

# VDAB cursus 764701

Object georiënteerde ( OO )principes  
en programmeerbeginselen

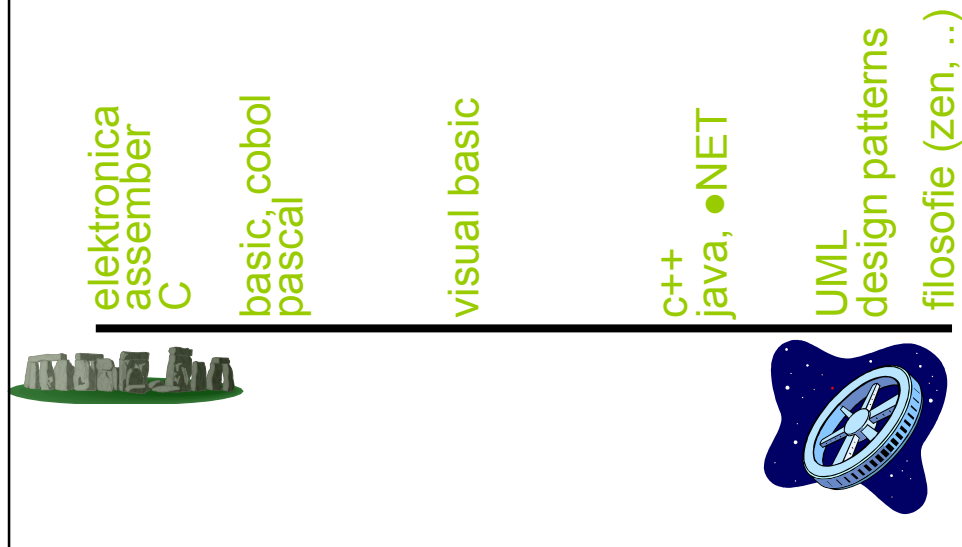
1

## inhoudstafel

- Wat, waarom, nadelen
- Eigenschappen
- Programmeertalen
- Methodologiën
- Distributed Objects

2

# Programmatieevolutie



## Wat is OO

- OO is een betere manier van werken omdat het nauwer aansluit bij de *real world* . En vooral bij onze wijze van denken dankzij de sleuteleigenschap **abstractie**.

## Wat is OO

Zonder abstractie zouden we onmogelijk de meest eenvoudige opgave kunnen uitvoeren, we zouden constact gehinderd worden door de overvloed van details. Er zouden immers alleen details zijn! Onze wereld bestaat uit objecten, dingen die iets zijn, doen, hebben, ...

5

## Wat is OO

- Klassiek programmeren
  - We schrijven een programma dat facturen verwerkt
- OO programmeren
  - We programmeren een factuur
  - We programmeren een factuurlijn
  - We programmeren een ...
  - Uiteindelijk programmeren we een programma dat factuurobjecten verwerkt

6

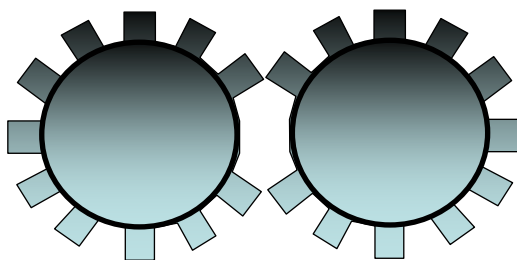
## Waarom OO

- ingebouwd hergebruik en gebruik van bestaande software
- 'eenvoudiger onderhoud'
- 'snellere ontwikkeling'
- 'minder fouten'

7

## Zijn er dan geen nadelen?

- Stel u voor een gigantische machine met duizenden kleine tandwielletjes van dezelfde soort waarbij in de oorspronkelijke mal één tandje ontbrak!



8

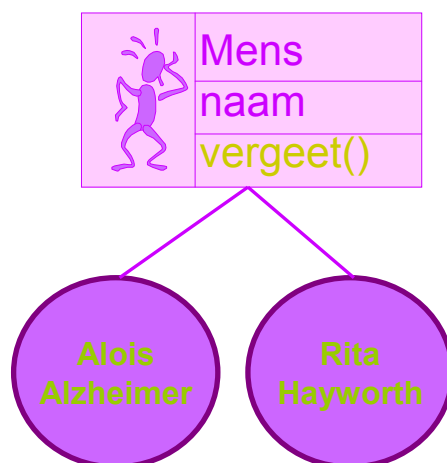
## Zijn er dan geen nadelen?

- Een kleine bug kan heel dikwijls worden hergebruikt.
- Starten zonder een degelijke analyse kan leiden tot het overboord gooien van de oo-principes halverwege het programmeren of afbraak van de bestaande code en een frisse herstart

9

## Wat is een Object

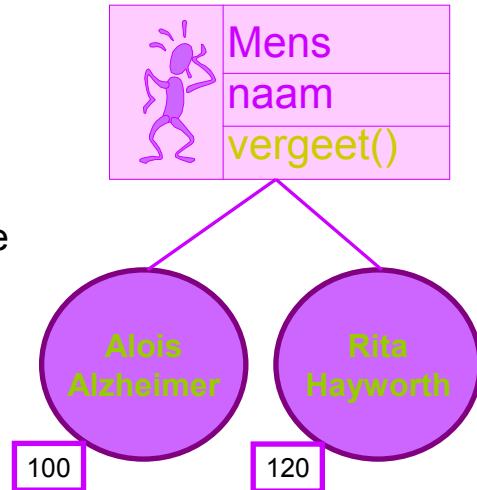
- Een object is een voorkomen/instance van een class.
- Zo zijn Alois A. en Rita H. voorkomens van de class Mens.



10

## Class vs. Object

- De class is de beschrijving van de objecten.
- Een object is een blokje geheugen, een variabele van een type van een class.
- Methods worden gedeeld op het niveau van de class.



11

## Wat is een class ?

- Een class is de beschrijving van een ding dat voorkomt in de scope die we moeten automatiseren.
- Wat zijn kandidaat classes ?
  - Tastbare dingen - een product
  - Specificaties van dingen – vluchtschema
  - Plaatsen – een vakje op een schaakbord
  - Een rol - een oude winkeldochter<sup>1</sup>
  - Een interactie die moet onthouden worden - een aankoop
  - Een container - een parking
  - Dingen in een container – een parkeerplaats

12

<sup>1</sup> een oude winkeldochter is een product dat reeds lang op stock is.

## Eigenschappen van OO

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

13

## Abstraction



- Dit is de basiseigenschap.  
We denken steeds in objecten.  
Een auto is een object. De  
tekening hiernaast herkennen  
wij ook als een auto.
- Niemand van ons heeft behoefte  
aan alle mogelijke  
detailinformatie om een auto te  
kunnen herkennen.

14

## Encapsulation

- Een auto bevat een motor, zitplaatsen, maximum snelheid, ... Dit zijn eigenschappen of onderdelen - attributen/variabelen/datamembers - deze worden vastgelegd bij de creatie van een auto-object.
- Een auto bevat ook functies/memberfuncties/methods zoals :
  - open portier
  - start motor

15

## Encapsulation

- Een verwante term is data-hiding
- Dit betekent dat de eigenschap - aantalZitplaatsen- niet rechtstreeks kan gewijzigd worden. De data wordt verborgen voor niet-auto-objecten. De data is voor de buitenwereld -vb een mens-object - alleen manipuleerbaar via functies - ook methods genoemd.  
Zo zou de functie klapAchterbank( *[op/nee]* ) het attribuut aantalZitplaatsen kunnen beïnvloeden.

