A large collection of various objects, including plastic bottles, metal tools, and natural materials, arranged on a surface. The objects are diverse in shape, size, and material, creating a textured and colorful background.

Object-Oriented Programming in C++ (*basic*)

1

edwin.vanouwerkerkmoria@hku.nl

Over deze lessen

Flink veel programmeer werk

Programmeren (redelijk) nieuw? Lynda!

“Learning C++” van Peggy Fisher

Module is 2 EC = 56 uur

3 (!) (werk)colleges

ong. 6 uur/week eigen werktijd

Over deze lessen: opdrachten per keer

Geen aparte eindopdracht

Eindbeoordeling is op basis van de opdrachten die je per sessie krijgt

Lever de uitgewerkte opdrachten vóór de volgende sessie in!

Ruim voor de volgende keer in als je feedback wilt!

We bespreken delen van de oplossingen/problemen gezamenlijk.

Over deze lessen: inhoud

Basiskennis OO en C++

Classes, objects, methods, instance-variables

Basisprincipes object-georiënteerd programmeren:

inheritance, polymorphism, overloading, abstract classes

stack vs. heap, pointers, references, pass-by-value, pass-by-reference

the 'Big Three': constructor, copy-constructor, assignment operator

IO streams, stream operators

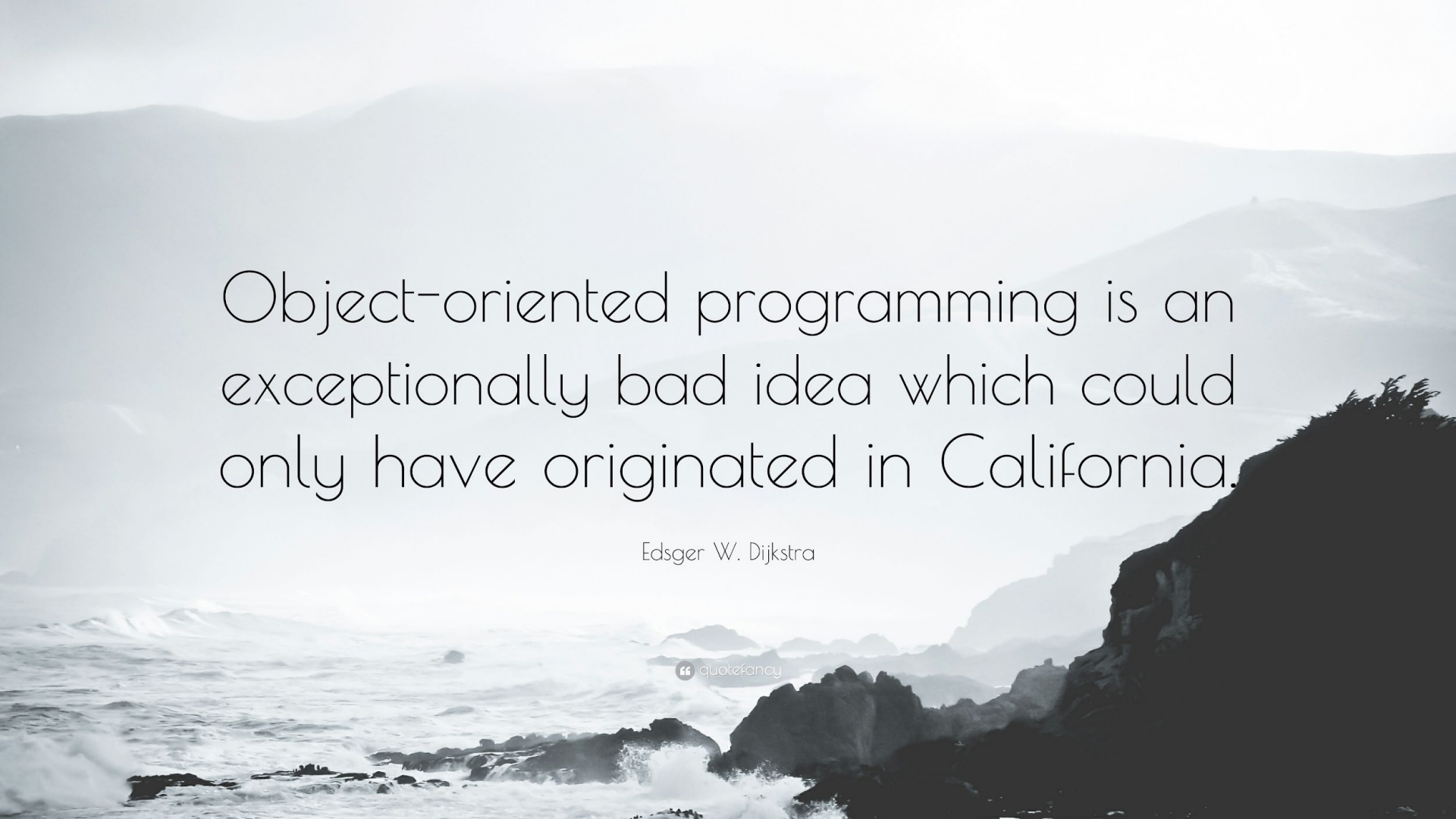
Tools die je nodig gaat hebben

Een C++ ontwikkelomgeving (IDE)

- editor
- compiler / linker

Alle IDEs zijn bruikbaar: Visual Studio (Windows), XCode (OS X), CLion
