

Wellington Java User Group
For people interested in Java

Java Tidbits

13th June 2007

NZ Daylight Savings Time Changes

Nigel Charman

Wellington Java User Group

For people interested in Java

- Starting 30th Sept 07, clocks go forward 1 week earlier, and back 2 weeks later than normal.
- You need to do something if your Java applications use times and dates, and it is important that the times and dates are correct over this period.
- Why? Java maintains its own timezone database, independent of the OS.
- You need to:
 1. Update your OS
 2. Update your JDK/JREs, either patch or later version
 3. If using JodaTime, update its timezone database
 4. Update your middleware
- See <http://jug.wellington.net.nz/DST-2007.html> for more details and links to vendor sites.
- Please feedback any experiences or updates.



JPA in 10Mins!

Java User Group – Java Tidbits

Mark de Reeper

Sun Microsystems



Background

- Java Persistence API introduced in JSR-220 (Enterprise JavaBeans™ (EJB™) 3.0)
- Unifying POJO persistence technology into a standard enterprise API
- Works with either Java EE 5 or Java SE 5
 - > Superior ease of use within host container
 - > Client API with local transactions in Java SE platform
- Service Provider Interface (SPI) for container/persistence provider pluggability



Primary Features

- POJO-based persistence model
 - > Simple Java class files
- Supports traditional O-O modelling concepts
 - > Inheritance, polymorphism, encapsulation, etc.
- Standard abstract relational query language
- Standard O/R mapping metadata
 - > Using annotations and/or XML
- Portability across providers (implementations)



Implementations

- Persistence provider vendors include:
 - > Oracle, Sun/TopLink Essentials (RI)
 - > Eclipse JPA—EclipseLink Project
 - > BEA Kodo/Apache OpenJPA
 - > RedHat/JBoss Hibernate
 - > SAP JPA
- JPA containers
 - > Sun, Oracle, SAP, BEA, JBoss, Spring 2.0
- IDEs
 - > Eclipse, NetBeans, IntelliJ, JDeveloper

Demo



JPA Based Swing Client



JPA in 10Mins!

Mark de Reeper

mark.dereeper@sun.com

Wellington Java User Group

For people interested in Java

Enums, Databases, and the JDBC: moving towards an Active Record for Java

Tim Wright

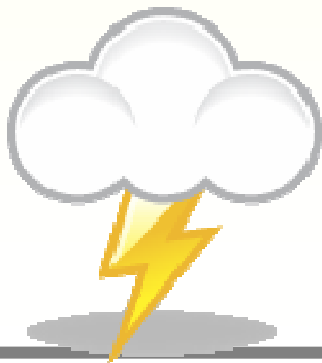
Unit Testing and the Database

Hibernate to the rescue!

Richard Schmidt

Metservice

hangstrap@gmail.com



Damn the database!

- Java code that access database is difficult to test.
 - Lots of code access the database,
 - If not directly, via lower level classes.
 - Need to start every test with database in known state
 - Check results after tests.



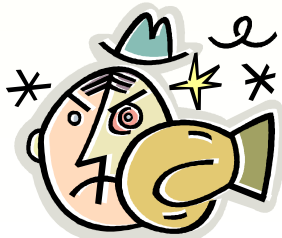
Mock it!

```
Connection conn = createMock(Connection.class);
Statement statement = createMock(Statement.class);
expect(conn.createStatement()).andReturn(statement);
expect(statement.execute("SELECT ID FROM
PERSON;")).andStubReturn(true);
statement.close();
expectLastCall();
replay(conn);
replay(statement);

ClassUnderTest classUnderTest = new ClassUnderTest ();
classUnderTest .findPersonIds();

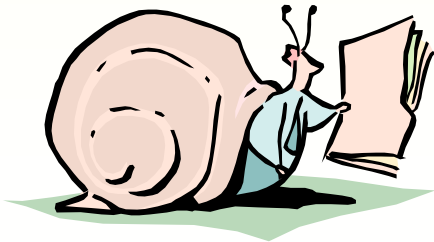
verify(conn);
verify(statement);
```

You don't want to do this!
Not really testing your SQL



dbUnit

- ❑ Setup() loads database to known state.
- ❑ Uses XML files to store data.
 - ❑ Difficult to manage xml files.
- ❑ Tends to be slow.
- ❑ Breaks when database changes.
- ❑ Can work well.



