Meta-programming & Aspect-Oriented Programming

Prof. dr. Kris Luyten

Jo Vermeulen

"Geavanceerde Programmeertechnologie"

Academiejaar 2011-2012

Dagindeling

- Les
- Oefeningen en zelfstudie
- Geen responsie

Meta-programming

Metaprogramming *is the act of writing* ***computer programs*** *(metaprograms) that* ***manipulate*** *other* ***programs*** *(or themselves)* ***as their data****.*

Meta-programming

Ben je al mee in aanraking gekomen …

- Een compiler
- C macro’s
- C++ templates
- …

Meta-programming

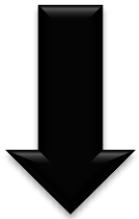
Meta-taal = taal waarin het *metaprogramma* geschreven is (bijv. C)

Object-taal = taal waarin de programma's die gemanipuleerd worden geschreven zijn (bijv. Java)

Meta-programming

Reflectie = erg handig voor meta-programming!

Reflectie laat programma's toe om @ runtime hun eigen structuur en gedrag te observeren …



Meta-taal = Object-taal

Waarom meta-programming gebruiken?

- Complexiteit verbergen
- Flexibiliteit
- Eenvoudig domain-specific languages maken

- *Nieuwe approaches* toelaten bovenop bestaande taal
  - Aspect-oriented (zie later)
  - Prototype-based

Waarom meta-programming gebruiken?

- Complexiteit verbergen
- Flexibiliteit
- Eenvoudig **domain-specific languages** maken

- *Nieuwe approaches* toelaten bovenop bestaande taal
  - Aspect-oriented (zie later)
  - Prototype-based

Domain-specific languages (DSL)

- Programmeertaal **toespitsen** op problemen in een **specifiek domein**
  - leesbaarder
  - eenvoudiger voor high-level programmeurs

- **Externe DSL**: eigen custom taal en syntax (parser nodig!)
  - Bijv. **Makefiles** voor het domein “builden van software”

- **Interne DSL**: geïntegreerd in general-purpose taal via *meta-programming*
  - Bijv. **Ruby on Rails** voor het domein “websiteontwikkeling”

http://www.martinfowler.com/bliki/DomainSpecificLanguage.html

DSL voorbeeld: Ruby on Rails

```ruby
class Post < ActiveRecord::Base
  has_many :comments
end

class Comment < ActiveRecord::Base
  def self.search(query)
    find(:all, :conditions => [ "body = ?", query ])
  end
end

# SELECT * from comments WHERE post_id = 1 AND body = 'hi'
Post.find(1).comments.search "hi"
```

Geen leerstof

http://rubyonrails.org/
http://en.wikipedia.org/wiki/Active_record_pattern

Meta-programming in Python

- Reflectie (natuurlijk)
- Dynamisme: classes on-the-fly aanpassen
- Function decorators
- Metaclasses[1]

Geen leerstof

[1] http://www.webcitation.org/5lubkaJRc

Meta-programming in Python

- Reflectie (natuurlijk)
- Dynamisme: classes on-the-fly aanpassen
- **Function decorators**
- Metaclasses[1]

Geen leerstof

[1] http://www.webcitation.org/5lubkaJRc

Python function decorators

- *A function returning another function, usually applied as a function transformation using the @wrapper syntax.*

Python function decorators

- *A **function returning another function**, usually applied as a function transformation using the @wrapper syntax.*
  - Denk aan Haskell (higher-order functions)

- Typisch voorbeeld:

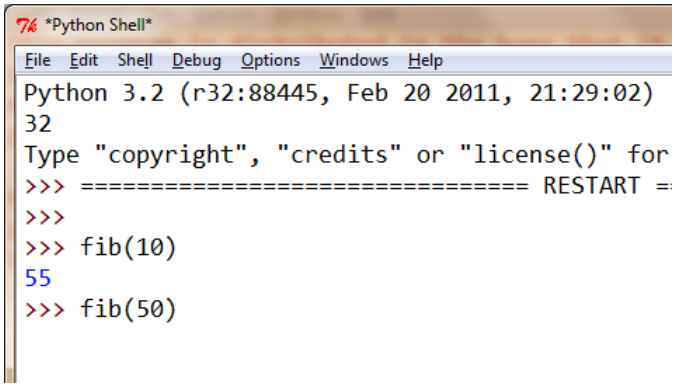
```
class C:
    @classmethod
    def f(cls, arg1, arg2, ...): ...
```

Decorator voorbeeld: memoization

- Fibonacci in Python

```
def fib(n):
    if n in (0, 1):
        return n
    else:
        return fib(n - 1) + fib(n - 2)
```

- Probleem: traag!



Decorator voorbeeld: memoization

- Oplossing: *memoization*
  - Handig voor recursieve functies
  - Resultaat van aanroep met bepaalde invoer cachen

- We kunnen dit doen in de fibonacci functie zelf, maar met een decorator kan memoization ook toegepast worden op andere recursieve functies.

