METODY PROGRAMOWANIA

Odwrócenie sterowania i wstrzykiwanie zależności na przykładzie Spring Framework

13 stycznia 2018

Krzysztof Pawłowski
kpawlowski@pjwstk.edu.pl

CZYM JEST ODWRÓCENIE STEROWANIA?

- Odwrócenie sterowania (*ang. invertion of control*) - zasada projektowa polegająca na przeniesieniu na zewnątrz komponentu (np. obiektu) odpowiedzialności za kontrolę wybranych czynności

- wspiera modularność i rozszerzalność

- przykłady: dependency injection, aspect oriented programming, wzorzec strategy

WSTRZYKIWANIE ZALEŻNOŚCI

- Wstrzykiwanie zależności (*ang. dependency injection*) - proces, w którym obiekty definiują inne obiekty, od których zależą poprzez luźne powiązania

- Jedna z realizacji zasady odwrócenia sterowania

- Wstrzykiwanie możemy realizować przez:

  - konstruktor

  - “settery"

WSTRZYKIWANIE ZALEŻNOŚCI

- "tradycyjne" podejście:

```
public class Foo { }

public class Bar {
  private Foo foo;

  public Bar() {
    this.foo = new Foo();
  }
}
```

WSTRZYKIWANIE ZALEŻNOŚCI

- wstrzykiwanie zależności (DI) poprzez konstruktor:

```
public class Foo { }
```

```
public class Bar {
  private Foo foo;

  public Bar(Foo foo) {
    this.foo = foo;
  }
}
```

WSTRZYKIWANIE ZALEŻNOŚCI

- wstrzykiwanie zależności (DI) poprzez setter:

```
public class Foo { }
```

```
public class Bar {
  private Foo foo;

  public void setFoo(Foo foo) {
    this.foo = foo;
  }
}
```

