Java

Obiektowość

Piotr Fulmański

WydziaĹ, Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

January 15, 2011

Spis treści

Enkapsulacja

- Czym jest enkapsulacja?
- Jak enkapsulować?
  1. Zmienne instancyjne powinny być dostępne tylko przez odpowiednie metody.
  2. Używać konwencji nazewniczej JavaBeans: `set<someProperty>` oraz `get<someProperty>`.

Enkapsulacja

- Czym jest enkapsulacja?
- Jak enkapsulować?
  1. Zmienne instancyjne powinny być dostępne tylko przez odpowiednie metody.
  2. Używać konwencji nazewniczej JavaBeans: `set<someProperty>` oraz `get<someProperty>`.

Enkapsulacja

- Czym jest enkapsulacja?
- Jak enkapsulować?
  1. Zmienne instancyjne powinny być dostępne tylko przez odpowiednie metody.
  2. Używać konwencji nazewniczej JavaBeans: `set<someProperty>` oraz `get<someProperty>`.

Enkapsulacja

- Czym jest enkapsulacja?
- Jak enkapsulować?
  1. Zmienne instancyjne powinny być dostępne tylko przez odpowiednie metody.
  2. Używać konwencji nazewniczej JavaBeans: `set<someProperty>` oraz `get<someProperty>`.

Dobra praktyka OO

Nawet jeśli podczas tworzenia pewnego kodu wydaje nam się, że pewne dane nie będą wymagały „dodatkowej uwagi", to powinniśmy uwzględnić, że taka potrzeba może jednak kiedyś zaistnieć. W tym momencie nie będzie to duży nakład czasu a dzięki temu być może uda nam się zaoszczędzić go wiele.

reuse

use – (act of using) (of substance, object, machine) wykorzystanie, zastosowanie (of sth czegoś);[a]

[a] Wielki słownik PWN-OXFORD, 2004.

polimorphism

Istotą polimorfizmu (ang. *polymorphism*), tj. wielopostaciowości jest to, że każdą klasę potomną (pochodną, podtyp, podklasa, ang. *subclass*) można traktować jak klasę bazową.

polimorphism

Ćwiczenie na polimorfizm. Obserwujemy jak działa mechanizm wielopostaciowości.

Referencja

Należy mieć na uwadze, że jedynym sposobem na uzyskanie dostępu do obiektu, jest wykorzystanie zmiennej referencyjnej. Należy przy tym pamiętać o:

1. Zmienna referencyjna może być tylko jednego rodaju. O ile rodzaj ten nie może się już nigdy zmienić, to wskazywane obiekty tak.
2. Zmienna referencyjna, jak każda zmienna, może się zmieniać, czyli może wskazywać na inne obiekty.
3. Typ zmiennej referencyjnej determinuje jakie metody wskazywanego obiektu można wywołać.
4. Zmienna referencyjna może wskazywać dowolny obiekt typu zgodnego z typem zmiennej referencyjnej lub dowolny obiekt będący instancją klasy pochodnej (podtypu).
5. Zmienna referencyjna może wskazywać na obiekty danej klasy (declared as a class type) lub obiety klas implementujących dany interfejs (declared as an interface type).

Referencja

Należy mieć na uwadze, że jedynym sposobem na uzyskanie dostępu do obiektu, jest wykorzystanie zmiennej referencyjnej. Należy przy tym pamiętać o:

1. Zmienna referencyjna może być tylko jednego rodaju. O ile rodzaj ten nie może się już nigdy zmienić, to wskazywane obiekty tak.
2. Zmienna referencyjna, jak każda zmienna, może się zmieniać, czyli może wskazywać na inne obiekty.
3. Typ zmiennej referencyjnej determinuje jakie metody wskazywanego obiektu można wywołać.
4. Zmienna referencyjna może wskazywać dowolny obiekt typu zgodnego z typem zmiennej referencyjnej lub dowolny obiekt będący instancją klasy pochodnej (podtypu).
5. Zmienna referencyjna może wskazywać na obiekty danej klasy (declared as a class type) lub obiety klas implementujących dany interfejs (declared as an interface type).

Referencja

Należy mieć na uwadze, że jedynym sposobem na uzyskanie dostępu do obiektu, jest wykorzystanie zmiennej referencyjnej. Należy przy tym pamiętać o:

1. Zmienna referencyjna może być tylko jednego rodaju. O ile rodzaj ten nie może się już nigdy zmienić, to wskazywane obiekty tak.
2. Zmienna referencyjna, jak każda zmienna, może się zmieniać, czyli może wskazywać na inne obiekty.
3. Typ zmiennej referencyjnej determinuje jakie metody wskazywanego obiektu można wywołać.
4. Zmienna referencyjna może wskazywać dowolny obiekt typu zgodnego z typem zmiennej referencyjnej lub dowolny obiekt będący instancją klasy pochodnej (podtypu).
5. Zmienna referencyjna może wskazywać na obiekty danej klasy (declared as a class type) lub obiety klas implementujących dany interfejs (declared as an interface type).