gebruik een tool die je al een (beetje) kent. CLion is een simpel alternatief

Een versiebeheer-tool: Github Desktop, SourceTree, etc.



Object-oriented programming is an exceptionally bad idea which could only have originated in California.

Edsger W. Dijkstra

“ quote fancy

Object-georiënteerde talen

Modern

Java, C#, Python, PHP, Ruby, Scala, Objective-C, Swift,

Ouder

C++, Smalltalk, Eiffel

Object-Oriented programming?

Anders dan 'gewoon' programmeren: Manier om een (geprogrammeerd) systeem op te bouwen uit objecten.

In plaats van losse 'acties' (subroutines, functies), gecombineerd in een pakketje: een **Object**

In 1 pakketje: **gedrag**, en '**state**' (data)

gedrag: 'methodes' (*geen 'functies'*)

state: 'fields' of 'attributes'

Objecten kunnen **hergebruikt** worden

Definitie van een Object: **Class**

Object-oriented programming

Het is nog steeds 'gewoon' programmeren:

variabelen

```
int totaalAantalPunten = 15;
```

methodes / functies

```
float getInventoryWeight() {  
    return total*weight;  
}
```

```
for (int player=0; player<maxPlayers; player++) {  
    calculateHealth(player);  
}
```

C++ : “C met objecten”

Gebaseerd op oudere programmeertaal C (eind jaren '70)

Veel embedded software in C, want compact, snel, dicht tegen hardware

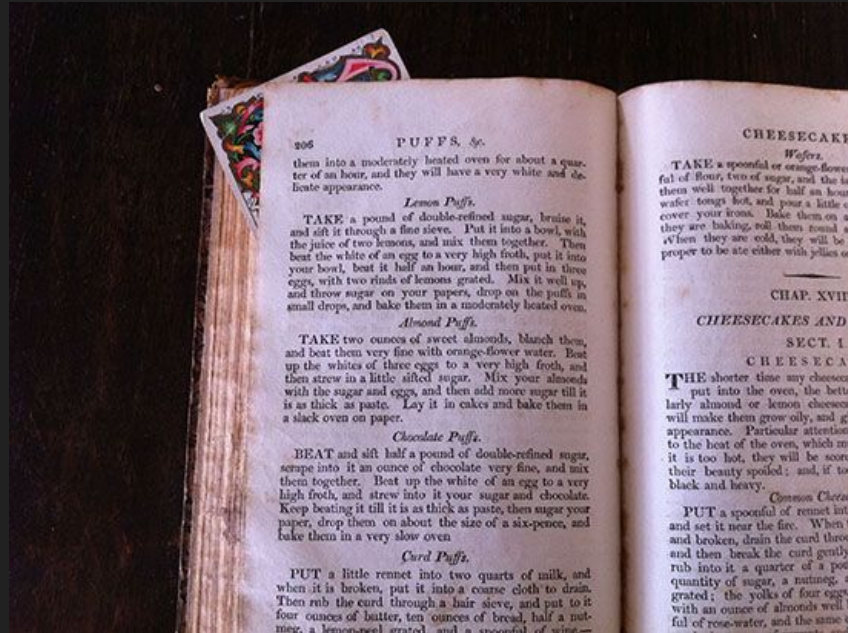
Arduino: aangepaste versie van C & C++

Begin jaren 80: Objecten en Classes toegevoegd aan C

C++ is in concept 'laagje over C'

