Spring Training

TechFerry Infotech Pvt. Ltd.
(http://www.techferry.com/)
Conversations

○ Introduction to Spring
○ Concepts: Annotations, MVC, IOC/DI, Auto wiring
○ Spring Bean/Resource Management
○ Spring MVC, Form Validations.
○ Unit Testing
○ Spring Security – Users, Roles, Permissions.
○ Code Demo
  ■ CRUD using Spring, Hibernate, MySQL.
  ■ Spring security example.
  ■ REST/jQuery/Ajax example
Spring - Introduction

Exercise: What do we need in an enterprise application?

- Database Access, Connection Pools?
- Transactions?
- Security, Authentication, Authorization?
- Business Logic Objects?
- Workflow/Screen Flow?
- Messaging/emails?
- Service Bus?
- Concurrency/Scalability?

Can somebody wire all the needed components?
Do we have to learn everything before we can start?
Hello Spring

- Spring is potentially a one-stop shop, addressing most infrastructure concerns of typical web applications
  - so you focus only on your business logic.
- Spring is both comprehensive and modular
  - use just about any part of it in isolation, yet its architecture is internally consistent.
  - maximum value from your learning curve.
What is Spring?

- Open source and lightweight web-application framework
- Framework for wiring the entire application
- Collection of many different components
- Reduces code and speeds up development

Spring is essentially a technology dedicated to enabling you to build applications using POJOs.
Why Spring?

- Spring Enables POJO Programming
  - Application code does not depend on spring API’s
- Dependency Injection and Inversion of Control simplifies coding
  - Promotes decoupling and re-usability

Features:
- Lightweight
- Inversion of Control (IoC)
- Aspect oriented (AOP)
- MVC Framework
- Transaction Management
- JDBC
- Ibatis / Hibernate
Spring Modules

- **Spring AOP**
  - Source-level Metadata
  - AOP Infrastructure

- **Spring ORM**
  - Hibernate, iBATIS and JDO Support

- **Spring Web**
  - WebApplicationContext
  - Multipart Resolver
  - Web Utilities

- **Spring DAO**
  - Transaction Infrastructure
  - JDBC and DAO Support

- **Spring Context**
  - ApplicationContext
  - UI Support
  - Validation
  - JNDI, EJB & Remoting Support
  - Mail

- **Spring MVC**
  - Web Framework
  - Web Views
  - JSP, Velocity, Freemarker, PDF, Excel, XML/XSL

- **Spring Core**
  - Supporting Utilities
  - Bean Factory/Container
What else Spring do?

Spring Web Flow
Spring Integration
Spring Web-Services
Spring MVC
Spring Security
Spring Batch
Spring Social
Spring Mobile

... and let it ever expand ...
Inversion of Control/Dependency Injection

"Don't call me, I'll call you."

- IoC moves the responsibility for making things happen into the framework
- Eliminates lookup code from within the application
- Loose coupling, minimum effort and least intrusive mechanism
IOC/DI

Non-IoC

IoC
Non IOC Example:
class MovieLister...
   private MovieFinder finder;
   public MovieLister() {
      finder = \textcolor{red}{\textbf{new MovieFinderImpl}}();
   }

public interface MovieFinder {
   List findAll();
}

class MovieFinderImpl ... {
   public List findAll() {
      ...
   }
}

IOC/DI

IoC Example: DI exists in major two variants:

Setter Injection

```java
public class MovieLister {
    private MovieFinder movieFinder;
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }
}
```

Constructor Injection

```java
public class MovieLister {
    private MovieFinder movieFinder;
    public MovieLister(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }
}
```
Spring Bean Management

Code Demo ....

- Annotations: `@Component`, `@Service`, `@Repository`
- Annotation: `@Autowired`
- `web.xml` - Context loader listener to scan components
- `<context:annotation-config />
  <context:component-scan base-package="..." />`
Bean Scopes

**singleton**
Scopes a single bean definition to a single object instance per Spring IoC container.

**prototype**
Scopes a single bean definition to any number of object instances.

**request**
Scopes a single bean definition to the lifecycle of a single HTTP request.

**session**
Scopes a single bean definition to the lifecycle of a HTTP Session.

**global session**
Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context.
Singleton Bean

Only one instance is ever created...

... and this same shared instance is injected into each collaborating object
Prototype Beans

- Use `@Scope("prototype")`
- **Caution:** dependencies are resolved at instantiation time. *It does NOT create* a new instance at runtime more than once.

```
<bean id="..." class="...">
  <property name="accountDao" ref="accountDao"/>
</bean>
```

```
<bean id="..." class="...">
  <property name="accountDao" ref="accountDao"/>
</bean>
```

```
<bean id="..." class="...">
  <property name="accountDao" ref="accountDao"/>
</bean>
```

```
<bean id="accountDao" class="..." scope="prototype"/>
```

... each and every time the prototype is referenced by collaborating beans
Bean Scopes Contd..

- As a rule of thumb, you should use the prototype scope for all beans that are stateful, while the singleton scope should be used for stateless beans.
- RequestContextListener is needed in web.xml for request/session scopes.
- Annotation:@Scope("request") @Scope("prototype")

Homework:
- Singleton bean referring a prototype/request bean?
- @Qualifier, Method Injection.

Hate Homework?
- Stick to stateless beans. :)
Wiring Beans

**no**
No autowiring at all. Bean references must be defined via a ref element. This is the default.

**byName**
Autowiring by property name.

**byType**
Allows a property to be autowired if there is exactly one bean of the property type in the container. If there is more than one, a fatal exception is thrown.

**constructor**
This is analogous to *byType*, but applies to constructor arguments.

**autodetect**
Chooses *constructor* or *byType* through introspection of the bean class.
Homework :) 

1. What wiring method is used with `@Autowire` annotation?
2. Other annotations you may find useful:
   ○ `@Required`
   ○ `@Resource`

Also review the Spring annotation article:
http://www.techferry.com/articles/spring-annotations.html
MVC - Model View Controller

- Better organization and code reuse.
- Separation of Concern
- Can support multiple views
Spring MVC

Code Demo ....

- Annotations: `@Controller`, `@RequestMapping`, `@ModelAttribute`, `@PathVariable`
- Spring DispatcherServlet config - just scan controllers
- `web.xml` - Context loader listener to scan other components
- `ResourceBundleMessageSource` and `<spring:message>` tag


- `@RequestMapping` Details
- Handler method arguments and Return Types
Pre-populate Model and Session Objects

@Controller
@RequestMapping("/owners/{ownerId}/pets/{petId}/edit")
@SessionAttributes("pet")
public class EditPetForm {

    @ModelAttribute("types")
    public Collection<PetType> populatePetTypes() {
        return this.clinic.getPetTypes();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String processSubmit(@ModelAttribute("pet") Pet pet, BindingResult result,
                                SessionStatus status) {
        new PetValidator().validate(pet, result);
        if (result.hasErrors()) {
            return "petForm";
        } else {
            this.clinic.storePet(pet);
            status.setComplete();
            return "redirect:owner.do?ownerId=" + pet.getOwner().getId();
        }
    }
}
Form Validation

Code Demo ...
- BindingResult
- Validator.validate()
- <form:errors> tag

Alternative: Hibernate Validator can also be used for annotation based validation.

```java
public class PersonForm {
    @NotNull
    @Size(max=64)
    private String name;

    @Min(0)
    private int age;

    @RequestMapping("/foo")
    public void processFoo(@Valid Foo foo) {
        /* ... */
    }
}
```
Unit Testing

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/spring-servlet-test.xml" })
@Test

Other useful Annotations:

@DirtiesContext
@ExpectedException(SomeBusinessException.class)
@Timed(millis=1000)
@NotTransactional
Spring Security

Code Demo ...

- `<sec:authorize>` tag
- Annotations: `@PreAuthorize`
- `applicationContext-security.xml`
- DB Schema: Users, Authorities
Thank you and Questions?