

TDD++

Test-driven Development with Groovy

Björn Beskow

bjorn.beskow@callistaenterprise.se

2014-01-29

Magnus Ekstrand

magnus.ekstrand@callistaenterprise.se

Peter Merikan

peter.merikan@callistaenterprise.se

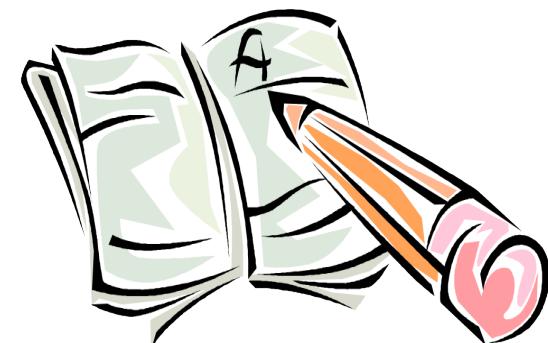


Logistics

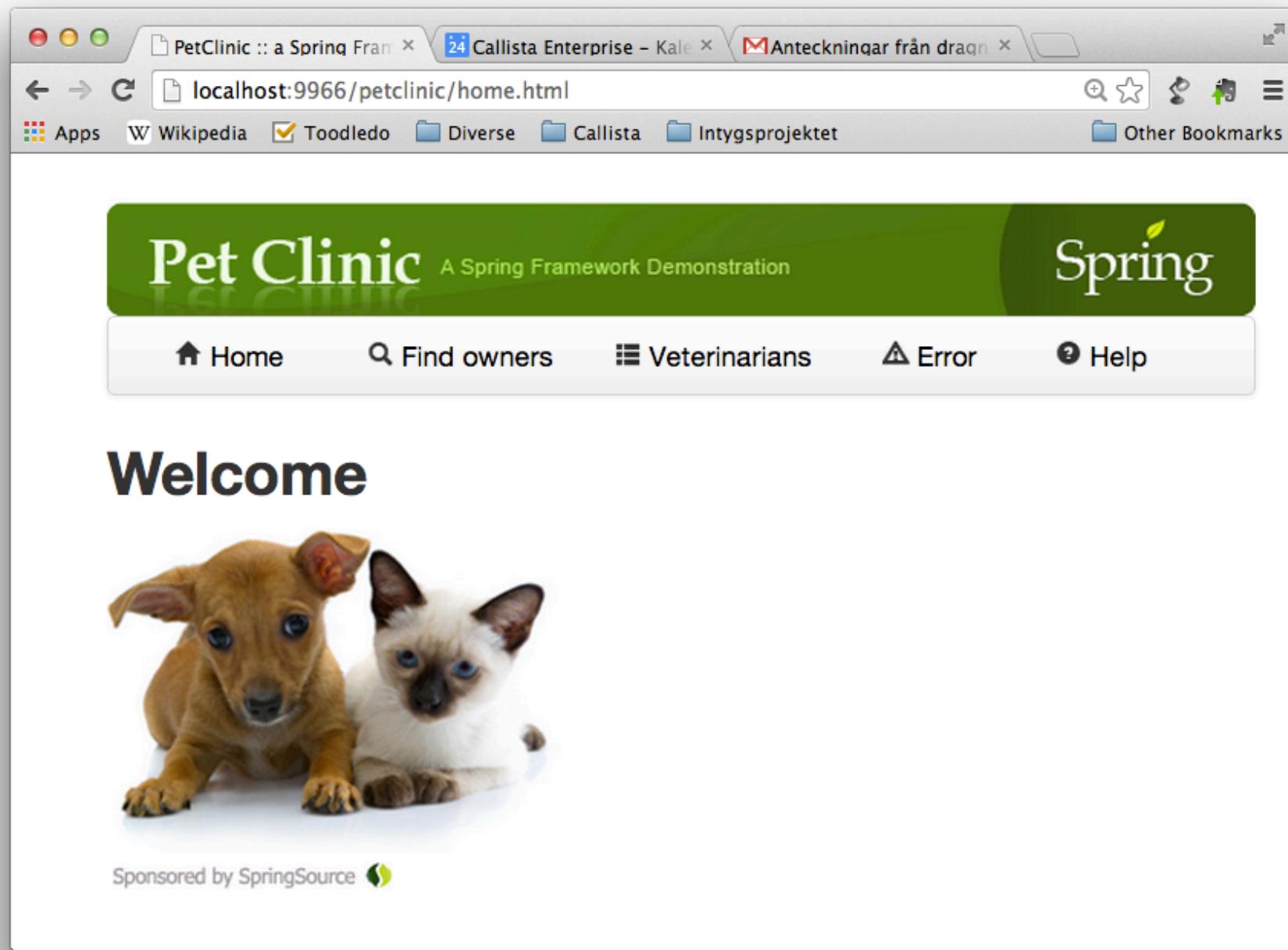


Agenda

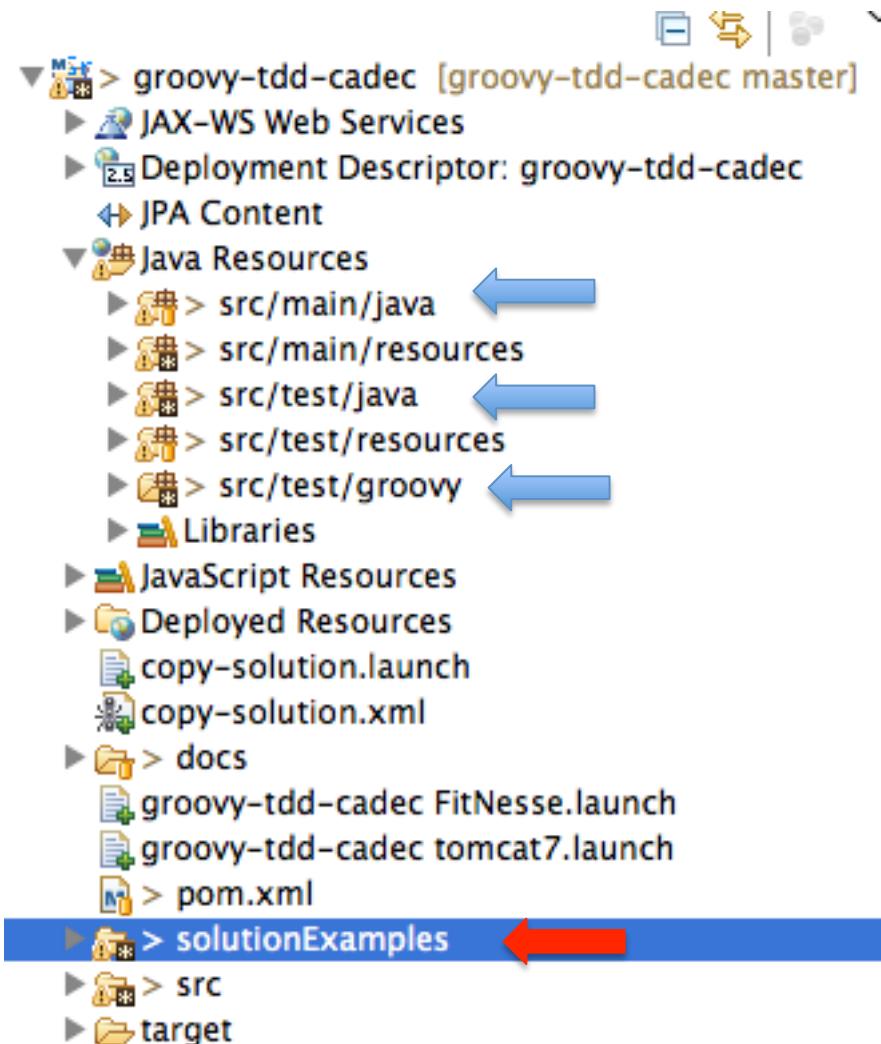
- Test Automation Basics
- Groovy Basics
- Groovy Unit Testing
 - Mocking collaborators
 - Assertions & matchers
- Integration Testing
 - Managing test data
 - Working with SQL databases
- API testing
 - Working with XML
 - Working with JSON
- Acceptance Testing
 - Specifications
 - FitNesse and Groovy
 - Web Acceptance Testing



