Heavy Object-Oriented Programming in C++ 3

edwin.vanouwerkerkmoria@hku.nl

Vorige keer

- Simpele datastructuren: `vector<>`, `array`
- Iterators

    iterators over `vector` met objecten

    iterator over `vector` met pointers: dereferencing

- dereference operator (*)

Opdracht volgende keer

Ga uit van 3 classes: Module, Student, Docent
module: *naam, ec*, Student: *naam*, Docent: *naam*

- maak 3 Modules, 10 studenten, 3 docenten (*op de heap!)*
- wijs docent toe aan module
- wijs studenten toe aan 1-3 modules (verschillend!)
- print een lijst van modules, met docent & modules
- toon totaal EC per student

1) Wijzig EC van 1 module: toon totaal EC per student
2) Verwijder student uit module, & toon lijst opnieuw

Vandaag

- **C++ 11 feature**: `auto`

- Design patterns: **Strategy**

- **C++ Templates**
  de C++ implementatie van 'generic programming'

Nieuw in C++ 11: **auto** type

geen zin om variabele type uit te tikken? **auto!**

```
vector<Student>::iterator i = studenten.begin(); // arg

auto i = studenten.begin(); // hoera!
```

Gotcha's:

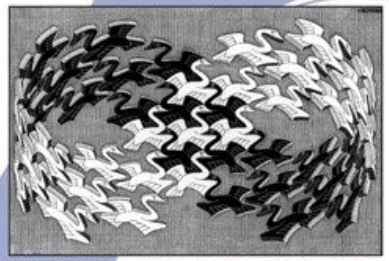
- C++ is een statically-typed language, dus type moet *compile-time* bekend zijn
- ***`auto`** is niet meer een storage class!*

```
auto int pietje = 17; // error in C++ 11
```

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

DESIGN PATTERNS

Explained **simply**

Design patterns

"Elements of reusable object-oriented software"

Bewezen oplossingen voor veel voorkomende programmeer-problemen.

Geen wedstrijd om ze allemaal in je programma toe te passen.

- Creational: *verschillende manieren van maken van objecten*
  `Builder, ObjectPool, Factory`
- Structural: *samenstelling van objecten & classes*
  `Adapter, Facade, Decorator`
- Behavioural: *communicatie tussen objecten*
  `Iterator, Command, Observer`

SOLID principles

Single responsibility

a class as a single responsibility

Open/Closed

a class is open for extension, closed for modification

Liskov substitution

objects should be replacable by instances of their subtypes

Interface segregation

many interfaces are better than one general-purpose interface

Dependency Inversion

depend on abstractions, not concretions

Design pattern: *Strategy*

In plaats van allemaal losse subclasses om verschillend gedrag te implementeren, isoleer je 't gedrag in een *strategy-class*, die je in de bestaande class kan *"inpluggen"*

- Definieer een (abstracte) versie van 't gedrag
- Maak subclasses voor specifieke versies van 't gedrag
- Koppel (runtime of compile-time) gedrag aan gebruiker

Invulling van *Open/Closed* & *Dependency Inversion* principes


1040
U.S. Individual Income Tax
Label
Filing Status
Check only

Opdracht volgende keer

Maak een (tekst-mode) implementatie van Conway's *Game of Life*
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Vul een matrix (25x25) random met een cel. Voor iedere volgende generatie bereken het aantal buren: Cellen met <=2 of >3 buren sterven. Cellen met 2 of 3 buren overleven. Lege cellen met 3 buren krijgen nieuwe cel. Herhaal…

- Maak classes voor de verschillende entiteiten
- Implementeer de standaardregels (hierboven) volgens 't strategy-pattern.
- Maak 2 alternatieve implementaties van deze strategy