SPRING FRAMEWORK

AGENDA

- Overview
- IOC and Dependency Injection
- Application Context
- Spring data access
- Aspect oriented programming
- Transactions
- Spring MVC
- REST using Spring

WHAT IS SPRING?

Spring is the most popular application development framework for enterprise Java.

Millions of developers use Spring to create high performing, easily testable, reusable code without any lock-in

OVERVIEW

- History
- Goals
- Spring Modules

SPRING FRAMEWORK HISTORY

- The first version was written by Rod Johnson
- Expert One-on-One J2EE Design and Development
- First released in June 2003
- Milestone releases in 2004 and 2005

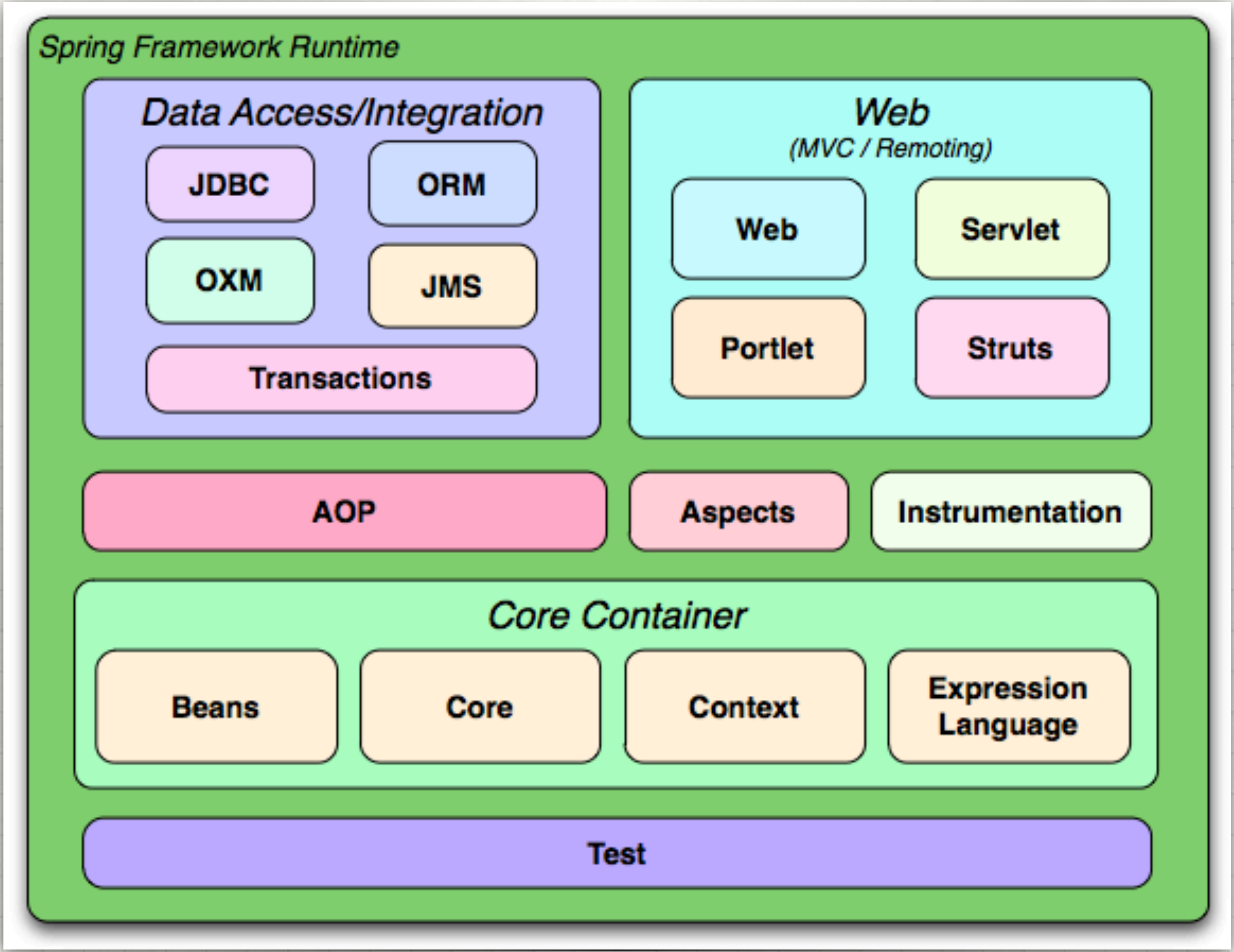
SPRING FRAMEWORK HISTORY

- Spring 2.0 (released Oct 2006)
  - Java 1.3+, AspectJ support, JPA
- Spring 2.5 (released Nov 2007)
  - Java 1.5+, XML namespaces, annotations
- Spring 3.0 (released Dec 2009)
  - REST support, SpEL, more annotations, JavaConfig

GOALS

- Make J2EE easier to use
- Make the common tasks easier
- Promote good programming practice
- You can focus on the domain problems

SPRING MODULES



CORE CONTAINER

- Core and beans: provide the fundamental parts of the framework, including IoC and Dependency Injection features
- Context: it is a means to access objects in a framework-style manner that is similar to a JNDI registry
- Expression language: provides a powerful expression language for querying and manipulating an object graph at runtime

AOP, ASPECT, INSTRUMENTATION

- AOP: provides an AOP Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated
- Aspect: provides integration with AspectJ
- Instrumentation: provides class instrumentation support and classloader implementations to be used in certain application servers

DATA ACCESS/INTEGRATION

- JDBC: provides a JDBC-abstraction layer
- ORM: provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate and iBatis
- OXM: provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream
- JMS: contains features for producing and consuming messages
- Transaction: supports programmatic and declarative transaction management

SPRING WEB

- Web: provides basic web-oriented integration features
- Web-Servlet: Spring's model-view-controller (MVC) implentation
- Web-Struts: contains the classes for integrating a classic Struts WEB tier within a Spring application
- Web-Portlet: provides the MVC implementation to be used in a portlet environment