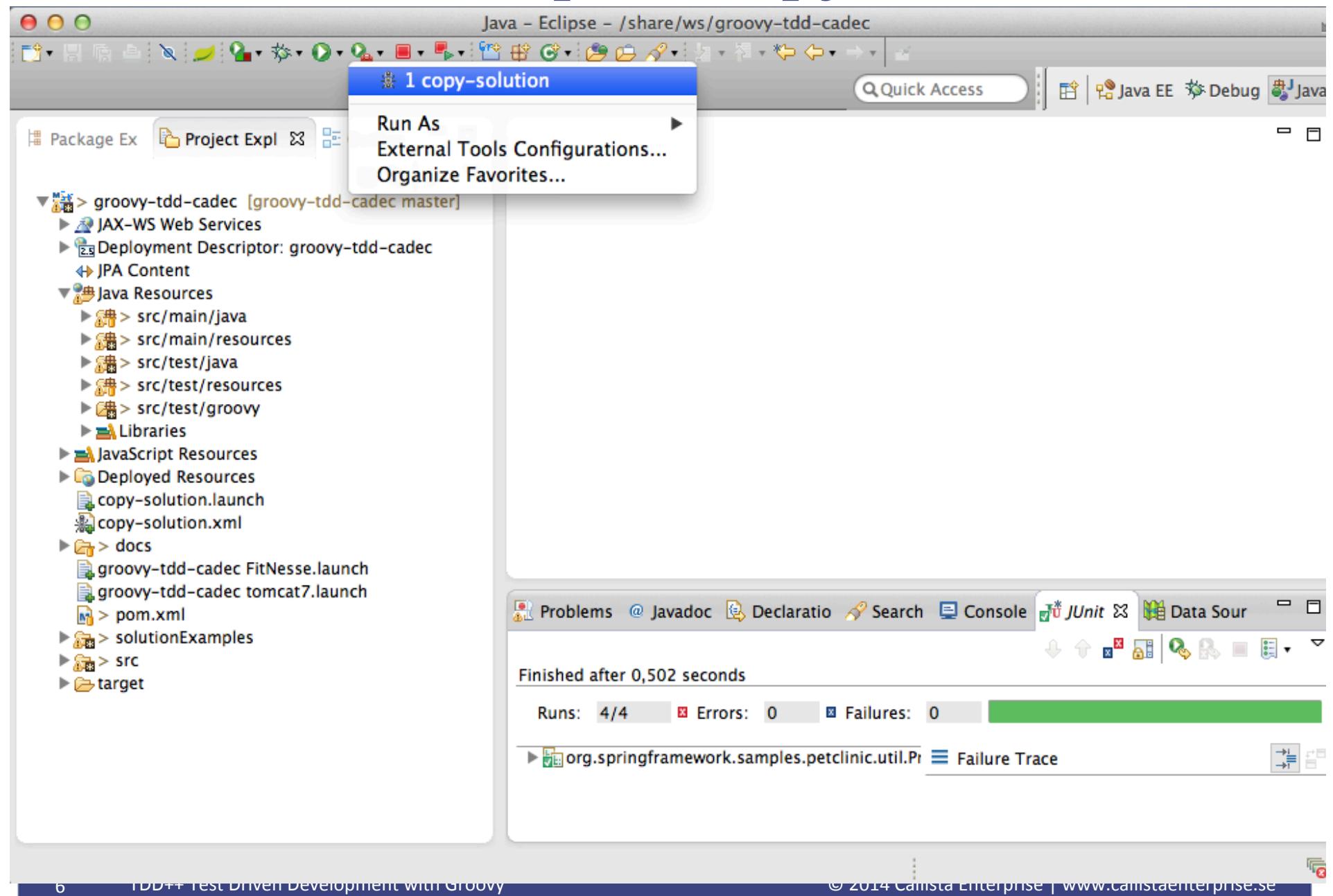
Example app: Spring Pet Clinic

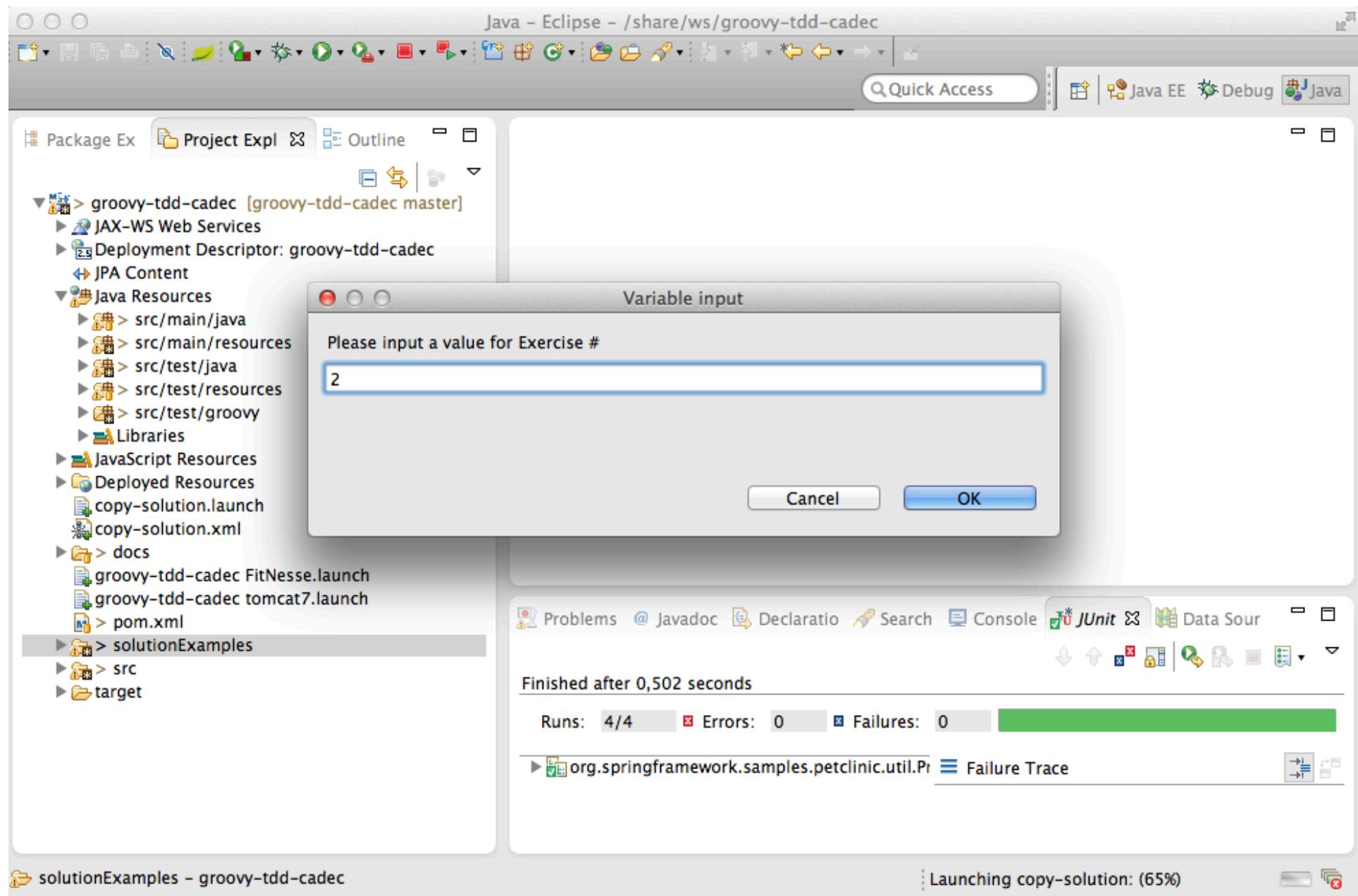


Example app: Project Layout



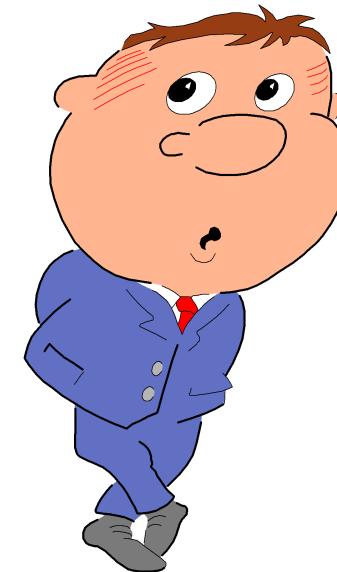
Solution Examples: copy-solution





About Tests ...

- Everybody knows they should, but few actually do
- “Why isn’t this tested before”?
 - Because it has been too expensive, difficult, cumbersome to test
 - Because we have been too busy



Quality Assurance precedes Quality Assessment

- Testing is about Quality Assurance, not just Quality Assessment
- Quality Assessment only indirectly affect quality
- Testing *reveals information*
- Testing helps *focus project activity*

TDD
ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT



Automated Tests must be

- easy to write
- easy to find
- easy to run
- easy to maintain

otherwise

- they will slow you down
- they will get left behind
- you'll go back to manual testing



Critical Success Factors

- **Repeatability and Consistency**
 - Once the test is complete, it should pass repeatedly, whether it executes by itself or within a test suite.
 - When a completed test fails, we need to quickly and accurately pinpoint the cause: did the test uncover a bug in the system, or is the test itself faulty?
- **Readability**
 - The tests are the definitive reference for the system requirements.
- **Maintainability**
 - Sufficient test coverage usually yields as much (or more) test code than system code, hence we have to be as concerned (or more) with the maintenance costs of test code as compared to system code.

Some aspects of Java may (sometimes) harm your productivity

```
public void testInsertDuplicateVisitShouldThrowException() {  
    Pet pet = new Pet();  
    List<Visit> visits = pet.getVisits();  
    Visit firstVisit = visits.get(0);  
    Visit copy = null;  
    try {  
        copy = (Visit) firstVisit.clone();  
    } catch (CloneNotSupportedException e) {  
        // Should never happen  
    }  
    copy.setId(0);  
    visits.add(copy);  
    this.clinicService.saveVisit(copy);  
    try {  
        this.clinicService.savePet(pet);  
        fail("DataIntegrityViolationException expected");  
    } catch (DataIntegrityViolationException e) {  
        // Expected  
    }  
}
```

Groovy Basics

- Groovy is a dynamic Java derivative on and for the JVM
 - Dynamically typed, with optional static typing
 - Compiles directly down to bytecode
- Inspired by other dynamic OO languages: Smalltalk, Ruby, Python
- Totally object-oriented
- Goal is to *greatly simplify* the life for developers, ***without a steep learning curve***

Syntax basics: A Java program

```
public class HelloWorld {  
  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String sayHello() {  
        return "Hello, " + name + "!";  
    }  
  
    public static void main(String[] args) {  
        HelloWorld helloWorld = new HelloWorld();  
        helloWorld.setName("Groovy");  
        System.out.println(helloWorld.sayHello());  
    }  
}
```

Corresponding Groovy program

```
public class HelloWorld {  
  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String sayHello() {  
        return "Hello, " + name + "!";  
    }  
  
    public static void main(String[] args) {  
        HelloWorld helloWorld = new HelloWorld();  
        helloWorld.setName("Groovy");  
        System.out.println(helloWorld.sayHello());  
    }  
}
```

Syntax: What is the same?

- Keywords and statements (but Groovy adds `in` and `def`)
- Try/catch/finally exception handling
- Class, interface, field and method definitions
- Packaging and imports
- Operators, expressions and assignments
- Control structures
- Annotations, Generics, Enums

Syntax Gotchas: Subtle differences

- Array literals:

Java: String[] languages = {"java", "groovy", "perl"};

Groovy: String[] languages = ["java", "groovy", "perl"]

- Equals and identity:

Java: x.equals(y) Java: x == y

Groovy: x == y Groovy : x.is(y)

Optional embellishments

- Parentheses, return statements and semicolons are optional, as long as the result is unambiguous

```
public class HelloWorld {  
  
    public String sayHello() {  
        "Hello!"  
    }  
  
    public static void main(String[] args) {  
        HelloWorld helloWorld = new HelloWorld()  
        println helloWorld.sayHello()  
    }  
}
```

Optional import statements

- Import statements are optional for the following packages:
 - groovy.lang.*
 - groovy.util.*
 - java.lang.*
 - java.util.*
 - java.net.*
 - java.io.*
 - java.math.BigDecimal, java.math.BigInteger

Groovy Beans: Properties

- In Groovy, a JavaBean have true properties
 - Syntax support for property access
 - Corresponding getter and setter method generated automatically

```
public class HelloWorld {  
  
    String name  
  
    public String sayHello() {  
        return "Hello, " + name + "!"  
    }  
  
    public static void main(String[] args) {  
        HelloWorld helloWorld = new HelloWorld()  
        // helloWorld.setName("Groovy")  
        helloWorld.name = "Groovy"  
        println(helloWorld.sayHello());  
    }  
}
```

Groovy Beans: Named Parameters

- Named parameters for constructors, which sets the corresponding JavaBeans properties

```
Visit v = new Visit(description:'Broken tail', price: 480.00)
```

instead of

```
Visit v = new Visit()  
v.description = 'Broken tail'  
v.price = 480.00
```

Groovy Strings (GStrings)

- Variable interpolation in strings using \${} notation:

```
int age = 20  
println "Age: ${age}"
```

- " enclosed strings performs variable interpolation,
' enclosed strings doesn't:

```
println 'Literal ${value}'
```

- """ for multiline strings:

```
println """This is a  
string that spans  
several rows"""
```

Lists, Maps and Ranges

- Lists

```
List list = [1,2,3,4,5]
println "first element is ${list[0]}"
println "last element is ${list[-1]}"
list << 6
List threeToFive= list[2..4]
```

- Maps

```
Map frameworks = [groovy:"Grails", ruby:"Rails"]
println """The groovy framework is
${frameworks['groovy']}"""
```

- Ranges

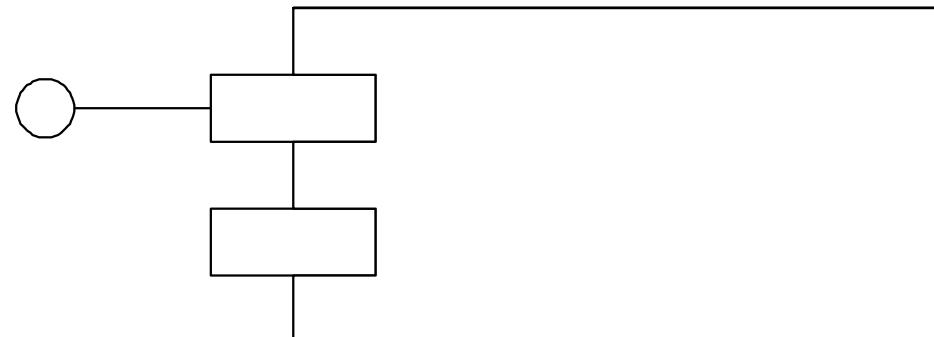
```
for (i in 1..10) {
    println "$i"
}
```

Exercise 0 – Gettings started

- Import the project named 'groovy-tdd-cadec' (File -> Import -> General-> Existing Projects into workspace)
- Locate the unit test *cadec.GettingStarted* in the *src/test/groovy* folder
- Run the unit test by right-clicking it and choose Run As -> JUnit test and see it fail
- Fix it

Unit Tests

- Black-box or White-box test of a *logical unit*, which verifies that the logical unit behaves correctly – *honors its contract*.
- A self-contained software module (typically a Class) containing one or more test scenarios which tests a Unit Under Test *in isolation*.