16



## Encapsulation

- De achterliggende bedoeling is de datamember te beschermen. Hierdoor wordt de consistentie bewaakt en kan getest worden of de overgang van de ene naar de andere status toegestaan is. Via de functie `wijzigVersnelling` kan de wagen van status vooruit rijden naar achteruit rijden gebracht worden. De functie zal dan controleren of wijziging van status is toegestaan.

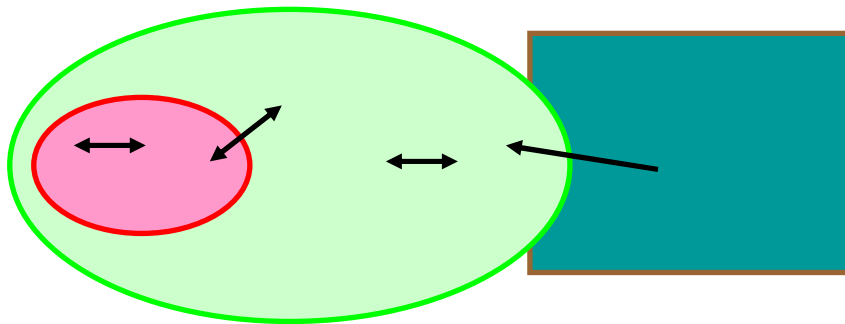
17

## Encapsulation

- Het beveiligde gebied is het **private** deel
- Het onbeveiligde deel is het **public** deel
- Zowel het **private** als **public** deel kunnen datamembers als memberfuncties bevatten.

18

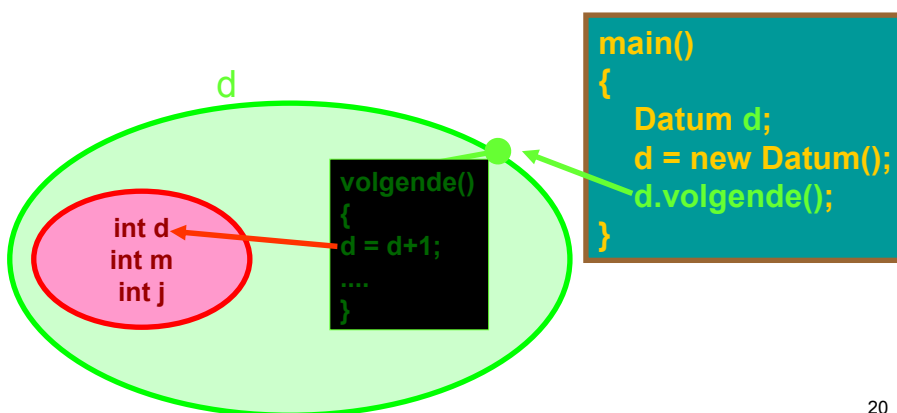
# Encapsulation



19

# Encapsulation

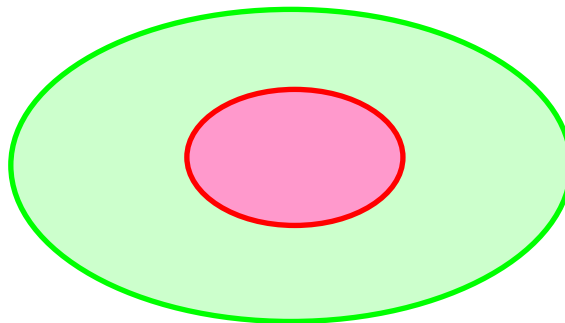
methods & datamembers ifv “buitenwereld”



20

## Oefening Encapsulation

- Welke datamembers en memberfuncties zou een datumobject hebben? Plaats de private in het rode<sup>1</sup> deel, de publieke in het groene<sup>2</sup> deel.



1 buiten cirkel  
2 binnen cirkel

21

## Oefening Encapsulation

- **Private :**
  - dag, maand, jaar
- **public :**
  - stelDatumIn (d, m, j),
  - gaNaarVorigeDag()
  - gaNaarVolgendeDag()
  - gaVerder( dagen )
  - gaTerug( dagen )

VerfierDatum(d,m,j)

Dag() <- dag

Maand() <- maand

Jaar() <- jaar

22

# Encapsulation

- Objecten maken:
  - Nieuwe objecten moeten correct geïnitieerd zijn.
  - Er is een speciale method om objecten te initialiseren. De constructor.
  - `Datum d = new Datum();`
    - New vraagt geheugen aan het OS.
    - De constructor zorgt voor de initialisatie.

23

# Encapsulation

- Objecten vernietigen
  - Objecten moeten opgeruimd zijn voor dat ze de vuilbak in gaan.
  - Er is een speciale method om objecten op te ruimen. De destructor.
  - `d=null;`
    - Wanneer we de referentie naar het object op null zetten, wordt de destructor uitgevoerd.
    - Het object zit klaar om uit het geheugen verwijderd te worden.

24

## Encapsulation

- geheugenbeheer gebeurt door de garbagecollector.
- LET OP! JIJ bepaalt NIET wanneer geheugen opgeruimd wordt, alleen hoeveel je wegwerpt!

25

## Encapsulation

- overloading
- n versies van dezelfde method
  - Datum()
  - Datum(int dag, int maand, int jaar)
- 2 functies zijn verschillend als
  - hun naam verschilt
  - hun naam gelijk is maar het parameterbeeld verschilt
- waarom? functie gelijk -> naam gelijk

26

## Encapsulation

- Objecten zijn niet steeds onderling uitwisselbaar.
- Een auto object gemaakt door een autofabrikant kan niet gebruikt worden binnen de context van een leasings-maatschappij.
- Datamembers en methods die een class bevat worden bepaald door de context waarin de class zal gebruikt worden.

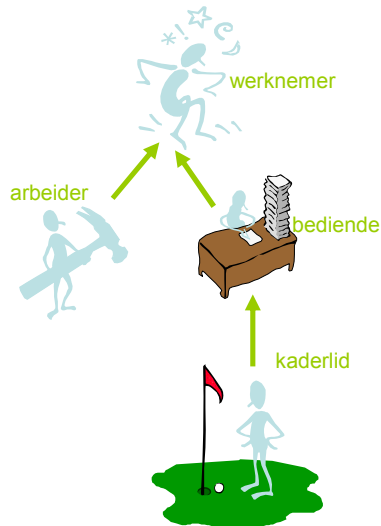
27

## Inheritance

- Dit is een natuurlijk klassificatiesysteem
- gebaseerd op specialisatie en generalisatie. De relatie tussen de classes is gebaseerd op 'is een', een arbeider is een werknemer.
- Een arbeider en bediende zijn beide werknemers. Een kaderlid is een bediende.
- De hoogste class in de hiërarchie is de base class, onderliggende classes zijn derived classes

28

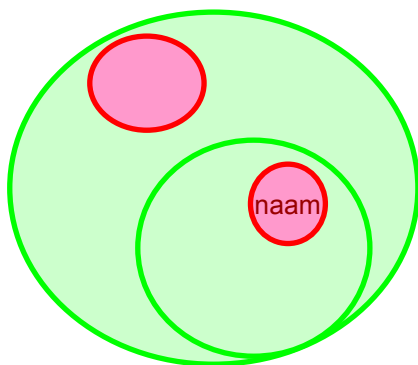
# Inheritance



- Door inheritance erft een class ( datamembers en memberfuncties) van z'n ouder.
- Er kunnen eigenschappen toegevoegd, onderdrukt of gewijzigd worden.

