**Wstrzykiwanie zależności:**

Nasz kontener dostarcza trzy sposoby na wstrzyknięcie zależności:

* poprzez konstruktor

-jeśli konstruktor posiada adnotację @Inject to on jest wybierany

-jeżeli żaden z konstruktorów nie ma adnotacji to jest wybierany ten, który posiada najwięcej parametrów

-jeżeli nie ma żadnych konstruktorów, jest wybierany domyślny konstruktor bezparametrowy

* poprzez metody

**public class** Gun **implements** IWeapon {  
  
 **private** GunSilencer **gunSilencer**;  
  
 **public void** hit(String target) {  
 System.***out***.println(**"Pif paf"**);  
 }  
  
 GunSilencer getSilencer(){  
 **return this**.**gunSilencer**;  
 }  
  
 @Inject  
 **private void** setGunSilencer(GunSilencer gunSilencer){  
 **this**.**gunSilencer** = gunSilencer;  
 }  
}  
  
**public class** GunSilencer {  
  
}

* poprzez pola

**public class** Gun **implements** IWeapon {

@Inject  
 **private** GunSilencer **gunSilencer**;  
  
 **public void** hit(String target) {  
 System.***out***.println(**"Pif paf"**);  
 }  
  
 GunSilencer getSilencer(){  
 **return this**.**gunSilencer**;  
 }  
  
}  
  
**public class** GunSilencer {  
   
}

**Zależności nazwane**

Zależności nazwane są jedną z prostszych i najczęściej używanych sposobów formowania zależności. W następującym przykładzie mamy klasy Sword i Katana dziedziczące po interfejsie IWeapon, oraz klasę Warrior, która posiada obiekt typu IWeapon:

**public class** Warrior {  
 **private** IWeapon **weapon**;  
  
 **public** Warrior(@Named(name=**"Katana"**) IWeapon weapon){  
 **this**.**weapon** = weapon;  
 }  
  
 **public void** attack(String target)  
 {  
 **this**.**weapon**.hit(target);  
 }  
  
 **public** IWeapon getWeapon(){  
 **return weapon**;  
 }  
}  
  
**public class** Sword **implements** IWeapon {  
 **public void** hit(String target){  
 System.***out***.println(String.*format*(**"Cut %s in half"**, target));  
 }  
}  
  
**public class** Katana **implements** IWeapon{  
 **public void** hit(String target){  
 System.***out***.println(String.*format*(**"Cut %s in half with Katana"**, target));  
 }  
}

Jak widać, dodaliśmy adnotację w konstruktorze klasy Warrior, która opisuje jakiego obiektu oczekujemy. Jednocześnie, gdy dodajemy zależności, musimy ustalić jaka nazwa odpowiada jakiemu obiektowi:

**public void** load() {  
 addBinding(IWeapon.**class**, Sword.**class**);  
 addBinding(IWeapon.**class**, Katana.**class**).setBindingName(**"Katana"**);  
}

**Definiowanie zasięgu Singleton**

W wielu przypadkach taki sam obiekt typu musi być użyty w trakcie trwania aplikacji. Powszechnym rozwiązaniem tego problemu jest użycie wzorca Singleton. Możemy tego dokonać na dwa sposoby:

* poprzez adnotacje

@Singleton  
**public class** Sword **implements** IWeapon {  
  
 **public void** hit(String target){  
 System.***out***.println(String.*format*(**"Cut %s in half"**, target));  
 }  
}

* poprzez funkcje

**public void** load() {  
 addBinding(IWeapon.**class**, Sword.**class**).setSingleton();  
}

**Wspieranie tworzenia kolekcji**

Nasz kontener pozwala na wstrzykiwanie kolekcji do odpowiednich typów lub interfejsów. Na przykład, kiedy mamy dwie klasy Car i Tire:

**public class** Car {  
 **private** List<Tire> **tires**;  
  
 @Inject  
 **public** Car(@Multiple(number=4)LinkedList<Tire> tires){  
 **this**.**tires** = tires;  
 }  
  
 **public** List<Tire> getTires() {  
 **return tires**;  
 }  
}  
  
**public class** Tire {  
  
}

Widać, że konstruktor klasy Car otrzymuje listę obiektów Tire. Chcemy stworzyć 4 obiekty tego typu.

**Wstrzykiwanie zależności typów prostych**

Kiedy chcemy wstrzykiwać jeden z typów prostych, sytuacja nieco się komplikuje. Na przykład:

**public class** BestDriver **implements** IDriver {  
 @Override  
 **public void** connect(String url) {  
 System.***out***.println(**"Connected to "** + url);  
 }  
}

**public class** DatabaseConnector {  
 **private final** IDriver **driver**;  
 **private** String **url**;  
  
 **public** DatabaseConnector(@Named(name = **"URL"**) String url, IDriver driver){  
 **this**.**url** = url;  
 **this**.**driver** = driver;  
 }  
}

Jak widzimy, dodaliśmy zależność nazwaną do konstruktora. Należy również pamiętać o dodaniu specyficznej zależności:

**public void** load() {  
 addBinding(DatabaseConnector.**class**, DatabaseConnector.**class**);  
 addBinding(IDriver.**class**, BestDriver.**class**);  
 addBinding(String.**class**, String.**class**).setBindingName(**"URL"**).toInstance(**"url://myurl"**);  
}

**Zależności opcjonalne**

Nasz kontener wspiera zależności opcjonalne, wystarczy dodać adnotację @Optional przed metodą lub polem. Jeżeli nie da się wstrzyknąć zależności opcjonalnej, nie zostanie zwrócony żaden błąd.

**public class** Sword **implements** IWeapon {  
 **public** ISwordName **swordName**;  
  
 **public** Sword(){  
 }  
   
 @Inject  
 @Optional  
 **public void** setSwordName(ISwordName name){  
 **this**.**swordName** = name;  
 }  
}