JUnit Test Example

```
public interface Account {  
    public void withdraw(double amount);  
    public void deposit(double amount);  
    public double balance();  
    //...  
}  
public class AccountImplTest {  
    @Test  
    public void withdraw() {  
        AccountImpl account = new AccountImpl("1234", 2000);  
        account.withdraw(300);  
        Assert.assertEquals(1700, account.balance());  
    }  
    @Test  
    public void withdrawTooMuch() { ... }  
    //...  
}
```

Exercise 1 – Basics

- Locate and inspect
org.springframework.samples.petclinic.util.PriceCalculator
within the *src/main/java* folder: This is your Unit under Test
- Locate the Unit Test
org.springframework.samples.petclinic.util.PriceCalculatorTest
within the *src/test/groovy* folder.
- Implement test methods to
 - Verify that pets older than 3 years pay an additional 20%
 - Verify that from 3rd visit, you get 20% rebate
- Utilize the Groovy language syntax constructs for creating test data

JUnit Assert woes

```
public class Assert  
extends java.lang.Object
```

A set of assertion methods useful for writing tests. Only failed assertions are recorded. These methods can be used directly: `Assert.assertEquals(...)`, however, they read better if they are referenced through static import:

```
import static org.junit.Assert.*;  
...  
assertEquals(...);
```

See Also:

[AssertionError](#)

JUnit Assert woes

```
Node xml = ...  
String expected = "y";  
String actual = xml.getText();  
assertEquals(expected, actual);  
  
assertTrue("Node is invalid", xml.isValid());
```

Groovy Power Assert

- assert is a Java keyword, but of limited use
- In Groovy, the output of a failing assert provides extremely readable output:

```
def xml = new XmlParser().parseText("<test>x</test>")  
assert xml.text() == "y"
```

Assertion failed:

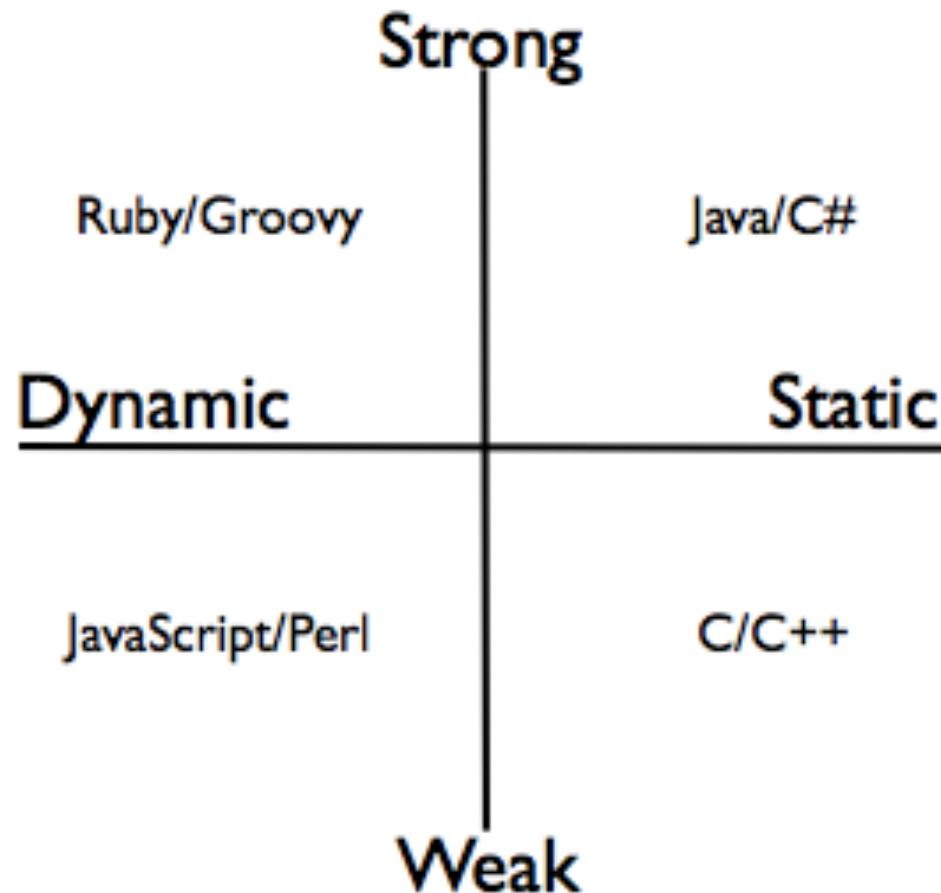
```
assert xml.text() == "y"  
| | |  
| x false  
test[attributes={} ; value=[x]]
```

Groovy Power Assert

```
def xml = ...  
assert xml.valid, "xml is invalid"
```

```
java.lang.AssertionError: xml is invalid.  
Expression: xml.valid
```

Dynamic vs Static Typing



Dynamic Typing (a.k.a Duck Typing)

- "If it walks like a duck and quacks like a duck, it is probably a duck"

```
public class HelloWorld {  
  
    def person  
  
    public String sayHello() {  
        return "Hello, " + person.name + "!"  
    }  
  
}
```

Optional static typing

- Static types may be optionally provided:

```
def x = 25
int y = 50
println x + y
```

- Useful for interoperability with Java

AST transformations: @TypeChecked

- Brings Static Type Checking into the dynamic world of Groovy
- Compiler performs sophisticated type inference

AST transformations: `@CompileStatic`

- One step further:
 - Statically compile a class or block of code, which makes the bytecode (almost) identical to corresponding compiled Java code, with identical performance
 - Can prevent dynamic behavior to be inadvertently applied to a block of code (a.k.a. “Monkey Patching”)

Groovy Closures

- A reusable / assignable block of code delimited by curly braces
 - Can take parameters
 - Can be assigned to a variable
 - Can be passed as parameters to methods, or inlined
 - Can refer to its surrounding context

```
def greeting = "Hello"
def printGreeting = {toWhom ->
    println "${greeting} ${toWhom}" }

printGreeting("Groovy")

["Björn", "Magnus", "Peter"].each {
    printGreeting( it )
}
```

Closures in typical action

- Capture prototypical usage, and provide the essential work as a closure:

```
def sql = new Sql(dataSource)
def list = sql.eachRow("select * from USER") {
    println it.name
}
```

- Iterating over collections, performing essential work for each item:

```
List<Orders> orders = ...
orders.each { order ->
    if (order.valid()) order.dispatch()
}
```

Iterables in Groovy

- Every object that implements Iterator provides numerous useful methods that iterates over the items using a closure:

```
def list = []
5.times {
    list << it
}
list.each { println it }
assert list.any { it == 3 }
assert list.every { it >= 0 }
assert list.collect { it * it } == [0, 1, 4, 9, 16]
assert list.findAll { it > 1 && it < 4 } == [2, 3]
assert list.find { it > 1 } == 2
assert (1..9).findAll {it % 2 == 1} == [1,3,5,7,9]
```

Exercise 2 – Asserts and Dynamic Typing

- Update your recent test to use Groovy asserts and Dynamic Typing
- Use a closure to create dynamic test data
- Add a test to verify that additional visits don't give additional rebates

Test Doubles

- Frequently used technique to
 - Isolate code under test
 - Make test execute faster
 - Make execution deterministic
 - Simulate special conditions



Test Doubles come in different flavors

- **Dummy** objects are passed around but never actually used.
- **Fake** objects actually have working implementations, but usually take some shortcut.
- **Stubs** provide canned answers to calls made during the test.
- **Spies** are stubs that also record some information based on how they were called.
- **Mocks** are pre-programmed with expectations which form a specification of the calls they are expected to receive.

Test Doubles in Java: Mockito *et. al.*

```
@Service
public class ClinicServiceImpl implements ClinicService {

    @Autowired
    public void setVisitRepository(VisitRepository visitRepository) {
        this.visitRepository = visitRepository;
    }
    @Autowired
    public void setConfirmationService(ConfirmationService confirmationService) {
        this.confirmationService = confirmationService;
    }

    public void saveVisit(Visit visit) throws DataAccessException {
        visitRepository.save(visit);
        try {
            confirmationService.sendConfirmationMessage(visit);
        } catch (Throwable t) {
            log.error("Failed to send confirmation message", t);
        }
    }
    ...
}
```

Test Doubles in Java: Mockito *et. al.*

```
public interface VisitRepository {  
    void save(Visit visit) throws DataAccessException;  
    ...  
}  
  
public interface ConfirmationService {  
    public void sendConfirmationMessage(Visit visit);  
}
```

Test Doubles in Java: Mockito *et. al.*

```
@Test
public void testSaveVisitSendsConfirmation() {
    VisitRepository visitStub = mock(VisitRepository.class);
    ConfirmationService confirmationMock = mock(ConfirmationService.class);
    ClinicServiceImpl service = new ClinicServiceImpl();
    service.setVisitRepository(visitStub);
    service.setConfirmationService(confirmationMock);
    service.saveVisit(visit);
    verify(confirmationMock).sendConfirmationMessage(visit);
}
```

Mocks using Groovy Closures

- A Closure may be used as an implementation of a Java interface method
- A Map of closures may be used as implementations of different methods in an Interface

```
def visitStub = {true} as VisitRepository
def confirmedVisit
def confirmationMock =
    [sendConfirmationMessage: { v -> confirmedVisit = v }]
    as ConfirmationService
...
assert confirmedVisit == visit
```

Example – Groovy Interactions

- The `testSaveVisitSendsConfirmation()` test in `org.springframework.samples.petclinic.service.ClinicServiceImplTest` shows an example of managing interactions in `ClinicServiceImpl`:
 - Stub out the `VisitRepository`
 - Mock `ConfirmationService`, and verify that a confirmation is sent on saving a visit

Working with Exceptions

- Unexpected exceptions thrown during execution of a test will be caught by the JUnit framework and reported as Errors.
- A Test method must declare that it throws any checked exceptions that the Unit under Test may throw. If there are several checked exceptions that may occur, it is perfectly valid for a test method to declare throwing `java.lang.Exception`.
- Expected exceptions (exceptions that the test is expecting the Unit under Test should throw in a certain situation) are expressed using the `@Test(expected=ExpectedException.class)` attribute

```
@Test(expected=NastyException.class)
public void doSomethingNastyTest() {
    SomeUnit target = new SomeUnit();
    target.doSomethingNasty();
}
```

Working with Exceptions (Contd.)

- Or using the following idiom:

```
SomeUnit unitUnderTest = new SomeUnit();

try {
    unitUnderTest.doSomethingNasty();
    fail("NastyException expected");
} catch (NastyException expected) {
    // Expected
}

assertTrue("Invariant violated", unitUnderTest.isValid());
```

In Groovy, we can do better:

```
SomeUnit unitUnderTest = new SomeUnit()  
  
shouldFail(NastyException) {  
    unitUnderTest.doSomethingNasty()  
}  
  
assert unitUnderTest.valid, "Invariant violated"
```

Simulating exceptional situations

```
public void saveVisit(Visit visit) throws DataAccessException {  
    ...  
    visitRepository.save(visit);  
    ...  
}  
  
@Test  
public void testSaveVisitThrowsDataAccessException() {  
    def visitStub = {  
        throw new DataIntegrityViolationException("Oops")  
    } as VisitRepository  
    ...  
}
```

Example – Simulating Exceptions

- The `testSaveVisitThrowsDataAccessException()` test in `org.springframework.samples.petclinic.service.ClinicServiceImplTest` shows an example of simulating an exception from a collaborator for `ClinicServiceImpl`:
 - Mock the `VisitRepository` to throw an exception
 - Verify that a `DataAccessException` is passed through from the `VisitRepository`

Injecting collaborators

```
public class ClinicServiceImpl implements ClinicService {  
  
    private static Logger log =  
        LoggerFactory.getLogger(ClinicServiceImpl.class);  
  
    ...  
  
    public void saveVisit(Visit visit) throws DataAccessException {  
        calculatePrice(visit);  
        visitRepository.save(visit);  
        try {  
            confirmationService.sendConfirmationMessage(visit);  
        } catch (Throwable t) {  
            log.error("Failed to send confirmation message: " +  
                t.getMessage());  
        }  
    }  
}
```

Injecting collaborators: Java

```
Logger loggerMock = mock(Logger.class);
ClinicServiceImpl service = new ClinicServiceImpl(...);

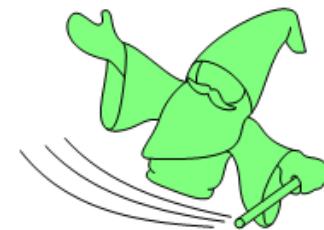
ReflectionUtils.setStaticAttribute(ClinicServiceImpl.class,
    "log", loggerMock);

service.saveVisit(visit);

verify(loggerMock).error(anyString(), any(Throwable.class));
```

Meta-programming: Meta Object Protocol

- Meta-programming: ≈ "programming the program"
 - enables extensions to a program at runtime
- Groovy gives every class or instance a `metaClass` property, which allows you to
 - Add methods and properties
 - Intercept missing methods
- Extremely powerful mechanism, which provides the foundation for DSLs and Builders



The MetaClass

- Obtain a Class' MetaClass with:
`def metaClass = String.metaClass`
- Obtain an object's MetaClass with:
`String s = "Hello"`
`def metaClass = s.metaClass`
- The MetaClass provides information about the Class or object:
`String.metaClass.methods.each {println it.name}`
`String.metaClass.properties.each {println it.name}`

ExandoMetaClass

- The MetaClass is actually an *extensible* Metaclass (a.k.a. ExandoMetaClass), which allows us to dynamically add methods, properties and fields:

```
class Dog {  
    private String secret = "...";  
}  
Dog.metaClass.bark = { "Woof!" }  
println new Dog().bark()  
def dog = new Dog()  
dog.metaClass.getBreed = { "Poodle" }  
println dog.breed  
Dog.metaClass.setAttribute(dog, "secret", "42")
```

ExandoMetaClass

- Add static methods using the static prefix:

```
Dog.metaClass.static.create = { new Dog() }
```

- Add constructors using the constructor prefix:

```
Dog.metaClass.constructor = {n -> new Dog(name:n)}
```

