JUDCon

JBoss Users & Developers Conference

2012:India

NEEDLE

for Java EE

Effective Unit Testing of Java EE

Heinz Wilming
Carsten Erker

JBoss Users & Developers Conference JUDCon 2012:India

Why do we Test?

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

The Levels of Testing

- Unit
- Integration
- Acceptance
- Exploratory

Need(le) for Speed

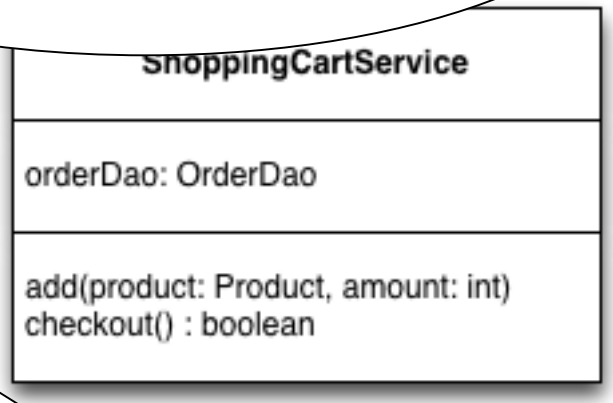
- out of container testing
- test components in isolation
- reduce test setup code
- analyze dependencies and provide mocks
- fast in development and execution

Basi

```
public class ShoppingCartServiceTest {

    @Rule
    public NeedleRule needleRule = new NeedleRule();

    @ObjectUnderTest
    private ShoppingCartService shoppingCartService;

    @Test
    public void testCheckout() {
        boolean checkout = shoppingCartService.checkout();

        assertTrue(checkout);
    }

}
```

Needle JUnit Rule to instantiate and initialize the objects under test.

Provides access to dependencies of the the objects under test



initialized object under test instance with injected dependencies

Needle Testcase

- Instantiation of `@ObjectUnderTest` Components
- Dependency Injection
  - Field
  - Method
  - Constuctor
  - Default are Mock Objects

Injection Provider

- Out of the box injection for
  - @Inject, @EJB, @Resource,
    @PersistenceContext, @PersistenceUnit
- Configuration of additional Annotations
  - e.g. Seam 2 injection
- Configuration of addtional injection provider
  - e.g. javax.inject.Qualifier

Custom Injection Provider

```
public class CurrentUserInjectionProvider implements InjectionProvider<User> {

    private final User currentUser = new User();

    @Override
    public User getInjectedObject(Class<?> injectionPointType) {
        return currentUser;
    }

    @Override
    public boolean verify(InjectionTargetInformation information) {
        return information.isAnnotationPresent(CurrentUser.class);
    }

    @Override
    public Object getKey(InjectionTargetInformation information) {
        return CurrentUser.class;
    }

}
```

Custom Injection Provider

1. NeedleRule

```
@Rule
public NeedleRule needleRule =
        new NeedleRule(new CurrentUserInjectionProvider());
```

2. needle.properties

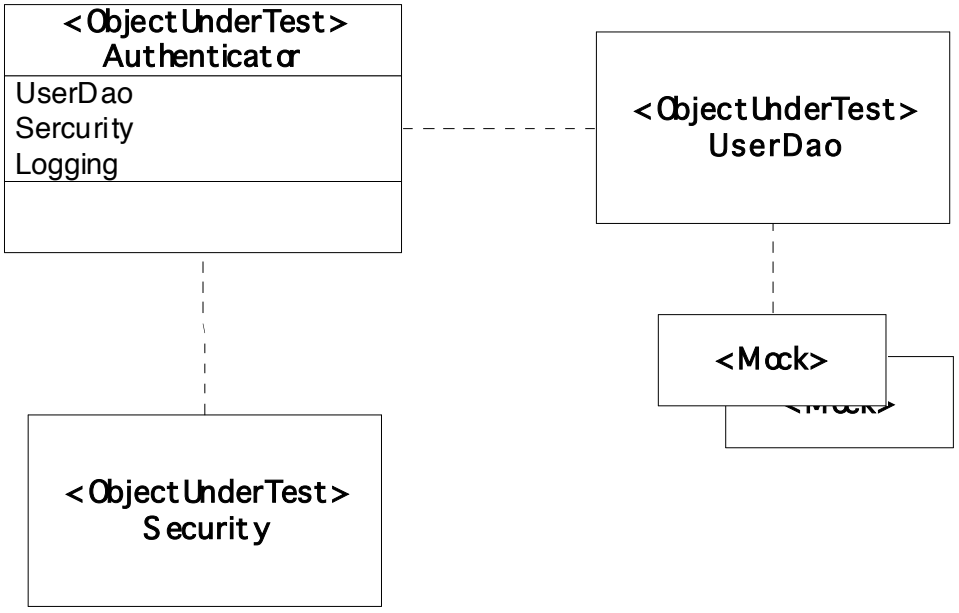
```
custom.injection.annotations=org.jboss.seam.annotations.In,
        org.jboss.seam.annotations.Logger

custom.injection.provider.classes=de.akquinet.judcon.CurrentUserInjectionProvider
```

