Geavanceerde Programmeertechnologie
Les 5: Gevorderde Programmeertechnieken

Prof. dr. Kris Luyten

Jo Vermeulen

Doel van deze les

- Programmeerparadigma's uitdieping
- Closures
- Inspectie en reflectie

Uit les 1: Programmeertalen

http://people.ku.edu/~nkinners/LangList/Extras/classif.htm

http://oreilly.com/news/graphics/prog_lang_poster.pdf

- Talen: Postscript, Oberon, Tcl/Tk, Fortran, Prolog, Pascal, Delphi, Python, Cobol, Modula, Ada, Rexx, ISO C, C#, Javascript, ANSI/ISO C++, Ruby, Self, Eiffel, PHP, Perl, ML, Lisp, Objective-C, VB, Scheme, Haskell, Caml, Smalltalk, Miranda, PL/1, Simula, Java...
- Belangrijkste klassen: imperatief, functioneel, object-georiënteerd en logisch

Wat kan je nu?

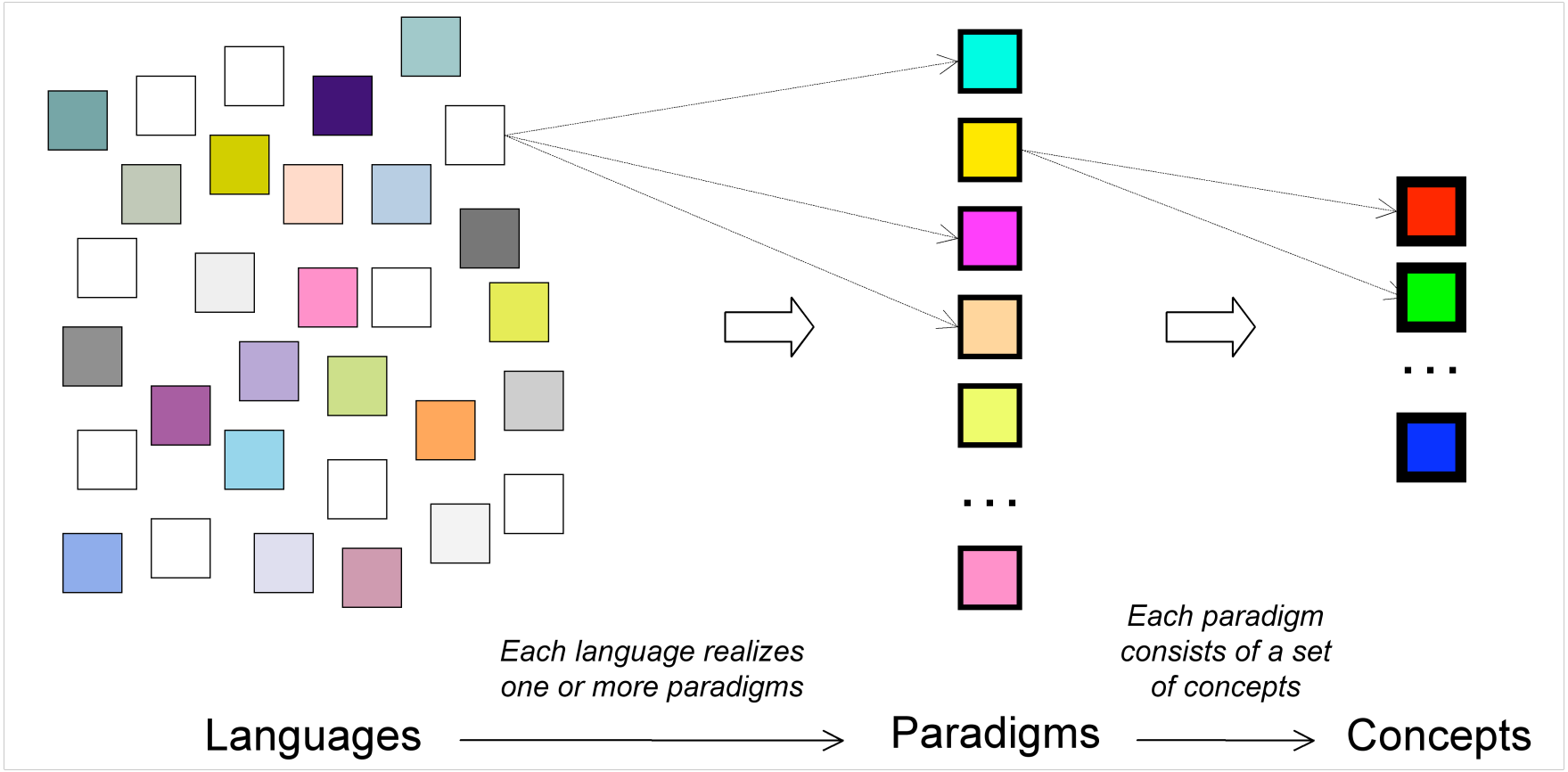
- Imperatief programmeren (C)
- Object-georiënteerd programmeren (C++, Java)
- Functioneel programmeren (Haskell)
- Gebruik van multi-paradigma talen (Python)

