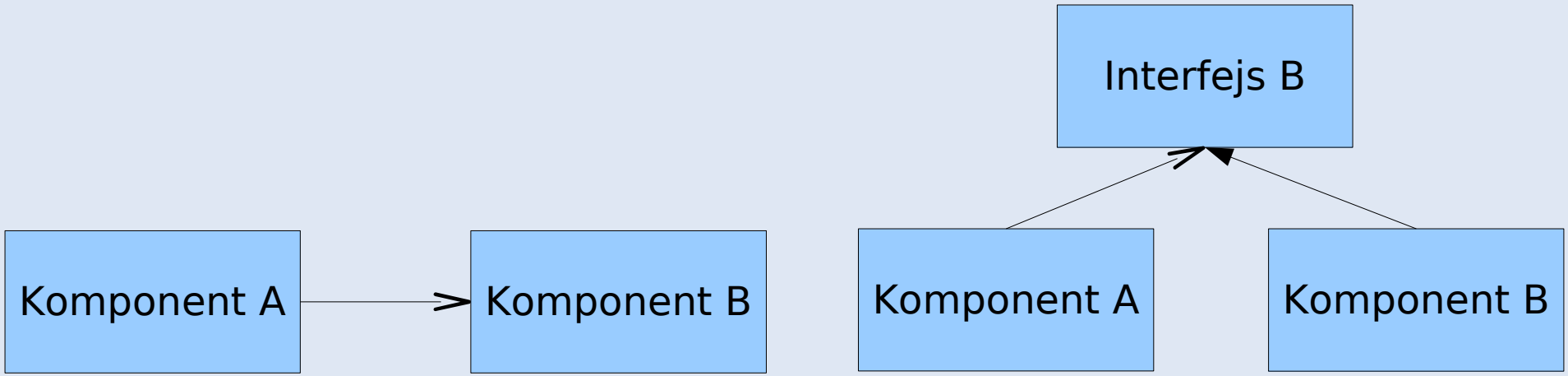
IoC do kwadratu – czyli praktyczny przewodnik programisty

Jarosław Pałka

O co to całe zamieszanie czyli słów kilka o IoC

- wzorzec projektowy
- “Hollywood principle”
- pierwszych śladów należy szukać w artykule “Family Values: A Behavioral Notion of Subtyping” autorstwa Barbary Liskov oraz Jeannette Wing
- **P**lain **O**ld **J**ava **O**bject
- złamanie zależności pomiędzy klasami (poprzez wprowadzenie interfejsu)
- poprzez to narzuca stosowanie technik **S**eparation **O**f **C**oncerns
- **D**ependency **I**njection jako jedna z technik
- od początku postrzegany jako propozycja powstała w opozycji do EJB
- dobre praktyki programistyczne
- “hard coded” versus “XML glue”?



Dependency Injection

Rozróżniane type Dependency Injection według Martin'a Fowler'a (w artykule “Inversion of Control Containers and the Dependency Injection pattern”):

- Constructor Injection

```
 public OrderProcessor(OrderDAO dao){
 this.dao = dao;
}
```

- Setter Injection

```
public void setOrderDAO(OrderDAO dao){
      this.dao = dao;
}
```

- Interface Injection

```
public interface OrderDAOInject {
      void injectOrderDAO(OrderDAO dao);
}
```

- ServiceLocator

IoC w praktyce czyli targowisko próżności

- Spring (http://springframework.org)
- Pico and Nanocontainer (http://picocontainer.org)
- Avalon/Merlin (closed) (http://avalon.apache.org)
- Metro (http://www.dpml.net/metro/concepts/index.html)
- JBoss Microcontainer (http://www.jboss.com/products/jbossmc)
- HiveMind (http://jakarta.apache.org/hivemind/index.html)
- Lazarus (http://lazarus.dev.java.net)
- Yan (http://yan.codehaus.org/)
- Peapod (http://www.peapod.org/)
- Carbon (http://carbon.sourceforge.net/)
- Gravity (http://gravity.dev.java.net/)
- Soto (http://www.sapia-oss.org/projects/soto/)
- Loom (http://loom.codehaus.org/)
- JICE (http://jicengine.sourceforge.net/)
- Xwork (http://www.opensymphony.com/xwork/)
- Seasar (http://www.seasar.org/en/)

Autowiring czyli kto nam zawiąże sznurówki

- automatyczne wiązanie zależności pomiedzy komponentami w kontenerach IoC w znaczący sposób ułatwia pracę w tym środowisku.
- odpowiedzialność za powiązanie komponentów przeniesiona jest na kontener.

