



# The Spring Framework

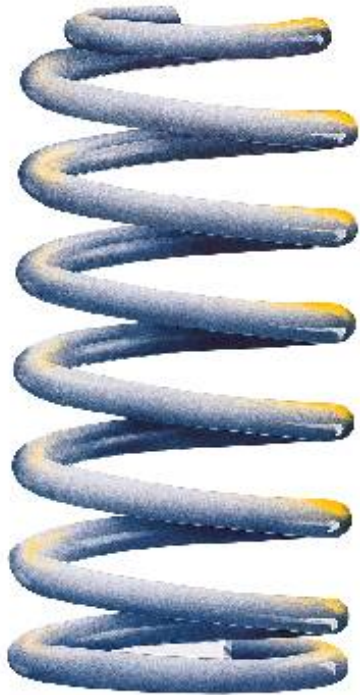
---

## A CeleritasWorks™ Introduction



# What is Spring?

---



- $F = -kx$  (physics)
- One of 4 Seasons
- An Open-Source, Application Framework



# Spring Application Framework

---

Spring is an open source, lightweight, application framework that is intended to help structure entire applications in a consistent manner, pulling together best of breed single-tier frameworks in a coherent architecture.



# Spring Framework Overview

---

- **When considering frameworks think about...**
  - Complexity of solution
  - Timeline
  - Maintainability
  - Familiarity with framework
  - Community and Documentation
  - Framework Licensing



# Spring Framework Overview

---

- ***Do we really need yet another Java framework?***
  - JavaEE does a fine job of standardizing low-level infrastructure but is not and easily usable view for application code
  - Many JavaEE applications in practice are overly complex and take excessive effort to develop
    - *EJB is hard, JNDI is non-intuitive, JDBC is verbose, etc.*
    - *Much of JavaEE development is configuration, setup and “plumbing” code*
- ***Spring aims to make JavaEE development easier.***
  - *Spring provides enterprise services to Plain Old Java Objects (POJOs).*
  - Spring has a nice balance between constraint and freedom. A good framework should provide guidance with respect to good practice, making the right think easy to do, but should not be overly restrictive placing requirements on code using it causing lock in and constraining developers in inappropriate ways.