INVERSION OF CONTROL

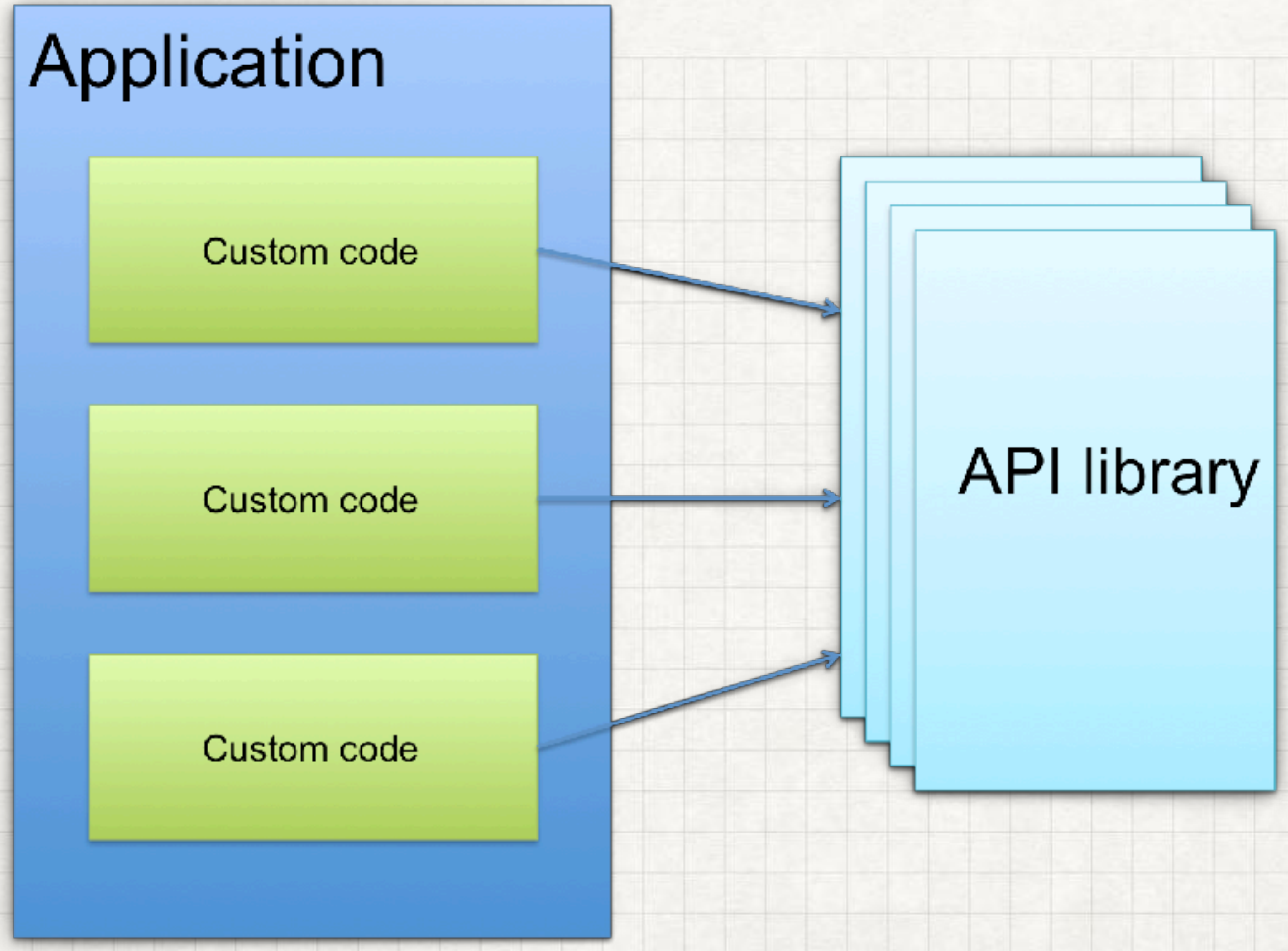
WHAT IS IOC?

- It is a design principle in which custom-written portions of a computer program receive the flow of control from a generic framework