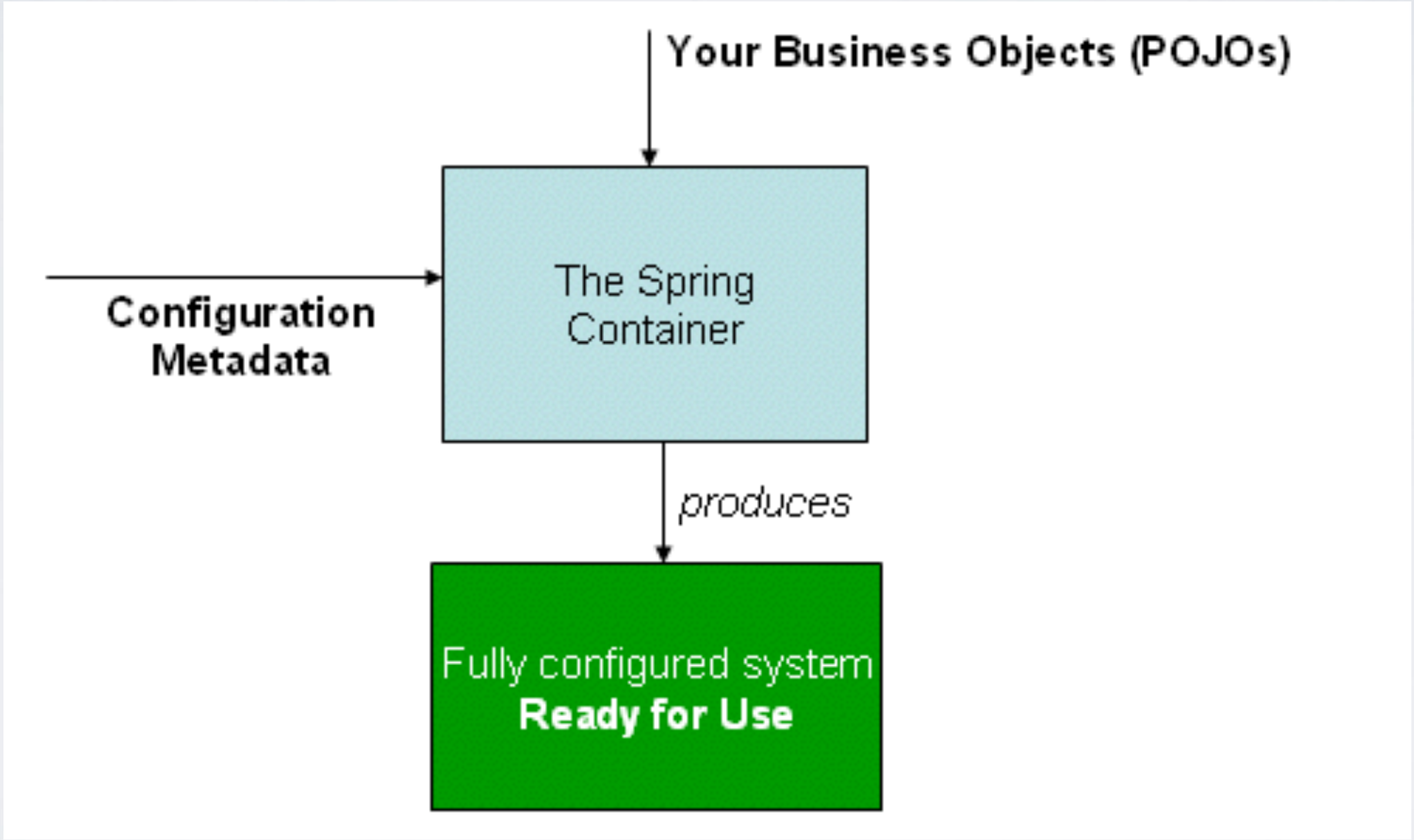
SPRING FRAMEWORK

- Najpopularniejszy framework do wytwarzania aplikacji klasy enterprise w języku Java
- Open source
- Implementuje IoC
- Wspiera serwery aplikacji (aplikacje webowe), zawiera dodatkowe moduły, np.: AOP, Spring Security, Spring Data

IOC CONTAINER

- Spring Container jest rdzeniem Spring Framework
- Kontener jest odpowiedzialny za tworzenie obiektów, ich powiązanie, konfiguracje i zarządzanie ich cyklem życia

IOC CONTAINER



IOC CONTAINER

- Kontener tworzy i konfiguruje obiekty na podstawie metadanych, które mogą być reprezentowane przez:
  - plik konfiguracyjny w formacie XML,
  - kod,
  - adnotacje.

KONFIGURACJA KONTENERA (XML)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- services -->

    <bean id="petStore"
         class="org.springframework.samples.jpetstore.services.PetStoreServiceImpl">
        <property name="accountDao" ref="accountDao"/>
        <property name="itemDao" ref="itemDao"/>
        <!-- additional collaborators and configuration for this bean go here -->
    </bean>

    <!-- more bean definitions for services go here -->

</beans>
```

JAK ZAINICJOWAC KONTENER?

```
public class Application {

  public static void main(String[] args) {

    ApplicationContext context =
      new ClassPathXmlApplicationContext(new String[] {"services.xml"});

  }
}
```

JAK UŻYWAĆ KONTENERA?

```
// create and configure beans
ApplicationContext context =
    new ClassPathXmlApplicationContext(new String[] {"services.xml", "daos.xml"});

// retrieve configured instance
PetStoreService service = context.getBean("petStore", PetStoreService.class);

// use configured instance
List<String> userList = service.getUsernameList();
```

BEAN

- Bean to instancja klasy utworzona i skonfigurowana przez kontener IoC

BEAN - INICJACJA PRZEZ KONSTRUKTOR - PRZYKŁAD 1

```
package x.y;

public class Foo {

    public Foo(Bar bar, Baz baz) {
        // ...
    }

}
```

BEAN - INICJACJA PRZEZ KONSTRUKTOR - PRZYKŁAD 1

```
<beans>
    <bean id="foo" class="x.y.Foo">
        <constructor-arg ref="bar"/>
        <constructor-arg ref="baz"/>
    </bean>