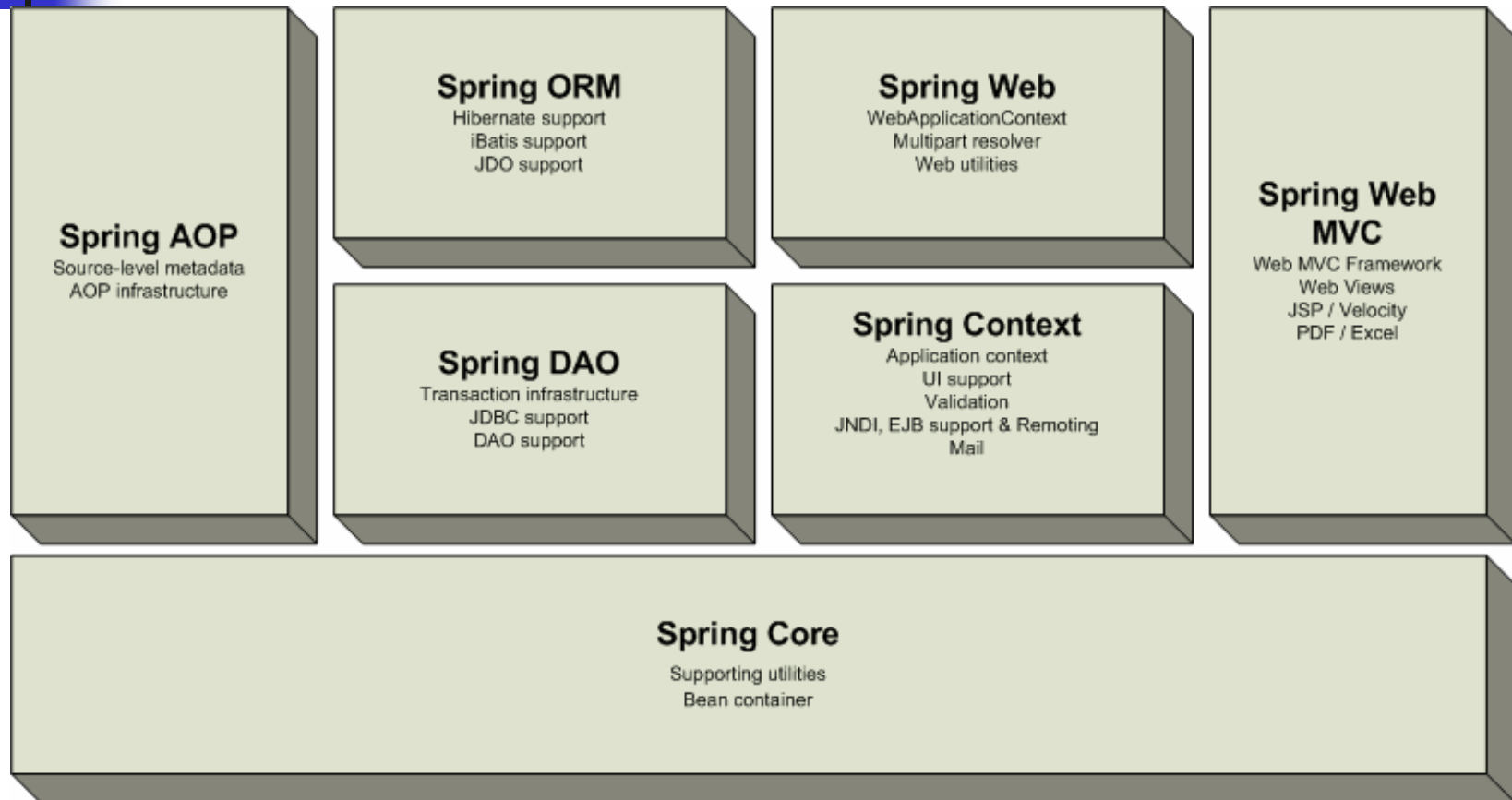


# Spring Framework Overview

---

- ***Modules of the Framework***
  - *IoC Container*
  - *Aspect-Oriented Programming Framework (AOP)*
  - *Data access abstraction and JDBC simplifications*
  - *Transaction Management*
  - *MVC web framework*
  - *Simplification for working with J2EE APIs such as JNDI, JTA, etc.*
  - *Lightweight remoting, JMS support, JMX support*
  - *Support for a comprehensive testing strategy for application developers*

# Spring Framework Modules



3



# Spring Framework Overview

---

- ***Spring's Values include***

- Non-invasive framework – code works outside framework, minimal lock-in, easy migration
- Consistent model in any environment
- Promotes code reuse and deferment of architectural decisions.
- Facilitates OO code and good practices such as programming to interfaces
- Extraction of configuration to consistent xml model
- Promotes Architectural choice – choose best of breed frameworks
- Does not reinvent the wheel – O/R, logging, etc.



# Spring Framework Overview

---

- ***Spring also...***

- **allows for Inversion of Control (IoC)**

- *The Hollywood Principle, "Don't call us, we'll call you." IoC can be thought of in terms of what distinguishes a framework from library.*
  - A Library performs some work when called and returns to caller. Framework encapsulates some abstract design incorporated with behavior but to use you must incorporate you unique behavior via call backs or sub-classing.
- *IoC is a principle that is used to wire an application together, how dependencies or object graphs are created.*

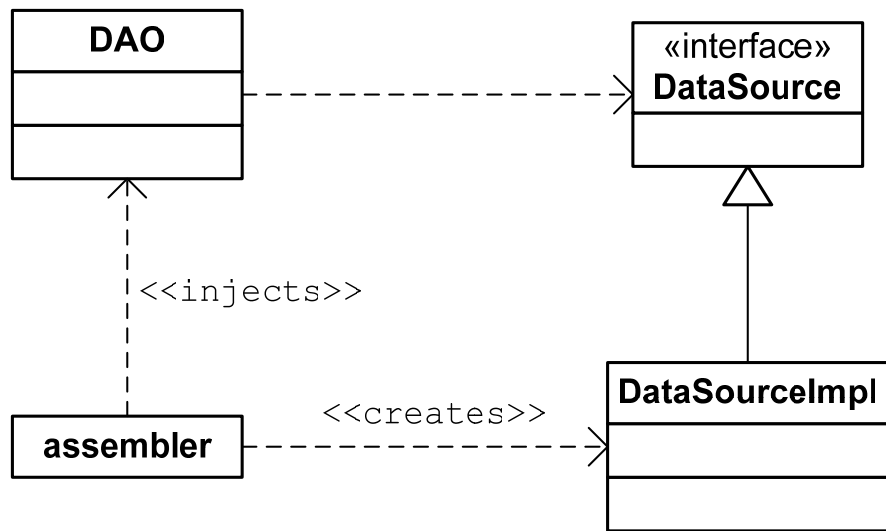


# Spring Dependency Injection

---

- ***In Spring, the IoC “flavor” is referred to as Dependency Injection.***
  - Theoretically – D.I. is based on Java language constructs rather than framework specific interfaces. Application classes expose their dependencies through methods or constructors that the framework can call with appropriate values at runtime, based on configuration.<sup>1</sup>

