Programowanie obiektowe

Wykład 14

Marcin Młotkowski

1 czerwca 2017

Plan wykładu

Krótka historia

- Sposób projektowania i implementacji oprogramowania zaproponowany przez Bertranda Meyera w latach 80
- Wspierany przez język Eiffel

Design by contract

Idea przewodnia

Kontrakt między klientem (miejsce wywołania procedury) i dostawcą (procedurą)

Realizacja kontraktu

Wymagania (obligations)

klient dostarcza dane spełniające określone warunki

Zapewnienia (benefits)

klient otrzymuje wyniki spełniające określone wymagania

Przestrzeganie kontraktu

Asercje

warunki (formuły logiczne) jakie powinne być spełnione

Przestrzeganie kontraktu

Asercje

warunki (formuły logiczne) jakie powinne być spełnione

Rodzaje asercji

- Warunki wstępne (preconditions)
- Warunki końcowe (postconditions)
- Niezmienniki klasy (invariants)

Przykład

```
put_child(new: NODE) is
    -- Dodanie nowego podwęzła
require
    new /= Void
do
    ...
ensure
    new.Parent = Current
    child_count = old child_count + 1
end
```

Niezmienniki klasy

Niezmienniki klasy

Warunki, jakie powinny spełniać stany obiektów

Niezmienniki klasy

Niezmienniki klasy

Warunki, jakie powinny spełniać stany obiektów

Implementacja

W praktyce warunek ten jest dodawany do warunków wstępnych i końcowych metod

Niezmiennik klasy — przykład

```
class BINARY_TREE[T] feature
    ...
invariant
    left /= Void implies (left.Parent = Current)
    right /= Void implies (right.Parent = Current)
end
```

Jeszcze o warunkach

- Niezmienniki są dziedziczone
- Warunki są sprawdzane dynamicznie
- Reakcja na niespełnienie warunku zależy od sposobu kompilacji

Inne realizacje koncepcji kontraktów

Extended ML

```
val x:int = ?
axiom x>7 andalso isprime x
```

Spec#

```
static void Main(string![] args)
        requires args.Length > 0;
{
    foreach(string arg in  args)
    {
        Console.WriteLine(arg);
    }
}
```

Plan wykładu

Programowanie aspektowe

Obserwacja

Procedury/metody oprócz podstawowego algorytmu często realizują jeszcze zadania poboczne (concerns), np. autoryzacja, logowanie zdarzeń, weryfikacja środowiska etc. Zadania te przeplatają się, np autoryzacja klienta, przyjęcie zlecenia, zapis zlecenia, logowanie transakcji.

Problemy

- Metoda ma dość szeroką funkcjonalność
- Przeplatanie zadań zmniejsza możliwość ponownego użycia
- Realizacja dodatkowego aspektu może być rozproszona po wielu modułach

Próba załatania

Aspekt

jednostka programistyczna realizująca zadania poboczne

Tkanie

dołączanie do wskazanych miejsc kodu zawartego w aspektach

Przykład

```
public class HelloWorld {
    public static void say(String message) {
        System.out.println(message);
    }

    public static void
            sayToPerson(String message, String name) {
    System.out.println(name + ", " + message);
    }
}
```

Przykład cd

```
public aspect MannersAspect {
    pointcut callSayMessage()
            : call(public static void HelloWorld.say*(..));
    before() : callSayMessage() {
        System.out.println("Good day!");
    }
    after() : callSayMessage() {
        System.out.println("Thank you!");
    }
}
```

Historia i realizacje

- AspectJ: rozszerzenie Javy (Georg Kiczales, Parc XEROX)
- Demeter (C++/Java)
- LOOM.NET i WEAVER.NET

Plan wykładu


inżynieria
oprogramowania
oprogramowanie
komponentowe
obiekty
to za mało
io
Cetnarski

Założenia ogólne

- Komponent jest podstawową jednostką oprogramowania
- Komponent ma deklaratywnie opisane interfejsy i różne zależności
- Komponent może być łatwo użyty przez inny komponent
- Użycie może być wyłącznie na podstawie specyfikacji

Cechy komponentów

- Komponenty są niezależne od środowiska
- Komponenty są skompilowane
- Komponenty mogą się między sobą komunikować

Wymagania dla środowisk komponentowych

- Ujednolicony sposób komunikacji między komponentami
- Przenośność komponentów między środowiskami/systemami

Implementacje

- Microsoft COM/DCOM (RPC), OLE
- .NET
- CORBA
- JavaBeans
- Component Pascal
- Wtyczki w przeglądarkach

Plan wykładu

Cechy programowania agentowego

- Niepewne środowisko: dane mogą być błędne lub wogóle nie dotrzeć
- Część zadań jest dublowanych
- Wyniki obliczeń mogą być przybliżone

Agent – podstawowe cechy

- Autonomiczny
- Komunikuje się z innymi agentami
- Reaguje na zmiany środowiska

Przykłady agentów

- Spinacz w MS Word (tzw. Asystent)
- boty

Środowiska wieloagentowe

- Agent Building Environment
- JACK
- JADE
- SemanticsAgent