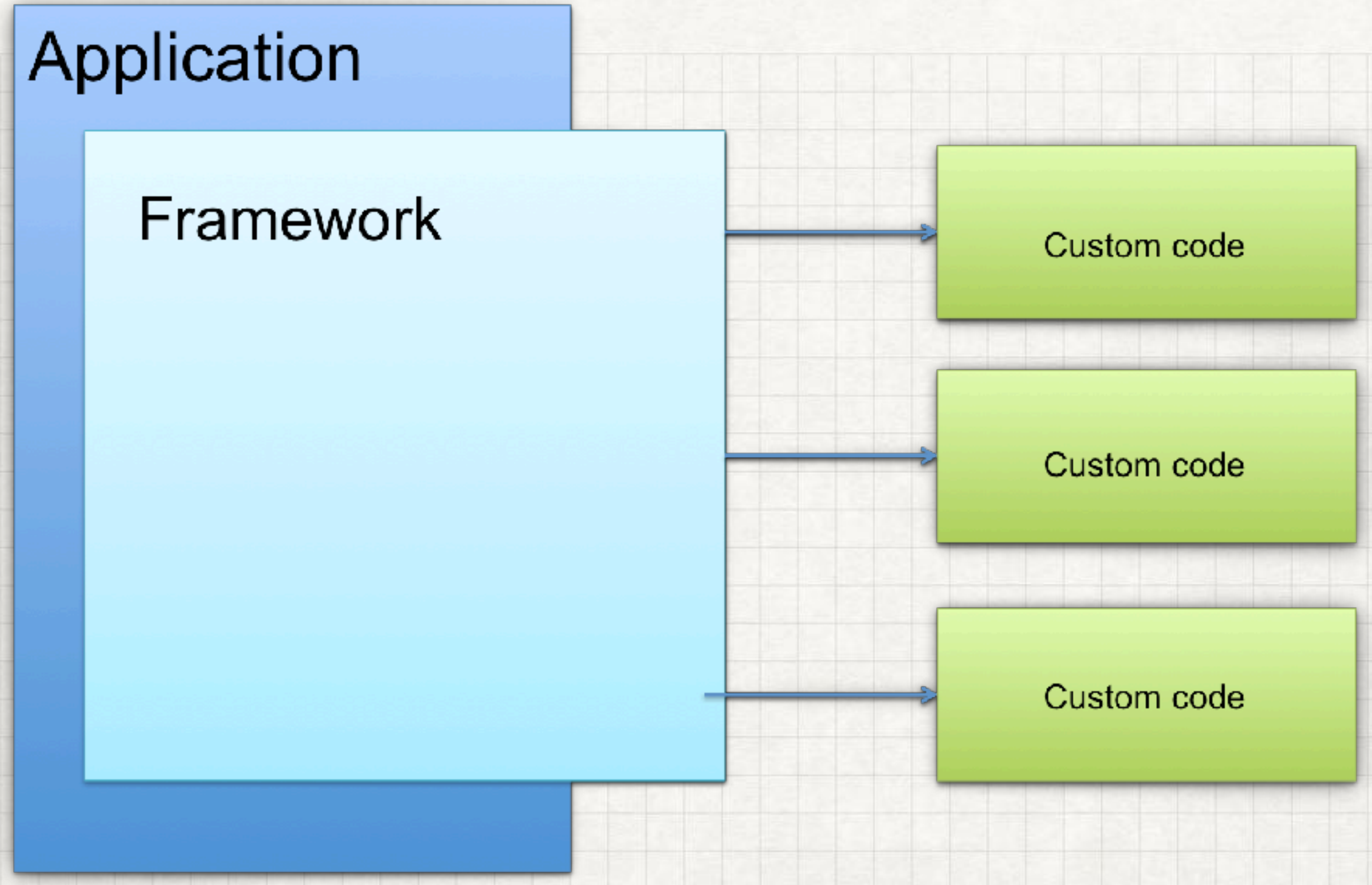
WHAT IS IOC?

- Hollywood style - "don’t call me, I’ll call you"
- one form is Dependency Injection (DI)

TRADITIONAL PROGRAMMING



AN IOC APPLICATION



DEPENDENCY INJECTION

- Dependency Injection is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method

