Spring in Action

Agenda

- What is Spring?
- Loose coupling with Dependency Injection
- Declarative programming with AOP
- Eliminating Boilerplate with templates

Spring-Loaded

What is Spring?

What is Spring?

- Lightweight container framework
  - Lightweight – minimally invasive
  - Container – manages app component lifecycle
  - Framework – basis for enterprise Java apps
- Open source
  - Apache licensed

Spring-Loaded

What it's not

- An application server
  - Although...there's tcServer, dmServer
  - And Spring supports a lot of the EJB3 model

SPRING-LOADED

The "Spring Way"

- Spring's overall theme: Simplifying Enterprise Java Development
- Strategies
  - Loose coupling with dependency injection
  - Declarative programming with AOP
  - Boilerplate reduction with templates
  - Minimally invasive and POJO-oriented

SPRING-LOADED

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

A brief history of Spring

- Spring 1.0
  - DI, AOP, web framework
- Spring 2.0
  - Extensible config, bean scoping, dynamic language support, new tag library
- Spring 2.5
  - Annotation-driven, automatic bean discovery, new web framework, JUnit 4 integration
- Spring 3.0
  - REST, SpEL, declarative validation, ETag support, Java-based configuration

SPRING-LOADED

Beyond the framework

- Spring Web Flow
- BlazeDS Integration
- Spring-WS
- Spring Security
- Spring-DM
- dmServer
- Bundlor
- tcServer
- Spring Batch
- Spring Integration
- Spring LDAP
- Spring IDE / STS
- Spring Rich Client
- Spring .NET
- Spring BeanDoc
- Groovy/Grails

SPRING-LOADED

Spring community

- Forum: http://forum.springframework.org
- Issues: http://jira.springframework.org
- Extensions:
  - http://www.springframework.org/extensions
- Conferences, user groups, etc
- GET INVOLVED!

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Loose Coupling with Dependency Injection

What's wrong here?

```
public class Mechanic {
  public void fixCar() {
    PhillipsScrewdriver tool =
          new PhillipsScrewdriver();
    tool.use();
  }
}
```


E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

How about now?

```
public class Mechanic {
  public void fixCar() {
    Tool tool =
          new PhillipsScrewdriver();
    tool.use();
  }
}
```


E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

Any better?

```
public class Mechanic {
  public void fixCar() {
    ToolFactory tf =
          ToolFactory.getInstance();
    Tool tool = tf.getTool();
    tool.use();
  }
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Certainly this is better...

```
public class Mechanic {
  public void fixCar() {
    InitialContext ctx = null;
    try {
      ctx = new InitialContext();
      Tool quest = (Tool) ctx.lookup(
          "java:comp/env/Tool");
      tool.use();
    } catch (NamingException e) {
    } finally {
      if(ctx != null) {
        try {ctx.close(); }
        catch (Exception e) {}
      }
    }
  }
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Let's try again...

```
public class Mechanic {
  private Tool tool;
  public Mechanic(Tool tool) {
    this.tool = tool;
  }

  public void fixCar() {
    tool.use();
  }
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Or maybe this...

```
public class Mechanic {
  private Tool tool;
  public void setTool(Tool tool) {
    this.tool = tool;
  }

  public void fixCar() {
    tool.use();
  }
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Dependency injection

- Objects are given what they need
- Coupling is low when used with interfaces
- Makes classes easier to swap out
- Makes classes easier to unit test

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

DI in Spring

- Several options
  - XML
  - Annotation-driven
  - Java-based configuration
- None are mutually exclusive

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

XML-based wiring

```
<bean id="screwdriver"
    class="com.habuma.tools.PhillipsScrewdriver" />

<bean id="mechanic"
    class="com.habuma.mechanic.AutoMechanic">
  <constructor-arg ref="screwdriver" />
</bean>
```

```
<bean id="screwdriver"
    class="com.habuma.tools.PhillipsScrewdriver" />

<bean id="mechanic"
    class="com.habuma.mechanic.AutoMechanic">
  <property name="tool" ref="screwdriver" />
</bean>
```