29

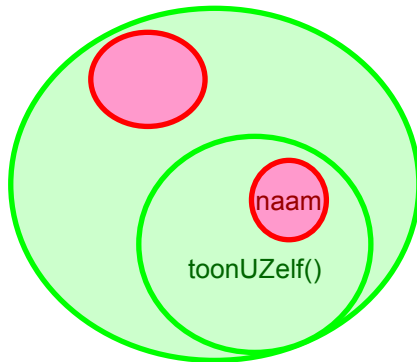
# Inheritance



- Wanneer een werknemer een naam heeft als private datamember, dan hebben alle afgeleide objecten ook deze naam.
- Deze naam kan opgevraagd worden door de public memberfunctie `toonUZelf()` en ingevuld door de publieke memberfunctie `leesDataInVanToetsenbord()`

30

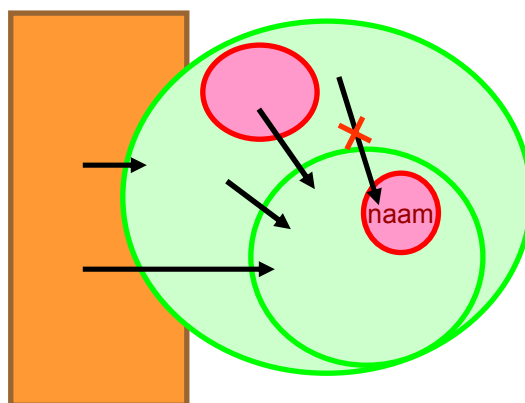
# Inheritance



- Het publiek deel van de base class loopt over in het publiek deel van de derived class. De buitenwereld -van de derived class- kan geen onderscheid maken tussen beide !
- De memberfunctie toonUZelf lijkt deel uit te maken van de derived class.

31

# Inheritance



- Het is echter niet toegestaan de naam te wijzigen vanuit een memberfunctie van een der afgeleide objecten.
- De naam is immers private aan werknemer. Hiervoor moet gebruik gemaakt worden van de memberfuncties van de base class.

32

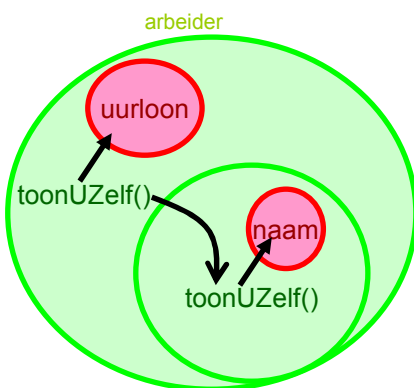


# Inheritance

- overriding
- functie herdefiniëren in afgeleide class
- voorwaarden
  - naam gelijk
  - parameterbeeld gelijk
  - returntype gelijk

33

# Inheritance



- Een Arbeider **is een** werknemer met een uurloon.
- Het is evident dat de oproep `toonUZelf()` van arbeider ook de naam toont. Dit kan doordat de memberfunctie `toonUZelf` van arbeider die van werknemer oproept. Wel op zijn beurt de naam toont.

34

## Inheritance

- Multiple inheritance ontstaat wanneer een class meerdere baseclasses heeft. Dit leidt tot vrij veel problemen omdat bv. een memberfunctie tot meerdere parents kan behoren en de compiler niet meer automatisch kan bepalen welke moet worden uitgevoerd

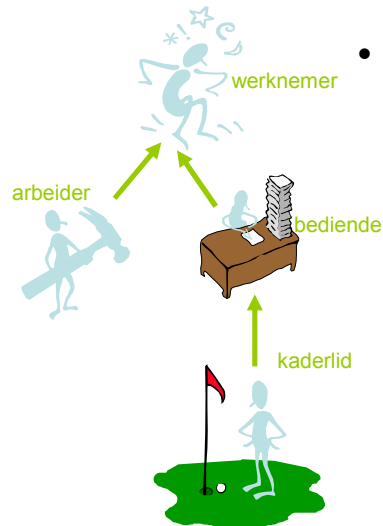
35

## Inheritance

- Dynamische inheritance maakt het mogelijk dat een object z'n parent wijzigt in de loop van het programma, *'at runtime'*

36

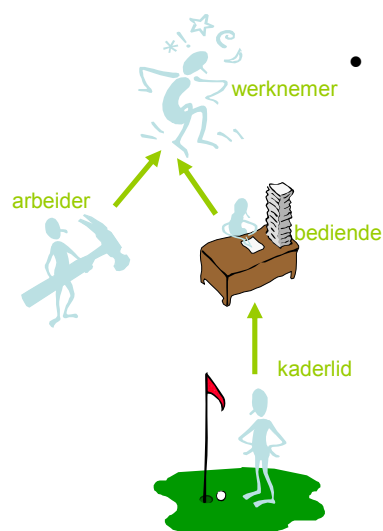
## Oefening Inheritance



- Welke members zou u toekennen aan arbeider, bediende en kaderlid

37

## Inheritance



- Wat zou het effect zijn op de derived classes als we in de class werknemer een datamember datumIndienst zouden toevoegen ?

**Antwoord :**  
**theoretisch GEEN !**

38

# Inheritance

- Praktisch moeten afgeleide classes aangepast worden.

```
Werknemer( Persnr, Naam)      -> Werknemer(Persnr, Naam, DatumInDienst)
  m_Persnr = Persnr;
  m_Naam = Naam;
  end Constructor

Arbeider( Persnr, Naam, Uurln) -> Arbeider(Persnr, Naam, Uurln, DatumInDienst)
  base(Persnr, Naam);
  m_Uurln = uurln;
  end Constructor

Werknemer(Persnr, Naam, DatumInDienst)
  m_Persnr = Persnr;
  m_Naam = Naam;
  m_DatumInDienst = DatumInDienst;
  end Constructor

Arbeider(Persnr, Naam, Uurln, DatumInDienst)
  base(Persnr, Naam, DatumInDienst);
  m_Uurln = uurln;
  end Constructor
```

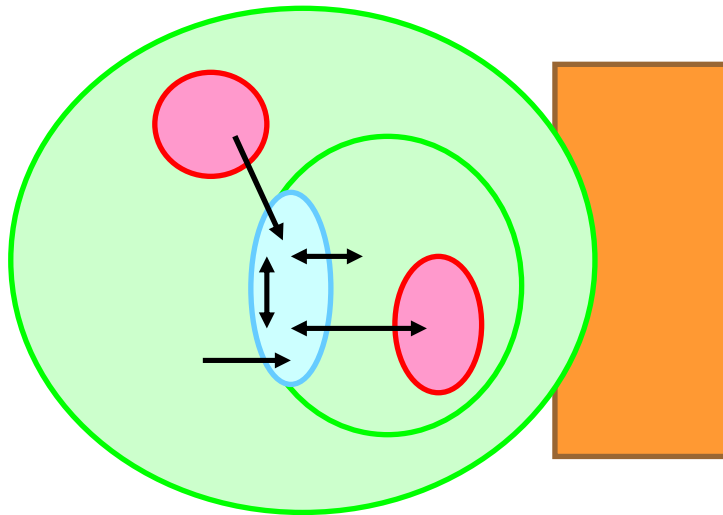
39

# Inheritance

- beveiligingsniveau **protected**
  - voor de buitenwereld lijkt dit gelijk aan private
  - voor een derived class is het evenwel toegankelijk
  - maar niet voor de buitenwereld van de derived class

40

## Inheritance



41

## Inheritance

- Wees extra spaarzaam met protected!
- Nodig?
  - Wijzig de inheritance structuur
  - Hoe hoger in de hiërarchie hoe flexibeler de regels die een class toepast
  - Hoe lager in de hiërarchie hoe strikter de regels die een class toepast

