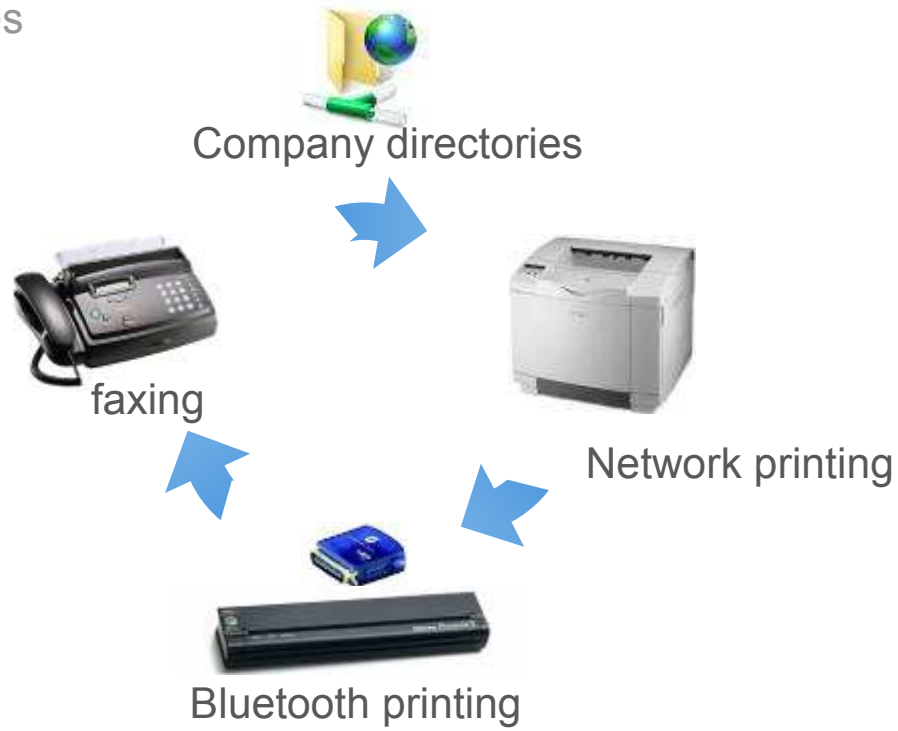
Who‘s speaking?

BlackBerry
DEVELOPER CONFERENCE
08 | BUILDING FOR SUCCESS

- Writes professional Java applications for 8 years
- Sun certified Java programmer
- Develops for the BlackBerry platform since 2004
  for Cortado, a Division of ThinPrint, a BlackBerry Alliance member

Who‘s speaking?

- Client-Server business applications for
  - more than 9.000 customers world wide and
  - more than 500 partners in 47 countries
- Rely on well tested software
- Guaranteed by:
  - Quality Assurance department (BlackBox testing)
  - Research and Development (WhiteBox testing)



Unit-testing

- Unit testing common for years in professional Java projects
- Running JSE and JEE projects without efficient unit tests hard to believe today
- Testing frameworks available for nearly every programming language
- in JME environments still uncommon

Why should I unit test?

- Ensure a high degree of application quality
- Reduce testing, support and porting costs
- Testing early → helps finding and fixing problems early

- Prevent long debugging nights!

Unit-testing

Why uncommon in JME projects?

- Limitations of mobile phone APIs
- Complexity of phone environments → hard to test

But:

- Approaches from open source / phone manufacturers: provide tools and frameworks for mobile world

Unit-Testing – The JUNIT way

- The “Original”
- Unit testing introduced by Kent Beck for Smalltalk: SUnit
- With Erich Gamma 1999 ported to Java: JUnit
- Based on Java Reflection
- Supported in common Java IDEs
- Various extensions and frameworks available

Unit-Testing – The JUNIT way

BlackBerry
DEVELOPER CONFERENCE
08 | BUILDING FOR SUCCESS

- HowTo?
  - Write a „test“ class for every class to test
  - Extend from „`junit.framework.TestCase`“
  - All test methods must be „`public void`“ and start with „`test`“

```
assertFalse(...)                    fail(...)
    assertNotSame(...)
                              assertTrue(...)
 assertEquals(...)
```

Unit-Testing – The mobile way

- Testing mostly done in simulators, because:
  - mobile-environment must be simulated
  - Reflection not available in JME
- Various projects available:
  - CLDCUnit
  - J2MEUnit
  - BUnit
  - JMUnit

J2MEUnit

BlackBerry
DEVELOPER CONFERENCE
08 | BUILDING FOR SUCCESS

- "TestRunner" for command line / build files
- "MidletRunner" for simulator

- Components:
  - Single 16KB jar file
- License:
  - Open source, Common Public License