Referencja

Należy mieć na uwadze, że jedynym sposobem na uzyskanie dostępu do obiektu, jest wykorzystanie zmiennej referencyjnej. Należy przy tym pamiętać o:

1. Zmienna referencyjna może być tylko jednego rodaju. O ile rodzaj ten nie może się już nigdy zmienić, to wskazywane obiekty tak.
2. Zmienna referencyjna, jak każda zmienna, może się zmieniać, czyli może wskazywać na inne obiekty.
3. Typ zmiennej referencyjnej determinuje jakie metody wskazywanego obiektu można wywołać.
4. Zmienna referencyjna może wskazywać dowolny obiekt typu zgodnego z typem zmiennej referencyjnej lub dowolny obiekt będący instancją klasy pochodnej (podtypu).
5. Zmienna referencyjna może wskazywać na obiekty danej klasy (declared as a class type) lub obiety klas implementujących dany interfejs (declared as an interface type).

Referencja

Należy mieć na uwadze, że jedynym sposobem na uzyskanie dostępu do obiektu, jest wykorzystanie zmiennej referencyjnej. Należy przy tym pamiętać o:

1. Zmienna referencyjna może być tylko jednego rodaju. O ile rodzaj ten nie może się już nigdy zmienić, to wskazywane obiekty tak.
2. Zmienna referencyjna, jak każda zmienna, może się zmieniać, czyli może wskazywać na inne obiekty.
3. Typ zmiennej referencyjnej determinuje jakie metody wskazywanego obiektu można wywołać.
4. Zmienna referencyjna może wskazywać dowolny obiekt typu zgodnego z typem zmiennej referencyjnej lub dowolny obiekt będący instancją klasy pochodnej (podtypu).
5. Zmienna referencyjna może wskazywać na obiekty danej klasy (declared as a class type) lub obiety klas implementujących dany interfejs (declared as an interface type).

Referencja

Należy mieć na uwadze, że jedynym sposobem na uzyskanie dostępu do obiektu, jest wykorzystanie zmiennej referencyjnej. Należy przy tym pamiętać o:

1. Zmienna referencyjna może być tylko jednego rodaju. O ile rodzaj ten nie może się już nigdy zmienić, to wskazywane obiekty tak.
2. Zmienna referencyjna, jak każda zmienna, może się zmieniać, czyli może wskazywać na inne obiekty.
3. Typ zmiennej referencyjnej determinuje jakie metody wskazywanego obiektu można wywołać.
4. Zmienna referencyjna może wskazywać dowolny obiekt typu zgodnego z typem zmiennej referencyjnej lub dowolny obiekt będący instancją klasy pochodnej (podtypu).
5. Zmienna referencyjna może wskazywać na obiekty danej klasy (declared as a class type) lub obiety klas implementujących dany interfejs (declared as an interface type).

polimorphism

Ćwiczenie na referencje. Sprawdzamy zależność dostępnych metod od rodzaju referencji.

Zależność typu IS-A

Idea zależności typu IS-A opiera się o mechanizm dziedziczenia i/lub implementacji interfejsów. Z zależnością typu IS-A mamy do czynienie wówczas, gdy możemy powiedzieć, że obiekty pewnej klasy mogą pełnić rolę obiektów innej klasy, np. Nissan IS-A samochód, który z kolei IS-A pojazd.
Każdy obiekt Javy, który przechodzi więcej niż jeden test IS-A może być uważany za polimorficzny.

Zależność typu HAS-A

Z zależnością typu HAS-A mamy do czynienie wówczas, gdy w klasie A występuje referencja do obiektu klasy B (powiemy wtedy, że A HAS-A B). Kontrolowanie zależności HAS-A pozwala lepiej kontrolować jedną z ważniejszych zasad (dobrych praktyk) programowania zorientowanego obiektowo, tj. unikania tworzenia olbrzymich monolitycznych klas. Przyjmuje się, że im bardziej specjalizowana jest jakaś klasa, tym większe prawdopodobieństwo, że w przyszłości użyjemy jej ponownie.

Overriding / Overloading

Overriding

Method overriding – cecha języka pozwalająca w klasach potomnych redefiniować (nadpisać, zmienić) metodę zaimplementowaną już w klasach rodzicielskich.

Overloading

Constructor and method overloading – sytuacja gdy występuje wiele funkcji o takiej samej nazwie, ale różnych argumentach.

