Dependency Injection

spring

Agenda

1. Dependency injection
2. Spring @Bean wiring
3. Choosing your @Bean
4. Spring @Autowire
5. Choosing your @Component
6. Unit tests
7. More…

Dependency Injection

KdG
University of Applied Sciences and Arts

Voorbeeld

https://gitlab.com/kdg-ti/acs-programming-2.2/springknightfactory

If only he could go shopping as well…

```
package org.demo.knight.spring.knights;
public class LadiesKnight implements Hero {
 private RescueDamselQuest quest;

 public LadiesKnight() {
  this.quest = new RescueDamselQuest();
 }

 public void embarkOnMission() {
  quest.embark();
 }
}
// Quest>> God Save the Queen
```

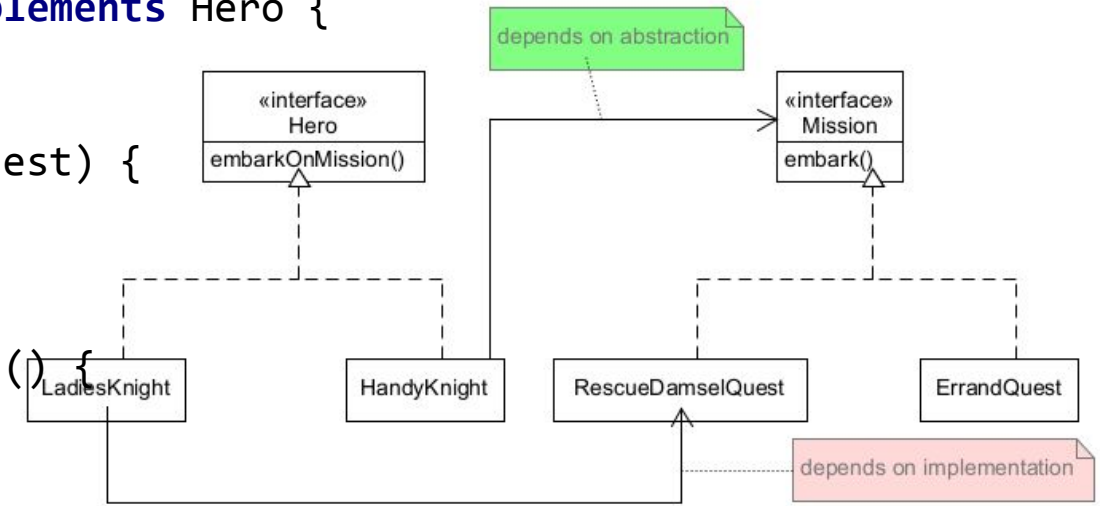
KdG University of Applied Sciences and Arts

Dependency Inversion Principe

```
package
org.demo.knight.spring.knights;
public class HandyKnight implements Hero {
 private Quest quest;

 public HandyKnight(Quest quest) {
  this.quest = quest;
 }

 public void embarkOnMission() {
  quest.embark();
 }
}
```

