Object-Oriented Programming in C++ *(basic)* 2

edwin.vanouwerkerkmoria@hku.nl

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.

Vandaag

OO principe: polymorphism

- via inheritance & overriding
- via *method overloading*

Input/Output (IO) streams

- character IO streams
  cin, cout, cerr (en clog)
- file-IO streams
  ifstream, ofstream

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

HEADER file (.h)

```
class Rectangle {
  private:
    int width, height;
    float transparency;

  public:
    Rectangle(int w,int h);
    void setTransparency(float a);
    int area();
    void draw();
};
```

IMPLEMENTATIE file (.cpp)

```
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() {
   …
}
```

Constructor

De *constructor* (kort: ctor) zorgt dat het object gemaakt en klaar voor gebruik is. Een class heeft altijd een constructor: je kunt 'm zelf maken, of compiler maakt 'm.

impliciet aanroepen van de constructor:

```
Student student;
```

expliciet:

```
Student student = Student();
```

Een constructor kan ook een parameter hebben:

```
Student student = Student("naam");
```

Constructor van superclass

Als de superclass van je class geen *default* constructor heeft (een constructor *zonder* parameters) heeft, moet je expliciet aangeven welke superclass constructor je wilt gebruiken:

Persoon.cpp	Student.cpp

`Persoon :public Persoon {`
`  public:`
`  Student(int age);`
`}`
	`Student::Student(int age) : Persoon(age) {`
	`  // ...`
	`}`

OO principes: ***polymorphism***

Wil zeggen: zelfde 'vorm', maar verschillend gedrag

Dit kan op verschillende manieren

- via *inheritance*

  in de subclass pas je het gedrag aan

- via *method overloading*

  uit verschillende methodes wordt 1 gekozen op basis van ***parameter-types***

Polymorphism via inheritance & overriding

Inheritance:

- Een manier om gedrag te delen.
  dingen die gemeenschappelijk waren, meekrijgen door te overerven.

- Een manier om gedrag te *specificeren*.
  Algemeen gedrag in de superclass, aanpassen naar (aan laten sluiten op) de specifieke subclass.

Afdwingen dat een methode *overridden* wordt

Polymorphism via *method overloading*

Als je meerdere methoden, met dezelfde naam hebt, maar met verschillende parameters, wordt automatisch de juiste aangeroepen afhankelijk van 't type.

```
public doSomething(Dog d) {
…
}

public doSomething(Cat c) {
…
}
```

Als je parameter van type 'Dog' is, wordt de 1e geselecteerd. Voor 'Cat' de 2e

Object slicing

Wanneer je een object gebruikt van een subclass-type, op een plek waar een superclass gespecificeerd staat, verlies je de specifieke eigenschappen van de subclass: Die veranderingen zijn er als het ware afgesneden

```
void makeNoise(Animal a) {
 …
}
```

Als je hier een variabele van type Dog gebruikt (een subclass van Animal), zie je alleen het 'Animal' deel...

References

Door *references* te gebruiken kun je voorkomen dat C++ een *copie* maakt van parameters die je aan een methode meegeeft.

Een reference is een *verwijzing* naar het originele object

```
void makeNoise(Animal animal); //copie!

void makeNoise(Animal& animal); //origineel!
```

Samenvattend

OO-technieken:

- Overriding
- Overloading

Object Slicing

Object References

Streams

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:

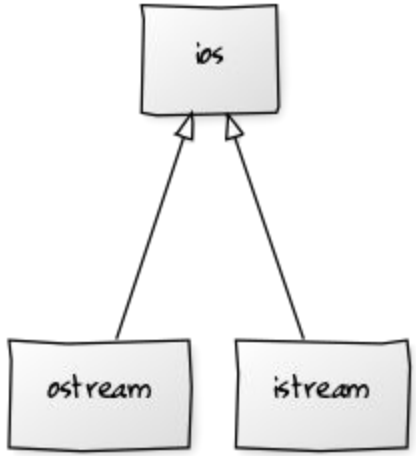
- `std::cout` tekst-uitvoer (de 'console')