DEPENDENCY INJECTION

- Constructor injection
- Setter injection

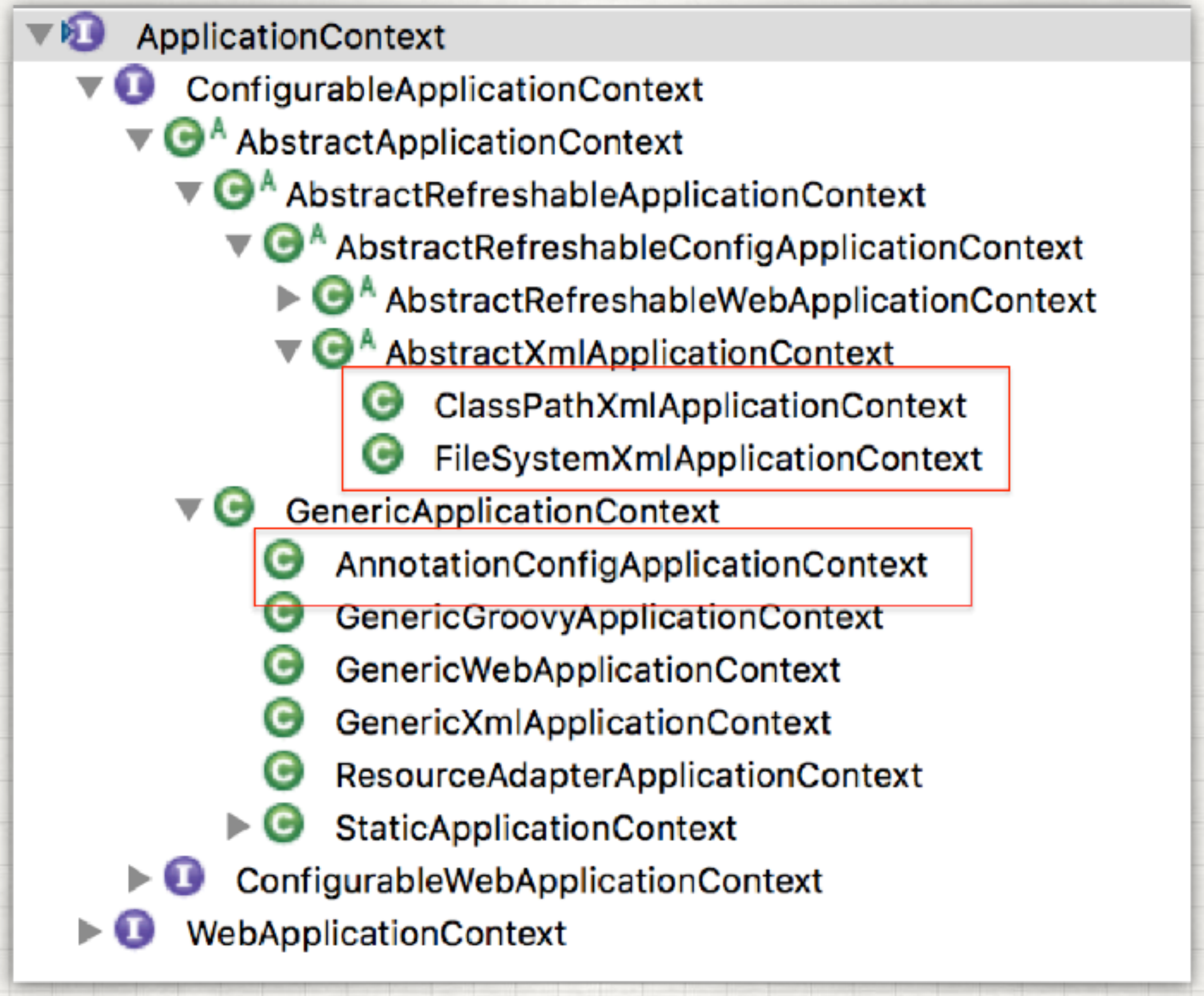
IOC VS DI VS FACTORY

- DI it is specific type of IoC
- The Factory pattern’s main concerns is creating
- The DI’s main concern is how things are connected together
- DI related to Factory and Strategy patterns, but mostly resembles to Build Pattern

TERMS

- *ApplicationContext* represents the Spring IoC container
- *Bean* is an object that managed by Spring IoC container
- *BeanDefinition* describe a bean instance

APPLICATION CONTEXT



```
▼ ApplicationContext
  ▼ ConfigurableApplicationContext
    ▼ AbstractApplicationContext
      ▼ AbstractRefreshableApplicationContext
        ▼ AbstractRefreshableConfigApplicationContext
          ▶ AbstractRefreshableWebApplicationContext
          ▼ AbstractXmlApplicationContext
              ClassPathXmlApplicationContext
              FileSystemXmlApplicationContext
      ▼ GenericApplicationContext
          AnnotationConfigApplicationContext
          GenericGroovyApplicationContext
          GenericWebApplicationContext
          GenericXmlApplicationContext
          ResourceAdapterApplicationContext
        ▶ StaticApplicationContext
    ▶ ConfigurableWebApplicationContext
  ▶ WebApplicationContext
```

APPLICATION CONTEXT

```
ApplicationContext ctx =

    new ClassPathXmlApplicationContext(“context.xml");


ProductDao dao = ctx.getBean(ProductDao.class);
```

BEAN DEFINITION

```
<?xml version="1.0"?>

<beans>

    <bean name="dao,productDao"

        class="co.vinod.dao.impl.JdbcProductDao">

    </bean>

</beans>
```

PROPERTY BASED INJECTION

```xml
<bean class="co.vinod.dao.impl.JdbcProductDao">

    <property name="driver" value="org.hsqldb.jdbcDriver" />

    <property name="url" value="jdbc:hsqldb:hsql://localhost/training" />

    <property name="username" value="sa" />

    <property name="password" value="" />

</bean>
```