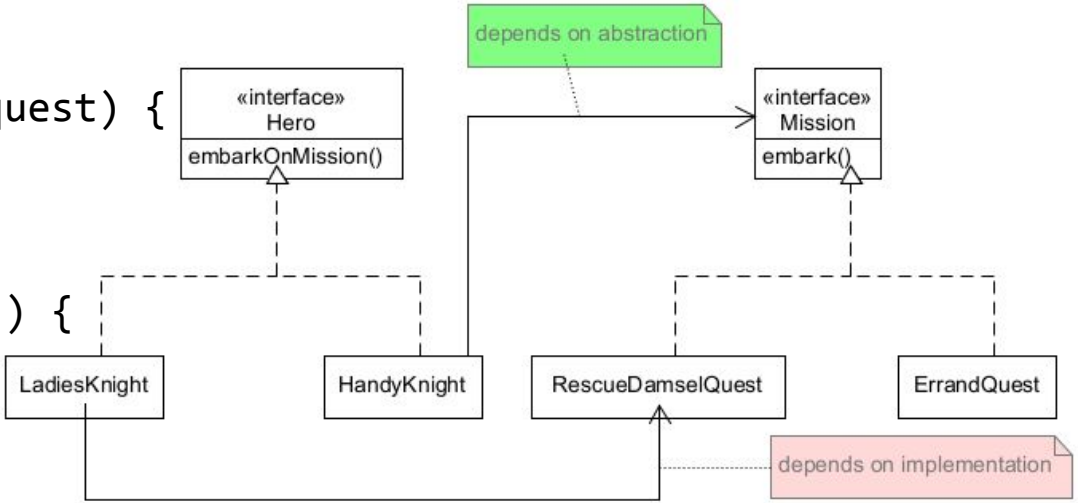


Dependency Inversion Principe

```
package org.demo.knight.spring.knights;
public class HandyKnight implements Hero {
 private Mission quest;

 public HandyKnight(Mission quest) {
  this.quest = quest;
 }

 public void embarkOnMission() {
  quest.embark();
 }
}
public class PlainMain1 {
 public static void main(String[] args) {
  new HandyKnight(new ErrandQuest()).embarkOnMission();
 }
```



depends on implementation

Abstract factory

```java
public class MedievalFactory implements BeanFactory {

  @Override
  public Hero hero(String type) {
    switch (type.toLowerCase()) {
      case "ladies":
        return new LadiesKnight();
      case "handy":
      default:
        return new HandyKnight(new ErrandQuest());
    }
  }
}
```

Use abstract factory

```
public class FactoryMain2 {

 public static void main(String[] args) {
  BeanFactory context = new MedievalFactory();
  context.hero("handy").embarkOnMission();
 }
}
// Quest>> Destroy this message after reading...
```

depends on abstraction

No **dependency** on concrete classsn. They are **injected** by the abstract factory.

@Bean wiring

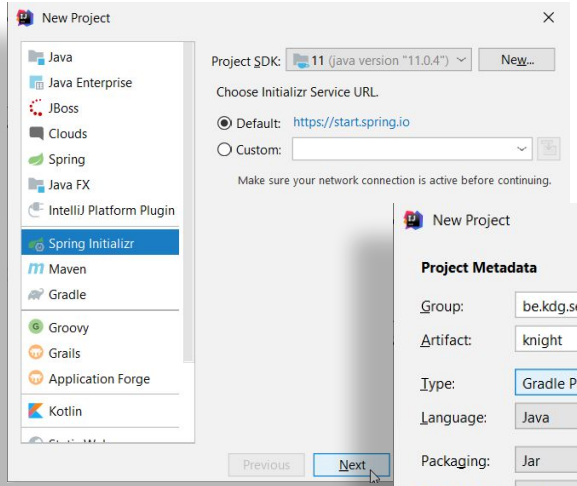
KdG
University of Applied
Sciences and Arts

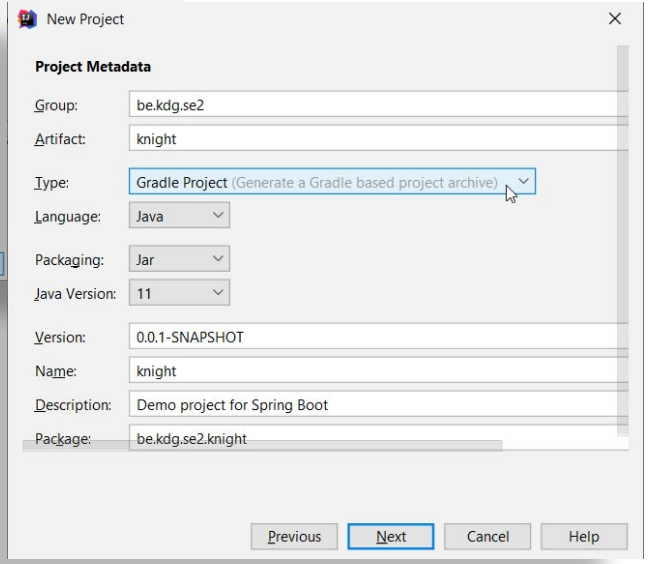
Dependency Injection Frameworks

- Enable dependency injection without factories
  - AKA Dependency Injection Container
    - Behaves like an abstract factory
    - Container manages the lifecycle of injected ojbects
- Examples
  - Enterprise Java Beans
  - Spring
    - Developed as an alternative for Enterprise Java Beans
  - Dagger (Android), Micronaut (microservices) …

Intellij Spring project

- File | new | Spring Initializr





- Choose Type: Gradle project, Next
- Enter groep (package),Next
- Next,Finish

Spring java configuration file

```java
package
org.demo.knight.spring.config;
@Configuration
public class HeroConfig {
  @Bean
  public Hero hero(Mission quest) {
    return new HandyKnight(quest);
  }

  @Bean
  public Mission mission() {
    return new ErrandQuest();
  }
}
```

- Declares beans (components to be injected)
- Injecting one @Bean component into another one is called wiring
- Configuratie can also be done in XML
- To inject another Quest, adapt the configuration.

