
loose-coupling
tight-coupling
spring-beans
ApplicationContext
@Bean
@Configuration
java-configuration
dependency-injection
programming-against-interface
POJO
AnnotationApplicationContext
single-responsibility-principle

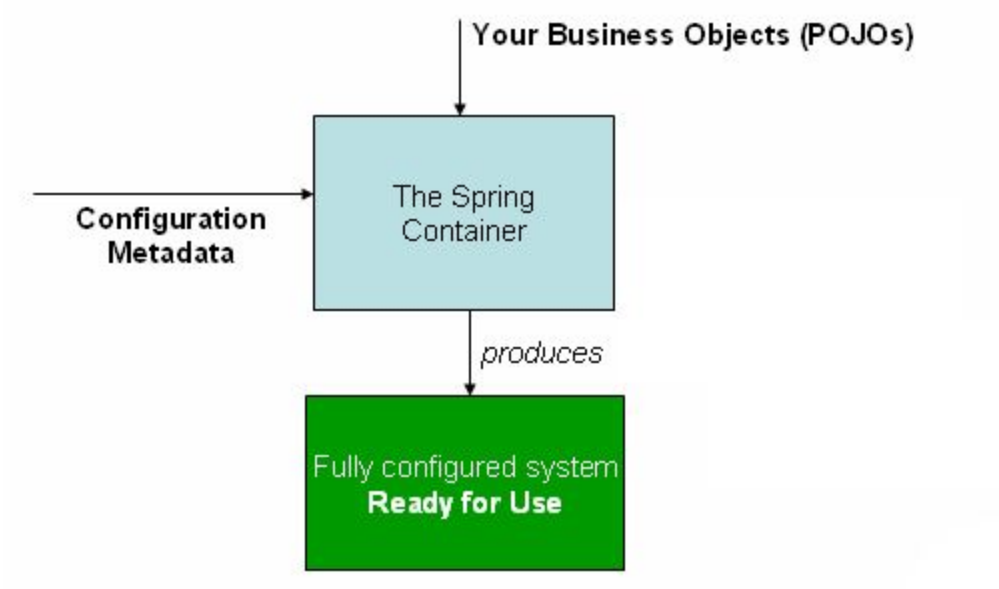
Dependency Injection / Inversion of Control

- **Spring Framework** is an application framework and inversion of control container for the Java platform. *https://en.wikipedia.org/wiki/Spring_Framework*

- **Dependency Injection** implies that clients delegate the dependency resolution to an external service. The client is not allowed to call the injector service, which is why this software pattern is characterized by *Inversion of Control* behavior, also known as the *Don't call us, we'll call you!* principle. - *Pivotal Certified Professional Spring Developer Exam: A Study Guide*

- **Dependency Injection** is a technique whereby one object supplies the dependencies of another object. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The service is made part of the client's state. Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern. *https://en.wikipedia.org/wiki/Dependency_injection*

Spring IoC Container



POJO - Single Responsibility Principle

- **Plain Old Java Object** (**POJO**) is an ordinary Java object, not bound by any special restriction and not requiring any class path *https://en.wikipedia.org/wiki/Plain_old_Java_object*

- **Plain Old Java Object** (**POJO**), simple Java objects each with a single responsibility - *Pivotal Certified Professional Spring Developer Exam: A Study Guide*

- **Single Responsibility Principle** is a computer programming principle that states that every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility. Robert C. Martin expresses the principle as, "A class should have only one reason to change."[1] *https://en.wikipedia.org/wiki/Single_responsibility_principle*

Summary

- **Dependency Injection** is a technique whereby one object supplies the dependencies of another object. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The service is made part of the client's state. Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern. *https://en.wikipedia.org/wiki/Dependency_injection*
- **Coupling -** In software engineering, **coupling** is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are;[1] the strength of the relationships between modules.[2] *https://en.wikipedia.org/wiki/Coupling_(computer_programming)*
- **Loose Coupling** - In computing and systems design a **loosely coupled** system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components. Subareas include the coupling of classes, interfaces, data, and services.[1] Loose coupling is the opposite of tight coupling. - *https://en.wikipedia.org/wiki/Loose_coupling*
- **Programming Against an Interface** - also known as **interface-based architecture**, is an architectural pattern for implementing modular programming at the component level in an object-oriented programming language which does not have a module system. An example of such a language is Java, which (as of 2015), does not have a module system at the level of components. Java has a *package* system, but Java software components typically consist of multiple Java packages – and in any case, interface programming can provide advantages over merely using Java packages, even if a component only consists of a single Java package.https://en.wikipedia.org/wiki/Interface-based_programming

Summary

- Programming Against an Interface - **Interface-based programming**, also known as **interface-based architecture**, is an architectural pattern for implementing modular programming at the component level in an object-oriented programming language which does not have a module system. An example of such a language is Java, which (as of 2015), does not have a module system at the level of components. Java has a *package* system, but Java software components typically consist of multiple Java packages – and in any case, interface programming can provide advantages over merely using Java packages, even if a component only consists of a single Java package.https://en.wikipedia.org/wiki/Interface-based_programming
- **Single Responsibility Principle** is a computer programming principle that states that every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility. Robert C. Martin expresses the principle as, "A class should have only one reason to change."[1] *https://en.wikipedia.org/wiki/Single_responsibility_principle*
- **Plain Old Java Object** (**POJO**), simple Java objects each with a single responsibility - *Pivotal Certified Professional Spring Developer Exam: A Study Guide*

References

- **Dependency Injection**
  - *https://en.wikipedia.org/wiki/Dependency_injection*
  - *https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-basics*
- **Coupling**
  - *https://en.wikipedia.org/wiki/Coupling_(computer_programming)*
- **Programming Against an Interface**
  - https://en.wikipedia.org/wiki/Interface-based_programming
- **Single Responsibility Principle**
  - *https://en.wikipedia.org/wiki/Single_responsibility_principle*
- **Plain Old Java Object** (**POJO**)
  - *Pivotal Certified Professional Spring Developer Exam: A Study Guide*


loose-coupling
tight-coupling
spring-beans
ApplicationContext
@Bean
@Configuration
java-configuration
dependency-injection
programming-against-interface
POJO
AnnotationApplicationContext
single-responsibility-principle