

Executable requirements: BDD with easyb and JDave

John Ferguson Smart
Principal Consultant
Wakaleo Consulting

Lasse Koskela
Consultant
Reaktor



Reaktor

Presentation Goals

Principles and Practices of Behavior-Driven Development
See two Java BDD frameworks in action: EasyB and JDave
See Lasse and John fight it out to see prove which is best

Speaker's qualifications

▶ **John Ferguson Smart**

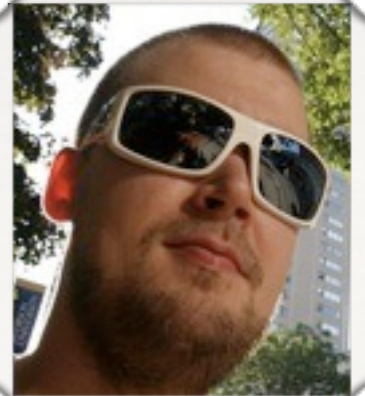
- ▶ Consultant, Trainer, Mentor, Author,...
- ▶ Works with Enterprise Java, Web Development, and Open Source technologies
- ▶ Author of 'Java Power Tools' (O'Reilly)
- ▶ Writes articles for sites like JavaWorld, DevX and Java.net, and blogs on Java.net
- ▶ Frequent speaker at conferences and Java User Groups



Speaker's qualifications

▶ **Lasse Koskela**

- ▶ (Also) Consultant, Trainer, Mentor, Author,...
- ▶ (Also) Works with Enterprise Java, Web, and Open Source technologies
- ▶ Author of 'Test Driven' (Manning)
- ▶ Long-time contributor and moderator at JavaRanch
- ▶ Frequent speaker at conferences like this



Agenda

- ▶ What will we cover today
 - ▶ What is Behavior-Driven Development (BDD)?
 - ▶ Introducing two BDD frameworks for Java: **easyb** and **JDave**
 - ▶ See how they cope with different BDD use cases:
 - ☑ Testing simple requirements
 - ☑ Handling fixtures and refactoring tests
 - ☑ SDLC integration and reporting
 - ☑ IDE integration

Test Driven Development

- ▶ Principles of Test-Driven Development (TDD)
 - ▶ Remember: It's not (just) about writing tests
 - ▶ Design and write better code
 - ▶ Refactor with confidence
 - ▶ Although TDD vocabulary uses the word “test” a lot...

Behavior Driven Development

- ▶ Behavior Driven Development (BDD)
 - ▶ A better vocabulary for teaching TDD
 - ▶ Uses words like “should” to describe the desired behavior, e.g.
 - ▶ “shouldDoThis”
 - ▶ “shouldBeThat”

Behavior Driven Development

- ▶ Behavior Driven Development (BDD)
 - ▶ Two main styles:
 - ▶ “TDD done right”
 - ▶ “Executable requirements”

BDD using plain old JUnit

- ▶ Can you do BDD with ordinary JUnit tests?
- ▶ Yes!
- ▶ Is it ideal? Not really...

Lots of “boiler-plate” code

```
package org.ebank.domain;
```

```
import org.junit.Test;  
import static org.junit.Assert.assertTrue;  
import java.math.BigDecimal;
```

```
public class AccountTest {
```

```
    @Test
```

```
    public void balanceShouldBeEqualToInitialDepositAfterInitialBalance() {
```

```
        Account account = new Account();
```

```
        BigDecimal initialAmount = new BigDecimal("100");
```

```
        account.makeDeposit(initialAmount);
```

```
        assertTrue(account.getBalance().equals(initialAmount));
```

```
    }
```

Long method names

Cumbersome asserts

Can do better?

BDD Frameworks for Java

- ▶ Use a BDD testing framework
 - ▶ Make testing clearer and easier to write
 - ▶ Make tests self-documenting
 - ▶ Help developers focus on the requirements

Introducing our BDD frameworks

▶ Choices, choices...

▶ **Easyb**

- ▶ Groovy-based BDD testing framework
- ▶ Uses a Groovy DSL to express BDD specifications
- ▶ Also works with Groovy and Grails
- ▶ “Executable Requirements”



▶ **JDave**

- ▶ Java-based BDD testing framework
- ▶ Extends JUnit, yielding excellent IDE support
- ▶ Tight integration into mock objects
- ▶ “TDD Done Right”™



Introducing Easyb

- ▶ Easyb use a narrative approach:
 - ▶ Stories describe a precise requirement
 - ▶ They can (usually) be understood by a stakeholder
 - ▶ A story contains a set of scenarios
 - ▶ Scenarios use an easy-to-understand structure:
 - ▶ Given [a context]...
 - ▶ When [something happens]...
 - ▶ Then [something else happens]...



BDD in Action

- ▶ A sample application - online banking
- ▶ Start by defining the behavior!



User Story 1 - Opening a bank account

As a customer, I want to open a bank account so that I can put my money in a safe place.

User Story 2 - withdraw money

As a customer, I want to open a bank account so that I can put my money in a safe place.

Task

Open new account

Task

Make initial deposit

Task

Withdraw money

BDD in Action

- ▶ A sample application - online banking
- ▶ Tasks are at the heart of our BDD stories



Task - Make initial deposit onto a new account

given "a newly created account"

when "an initial deposit is made into this account"

then "the account balance should be equal to the amount deposited".

Task
Open new
account

Task
Make initial
deposit

Task
Withdraw
money

Easyb in Action

- ▶ Writing an easyb story
- ▶ Start with the task...making an initial deposit



Task – Make initial deposit onto a new account

given "a newly created account"

when "an initial deposit is made into this account"

then "the account balance should be equal to the amount deposited".

```
scenario "Make initial deposit onto a new account", {  
  given "a newly created account"  
  when "an initial deposit is made into this account"  
  then "the account balance should be equal to the amount deposited"  
}
```

A valid easyb scenario!

Easyb in Action

▶ Anatomy of an easyb story

```
scenario "Make initial deposit onto a new account", {  
  given "a newly created account"  
  when "an initial deposit is made into this account"  
  then "the account balance should be equal to the amount deposited"  
}
```

- ▶ **“Scenario”**: corresponds to precise requirement
- ▶ **“Given”**: the context in which this requirement applies
- ▶ **“When”**: An event or action
- ▶ **“Then”**: The expected results of this action



-- bdd in java can't get any easier

Easyb in Action

- ▶ Implementing the scenario
 - ▶ Tests are written in Groovy
 - ▶ Can use all Java classes and APIs



```
package org.ebank.domain

scenario "Make initial deposit onto a new account", {
  given "a new account", {
    account = new Account()
  }
  when "an initial deposit is made", {
    initialAmount = 100
    account.makeDeposit(initialAmount)
  }
  then "the balance should be equal to the amount deposited", {
    account.balance.shouldBe initialAmount
  }
}
```

Automatic handling of
BigDecimal

Readable assertions

Introducing JDave



- ▶ Pure JUnit 4 extension
- ▶ JDave uses a structural approach:
 - ▶ Inner classes represent scenarios we call “contexts”
 - ▶ One test class often contains a set of contexts
 - ▶ Programmers can use inheritance to describe similar contexts concisely
- ▶ The structure forms a narrative of specified behavior:
 - ▶ (class) Specifying Http Caching...
 - ▶ (context) When Incoming Request Is For Static Content...
 - ▶ (specification) ...Caching Headers Are Set

JDave in Action

- ▶ Implementing the bank deposit scenario
- ▶ Tests are written in pure Java
- ▶ Can use all Java APIs and language features



```
package org.ebank.domain

@RunWith(JDaveRunner.class)
public class AccountSpec extends Specification<Account> {
    public class WhenInitialDepositIsMade {
        public Account create() {
            Account account = new Account();
            account.makeDeposit(100);
            return account;
        }

        public void balanceShouldBeEqualToAmountDeposited() {
            specify(context.balance(), should.equal(100));
        }
    }
}
```

The "fixture" or "context" is a first class concept in JDave

Readable assertions

Handling exception cases

- ▶ Let's look at a more complicated example...
- ▶ Withdrawals:

Task - Withdraw money from an account

given "an account with a certain balance"
when "a sum is withdrawn from the account"
then "the withdrawn sum is deducted from the balance".