Using Spring @Beans

```
1  @SpringBootApplication
2  public class SpringMain3 {
3
4    public static void main(String[] args) {
5      ConfigurableApplicationContext context =
6        SpringApplication.run(HeroConfig.class,args);
7      context.getBean(Hero.class).embarkOnMission();
8      context.close(); }
9  }
10 // Quest>> Destroy this message after reading...
```

depends on abstraction

ConfigurableApplicationContext is Spring's abstract factory. Injected @Beans are by default Singletons.

When to use dependency injection?

- You have classes that implement interfaces for which multiple implementations are possible
  - Frameworks and technologies with multiple implementations (persistentence, XML library, REST…)
- DDD Services, repositories…
  - Not for domain objects: DDD entities, Value objects, Events
- Architectural classes

Which beans?

- Java Beans: plain Java classes that conform to some conventions
  - Have a constructor without parameters
  - Attributes are private and have getters en setters with names get*AttributeNaam*, set*AttributeNaam*
  - Many frameworks (JAXB, JPA…) use Java beans
- Enterprise Java Beans and Spring @Bean
  - Components in Dependency Injection frameworks

Choosing your @Bean

spring

KdG
University of Applied Sciences and Arts

Choose your beans

```
package org.demo.knight.spring.config;
@Configuration
public class HeroConfig {
@Bean
public Hero handyKnight(Mission quest) {
  return new HandyKnight(quest);
}
@Bean
public Hero ladiesKnight(Mission quest) {
  return new LadiesKnight();
}
@Bean
public Mission quest() {
  return new ErrandQuest();
}
}
```

Exception in thread "main"
org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'org.demo.knight.spring.knights.Hero' available: expected single matching bean but found 2: handyKnight,knight

Choose your beans

```
@SpringBootApplication
public class SpringMain3 {

  public static void main(String[] args) {
    ConfigurableApplicationContext context =
      SpringApplication.run(HeroConfig.class, args);
    context.getBean("handyKnight", Hero.class).embarkOnMission();
    context.close();
  }
}
```

Add Bean name
(defaults to @Bean method name)

Voorbeeld

https://gitlab.com/kdg-ti/acs-programming-2.2/spring/knight-autowire

@Autowired

@Component

```
@Component
public class ErrandQuest implements Mission {

  @Override
  public void embark() {
    System.out.println(
    "Quest>> Destroy this message after
reading…");
  }
}
```

- Instead of declaring a @Bean in a @Configuration, you can also annotate the class with @Component
  - Some stereotypes (synonyms) for @Component indicating a specific use: @Service, @Repository…
  - Instead of @Component you can also user the standaard annotation @Named

@Component

https://gitlab.com/SE2s1/knight-autowire

```
@SpringBootApplication
public class Pos {

 public static void main(String[] args) {
  SpringApplication.run(Pos.class, args);
 }
}
```

Geen afzonderlijke @Configuration klasse meer nodig

- @SpringBootApplication scans for @Component (and @Configuration) classes in or under its package

@Autowired

```java
@Component
public class HandyKnight implements Hero {
  private final Mission quest;

  @Autowired
  public HandyKnight(Mission quest) {
    this.quest = quest;
  }

  @Override
  public void embarkOnMission() {
    quest.embark();
  }
}
```

@Component: HandyKnight is a Spring Bean

@Autowired: inject the Quest Bean in the HandyKnight constructor

You can replace @Autowired with the standard annotation @Inject

Using Spring @Autowired

```
@SpringBootApplication
public class KnightautoApplication {

 private final Hero knight;

 public static void main(String[] args) {
  SpringApplication.run(KnightautoApplication.class, args);
 }

 @Autowired
 public KnightautoApplication(Hero knight) {
   this.knight= knight;
 }

 @PostConstruct
 private void start() {  knight.embarkOnMission();}
}
// Quest>> Destroy this message after reading...
```

depends on abstraction

@PostConstruct: runs after constructor (and wiring)

Choosing your @Component

KdG
University of Applied
Sciences and Arts

Using Spring @Autowired

```
@SpringBootApplication
public class KnightautoApplication {

 private final Knight knight;

 public static void main(String[] args) {
  SpringApplication.run(KnightautoApplication.class, args);
 }

 @Autowired
 public KnightautoApplication(Knight knight) {
   this.knight= knight;
 }

 @PostConstruct
 private void start() {  knight.embarkOnMission();}
}
// Quest>> Destroy this message after reading…
```

Spring needs to find exactly one @Component
of type Knight.
What if you have more?