Decorator voorbeeld: memoization

- Decorators krijgen de functie die ze “decoreren” als argument mee (**f**).
- Ze returnen een *wrapper functie* (**helper**) die extra functionaliteit toevoegt aan de bestaande functie, en deze als het ware vervangt.

```
def memoize(f):
    cache = {}
    def helper(x):
        if x not in cache:
            cache[x] = f(x)
        return cache[x]
    return helper
```

Decorator voorbeeld: memoization

- Opdracht: vergelijk de uitvoeringssnelheid van de fib functie met een zonder de @memoize decorator.
- Opdracht: test een andere recursieve functie met de @memoize decorator.
- Opdracht: is de wrapper functie *helper* een closure?

```
@memoize
def fib(n):
    if n in (0, 1):
        return n
    else:
        return fib(n - 1) + fib(n - 2)
```

Decorator: excepties negeren

- Doel: decorator om excepties te negeren en in de plaats default waarde teruggeven.

- Opdracht: lees “More on Function Decorators” en probeer de @swallow decorator te begrijpen
  - http://programmingbits.pythonblogs.com/27_programmingbits/archive/51_more_on_function_decorators.html

```
@swallow(exceptions=ZeroDivisionError, default=0)
def divide(dividend=0, divisor=1):
    return dividend / divisor
```

Opdrachten

- Schrijf een decorator @profile die bij het uitvoeren van een functie uitschrijft hoeveel keer de functie in totaal uitgevoerd is, en hoe lang de uitvoering telkens duurde.
  - Tip: gebruik de functie `time.clock() hiervoor`
  - Gebruik deze decorator om de fib functie mét en zonder @memoize te profilen

    (tip: decorators kunnen genest worden)

- Lees “Charming Python: Decorators make magic easy”
  - http://www.ibm.com/developerworks/linux/library/l-cpdecor/index.html

Aspect-oriented programming

Aspect-oriented programming *probeert de* ***modulariteit*** *van code te* ***verhogen*** *door de core functionaliteit te scheiden van "cross-cutting concerns", oftewel handelingen die door het hele programma uitgevoerd moeten worden (tracing, security, logging).*

Separation of concerns

Frameworks voor AOP

- AspectJ / Java annotations
  - o.a. gebruikt in Spring (voor degenen die GWT zullen volgen)
- AspectC++
- Python decorators kunnen ook gebruikt worden om AOP te realiseren (bijv. @profile)

Terminologie

- **Join point**
  - Bepaald punt in de programmacode
- **Point cuts**
  - Groepeert verschillende join points en waardes op deze join points
- **Advice**
  - Stuk code dat uitgevoerd wordt als een join point bereikt wordt
- **Inter-type declaration**
  - Een inter-type declaration is een declaratie van een variabele die de structuur van een class of de relatie tussen classes beïnvloed
- **Aspect**
  - Eenheid van modulariteit voor cross-cutting concerns

Point cut

- Pointcut *move* groepeert alle plaatsen waar de methods setX en setY van de klasse Point opgeroepen worden:

```
pointcut move() :
        call (void Point.setX(int)) ||
        call (void Point.setY(int))
```

Point cut

- Alle setters met parameter int:

```
pointcut move() :
       call (void Point.set*(int))
```

Point cut

- De setX method op Point instantie p

```
pointcut moveX(Point p) :
        target{p} &&
        call (void setX(int))
```

Advice

- Wordt gekoppeld aan pointcut

- Voor move(), print “about to move”…

```
before(): move()
{
        System.out.println(“about to move”);
}
```

Advice

- Na gooien van exceptie

```
after() throwing(Exception e): move()
{
	System.out.println(e);
}
```

Advice

- around = uitvoeren van advice in plaats van join point
- proceed = toch nog join point aanroepen

```
void around(Point p, int x): moveX(p, x)
{
        if (x < 10)
                proceed(p, x);
}
```

Aspects

- Alles samenstellen

```
aspect checkX
{
	pointcut moveX(Point p, int x):
		target{p} &&
		args{x}
		call(void setX(int));

	void around(Point p, int x): moveX(p, x)
	{
		if ( (x < 100) && (x > 0) )
			proceed(p, x);
	}
}
```

Compilatie en uitvoeren

- AspectJ compiler
  - ajc
- Zet AspectJ libraries in classpath!
  - Bijv. ".:/usr/share/java/aspectjrt.jar"

Leesopdracht

- Neem de eerste twee hoofdstukken van de AspectJ Programming Guide door
  - http://www.eclipse.org/aspectj/doc/released/progguide/index.html
- Installeer de Eclipse AspectJ Development Tools
  - http://www.eclipse.org/ajdt/

Opdracht

- Volg het tracing voorbeeld in de AspectJ programming guide
  - http://www.eclipse.org/aspectj/doc/released/progguide/examples-development.html#tracing-using-aspects
- Lees over abstract aspects:
  - http://www.eclipse.org/aspectj/doc/released/progguide/examples-reusable.html

Opdracht

- Schrijf een abstract aspect om eenvoudig logging te kunnen toepassen op eender welke code (bijv. via log4j).
- Leid hiervan een concreet aspect af, om dit toe te passen op de voorbeeldcode in vieropenrij.tar.gz op de website.