42

# Inheritance

- Geboortedatum
    - valideer { geboortedatum <= nu }
    - setJaar { valideer dag, maand, jaar }
  - Geboortedatum18plus inherits from GeboorteDatum
    - valideer { geboortedatum <= nu-18jaar }
    - setJaar { valideer dag, maand, jaar }
- 

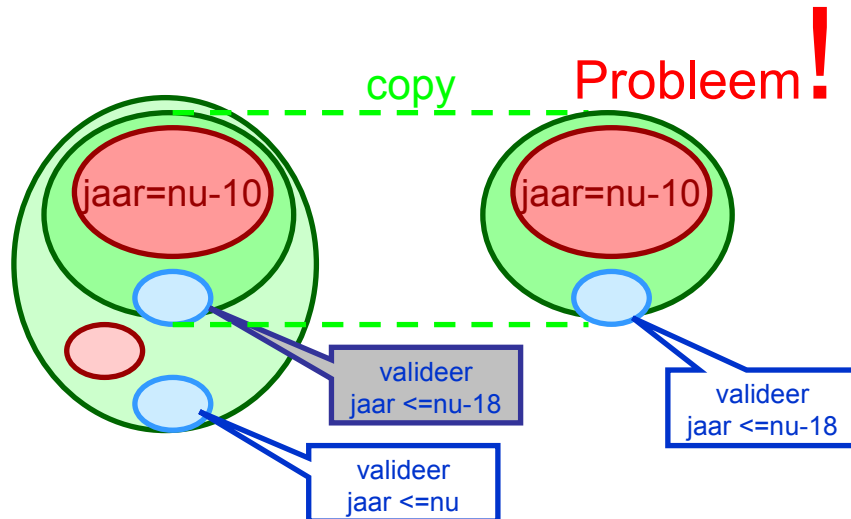
43

# Inheritance

- Geboortedatum18plus
    - valideer { geboortedatum <= nu-18jaar }
    - setJaar { valideer dag, maand, jaar }
  - Geboortedatum inherits from GeboorteDatum18plus
    - valideer { geboortedatum <= nu }
    - ~~– setJaar { valideer dag, maand, jaar }~~
- 

44

## Inheritance



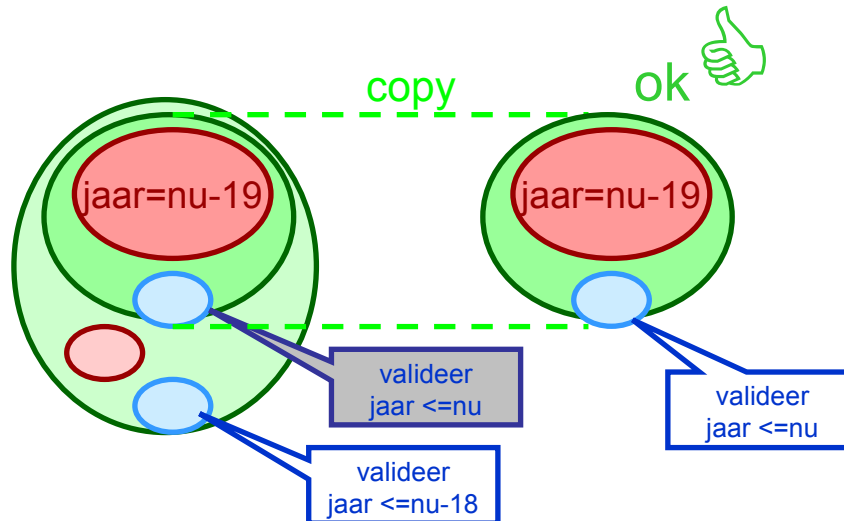
45

## Inheritance

- De class met minst strikte regels moet bovenaan de hierarchie staan.
- De meest restrictieve class onderaan

46

## Inheritance



47

## Polymorphism

- Wanneer we een verzameling -tabel- van werknemers hebben dan zullen dit arbeiders, bediende en kaderleden zijn.
- Op een verzameling van werknemers zullen we waarschijnlijk functies willen toepassen, afdrukken van een lijst, berekenen van het loon, controleren van de onkostendeclaratie, ...

48



# Inheritance

- `Werknemer w = new Bediende();`

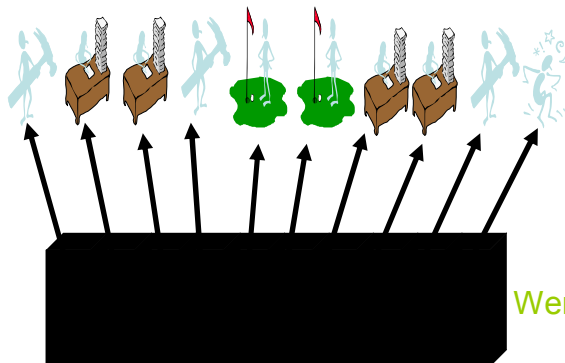
type van de referentie  
bepaald welke methods  
kunnen gebruikt worden

type van het object  
bepaald wat de  
methods doen

49

# polymorphism

- Met polymorphism wordt het mogelijk  
gelijkaardige functies uit te voeren op een  
heterogene verzameling.



50

# Polymorphism

- In C++ wordt polymorphism gerealiseerd d.m.v. virtuele functies.
- In Java is per definitie elke class polymorf. Maar het polymorphism kan door middel van een beveiliging worden opgeheven.
- Zonder polymorphism zou in voorgaand voorbeeld telkens de memberfunctie `toonUZelf` van `werknemer` worden uitgevoerd.

51

# Polymorphism

- Op een polymorfe verzameling kunnen alleen memberfuncties worden uitgevoerd die gedeclareerd zijn op het niveau van de base class van de verzameling.
- Op een verzameling werknemers kunnen alleen memberfuncties toegepast worden die bestaan in de class `werknemer`. De class `bediende` kan meer memberfuncties hebben. Op een polymorfe verzameling bedienden kunnen alle memberfuncties worden toegepast die bestaan op het niveau van `bediende`, geen specialistische functies van `arbeider`, noch van `kaderlid`.

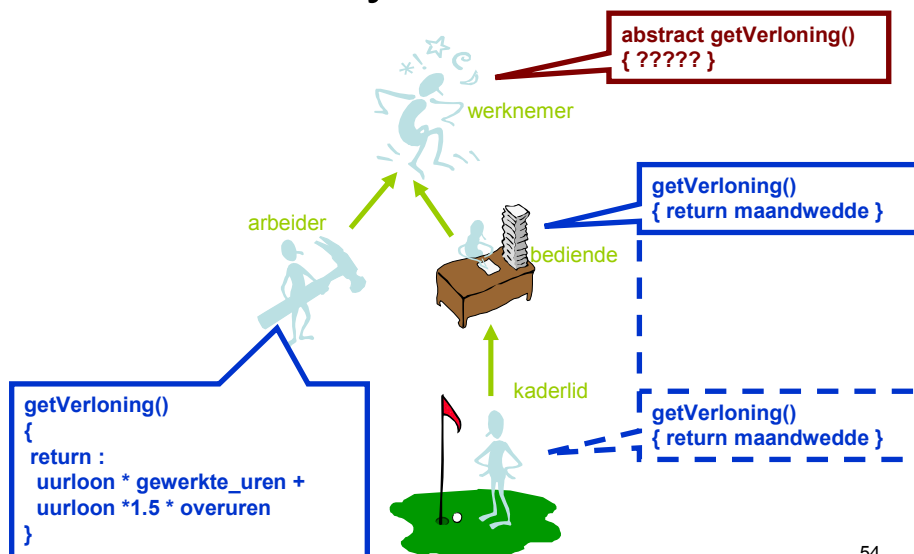
52

# Polymorphism

- Een class als werknemer wordt alleen gebruikt om classes te binden in de hiërarchie en om te kunnen werken met verzamelingen. Dit noemt men een **abstract base class**.
- Als een class een abstracte base class is kunnen er geen objecten van dat type gecreëerd worden.