    <bean id="bar" class="x.y.Bar"/>

    <bean id="baz" class="x.y.Baz"/>
</beans>
```

BEAN - INICJACJA PRZEZ KONSTRUKTOR - PRZYKŁAD 2

```
package examples;

public class ExampleBean {

    private int years;

    private String ultimateAnswer;

    public ExampleBean(int years, String ultimateAnswer) {
        this.years = years;
        this.ultimateAnswer = ultimateAnswer;
    }

}
```

BEAN - INICJACJA PRZEZ KONSTRUKTOR - PRZYKŁAD 2

```
<bean id="exampleBean" class="examples.ExampleBean">
    <constructor-arg type="int" value="7500000"/>
    <constructor-arg type="java.lang.String" value="42"/>
</bean>
```

BEAN - INICJACJA PRZEZ SETTERY - PRZYKŁAD

```
public class ExampleBean {

    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;
    private int i;

    public void setBeanOne(AnotherBean beanOne) {
        this.beanOne = beanOne;
    }

    public void setBeanTwo(YetAnotherBean beanTwo) {
        this.beanTwo = beanTwo;
    }

    public void setIntegerProperty(int i) {
        this.i = i;
    }

}
```

BEAN - INICJACJA PRZEZ SETTERY - PRZYKŁAD

```
<bean id="exampleBean" class="examples.ExampleBean">
    <!-- setter injection using the nested ref element -->
    <property name="beanOne">
        <ref bean="anotherExampleBean"/>
    </property>

    <!-- setter injection using the neater ref attribute -->
    <property name="beanTwo" ref="yetAnotherBean"/>
    <property name="integerProperty" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
```

BEAN - ZAGNIEŻDŻANIE

```
<bean id="outer" class="...">
    <!-- instead of using a reference to a target bean,
         simply define the target bean inline -->
    <property name="target">
        <bean class="com.example.Person">
            <!-- this is the inner bean -->
            <property name="name" value="Fiona Apple"/>
            <property name="age" value="25"/>
        </bean>
    </property>
</bean>
```

BEAN - DEFINIOWANIE LISTY

```
<bean id="moreComplexObject" class="example.ComplexObject">
    <!-- results in a setSomeList(java.util.List) call -->
    <property name="someList">
        <list>
            <value>a list element followed by a reference</value>
            <ref bean="myDataSource" />
        </list>
    </property>
</bean>
```

BEAN - DEFINIOWANIE MAPY

```
<bean id="moreComplexObject" class="example.ComplexObject">
    <!-- results in a setSomeMap(java.util.Map) call -->
    <property name="someMap">
        <map>
            <entry key="an entry" value="just some string"/>
            <entry key ="a ref" value-ref="myDataSource"/>
        </map>
    </property>
</bean>
```

BEAN - DEFINIOWANIE ZBIORU