Task - Withdraw too much money

given "an account with a certain balance"
when "a sum is withdrawn from the account"
then "an error is raised"
and "the balance remains unchanged"

Task

Open new account

Task

Make initial deposit

Task

Withdraw money

Easyb in Action again

▶ Let's look at a more complicated example...

▶ Withdrawals:



Task - Withdraw money from an account

given "an account with a certain balance"

when "a sum is withdrawn from the account"

then "the withdrawn sum is deducted from the account balance".

```
scenario "Withdraw money from an account", {  
  given "an account with a certain balance", {  
    initialBalance = 100  
    account = new Account()  
    account.balance = initialBalance  
  }  
  when "a sum is withdrawn from the account", {  
    amountWithdrawn = 20  
    account.withdraw(amountWithdrawn)  
  }  
  then "the withdrawn sum is deducted from the account balance", {  
    account.balance.shouldBe initialBalance - amountWithdrawn  
  }  
}
```

Easyb in Action again

▶ Let's look at a more complicated example...

▶ Withdrawals:



Task - Withdraw too much money

given "an a
when "a sur
then "an err
and "the bal

```
scenario "Withdraw more money than there is in the account", {  
  given "an account with a certain balance", {  
    initialBalance = 100  
    account = new Account()  
    account.balance = initialBalance  
  }  
  and "an amount is withdrawn that is greater than the balance", {  
    withdrawTooMuchMoney = {  
      account.withdraw(150)  
    }  
  }  
  then "an InsufficientFundsException is raised", {  
    ensureThrows(InsufficientFundsException.class) {  
      withdrawTooMuchMoney()  
    }  
  }  
  and "the account balance remains unchanged", {  
    account.balance.shouldBe initialBalance  
  }  
}
```

Use a closure to encapsulate the action

Check that an exception is thrown

And ensure that the balance is unchanged

JDave in Action again

- ▶ JDave has built-in support for specifying that an exception should be thrown



```
public class AccountSpec extends Specification<Account> {
    public class WhenWithdrawingMoreThanThereIsInTheAccount {
        int initialBalance = 100;
        Block that;

        public Account create() {
            final Account account = new Account(initialBalance = 100);
            that = new Block() {
                public void run() throws Throwable {
                    account.withdraw(initialBalance + 1);
                }
            };
            return account;
        }

        public void anInsufficientFundsExceptionIsThrown() {
            specify(that, should.raise(InsufficientFundsException.class));
        }

        public void theAccountBalanceRemainsUnchanged() {
            specify(after(that).balance(), should.equal(initialBalance));
        }
    }
}
```

Use a "block" (closure) to encapsulate the action

Check that an exception is thrown

And ensure that the balance is unchanged

Easyb assertions

- ▶ Tools for expressing behavior in easyb
 - ▶ The **shouldBe** syntax:
 - ▶ Intuitive, readable and flexible

```
account.balance.shouldBe initialAmount
```

```
account.balance.shouldBeEqualTo initialAmount
```

```
account.balance.shouldNotBe 0
```

```
account.balance.shouldBeGreaterThan 0
```

```
account.shouldHave(balance:initialAmount)
```



JDave assertions



- ▶ Fluent API for specifying desired behavior
- ▶ The **specify(x , should.*)** syntax:
 - ▶ Intuitive, readable and flexible

```
specify(context.balance(), should.equal(100));
```

```
specify(new Block() {}, should.raise(TypeErrorException.class));
```

```
specify(returnedObject, should.be == certainInstance);
```

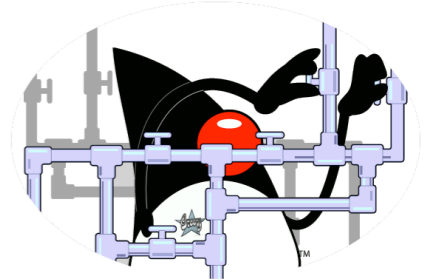
```
specify(aCollection, should.contain(someObject, anotherObject));
```

```
specify(aCollection, must.not().contain(thatOtherObject));
```

Some folks might prefer "must" over "should"

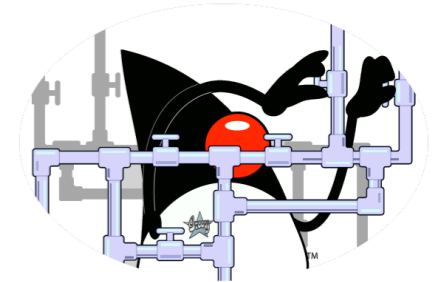
Organizing your tests

- ▶ More complex test may need to be refactored:
 - ▶ Get infrastructure code out of the test cases
 - ▶ Test cases are more readable and understandable
 - ▶ Test cases are easier to maintain
- ▶ JUnit has fixtures:
 - ▶ `@Before/@After` and `@BeforeClass/@AfterClass` in JUnit 4.x
 - ▶ `setup()` and `teardown()` in JUnit 3.x
- ▶ What about Easyb and JDave?



Easyb fixtures

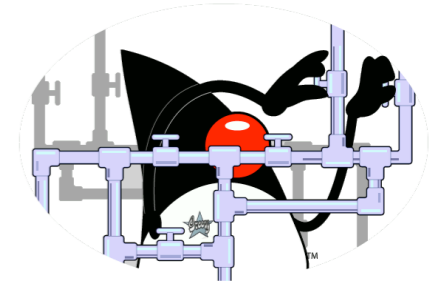
- ▶ Structuring your tests...fixtures in easyb
- ▶ Setting up the test environment...
 - ▶ **before** is run at the start of the whole story
 - ▶ **before_each** is run before each scenario
 - ▶ Useful for setting up databases, test servers, etc.



```
before_each "initialize an instance of selenium", {  
  given "selenium is up and running", {  
    selenium = new DefaultSelenium("localhost", 4444,  
      "*firefox", "http://testserver.wakaleo.com/onlinebank")  
    selenium.start()  
  }  
  and "given a person is at the home page", {  
    selenium.open("http://testserver.wakaleo.com/onlinebank/index.html")  
  }  
}
```

Easyb fixtures

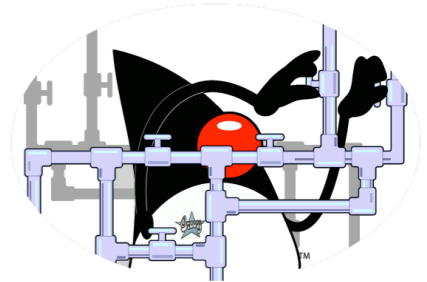
- ▶ Structuring your tests...fixtures in easyb
- ▶ Cleaning up afterwards
 - ▶ **after** is run at the end of the whole story
 - ▶ **after_each** is run after each scenario
 - ▶ Useful for shutting down test servers, cleaning up databases etc.



```
after_each "shut down selenium", {  
  then "selenium should be shutdown", {  
    selenium.stop()  
  }  
}
```