# Dependency Injection Example



## Example:

```
Class DAO {
    private DataSource datasource;
    public DAO(DataSource ds) {
        datasource = ds;
    }
    private Connection getConnection() {
        return datasource.getConnection();
    }
    ...
}
```

**With Dependency Injection, unlike a service locator pattern (JDNI), there is no dependency on the service locator mechanism. Dependencies are more apparent with Injection.**



# Spring Dependency Injection

- ***3 Types of Dependency Injection***
  - ***Constructor injection: previous example***
  - ***Property (setter injection).***

```
Class DAO {  
    private DataSource datasource;  
    public void setDataSource(DataSource ds) {  
        datasource = ds;  
    }  
    private Connection getConnection() {  
        return datasource.getConnection();  
    }  
    ...  
}
```

- ***Lookup-method injection***



# Spring Dependency Injection

---

- ***To use Spring Dependency Injection all you need is...***
  - *POJO with correct constructor (or setter)*
  - *Spring bean defined in spring-config.xml*
  - *Access the Bean through the Spring context bean factory.*



# Dependency Injection Examples

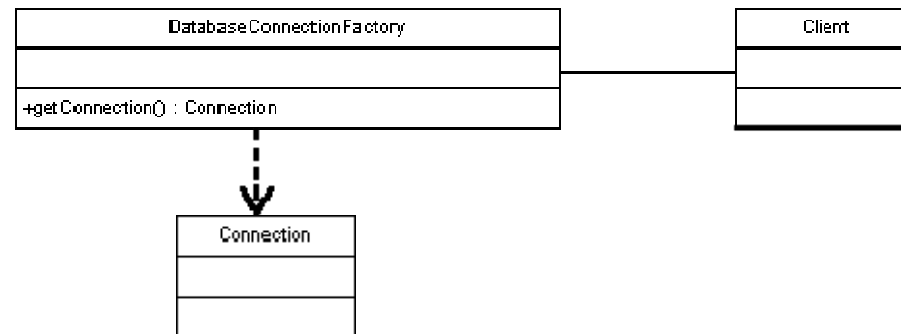
---

- ***POJOs***
  - *Constructor*
  - *Setter*
- ***Declarative Dependency Injection with Spring Beans***
  - *Constructor Injection*
  - *Setter Injection*
  - *Lookup Method Injection*
- ***Bean Factory Lookup***

```
SomeClass instance =  
    (SomeClass) context.getBean("beanName");  
Where context is an implementation of  
    org.springframework.beans.factory.BeanFactory
```

# Spring Bean Lookup Factory

- *The Factory Pattern: one object is responsible for creating and maintaining the lifecycle of another object.*
- The lifecycle of Spring Beans (singleton) is controlled by the Bean Factory.



**Factory Pattern Example**



# Spring Framework Overview

---

- ***Accessing the Spring Container***

- ***From Stand-alone***

```
ApplicationContext context = new
    ClassPathXmlApplicationContext(APP_CONTEXT_CLASSPATH);
(APP_CONTEXT_CLASSPATH is spring/pam-contact-loader-spring-context.xml)
```

- ***From Web Application***

```
ApplicationContext context =
    WebApplicationContextUtils.getWebApplicationContext(
        pRequest.getSession().getServletContext());
```

- ***web.xml modification***

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/classes/spring/pam-spring-context.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
```



# Spring's Practical Usage

---

- **Problems**
  - configuration
  - ad hoc implementation
  - tightly coupled objects
- **Celeritas Examples**
  - ComponentConfiguration
  - classname factory pattern
  - tight coupling: InitialContext()
- **Cleaner solutions with Spring**
  - Spring-config.xml
  - BeanFactory inherent to framework
  - Dependency Injection allows for loose coupling of object dependencies



# Spring's JDBC Api

---

- ***Why not just use JDBC***
  - Connection management
  - Exception Handling
  - Transaction management
  - Other persistence mechanisms
- **Spring Provides a Consistent JDBC Abstraction**
  - Spring Helper classes
    - JdbcTemplate
    - Dao Implementations
    - Query and BatchQueries



# Spring's JDBC Api

---

- ***Examples***

- *Traditional JDBC Dao: ContactDAO*
- *Spring JDBC Dao using JdbcTemplate: ContactLoadDAO*



# Spring Transactions

---

- *Transaction Overview*
  - Atomic, Consistent, Isolated, Durable (ACID)
    - Atomicity and Isolation are of most concern to us as developers. Consistency and Durability are more related to the Transaction source.
- *Similar advantages with Transactions as Spring's JDBC DAO has with DB.*
  - *TransactionTemplate class*
- *Alternative independent of DB Transaction Management and Transaction Management implementation*



# Spring Transactions

---

- Spring Transaction vs Connection.commit() and rollback()

- PlatformTransactionManager Example