Overriding

Reguły overridingu

- Identyczna lista argumentów.
- Typ zwracany musi być identyczny, lub musi być typem pochodnym zwracanego pierwotnie typu.
- Poziom dostępu nie może być bardziej restrykcyjny; może za to być mniej restrykcyjny.
- Nie można override metody oznaczonej jako `final` lub `static`.
- Jeśli metoda nie może być odziedziczona (czyli np. gdy jest oznaczona jako `private`), nie może być także override.

Overriding

Reguły overridingu

- Identyczna lista argumentów.
- Typ zwracany musi być identyczny, lub musi być typem pochodnym zwracanego pierwotnie typu.
- Poziom dostępu nie może być bardziej restrykcyjny; może za to być mniej restrykcyjny.
- Nie można override metody oznaczonej jako `final` lub `static`.
- Jeśli metoda nie może być odziedziczona (czyli np. gdy jest oznaczona jako `private`), nie może być także override.

Overriding

Reguły overridingu

- Identyczna lista argumentów.
- Typ zwracany musi być identyczny, lub musi być typem pochodnym zwracanego pierwotnie typu.
- Poziom dostępu nie może być bardziej restrykcyjny; może za to być mniej restrykcyjny.
- Nie można override metody oznaczonej jako `final` lub `static`.
- Jeśli metoda nie może być odziedziczona (czyli np. gdy jest oznaczona jako `private`), nie może być także override.

Overriding

Reguły overridingu

- Identyczna lista argumentów.
- Typ zwracany musi być identyczny, lub musi być typem pochodnym zwracanego pierwotnie typu.
- Poziom dostępu nie może być bardziej restrykcyjny; może za to być mniej restrykcyjny.
- Nie można override metody oznaczonej jako `final` lub `static`.
- Jeśli metoda nie może być odziedziczona (czyli np. gdy jest oznaczona jako `private`), nie może być także override.

Overriding

Reguły overridingu

- Identyczna lista argumentów.
- Typ zwracany musi być identyczny, lub musi być typem pochodnym zwracanego pierwotnie typu.
- Poziom dostępu nie może być bardziej restrykcyjny; może za to być mniej restrykcyjny.
- Nie można override metody oznaczonej jako `final` lub `static`.
- Jeśli metoda nie może być odziedziczona (czyli np. gdy jest oznaczona jako `private`), nie może być także override.

Overriding

Reguły overridingu

- Identyczna lista argumentów.
- Typ zwracany musi być identyczny, lub musi być typem pochodnym zwracanego pierwotnie typu.
- Poziom dostępu nie może być bardziej restrykcyjny; może za to być mniej restrykcyjny.
- Nie można override metody oznaczonej jako `final` lub `static`.
- Jeśli metoda nie może być odziedziczona (czyli np. gdy jest oznaczona jako `private`), nie może być także override.

Overloading

Reguły overloadingu

- Overloaded methods musi zmieniać listę argumentów.
- Overloaded methods może zmienić zwracany typ.
- Overloaded methods może zmienić modyfikator dostępu.
- Overloaded methods może zadeklarować nowy wyjątek
- Z metodą typu overload możemy mieć do czynienia w przypadku klasy lub podklasy.

Overloading

Reguły overloadingu

- Overloaded methods musi zmieniać listę argumentów.
- Overloaded methods może zmienić zwracany typ.
- Overloaded methods może zmienić modyfikator dostępu.
- Overloaded methods może zadeklarować nowy wyjątek
- Z metodą typu overload możemy mieć do czynienia w przypadku klasy lub podklasy.

Overloading

Reguły overloadingu

- Overloaded methods musi zmieniać listę argumentów.
- Overloaded methods może zmienić zwracany typ.
- Overloaded methods może zmienić modyfikator dostępu.
- Overloaded methods może zadeklarować nowy wyjątek
- Z metodą typu overload możemy mieć do czynienia w przypadku klasy lub podklasy.

Overloading

Reguły overloadingu

- Overloaded methods musi zmieniać listę argumentów.
- Overloaded methods może zmienić zwracany typ.
- Overloaded methods może zmienić modyfikator dostępu.
- Overloaded methods może zadeklarować nowy wyjątek
- Z metodą typu overload możemy mieć do czynienia w przypadku klasy lub podklasy.

Overloading

Reguły overloadingu

- Overloaded methods musi zmieniać listę argumentów.
- Overloaded methods może zmienić zwracany typ.
- Overloaded methods może zmienić modyfikator dostępu.
- Overloaded methods może zadeklarować nowy wyjątek
- Z metodą typu overload możemy mieć do czynienia w przypadku klasy lub podklasy.

Overloading

Reguły overloadingu

- Overloaded methods musi zmieniać listę argumentów.
- Overloaded methods może zmienić zwracany typ.
- Overloaded methods może zmienić modyfikator dostępu.
- Overloaded methods może zadeklarować nowy wyjątek
- Z metodą typu overload możemy mieć do czynienia w przypadku klasy lub podklasy.