CONSTRUCTOR BASED INJECTION

```
<bean class="co.vinod.dao.impl.JdbcProductDao">

    <constructor-arg value="org.hsqldb.jdbcDriver" />

    <constructor-arg value="jdbc:hsqldb:hsql://localhost/training" />

    <constructor-arg value="sa" />

    <constructor-arg value="" />

</bean>
```

THE "P" NAMESPACE

```
<beans xmlns:p="http://www.springframework.org/schema/p" . . . >



<bean

   class="co.vinod.dao.impl.JdbcProductDao"

    p:driver="org.hsqldb.jdbcDriver"

    p:url="jdbc:hsqldb:hsql://localhost/training"

    p:username="sa" p:password="" />
```

THE "C" NAMESPACE

```
<beans xmlns:c="http://www.springframework.org/schema/c” . . . >


<bean
   class="co.vinod.dao.impl.JdbcProductDao"
    c:driver="org.hsqldb.jdbcDriver"
    c:url="jdbc:hsqldb:hsql://localhost/training"
    c:username="sa" c:password="" />
```

WIRING BEANS

```
<bean id="dbcp"
    class="org.apache.commons.dbcp.BasicDataSource"
    p:driverClassName="org.hsqldb.jdbcDriver"
    p:url="jdbc:hsqldb:hsql://localhost/training"
    p:username="sa" p:password=""
    p:initialSize="5" p:maxActive="50"
    p:maxIdle="10" p:minIdle="5" p:maxWait="15" />


<bean id="dao"
    class="co.vinod.dao.impl.JdbcProductDao"
    p:dataSource-ref="dbcp" />
```

AUTO WIRING OPTIONS

```
<bean id="dbcp" name="dataSource"
    class="org.apache.commons.dbcp.BasicDataSource"
    p:driverClassName="org.hsqldb.jdbcDriver"
    p:url="jdbc:hsqldb:hsql://localhost/training"
    p:username="sa" p:password="" />


<bean id="dao"
    scope="prototype"
    class="spring.training.dao.impl.JdbcPersonDao"
    autowire="byName" />
```

ANNOTATION BEAN DEFINITIONS

```
@Configuration
public class AppConfig {
    @Bean(name="dao", autowire=Autowire.BY_NAME)
    public JdbcPersonDao jdbcPersonDao(){
        return new JdbcPersonDao();
    }
    @Bean
    public BasicDataSource dataSource(){
        BasicDataSource bds = new BasicDataSource();
        // bds.set...
        return bds;
    }
}
```

COMPONENTS

- @Component
- @Repository
- @Service
- @Controller
- @RestController (version 4.x)

@COMPONENT

- Indicates that an annotated class is a "component"
- Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning
- Introduced in version 2.5

@REPOSITORY

- Indicates that an annotated class is a "Repository", originally defined by Domain-Driven Design (Evans, 2003) as "a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects"

- Teams implementing traditional J2EE patterns such as "Data Access Object" may also apply this stereotype to DAO classes, though care should be taken to understand the distinction between Data Access Object and DDD-style repositories before doing so

- This annotation is a general-purpose stereotype and individual teams may narrow their semantics and use as appropriate

- As of Spring 2.5, this annotation also serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning

- Introduced in version 2.0

@SERVICE

- Indicates that an annotated class is a "Service", originally defined by Domain-Driven Design (Evans, 2003) as "an operation offered as an interface that stands alone in the model, with no encapsulated state."

- May also indicate that a class is a "Business Service Facade" (in the Core J2EE patterns sense), or something similar. This annotation is a general-purpose stereotype and individual teams may narrow their semantics and use as appropriate

- This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning

- Introduced in version 2.5

@CONTROLLER

- Indicates that an annotated class is a "Controller" (e.g. a web controller)
- This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning
- It is typically used in combination with annotated handler methods based on the RequestMapping annotation
- Introduced in version 2.5