Diepgaand overzicht programmeerparadigma's

- *Programming Paradigms for Dummies: What Every Programmer Should Know,* Peter Van Roy, in *New Computational Paradigms for Computer Music*, G. Assayag and A. Gerzso (eds.), IRCAM/Delatour France, 2009*, http://www.info.ucl.ac.be/~pvr/paradigms.html*

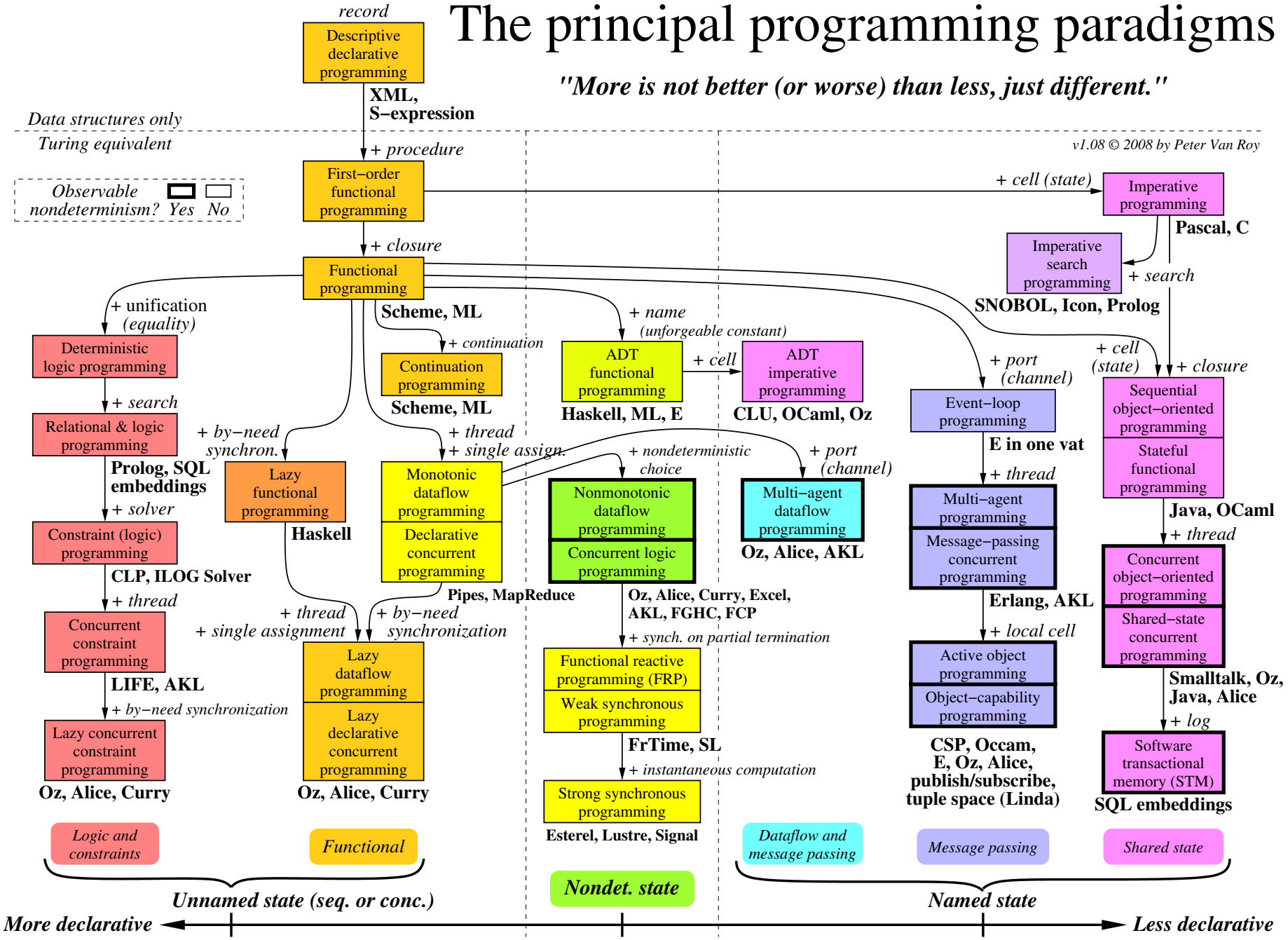
Lezen: secties 1, 2.1 en 4

!!! niet voor dummies !!!