JDave fixtures

- ▶ Structuring your tests...fixtures in JDave
- ▶ Setting up and tearing down the test environment...
 - ▶ **create()** is run before each scenario (*context* in JDave-speak)
 - ▶ **destroy()** is run after each *context*
 - ▶ Useful for setting active locale, clearing caches, etc.



```
public class AccountSpec {
    Locale previousLocale;

    @Override
    public void create(
        previousLocale =
        Locale.getDefault()
    )

    @Override
    public void destroy(
        Locale.getDefault()
    )

    public class WhenRegistered {
}

public class StackSpec extends Specification<Stack> {
    public class EmptyStack {
        public Stack create() {
            return new Stack();
        }

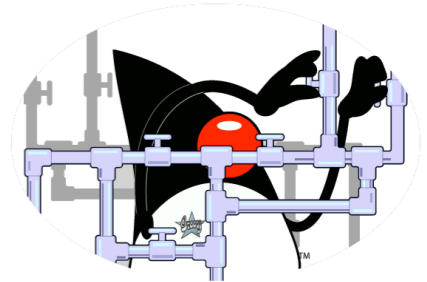
        public void destroy() {
            // we could tear down stuff here if we wanted to
        }

        public void isEmpty() {
            specify(context.isEmpty());
        }
    }
}

}
```

JDave fixtures

- ▶ Structuring your tests...fixtures in JDave
- ▶ One-time setup and tear-down
 - ▶ not implemented at the moment
 - ▶ need to use `create()` and a shutdown hook for, e.g. starting up and shutting down a fake SMTP server, etc.



```
public abstract class IntegrationSpec<T> extends Specification<T> {
    Wisier smtpServer;

    @Override
    public void create() {
        if (smtpServer == null) {
            smtpServer = new Wisier();
            smtpServer.start();
            Runtime.getRuntime().addShutdownHook(new Runnable() {
                public void run() {
                    smtpServer.stop();
                }
            });
        }
    }
}
```

Note to self (Lasse):
Make sure I'm up-to-date
with how latest JDave deals
with `create()`

Web testing with Easyb

- ▶ Many options - let's look at two
 - ▶ Selenium
 - ▶ Runs in a browser
 - ▶ High-level API
 - ▶ Runs slower and more work to set up
 - ▶ HtmlUnit
 - ▶ Simulates a browser
 - ▶ Runs faster, easy to set up
 - ▶ API slightly lower level

Web testing with Easyb

▶ Writing functional tests with HtmlUnit

Task - Deposit cash

Set up HtmlUnit web client

Display the home page

Check the initial balance

Enter a value in a text field

Click a button

Check result page

```
import com.gargoylesoftware.htmlunit.WebClient
import com.gargoylesoftware.htmlunit.html.*

before_each "initialize a web client", {
  given "a web client has been created", {
    webClient = new WebClient()

scenario "Deposit cash via the web interface", {
  given "the application home page is displayed", {
    page = webClient.getPage("http://localhost:8080/ebank-web")
  }
  and "the current balance is 0", {
    page.asText().contains "Current Balance: \$0"
  }
  when "the user enters a value in the 'deposit cash' field", {
    form = page.forms[0];
    depositButton = form.getInputByName("deposit");
    textField = form.getInputByName("depositAmount");
    textField.valueAttribute = "100";

    resultPage = depositButton.click();

  then "the balance should be equal to the amount deposited", {
    resultPage.asText().contains "Current Balance: \$100"
  }
}
```

Web testing with Easyb

▶ Writing functional tests with Selenium

```
import com.thoughtworks.selenium.DefaultSelenium;  
import com.thoughtworks.selenium.Selenium;  
  
before_each "initialize a web client", {  
  given "selenium is up and running", {  
    selenium = new DefaultSelenium("localhost", 4444,  
                                   "*firefox", "http://localhost:8080");  
    selenium.start();  
  }  
}  
  
after "stop selenium" , {  
  then "selenium should be shutdown", {  
    selenium.stop()  
  }  
}  
...
```

Set up a Selenium client

Shut down afterwards

Web testing with Easyb

▶ Writing functional tests with Selenium

```
import com.thoughtworks.selenium.DefaultSelenium;  
import com.thoughtworks.selenium.Selenium;
```

```
before_each "initialize a web client", {  
  given "selenium is up and running", {  
    selenium = new DefaultSelenium("localhost", 4444,
```

```
  }  
  scenario "Deposit cash via the web interface", {  
    given "the application home page is displayed", {  
      selenium.open("/ebank-web");  
      selenium.waitForPageToLoad("2000");  
    }  
    and "the current balance is 0", {  
      selenium.isTextPresent("Current balance: \">$100");  
    }  
    when "the user enters a value in the 'deposit cash' field", {  
      selenium.type("depositAmount", "100")  
      selenium.click("deposit")  
      selenium.waitForPageToLoad("2000")  
    }  
    then "the balance should be equal to the amount deposited", {  
      selenium.isTextPresent("Current balance: \">$100");  
    }  
  }  
}
```

Open home page

Check initial balance

Type a value

Click a button

Check result

Web testing with JDave

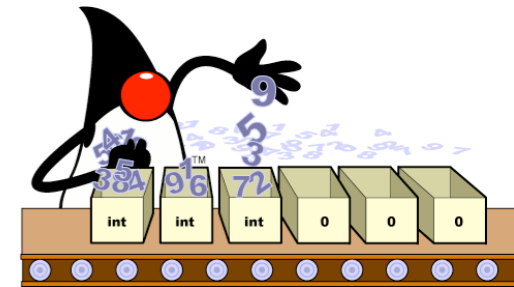
- ▶ JDave was never designed for functional testing of web applications, but...
- ▶ Remember the create/shutdown hook trick?
- ▶ It's not that difficult to create your own testing API for web testing

```
public class FunctionalTest extends WebTestingSpecification {  
    public class WhenDepositingCashViaTheWebInterface {  
        given.user().isAt("/ebank-web");  
        given.page().contains("Current balance: $100");  
        when.user().enters("100").to("depositAmount");  
        and.user().clicks("deposit");  
        then.page().contains("Current balance: \">$100");  
    }  
}
```

...but you have to do it yourself

BDD and the SDLC

- ▶ A test framework should blend into your build lifecycle
- ▶ Test automation
- ▶ Integrate with Ant and Maven
- ▶ Play nicely with Continuous Integration tools



Easyb in the SDLC

▶ Integrating easyb with Ant

▶ Use the easyb Ant task

The Ant easyb task



```
<taskdef name="easyb" classname="org.easyb.ant.BehaviorRunnerTask">
  <classpath>
    <pathelement location="${lib.dir}/easyb-0.9.5.2.jar"/>
    <pathelement location="${lib.dir}/groovy-all-1.6.3.jar"/>
  </classpath>
</taskdef>
...
<target name="easyb">
  <easyb>
    <classpath>
      <pathelement location="${lib.dir}/easyb-0.9.5.2.jar"/>
      <pathelement location="${lib.dir}/groovy-all-1.6.3.jar"/>
      <pathelement location="${lib.dir}/commons-csv-1.0.jar"/>
      <pathelement path="target/classes" />
    </classpath>
    <report location="target/stories.html" format="html" />
    <behaviors dir="src/test/easyb">
      <include name="**/*.story" />
    </behaviors>
  </easyb>
</target>
```

Generate pretty HTML reports

Run easyb against these stories

Easyb in the SDLC

- ▶ Integrating easyb with Ant
- ▶ Use the easyb Ant task



Invoke the Ant task

```
$ ant easyb
Buildfile: build.xml

easyb:
[easyb] easyb is preparing to process 2 file(s)
[easyb] Running create account story (createAccount.story)
[easyb] Scenarios run: 1, Failures: 0, Pending: 0, Time elapsed: 0.696 sec
[easyb] Running withdraw money from account story
(withdrawMoneyFromAccount.story)
[easyb] Scenarios run: 3, Failures: 0, Pending: 1
[easyb]
[easyb] 4 total behaviors ran (including 1 pending behavior) with no failures
[easyb] easyb execution passed

BUILD SUCCESSFUL
Total time: 4 seconds
```