Je kunt nog steeds prima C & C++ mixen

Object-oriented programming: Class vs Object



Oftewel: de *class* is het concept, het *object* de realisatie...

C++: definitie van een **class**

HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Rectangle { private: int width, height; float transparency; public: Rectangle(int w,int h); void setTransparency(float a); int area(); void draw(); }</pre>	<pre>Rectangle::Rectangle(int w,int h) { width = w; height = h; } void Rectangle::setTransparency (float a){ transparency = a; } int Rectangle::area() { return width*height; } void Rectangle::draw() { ... }</pre>

Op basis van een **class**, maak je een **object**

```
void main() {  
  
    //Rectangle.setTransparency(); // kan niet!  
  
    // met een Class kun je alleen objecten maken  
    Rectangle myRectangle = Rectangle();  
  
    // met een object kun je wél wat doen  
    myRectangle.draw();  
  
}
```

Een class = een type!

```
float eindCijfer = 5;
```

```
int eindCijfer = 5.7;
```

```
string studentNaam = "Pietje Puk";
```

```
int studentNaam = "Pietje Puk";
```

```
Taart mijnTaart = Taart();
```

```
Bankrekening rekening = Taart();
```

Een class = een type!

- ✓ `float eindCijfer = 5;`
- ✓ `int eindCijfer = 5.7;` ⚠ `// wordt afgekapt naar 5`
- ✓ `string studentNaam = "Pietje Puk";`
- ✗ `int studentNaam = "Pietje Puk";`
- ✓ `Taart mijnTaart = Taart();`
- ✗ `Bankrekening rekening = Taart();`

Het startpunt van je programma : `main()`

Ieder programma heeft een beginpunt nodig. Waar moet de computer beginnen met het runnen van je code?

In C/C++ heet dit beginpunt de `main()` functie (*functie! geen methode!*)

De `main()` functie ziet er meestal zo uit:

```
int main(int argc, char* argv[]) {  
    // start van je programma  
}
```

De parameters zijn optioneel, dus `int main()` mag ook!



IOStreams

Streams zijn de C++ abstractie voor dingen waar je naar kunt schrijven, of uit kunt lezen - bv files, het toetsenbord, het scherm...

Standaard voorgedefinieerde streams:

- `std::cout` tekst-uitvoer (de 'console')
- `std::cin` text-invoer (het toetsenbord)

`std::endl` is een speciaal object wat 'einde van de regel' aangeeft, en de stream 'flusht' (alle eventueel gebufferde data naar de stream schrijven).

Streams importeer je met de `#include <iostream>` header

std::cin en std::cout

Tekst op de console zetten:

```
std::cout << "Ooooh, cout!" << std::endl;
```

"<<" is de "insertion" ("put") operator

Tekst vanaf 't toetsenbord uitlezen:

```
char pietjepuk;  
std::cin >> pietjepuk
```

">>" is de "extraction" ("get") operator

Hello



OO-analyse

Bedenk welke objecten (concepten) er in je programma voor gaan komen

goed begin: alle zelfstandige naamwoorden

- identificeer objecten
- bepaal gedrag per object
- bepaal welke data (state) ieder object nodig heeft
- bepaal relaties tussen objecten