```
<bean id=“moreComplexObject"
class="example.ComplexObject">
    <!-- results in a setSomeSet(java.util.Set) call -->
    <property name="someSet">
        <set>
            <value>just some string</value>
            <ref bean="myDataSource" />
        </set>
    </property>
</bean>
```

AUTOMATYCZNE WYKRYWANIE ZALEŻNOŚCI

- Referencje do innych obiektów w Spring Framework mogą być rozwiązywane automatycznie:
  - poprzez nazwę (“byName”)
  - poprzez typ (“byType”)
  - poprzez konstruktor (“constructor”)

AUTOMATYCZNE WYKRYWANIE ZALEŻNOŚCI - PRZYKŁAD

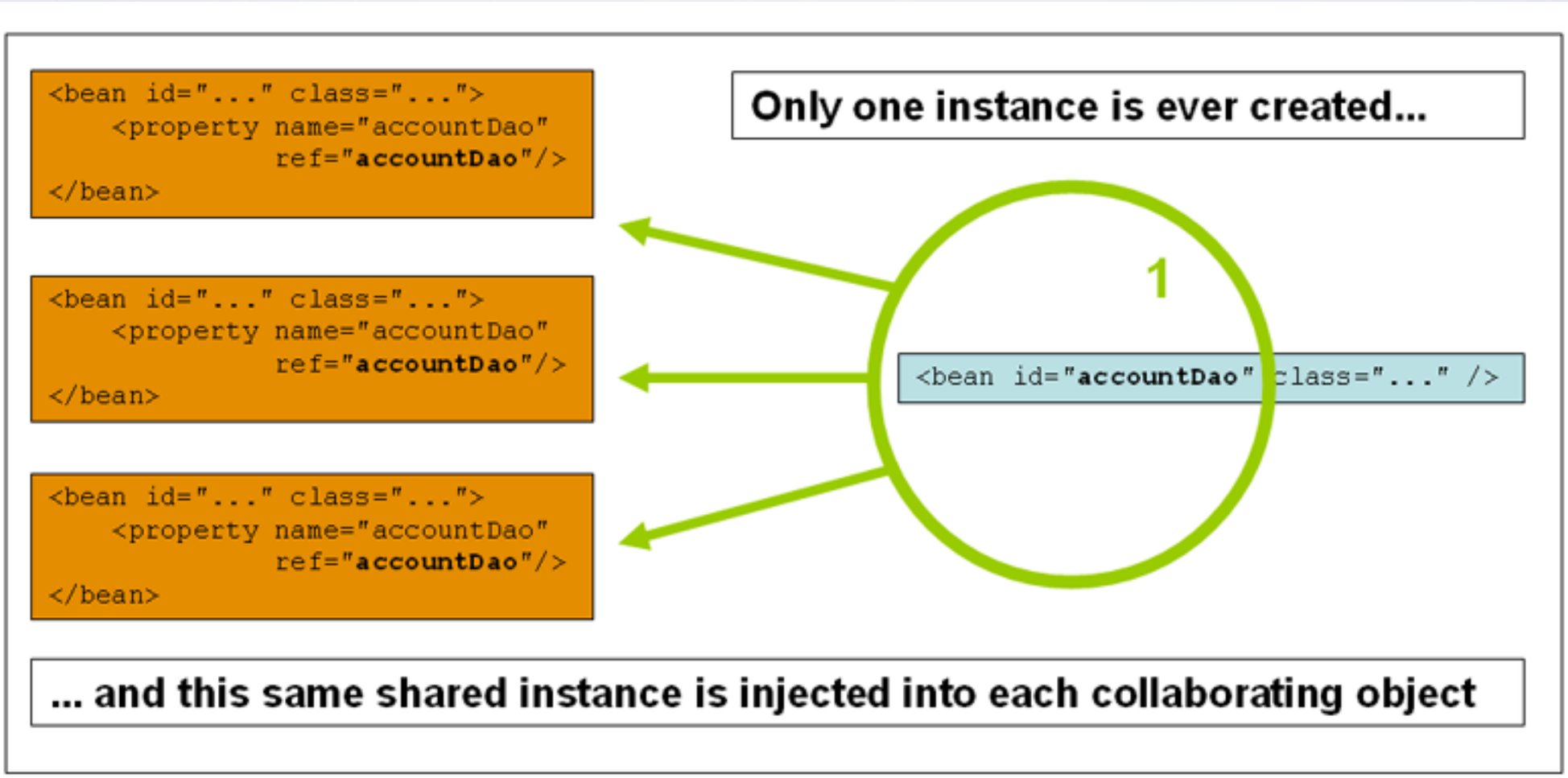
```
<bean id="moreComplexObject" class=“example.ComplexObject"
