



Pico

---

# Wstęp do kontenerów IoC

Michal.Malecki@man.poznan.pl



# Plan prezentacji

---

- Wzorzec Inversion of Control (IoC)
- Wyszukiwanie zależności (*Dependency Injection*)
- PicoContainer
- Case Study
- Podsumowanie

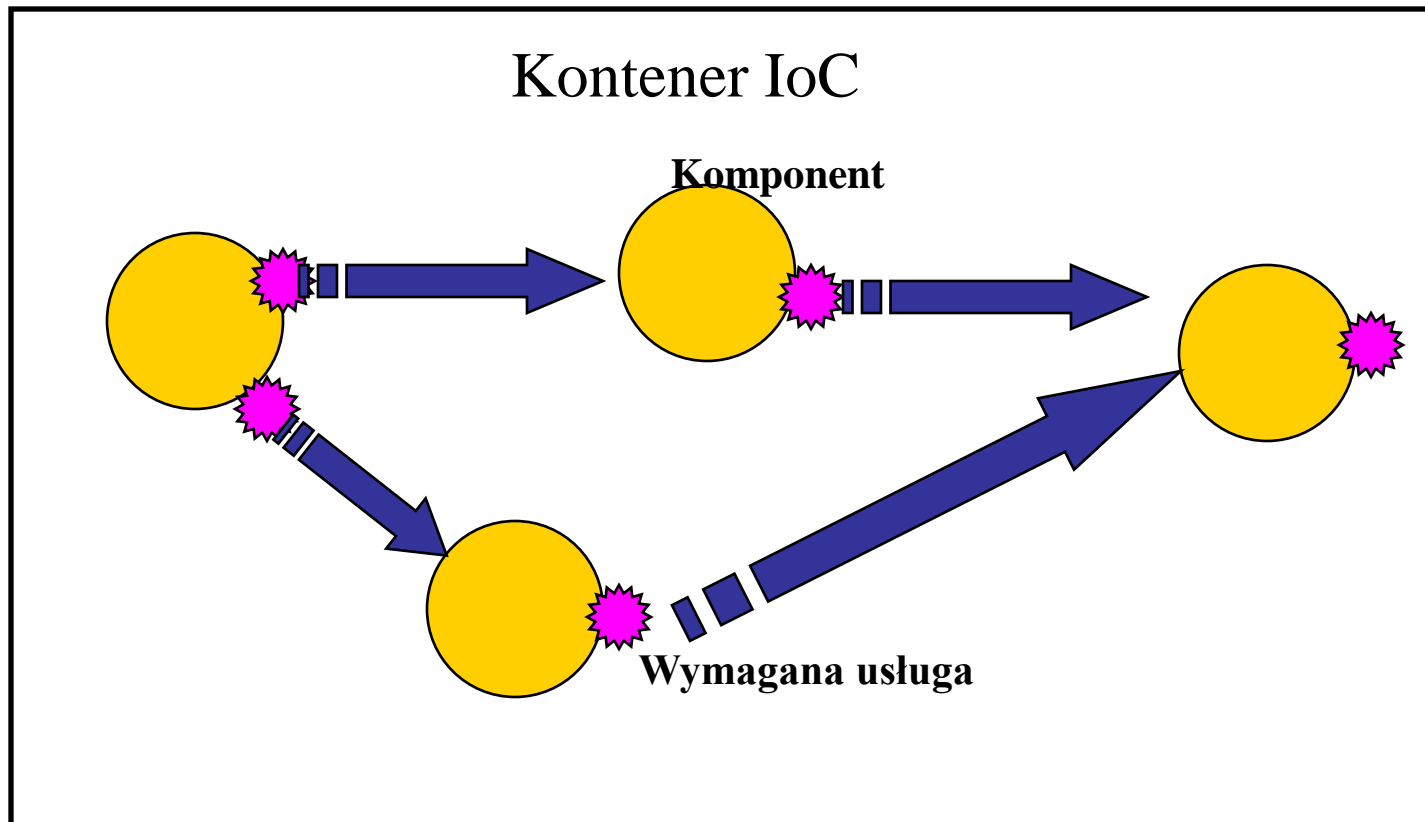


# Inversion of Control

---

- Wzorzec projektowy określający architekturę kontenerów komponentów
- Komponenty to klasy, zawierające konkretną funkcjonalność
- Kontener tworzy instancje komponentów, zapewnia im niezbędne usługi, zarządza ich cyklem życia
- Jakarta Avalon 1998r. Stefano Mazzocchi

# Inversion of Control





# Zalety IoC

---

- Automatyczne zarządzanie cyklem życia komponentów
- Automatyczne rozwiązywanie zależności pomiędzy komponentami
- Bezbolesne testowanie komponentów
- Powiązanie interfejsów, a nie konkretnych implementacji



# Wyszukiwanie zależności

---

- Komponenty wymagają usług, oferowanych przez inne komponenty
- Rodzaje wyszukiwania
  - **Dependency lookup**
  - **Constructor Injection**
  - **Setter injection**



