



# NEEDLE

for Java EE

Need(le) for Speed – Effective Unit Testing for Java EE

Heinz Wilming, akquinet AG



# Why do we Test?

- Confidence
- Cost Reduction
- Better Design
- Documentation
- Less Debug Time

# The levels of testing

- Unit
- Integration
- Acceptance

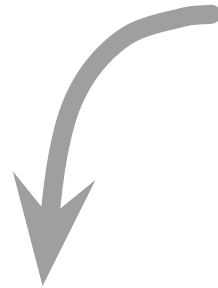


**NEEDLE**  
for Java EE



# Runtime environment

@EJB  
@Inject

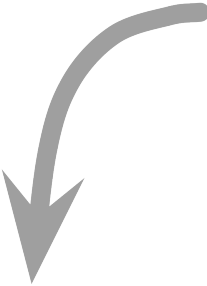


| OrderDao                     |
|------------------------------|
| entityManager: EntityManager |
| persist(order: Order)        |
| ...                          |

| ShoppingCart                       |
|------------------------------------|
| orderDao: OrderDao                 |
| add(product: Product, amount: int) |
| checkout(): boolean                |

# Unit-Test environment

~~@EJB~~  
~~@Inject~~



| OrderDao                     |
|------------------------------|
| entityManager: EntityManager |
| persist(order: Order)        |
| ...                          |

| ShoppingCart                       |
|------------------------------------|
| orderDao: OrderDao                 |
| add(product: Product, amount: int) |
| checkout(): boolean                |

# Unit-Test environment



| OrderDao                     |
|------------------------------|
| entityManager: EntityManager |
| persist(order: Order)        |
| ...                          |

```
OrderDao mock =
    mock (OrderDao.class);
```

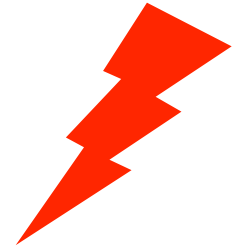
| ShoppingCart                       |
|------------------------------------|
| orderDao: OrderDao                 |
| add(product: Product, amount: int) |
| checkout(): boolean                |

```
new ShoppingCart ();
```



# Breaking the encapsulation

```
public class ShoppingCart {  
  
    @Inject  
    protected OrderDao orderDao;  
  
    ...  
  
}
```





# Breaking the encapsulation

```
public class ShoppingCart {  
  
    @Inject  
    private OrderDao orderDao;  
  
    ...  
  
    /*  
    * for unit test only  
    */  
    protected void setOrderDao (OrderDao dao) {  
        orderDao = dao;  
    }  
}
```







# Using reflection

```
public class ShoppingCartTest {  
    private ShoppingCart shoppingCart;  
    private OrderDao mock;  
  
    @Before  
    public void setup() throws Exception {  
        shoppingCart = new ShoppingCartService();  
        mock = mock(OrderDao.class);  
  
        Field field = ShoppingCart.class.getDeclaredField("orderDao");  
        field.setAccessible(true);  
        field.set(shoppingCart, mock);  
    }  
  
    @Test  
    public void testCheckout() {  
        ...  
    }  
}
```

| ShoppingCart                       |
|------------------------------------|
| orderDao: OrderDao                 |
| add(product: Product, amount: int) |
| checkout(): boolean                |

# Effective Unit Testing for Java EE

- out of container testing
- test components in isolation
- reduce test setup code
- analyze dependencies and provide mocks
- Fast in development and execution



```
public class ShoppingCartTest {  
  
    @Rule  
    public NeedleRule needleRule = new NeedleRule();  
  
    @ObjectUnderTest  
    private ShoppingCart shoppingCart;  
  
    @Test  
    public void testCheckout() {  
        boolean checkout = shoppingCart.checkout();  
        assertTrue(checkout);  
    }  
}
```


| ShoppingCart                       |
|------------------------------------|
| orderDao: OrderDao                 |
| add(product: Product, amount: int) |
| checkout(): boolean                |