J2MEUnit

- HowTo?
  - Inherited from "`j2meunit.framework.TestCase`"
  - Create a method: „`suite()`"
  - Create your test and every test method manually

```
public class MyTest extends TestCase
{
   public Test suite()
   {
      TestSuite aSuite = new TestSuite();
      aSuite.addTest(new MyTest(
          "testOne",
          new TestMethod()
          { public void run(TestCase tc)
            {((MyTest)tc).testDoSomethingThatShouldBeTested(); }
          }));
      return aSuite;
   }
```

→Small, but a lot of overhead

CLDCUnit

- Based on JUnit
- Tests can be run as
  - JUnit tests (reflection) or with
  - “MidletRunner” for simulators (instrumentation)
- Components:
  - Multiple jar files (incl. instrumentation libraries)
  - 600 KB
- License:
  - Open source, Apache License 2.0

CLDCUnit

- HowTo?
  - For JUnit: As always (extend "`TestCase`")
  - For Simulator:
    build a "`cldcunit.runner.TestRunner`":

```
public class MyTestRunner extends TestRunner
{
    protected void startApp()
    {
        start(new TestCase[] {new MyTest3()});
    }

}
```

→ Nearly as easy as JUnit

JMUnit

- Part of NetBeans mobility pack
  (will be merged with J2MUnit)
- Supports JUnit-like test cases and test suites
- Tests can be run on a device/emulator or Ant task
- Supports performance monitoring
- Components:
  - Multiple jar files (incl. hammock framework, jdom…)
  - Includes sources, tests for sources, examples, ant-files
- License:
  - Open source: Apache License 2.0
    + various redistributables (GPL2, GPL3)

JMUnit

- HowTo?
  - Extend from "`jmunit.framework.cldc11.TestCase`"
  - Return number of tests in constructor to parent
  - Build a method to call your tests:

```
public void test(int testNumber) throws Throwable
```

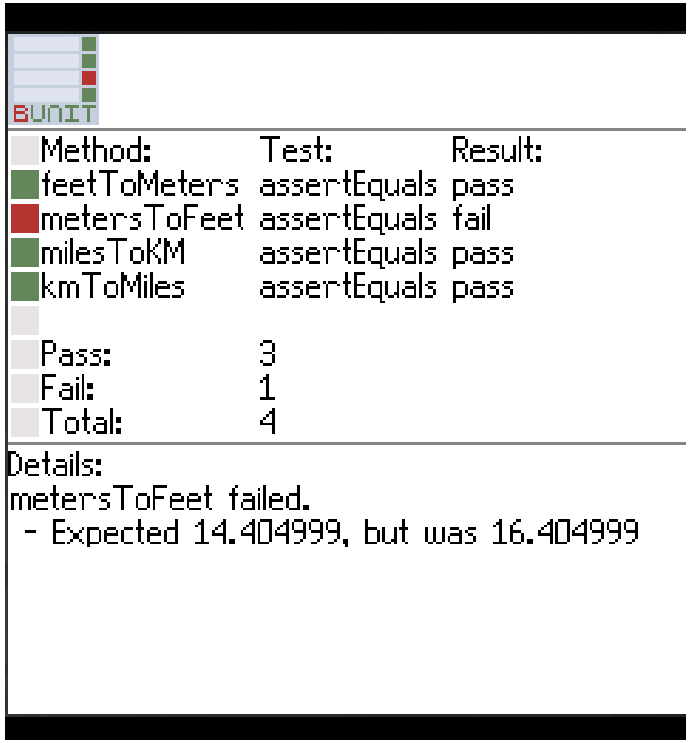
```
public void test(int testNumber) throws Throwable
{
        switch (testNumber)
        {
          case 0:
            testDoSomethingThatShouldBeTested();
            break;
        }
}
```

→ Huge featureset

BUnit

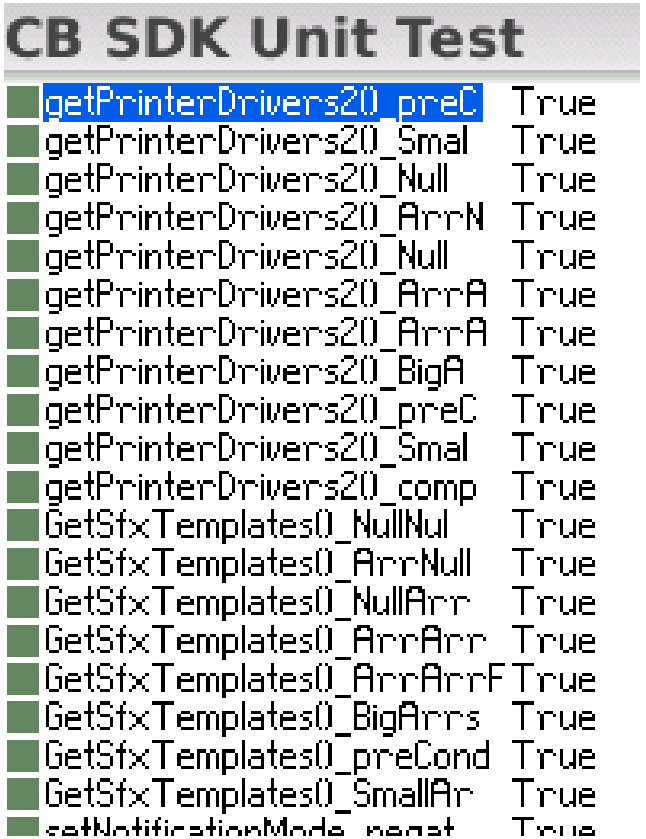
- Originally based on JMUnit, rewritten for BlackBerry
- Provides a RIM-CLDC application for
- Device/Simulator tests
- Project stopped