Voor 't weergeven van deze informatie is een teken-taal: **UML**

Student
-name:string -studentNr:int
+getName():string

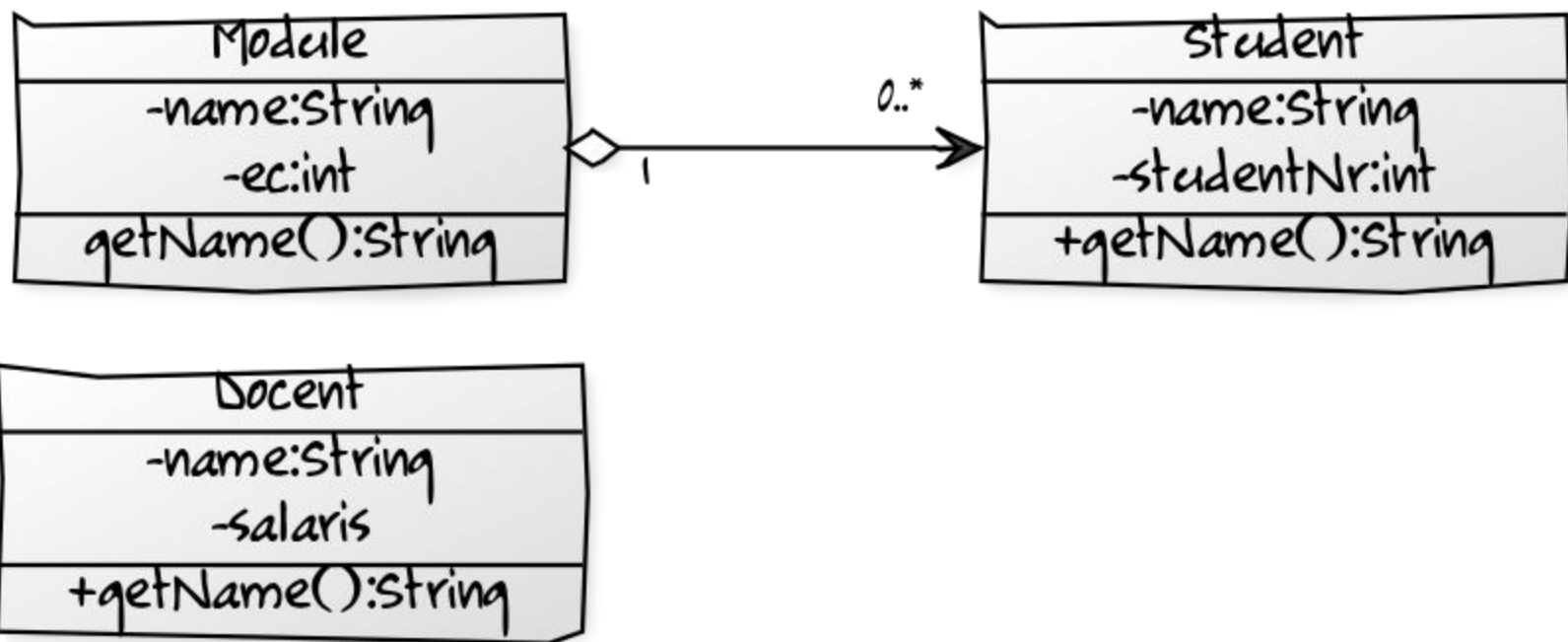
student
-name:String -studentNr:int
+getName():String

