**Summary – Spring framework**

The thing that Spring is popular for is dependency injection (which is the same as dependency inversion). It basically injects configured objects into classes that operates those objects in a decoupled manner.

The main implementation of that dependency injection is to have a spring.xml file that defines all the beans that are injected to those operating classes.

spring.xml tags

* **bean** – defines the bean. Has the following parameters:
  + **id** – the bean id (name)
  + **class** – the bean class (with package)
  + **autowire** – Spring supports the following autowiring modes:
    - **no** - It’s the default autowiring mode. It means no autowiring.
    - **byName** - injects the object dependency according to name of the bean. In such a case, the property and bean name should be the same. It internally calls the setter method.
    - **byType** - injects the object dependency according to type. So it can have a different property and bean name. It internally calls the setter method.
    - **Constructor** - injects the dependency by calling the constructor of the class. It calls the constructor having a large number of parameters.
    - **Autodetect** - Spring first tries to autowire by the constructor. If this fails, it tries to autowire by using byType.
  + **scope** – Spring supports the following scope modes:
    - **singleton** - Scopes a single bean definition to a single object instance
    - **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; that is each HTTP request will have its own instance of a bean created off the back of a single bean definition.
    - **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.
  + **parent** – the name of the bean that this bean is inherited from
* **property** – defines a class member in the bean. Has the following parameters:
  + **name** – the name of the property
  + **ref** – which class member the property refers to
* **list** – defines a collection. Has the following parameter:
  + **ref** – defines the items in the collection
* **context:component-scan** - scans all the components under that package and makes them beans
  + **base-package** – the package that the component inside it will become beans

Messaging mechanism

Spring implements a messaging mechanism that helps the developer publish and listen to events. All we need to do is:

1. Construct a message by creating a class that extends ApplicationEvent.
2. To publish the event, one must inject ApplicationEventPublisher object to the class and call publishEvent method with the message as an input.
3. To listen to the event, one must create a class that extends ApplicationListener.

Special annotations

* @Component (annotates a class) – defines a bean (replacing the definition in spring.xml)
* @Autowired (annotates a class member) – defines an object to be injected in the operating class
* @Qualifier(qualifier name) (annotates a setter) - when there are multiple objects in spring.xml, it qualifies the correct object to the operating class. Note that in spring.xml a qualifier should be defined