# Instantiation of @ObjectUnderTest Components

## Dependency Injection

- Field
- Method
- Constructor

Default are Mock Objects

 JUnit.org Next Generation  
**JAVA TESTING**  
TESTING AND ADVANCED CONCEPTS

# Injection Provider

## Out-of-the-box

`@Inject`, `@EJB`, `@Resource`,  
`@PersistenceContext`,  
`@PersistenceUnit`

## Configuration of additional Annotations

- e.g. Seam 2 - `@In`, `@Logger`

## Configuration of additional injection provider

- e.g. `javax.inject.Qualifier`

# Mock injection

```
public class ShoppingCartTest {  
  
    @Rule  
    public NeedleRule needleRule = new NeedleRule();  
  
    @ObjectUnderTest  
    private ShoppingCart shoppingCart;  
  
    @Inject  
    private OrderDao orderDaoMock;  
  
    @Test  
    public void testCheckout() {  
        when(orderDaoMock.find(anyLong())).thenReturn(new Order());  
  
        boolean checkout = shoppingCart.checkout();  
        assertTrue(checkout);  
    }  
}
```



# Testing object graphs

Provide own objects

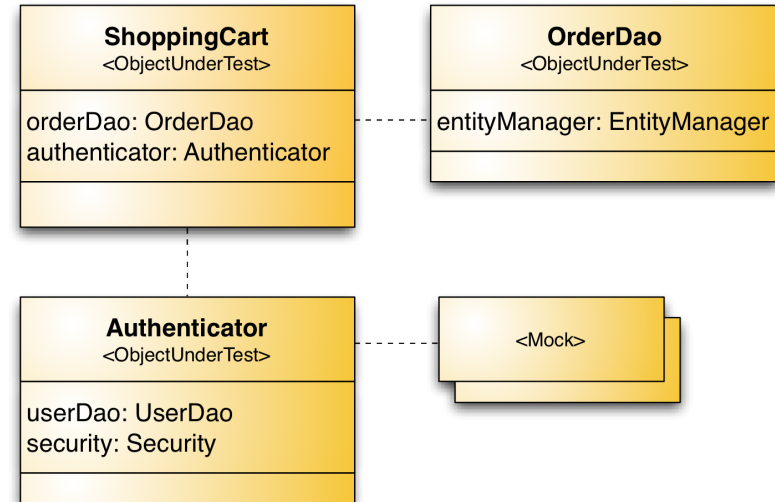
Multiple

@ObjectUnderTest

Components

Wiring complex object graph

- @InjectInto
- @InjectIntoMany





# Database testing

via JPA Provider, e.g. EclipseLink or Hibernate

EntityManager creation and injection

Optional: Execute Database operation on test setup and teardown

- Import SQL Scripts
- Deleting test data after the test execution



```
public class OrderDaoTest {

    @Rule
    public DatabaseRule dbRule = new DatabaseRule();

    @Rule
    public NeedleRule needleRule = new NeedleRule(dbRule);

    @ObjectUnderTest
    private OrderDao orderDao;

    @Test
    public void testFind() {
        Order order =
            new OrderTestDataBuilder().buildAndSave();
        Order orderFromDb = orderDao.find(order.getId());
        assertEquals(checkout);
    }
}
```

# DEMO

<http://seam-archetype.sourceforge.net/jbossc-seam-archetype/1.4/javaee.html>

# Summary

Fast

Less Glue Code

Keep Dependencies Private

Flexible & Extensible

Developer Happiness ;-)



Test success

**100,0%**

0 failures

0 errors

2.575 tests ▲

+2 skipped

2:46 min ▼

# Get Started Today!

```
<dependency>  
  <groupId>de.akquinet.jbosscc</groupId>  
  <artifactId>jbosscc-needle</artifactId>  
  <version>2.1</version>  
  <scope>test</scope>  
</dependency>
```



<http://needle.spree.de>

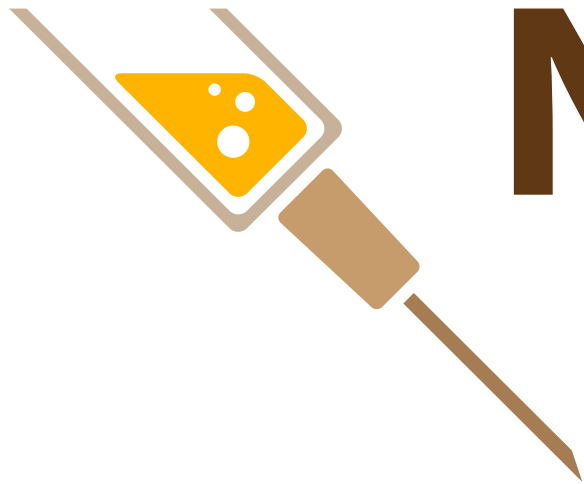


<http://github.com/akquinet/needle>

[heinz.wilming@akquinet.de](mailto:heinz.wilming@akquinet.de)

<http://blog.akquinet.de>





# NEEDLE

for Java EE