SPRING-LOADED

Annotation-based wiring

```
<context:component-scan
    base-package="com.habuma.mechanic" />
```

```
public class Mechanic {
  private Tool tool;
  @Autowired
  public void setTool(Tool tool) {
    this.tool = tool;
  }

  public void fixCar() {
    tool.use();
  }
```

```
public class Mechanic {
  @Autowired
  private Tool tool;

  public void fixCar() {
    tool.use();
  }
}
```

SPRING-LOADED

Java-based configuration

```
@Configuration
public class AutoShopConfig {
  @Bean
  public Tool screwdriver() {
    return new PhillipsScrewdriver();
  }

  @Bean
  public Mechanic mechanic() {
    return new AutoMechanic(screwdriver());
  }
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Declarative Programming with AOP

Aspects

- Separate loosely-related behavior
- Objects don't have to do work that isn't their job
  - Keeps them cohesive
  - Keeps them simple

Life without AOP

```
public void withdrawCash(double amount) {
   UserTransaction ut = context.getUserTransaction();

   try {
      ut.begin();
      updateChecking(amount);
      machineBalance -= amount;
      insertMachine(machineBalance);
      ut.commit();
   } catch (ATMException ex) {
       LOGGER.error("Withdrawal failed");
       try {
          ut.rollback();
       } catch (SystemException syex) {
          // ...
       }
    }
}
```

Life with AOP

```
public void withdrawCash(double amount) {
   try {
      updateChecking(amount);
      machineBalance -= amount;
      insertMachine(machineBalance);
   } catch (ATMException ex) {
      LOGGER.error("Withdrawal failed");
   }
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

With some more AOP

```
public void withdrawCash(double amount) {
   updateChecking(amount);
   machineBalance -= amount;
   insertMachine(machineBalance);
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Spring AOP

- Comes in 3 forms
  - XML-based
  - Annotation-based
  - Native AspectJ

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

AOP Terms

- Aspect
- Advice
- Pointcut
- Joinpoint

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Logging aspect

```
public class LoggingAspect {
  private static final Logger LOGGER =
      Logger.getLogger(LoggingAspect.class);
  public logBefore() {
    LOGGER.info("Starting withdrawal");
  }

  public logAfterSuccess() {
    LOGGER.info("Withdrawal complete");
  }

  public logFailure() {
    LOGGER.info("Withdrawal failed");
  }
}
```


XML-based AOP

```
<bean id="loggingAspect"
    class="LoggingAspect" />

<aop:config>
  <aop:aspect ref="loggingAspect">
    <aop:before
        pointcut="execution(* *.withdrawCash(..))"
        method="logBefore" />
   <aop:after-returning
        pointcut="execution(* *.withdrawCash(..))"
        method="logBefore" />
   <aop:after-throwing
        pointcut="execution(* *.withdrawCash(..))"
        method="logBefore" />
  </aop:aspect>
</aop:config>
```

Annotation-based AOP

```
@Aspect
public class LoggingAspect {
  private static final Logger LOGGER =
      Logger.getLogger(LoggingAspect.class);
  @Pointcut("execution(* *.withdrawCash(..))")
  public void withdrawal() {}

  @Before("withdrawal()")
  public logBefore() {
    LOGGER.info("Starting withdrawal");
  }
  @AfterReturning("withdrawal()")
  public logAfterSuccess() {
    LOGGER.info("Withdrawal complete");
  }

  @AfterThrowing("withdrawal()")
  public logFailure() {
    LOGGER.info("Withdrawal failed");
  }
}
```

```
<aop:aspectj-autoproxy />
```

SPRING-LOADED

What about transactions?

```
<tx:advice id="txAdvice">
  <tx:attributes>
    <tx:method name="withdraw*"
        propagation="REQUIRED" />
    <tx:method name="inquire*"
        propagation="SUPPORTS"
        read-only="true" />
  </tx:attributes>
</tx:advice>
```

SPRING-LOADED

E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

Annotating transactions

```
<tx:annotation-driven />
```

```
@Transactional(propagation=Propagation.REQUIRED)
public void withdrawCash(double amount) {
   updateChecking(amount);
   machineBalance -= amount;
   insertMachine(machineBalance);
}

@Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
public double inquireBalance(String acctNo) {
  // ...
}
```

SPRING-LOADED

Eliminating Boilerplate with Templates

Boilerplate

- Exists everywhere
  - JDBC
  - JNDI
  - JMS
  - ...all over JEE...
- Lots of plumbing code repeated over and over

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Recognize this?