53

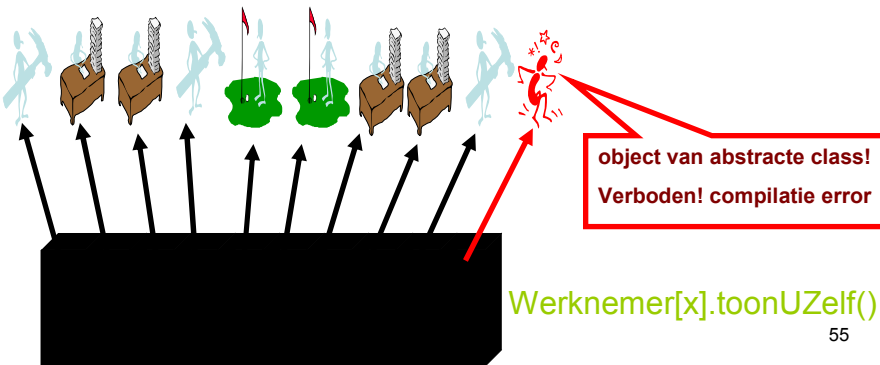
## Polymorfisme



54

# polymorphism

- Van abstracte classes kan men geen objecten maken. Referenties van abstracte classes kan wel.

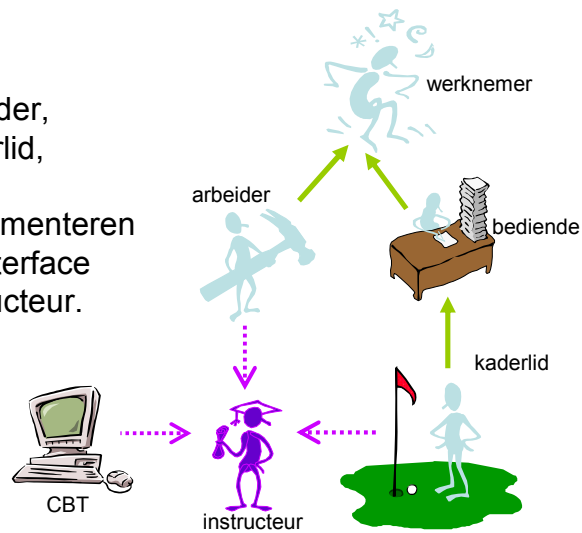


# Polymorfisme

- Interface
  - zuivere abstracte class
  - alleen abstracte methods
  - interface legt gedrag vast
    - inheritance bepaald wie/wat een class is
    - interface bepaald wat een class kan
    - gedrag wordt niet bepaald door afstamming
    - classes uit verschillende stammen kunnen zich hetzelfde gedragen, dus zelfde interface hebben
  - interface is een contract
    - class die een interface implementeert MOET alle methods implementeren anders is ze ook abstract.
    - een class kan meerdere interfaces implementeren

# polymorfisme

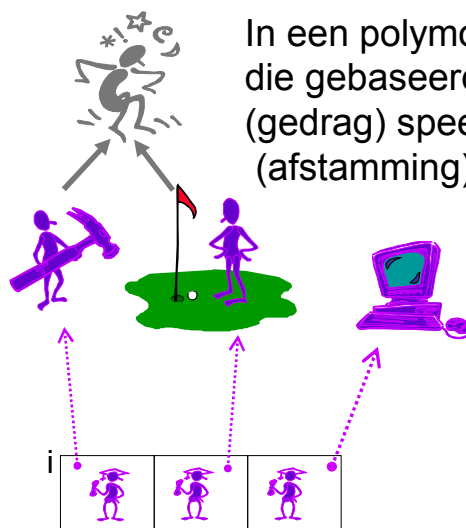
arbeider,  
kaderlid,  
CBT,  
implementeren  
de interface  
Instructeur.



57

# Polymorfisme

In een polymorfe verzameling  
die gebaseerd is op een interface  
(gedrag) speelt het type van de class  
(afstamming) geen enkel rol meer.



58

# Polymorphism

- Polymorphism wordt geïmplementeerd m.b.v. **pointers** naar tabellen van memberfuncties. Daarom werkt het alleen met pointers en referenties.
- Wanneer een functie een polymorfe objectverwijzing als parameter meekrijgt dan zal elke wijziging aan dit object onvermijdelijk het oorspronkelijke object wijzigen.

59

# Polymorfisme

```
 x = new  ();  
object.fun( x );
```

---


```
fun( y );  
  y.instructeursmethod() toegestaan  
  y.arbeidersmethod()   verboden
```

60

# Polymorfisme

```
 x = new  ();  
object.fun( x );
```

---

```
fun( y );  
  y.werknemersmethod() toegestaan  
  y.arbeidersmethod()   verboden
```

61

# Polymorphism

- Polymorphism wordt geïmplementeerd m.b.v. **pointers** naar tabellen van memberfuncties. Daarom werkt het alleen met pointers en referenties.
- Wanneer een functie een polymorfe objectverwijzing als parameter meekrijgt dan zal elke wijziging aan dit object onvermijdelijk het oorspronkelijke object wijzigen.

62

# Polymorfisme

```
x = new  ();  
object.fun( x );  
fun(  y );  
y.setPersoneelsnr(007)  
x.getPersoneelsnr()
```

The diagram illustrates polymorphism through a series of code snippets and callouts:

- `x = new ();`: A callout box indicates `personeelsnummer=001`.
- `object.fun( x );`: No callout.
- `fun( y );`: A callout box indicates `y.setPersoneelsnr(007)`.
- `y.setPersoneelsnr(007)`: A callout box indicates `y.setPersoneelsnr(007)`.
- `x.getPersoneelsnr()`: A callout box indicates `personeelsnummer=007`.

63

# Polymorphism

Als dit ongewenst is moet een locale kopie gemaakt worden.

- Het maken van een kopie moet door het object zelf gebeuren. Deze techniek noemt men cloning. De functie kan men overbelasten met alle mogelijk kennis van de afgeleide classes van de parameter maar dan moet bij een wijziging aan de inheritance tree ook deze functie wijzigen !



64



## Polymorfisme

```
 x = new  ();  
object.fun( x );
```

---

```
fun( y );  
ybis = new  ( y )    compilatieerror  
ybis = y.clone()    ok
```

65

## Polymorfisme

```
 x = new  ();  
object.fun( x );
```

---

```
fun( y );  
ybis = new  ( y )    informatieverlies  
ybis = y.clone()    ok
```

66

## Containment aka Encapsulation

- deze relatie is gebaseerd op hebben en niet op zijn.
    - een arbeider is een werknemer -> inheritance
    - een werknemer heeft een datum indienst -> containment
- Een datum object wordt gebruikt als datamember in een werknemer object.  
Dit “containt” object is niet meer dan een gewone variabele die private of public kan zijn

67

## Essentie van OO = verantwoordelijkheid

Elk object moet streven naar een beperkte verantwoordelijkheid.

- De klusjesman die het bedrijfsgrasmaschine van de autoconstructeur onderhoudt moet zich alleen bezighouden met wat tot de kern van z'n taak behoort. De loonadministratie en algemeen beleid horen daar niet toe, dus laat deze dat over aan anderen.