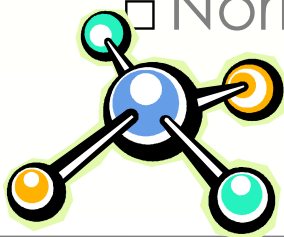
Hibernate

- Can be used to create a database
- Persists objects to database!
 - Generally persists Domain Model to the database.



Domain Model

- Core of most designs.
- Normally persisted to database.
- Need to be thoroughly tested.
- To Test:
 - Create a set of Domain Model instances in the state you require
 - Run the test.
 - Normally only a few sets are required.



```
public class TestModel {
```

```
    public Person joe;  
    public Person jane;  
    public Marriage joeToJane;
```

```
    public TestModel(){  
        joe = new Person( "Joe");  
        jane = new Person( "Jane");  
        joeToJane = new Marriage( joe, jane);  
    }  
}
```

```
public TestMarriage extends TestCase{
```

```
    public void testPolygamy(){  
        TestModel m = new TestModel();  
        try{  
            new Marriage( m.joe, new Person( "Sally"));  
            fail();  
        }catch( PolygamyException e){  
        }  
    }  
}
```



My light bulb moment

Create a set of Domain Model instances in the state you require.

- Use hibernate to
 - Create the database.
 - persist these instances.
- Database is now in a known state..
- Classes can then access the data in the normal way.
- Could use a in-memory database!



Setup()

- Open a new Connection
 - Hypersonic will create a new in-memory database
- Open new hibernate session.
- Create a new database
 - `new SchemeExport(...).create()`
- Create set of Domain model Instance
- Persist them with hibernate

START

Pros and Cons

- ❑ Bit slow to load the database.
- ❑ Cant use Toad to inspect the database.
- ❑ SQL not a standard.
- ❑ Simple and robust
- ❑ Hibernate needs to define the database



An Idea?

```
□ public TestModel{
□   //setup stuff here

□   public IDAO getDao(){

□       return new IDAO(){

□           public Person findPerson( String name){
□               if( name.equals( "joe")){
□                   return joe;
□               }
□               if( name.equals( "jane")){
□                   return jane;
□               }
□               return null;
□           }
□       }
□   };
□ }
```