Number of behaviors run

Also shows unimplemented stories

Easyb in the SDLC

- ▶ Integrating easyb with Maven
- ▶ Use the **maven-easyb-plugin** plugin



```
<project...>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.easyb</groupId>
        <artifactId>maven-easyb-plugin</artifactId>
        <version>0.9.5.2</version>
        <configuration>
          <storyType>html</storyType>
          <storyReport>target/easyb/easyb.html</storyReport>
          <xmlReport>target/easyb/report.xml</xmlReport>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

The Maven easyb plugin

Generate pretty HTML reports

Easyb in the SDLC

- ▶ Integrating easyb with Maven
- ▶ Use the **maven-easyb-plugin** plugin
- ▶ Just run **mvn test**



```
$ mvn test
```

Run the easyb tests

```
...
[INFO] [easyb:test {execution: default}]
[INFO] Using easyb dependency org.easyb:easyb:jar:0.9.5.2:compile
[INFO] Using easyb dependency commons-cli:commons-cli:jar:1.1:compile
[INFO] Using easyb dependency org.codehaus.groovy:groovy-all:jar:1.6.0:compile
[INFO] Using easyb dependency junit:junit:jar:3.8.2:compile
[INFO] Using easyb dependency org.apache.ant:ant:jar:1.7.1:compile
[INFO] Using easyb dependency org.apache.ant:ant-launcher:jar:1.7.1:compile
[INFO] Using easyb dependency jline:jline:jar:0.9.94:compile
[java] Running create account story (createAccount.story)
[java] Scenarios run: 1, Failures: 0, Pending: 0, Time elapsed: 0.647 sec
[java] Running withdraw money from account story (withdrawMoneyFromAccount.story)
[java] Scenarios run: 3, Failures: 0, Pending: 1, Time elapsed: 0.104 sec
[java] 4 total behaviors ran (including 1 pending behavior) with no failures
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
```

Easyb test results

Unimplemented stories

Easyb in the SDLC

▶ HTML test reports with Easyb



easyb -- bdd in java can't get any easier

sections

- Summary
- Stories
- Stories Text

Behaviors	Failed	Pending	Time (s)
4	1	1	0.738

Summary

Stories	Scenarios	Failed	Pending	Time (sec)
2	4	1	1	0.738

Stories Summary

Specifications	Failed	Pending	Time (sec)
0	0	0	0.0

Specifications Summary

Easyb in the SDLC

▶ HTML test reports with Easyb

The screenshot displays an HTML test report for Easyb. It features a 'Stories List' table with columns for Story, Scenarios, Failed, Pending, and Passed. Below the table, individual story details are shown, including Gherkin-style scenarios and their execution results. A failure is highlighted with a red background and a stack trace. A pending story is also shown at the bottom.

Story	Scenarios	Failed	Pending	Passed
create account	1	0	0	0.608
withdraw money from account	3	1	1	0.121

Withdraw money from an account success

given an account with a certain balance
when a sum is withdrawn from the account
then the withdrawn sum is deducted from the account balance success

Withdraw more money than there is in the account failure 0.026

given an account with a certain balance
when an amount is withdrawn that is greater than the balance
then an InsufficientFundsException is raised success

then the account balance remains unchanged failure

expected 150 but was 100

sun.reflect.NativeConstructorAccessorImpl.newInstance(...)
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructor...
java.lang.reflect.Constructor.newInstance(Constructor.java:494)
org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:77)
org.codehaus.groovy.reflection.CachedConstructor.doConstructorInvoke(CachedConstructor.java:71)
org.codehaus.groovy.runtime.callsite.ConstructorSite\$ConstructorSiteNoUnwrap.callConstructor(ConstructorSite.java:84)
org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:52)
org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:192)
org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:200)
org.easyb.BehaviorCategory.throwValidationException(BehaviorCategory.groovy:11)

Withdraw money from blocked account pending

given a blocked account
when money is withdrawn from the account
then an AccountBlockedException is raised pending

then the account balance remains unchanged pending

Test results summary

Stakeholder-friendly story details

Test failure details

Unimplemented stories

Easyb in the SDLC

- ▶ Easyb currently lacks good CI integration
- ▶ You can deploy the current HTML test results
- ▶ But XML test results are not JUnit-compatible
- ▶ So CI tools like Hudson and Bamboo cannot collect test results metrics



JDave in the SDLC

- ▶ JUnit extension
 - ▶ Ant, Maven, Buildr, ...
- ▶ JUnit XML output



JDave in the SDLC

- ▶ Running JDave specifications with Ant
 - ▶ Use the standard junit task



```
<target name="test">
  <junit>
    <classpath refid="classpath.test"/>
    <formatter type="xml"/>
    <batchtest todir="${target.reports}">
      <fileset dir="${target.test}">
        <include name="**/*Spec.class"/>
      </fileset>
    </batchtest>
  </junit>
</target>
```

The only difference to running standard JUnit tests is our naming convention.

JDave in the SDLC

- ▶ Running JDave specifications with Ant
 - ▶ Use the standard junit task



Invoke the Ant task

```
$ ant test
Buildfile: build.xml

compile:
  [mkdir] Created dir: /Work/jdave-examples/target/classes/prod
  [javac] Compiling 2 source files to /Work/jdave-examples/target/classes/prod
  [javac] Compiling 1 source file to /Work/jdave-examples/target/classes/prod

test:
  [mkdir] Created dir: /Work/jdave-examples/target/reports
  [junit] Running jdave.examples.StackSpec
  [junit] Tests run: 8, Failures: 0, Errors: 0, Time elapsed: 0.116 sec

BUILD SUCCESSFUL
Total time: 1 second
```

Behaviors being run are called "tests"...

...but at least they're fast

JDave in the SDLC

- ▶ Integrating JDave with Maven
 - ▶ Using the standard Surefire plugin



```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <configuration>
    <includes>
      <include>**/*Spec.java</include>
    </includes>
    <systemProperties>
      <property>
        <name>jdave.tools.specdox.format</name>
        <value>txt</value>
      </property>
    </systemProperties>
  </configuration>
</plugin>
```

Tell Surefire what your specs look like

Generate a plain text report (or XML for further processing and transformation)

JDave in the SDLC

- ▶ Integrating JDave with Maven
 - ▶ Using the standard Surefire plugin
 - ▶ JDave needs to be added as a dependency
 - ▶ And the reporting plugin (if you want them)



```
<dependencies>
  <dependency>
    <groupId>org.jdave</groupId>
    <artifactId>jdave-junit4</artifactId>
    <version>1.2-SNAPSHOT</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.jdave</groupId>
      <artifactId>jdave-report-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
```

JDave in the SDLC

- ▶ Integrating JDave with Maven
 - ▶ Using the standard Surefire plugin
 - ▶ JDave needs to be added as a dependency
 - ▶ And the reporting plugin (if you want them)
 - ▶ These are available from the LaughingPanda repository



```
<pluginRepositories>
  <pluginRepository>
    <id> laughing-panda </id>
    <url> http://www.laughingpanda.org/maven2/ </url>
  </pluginRepository>
</pluginRepositories>
<repositories>
  <repository>
    <id> laughing-panda </id>
    <url> http://www.laughingpanda.org/maven2/ </url>
  </repository>
</repositories>
```

JDave in the SDLC

- ▶ Integrating JDave with Maven
- ▶ Just run **mvn test**



Run the JDave specifications