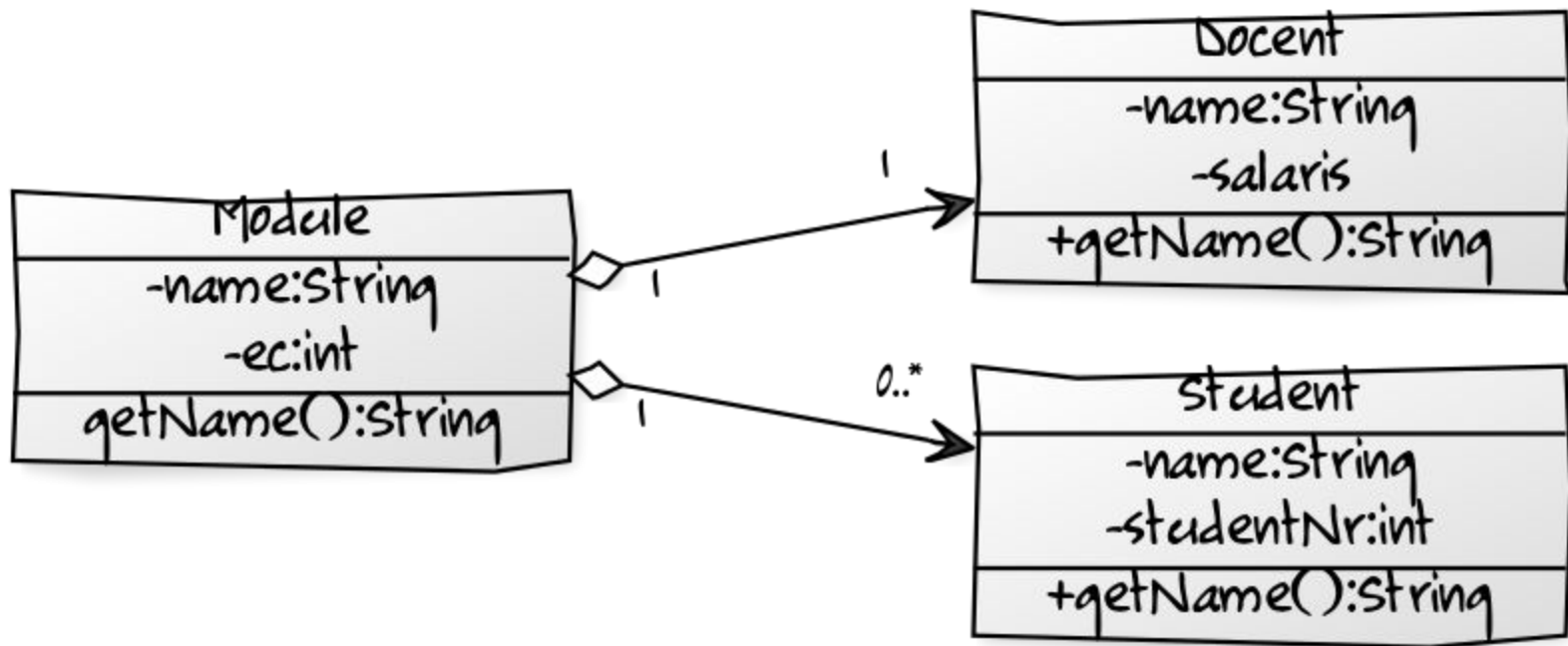
Module
-name:String -ec:int
getName():String

Docent
-name:String -salaris
+getName():String

Student
-name:String -studentNr:int
+getName():String

Module
-name:String -ec:int
getName():String







José Bonifácio

Codingstyle

Code schrijf je in eerste instantie voor jezelf, in tweede voor je collega(s), en pas op de laatste plaats voor je compiler.

- Schrijf *leesbaar*: duidelijke variabele- en methode namen
- Formatteer consistent (je IDE helpt)
- Voeg commentaar toe

Haakjes boeien me niet. Wel een paar basic C++ conventies (*geen C# stijl dus!*)

- Classes beginnen met een **HOOFDLETTER**
- methods beginnen met een **kleine letter**
- variabelen beginnen met een **kleine letter**



Testament

~

OO principes: *inheritance*

HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Rectangle { private: int width, height; float transparency; public: Rectangle(int w,int h); void setTransparency(float a); float area(); void draw(); }</pre>	<pre>Rectangle::Rectangle(int w,int h) { width = w; height = h; } void Rectangle::setTransparency (float a){ transparency = a; } float Rectangle::area() { return width*height; } void Rectangle::draw() { ... }</pre>