FRONDE

Stop writing code, start coding

*Andrew Bickerton,
Fronde Systems Group Ltd*

The Future of Business



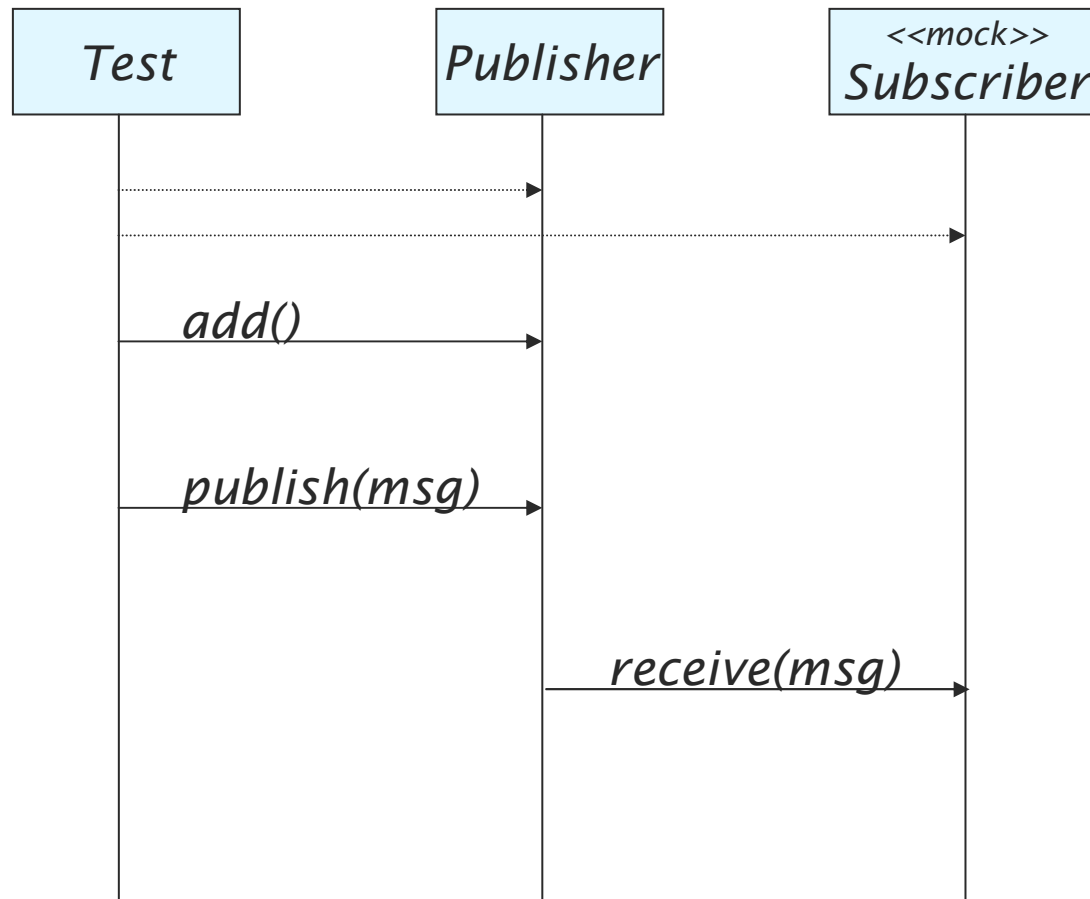
Outline

- *Coding demonstration*
 - *Test-Driven Development*
 - *Mock objects (jMock2)*
- *Using the IDE effectively*
 - *Make test-first a productive reality*

Using the IDE effectively

- *Modern IDEs: much more than code-completion*
- *IDE knows Java inside-out:*
 - *Interfaces & classes*
 - *Parameters & methods*
 - *Fields & variables*
 - *Control structures, e.g. iteration*
- *Adopt 'reverse code-completion'*
 - *Let the IDE write code while you code*

Publisher/Subscriber example



Summary

- *TDD:*
 - *Tests become part of design & development*
 - *Caller-Driven Development*
- *Reverse Code-Completion:*
 - *TDD becomes more productive*
 - *Consistent high-quality code style*
 - *Refactoring is routine, not occasional*
 - *Coding is more fun than writing code*

Why you might want to pay for continuous integration - an overview of JetBrains' TeamCity

Patrick Wilkes

Existing products

- Anthill / AnthillPro
- CruiseControl / CruiseControl Enterprise
- Bamboo from Atlassian
- TeamCity from JetBrains

TeamCity features

- build grid
- personal builds
- static analysis builds
- Ant, Maven, command line, plus Microsoft options
- Subversion, Perforce, CVS, VSS, Team Foundation Server, Clear Case
- plugins for IDEA, Eclipse, VisualStudio

Licensing

- US\$199 per user
- US\$99 per user if bought with an IDEA or ReSharper license
- Free for open source projects

Scratching Closures

Bruce Chapman

Part 1 – Closures Intro

Background

BGGA Proposal

Function Types

Closure Literals

Necessary Complexities

Syntactic Sugar

Loop Abstractions

Background

- Why?
 - Because Anonymous Inner Classes are ugly
- When? JDK7 (fingers crossed)
- Competing Proposals
 - BGGGA
 - CICE
 - ??
- Current Status

The BGGGA Proposal

- Most Complicated and Complete (Purist)
- But still grokkable
- Other proposals are just Anonymous Inner Classes in drag
- Closure – captures meaning with respect to enclosing lexical scope
 - Variables
 - return, this, super, break, continue

Details

Please refer to the closures proposal, and Neal Gafter's presentations.