Spring

Ręczne wiązanie:

```
<component name="exampleDataAccessObject" class="example.ExampleDataAccessObject">
    <dependencies>
        <property name="dataSource" ref="mySingleDataSource" />
    </dependencies>
</component>
```

Automatyczne wiązanie:

```
<component name="exampleBusinessObject" class="example.ExampleBusinessObject" autowire="byType">
</component>
```

Spring

Ręczne wiązanie:

```
<bean id="exampleDataAccessObject" class="example.ExampleDataAccessObject">
    <property name="dataSource" ref="mySingleDataSource" />
</bean>
```

Automatyczne wiązanie:

```
<bean id="exampleBusinessObject" class="example.ExampleBusinessObject" autowire="byType">
    <property name="exampleParam" value="10" />
</bean>
```

Spring

Ręczne wiązanie:

```
<bean id="exampleDataAccessObject" class="foo.Bar">
    <property name="dataSource" ref="mySingleDataSource" />
</bean>
```

Automatyczne wiązanie:

```
<bean id="exampleBusinessObject" class="example.ExampleBusinessObject" autowire="byType">
    <property name="exampleParam" value="10" />
</bean>
```

Lifecycle czyli co mi zrobisz jak mnie złapiesz

- metody wywoływane podczas inicjalizacji lub niszczenia komponenetów
- umożliwiają wywołanie standartowych akcji takich jak inicjalizacja, wstrzyknięcie konfiguracji itp
- w większości przypadków zdefiniowane jako zestaw interfejsów
- każdy z dostępnych kontenerów dostarcza zestaw standardowych akcji (interfejsów)
- dość często istnieje możliwość rozbudowania cyklu życia obiektów

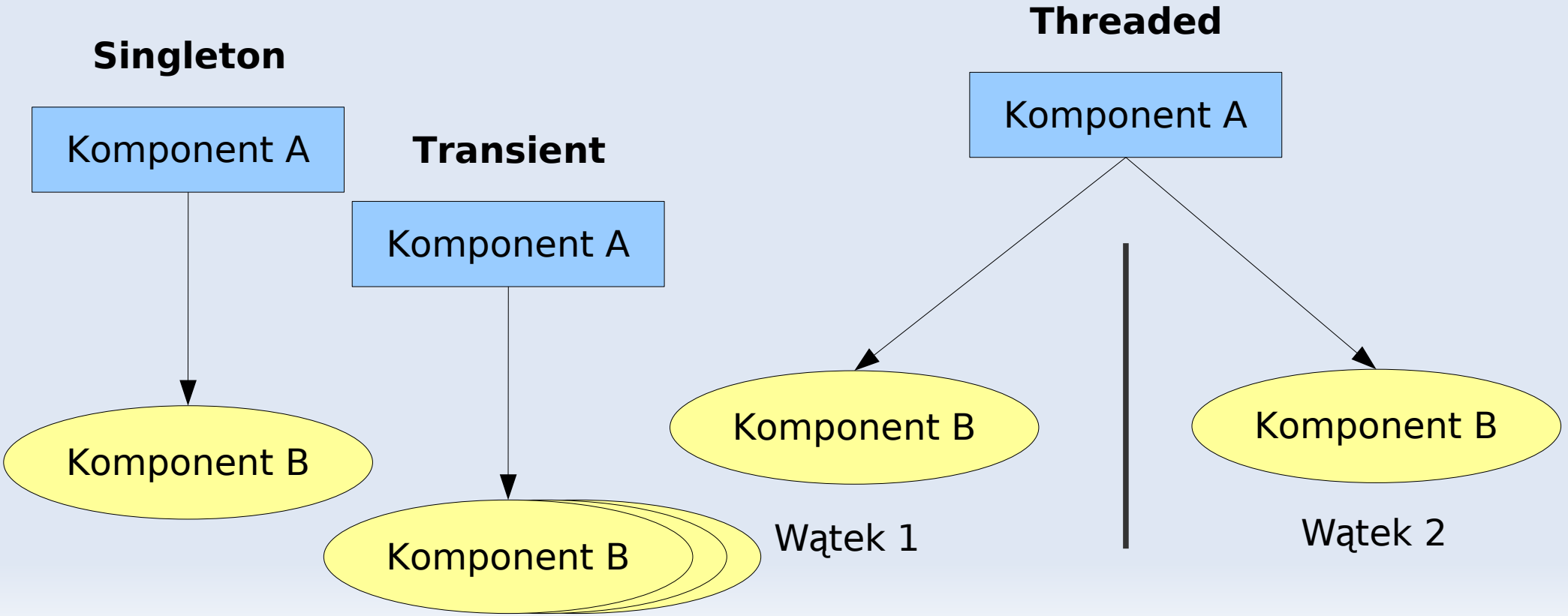
Spring:
org.springframework.beans.factory.InitializingBean
org.springframework.beans.factory.DisposableBean