Exercise 3 – Test Doubles

- Implement the `testSaveVisitLogsConfirmationError()` test method in `org.springframework.samples.petclinic.service.ClinicServiceImplTest` to verify that an exception thrown from the `ConfirmationService` collaborator is properly handled by logging the exception
 - Create a Mock object for the logger
 - Inject the logger using `metaClass.setAttribute`

Meta Hooks

- The MetaClass also allows intercepting and overriding key dynamic method dispatch hooks:
 - invokeMethod()
 - set/getProperty()
 - methodMissing()
 - propertyMissing()

Overriding methodMissing

- Overriding the methodMissing method allows for dynamic creation of capabilities and behavior:

```
class Dog {}  
Dog.metaClass.methodMissing = {String name, args ->  
    Dog.metaClass."$name" = {Object[] varargs->  
        "dynamic $name"  
    }  
    "dynamic $name"  
}  
assert new Dog().howl() == "dynamic howl"  
assert new Dog().crawl() == "dynamic crawl"
```

Overriding methodMissing

- Overriding the methodMissing method allows for dynamic creation of capabilities and behavior
- Typical idiom when overriding the methodMissing: Intercept, cache a dynamically created method, then invoke it:

```
class Dog {}  
Dog.metaClass.methodMissing = {String methodName, args ->  
    Dog.metaClass."$methodName" = {Object[] varargs->  
        "cached $methodName"  
    }  
    "dynamic $methodName"  
}  
assert new Dog().howl() == "dynamic howl"  
assert new Dog().howl() == "cached howl"  
assert new Dog().crawl() == "dynamic crawl"
```

Builders

- The need to produce and work with hierarchical (tree) structures are everywhere
 - XML documents
 - Configuration files
 - Ant tasks
 - GUI Components
- The Builder pattern can be used to construct such hierarchical data structures
- Trivial and elegant to implement in Groovy using `methodMissing()`, closures and chained method calls

XML Builder example

```
def builder = new groovy.xml.MarkupBuilder()  
builder.employees {  
    employee(name:'Björn') {  
        address(street:'Vallhamnsgatan', city:'Västra Frölunda')  
        phone('0733-519173')  
    }  
}
```

produces

```
<employees>  
    <employee name='Björn'>  
        <address street='Vallhamnsgatan' city='Västra Frölunda' />  
        <phone>0733-519173</phone>  
    </employee>  
</employees>
```

ObjectGraphBuilder example

```
def builder = new ObjectGraphBuilder()
builder.classNameResolver =
    "org.springframework.samples.petclinic.model"
visit = builder.visit(
    date: DateTime.now(),
    description: "visit description",
    { pet(name: "a pet",
        { petType(name: "type") },
        { owner(firstName: "firstName",
            lastName: "lastName",
            address: "address",
            city: "city",
            email: "name@gmail.com") } ) }
)
```

Parsing and navigating XML

- Given the following XML:

```
def xml = """<employees>
    <employee name='Björn'>
        <phone>0733-519173</phone>
    </employee>
    <employee name='Johan'>
        <phone>0733-519175</phone>
    </employee>
</employees>"""
```

- Parsing the XML and navigating the object graph using GPath is simple:

```
def root = new XmlSlurper().parseText(xml)
println root.employee[1].phone.text()
println root.employee.findAll{ it['@name'] =='Björn' }
```

Example – Verifying XML content

- Consider
 - org.springframework.samples.petclinic.service.ConfirmationServiceImpl and its dependency on the org.springframework.samples.petclinic.service.MessageSender collaborator
- The org.springframework.samples.petclinic.service.ConfirmationServiceImplTest shows an example of verifying XML content:
 - verifies that sendConfirmationMessage(Visit visit) calls its MessageSender collaborator with the correct recipient and a properly formatted XML message representing the visit

Typical Unit test scenario – The Three A's

1. Arrange - Instantiate Unit under Test and set up test data
2. Act - Execute one or more methods on the Unit Under Test
3. Assert - Verify the results

```
@Test
public void testWithdraw() {
    AccountImpl account = new AccountImpl("1234", 2000); // Arrange
    account.withdraw(300); // Act
    assertEquals(1700, account.balance()); // Assert
}
```

Spock



- A Groovy-based testing framework
- Fully compatible with JUnit
- Utilizes the power of Groovy to
 - Reduce the lines of test code
 - Make tests more readable
 - Turn tests into specifications

Spock: Basics

```
import spock.lang.*  
  
class MyFirstSpecification extends Specification {  
    // fields  
    // fixture methods  
    // feature methods  
    // helper methods  
}
```

Spock: Fields

```
class MyFirstSpecification extends Specification {  
    def obj = new ClassUnderSpecification()  
    def coll = new Collaborator()  
    @Shared res = new VeryExpensiveResource()  
    static final PI = 3.141592654  
    // fixture methods  
    // feature methods  
    // helper methods  
}
```

Spock: Fixture Methods

```
import spock.lang.*  
  
class MyFirstSpecification extends Specification {  
    // fields  
    def setup() {}          //run before every test  
    def cleanup() {}        //run after every test  
    def setupSpec() {}      //run before first test  
    def cleanupSpec() {}   //run after last test  
    // feature methods  
    // helper methods  
}
```

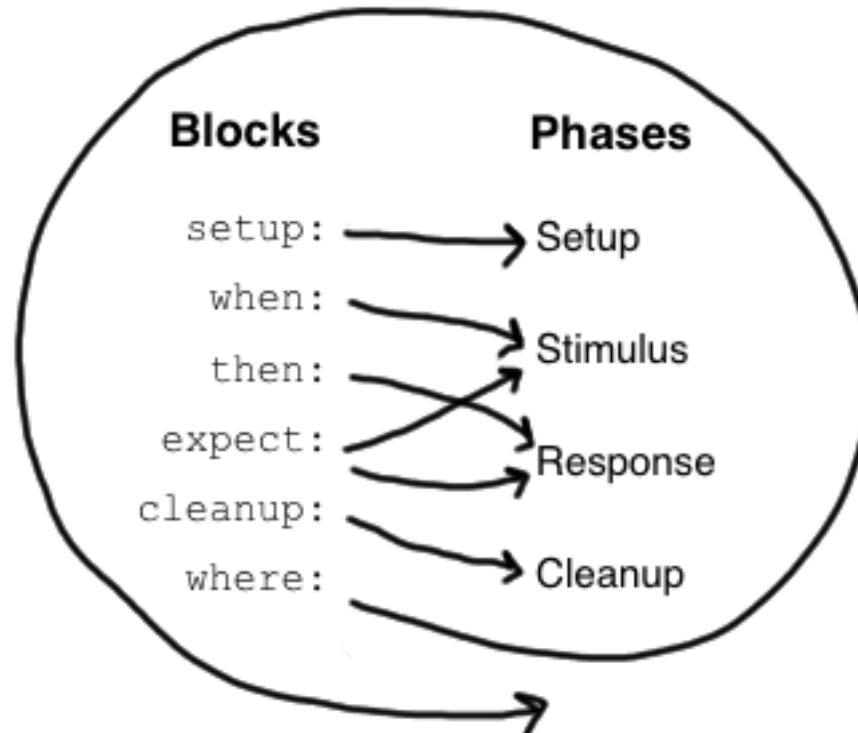
Spock: Feature Methods

```
class MyFirstSpecification extends Specification {  
    def "specification for a feature"() {  
        // blocks go here  
    }  
}
```

- A feature method consists of four phases:
 - Optional: Set up the feature's fixture
 - Provide a ***stimulus*** to the system under specification
 - Describe the ***response*** expected from the system
 - Optional: Clean up the feature's fixture

Spock: Feature Blocks

- A **block** is a section of a feature method, prefixed by a label that describe the purpose of the section:
 - setup:
 - when:
 - then:
 - expect:
 - cleanup:
 - where:



Spock: Setup

```
class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        setup:  
            def stack = new Stack()  
            def elem = "push me"  
            ...  
    }  
}
```

Spock: Given alias

```
class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        given:  
            def stack = new Stack()  
            def elem = "push me"  
            ...  
    }  
}
```

Spock: Implicit setup

```
class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        def stack = new Stack()  
        def elem = "push me"  
        ...  
    }  
}
```

Spock: Additional documentation

```
class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        given: "an empty stack"  
        def stack = new Stack()  
        and: "an element that can be pushed"  
        def elem = "push me"  
        ...  
    }  
}
```

Spock: Triggers and Conditions

```
class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        given: "an empty stack"  
        ...  
        when: "an element is pushed"          // stimulus  
        stack.push(elem)  
        then: "stack is not empty"           // response  
        !stack.empty  
        and: "pushed element is on top"  
        stack.peek() == elem  
    }  
}
```

Spock: Exception Conditions

```
Class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        given:  
        ...  
        when: "popping an empty stack"  
        stack.pop()  
        then: "an exception is thrown"  
        EmptyStackException e = thrown()  
        e.cause == null  
    }  
}
```

Spock: expect

```
class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        given: "a stack with one element"  
        def stack = new Stack()  
        def elem = "push me"  
        stack.push(elem)  
        expect: "pop returns that single element"  
        stack.pop() == elem  
    }  
}
```

Spock: corresponding when/then

```
class StackSpecification extends Specification {  
    def "specification for a stack"() {  
        given: "a stack with one element"  
        def stack = new Stack()  
        def elem = "push me"  
        stack.push(elem)  
        when: "pop is called"  
        def popped = stack.pop()  
        then: "that single element is returned"  
        popped == elem  
    }  
}
```

Spock: Data-driven testing

- A feature method may be *parameterized* using a where block:

```
def "specification for a feature"() {  
    expect:  
        x == someFunction(y)  
    where:  
        x << xDataProvider  
        y << yDataProvider  
}
```

Spock: Data-driven testing example

```
def "specification for square"() {  
    expect:  
        y == x * x  
    where:  
        x << [1,2,3,4]  
        y << [1,4,9,16]  
}
```

Multi-parameterizations

```
def "specification for square"() {  
    expect:  
        y == x * x  
    where:  
        [x, y] << [[1,1], [2,4], [3,9], [4,16]]  
}
```

Alternative syntax

```
def "specification for square"() {  
    expect:  
        y == x * x  
    where:  
        x | y  
        1 | 1  
        2 | 4  
        3 | 9  
        4 | 16  
}
```

More elaborate example

```
@Unroll
def "frameworks #x, #y and #z are compatible"() {
    expect:
    areCompatible(x, y, z)
    where:
    [x, y, z] << [[["junit", "mockito", "java"],
                      ["junit", "mockito", "groovy"],
                      ["junit", "easymock", "java"],
                      ["junit", "easymock", "groovy"],
                      ["spock", "mockito", "java"],
                      ["spock", "mockito", "groovy"],
                      ["spock", "easymock", "java"],
                      ["spock", "easymock", "groovy"]]]}
```

... or more elegantly

```
@Unroll
def "frameworks #x, #y and #z are compatible"() {
    expect:
    areCompatible(x, y, z)
    where:
    [x, y, z] << [
        ["junit", "spock"],
        ["mockito", "easymock"],
        ["java", "groovy"]
    ].combinations()
}
```

Exercise 4 – Spock Data-driven test

- Implement the
"price for pet with age #age and #noOfVisits visits is #price"()
feature method in the
org.springframework.samples.petclinic.util.PriceCalculatorSpec
specification

Spock: Interaction-based testing

- Spock provides built-in support for stubbing/mocking interactions using a highly readable syntax:

given:

```
def messageSender = Mock(MessageSender)
```

```
messageSender.isActive() >> true
```

when:

```
messageSender.setFormat("xml")
```

```
service.sendConfirmationMessage(visit)
```

then:

```
1 * messageSender.sendMessage(owner.email, _)
```

Exercise 5 – Spock Interactions

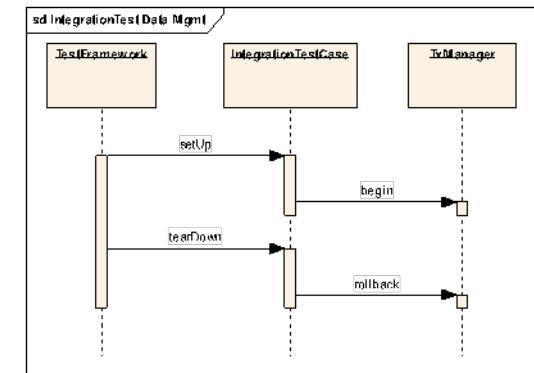
- Implement the "a confirmation message ..."() feature method in the org.springframework.samples.petclinic.service.ConfirmationServiceImplSpec specification

Integration Tests

- An Integration Test is any test which tests a logical unit *together with other units that it depends on*, such as other software units but more frequently external resources such as Databases or Message Queues.
- Thus the integration tests share many of the characteristics of Unit Tests, but the granularity is much bigger.
- Due to the performance costs in accessing external resources, the integration tests usually takes much longer time to execute.