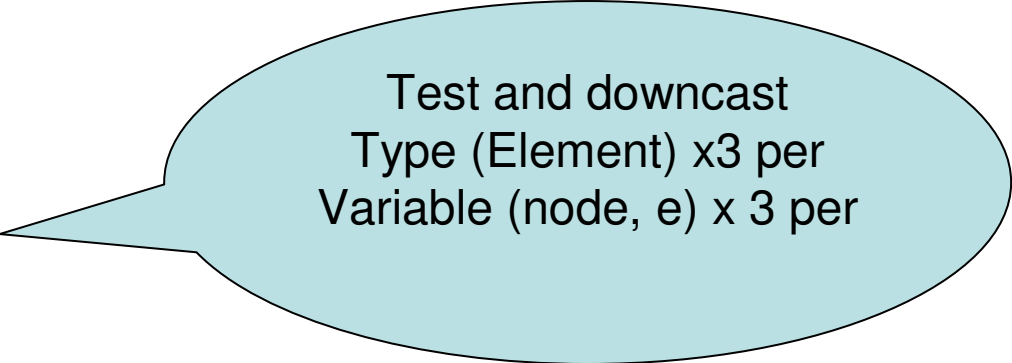
Part 2 -

- So can it Scratch my Itch?
- A non trivial control statement
 - If I push it hard does it break or shine?

The Itch

```
import org.w3c.dom.*;

Node node = ...;
If(node instanceof Element) {
    Element e = (Element)node;
    visitElement(e);
} else if(node instanceof Text) {
    Text t = (Text)node;
    visitText(t);
} else if(node instanceof Comment) {
    Comment c = (Comment)node;
    visitComment(c);
} else {
    visitOtherNode(node);
}
```



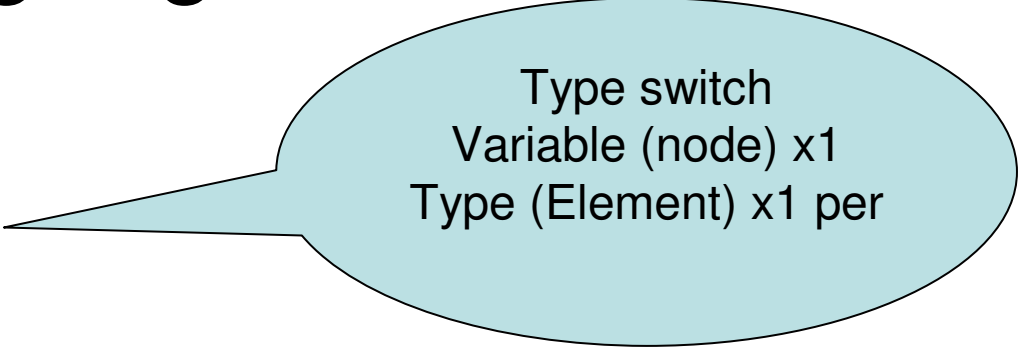
Test and downcast
Type (Element) x3 per
Variable (node, e) x 3 per

The Dream Team Itch Scratcher

A new language construct

```
import org.w3c.dom.*;

Node node = ...;
switch(node instanceof ?) {
    case Element:
        visitElement(node);
        break;
    case Text:
        visitText(node);
        break;
    case Comment:
        visitComment(node);
        break;
    default:
        visitOtherNode(node);
}
```



Type switch
Variable (node) x1
Type (Element) x1 per

A Basic Solution – Using Closures

Control Library Implementation

```
public class TypeSwitch<T> {
    private T obj;
    private boolean done;
    private TypeSwitch(T obj) {
        this.obj = obj;
    }
    public static <U> void on(U obj, {TypeSwitch<U> =>void} body) {
        body.invoke(new TypeSwitch<U>(obj));
    }

    public <U extends T> void when(Class<U> kind, {U=>void} body) {
        if((! done) && kind.isInstance(obj)) {
            U downcast = kind.cast(obj);
            body.invoke(downcast);
            done = true;
        }
    }

    public void otherwise({=> void} body) {
        if(! done) body.invoke();
    }
}
```

A Basic Solution – Using Closures Usage

```
import static TypeSwitch.on;
import org.w3c.dom.*;

Node node = ...;
on(TypeSwitch<Node> cases : node) {
    cases.when(Element node : Element.class) {
        visitElement(node);
    } cases.when(Text node : Text.class) {
        visitText(node);
    } cases.when(Comment node : Comment.class) {
        visitComment(node);
    } cases.otherwise() {
        visitOtherNode(node);
    }
}
```