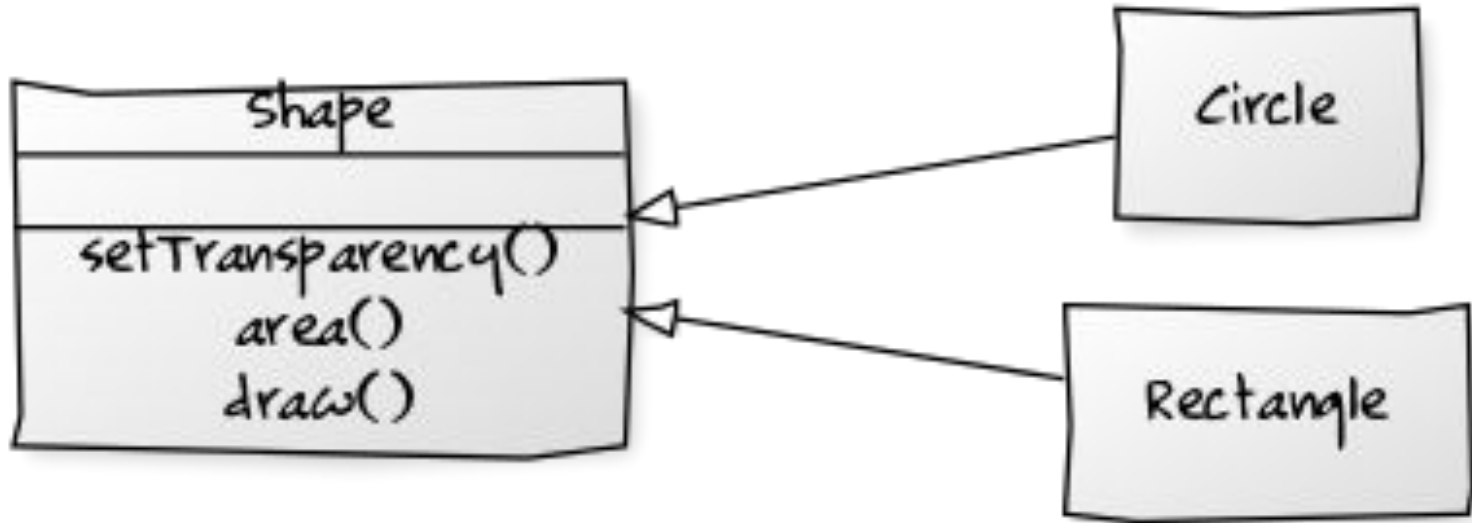
OO principes: *inheritance*

HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Circle { private: int radius; float transparency; public: Circle(int r); void setTransparency(float a); float area(); void draw(); }</pre>	<pre>Circle::Circle(int r) : radius(r) { } void Circle::setTransparency (float a){ transparency = a; } float Circle::area() { return PI*radius*radius; } void Circle::draw() { ofDrawCircle(...); }</pre>

OO principes: *inheritance*

HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Circle { private: int radius; float transparency; public: Circle(int r); void setTransparency(float a); int area(); void draw(); }</pre>	<pre>Circle::Circle(int r) : radius(r) { } void Circle::setTransparency(float a) { transparency = a; } int Circle::area() { return 2*PI*radius; } void Circle::draw() { ofDrawCircle(...); }</pre>

OO principles: *inheritance*



Gemeenschappelijk gedrag kan naar de superclass

Inheritance: de *superclass* bevat wat gedeeld is

HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Shape { public: Shape(); void setTransparency(float a); virtual float area(); virtual void draw(); }</pre>	<pre>Shape::Shape() { } void Shape::setTransparency (float a){ transparency = a; } float Shape::area() { return 0; } void Shape::draw() { }</pre>

OO principes: *Inheritance*

HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Rectangle : public Shape { private: int width, height; public: Rectangle(int w,int h); float area(); void draw(); }</pre>	<pre>Rectangle::Rectangle(int w,int h) { width = w; height = h; } float Rectangle::area() { return width*height; } void Rectangle::draw() { ... }</pre>

Inheritance: gedrag uit *superclass* meenemen

HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Rectangle : public Shape { private: int width, height; public: Rectangle(int w,int h); float area(); void draw(); }</pre>	<pre>Rectangle::Rectangle(int w,int h) { width = w; height = h; } float Rectangle::area() { return width*height; } void Rectangle::draw() { imgLib.setTransparency(transparency); // etc... }</pre>

Overriding: gedrag uit *superclass* vervangen

IMPLEMENTATIE file (.cpp)

```
Circle::Circle(int r) {
    radius = r;
}

float Circle::area() {
    return PI*radius*radius;
}

void Circle::draw() {
    imgLib.setTransparency(transparency);
    imgLib.drawCircle(...)
}
```