Test Data, concurrency and repeatability

- Integration tests which have side effects (i.e. which affects persistent data) are problematic:
 - Modifying data which other tests may depend on, may cause subsequent test failures
 - Several instances of tests which uses the same data may run concurrently, which may cause test failures
- Transaction demarcation is a common idiom to protect test data from modification:
 - Start a transaction in `@Before setUp()`
 - Rollback the transaction in `@After tearDown()`



Working with SQL data in Java

- The JDBC api is far to low-level. Hence add-on libraries such as DbUnit or Spring-test are necessary.

```
static boolean testDataSetup = false;  
@BeforeTransaction  
public void setupTestData() {  
    if (!testDataSetup) {  
        testDataSetup = true;  
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);  
        Resource script =  
            new ClassPathResource("/db/h2/populateDb.sql");  
        JdbcTestUtils.executeSqlScript(jdbcTemplate, script, true);  
    }  
}
```

DbUnit example

```
<?xml version='1.0' encoding='UTF-8'?>
<dataset>
    <AREA_TYPE AREA_TYPE_ID='11' NAME='test' SORT_ORDER='11' VERSION='1' />
    <AREA AREA_ID='10' NAME ='testParent' AREA_TYPE_ID = '11' Version = '1' />
    <AREA AREA_ID='11' NAME ='testChild' AREA_TYPE_ID = '11'
          PARENT_AREA_ID = '10' Version = '1' />

    <AREA_BUDGET AREA_BUDGET_ID='11' BUDGETED_WIP='50' MIN_WIP='0'
                  MAX_WIP='100' VERSION='1' />

    <REGISTRATION_POINT REGISTRATION_POINT_ID = '1'
                          MACHINE_NUMBER='55' VERSION='1' />
</dataset>

IDataSet dataSet = new FlatXmlDataSet(new FileInputStream(FILE));
DatabaseOperation.CLEAN_INSERT.execute(connection, dataSet);
```

Working with SQL data in Groovy

```
import groovy.sql.Sql

sql = Sql.newInstance( 'jdbc:hsqldb:mem:petclinic',
    'sa', '', 'org.hsqldb.jdbcDriver' )
sql.eachRow( 'select * from OWNER' ) {
    println "$it.id -- ${it.firstName} --"
}

def sql = new Sql(dataSource)
def insert = "INSERT INTO owners (id, first_name) VALUES (?, ?)"
sql.executeUpdate(insert, [owner.id, owner.firstName])
```

Example – Working with SQL

- The `org.springframework.samples.petclinic.service.ClinicServiceJpaTest` shows an example of working with SQL:
 - The `setupTestData()` method inserts test data from an external file
 - The `insertVisit()` test method uses an `insert` SQL statement to insert a test Pet, and an SQL query to verify the inserted Visit entity

Parsing and navigating JSON

- Given the following Json string:

```
def json = '{"employees" : [{"name" : "Björn", "phone" :  
"0733-519173"}, {"name" : "Johan", "phone" : "0733-519175"}]}'
```

- Parsing the Json and navigating the object graph is simple:

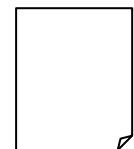
```
def root = new JsonSlurper().parseText(json)  
println root.employees[1].name  
println root.employees.findAll{ it.name =='Björn' }
```

Example – Working with Json

- The `org.springframework.samples.petclinic.api.RestControllerPetTest` shows an example of working with Json:
 - The constructor uses the `metaClass` to inject a `toJson()` method into all Entities
 - `JsonSlurper` is used to parse and navigate Json objects used in assertions

Acceptance Tests

- Tests a system or part of a system from the outside, in terms of observable behavior
- Typically structured around a comprehensible chunk of system functionality
- Ideally written by the customer
- Either written in terms of User Interface actions, or in terms of “words” and “facts”
- Ideally run near 100% correct at the end of a release/project



Business people wants to express
the required behaviour, not
necessarily write tests



Then we should ...

State required behaviour in a precise enough way, using the ***vocabulary of the business***

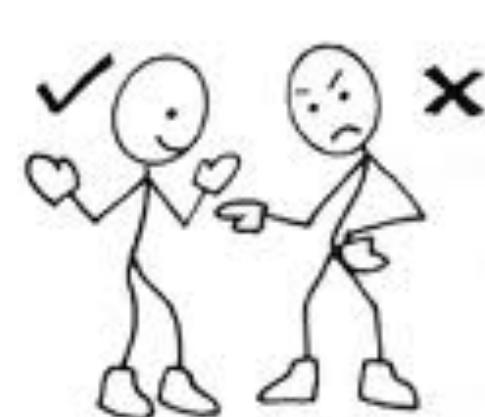


Yes,
I want to **STOP** my dog from,
being aggressive, chewing,
digging, **barking**, jumping,
biting, whining

And **All** other behavior problems!

Then we should ...

State required behaviour using
examples



Acceptance Test Driven Development, a.k.a. Behavior Driven Development, a.k.a. Test Driven Requirements

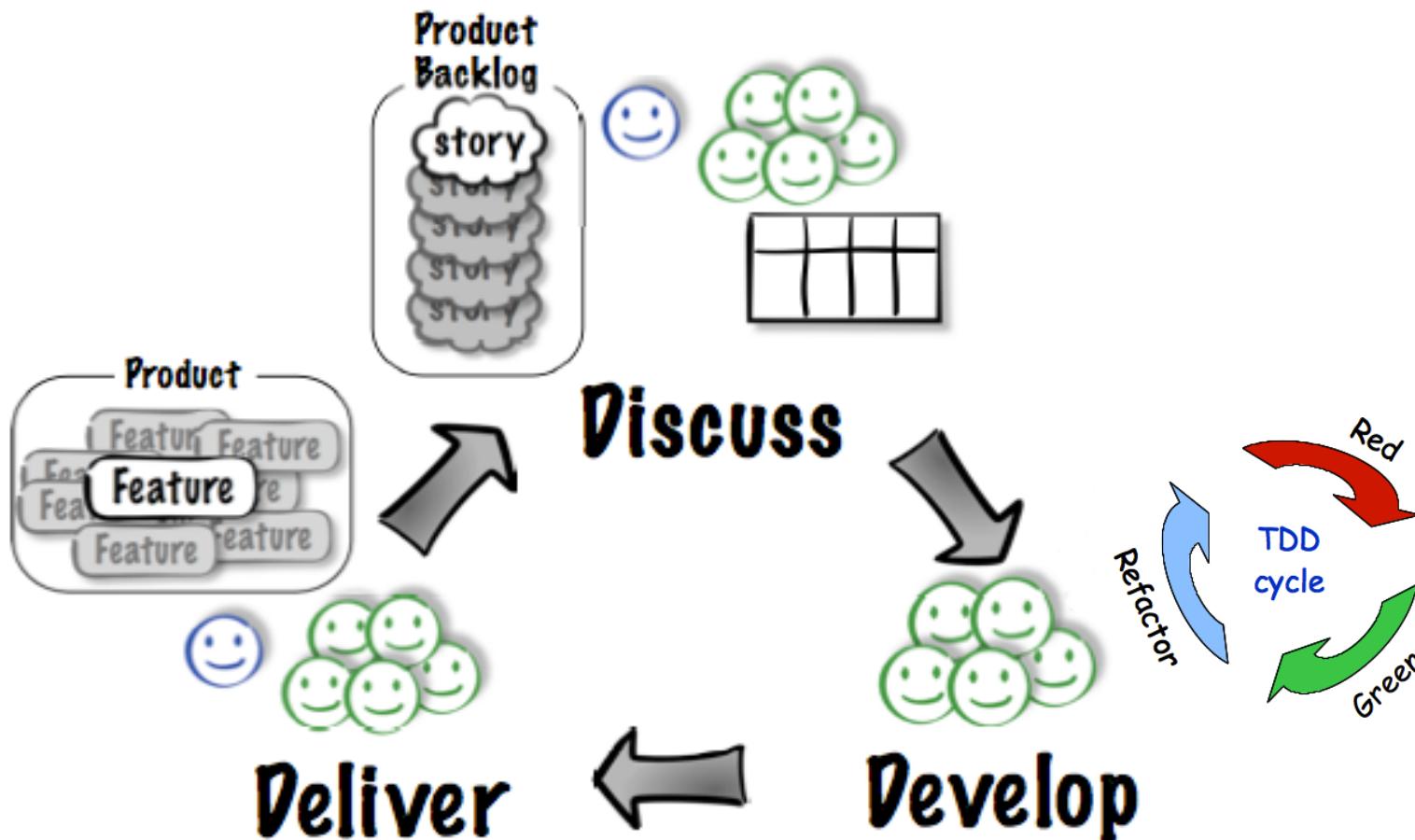


Image courtesy Elisabeth Hendrickson, www.qualitytree.com

FitNesse: A standalone Wiki and FIT runner

- Collaborative Testing and Documentation Tool, providing a ***very simple*** way to
 - create documents
 - specify tests
 - execute tests



Minimal FitNesse example: Test table

```
! |cadec.DivisionFixture|  
| numerator |denominator|quotient?|  
| 10        | 2          | 5.0      |  
| 12.6      | 3          | 4.2      |  
| 100       | 3          | ~=33.3   |
```

cadec.DivisionFixture		
numerator	denominator	quotient?
10	2	5.0
12.6	3	4.2
100	3	~=33.3

Minimal example: SLIM Fixture

```
package cadec

class DivisionFixture {
    double numerator
    double denominator
    double quotient() {
        numerator / denominator
    }
}
```

Minimal FitNesse example: Test execution

The screenshot shows a web browser window with the following details:

- Address Bar:** localhost:9123/FrontPage.DivisionTest?test
- Toolbar:** Shows various application icons.
- Bookmark Bar:** Apps, Wikipedia, Toodledo, Diverse, Callista, Intygsprojektet, Other Bookmarks.
- Page Content:**
 - FrontPage DivisionTest**: The title of the page.
 - FitNesse Logo**: A gauge-style logo with "FitNesse" written on it.
 - Buttons:** Tests Executed OK (highlighted), Test, Edit, Add, Tools.
 - Assertions:** 3 right, 0 wrong, 0 ignored, 0 exceptions (0,682 seconds)
 - Data Table:** A table showing division results.

cadec.DivisionFixture		
numerator	denominator	quotient?
10	2	5.0
12.6	3	4.2
100	3	33.3333333333336~=33.3
 - Page Footer:** 26 jan, 2014 14:33, Front Page | User Guide, root (for global !path's, etc.)