```
PlatformTransactionManager transactionManager;  
TransactionDefinition tranDef = new DefaultTransactionDefinition();  
TransactionStatus tranStatus = transactionManager.getTransaction(tranDef);  
try {  
    ...  
    transactionManager.commit(tranStatus);  
} catch (Exception ex) {  
    transactionManager.rollback(tranStatus);  
}
```

- *TransactionTemplate could be used in this example as well.*
    - *Declarative Transactions*



# Spring AOP

---

- ***Three pillars of Spring: IoC, Consistency of Service Abstraction, AOP.***
- ***What is AOP***
  - *“Aspect-Oriented Programming (AOP) complements OOP by providing another way of thinking about program structure. While OO decomposes applications into a hierarchy of objects, AOP decomposes programs into aspects or concerns. This enables modularization of concerns such as transaction management that would otherwise cut across multiple objects. (Such concerns are often termed crosscutting concerns.)”<sup>3</sup>*



# Spring AOP

---

- ***How does Spring enable and make use of AOP***
  - *To provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management, which builds on Spring's transaction abstraction.*
  - *To allow users to implement custom aspects, complementing their use of OOP with AOP.*

# Declarative Spring Transactions

```
<bean id="transactionManager" class=
    "org.springframework.orm.hibernate.HibernateTransactionManager">
  <property name="sessionFactory">
    <ref bean="sessionFactory"/>
  </property>
</bean>
```

**Declare transaction manager**

6

```
<bean id="studentService" class=
"org.springframework.transaction.interceptor.
  TransactionProxyFactoryBean">
```

```
<property name="target">
  <ref bean="studentServiceTarget"/>
</property>
```

← **Apply transactions**

```
<property name="transactionAttributes">
  <props>
```

```
    <prop key="registerForCourse">
      PROPAGATION_REQUIRES_NEW, ISOLATION_DEFAULT
```

**Declare  
transaction**

```
    </prop>
  </props>
</property>
```

```
<property name="transactionManager">
  <ref bean="transactionManager"/>
</property>
```

← **Inject transaction**

```
</bean>
```



# Spring Propagation/Isolation

---

- Propagation Settings
  - PROPAGATION\_MANDATORY
  - PROPAGATION\_NESTED
  - PROPAGATION\_NEVER
  - PROPAGATION\_NOT\_SUPPORTED
  - PROPAGATION\_REQUIRED
  - PROPAGATION\_REQUIRES\_NEW
  - PROPAGATION\_SUPPORTS
- Isolation Settings:
  - ISOLATION\_DEFAULT
  - ISOLATION\_READ\_COMMITTED
  - ISOLATION\_READ\_UNCOMMITTED
  - ISOLATION\_REPEATABLE\_READ
  - ISOLATION\_SERIALIZABLE



# References

---

1. "Professional Java Development with the Spring Framework", Rod Johnson et al., Wiley 2005. ISBN-0-7645-7483-3
2. <http://www.martinfowler.com/>, Martin Fowler.
3. <http://www.springframework.org/>
4. "Introduction to the Spring Framework", Rod Johnson.  
<http://www.theserverside.com/articles/article.tss?l=SpringFramework>,  
May 2005
5. <http://www.picocontainer.org/>
6. [http://www.developer.com/java/ejb/article.php/10931\\_3500516\\_2](http://www.developer.com/java/ejb/article.php/10931_3500516_2)
7. [http://www.oracle.com/technology/pub/articles/marx\\_spring.html](http://www.oracle.com/technology/pub/articles/marx_spring.html)



# Celeritas Technologies

---



7101 College Blvd, Sixth Floor  
Overland Park, KS 66210  
913.491.9000

[www.celeritas.com](http://www.celeritas.com)

<http://java.celeritas.com>