Each language realizes
one or more paradigms
Each paradigm
consists of a set
of concepts
Languages
Paradigms
Concepts

The principal programming paradigms

"More is not better (or worse) than less, just different."



Source: http://www.info.ucl.ac.be/~pvr/paradigms.html

Eigenschap 1: Observable *Non*determinism

Determinism = er is één en slechts één uitvoering van het programma, waarbij de mogelijke program flows volledig door de ontwikkelaar bepaald worden

Eigenschap 1: Observable *Non*determinism

Nondeterminism = tijdens de programma-uitvoering zijn er momenten waarop een systeem (*scheduler)* kiest wat de volgende stap is uit verscheidene mogelijkheden. De keuze kan niet voorspeld worden.

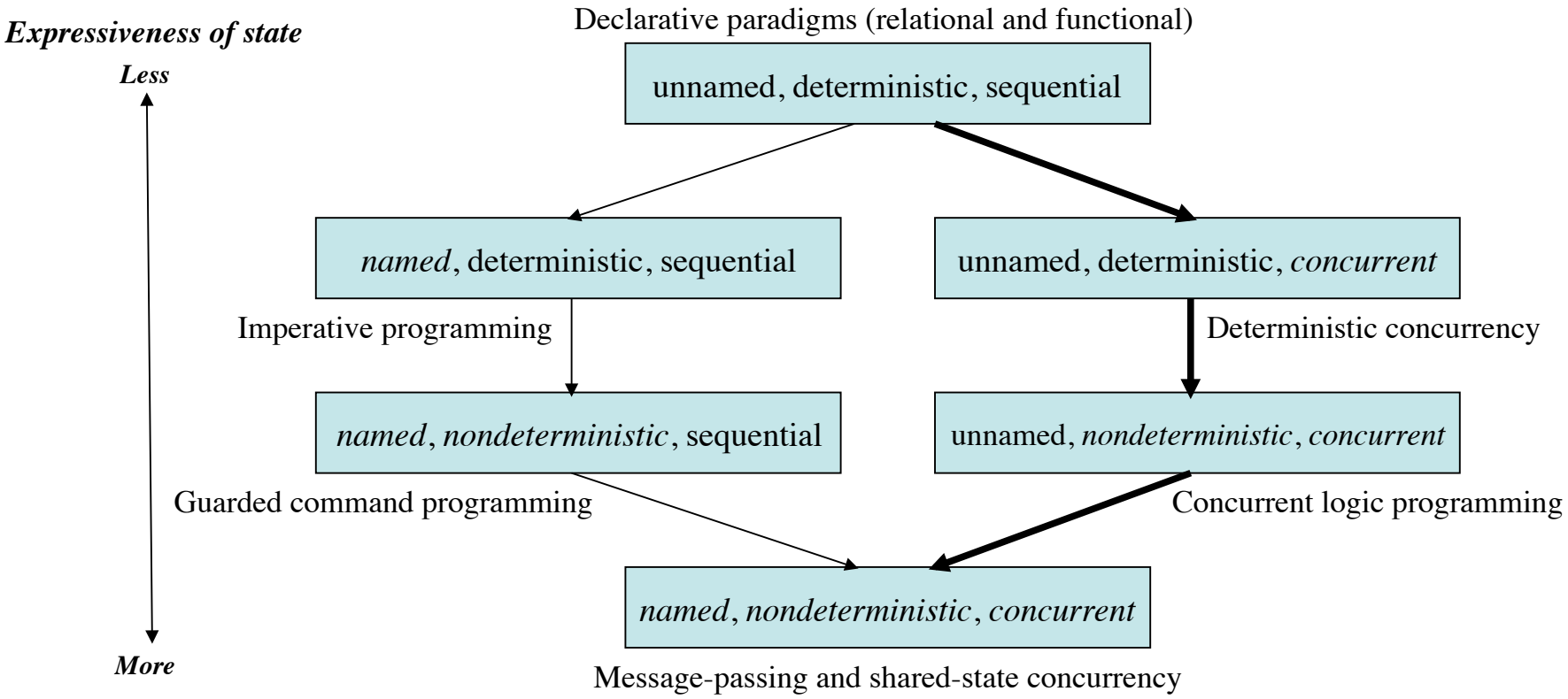
Eigenschap 1: Observable *Non*determinism

Observable Nondeterminism = de gebruiker (ontwikkelaar) kan zien dat, gegeven eenzelfde startconfiguratie, de programmauitvoering een verschillend resultaat kan hebben.