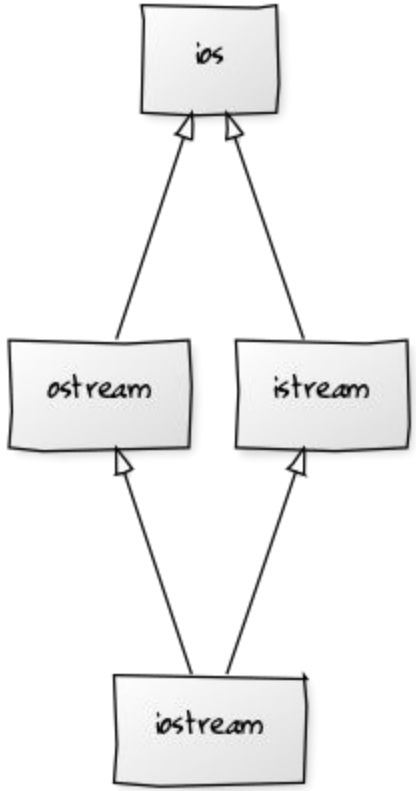
  schrijven naar cout: tekst verschijnt op het scherm

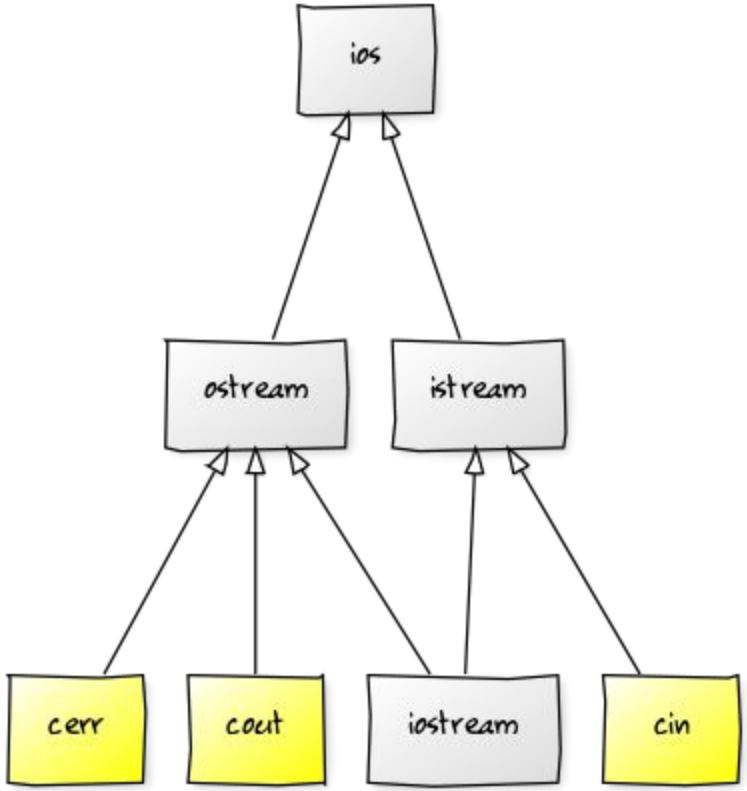
- `std::cin` text-invoer (het toetsenbord)