# Dependency lookup

---

- Jakarta Avalon
- Każdy komponent musi implementować interfejs, którego metoda jest wykorzystywana do inicjowania
- Wada – komponent może istnieć jedynie wewnątrz danego kontenera



# Dependency lookup - przykład

---

```
class Komponent implements Service {  
    public void service(ServiceManager sm)  
        throws ServiceException {  
        dependency = (Depend1) sm.lookup("idZależności");  
    }  
}
```



# Constructor Injection

---

- Pico
- Komponenty (klasy) określają swoje zależności za pomocą parametrów konstruktora
- Komponenty spełniają wzorzec „Dobrego Obywatela” (są w stanie spójnym w momencie utworzenia)
- Uniemożliwiają wykorzystanie cyklicznych zależności
- Utrudniony „auto-wire” w porównaniu do setter injection



# Constructor Injection - przykład

---

```
class Komponent {  
    public Komponent(Depend1 d){  
        dependency =d;  
    }  
}
```



# Setter injection

---

- Spring
- Zależności są określone przez metody setter, zgodne z JavaBean
- Pozwala na cykliczność zależności



# Setter injection - przykład

---

```
class Komponent {  
    public Komponent(){  
        //bezparametrowy konstruktor JavaBean  
    }  
    public void setDependency(Depend1 d){  
        dependency = d;  
    }  
}
```



Pico

---



*I was expecting a paradigm shift  
and all I got was a lousy constructor!*



# Pico

---

- „Czysty” kontener IoC (<50kB jar)
- Brak wbudowanych usług
  - rozwinięciem jest Nano, integrujący się m.in. z Hibernate, Struts
- Konfiguracja za pomocą api
  - Nano - Groovy, Javascript, Beanshell oraz XML



# org.picocontainer

---

- Interfejsy i wyjątki – uniezależnienie się od konkretnej implementacji
- Picocontainer – dostęp do zarejestrowanych komponentów
- MutablePicoContainer – zarządzanie komponentami (rejestrwanie, usuwanie)
- ComponentAdapter - obsługa pojedynczej instancji komponentu



# Rejestrowanie komponentów

---

- Każdy komponent jest unikalnie identyfikowany przez identyfikator (obiekt bądź klasę)
- Komponent swoim identyfikatorem  
`pico.registerComponentImplementation(Resource.class);`
- Komponent & identyfikator  
`pico.registerComponentImplementation("UDP", UDPCom.class);`
- Komponent&identyfikator&lista parametrów  
`pico.registerComponentImplementation("messenger", Messenger.class,  
new Parameter[] { new ComponentParameter("UDP"),  
new ConstantParameter("bar") });`



# Rejestrowanie komponentów

---

- Adapter – precyzyjnie określamy sposób tworzenia i obsługi komponentów

```
pico.registerComponent(  
new SetterInjectionComponentAdapter(  
„id”, ThreadWorker.class,  
new Parameter[] { new ComponentParameter("Manager") }));
```

- Obiekt – gdy pico nie zarządza danym komponentem  
`pico.registerComponentInstance(notManagableComponent);`



# Usługi Pico

---

- Najważniejszym elementem Pico, umożliwiającym rozszerzenia są Adaptery
- Nowe implementacje interfejsów PicoContainer i MutablePicoContainer
- Niestety znacznie uboższe w porównaniu do Spring



# Case study

---

Kontener danych  
(element Systemu Zarządzania Danymi) –  
cztery rodzaje, wiele konfiguracji

- Elementy:
  - Komunikacja sieciowa
    - SOAP
  - Obsługa nośników trwałych
    - XML, Berkeley
  - Kontrola zajętości
    - File.renameTo, fuser
  - Zadania cykliczne (wątki)



# Case study – podejście 1

---

- Jedna klasa główna + 2 wątki
  - Absolutnie nietestowalne
  - Nieczytelne
  - Możliwość dodania nowego rodzaju kontenera przez CnP



# Case study – podejście 2

---

- Rozbicie funkcjonalności na zbiór klas (komponenty bez kontenera)
- Znaczne poprawienie parametrów, rozszerzalności
- Wybór implementacji poszczególnych komponentów jest rozproszony po całej aplikacji
- Nadal problemy z testami jednostkowymi



## Case study – podejście 3

---

- Wdrożenie Pico w klasie głównej
- Konkretnie komponenty są wybierane na podstawie konfiguracji w jednym miejscu (podczas rejestracji komponentów)
- Każdy komponent może być testowany automatycznie



# Podsumowanie

---

- Malutki kontener znacznie poprawiający jakość budowanych aplikacji
- Porty – C#, PHP, Ruby
- Brak standardowych usług, oferowanych przez Spring



# Odnośniki

---

- <http://picocontainer.org/>
- <http://www.springframework.org/>
- <http://jdn.pl>



# Podsumowanie

---

Pytania??