autowire=“byName” />
```

BEAN - ZAKRES

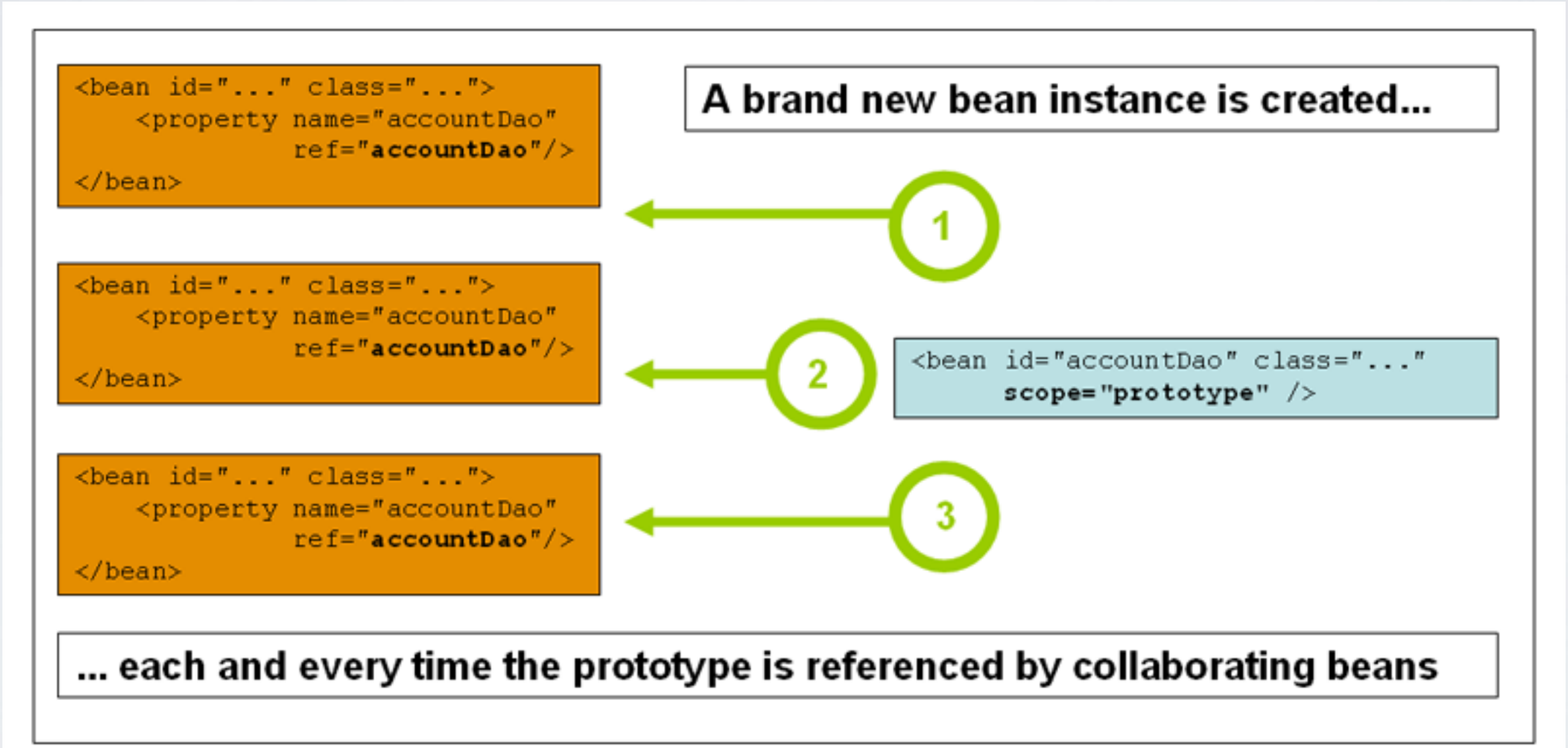
- singleton

- prototype

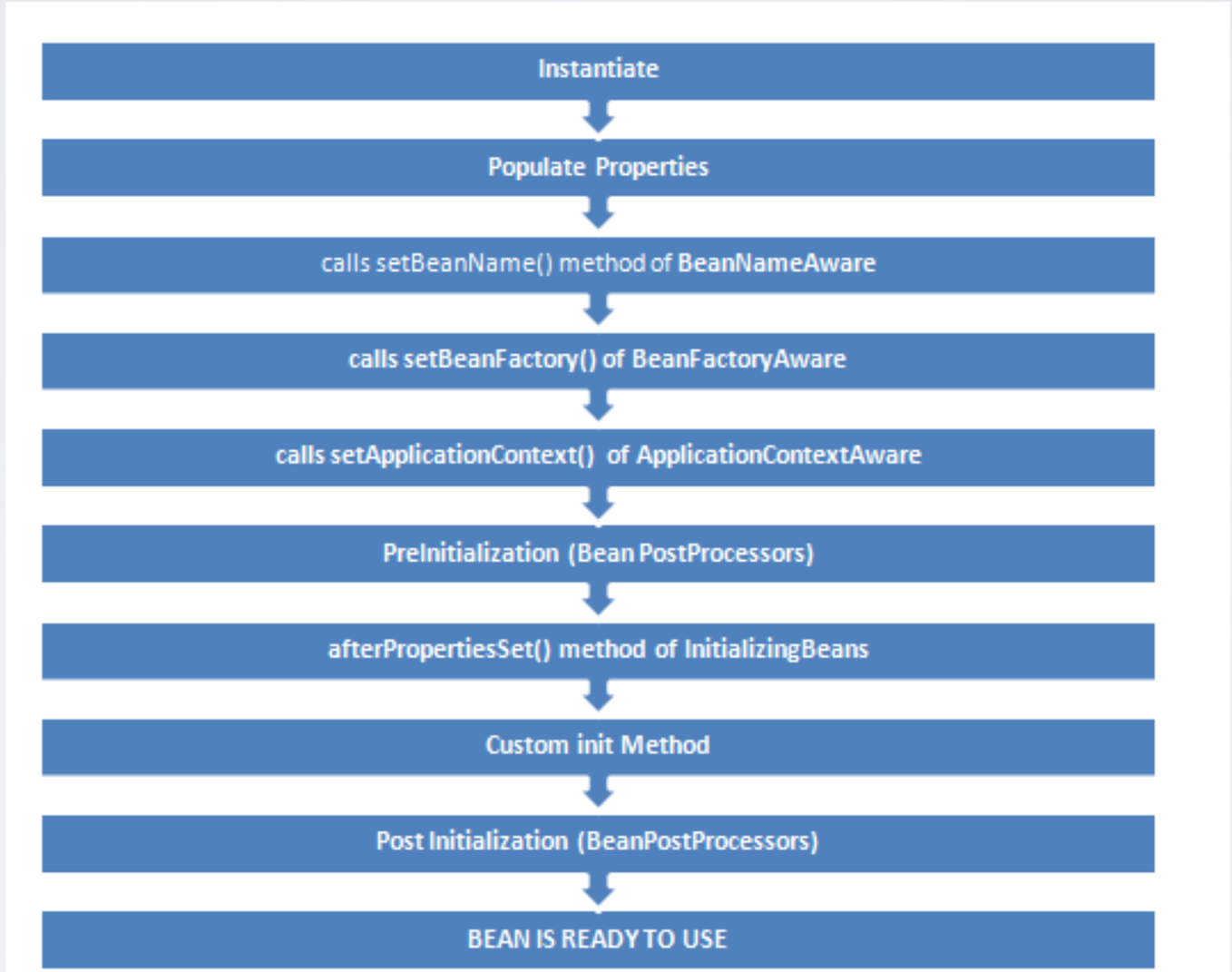
BEAN - ZAKRES SINGLETON



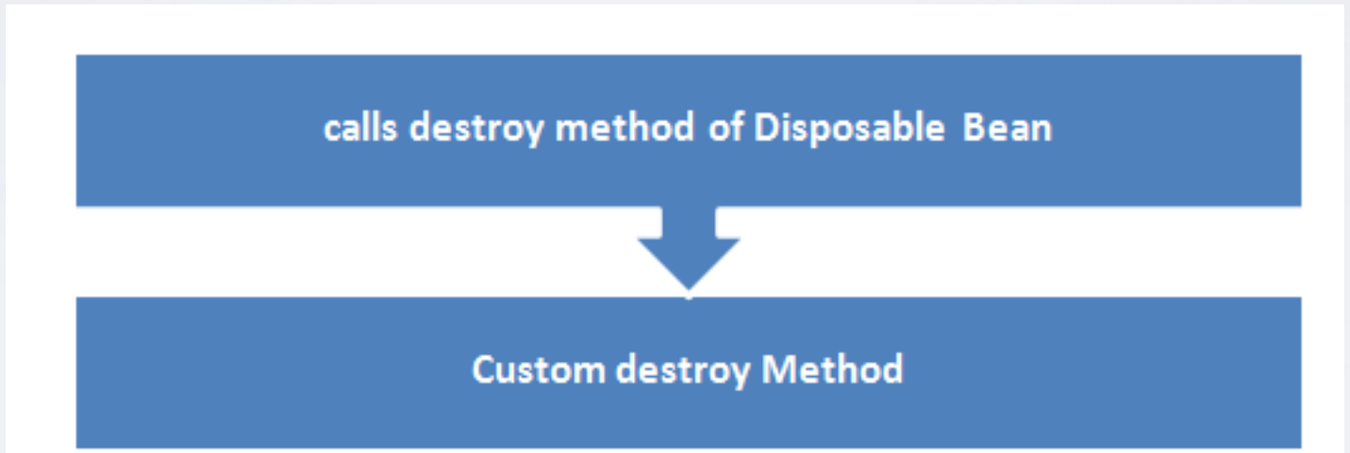
BEAN - ZAKRES PROTOTYPE



BEAN - CYKL ŻYCIA



BEAN - CYKL ŻYCIA



BEAN - CYKL ŻYCIA