```
$ mvn test
...
[INFO] [surefire:test]
[INFO] Surefire report directory: /Work/jdave-examples
-----
T E S T S
-----
Running jdave.examples.StackSpec
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time
Results :
Tests run: 8
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 4 seconds
```

JDave specifications are again reported just like JUnit tests...

StackSpec:

Empty stack

- is empty
- is no longer empty after push

Full stack

- is full
- complains on push
- contains all items
- does not contain removed item
- contains all but removed item

Stack which is neither empty nor full

- adds to the top when sent push

...and in a more human-readable format

JDave in the SDLC

- ▶ Running JDave specifications with Buildr
- ▶ List dependencies and include "*Spec"



```
define 'jdave-examples' do
  project.group = 'org.agile2009'
  project.version = '1.0-SNAPSHOT'

  test.with 'junit:junit:jar:4.6',
            'org.jmock:jmock:jar:2.4.0',
            'org.jmock:jmock-legacy:jar:2.4.0',
            'org.jmock:jmock-junit4:jar:2.4.0',
            'asm:asm:jar:1.5.3',
            'cglib:cglib-nodep:jar:2.1_3',
            'org.objenesis:objenesis:jar:1.1',
            'org.hamcrest:hamcrest-core:jar:1.1',
            'org.hamcrest:hamcrest-library:jar:1.1',
            'org.jdave:jdave-core:jar:1.2-SNAPSHOT',
            'org.jdave:jdave-junit4:jar:1.2-SNAPSHOT'

  test.include '*Spec'
end
```

Our spec classes end with "Spec"

JDave in the SDLC

- ▶ Running JDave specifications with Buildr
- ▶ List dependencies and include "*Spec"



Invoke Buildr

```
$ buildr test
(in /Work/jdave-examples, development)
Testing jdave-examples
Running tests in jdave-examples
Trying to override old definition of datatype junit
[junit] Testsuite: jdave.examples.StackSpec
[junit] Tests run: 8, Failures: 0, Errors: 0, Time elapsed: 0.114 sec
[junit]
[junit] Testcase: isEmpty took 0.018 sec
[junit] Testcase: isNoLongerEmptyAfterPush took
[junit] Testcase: isFull took 0.001 sec
[junit] Testcase: complainsOnPush took 0.001 sec
[junit] Testcase: containsAllItems took 0.002 sec
[junit] Testcase: doesNotContainRemovedItem took 0.001 sec
[junit] Testcase: containsAllButRemovedItem took
[junit] Testcase: addsToTheTopWhenSentPush took
Completed in 0.987s
```

Each spec class is treated as a JUnit test suite...

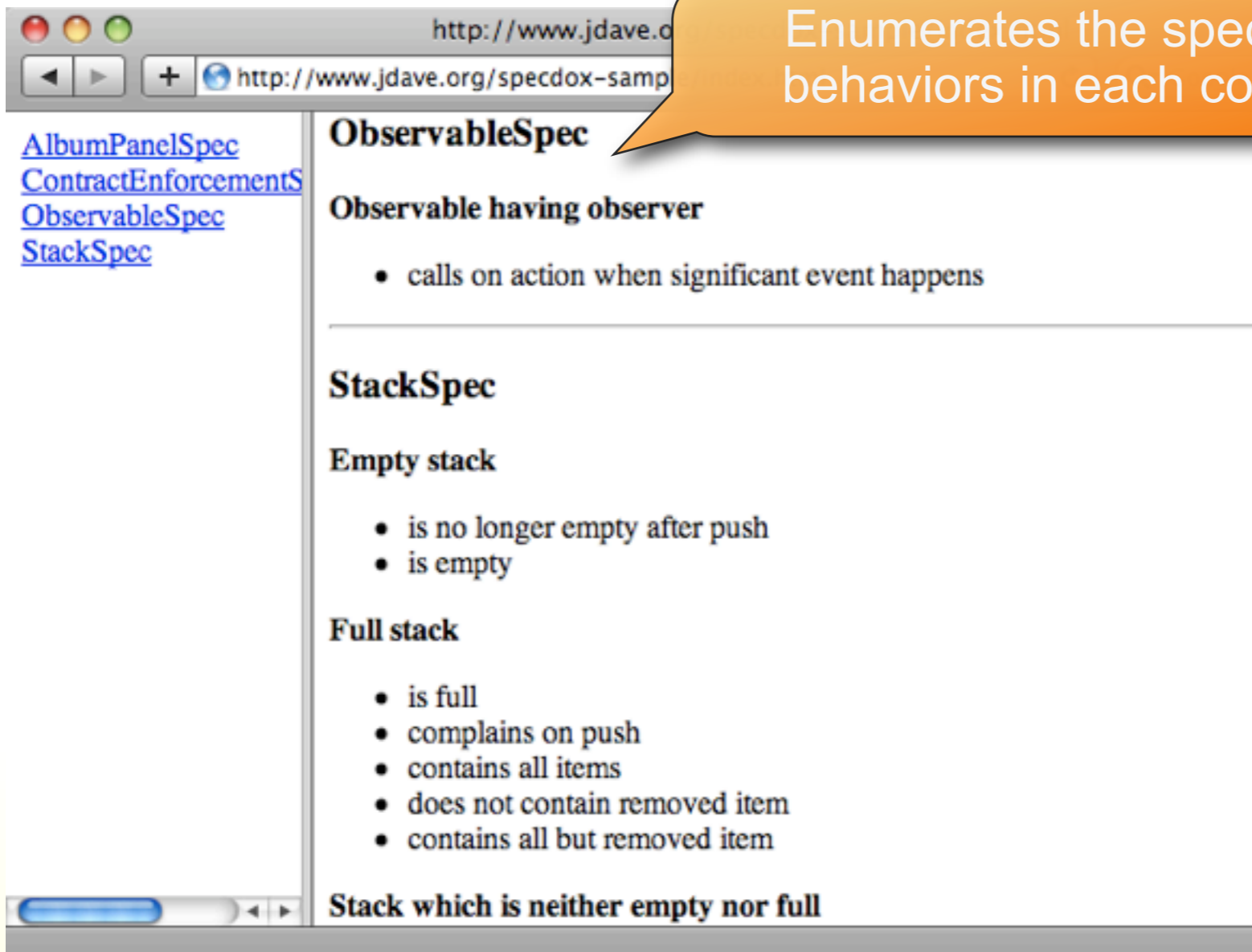
...and the individual specifications as test cases.

JDave in the SDLC

▶ HTML test reports with JDave



Enumerates the specified behaviors in each context

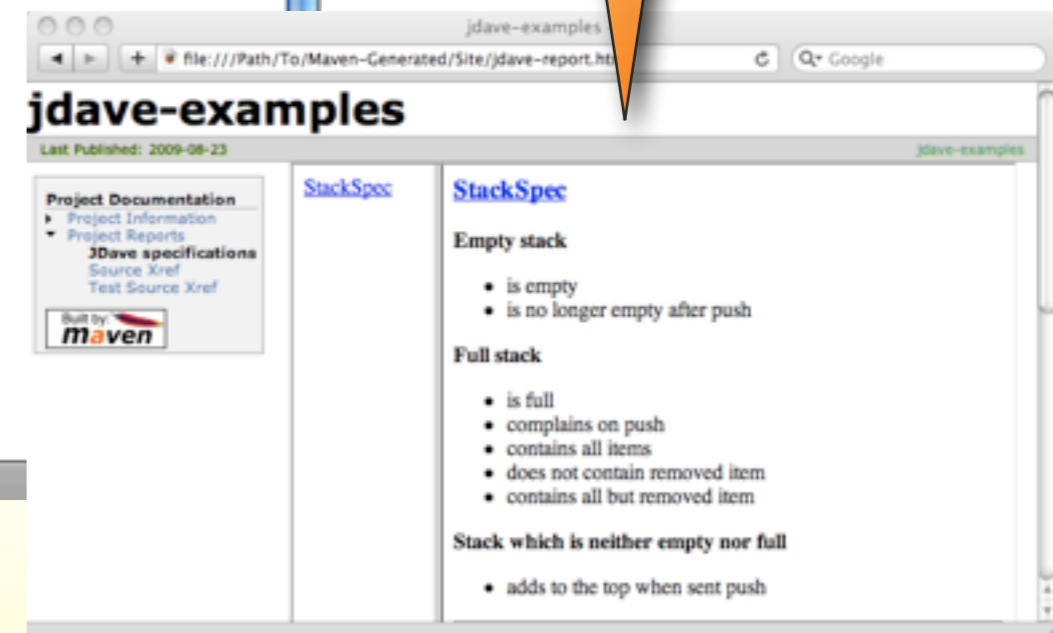


A screenshot of a web browser displaying a JDave HTML test report. The browser's address bar shows the URL <http://www.jdave.org/specdox-samp>. On the left side, there is a navigation menu with links for [AlbumPanelSpec](#), [ContractEnforcementS](#), [ObservableSpec](#), and [StackSpec](#). The main content area is titled "ObservableSpec" and contains the following sections:

- Observable having observer**
 - calls on action when significant event happens

- StackSpec**
 - Empty stack**
 - is no longer empty after push
 - is empty
 - Full stack**
 - is full
 - complains on push
 - contains all items
 - does not contain removed item
 - contains all but removed item
 - Stack which is neither empty nor full**

Integrated with Maven site



A screenshot of a web browser displaying a JDave HTML test report integrated with a Maven site. The browser's address bar shows the file path `file:///Path/To/Maven-Generated/Site/jdave-report.ht`. The page title is "jdave-examples" and it includes a "Last Published: 2009-08-23" timestamp. On the left, there is a "Project Documentation" sidebar with a "maven" logo. The main content area is titled "StackSpec" and contains the following sections:

- Empty stack**
 - is empty
 - is no longer empty after push
- Full stack**
 - is full
 - complains on push
 - contains all items
 - does not contain removed item
 - contains all but removed item
- Stack which is neither empty nor full**
 - adds to the top when sent push

JDave in the SDLC

▶ HTML test reports with JDave



The screenshot shows a web browser window titled "jdave-examples - Surefire Report". The address bar contains the file path "file:///Path/To/Maven-Generated/Site/surefire-report.html". The page content is organized into sections for different test classes:

- Test Cases**
 - [Summary] [Package List] [Test Cases]
 - StackSpec\$EmptyStack**

isEmpty	0.024
isNoLongerEmptyAfterPush	0.005
 - StackSpec\$FullStack**

isFull	0
complainsOnPush	0.001
containsAllItems	0.002
doesNotContainRemovedItem	0.001
containsAllButRemovedItem	0.001
 - StackSpec\$StackWhichIsNeitherEmptyNorFull**

addsToTheTopWhenSentPush	0.001
--------------------------	-------

Maven's Surefire plugin doesn't quite understand our conventions

JDave in the SDLC

- ▶ Ant's `<junitreport/>` task is somewhat insufficient for our purposes

A screenshot of a web browser window titled "Unit Test Results." The address bar shows the path "/Path/To/Ant-Generated/JUnit/Report.html". The page content includes a sidebar with navigation links for "Home", "Packages" (jdave.examples), and "Classes" (StackSpec). The main content area displays the following information:

Unit Test Results
Designed for use with [JUnit](#) and [Ant](#).

Class jdave.examples.StackSpec

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
StackSpec	8	0	0	0.121	2009-08-23T12:51:14	l162.local

Tests

Name	Status	Type	Time(s)
isEmpty	Success		0.023
isNoLongerEmptyAfterPush	Success		0.001
isFull	Success		0.001
complainsOnPush	Success		0.001
containsAllItems	Success		
doesNotContainRemovedItem	Success		
containsAllButRemovedItem	Success		
addsToTheTopWhenSentPush	Success		0.002

[Properties »](#)

Contexts are flattened, making the standard test report almost unusable.

What about IDE support?

- ▶ You work with an IDE.
So should your test framework.
- ▶ How well are the BDD frameworks supported in the major IDEs?
 - ▶ Syntax highlighting
 - ▶ Code completion
 - ▶ Running stories from within the IDE
 - ▶ ...



What about IDE support?

- ▶ IDE Support for easyb:
 - ▶ Several options
 - ▶ IntelliJ 
 - ▶ Eclipse 
 - ▶ NetBeans  **NetBeans**
 - ▶ Not all options are equal...



Easyb in your IDE

▶ Running easyb in IntelliJ

- ▶ Excellent Groovy support
- ▶ An easyb plugin is available