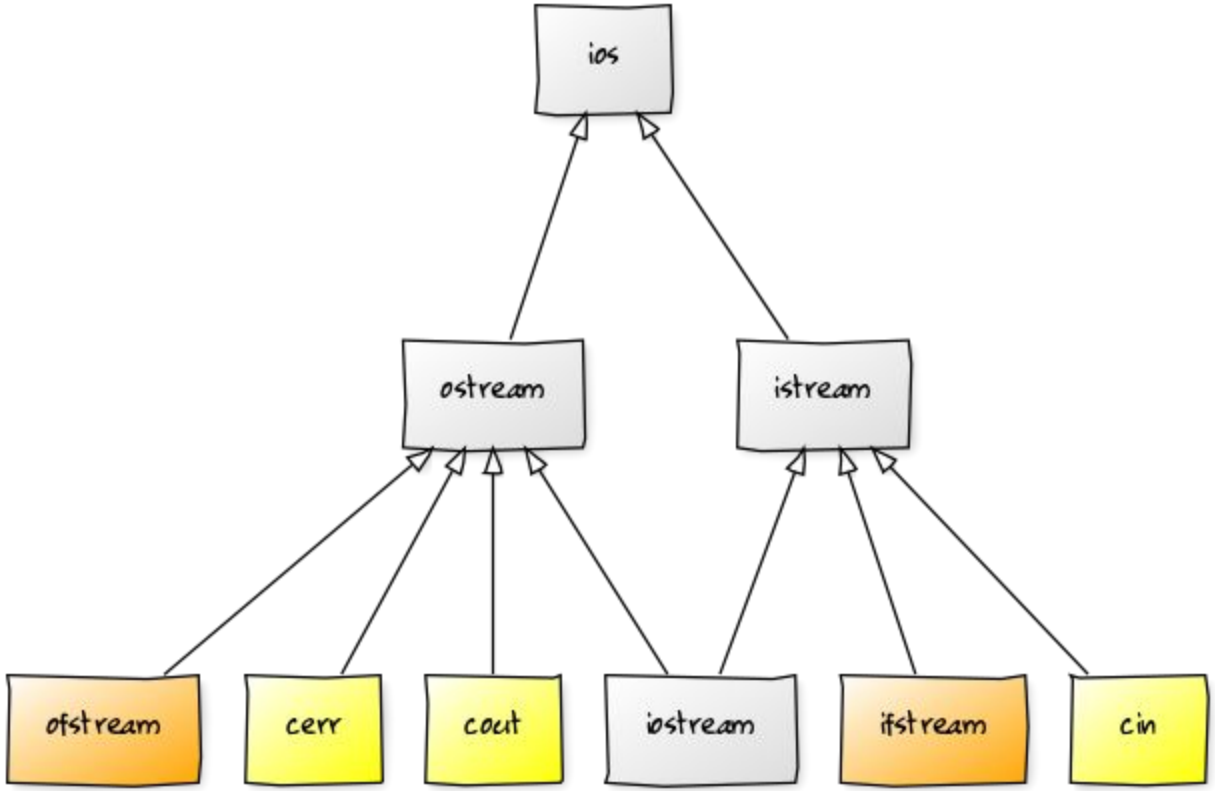
  lezen uit cin: krijgt tekst die op keyboard ingetikt is