Lazarus:
net.lazarus.activity.Startable
net.lazarus.configuration.Configurable

Picocontainer:
org.picocontainer.Startable
org.picocontainer.Disposable

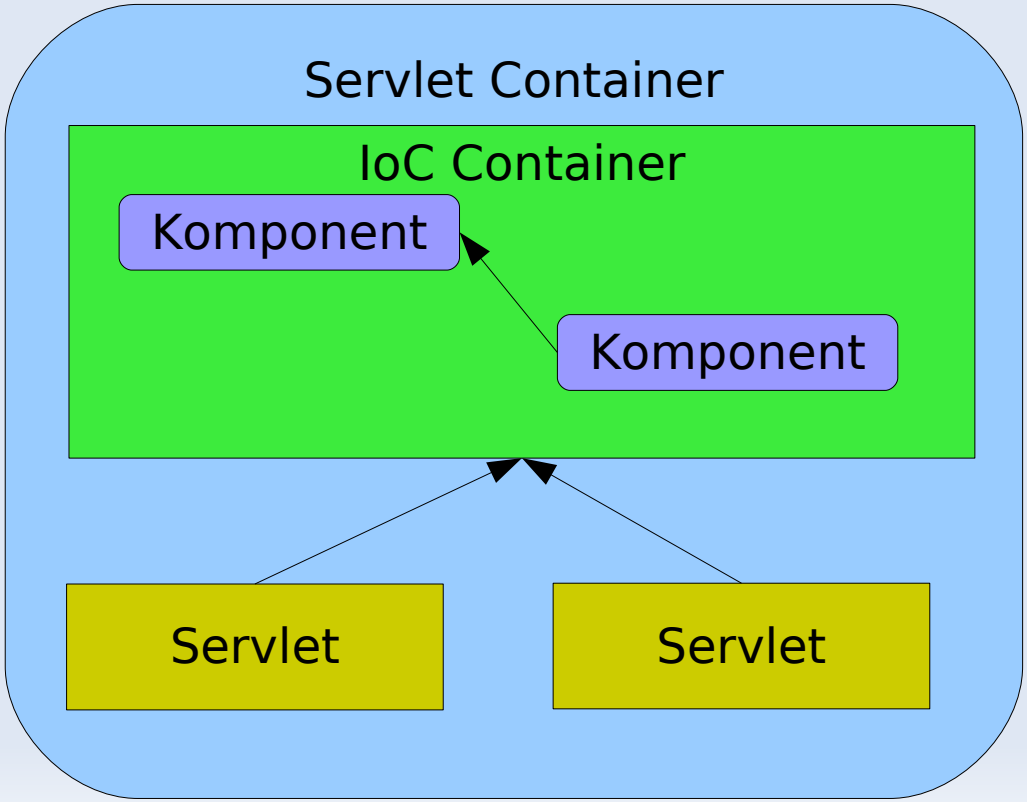
Lifestyle i inne rzeczy, które uprzyjemniają nasze życie

- zarządzanie instancjami komponentów, w większości przypadków kontenery umożliwiają tworzenie wyłacznie Singleton'ów
- dostępne głównie w kontenerach IoC, które korzystają z wzorca ServiceLocator lub Interface Injection
- w innych przypadkach możliwe poprzez wykorzystanie wzorcu Factory
- najczęsciej stosowane to Singleton,Transient i Threaded
- dostępne w Metro,Lazarus oraz Hivemind



Podróż do wnętrza Ziemi czyli osadzanie kontenerów

- kontenery IoC często określane są jako “lekkie” (lightweight containers)
  to określenie dotyczy nie tylko ich rozmiaru i narzutu w trakcie działania
- możliwość uruchomienia kontenera wewnątrz aplikacji lub innego kontenera
- dostęp do komponentów wewnątrz kontenera
- zastosowane w WebWork oraz Shale
- zastosowanie kontererów IoC jak tzw. “black box”, odseparowanie warstwy biznesowej od warstwy prezentacji w obrębie pojedynczej maszyny wirtualnej



Kilka faktów na zakończenie

Oczekiwanie wobec kontenerów:

- wsparcie dla istniejących technologii (Hibernate,SOAP,JMX,JMS, języki skryptowe,JSP/Servlet)
- wsparcie dla AOP
- możliwośc pisania testów (Junit,TestNG)
- wsparcie dla “annotations”
- wsparcie dla modułów
- hierarchia kontenerów
- inne jęzki programowania i IoC

Szeroka akceptacja przez środowisko związane z technologią Java:

- popularność Spring'a
- nowe, lepsze Struts czyli Shale oparte na Springu oraz XWork
- EJB3
- ilość dostępnych implementacji

Skutki uboczne stowanie tej terapii

Zalety:

- modele CRC **(**Class **R**esponsibility **C**ollaborator), jak technika projektowania, szeroko propagowana przez metolodogie z rodziny Agile
- prostsze, zrozumiałe dla wszystkich projekty systemu
- efektywnie krótszy czas “refactoringu”
- duży nacisk na fazę projektowania, jednak bez potrzeby tworzenia “skończonych” projektów (kolejny argument za stosowaniem tego razem z technikami Agile)
- brak narzutu wprowadzanego przez kontenery JEE
- możliwość utworzenia środowiska aplikacji według własnych potrzeb

Wady:

- stwierdzono brak wad tego podejścia :))

Keep **I**t **S**imple **S**tupid