@ConditionalOnProperty

- Specify in application.properties which bean is to be used:

```
hero.type=Handy
```

- On your @Component

```
@Component
@ConditionalOnProperty(name="hero.type",havingValue="Handy")
public class HandyKnight implements Hero {
```

```
@Component
@ConditionalOnProperty(name="hero.type",havingValue="Ladies")
public class LadiesKnight implements Hero {
```

Using implementations simultaneously; @Qualifier

```
@Qualifier("Lady")
@Component
public class LadiesKnight implements K
 private RescueDamselQuest quest;

 public LadiesKnight() {
  this.quest = new RescueDamselQuest();
 }

 public void embarkOnMission() {
  quest.embark();
 }
}
```

Without @Qualifier, the @Component Bean name defaults ladiesKnight (classname, lower camelCase)

@Qualifier
public @interface

Make your own Qualifier using anotation

Check out the qualifier branch of the knight-autowire project

https://gitlab.com/kdg-ti/acs-programming-2.2/spring/knight-autowire/-/tree/qualifier

Using Spring @Qualifier

```
@SpringBootApplication
public class KnightautoApplication {

 private final Knight knight;

 public static void main(String[] args) {
  SpringApplication.run(KnightautoApplication.class, args);
 }

 @Autowired
 public KnightautoApplication(@Qualifier("Lady") Knight knight) {
   this.knight= knight;
 }

 @PostConstruct
 private void start() {  knight.embarkOnMission();}
}
// Quest>> God save the Queen!
```

@Autowire
@Component of clase Knight
with @Qualifier "Lady"

Note: It is also possible to make your own Qualifier annotation.
Example: @Lady

Making your own @Qualifier

- Declaration

```
@Qualifier
@Retention(RetentionPolicy.RUNTI
ME)
Public @interface Handy{
}
```

- Usage

```
@Handy
@Autowired
private Hero knight;
```

Test

spring 6

KdG
University of Applied
Sciences and Arts

Junit 5 tests

```
package demo.pos;


@SpringBootTest
class POSTest {

@Autowired
SaleService store;
@Autowired
PosController register;


@Test
void createNewSaleTest()  {
 long saleId = register.makeNewSale();

 assertTrue(saleId > 0);
 Sale s = store.getSale(saleId);
 assertNotNull(s);
 assertNotNull(s.getSalesLineItems());
 assertFalse(s.isComplete());
}
```

@SpringBootTest scans for a @SpringBootApplication annotation in the same package or higher

@Autowire beans

@Test

Application property → what Bean you're using
↳ hero.type = handy
→ annotate using @ conditionalOnProperty (name = "hero.type")

Check out the SpringDI branch of the knight-autowire project

https://gitlab.com/SE2s1/POS/tree/SpringDI

Spring Properties

(↳ having value = "handy")

- The test code uses another mechanism than @Qualifier to select beans: properties files