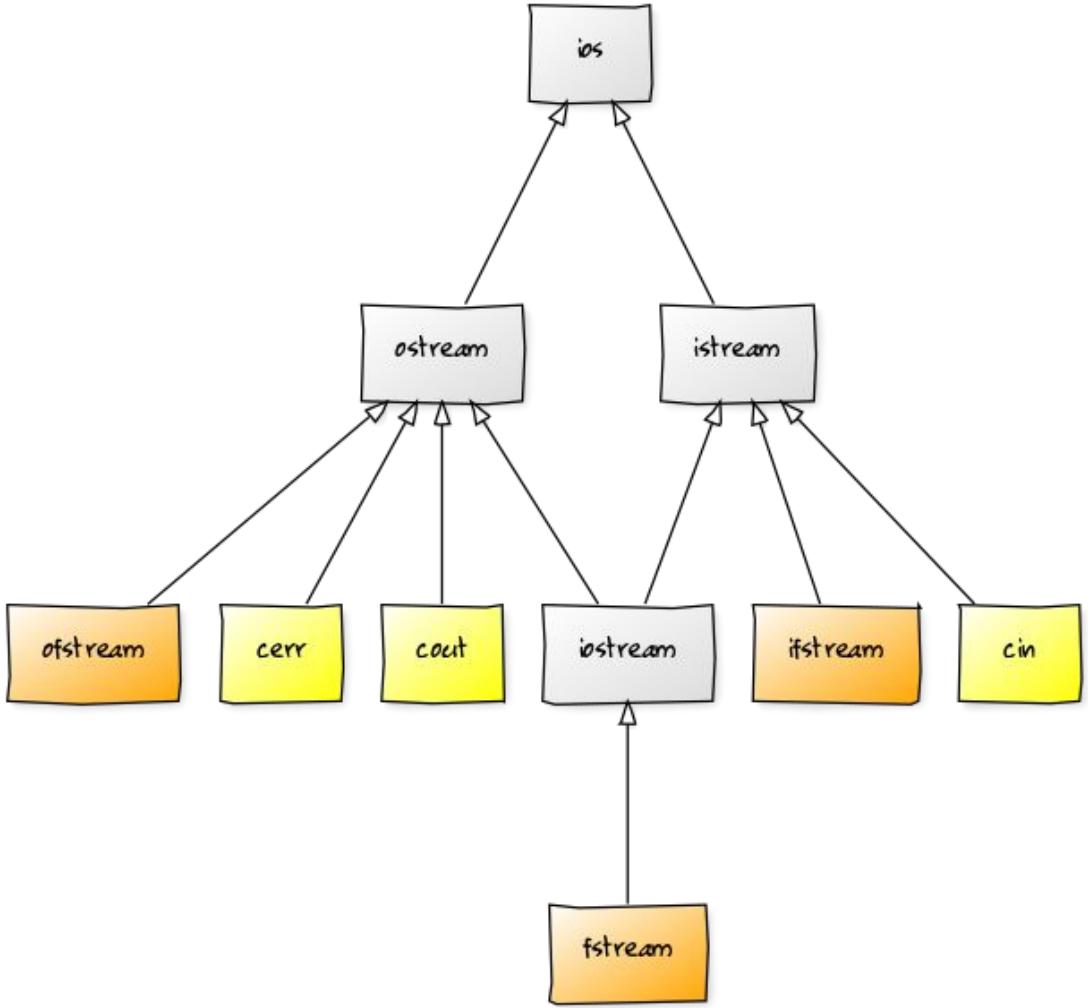

ios


ios
ostream
istream


ios
ostream
istream
iostream


ios
ostream
istream
cerr
cout
iostream
cin


ios
ostream
istream
ofstream
cerr
cout
iostream
ifstream
cin


ios
ostream
istream
ofstream
cerr
cout
iostream
ifstream
cin
fstream

Lezen uit een file: `ifstream`

```
std::ifstream in_stream;

in_stream.open("volledig/pad/naar/het/bestand");

if (!in_stream) {
    std::cout << "Probleem bij openen file" << std::endl;
    exit(1);
}

char ch;
in_stream.get(ch);
```

Schrijven naar een file: `ofstream`

```
std::ofstream out_stream;

out_stream.open("volledig/pad/naar/het/bestand");

if (!out_stream) {
    std::cout << "Probleem bij openen file" << std::endl;
    exit(1);
}

char ch = 'a';
out_stream.put(ch);
```

stream operators

>> ***extraction*** **operator ('get' operator)**

hetzelfde als istream::get()

<< ***insertion*** **operator ('put' operator)**

hetzelfde als ostream::put()

stream manipulators

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

Andere manipulators:

`std::noskipws` : sla 'whitespace' (spaties, tabs, newline) NIET over bij lezen

`std::dec`,
`std::hex` : toon getallen als decimaal, of hexadecimaal

vaak hebben 'manipulators' iets van doen met formattering

Opdracht 1 volgende keer

Maak een serie classes die muntgeld vertegenwoordigen: een baseclass, drie subclasses voor een 50-cent muntstuk, een 1-euro muntstuk, en een class voor een speciaal 'koffiemuntje'. De classes hebben allemaal hun eigen waarde, en een naam.

Maak een class die een koffieautomaat gaat beschrijven. De class heeft methodes nodig om de muntjes te accepteren, en kan 3 verschillende soorten koffie leveren, tegen 3 verschillende prijzen. Met het speciale 'koffiemuntje' kun je alle koffie krijgen.

Als je een muntje in de automaat gooit, beeldt de automaat af wat voor munt 't was, en welke soorten koffie je kunt kopen.

Opdracht 2 volgende keer

1) Schrijf een programma dat de text in een file achterstevoren in een ander file wegschrijft, zonder gebruik te maken van een array of vector.
   Hint: tel eerst het aantal tekens in 't bestand.

2) Schrijf een programma dat de text in twee files combineert, door om-en-om een regel uit 't ene, en uit 't andere file in het outputfile te schrijven