68

## Essentie van OO = verantwoordelijkheid

Elk object moet streven naar een beperkte verantwoordelijkheid.

- Een factuur moet zich alleen bezig houden met een factuur te zijn. M.a.w het moet zich niet bezig houden met de controle of dat een datum een echte goede datum is. Het factuur moet zich ook niet bezig houden met de controle of dat de gefactureerde artikels echt bestaan.

69

## Essentie van OO = verantwoordelijkheid

Om een bepaalde verantwoordelijkheid op te kunnen nemen moet het object beschikken over voldoende data.

- De PDG van een autoconstructeur zal het onderhoud van de bedrijfsgrasmachine delegeren aan diegene die over de nodige “data” beschikt om deze te kunnen onderhouden

70

## Essentie van OO = verantwoordelijkheid

Om een bepaalde verantwoordelijkheid op te kunnen nemen moet het object beschikken over voldoende data.

- Een factuur moet zichzelf niet kunnen nummeren anders zou elk factuur van het bestaan van elk ander factuur moeten op de hoogte zijn. We zouden dit kunnen over laten aan een factuurmanager.

71

## Essentie van OO = verantwoordelijkheid

De verantwoordelijkheid moet tot de essentie van het “zijn van het object” behoren.

- De PDG van een autoconstructeur zal zo veel mogelijk taken delegeren. En indien hij dit doet zullen zij ook delegeren. Dit proces gaat verder totdat de verantwoordelijkheid komt bij iemand die niet de verantwoordelijkheid heeft om te delegeren of het zijn taak vindt.

Het personeelsbeleid wordt bepaald door de personeelsmanager, maar het berekenen van de lonen zal deze op haar beurt delegeren.

72

## Essentie van OO = verantwoordelijkheid

De verantwoordelijkheid moet tot de essentie van het “zijn van het object” behoren.

- Een factuur zal voor het noteren van de factuurlijnen beroep doen op gespecialiseerde objecten van het type FactuurLijn. Voor de klantgegevens zal het factuur beroep doen een object van het type Klant. De taak van een factuur is deze gegevens te binden.

73

## Essentie van OO = verantwoordelijkheid

- Let op, het is niet omdat de PDG het personeelsbeleid delegeert dat het niet meer tot z'n verantwoordelijkheid behoort.
- De kerntaak van de PDG is niet het personeelsbeleid dus delegeert hij de verantwoordelijkheid.

74

## Essentie van OO = verantwoordelijkheid

- Let op, het is niet omdat de Factuur de detail van de factuurlijnen delegeert aan FactuurLijn objecten dat het niet meer tot z'n verantwoordelijkheid behoort. Het Factuur object zal FactuurLijn objecten aanmaken laten aanmaken of ontvangen.
- De kerntaak van een Factuur is een klant te bewijzen hoeveel hij moet betalen en waarom. Het waarom bewijzen is gedelegeerd naar de FactuurLijn objecten.

75

## Essentie van OO = verantwoordelijkheid

Om verantwoordelijkheden afdwingbaar te maken werken we in de realiteit met contracten. Dit zal ook in de programmatiewereld gebeuren. Voor elke functionaliteit die we in een object inbouwen zullen we een contract met het object afsluiten.

76

## Essentie van OO = verantwoordelijkheid

De minimale inhoud van een contract zijn de pre en postcondities.

- De preconditionie beschrijft onder welke voorwaarden op voorhand moeten vervuld zijn opdat het object z'n verantwoordelijkheid zou kunnen nemen.
- De postconditie beschrijft wat er gebeurd is als het object z'n verantwoordelijkheid heeft genomen. De postconditie wordt best in de verleden tijd opgesteld.

77

## OO programmeertalen

- De bekendste zijn : (willekeurige volgorde)
  - Java
  - C#, VB.NET
  - C++
  - Smalltalk
  - Eiffel
  - Ada
  - perl
  - Python
  - Objective-C
  - Object Pascal
  - Modula-3

78

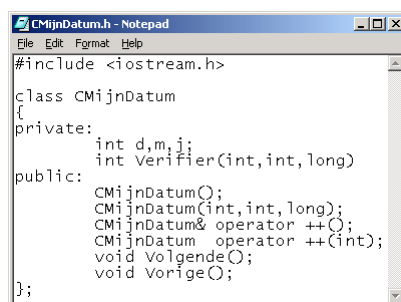
## OO Programmeertalen C++

- Afgeleid van C
- Vrij complexe en uitgebreide taal.
- De enige objecten die meegeleverd worden zijn de input en output objecten.
- Een der meest bekende libraries van classes is MFC, ingebed in visual C++
- ondersteuning voor
  - multiple inheritance
  - private en protected inheritance
  - metaclasses of m.a.w. parametertypes

79

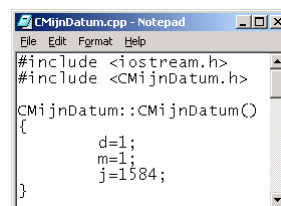
## OO programmeertalen C++

- Een voorbeeldje



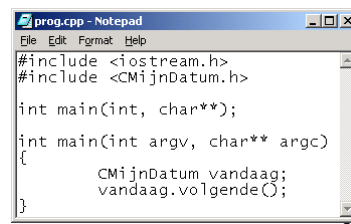
```
CMijnDatum.h - Notepad
File Edit Format Help
#include <iostream.h>

class CMijnDatum
{
private:
    int d,m,j;
    int Verifier(int,int,long)
public:
    CMijnDatum();
    CMijnDatum(int,int,long);
    CMijnDatum& operator ++();
    CMijnDatum operator ++(int);
    void volgende();
    void vorige();
};
```



```
CMijnDatum.cpp - Notepad
File Edit Format Help
#include <iostream.h>
#include <CMijnDatum.h>

CMijnDatum::CMijnDatum()
{
    d=1;
    m=1;
    j=1584;
}
```



```
prog.cpp - Notepad
File Edit Format Help
#include <iostream.h>
#include <CMijnDatum.h>

int main(int, char**);

int main(int argv, char** argc)
{
    CMijnDatum vandaag;
    vandaag.volgende();
}
```



## OO Programmeertalen C++ & MFC

- MFC is een library met ongeveer basis 315 classes
- Communicatie tussen objecten verloopt dikwijls door het versturen van messages via het operating systeem ipv rechtstreekse method/functie-aanroepen

81

## Java

```
public class Datum
{
    private int dag=1,int maand=1;
    private int jaar=1584;

    public Datum() { }
    public Datum(int dag,int maand,int jaar)
    { setDatum(dag, maand, jaar); }

    public void setDatum(int dag, int maand, int jaar)
    {
        if( valideerDatum(dag, maand, jaar) )
        {
            this.dag=dag;
            this.maand=maand;
            this.jaar=jaar;
        } }
}
```

82

# Java

```
public int getDag()    { return this.dag; }
public int getMaand() { return this.maand; }
public int getJaar()   { return this.jaar; }

public void setDag(int dag) {
    if( valideerDatum(dag, maand, jaar) )
        this.dag = dag; }
public void setMaand(int maand) {
    if( valideerDatum(dag, maand, jaar) )
        this.maand = maand; }
public void setJaar(int jaar) {
    if( valideerDatum(dag, maand, jaar) )
        this.jaar = jaar; }
```

83

# Java

```
private boolean valideerDatum(int dag, int maand, int jaar)
{
    int dvdm[]={31,28,31,30,31,30,31,30,31,30,31};
    if( jaar%4==0 && jaar%100!=0 || jaar%400==0)
        dvdm[1]+=1;
    return  maand>0    && maand < 13    &&
           dag  >0    && dag   <= dvdm[maand-1] &&
           jaar  > 1583 && jaar  < 4100;
}

public void toString ()
{
    return dag+"/"+maand+"/"+jaar;
}
}
```

84

## OO Programmeertalen Java

- **Get started quickly:** Although Java is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in Java can be four times smaller than the same program in C++.
- **Write better code:** The Java language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Java's object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.