```
import com.wakaleo.bankonline.domain.Account

/**
 * A simple scenario
 */
scenario "Make initial deposit onto a new account", {
    given "a newly created account"
    when "an initial deposit is made into this account"
    then "the account balance should be equal to the amount deposited"
}

scenario "depositing money to an existing account", {
    given "an account with $100"
    when "an amount is deposited"
    then "the amount is added to the balance"
}
```

Run AccountDepositsStory.groovy

- Story account deposits
 - Scenario Make initial deposit onto a new account
 - Given a newly created account
 - When an initial deposit is made into this account
 - Then the account balance should be equal to the amount deposited
 - Scenario Make initial deposit onto a new account
 - Given a new account
 - When an initial deposit is made
 - Then the balance should be equal to the amount deposited
 - Scenario depositing money to an existing account
 - Given an account with \$100
 - When an amount is deposited
 - Then the amount is added to the balance

Easyb in your IDE

- ▶ Running easyb in Eclipse
 - ▶ Some Groovy support available
 - ▶ No dedicated easyb plugin yet
 - ▶ Easyb stories run via Ant or Maven



```
1 Easyb stories
2 AccountDepositsStory
3 SchemaSpyMojoTest
4 maven-schemaspys-plugin
Run As
Run Configurations...
Organize Favorites...
```

```
scenario "Make initial deposit onto a new account", {
    given "a newly created account"
    "an initial deposit is made into this account"
    then "the account balance should be equal to the amount deposited"
}