resources/application.properties

```
repository.type=DB
```

This properties file is read by default by Spring

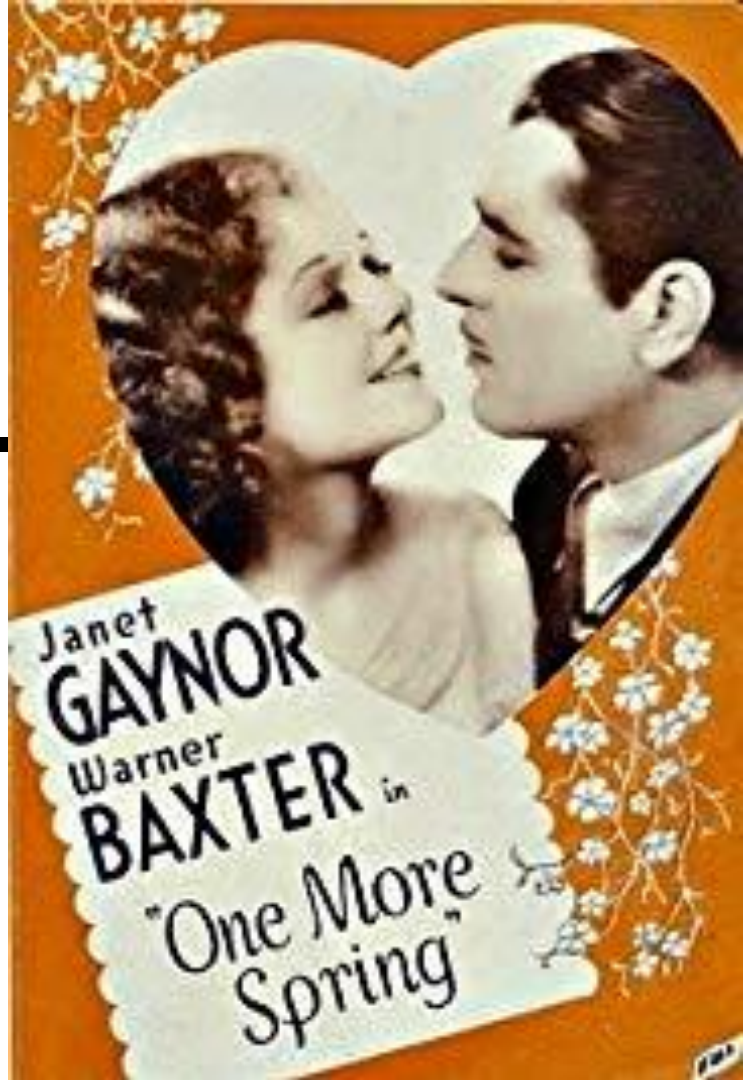
demo/pos/persistence/ProductDescriptionDbRepository.java

```
@Repository
@ConditionalOnProperty(name = "repository.type",
havingValue = "DB")
 public class ProductDescriptionDbRepository implements
ProductDescriptionRepository {…}
```

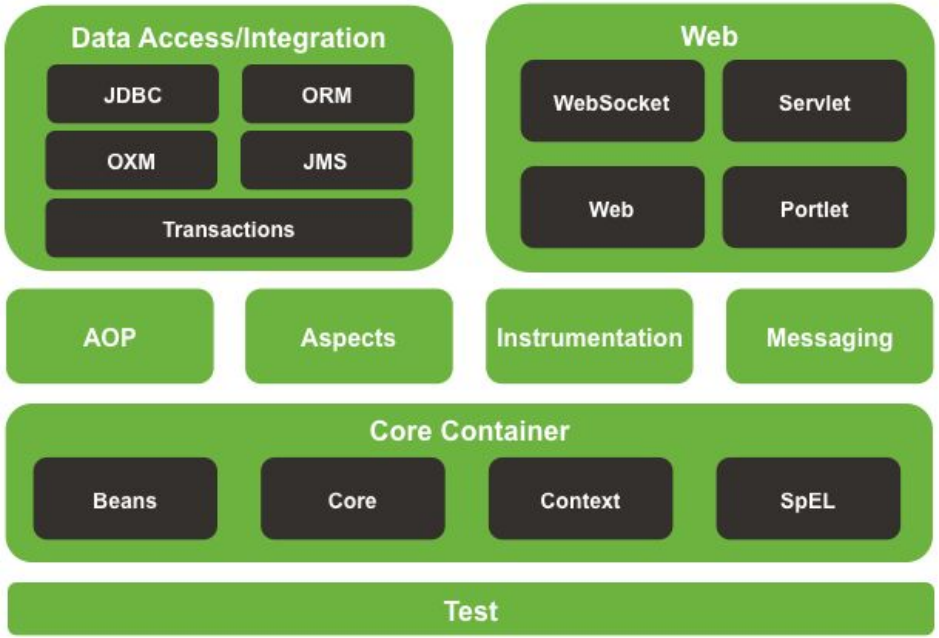
This bean is only created if the property exists with this value

Note: properties are a general java configuration mechanism that is used to configure many other aspects of Spring

More…



Spring framework is more than dependency injection



Overzicht

1. Dependency injection
2. Spring @Bean wiring
3. Choosing your @Bean
4. Spring @Autowire
5. Choosing your @Component
6. Unit tests
7. More…