- Components:
  - As source (<100k)
- License:
  - Open source: Apache License 2.0



BUnit at Cortado

- Cortado SDK allows printing, faxing or corporate document access for 3rd party applications
- Cortado‘s Quality Assurance department uses blackbox tests based on JMUnit/BUnit within BlackBerry simulators for verifying described functionality



BUnit at Cortado


```
private void testPrintableMessage0() throws
  AssertionFailedException
{
…
  Message msg = new Message();
  msg.setSubject("Test");
  msg.setContent("hello");
  Printable iPrintable = createPrintable(msg);
  assertTrue(
     CortadoPrintsystem.print(iPrintable,…));
```

Unit-Testing – The mobile way

- Even more ME testing frameworks available
  - Mobile JUnit
    Sony Ericsson
    https://developer.sonyericsson.com
  - ME Framework for JT harness
    https://cqme.dev.java.net/framework.html

“Runner”s vs Simulatortests

- Pro Runners:
  - Faster
  - Usable in scripted builds (ant, maven), build servers “Continuous integration”
  - Text based output → logfiles; parseable
  - Directly jumping to source on error in IDE
  - Easy to debug
- Pro Simulator:
  - Device-near
  - Can be used on actual device

JUNIT Runner

- Pro JUnit (1):
  - IDE Support:
    - Eclipse, NetBeans…
  - Build engine support
    - Ant, Maven, Hudson…
  - Mocking frameworks
    - EasyMock, JMock, Hammock

JUNIT Runner

- Pro JUnit (2):
  - Coverage Frameworks:
    - Coverlipse, Coverage, EMMA, …

```
import javax.wireless.messaging.TextMessage;

public class MyTest
{
    public boolean doSomethingThatShouldBeTested(TextMessage itsAMessage)
    {
        if (itsAMessage == null)
        {
            throw new IllegalArgumentException("null message not supported");
        }

        String message = itsAMessage.getPayloadText();
        if ((message == null) || (message.length() == 0))
        {
            return false;
        }
```

JUnit and Mobile
How to test outside the simulator

BlackBerry DEVELOPER CONFERENCE
08 | BUILDING FOR SUCCESS

- The problem: APIs often declared native

- **java.lang.ClassFormatError**: Method <init> in class javax/microedition/lcdui/Alert has illegal modifiers: 0x101
- at java.lang.ClassLoader.defineClass1(Native Method)
- at java.lang.ClassLoader.defineClass(Unknown Source)

- **java.lang.UnsatisfiedLinkError**: java.lang.UnsatisfiedLinkError: javax.microedition.lcdui.Font.init(III)V
- at javax.microedition.lcdui.Font.init(Native Method)

- **Best practise I:** Use common design principles
  - Separate UI from business logic
  - Use interfaces
  - Dependency Injection ....
  - Use Stubs / Mocking frameworks

- **Best practise II:** Separate your dependencies!
  - RIM API consists of 2 parts:
    - JME APIs:
      - CDLC (`java.lang…`)
      - MIDP (`javax.microedition`)
      - JSRs (i.e. `javax.bluetooth`,...)
    - RIM-APIs (`net.rim….`)
  - Everything found in one library: net_rim_api.jar

- **Best practise II:** Separate your dependencies!
  - JME part of API available by other vendors:
  - Use Sun Wireless Tookit:
    - midpapi20.jar
    - cldcapi11.jar
    - jsr75.jar…
  - Use M4SE (GPL/Commercial)

- **Best practise III:** Testing connectivity
  - Important difference between JSE and JME:
    The connectivity:
    - Generic Connection Framework (GCF)
      `connector.open(…)`

- **Best practise III:** Testing connectivity
  - GCF uses good design:
    - Completely based on Interfaces:
      `HttpConnection, SocketConnetion…`
      → good to test
    - Every connection starts at "`Connector.open()`"
      → single point of replacement for testing