IMPLEMENTATIE file (.cpp)

```
Rectangle::Rectangle(int w,int h) {
    width = w;
    height = h;
}

float Rectangle::area() {
    return width*height;
}

void Rectangle::draw() {
    imgLib.setTransparency(transparency);
    imgLib.drawRect(...)
}
```



GitHub

GitHub: versiebeheer op 't web

Versiebeheer: historie van je programma

Meestal ontwikkel je stap-voor-stap: verschillende versies

- nieuwe features

- bugfixes

Delen van je code (vandaar GitHub)

Git / GitHub termen

Repository ('repo')

De database waar je code in zit. 2 stuks: lokaal, en op website.

Commit

Opslaan van je huidige wijzigingen in de locale database

Push

Verstuur je laatste wijzigingen naar een remote repo

Pull

Ophalen van wijzigingen van anderen

Module-repository op GitHub

<https://github.com/edovino/OBOPB>

Uitgewerkte voorbeelden uit de bijeenkomsten

Evt. notities

Opdrachten

Github Desktop

GitHub repository: staat op de GitHub website

Locale code versturen naar de GitHub website: **client**

- command-line: 'git'
- grafische omgeving: **Github Desktop**

`https://desktop.github.com/`

Of gebruik de ingebouwde GIT ondersteuning van je ontwikkelomgeving...

Github: je eigen repository

Code inleveren doe je via GitHub

- maak een repository aan op github.com
maak een github account als je die nog niet hebt

op het *Create New Repository* scherm:

- selecteer 'Public' repository
- selecteer een .gitignore file (*kies C++ uit de dropdown*)

Na aanmaken, kopieer de inhoud van de .gitignore uit de klas repo, en plak die in je .gitignore in je eigen repo.

Mail mij de link naar je repository!



Geheugen in C++

Twee soorten geheugen

- Stack: *automatic allocation*

beperkte ruimte

meestal: variabelen binnen een methode, parameters(!)

- Heap: *dynamic allocation*

max. geheugen in computer

grote, of grote aantallen objecten (videobeelden, particle clouds)

Geheugen in C++: stack

- Stack (*automatic allocation*)

```
void updateEC() {  
  
    Student student;  
  
    student.updateEC();  
  
}
```

Bestaat alleen binnen huidige scope - automatisch opgeruimd!

Geheugen in C++: Heap

pointers: **adres** van een variabele

expliciet aanmaken met **new** operator

```
void calculateTotals() {  
    Student* student = new Student();  
    student->updateEC();  
}
```

Zelf opruimen!

```
delete student;
```

Objecten op de heap: pointers

Een *pointer* is een *verwijzing* naar een object - het adres, niet het object zelf!

Bevat dus ook niet de methodes & attributen van 't object.

Om methodes aan te roepen: eerst van de pointer, het bijbehorende object zoeken: met "->"

```
Student* s = new Student();
```

```
s->getName(); // in plaats van student.getName()
```


Opdracht 1

Modelleer het lesrooster

- welke objecten spelen volgens jou een rol?
- wat zijn de attributen (variabelen) van die objecten?
- wat zijn de relaties tussen die objecten?
- wat voor gedrag moeten die objecten hebben?

Maak C++ classes voor de objecten die je bedacht hebt.

Je hoeft de methodes niet te implementeren - geef ze alleen een naam, en commentaar wat ze zouden moeten doen.

Declareer wel de attributen (variabelen) die je nodig denkt te hebben.

Opdracht 2

Start met een nieuw, leeg project, en maak daar in drie classes: Persoon, Student, en Docent

Geef de class Persoon 2 attributen: leeftijd (int), naam (string), en maak methodes om die op te vragen: getAge(), getName()

Laat Student en Docent overerven van Persoon, en print de twee attributen.
*(hoe ga je de variabelen leeftijd & naam van deze classes van waarde voorzien?
hint: de Rectangle class)*

Voeg in Student en Docent een attribuut toe wat uniek is voor beiden (wat heeft een Docent wel, wat een Student niet heeft, en vice versa), en voeg die toe.