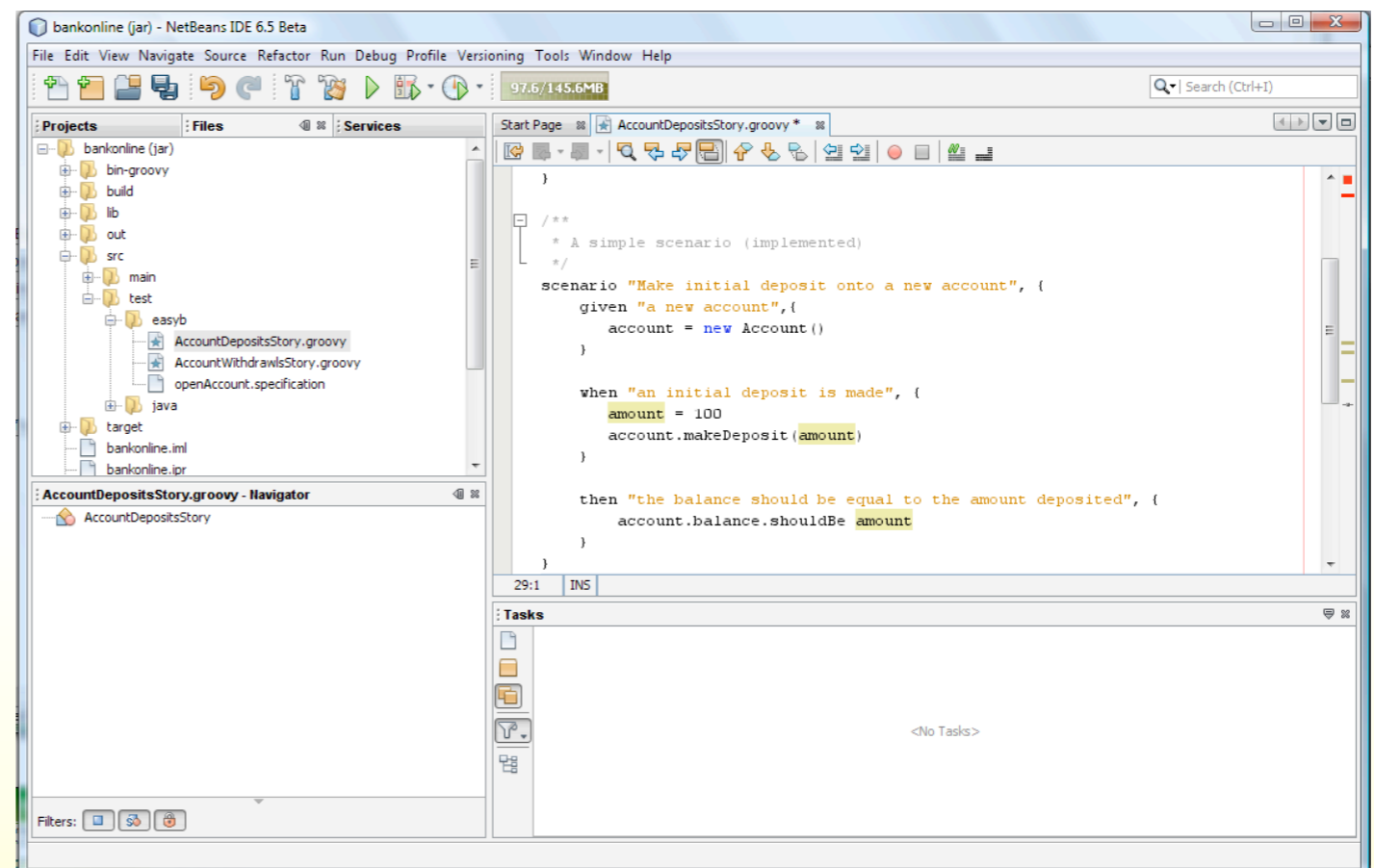
/**
 * A simple scenario (implemented)
 */
scenario "Make initial deposit onto a new account", {
    given "a new account", {
        account = new Account ()
    }
    when "an initial deposit is made", {
```

```
<terminated> Easyb stories [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (3/09/2008 11:18:41 PM)
Running account deposits story (AccountDepositsStory.groovy)
Scenarios run: 3, Failures: 0, Pending: 1, Time Elapsed: 0.913 sec
Running account withdrawls story (AccountWithdrawlsStory.groovy)
Scenarios run: 2, Failures: 0, Pending: 0, Time Elapsed: 0.344 sec