Exercise 6

- Implement and run the test for the minimal Division example!
 - Start FitNesse using Run -> 'groovy-tdd-cadec FitNesse'
 - Start a browser, and navigate to <http://localhost:9123>
 - Click 'Add' -> 'Test Page', name the page DivisionTest
 - » Add the table for the test
 - Implement the cadec.DivisionFixture
 - Run the test

FitNesse: Wiki pages and Subpages

- FitNesse is a Wiki Web Server
 - FIT tests lives on a Wiki page
 - FitNesse Wiki pages have hierarchical structure
- FitNesse Wiki pages have additional metadata (a.k.a. “Properties”)



FitNesse: Wiki markup

- Wiki words are in CamelCase: `ThisIsAnExample`
- Hierarchy is expressed using . notation, or > and < for relative links:
 - `.RootPage.SubPage.AnotherSubPage`
 - `>SubPage`
 - `<ParentPage`
- Simple markup for formatting
 - `'''bold'''` -> **bold**, `''italic''` -> *italic*
 - `!c` centers a line, `!1` gives first heading `!2` second heading
 - `|cell 1|cell 2|` specifies a table

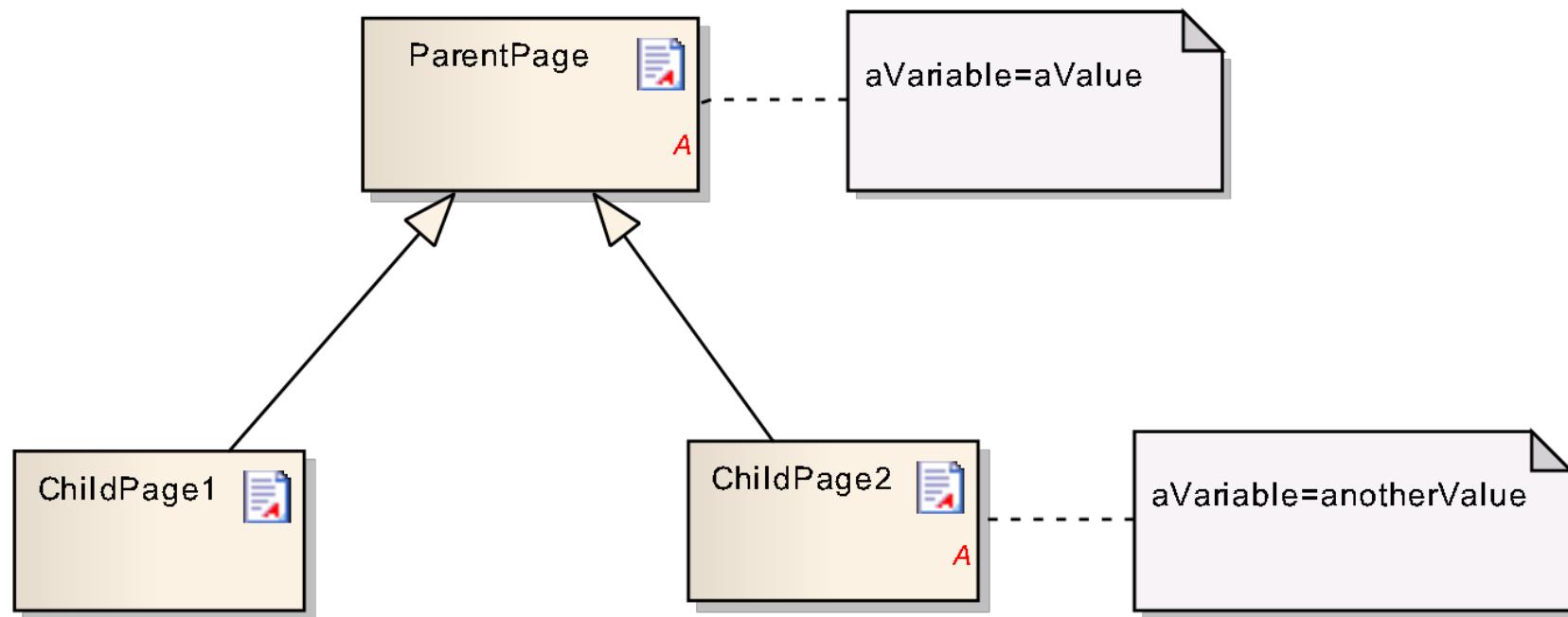
FitNesse: Wiki markup (cont)

- Escape wiki words using ! - - !:
! -ThisIsNotAWikiWord- !
- Escape all content in a table using !:
! |a.java.ClassName|
- Special markup ‘widgets’:
 - **!path**
 - **!contents -R2**
 - **!define**

FitNesse variables

- FitNesse **variables** allow variable content to be specified once, then used in multiple tests
- FitNesse variables are inherited from parent pages, if not defined explicitly (a.k.a. *default inheritance*)
- Variable Definition
 - **`!define name {value}`**
 - **`!define name (value)`**
 - **`!define name [value]`**
- Variable Usage
 - **`$(name)`**

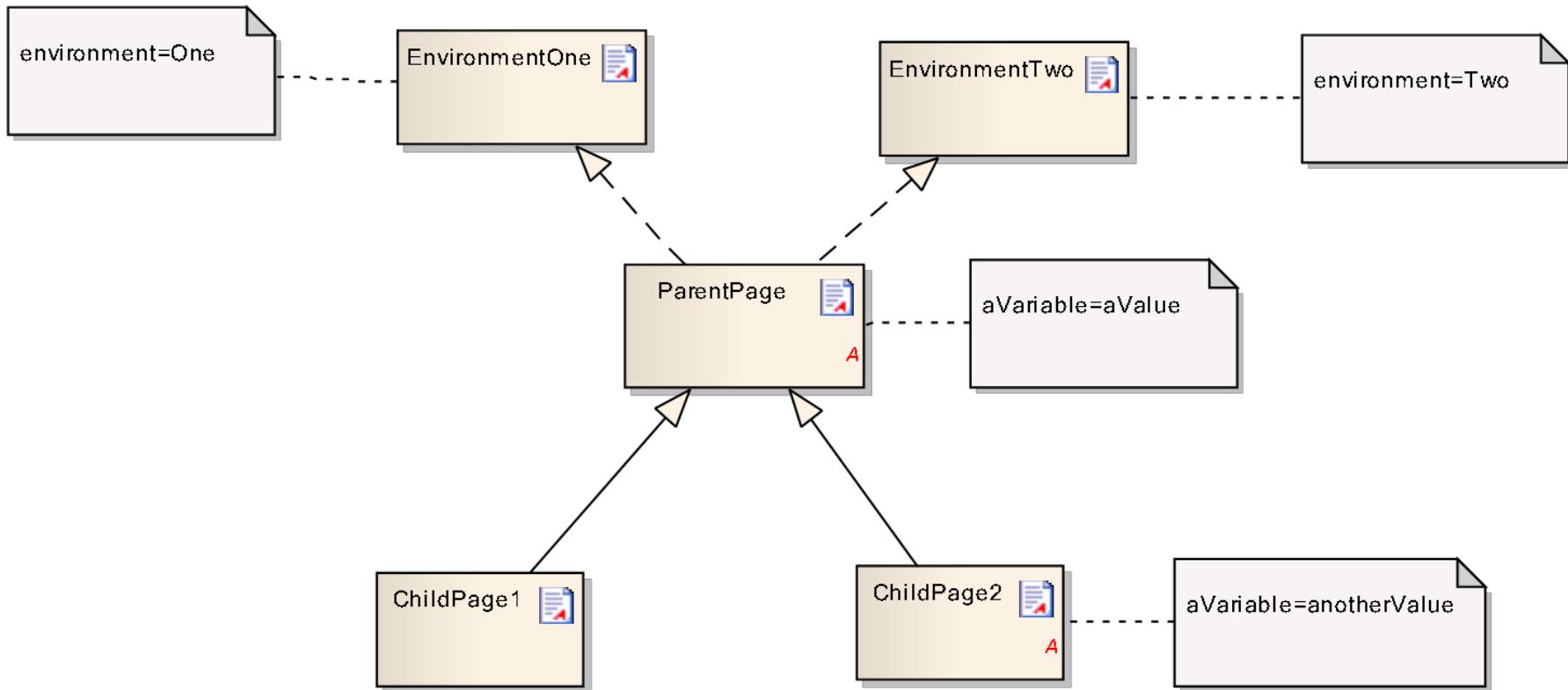
Hierarchical structure of variables



Symbolic links

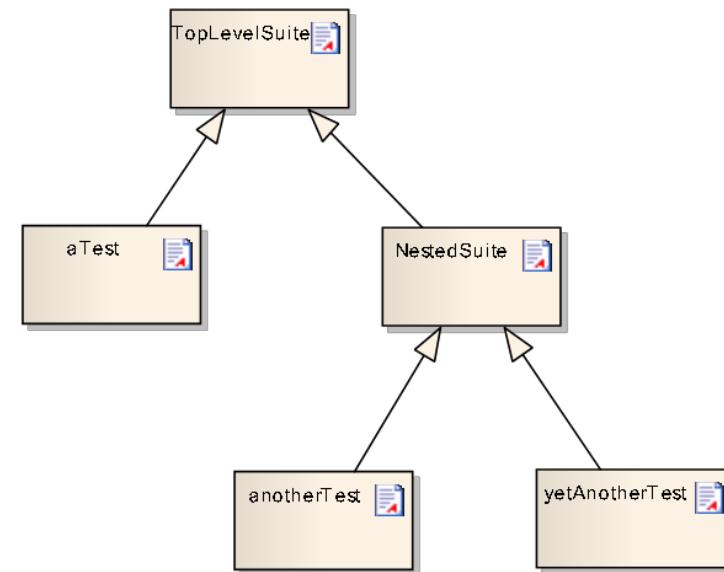
- Page structure forms a proper hierarchy (i.e. a tree)
- Sometimes, variable information exists in several, crosscutting dimensions
 - E.g. environmental differences (Test environment vs. Production environment)
- FitNesse ***Symbolic Links*** provide a way to introduce multiple inheritance between pages

Symbolic Links



Test Suites

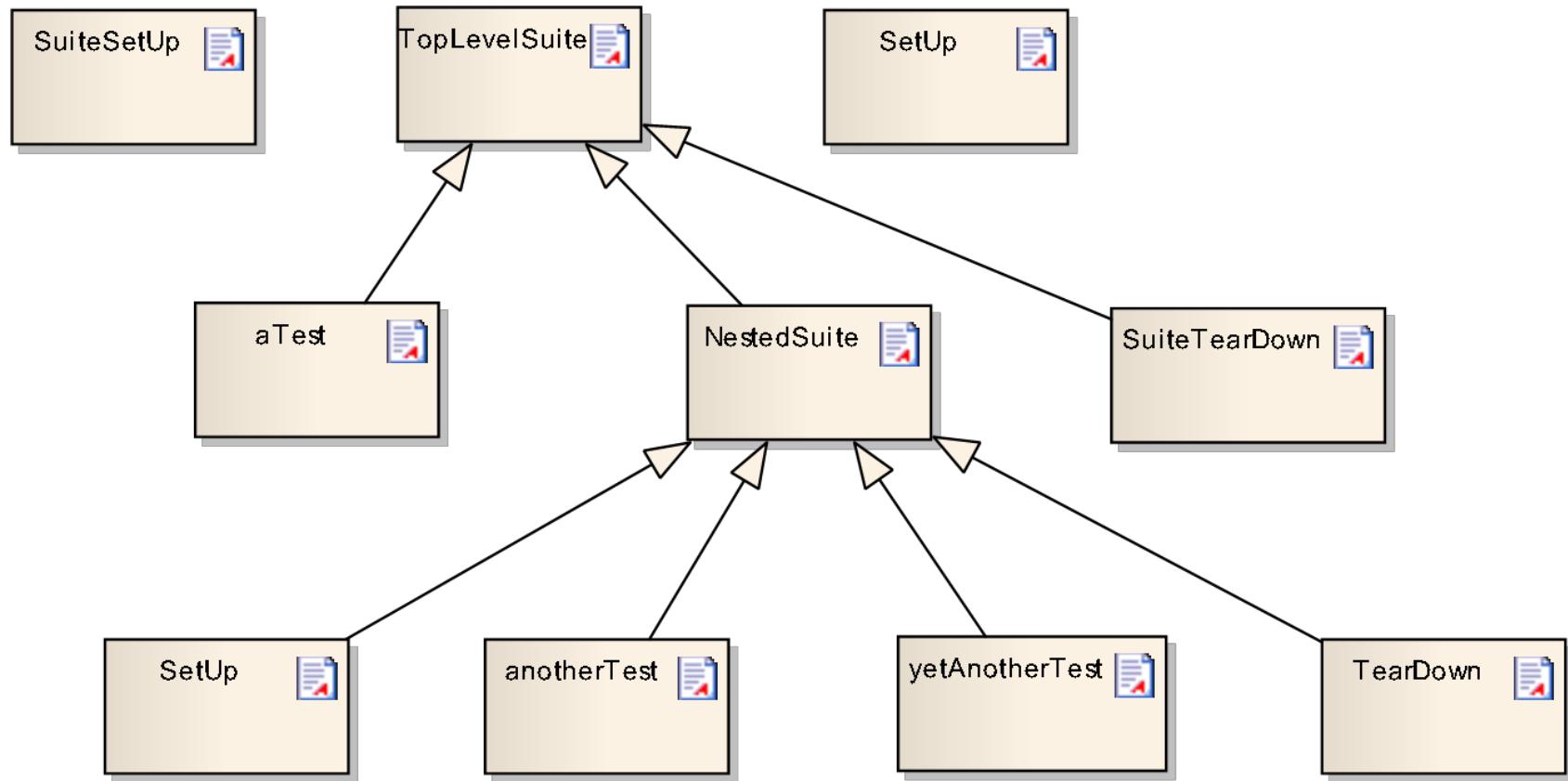
- A Test Suite is a page that has children containing tests (or other suites)
- Test Suites can be executed, just as individual test pages. Doing so will execute each contained test page in sequence



SetUp and TearDown

- If a sibling page ***SetUp*** exists for a Suite or a Test (or any of its parents), that SetUp page will automatically be included ***before*** the actual test(s)
- Correspondingly, if a sibling page ***TearDown*** exists for a Suite or a Test (or any of its parents), that TearDown page will automatically be included ***after*** the actual test(s)
- If a sibling page ***SuiteSetUp*** exists for a Suite, that SetUp page will automatically be included once ***before*** the actual suite
- Dito for ***SuiteTearDown***