85

## OO Programmeertalen Java

- **Develop programs faster:** Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code with Java and Java is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java:** You can keep your program portable by following the purity tips mentioned throughout this tutorial and avoiding the use of libraries written in other languages.
- **Write once, run anywhere:** Because 100% Pure Java programs are compiled into machine-independent bytecodes, they run consistently on any Java platform.
- **Distribute software more easily:** You can upgrade applets easily from a central server. Applets take advantage of the Java feature of allowing new classes to be loaded "on the fly," without recompiling the entire program.

86

## OO Programmeertalen Java

- Bijna volledige oo taal, uitgezonderd primitieve types zoals int, float, char, ...
- Bevat bij benadering **2900 classes**
- Conversie naar bytecode, wordt opgehaald, grondig gecontroleerd - veiligheid- en uitgevoerd door een java processor bestaat zowel in soft -jre- als hardware vorm !

87

## OO Programmeertalen Java

- **The Essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by Java applets.
- **Networking**: URLs, TCP and UDP sockets, and IP addresses.
- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security**: Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.

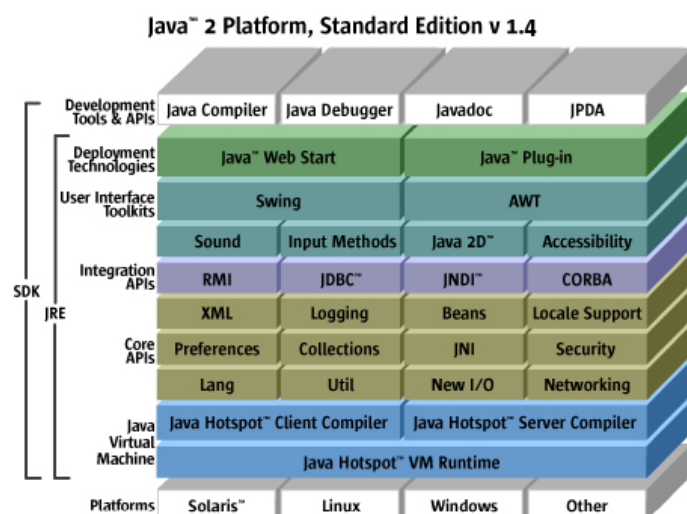
88

## OO Programmeertalen Java

- **Software components:** Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Live Connect.
- **Object serialization:** Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC):** Provides uniform access to a wide range of relational databases.

89

## OO Programmeertalen Java



90

# OO Programmeertalen Java

Java wordt constant uitgebreidt

- Java 2D, Java 3D, Java sound, Java™ Speech  
Support for easy manipulation of 2d and 3d object, light effects and multimedia
- Enterprise JavaBeans™ Architecture  
The Enterprise JavaBeans specification defines an API that will make it easy for developers to create, deploy and manage cross-platform, component-based enterprise applications that work within the framework of the systems currently in use.
- JavaServer Pages™  
The JavaServer Pages™ technology provides a simplified, fast way to create dynamic web content. JSP technology enables rapid development of web-based applications that are server and platform independent.
- Java™ Servlet  
The Java™ Servlet API provides web application developers with a simple, consistent mechanism for extending the functionality of a web server.

91

# OO Programmeertalen Java

- J2EE™ Connector  
The J2EE Connector architecture defines a standard architecture for connecting the J2EE platform to heterogenous Enterprise Information Systems.
- Java Naming and Directory Interface™ (JNDI)  
Provides uniform, industry-standard, seamless connectivity from the Java platform to business information assets, thus allowing developers to deliver Java applications with unified access to multiple naming and directory services across the enterprise.
- Java™ Interface Definition Language (IDL)  
Provides interoperability with CORBA, the industry standard for heterogeneous computing. Java IDL includes an IDL-to-Java compiler and a lightweight ORB that supports IIOP.
- Java™ Message Service (JMS)  
The Java Message Service specification provides developers with a standard Java API for enterprise messaging services such as reliable queuing, publish and subscribe communication and various aspects of push/pull technologies.

92

# OO Programmeertalen Java

- **Java™ Transaction API (JTA)**

Java Transaction API (JTA) defines a high-level transaction management specification intended for resource managers and transactional applications in distributed transaction systems.

- **Java™ Transaction Service (JTS)**

The Java Transaction Service (JTS) API technology ensures interoperability with sophisticated transaction resources such as transactional application programs, resource managers, transaction processing monitors and transaction managers. Since these components are provided by different vendors, JTS provides open, standard access to these transaction resources.

- **JavaMail**

The JavaMail API provides a set of abstract classes that models a mail system. The API is meant to provide a platform independent and protocol independent framework to build Java-based mail and messaging applications.

93

# OO Programmeertalen Java

- **Jini™ connection technology**

Jini technology is based on a simple concept. Devices should work together. They should simply connect. No drivers to find, no operating system issues, no weird cables and connectors. Yet Jini technology also introduces some exciting new concepts and capabilities.

- **JavaSpaces™ technology**



This is a simple unified mechanism for dynamic communication, coordination, and sharing of objects between Java™ technology-based network resources like clients and servers. In a distributed environment, JavaSpaces technology acts as a virtual glue between providers and requesters of network resources or objects. This allows participants in a distributed solution to exchange tasks, requests and information in the form of Java technology-based objects.

JavaSpaces technology provides developers with the ability to create and store objects with persistence, which allows for process integrity

94

## OO Programmeertalen •net

- “Strategie om software aan te bieden als een dienst” ( Microsoft .NET FAQ)
- Van losse sites en apparaten naar samenwerkende sites en samenwerkende apparaten.
- •net zorgt voor gedistribueerde diensten voer het internet

95

## OO Programmeertalen •net

- Applicaties bieden hun diensten aan als Webservices.
- Ontwikkelaars integreren Web services in hun applicaties door Web APIs op te roepen op dezelfde manier als ze locale methods oproepen.
- Component-programming over het Web.
- The .NET Runtime is ontworpen om een eerste klasse ondersteuning te bieden voor de ontwikkeling van componenten.

96



## OO Programmeertalen •net

- integratie door publieke standaarden
- multilanguage support
- schaalbaar door een losjes gekoppelde architectuur
- versterkt de productiviteit van de ontwikkelaar
- geavanceerde security
- ondersteunt door diensten van het OS

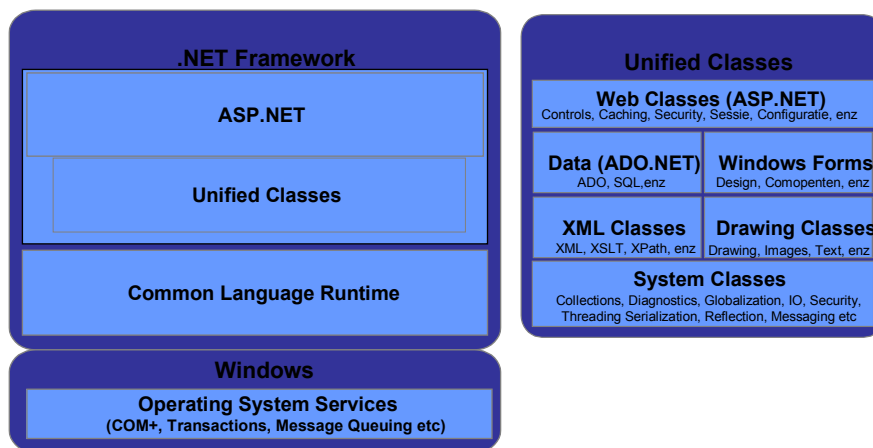
97

## OO Programmeertalen •net

- Next Generation Framework:
  - COM+ (COM+MTS+DCOM+... already in W2K )
  - ASP+ (ASP.NET)
  - ADO+ (ADO.NET)
- Next Generation Software:
  - Visual Studio .NET
  - Office .NET
  - Windows .NET (.NET integrated within the OS)