Eigenschap 2: Named State

State = de invulling van alle variabelen op opeenvolgende tijdstippen tijdens de uitvoering van het programma

Eigenschap 2: Named State

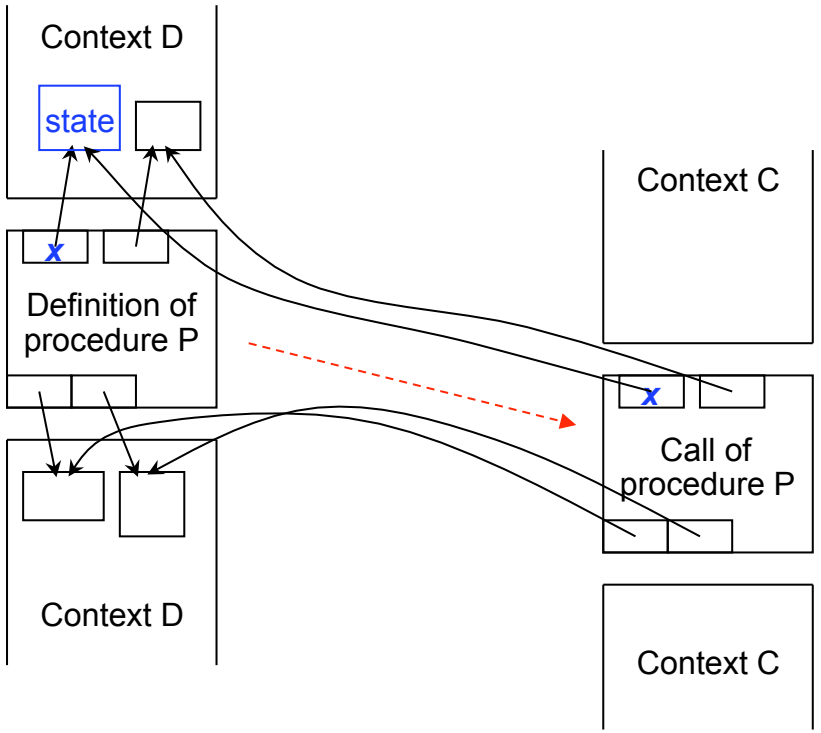


Eigenschap 3: Record

Datastructuur die verschillende data-elementen groepeert en indexeert.

Eigenschap 4: Lexically Scoped Closure

- Een closure is *een procedure* (functie) *samen met referenties buiten de scope van de procedure*



Eigenschap 4: Lexically Scoped Closure

- Functie + “set van” variabelen die in de functie gebruikt worden maar niet in de functie-scope zelf zitten (niet-lokale of vrije variabelen).
  - Variabelen zijn nodig aan de uitvoering
- Vrije variabelen blijven “geldig” voor de uitvoering van de functie.
- Oorspronkelijk voor functionele programmeeertalen (Lisp, ML) maar ook in andere paradigma’s.

Closures praktisch

- Twee toepassingen van closures:
  - Lambda functies (functies zonder naam die “inline” uitgevoerd kunnen worden) en anonieme classes.
  - Functies die functies maken en afleveren.
- Een closure zorgt er voor dat de “lokale omgeving” voor de uitvoering van een functie volledig is.

Closure in Haskell

- Lambda functie

```
> (\x y -> x + y) 3 5
> 8
```

```
> map (\x -> x + 1) lst
```

- Closure: variabele *x* buiten de scope van de functie, maar kan toch gebruikt worden

```
f x = (\y -> x + y)
```

Closure in JavaScript

- Typisch gebruikt voor “generator” functies: functies die andere functies produceren

```
var createFunction = function () {
  var x = 3;
  return function () {
    return 4 + x;
  };
};

var telop = createFunction();
document.write(telop()); // 7
```

Closure in Python

```
def counter():
        x = 0
        def increment():
                nonlocal x
                x += 1
                print(x)
        return increment
```

```
counter1_increment = counter()
counter2_increment = counter()

counter1_increment()    # 1
counter1_increment()    # 2
counter2_increment()    # 1
counter1_increment()    # 3
```

Bron: http://en.wikipedia.org/wiki/Closure_(computer_science)

Closure-like constructies in Java: anonymous classes

```
ActionListener listener = new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      System.out.println("Anonymous");
 }
};

…

myWidget.addActionListener(listener);
```

Closure-like constructies in Java: anonymous classes

```
myWidget.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      System.out.println("Anonymous");
 }
}; )
```