Testing an Object Graph

- Provide own objects
- Wiring complex object graph

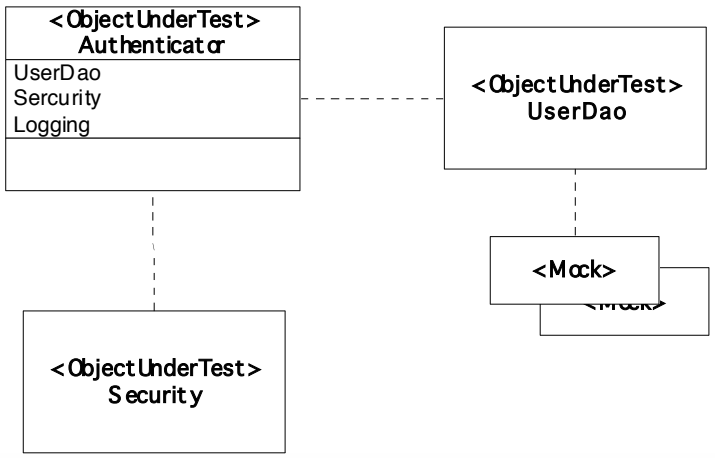
@InjectInto

@InjectIntoMany



Testing an Object Graph

```
public class AuthenticatorTest {

    @Rule
    public NeedleRule needleRule = new NeedleRule();

    @ObjectUnderTest
    private Authenticator authenticator;

    @ObjectUnderTest
    @InjectInto(targetComponentId="authenticator")
    private UserDaoBean userDao;

    @InjectIntoMany
    private Security security;

     @Test
     public void test () throws Exception {
            ...
     }
}
```



Mock Objects

```
public class AuthenticatorTest {

    @Rule
    public NeedleRule needleRule = new NeedleRule();

    @ObjectUnderTest
    private Authenticator authenticator;

    private EasyMockProvider mockProvider = needleRule.getMockProvider();


    @Test
    public void test () throws Exception {
      UserDao userDaoMock = needleRule.getInjectedObject(UserDao.class);

      EasyMock.expect(userDaoMock.find((String)EasyMock.anyObj
          ...
      mockProvider.replayAll();

      authenticator.authenticate();

      mockProvider.verifyAll();

    }
}
```

An EasyMock specific MockProvider.
Inherits from EasyMockSupport.

Access to injecting dependencies with the key of the injection provider.

Database Testing

- via JPA Provider, e.g. EclipseLink or Hibernate
- EntityManager creation and injection
- Transaction Utilities
- Optional: Execute Database operation on test setup and teardown
  - Import SQL Scripts
  - Deleting test data after the test execution

```
public class UserDaoTest {

    @Rule
    public DatabaseRule dbRule = new DatabaseRule();

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

    @ObjectUnderTest(implementation = UserDaoBean.class)
    private UserDao userDao;

     @Test
     public void testFind() throws Exception {
       User user = new UserTestdataBuilder(
            dbRule.getEntityManager()).withName("kbeck")
            .buildAndSave();

       User userFromDb = userDao.find("kbeck");

        assertEquals(...);
     }
}
```

Provides access to the database via JPA.

Execute DB Operations

Injection Provider of the EntityManger

Creates and persists an user object

Summary

- Fast
- Less Glue Code
- Keep Dependencies Private
- Extensible
- Developer Happiness ;-)

Q&A

heinz.wilming@akquinet.de

carsten.erker@akquinet.de

http://sourceforge.net/projects/jbosscc-needle/

http://needle.spree.de

http://blog.akquinet.de/