A Basic Solution –Grokking it

```
public class TypeSwitch<T> {  
    public static <U> void on(U obj, {TypeSwitch<U> => void} body) {}  
    public <U extends T> void when(Class<U> kind, {U => void} body){}  
    public void otherwise({=> void} body) {}  
}
```

=====

```
on(TypeSwitch<Node> cases : node) {  
    cases.when(Element node : Element.class) {  
        visitElement(node);  
    }  
    // etc  
    cases.otherwise() {  
        visitOtherNode(node);  
    }  
}
```

Complete Solution

- Disallowing case after default
- Propagating Exceptions

Complete Library

Almost – sans javadoc

```
public class TypeSwitch<T> {
    private T obj;
    private boolean done;
    private boolean finished;
    private TypeSwitch(T obj) {
        this.obj = obj;
    }

    public static <U, throws E> void on(
        U obj,
        {TypeSwitch<U> => void throws E} body
    ) throws E {
        body.invoke(new TypeSwitch<U>(obj));
    }
}
```

Complete Library

Almost – sans javadoc

```
public <U extends T, throws E> void when(Class<U> kind, {U=>void throws E}
    body) throws E {
    if(finished)
        throw new IllegalStateException("calling when after otherwise");
    if((! done) && kind.isInstance(obj)) {
        U downcast = kind.cast(obj);
        body.invoke(downcast);
        done = true;
    }
}

public <throws E> void otherwise(={=> void throws E} body) {
    if(finished) throw new
        IllegalStateException("otherwise has already been called once");
    if(! done) body.invoke();
    finished = true;
}
}
```

Complete Usage

```
import static TypeSwitch.on;
import org.w3c.dom.*;

Node node = ...;
try {
    on(TypeSwitch<Node> cases : node) {
        cases.when(Element node : Element.class) {
            visitElement(node);
            throw new JustADemoException("Wellington JUG");
        } cases.when(Text node : Text.class) {
            visitText(node);
        } cases.when(Comment node : Comment.class) {
            visitComment(node);
        } cases.otherwise() {
            visitOtherNode(node);
        }
    }
} catch (JustADemoException jade) {
    jade.printStackTrace();
}
```

Evaluation

	switch part (once) Case part (ad nauseum)
JDK 5	nil <pre>} else if(node instanceof Element) { Element element = (Element)node;</pre>
Language Enhancement	<pre>switch(node instanceof ?) { break; case Element:</pre>
Closures	<pre>on(TypeSwitch<Node> cases : e) { } cases.when(Element element : Element.class) {</pre>

But Wait there's More

- Generic Type Reification (JDK 7 ?)
 - Pass Type parameter values at runtime (on demand) Just an Idea
 - now

```
<T> void doSomething(T arg, Class<T>) {
```

– With Type reification

```
<class T> void doSomething(T arg) {  
    (foo instanceof T); ✓  
    (T)foo; ✓
```

Library with Type Reification

```
public <class U extends T> void when({U=>void} body) {  
    if((! done) && obj instanceof U) {  
        body.invoke((U) obj);  
        done = true;  
    }  
}
```

Was

```
public <U extends T> void when(Class<U> kind, {U=>void} body) {  
    if((! done) && kind.isInstance(obj)) {  
        U downcast = kind.cast(obj);  
        body.invoke(downcast);  
        done = true;  
    }  
}
```

Usage with Type Reification

Library

```
import static TypeSwitch.on;
import org.w3c.dom.*;

Node node = ...;
on(TypeSwitch<Node> cases : node) {
    cases.when(Element node :) {
        visitElement(node);
    } cases.when(Text node :) {
        visitText(node);
    } cases.when(Comment node :) {
        visitComment(node);
    } cases.otherwise() {
        visitOtherNode(node);
    }
}
```

Language Change

```
import org.w3c.dom.*;

Node node = ...;
switch(node instanceof ?) {
    case Element:
        visitElement(node);
        break;
    case Text:
        visitText(node);
        break;
    case Comment:
        visitComment(node);
        break;
    default:
        visitOtherNode(node);
}
```

Thanks to JetBrains:
IntelliJ IDEA License
giveaway

13th June 2007