```
public void addSpitter(Spitter spitter) {
  Connection conn = null;
  PreparedStatement stmt = null;
  try {
    conn = dataSource.getConnection();
    stmt = conn.prepareStatement(SQL_INSERT_SPITTER);
    stmt.setString(1, spitter.getUsername());
    stmt.setString(2, spitter.getPassword());
    stmt.setString(3, spitter.getFullName());
    stmt.execute();
  } catch (SQLException e) {
    // do something...not sure what, though
  } finally {
    try {
      if (stmt != null) {
        stmt.close();
      }
      if (conn != null) {
        stmt.close();
      }
    } catch (SQLException e) {
      // I'm even less sure about what to do here
    }
  }
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Ring any bells?

```
InitialContext ctx = null;
try {
  ctx = new InitialContext();
  DataSource ds = (DataSource) ctx.lookup(
                          "java:comp/env/jdbc/SpitterDatasource");
} catch (NamingException ne) {
  // handle naming exception
  …
} finally {
  if(ctx != null) {
    try {
      ctx.close();
    } catch (NamingException ne) {}
  }
}
```

SPRING-LOADED

Get the message?

```
ConnectionFactory cf =
    new ActiveMQConnectionFactory("tcp://localhost:61616");
Connection conn = null;
Session session = null;
try {
  conn = cf.createConnection();
  session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
  Destination destination = new ActiveMQQueue("myQueue");
  MessageProducer producer = session.createProducer(destination);
  TextMessage message = session.createTextMessage();
  message.setText("Hello world!");
  producer.send(message);
} catch (JMSException e) {
…
} finally {
  try {
    if(session != null) { session.close(); }
    if(conn != null) { conn.close(); }
  } catch (JMSException ex) {}
}
```

SPRING-LOADED

JDBC TEMPLATE

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="${db.driver}" />
   <property name="url" value="${db.url}" />
   <property name="username" value="${db.username}" />
   <property name="password" value="${db.password}" />
</bean>
<bean id="jdbcTemplate"
      class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
   <constructor-arg ref="dataSource" />
</bean>
<bean id="spitterDao"
      class="com.habuma.spitter.persistence.SimpleJdbcTemplateSpitterDao">
  <property name="jdbcTemplate" ref="jdbcTemplate" />
</bean>
```

```
public void addSpitter(Spitter spitter) {
  jdbcTemplate.update(SQL_INSERT_SPITTER,
          spitter.getUsername(),
          spitter.getPassword(),
          spitter.getFullName());
  spitter.setId(queryForIdentity());
}
```


E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

SPRING-FLAVORED JNDI

```
<jee:jndi-lookup id="dataSource"
    jndi-name="jdbc/SpitterDatasource"
    resource-ref="true" />
```


E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

Spring-flavored JNDI

```
<bean id="jmsTemplate"
    class="org.springframework.jms.core.JmsTemplate">
  <property name="connectionFactory" ref="connectionFactory" />
</bean>
```

```
public void sendMotoristInfo(final Motorist motorist) {
  jmsTemplate.send(destination,
    new MessageCreator() {
      public Message createMessage(Session session)
          throws JMSException {
        MapMessage message = session.createMapMessage();
        message.setString("lastName", motorist.getLastName());
        message.setString("firstName", motorist.getFirstName());
        message.setString("email", motorist.getEmail());
        return message;
      }
    });
}
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Summary

Spring...

- Is a lightweight container framework
- Simplifies Java development
- POJO-oriented
- Promotes loose coupling with dependency injection
- Supports declarative programming with AOP
- Eliminates boilerplate code with templates

Spring-Loaded

Where we're headed next

- Spring 3 without XML and the ADD developer
- Building web applications with Spring @MVC
- Spring + JPA + Hibernate
- Spring Security
- Grails
- Spring-DM
- Panel discussion

Spring-Loaded

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma