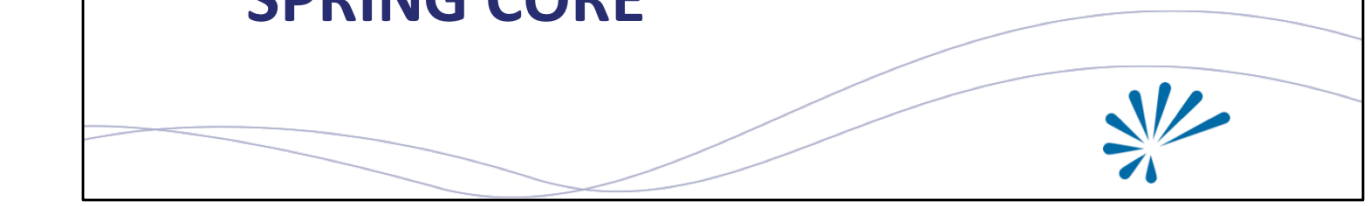
Introductions

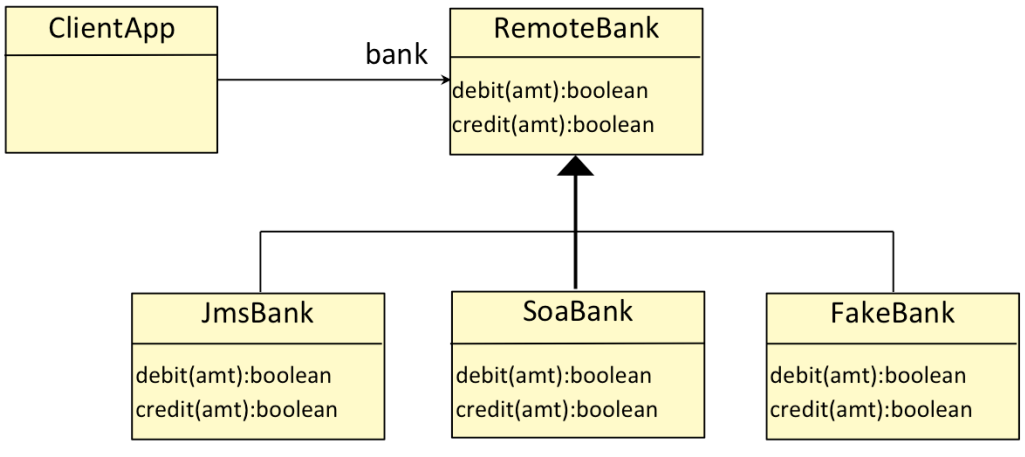
- Hello!
- About us
- About you
  - Name
  - Title/function
  - Experience
  - **Expectations for the class**

improving
It's what we do. ™

SPRING CORE



Dependency Inversion



Dependency Inversion

```
public class ClientApp {

    private RemoteBank bank;

    public boolean debit(Money amount) {

        // ... some logic

        try {
            return bank.debit(amount);

        }catch(RemoteException e) {
            // ... some handling
            return false;
        }

    }
}
```

No knowledge of implementations

- So who does have knowledge?
- How does it get to this class?
- Who creates the object?

Dependency Injection

The act of inserting an object reference into a class with the purpose of reducing dependencies.

Dependency Injection - Constructor

```
public class ClientApp {

    private RemoteBank bank;

    public ClientApp(RemoteBank bank) {
        this.bank = bank;
    }

    public boolean debit(Money amount) {

        // ... some logic
    }
}
```

Constructor injection

Dependency Injection - Setter

```
public class ClientApp {

    private RemoteBank bank;

    public void setBank(RemoteBank bank) {
        this.bank = bank;
    }

    public boolean debit(Money amount) {

        // ... some logic
    }
}
```

Setter Injection

Dependency Injection - Field

```
public class ClientApp {

    private RemoteBank bank;

    public boolean debit(Money amount) {

        // ... some logic
    }
}
```

Field Injection (Reflection)

Spring

- Inversion of control container for the Java platform
  - Reflection
  - Java Beans & POJOs
  - Aspect Oriented Programming

JavaBeans

- A JavaBean is a class that follows a few rules allowing it to be used dynamically:
  - Public, concrete class.
  - Public, no-args constructor.
- Can define properties:
  - Same as attribute with public getter and setter.
  - Must follow naming convention:
    - <Property Type> get<PropertyName>()
    - void set<PropertyName>(<PropertyType>)

Configuring Beans

… with xml & constructor injection.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans htt

    <bean name="jmsBank" class="com.improving.JmsBank"/>
    <bean name="soaBank" class="com.improving.SoaBank"/>
    <bean name="fakeBank" class="com.improving.FakeBank"/>

    <bean name="clientApp" class="com.improving.ClientApp" >
        <constructor-arg ref="jmsBank"></constructor-arg>
    </bean>
</beans>
```

Application Context

... accessing beans from xml.

```
public static void main(String[] args) {
    ApplicationContext context =
        new ClassPathXmlApplicationContext("applicationContext.xml");

    ClientApp app = context.getBean("clientApp", ClientApp.class);

    app.debit(money);

}
```

Injecting Beans - Setter

… with xml & setter injection.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans htt

    <bean name="jmsBank" class="com.improving.JmsBank"/>
    <bean name="soaBank" class="com.improving.SoaBank"/>
    <bean name="fakeBank" class="com.improving.FakeBank"/>

    <bean name="clientApp" class="com.improving.ClientApp" >
        <property name="bank" ref="fakeBank"/>
    </bean>
</beans>
```

Autowiring

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http

    <bean name="bank" class="com.improving.JmsBank"/>

    <bean name="clientApp"
          class="com.improving.ClientApp"
          autowire="byName"/>
</beans>
```

Looks for bean with same name as property.
Other Options: byType, constructor

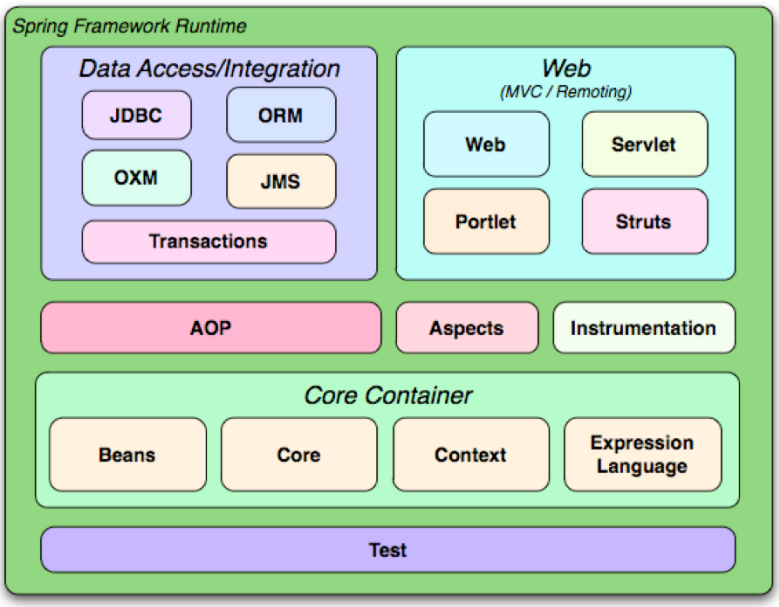
Bean Scopes

- Singleton (default)
- Prototype
- Web Scopes:
  - Request
  - Session
  - Global

More Later

```
<bean name="clientApp"
      class="com.improving.ClientApp"
      scope="prototype"/>
```

Spring Framework



What is an Annotation?

- Metadata that can be added to Java source code.
- Classes, methods, variables, parameters and packages may be annotated.

```
@Test
public void test() {
    fail("Not yet implemented");
}
```

Spring Bean Annotations

- `@Component`
  generic stereotype for any Spring-managed component
- `@Service`
  stereotype for service layer
- `@Repository`
  stereotype for persistence layer
- `@Controller`
  stereotype for presentation layer (spring-mvc)

Autowiring with Annotations

```java
@Service("clientApp")
public class ClientApp {

    private RemoteBank bank;

    public ClientApp() {}

    @Autowired
    public ClientApp(RemoteBank bank) {
        this.bank = bank;
    }

//  public void setBank(RemoteBank bank) {
//      this.bank = bank;
//  }

    public boolean debit(Money amount) {
        // ... some logic
    }
}
```

```java
@Repository("soaBank")
public class SoaBank extends RemoteBank {

    @Override
    public boolean debit(Money amount){
        // ... some logic
    }
}
```

Constructor Injection

Autowiring with Annotations

```java
@Service("clientApp")
public class ClientApp {

    private RemoteBank bank;

    public ClientApp() {}

//  public ClientApp(RemoteBank bank) {
//      this.bank = bank;
//  }

    @Autowired
    public void setBank(RemoteBank bank) {
        this.bank = bank;
    }

    public boolean debit(Money amount) {
        // ... some logic
    }
}
```

```java
@Repository("soaBank")
public class SoaBank extends RemoteBank {

    @Override
    public boolean debit(Money amount){
        // ... some logic
    }
}
```

Setter Injection

Autowiring with Annotations

```java
@Service("clientApp")
public class ClientApp {

    @Autowired
    private RemoteBank bank;

    public ClientApp() {}

//  public ClientApp(RemoteBank bank) {
//      this.bank = bank;
//  }

//  public void setBank(RemoteBank bank) {
//      this.bank = bank;
//  }

    public boolean debit(Money amount) {
        // ... some logic
    }
}
```

```java
@Repository("soaBank")
public class SoaBank extends RemoteBank {

    @Override
    public boolean debit(Money amount){
        // ... some logic
    }
}
```

Field Injection

Autowiring with Annotations

```java
@Service("clientApp")
public class ClientApp {

    @Autowired
    @Qualifier("soaBank")
    private RemoteBank bank;

    public ClientApp() {}

//  public ClientApp(RemoteBank bank) {
//      this.bank = bank;
//  }

//  public void setBank(RemoteBank bank) {
//      this.bank = bank;
//  }

    public boolean debit(Money amount) {
        // ... some logic
    }
}
```

```java
@Repository("soaBank")
public class SoaBank extends RemoteBank {

    @Override
    public boolean debit(Money amount){
        // ... some logic
    }
}
```

Field Injection + Qualifier

Configurating for Annotations

- Just two things:

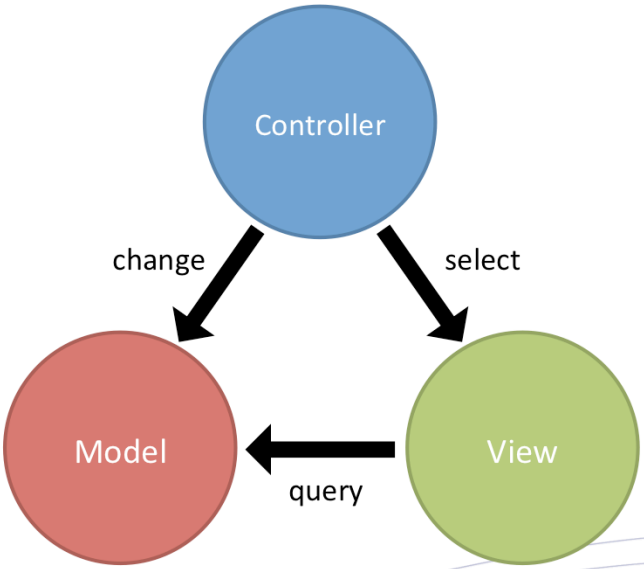
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframe                /beans http
        http://www.springframework.org/schema/                ww.springfr

    <context:annotation-config />
    <context:component-scan base-package="com.improving" />
</beans>
```

Tell Spring to honor annotations

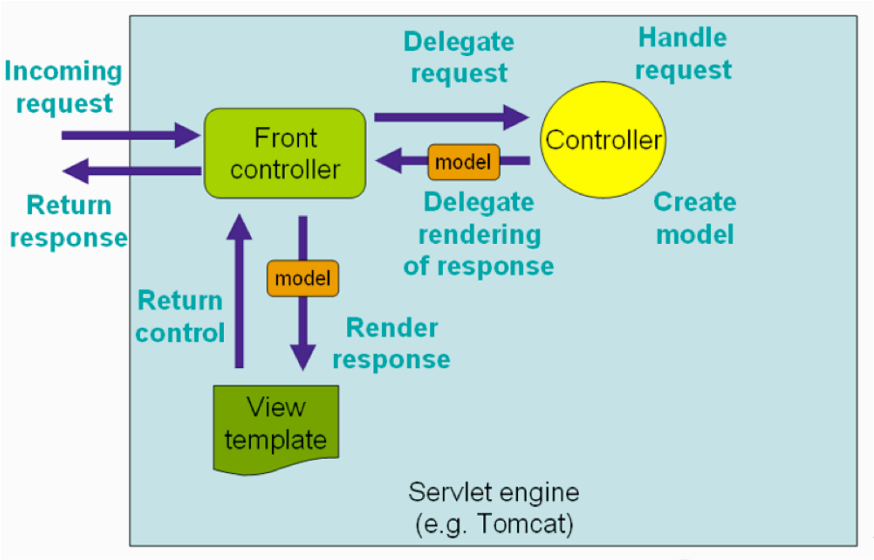
Annotations vs XML

- Configuration in Java or in XML?

MVC Architecture

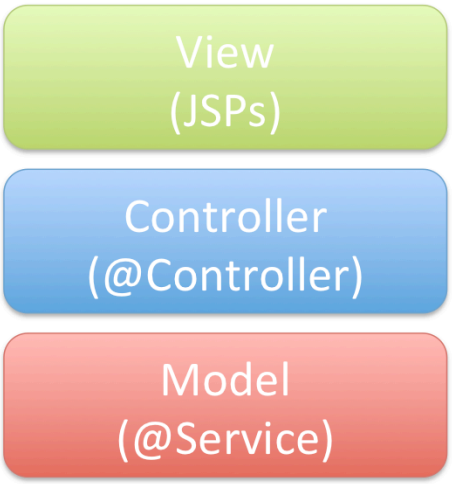


Spring MVC Architecture



http://docs.spring.io/spring-framework/docs/3.0.x/reference/mvc.html

Spring MVC Layers



@Controller

- Simple MVC Controller

Simple POJO with Annotation

```
@Controller
public class HomeController {

    @RequestMapping(value = "/")
    public String home() {

        return "home";
    }
}
```

URL Mapping for this method

Maps to a view (JSP)

home.jsp

```
<html>
<head>
    <title>Home</title>
</head>
<body>
<h2>Welcome Home</h2>
</body>
</html>
```

Spring MVC Configuration (web.xml)

```
<context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<listener>
     <listener-class>org.springframework.web.context.ContextLoaderListener</li
</listener>

<servlet>
     <servlet-name>appServlet</servlet-name>
     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
     <init-param>
          <param-name>contextConfigLocation</param-name>
          <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
     </init-param>
     <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
     <servlet-name>appServlet</servlet-name>
     <url-pattern>/</url-pattern>
</servlet-mapping>
```

Front Controller Servlet

Bean Configuration File for a Spring MVC App

Spring MVC Configuration (config xml)

```
<beans:beans …>

        <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />
    <context:component-scan base-package="com.improving.spring" />

    <resources mapping="/resources/**" location="/resources/" />

    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

</beans:beans>
```

Resolving Controller Mapping to Views

Views (JSPs)

- home.jsp

```
<html>
<head>
    <title>Home</title>
</head>
<body>
<h2>Welcome Home</h2>
</body>
</html>
```

Model Parameter Example

home.jsp

```
<html>
<head>
    <title>Home</title>
</head>
<body>
<h2>Welcome Home, ${name}</h2>
</body>
</html>
```

```
@Controller
public class HomeController {

    @RequestMapping(value = "/")
    public String home(Model model) {

        model.addAttribute("name", "Bob");
        return "home";
    }
}
```

Simple Model (Request Scope)

Model Parameter Example

home.jsp

```
<html>
<head>
    <title>Home</title>
</head>
<body>
<h2>Welcome Home, ${name}</h2>
</body>
</html>
```

```
@Controller
public class HomeController {

    @RequestMapping(value = "/")
    public String home(HttpSession session) {

        session.setAttribute("name", "Bob");
        return "home";
    }
}
```

Session Scope

Model Parameter Example

Adding Model Attributes to Session

```
@Controller
@SessionAttributes("name")
public class HomeController {

    @RequestMapping(value = "/")
    public String home(Model model) {

        model.addAttribute("name", "Bob");
        return "home";
    }
}
```

home.jsp

```
<html>
<head>
    <title>Home</title>
</head>
<body>
<h2>Welcome Home, ${name}</h2>
</body>
</html>
```

@RequestMapping

- Default `@RequestMapping()`
- All `@RequestMapping(value="*")`
- Method `@RequestMapping(method = RequestMethod.POST)`
- Content `@RequestMapping(produces = "application/json")`

Common Parameters

```
@RequestMapping(value = ”/{id}")
public String login(@PathVariable("id”) int id) {
```

Others:

- @RequestParam("id")
- @ModelAttribute
- @Valid

Spring Form Tags

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
```

<form:

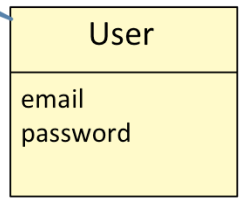
- checkbox
- checkboxes
- errors
- form
- hidden
- input
- label
- option
- password
- select

Spring Form Tags Example

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<html>
<head>
     <title>Home</title>
</head>
<body>
<h2>Welcome Home, ${name}</h2>

<form:form commandName="user" action="login">
Email: <form:input path="email"/>
Password: <form:password path="password"/>
<input type="submit" value="Login"/>
</form:form>

</body>
</html>
```

Maps to a Bean

Spring MVC Forms Taglib



Spring Form Tags Example

```
@RequestMapping(value = "/")
public String home(HttpSession session, Model model) {
     model.addAttribute("name", "Bob");
     model.addAttribute("user", new User());
     return "home";
}

@RequestMapping(value = "/login")
public String login(@ModelAttribute User user) {
     try {
          business.login(user.getEmail(), user.getPassword());
     }catch (InvalidLoginException e) {
          return "home";
     }

     return "forward:menu";
}
```

User created for form load

Shortcut for accessing model attributes

Forward to another controller (instead of a view)

Resource Bundles

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<html>
<head>
    <title>Home</title>
</head>
<body>
<h2><spring:message code="title.text" />, ${name}</h2>

<form:form commandName="user" action="login">
<spring:message code="label.email" />: <form:input path="email"/>
<spring:message code="label.password" />: <form:password path="password"/>
<input type="submit" value="Login"/>
</form:form>

</body>
</html>
```

messages.properties

```
title.text=Welcome home
label.email=Email
label.password=Password
```

Resource Bundles: Configuration

servlet-context.xml

```
<beans:bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
     <beans:property name="basename" value="classpath:messages" />
</beans:bean>
```

Looks for messages.properties in the classpath

improving
It's what we do. ™

SPRING VALIDATION

Spring Validation

Validator Interface & ValidationUtils

or

JSR-303 & Hibernate Validator

(Latest Java Standard for Validation)

Spring Validation

- From Bean

```
public class User {

    @Email
    @NotBlank
    private String email;

    @Pattern(regexp="^(?=.*\\d).{4,8}$")
    @NotBlank
    private String password;

    public User() { }
    …
}
```

Bean Validator Constraints

@AssertFalse
@AssertTrue
@DecimalMax
(value=,inclusive=)
@DecimalMin
(value=,inclusive=)
@Digits(integer=,fraction=)
@Future
@Max(value=)
@Min(value=)
@NotNull
@Null
@Past
@Pattern(regexp=,flag=)
@Size(min=, max=)
@Valid

Recursively Validates Object

Additional Hibernate Constraints

@CreditCardNumber(ignoreNonDigitCharacters=)
@EAN
@Email
@Length(min=, max=)
@LuhnCheck(startIndex=, endIndex=,checkDigitIndex=, ignoreNonDigitCharacters=)
@Mod10Check(multiplier=,weight=,
startIndex=, endIndex=,checkDigitIndex=, ignoreNonDigitCharacters=)
@Mod11Check(threshold=,startIndex=,
endIndex=, checkDigitIndex=,ignoreNonDigitCharacters=, treatCheck10As=,treatCheck11As=)
@NotBlank
@NotEmpty
@Range(min=, max=)
@SafeHtml(whitelistType=, additionalTags=, additionalTagsWithAttributes=)
@ScriptAssert(lang=, script=, alias=)
@URL(protocol=, host=, port= regexp=, flags=)

Spring Validation

- From Controller

```
@RequestMapping(value = "/login")
public String login(@Valid @ModelAttribute User user, BindingResult result) {

    if (result.hasErrors()) {
        return "home";
    }

    try {
        business.login(user.getEmail(), user.getPassword());
    }catch (InvalidLoginException e) {
        result.addError(new FieldError("user", "email", "Invalid Login!"));
        return "home";
    }

    return "forward:menu";
}
```

Front end error handling (prima facie)

Business logic error handling

Name of bean

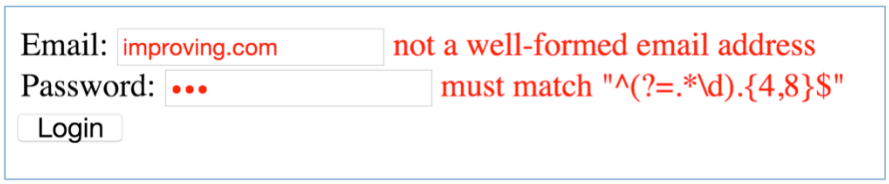
Name of field

Message

Handling Errors

- form:errors tag

```
Email: <form:input path="email" cssErrorClass="error"/>
<form:errors path="email" cssClass="error"/> <br>
Password: <form:password path="password" cssErrorClass="error"/>
<form:errors path="password" cssClass="error"/> <br>
```



Handling Errors

- Custom error messages:

messages.properties

```
title.text=Welcome home
label.email=Email
label.password=Password
Pattern.user.password=Password must be between 4 and 8 digits long and include at least one numeric digit.
```