- **Best practise III:** Testing connectivity
  - Test with your own implementations of Connection-interfaces
  - Use Stubs, Mocks
  - Use existing implementations:
    - Generic Connection Framework for JSE already a JSR request: JSR-197
    - JSR-82 (Bluetooth API) – commercial JSE implementations available
  - Build your own partial GCF-implementations

JUnit and Mobile
How to test outside the simulator

BlackBerry DEVELOPER CONFERENCE
08 | BUILDING FOR SUCCESS

- **Best practise III:** Testing connectivity
  - Build your own partial GCF-implementations

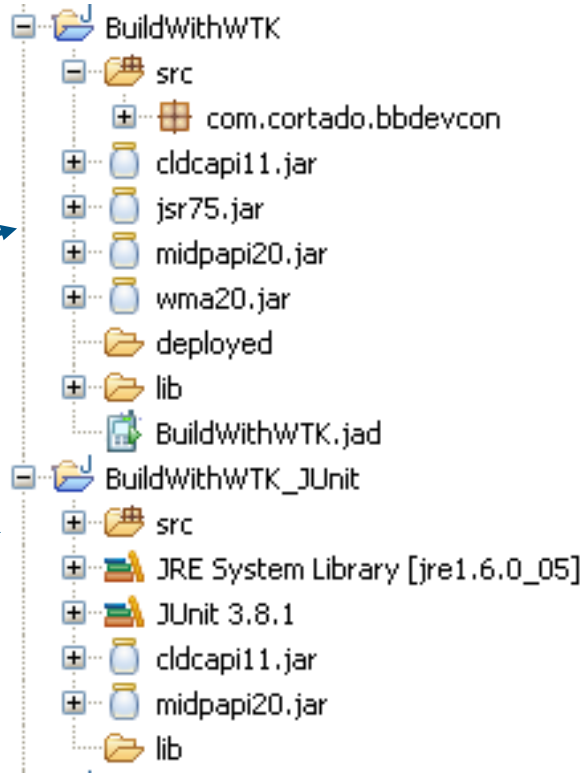
```
package javax.microedition.io;
..
public class Connector
{
  public static Connection open(String strURL) throws IOException
  {
     if (strURL.startsWith("http"))  {
          return new MyHttpImpl(strURL);
     }
..

class MyHttpImpl implements HttpConnection
{
  // TODO implementation via java.net.HttpURLConnection
```

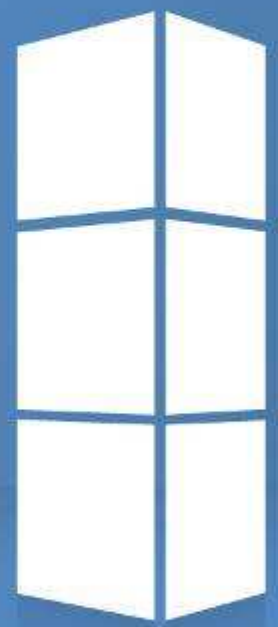
- **Best practise IV:** Testing against RIM classes
  - Use Interfaces:
    - For device implemented with RIM class
    - For test/desktop: implemented with SE class
  - Build your own implementations of needed classes:
    - based on API documentation
      (can be partially scripted, i.e. „Javadoc to Java Convertor")
    - link before *net_rim_api.jar*!
      (on problems with Eclipse: modify *.classpath* file)

JUnit and Mobile
How to test outside the simulator

- **Best practise V**: organising the project
  - Build tests in separate project
  - 2nd project links against 1st one
  - Use correct API in 1st project (*net_rim_api*, ME api)
  - Full modern JRE support in 2nd project (i.e. usage of JUnit4 -> annotations…)



Time for discussions



BlackBerry DEVELOPER CONFERENCE 08 | BUILDING FOR SUCCESS

For more information

BlackBerry
DEVELOPER CONFERENCE
08 | BUILDING FOR SUCCESS

- Testing frameworks:
  - JUnit *http://www.junit.org*
  - J2MEUnit *http://j2meunit.sourceforge.net/*
  - CLDCUnit *http://snapshot.pyx4me.com/pyx4me-cldcunit/*
  - JMUnit *http://sourceforge.net/projects/jmunit/*
  - BUnit *http://sourceforge.net/projects/b-unittesting/*
- Mocks and helpers:
  - Easymock *http://www.easymock.org/*
  - JMock *http://www.jmock.org/*
  - Hammock *http://hammingweight.com/modules/hammock/*
  - ME4SE *http://kobjects.sourceforge.net/me4se/*
  - Javadoc to Java Convertor (jd2j), *http://ejohn.org/projects/jd2j*

Thank You

Danny Preussler,
Development Team Lead, Cortado. A Division of Thinprint

B09

BlackBerry DEVELOPER CONFERENCE

08 | BUILDING FOR SUCCESS