SetUp and TearDown example



Example – Fixtures in Groovy

- The <http://localhost:9123/PetClinic.RestApiTests.GetOwner> test specification provides a full-blown example of using Groovy power features and api:s in implementing Acceptance Tests for a RESTful API:
 - The `org.springframework.samples.petclinic.slim.InsertOwners` and `org.springframework.samples.petclinic.slim.DeleteOwners` fixtures uses SQL statements to insert and delete test data before and after execution
 - The `org.springframework.samples.petclinic.slim.OwnerRestApiFixture` uses `groovyx.net.http.RESTClient` to issue REST requests and parsing resulting Json data

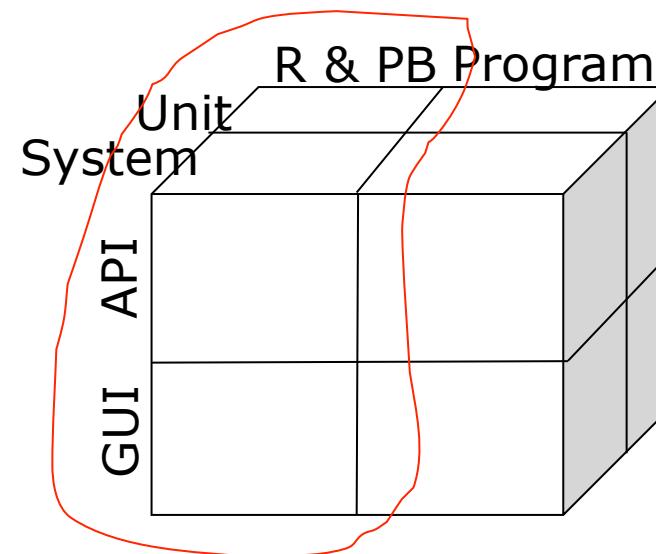
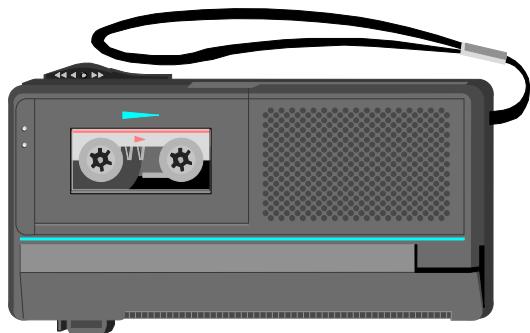
Testing Web User Interfaces is ...



... still a challenge

Record/Playback

- A Test Automation tool records “events” that make up a Test Case into a Test Script
- Events can be User Interface interactions or API calls
- The scripts can be played back later for regression testing

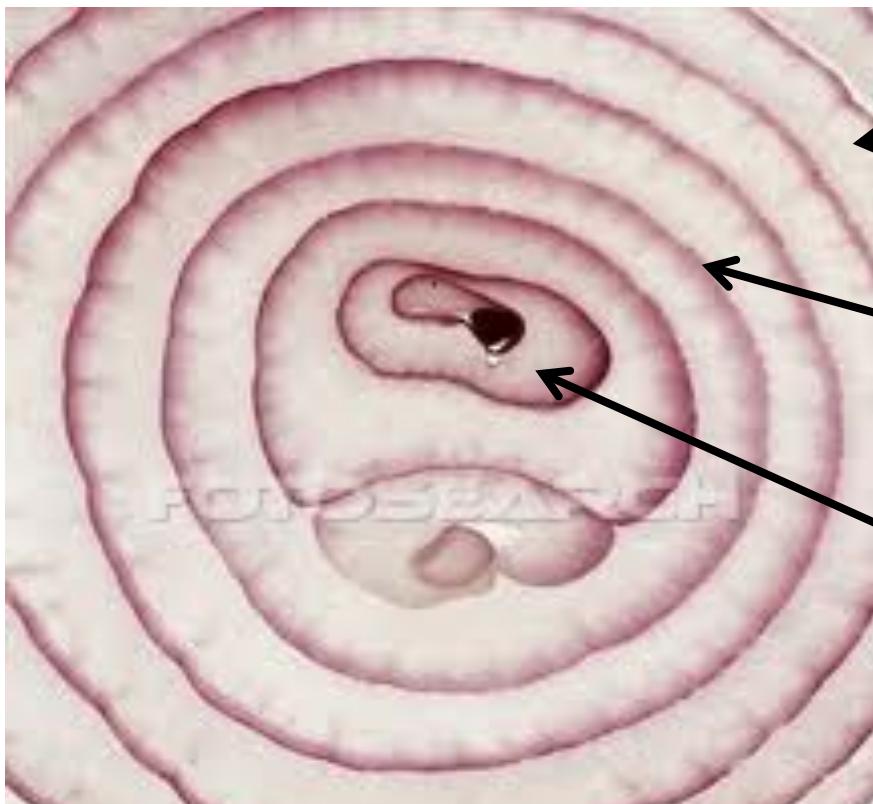


Drawbacks of Record/Playback approaches

- Tests tend to be fragile
- Maintenance tends to be expensive
- Complex, expensive tools
- **Tests cannot be pre-built**



Hence we need some layers of abstraction



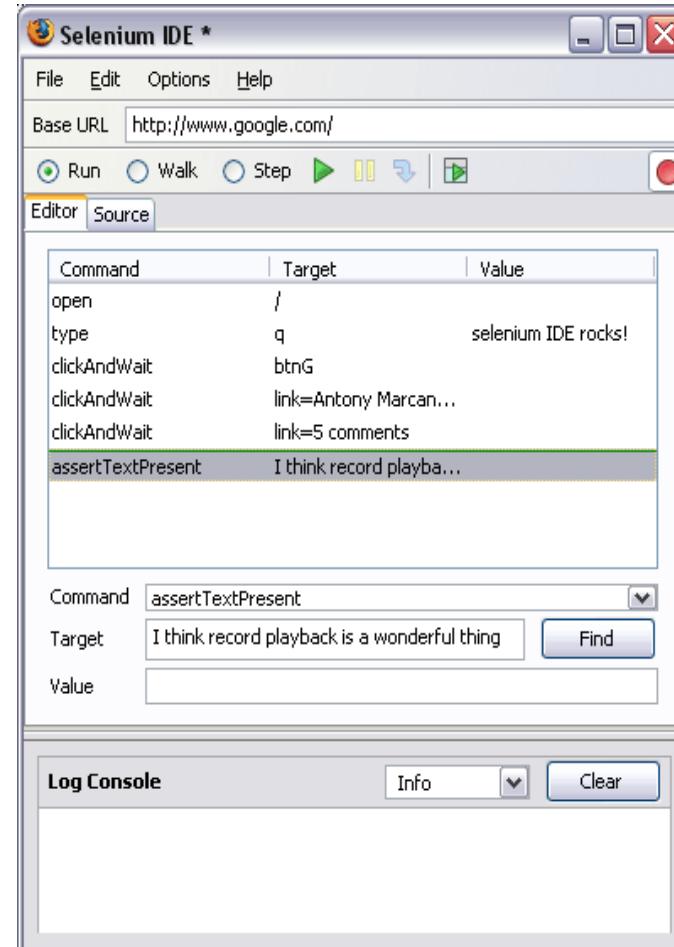
Specifications

DOM Abstraction

Browser Automation

Selenium: A framework for Browser Automation

- The Selenium engine executes tests **directly in a browser**, just as real users do.
- The Selenium 1.0 test engine is implemented in JavaScript, and thus run in most browser (Internet Explorer, Mozilla, Firefox, Opera, Safari ...) on most platforms.



Selenium + WebDriver = Selenium 2.0

- First production-ready version in July 2011
- Greatly enhanced model compared to Selenium 1.0
 - Object-based API for DOM Interaction
 - Browser Driver Implementations
 - Advanced User Interactions



At the heart of Selenium: *Selenese*

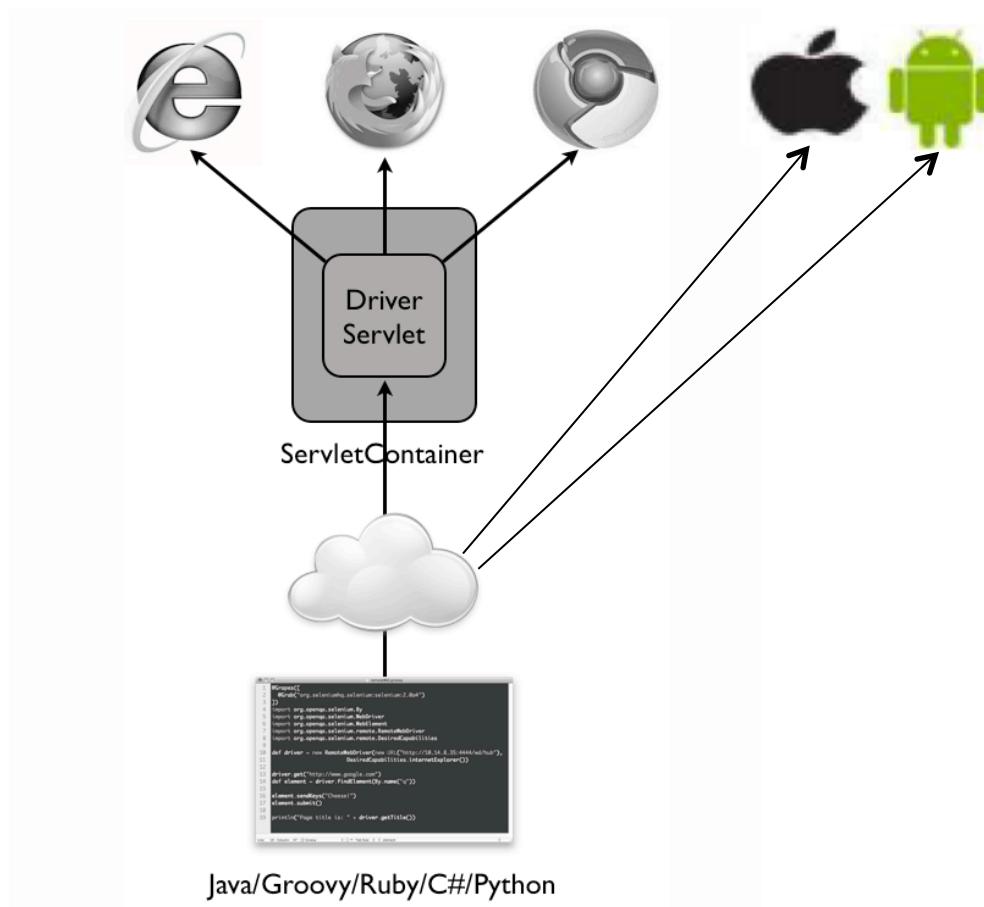
click	getHtmlSource	isVisible
close	getTitle	keyPress
createCookie	getValue	mouseOver
dragdrop	goBack	open
fireEvent	isElementPresent	refresh
getEval	isTextPresent	type

