**Inversion of control (IOC) / Dependency Injection**

**Loose coupling** is achieved in spring using the technique Inversion of Control.

The objects give their dependencies instead of creating or looking for dependent objects.

BeanFactory just instantiates and configures beans.

ApplicationContext also does that and it provides the supporting infrastructure to enable lots of enterprise-specific features such as transactions and AOP.

**Constructor Injection Vs Setter Injection**

Constructor-injection enforces the order of initialization and prevents circular dependencies.

With setter-injection it is not clear in which order things need to be instantiated and when the wiring is done.

In a typical application there may be hundreds of collaborators with at least as many setter calls to wire them together.

It is easy to miss a few setter calls when wiring the application together.

On the other hand constructor-injection automatically enforces the order and completeness of the instantiated.

Furthermore, when the last object is instantiated the wiring phase of your application is completed.

**Autowiring**

byType, byname, constructor, no Autowiring

A **bean** is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.

**Scopes of beans** – Singleton (default), prototype

Valid in web-aware spring projects like spring mvc - request, session, global.

**Spring bean life cycle**

<bean id="exampleBean" class="examples.ExampleBean" init-method="init"/>

public void **init**()

public class ExampleBean {

public void init() {

// do some initialization work

}

}

public void **afterPropertiesSet**()

public class ExampleBean implements InitializingBean {

public void afterPropertiesSet() {

// do some initialization work

}

}

XML Based

<bean id="exampleBean" class="examples.ExampleBean" destroy-method="destroy"/>

public void **destroy**()

public class ExampleBean implements DisposableBean {

public void destroy() {

// do some destruction work

}

}