```
<bean id="moreComplexObject" class=“example.ComplexObject"
init-method=“init” destroy-method=“destroy” />
```

KONFIGURACJA KONTENERA PRZY UŻYCIU ADNOTACJI

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config/>

</beans>
```

@REQUIRED

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Required
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...

}
```

@AUTOWIRED

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired(required=false)
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...

}
```

@CONFIGURATION, @BEAN, @PRIMARY

```
@Configuration
public class MovieConfiguration {

    @Bean
    @Primary
    public MovieCatalog firstMovieCatalog() { ... }

    @Bean
    public MovieCatalog secondMovieCatalog() { ... }

    // ...

}
```

@QUALIFIER

```
public class MovieRecommender {

    @Autowired
    @Qualifier("main")
    private MovieCatalog movieCatalog;

    // ...

}
```

@RESOURCE

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Resource(name="myMovieFinder")
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

}
```

@POSTCONSTRUCT, @PREDESTROY

```
public class CachingMovieLister {

    @PostConstruct
    public void populateMovieCache() {
        // populates the movie cache upon initialization...
    }

    @PreDestroy
    public void clearMovieCache() {
        // clears the movie cache upon destruction...
    }

}
```

AUTOMATYCZNE SKANOWANIE OBIEKTÓW

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="org.example"/>

</beans>
```

- lub

```
@Configuration
@ComponentScan(basePackages = "org.example")
public class AppConfig  {
    ...
}
```

@COMPONENT

```
@Component
public class FactoryMethodComponent {

    @Bean @Scope("prototype")
    public TestBean prototypeInstance(InjectionPoint injectionPoint) {
        return new TestBean("prototypeInstance for " + injectionPoint.getMember());
    }
}
```

@NAMED, @INJECT

```
import javax.inject.Inject;
import javax.inject.Named;

@Named("movieListener") // @ManagedBean("movieListener") could be used as well
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Inject
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...
}
```

PRZYKŁAD

http://websystique.com/spring/spring-auto-detection-autowire-component-scanning-example-with-annotations/

Dziękuję!

DZIĘKUJĘ