Dependency injection

Jonas Verhoelen

September 21, 2015

Table of contents

What you can expect

- Minimal theory about Dependency injection and all connected topics
- Examples to quickly understand the concepts
- Tasks and exercises for the purpose of self-study

What is Dependency injection?

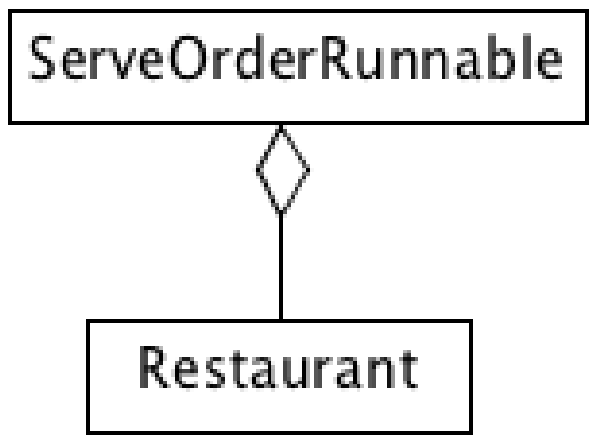
- Design pattern and concept in Object oriented programming
- Get rid of hard-coded dependencies and replace by loose coupling
- Moves the resolution of dependencies from compile-time to runtime
- Very practical to create extendable and maintainable software, especially in big projects
- Part of a lot of Frameworks which take even more workload from you
- ...something you **definitely have to** get familiar with!

Classic coupling (composition)

```
package restaurant;

public class ServeOrderRunnable implements
    Runnable {

    private final Restaurant hostRestaurant;

    public ServeOrderRunnable(Restaurant
        hostRestaurant) {
        this.hostRestaurant =
            hostRestaurant;
    }

    @Override
    public void run() {
        System.out.println("Serving order:
            " + hostRestaurant.getNextMeal()
            + "\n");
```

Classic coupling (aggregation) - explanation

This was a minimal example for a simple aggregation between the classes ServeOrderRunnable and Restaurant. ServeOrderRunnable owns Restaurant but also other classes and objects can own it. The Restaurant is independent from ServeOrderRunnable and does not need it in order to be created or maintained.



Simplifying the situation - DI

```
package restaurant;

public class ServeOrderRunnable implements
    Runnable {

    @Autowired
    private final Restaurant restaurant;

    public ServeOrderRunnable() {
        // Became useless
    }

    @Override
    public void run() {
        System.out.println("Order: " +
            restaurant.getNextMeal() + "\n");
    }
```

Simplifying the situation - DI2

Huh? What happened?
We do not have to pass the Restaurant to the ServeOrderRunnable via the constructor anymore. The instance variable "restaurant" furthermore has a strange annotation above - @Autowired.
It's very simple: the annotation tells the framework (could be Java Spring in this example) that this instance variable should be injected from the available services.
So:

- The class Restaurant is a service
- @Autowired marks that this instance variable should be initialized with an object of the class Restaurant
- A framework or self-written facility maintains instances of services and can inject them

Conclusion from the example

Dependency injection...

- makes dependencies the problems of someone different.
- takes responsibilities from classes (one problem less).
- transfers responsibility to create, maintain and inject dependencies to one central mechanism.
- is handled by a lot of frameworks so you can just use it out of the box without caring.
- also helps you testing, when complex dependencies are involved.

Get started

This workshop contains a ready to go Java-application using the Spring framework. It lets you easily play around with Dependency Injection.

- Clone https://github.com/sebivenlo/dependency-injection.git
- Open a command line and cd into 'workshop'
- Execute './gradlew build'
- Execute 'java -jar build/libs/gs-spring-boot-0.1.0.jar'

The server-app should run now. It is available under the URL localhost:8080/ Open the project in NetBeans now and see what happens. Feel free to play around.