

Dependency Injection and Testing with the Spring framework

Dependency Injection and Testing with the Spring framework

- Why dependency injection?
 - It frees your code of knowing about the underlying implementation
 - Eliminates the necessity of lookup codes, thus saving development time
 - Eases the testing environment

Dependency Injection and Testing with the Spring framework

- A first example

```
public class MyFirstServiceClass {  
    public Hotel getHotel(Integer id) {  
        return new HotelJdbcDAO().get(id);  
    }  
    public String getFirstCharsFromHotelName(int hotelId, int chars) {  
        return this.getHotel(hotelId).getName().substring(0, chars);  
    }  
}
```

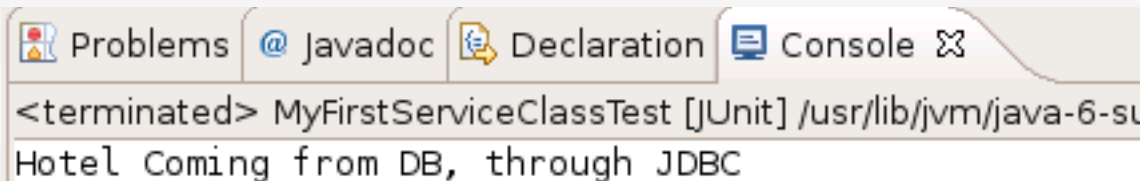
Dependency Injection and Testing with the Spring framework

- Testing the Service class

```
@Test
public void testGetFirstCharsFromHotelName() {
    MyFirstServiceClass service = new MyFirstServiceClass();
    String firstChars = service.getFirstCharsFromHotelName(1, 5);

    System.out.println(firstChars);

    Assert.assertEquals("Hotel", firstChars);
}
}
```



The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of a test run for MyFirstServiceClassTest [JUnit]. The output shows the test has terminated and the result is "Hotel Coming from DB, through JDBC".

```
<terminated> MyFirstServiceClassTest [JUnit] /usr/lib/jvm/java-6-su
Hotel Coming from DB, through JDBC
```

Dependency Injection and Testing with the Spring framework

- Great! A JDBC DAO! But I was planning to use Hibernate!
 - Problem
 - The service object knows the DAO's implementation class
 - Solution
 - Make the DAO classes implement an interface

Dependency Injection and Testing with the Spring framework

- The new DAO classes

```
public class HotelJdbcDAO implements HotelDAO {  
    public Hotel get(Integer id) {  
        return new Hotel(id, "Hotel Coming from DB, through JDBC");  
    }  
}
```

```
public class HotelHibernateDAO implements HotelDAO {  
    @Override  
    public Hotel get(Integer id) {  
        return new Hotel(id, "Hotel Coming from DB, through Hibernate");  
    }  
}
```

Dependency Injection and Testing with the Spring framework

- But to achieve independency, we need to change the service class

```
public class MyNewEnhancedServiceClass {  
    public Hotel getHotel(Integer id) {  
        return HotelDAOFactory.get(HotelDAOFactory.HIBERNATE_DAO).get(id);  
    }  
  
    public String getFirstCharsFromHotelName(int hotelId, int chars) {  
        return this.getHotel(hotelId).getName().substring(0, chars);  
    }  
}
```

Dependency Injection and Testing with the Spring framework

- Dependency Injection kicks in

```
public class MyNewSpringEnhancedServiceClass {  
    @Autowired @Qualifier("hotelDao")  
    private HotelDAO hotelDAO;  
  
    public Hotel getHotel(Integer id) {  
        return hotelDAO.get(id);  
    }  
  
    public String getFirstCharsFromHotelName(int hotelId, int chars) {  
        return this.getHotel(hotelId).getName().substring(0, chars);  
    }  
}
```

Dependency Injection and Testing with the Spring framework

- It's all about context

```
<beans ...>  
    <context:annotation-config/>  
    <bean id="service" class="com.mirai.spring.MyNewSpringEnhancedServiceClass" />  
    <bean id="hotelDao" class="com.mirai.dao.HibernateDAO" />  
</beans>
```

Dependency Injection and Testing with the Spring framework

- Testing the new Service class

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"/beans-test.xml"})
public class MyNewSpringEnhancedServiceClassTest {

    @Autowired @Qualifier("service")
    MyNewSpringEnhancedServiceClass service;

    @Test
    public void testGetFirstCharsFromHotelName() {
        String firstChars = service.getFirstCharsFromHotelName(1, 5);

        System.out.println(firstChars);

        Assert.assertEquals("Hotel", firstChars);
    }
}
```

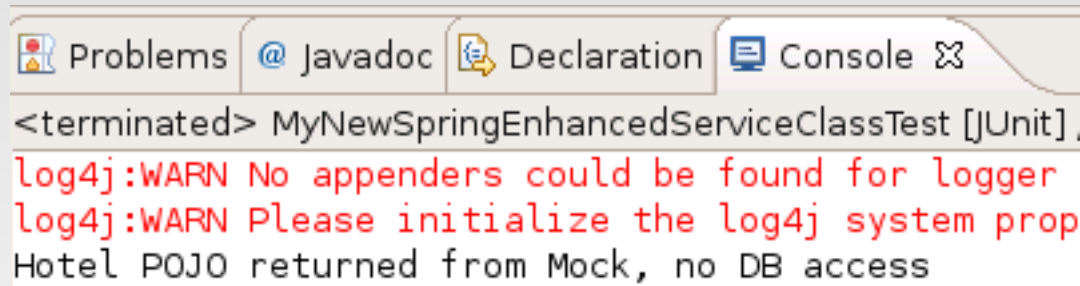
Dependency Injection and Testing with the Spring framework

- It's all about [test] context

```
<beans ...>  
  <context:annotation-config/>  
  <bean id="service" class="com.mirai.spring.MyNewSpringEnhancedServiceClass"/>  
  <bean id="hotelDao" class="com.mirai.dao.MockHibernateDAO"/>  
</beans>
```

Dependency Injection and Testing with the Spring framework

- Results from testing the new Service class



The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, and Console. The console output displays the following text:

```
<terminated> MyNewSpringEnhancedServiceClassTest [JUnit],  
log4j:WARN No appenders could be found for logger  
log4j:WARN Please initialize the log4j system prop  
Hotel POJO returned from Mock, no DB access
```

Dependency Injection and Testing with the Spring framework

- Conclusions
 - With dependency injection and spring, you can achieve a great level of independence while keeping your code highly testable through the easy substitution of your real objects by mocks.

- Further Resources
 - www.springframework.org
 - www.leonardoborges.com