5 total behaviors run (including 1 pending behavior) with no failures
```

Easyb in your IDE

- ▶ Running easyb in NetBeans
 - ▶ Good Groovy support available in NetBeans 6.5 and 6.7
 - ▶ No dedicated easyb plugin yet
 - ▶ Easyb stories run via Ant or Maven



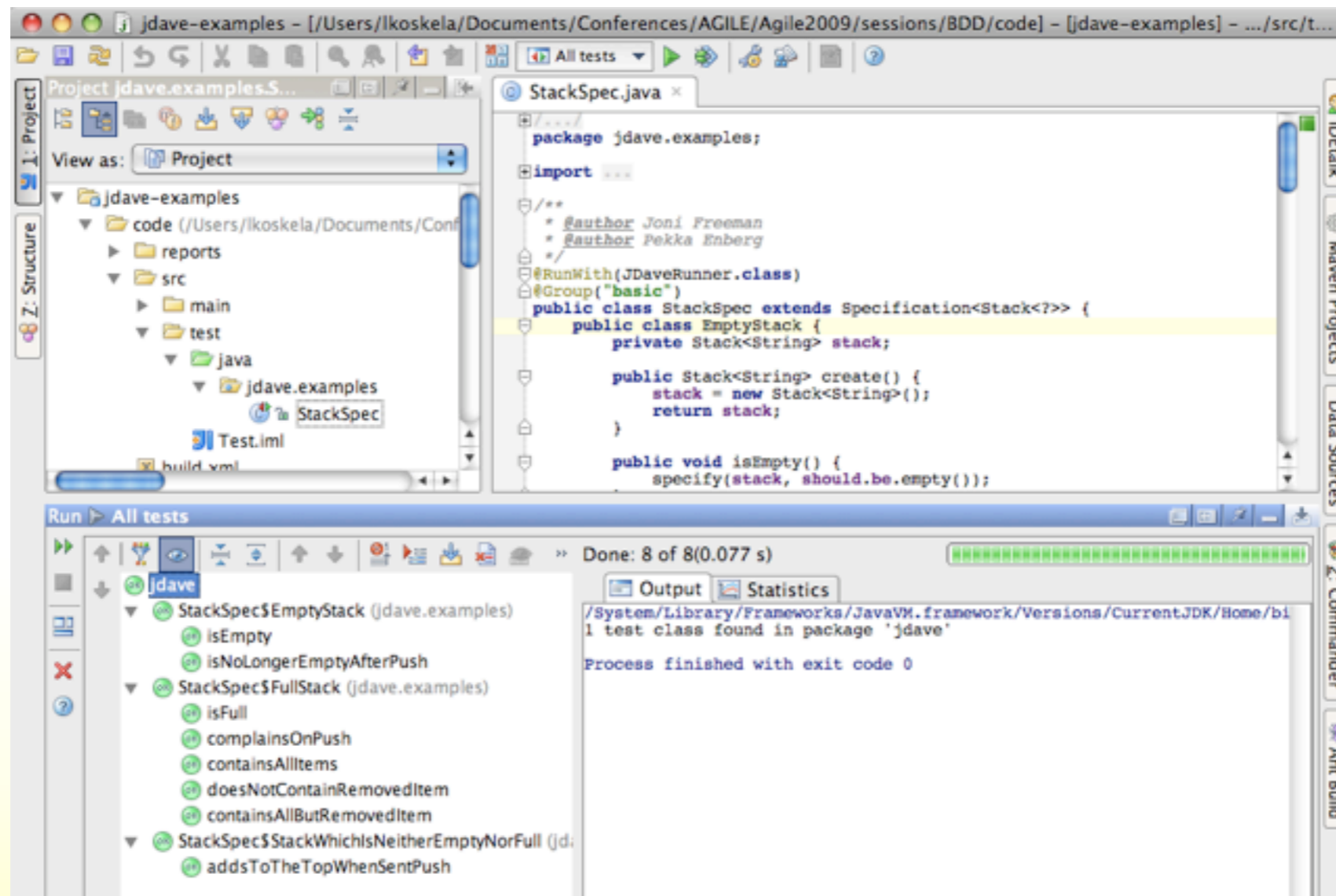
JDave in your IDE

- ▶ Standard JUnit, remember?
- ▶ All mainstream IDEs support JDave:
 - ▶ IntelliJ 
 - ▶ Eclipse 
 - ▶ NetBeans  **NetBeans**
 - ▶ (are there any others?)



JDave in your IDE

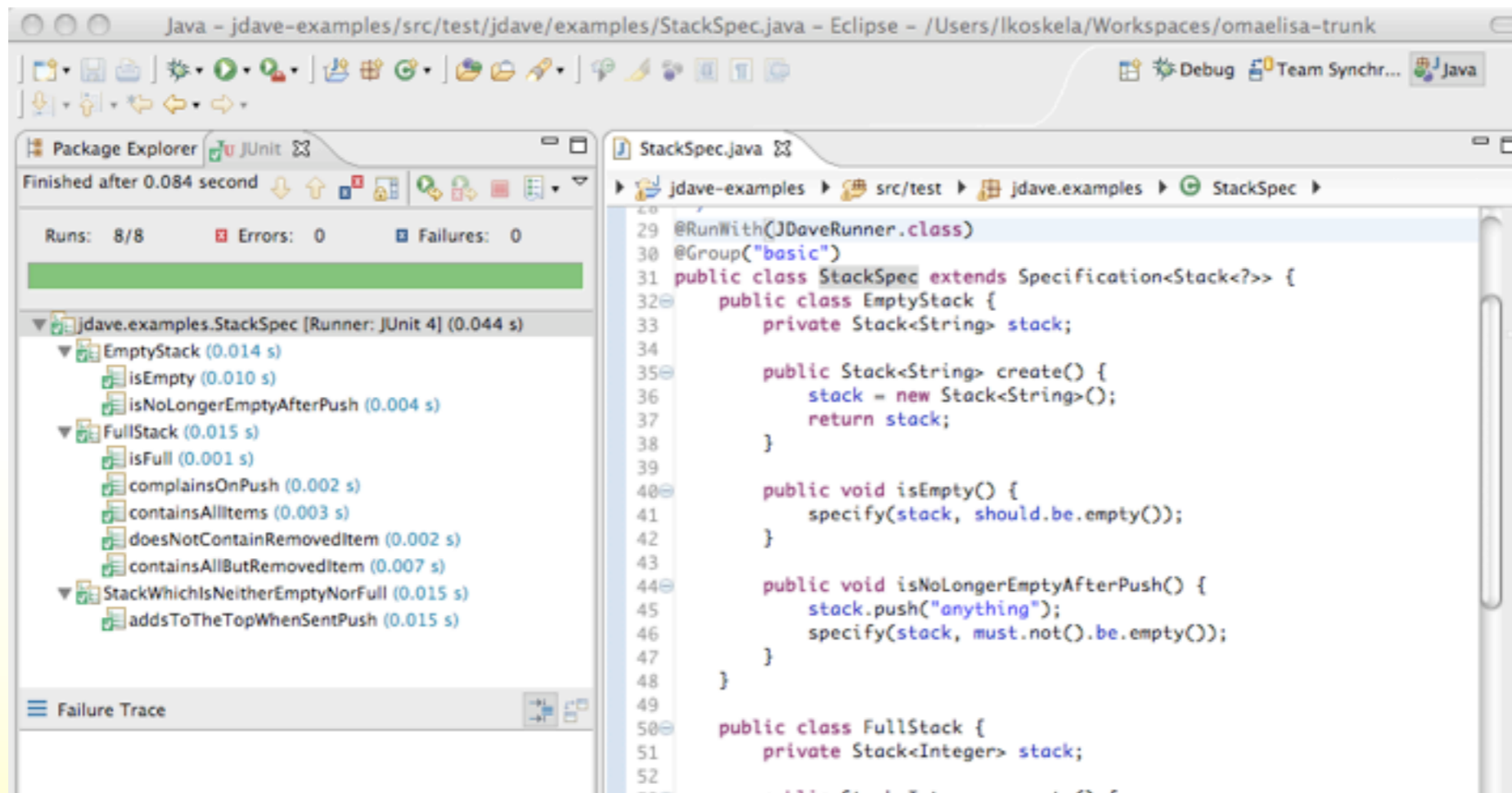
- ▶ Running JDave in IntelliJ
 - ▶ Through the built-in JUnit runner
 - ▶ Click a spec and IDEA navigates to the code



JDave in your IDE

▶ Running JDave in Eclipse

- ▶ Through the built-in JUnit runner
- ▶ Click a spec and Eclipse navigates to the code
- ▶ (only works for classes and specifications, not for contexts)



JDave in your IDE

▶ Running JDave in NetBeans

- ▶ Through the built-in JUnit runner
- ▶ Runs contexts but doesn't report them

A screenshot of the NetBeans IDE 6.7.1 interface. The title bar reads "jdave-examples (jar) - NetBeans IDE 6.7.1". The left sidebar shows a project tree with "jdave-examples (jar)" expanded to show "StackSpec.java". The main editor window displays the code for "StackSpec.java", which includes annotations like "@RunWith(JDaveRunner.class)" and "@Group('basic')", and defines a "StackSpec" class with an inner "EmptyStack" class. The "Test Results" window at the bottom left shows a green progress bar at "100.0 %" and a list of 8 tests that all passed. The "Output" window at the bottom right shows the message "Tests run: 8, Failures: 0, Errors: 0, Skipped: 0" and "BUILD SUCCESSFUL". An orange callout bubble points to the test results with the text "NetBeans flattens contexts into one blob".

```
/**...*/
@RunWith(JDaveRunner.class)
@Group("basic")
public class StackSpec extends Specification<Stack<?>> {
    public class EmptyStack {
        private Stack<String> stack;

        public Stack<String> create() {
            stack = new Stack<String>();
            return stack;
        }

        public void isEmpty() {
            specify(stack, should.be.empty());
        }

        public void isNoLongerEmptyAfterPush() {
            stack.push("anything");
            specify(stack, must.not().be.empty());
        }
    }
}
```

Test Results

100.0 %

All 8 tests passed.(0.138 s)

- jdave.examples.StackSpec passed
- isEmpty passed (0.051 s)
- isNoLongerEmptyAfterPush passed (0.004 s)
- isFull passed (0.0 s)
- complainsOnPush passed (0.002 s)
- containsAllItems passed (0.004 s)
- doesNotContainRemovedItem passed (0.002 s)
- containsAllButRemovedItem passed (0.002 s)
- addsToTheTopWhenSentPush passed (0.002 s)

Output - org.agile2009_jdave-examples_jar_1.0-SNA...

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0

BUILD SUCCESSFUL

NetBeans flattens contexts into one blob

Limitations of Easyb

▶ What are the weak points of easyb?

▶ Script errors can be hard to find

```
scenario "Deposit cash via the web interface", {  
  given "the application home page is displayed", {  
    selenium.open("/ebank-web")  
    selenium.waitForPageToLoad("2000")  
  }  
  and "the current balance is 0" {  
    selenium.isTextPresent("Current balance: \">$100")  
  }  
}
```

Missing comma

Obscure error message

```
[java] There was an error running your easyb story or specification  
[java] groovy.lang.MissingMethodException: No signature of method: Script1.the current balance is 0() is applicable for argument  
types: (Script1$_run_closure3_closure7) values: [Script1$_run_closure3_closure7@70c116]  
[java]   at org.codehaus.groovy.runtime.ScriptBytecodeAdapter.unwrap(ScriptBytecodeAdapter.java:54)  
[java]   at org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.callCurrent(PogoMetaClassSite.java:78)  
...
```

▶ Documentation a little light

▶ Limited IDE support (Groovy)

▶ XML output not JUnit-compatible



Limitations of JDave

- ▶ What are the weak points of JDave?
 - ▶ Java's syntax isn't quite as flexible as one might hope
 - ▶ IDE support is limited as far as editing goes
 - ▶ Inner classes
 - ▶ Anonymous subclasses
 - ▶ Carpal tunnel syndrome
 - ▶ The built-in specify(...) matchers could render more descriptive error messages upon failure



Thank You