Browser Drivers

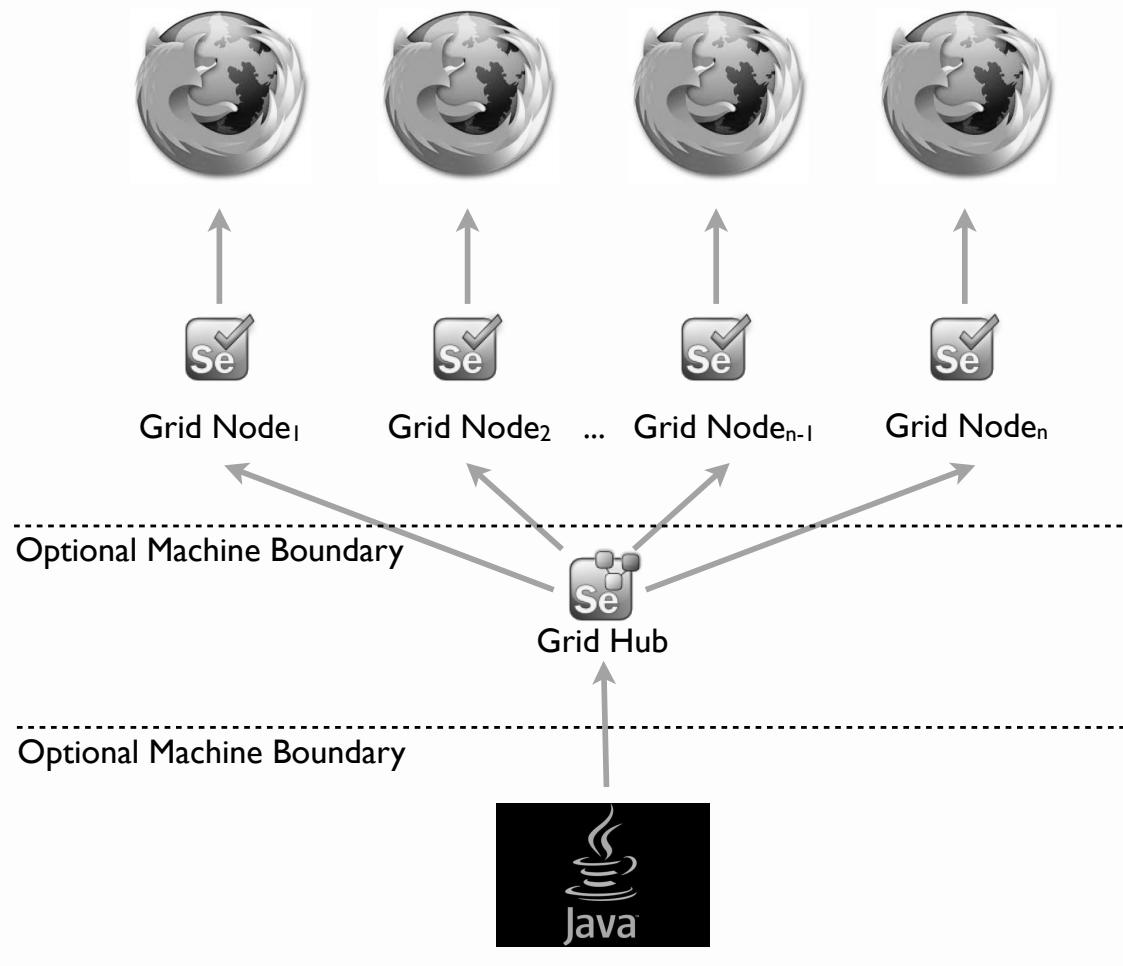
- Native implementations of the WebDriver API
 - Firefox
 - Chrome
 - Internet Explorer
 - Opera
 - Headless (HtmlUnit)
 - iOS
 - Android
 - ...



Selenium Remote Driver

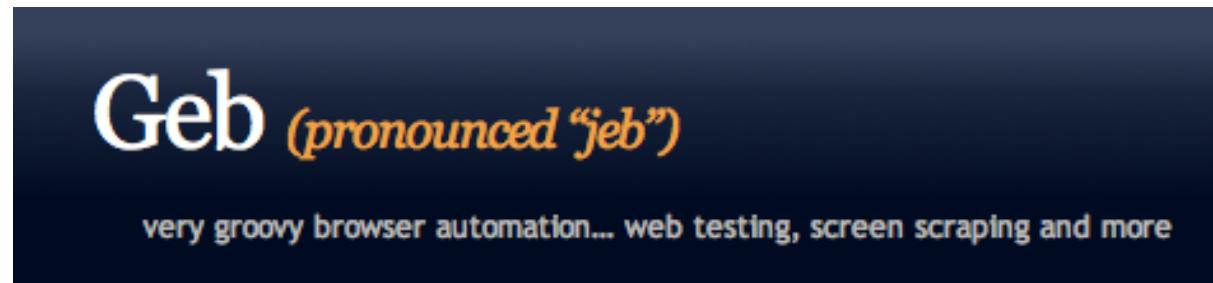


Selenium Grid



DOM Abstraction: Geb

- Brings together
 - The power of Selenium / Web Driver
 - The elegance of JQuery
 - The robustness of the Page Object pattern
 - The flexibility and pure joy of Groovy



Navigator API

- JQuery-like expressions provide a concise and effective way to navigate the DOM

```
$("p", text: contains("Timecard"))
$("input", name: "username").value("bjorn")
$("div.message").text()
```

Page Objects

- Use proper object-orientation techniques to avoid brittleness and duplication

```
$("input", name: "username").value("bjorn")
$("input", name: "password").value("secret")
$("input", name: "submit").click()
```

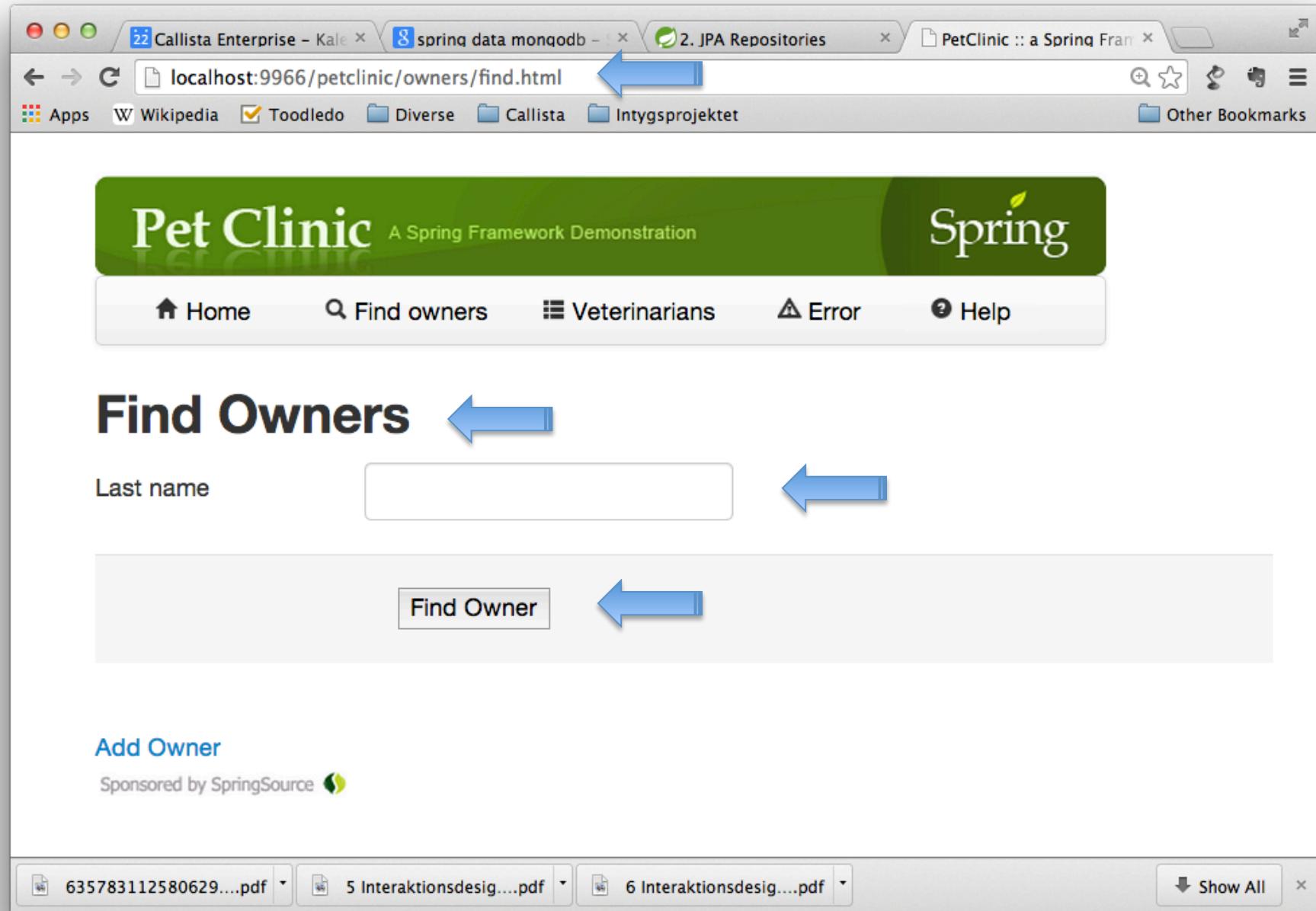
Don't do this

Page Objects

- Use proper object-orientation techniques to avoid brittleness and duplication

```
page.login("bjorn", "secret")
```

Do this instead



Geb Page definition

```
import geb.Page

class FindOwnersPage extends PetClinicPage {

    static url = "owners/find.html"

    static at = { $(("h2")).text() == "Find Owners" }

    static content = {
        lastName { $("input", name: "lastName") }
        search { $("submit") }
    }

    def findByLastName(name) {
        lastName.value(name)
        search.click()
    }
}
```

Geb Page definition: navigation

```
import geb.Page

class FindOwnersPage extends PetClinicPage {

    static url = "owners/find.html"

    static at = { $("h2").text() == "Find Owners" }

    static content = {
        lastName { $("input", name: "lastName") }
        search { $("submit") }
    }

    def findByLastName(name) {
        lastName.value(name)
        search.click()
    }
}
```

Geb Page definition: *at* predicate

```
import geb.Page

class FindOwnersPage extends PetClinicPage {

    static url = "owners/find.html"

    static at = { $(("h2")).text() == "Find Owners" }

    static content = {
        lastName { $("input", name: "lastName") }
        search { $("submit") }
    }

    def findByLastName(name) {
        lastName.value(name)
        search.click()
    }
}
```

Geb Page definition: logical *content*

```
import geb.Page

class FindOwnersPage extends PetClinicPage {

    static url = "owners/find.html"

    static at = { $("h2").text() == "Find Owners" }

    static content = {
        lastName { $("input", name: "lastName") }
        search { $("submit") }
    }

    def findByLastName(name) {
        lastName.value(name)
        search.click()
    }
}
```

Geb Page definition: *actions*

```
import geb.Page

class FindOwnersPage extends PetClinicPage {

    static url = "owners/find.html"

    static at = { $("h2").text() == "Find Owners" }

    static content = {
        lastName { $("input", name: "lastName") }
        search { $("submit") }
    }

    def findByLastName(name) {
        lastName.value(name)
        search.click()
    }
}
```

Spock Geb specification

```
class NavigationSpec extends GebSpec {

    def "first page is HomePage"() {
        when:
        go ""
        then:
        at HomePage
    }

    def "from Home page you can navigate to Vets page"() {
        given:
        to HomePage
        when:
        navigateToVetsPage()
        then:
        at VetsPage
    }
}
```

Exercise 7 – Geb Tests

- Implement navigation test scenarios for the Error page:
 - Implement the `url` and `at` properties for
`org.springframework.samples.petclinic.web.pages.ErrorPage`
 - Implement the
 - "from Home page you can navigate to Error page"() and
 - "from Error page you can navigate back to Home page"() feature methods in
`org.springframework.samples.petclinic.web.spec.NavigationSpec`

FitNesse script Fixture using Geb

```
class Navigation {  
  
    boolean firstPageIsHomePage() {  
        Browser.drive {  
            go ""  
            waitFor {  
                at HomePage  
            }  
        }  
        true  
    }  
  
    void navigateToFindOwnersPage() {  
        Browser.drive {  
            page.navigateToFindOwnersPage()  
        }  
    }  
    ...  
}
```

FitNesse specification using Geb

script	Navigation
ensure	first page is home page
	navigate to find owners page
ensure	at find owners page
	navigate to vets page
ensure	at vets page
	navigate to home page
ensure	at home page

Geb Module definition

```
import geb.Module

class OwnerRow extends Module {
    static content = {
        cell { i -> $("td", i) }
        name { cell(0).text() }
        address { cell(1).text() }
        city { cell(2).text() }
        telephone { cell(3).text() }
        pets { cell(4).text() }
    }
}
```

Using a Module

```
class OwnersPage extends PetClinicPage {

    static url = "owners.html"

    static at = { $("h2").text() == "Owners" }

    static content = {
        owners {
            $("#owners tbody tr").collect {
                module OwnerRow, it
            }
        }
    }
}
```

Example – Geb in action, driven by Spock and by FitNesse

- The `org.springframework.samples.petclinic.web.spec.FindOwnersSpec` specification provides a full-blown example of using Geb page with Spock:
 - The `org.springframework.samples.petclinic.web.pages.FindOwnersPage` and `org.springframework.samples.petclinic.web.pages.OwnersPage` pages shows Geb page abstractions
 - The `FindOwnersSpec` uses SQL to prepopulate test data, then uses Geb to drive the test scenario
- The `localhost:9123/PetClinic.WebTests.AllTests.FindOwnersTest` test specification provides a corresponding FitNesse based specification:
 - Prepopulating test data, then using Geb for the test scenario

Resources

- <http://junit.org/>
- <http://groovy.codehaus.org/>
- <http://groovy.codehaus.org/Eclipse+Plugin>
- <https://code.google.com/p/spock/>
- <http://www.fitnesse.org/>
- <http://www.gebish.org/>

Questions