Closure-like constructies in Java: anonymous classes

```
myWidget.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      System.out.println("Anonymous");
 }
}; )
```

Gedeclareerd en geïnstantieerd “ter plekke”

Class definitie zit in de constructor

Closure-like constructies in Java: anonymous classes

```
public class Magic{
        public static void main(String[] args) {
                Trick t =  new Trick(){
                                public void execute(){
                                        System.out.println("sim sala bim");
                                }
                        }
                t.execute();
        }

        public interface Trick{
                void execute();
        }
}
```

Closures opdrachten & materiaal

- Lesmateriaal:
  - http://www.martinfowler.com/bliki/Closure.html
  - *Programming Paradigms for Dummies: What Every Programmer Should Know*, sectie 4
  - Crossing Bordurer: Closures (https://www.ibm.com/developerworks/java/library/j-cb01097/)
- Opdrachten: zoek twee veel voorkomende redenen om closures te gebruiken
  - Tip 1: zoek twee situaties op waarin de code korter en leesbaarder of efficienter wordt door closures te gebruiken.
  - Tip 2: Ruby is een programmeertaal waarin closures nuttig gebruikt kunnen worden…

Eigenschap 5: Independence (concurrency)

- Cfr. Processes en threads in OSes

Inspectie en Reflectie

- Uitvoerbare code die zichzelf kan ondervragen mbt de code structuur
- Dynamische uitvoering van code
- Vooral aanwezig bij talen die op een virtual machine werken
- Via een aparte API kan men objecten en classes “ondervragen”
  - In Java: java.lang.reflect package

Reflectie in Java

- Inspectie van een object:
  - Van welke class is dit object?
  - Van welke classes is dit object afgeleid?
  - Welke methods biedt het aan? Met welke variabelen?
- Dynamische uitvoering:
  - Gegeven de naam van een method als een string, invoke deze
  - Gegeven de naam van een member variabele als een string, verander de waarde van deze variabele

Reflectie in Java

- De Class Class
  http://download.oracle.com/javase/1.5.0/docs/api/java/lang/Class.html
- Alle methods om dynamisch klasses te ondervragen en te manipuleren

Reflectie in Java

```
void printClassName(Object obj) {
      System.out.println("The class of " + obj +
                         " is " + obj.getClass().getName());
   }
```

```
Class c = Class.forName(“java.lang.String”);
```

```
Class c = int.class;
```

Reflectie in Java

```
Class c = Class.forName("java.lang.String");
Method m[] = c.getDeclaredMethods();
for(int i=0; i<m.length; i++)
        System.out.println(m[i].toString());
```

Reflectie in Java

```
import java.lang.reflect.*;
public class constructor2 {
        public constructor2() { }
        public constructor2(int a, int b) {
        System.out.println( "a = " + a + " b = " + b);
        }

public static void main(String args[]) {
        try {
        Class cls = Class.forName("constructor2");
        Class partypes[] = new Class[2];
         partypes[0] = Integer.TYPE;  partypes[1] = Integer.TYPE;
        Constructor ct = cls.getConstructor(partypes);
        Object arglist[] = new Object[2];
        arglist[0] = new Integer(37);
        arglist[1] = new Integer(47);
        Object retobj = ct.newInstance(arglist); }
catch (Throwable e) { System.err.println(e); } } }
```

Reflectie in Java: opdracht

- Schrijf een Java programma waar je via de command line een class name kan ingeven, en die je dan een lijst van methods voor die klasse toont.

```
>java MethodPrinter
Class name? java.lang.Math
Fields:
public static double E
public static double PI
Methods:
public static double abs(double);
Public static float abs(float);
….
```

Reflectie in Java: opdracht

- Schrijf een Java programma dat toelaat via de command line methods van een arbitraire klasse uit te voeren. Dit hoeft enkel te werken voor classes met een constructor zonder parameters, voor static methods of voor static classes.
- Na een vraagteken verwacht je programma input
- Je tracht de input voor de parameters automatisch om te zetten naar een van de ondersteunde types (volgorde van proberen mag je zelf kiezen). Als geen enkel type lukt stop je met een foutmelding.
- Schrijf het resultaat van de uitvoering van de method naar de output
- Schrijf vervolgens ook de waarde van de toString method naar de output van het gebruikte object (indien het geen static class betrof)

Reflectie in Java: opdracht

```
>java MethodExecutor
What method to invoke? java.lang.StringBuffer.append
Needs maximum 3 parameters. How many will you provide? 1
parameter 1? “De kat krrrrabt de krrrrrollen van de trrrrap”
Uitvoering gelukt!
De kat krrrrabt de krrrrrollen van de trrrrap
….
```