98

# OO Programmeertalen •net



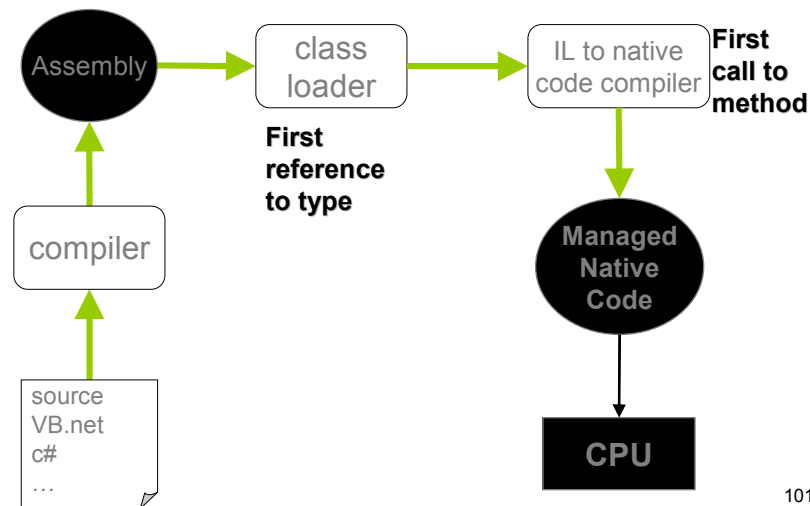
99

# OO Programmeertalen •net

- De source wordt gecompileerd naar Intermediate Language (IL)
- Een "common type" systeem laat toe om componenten geschreven in verschillende programmeertalen te integreren
- De Runtime voert de applicatie uit in een veilige omgeving nadat de verificatie en de conversie naar native code gebeurt is door de IL.

100

## OO Programmeertalen •net



101

## OO Programmeertalen •net

- De Common Language Runtime is de "execution engine" voor de .NET Framework applications:
  - CTS Common Type System
  - CLS Common Language Specification
  - CIL Common Language Infrastructure & Metadata
  - VES Virtual Execution System

102

## OO Programmeertalen •net

- Het Common Type Systeem :
  - is rijk aan types die ingebouwd zijn in de Common Language Runtime.
  - ondersteunt types uit de meeste programmeertalen.

103

## OO Programmeertalen •net

- De Common Language Specification is een set van constructs en constraints die dienen als een gids voor library ontwikkelaars en compiler ontwikkelaars.
- De CLS allows libraries to be fully usable from any language supporting the CLS, and for those languages to integrate with each other.
- Supported languages: C#, VB, C++, Python, Perl, Eiffel, Cobol, Smalltalk, Haskell, ML, Scheme.

104

## OO Programmeertalen •net

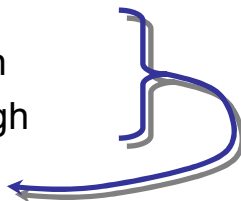
- MSIL = Microsoft Intermediate Language = Microsoft CIL
- MSIL is de CPU-onafhankelijke instructie set waarin .NET Framework programma's worden gecompileerd.
- MSIL wordt omgezet naar machine code door een JITcompiler. JIT=(Just In Time). Dus geen interpreter!

105

## OO methodologieën

- De bekendste zijn:

- Coad/Yourdon
- Booch
- Jacobson
- Rumbaugh
- UML



106

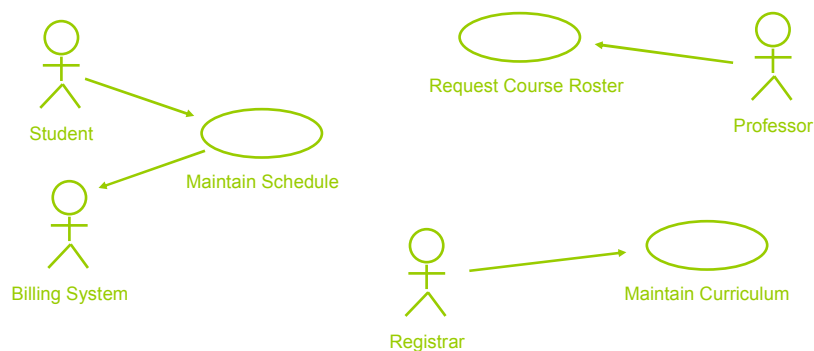
## OO methodologieën UML

- UML is een vrij verspreide methodologie voor OO.
- Er is softwareondersteuning in de vorm van [Together](#), [Poseidon](#), [Rational Rose](#)
- Bestaat uit een reeks van views die samen het probleemgebied beschrijven
  - Er wordt vertrokken vanuit de **use case**, dit is het gebruikerszicht op het probleem  
De schema noemt men **case diagrams**

107

## OO methodologiën UML

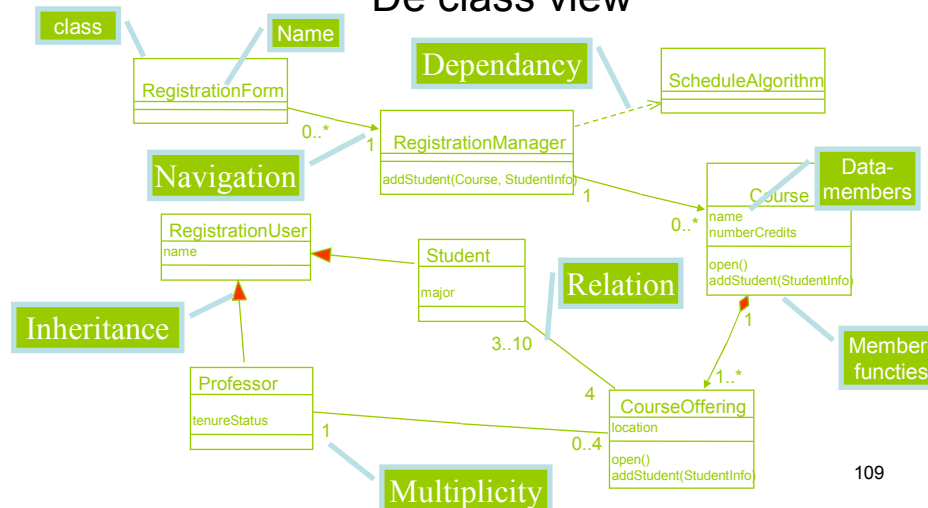
Een case diagram, de ovalen zijn de use cases, de mannetjes zijn de actors



108

# OO Methodologiën UML

## De class view



109

# OO Methodologiën UML

- associaties tussen Classes
  - Associaties beschrijven structurele verbanden.
  - Het verband blijft een .... tijd bestaan.



- Mulplicity

- 1 op 1 relaties: Mens — Geboortedatum
- 1 op n relaties:
  - Mens — 33 Wervels
  - Mens — \* Vriend

110

## OO Methodologiën UML

- benoemde relaties

- schept duidelijkheid.



- Rol in de relatie



111

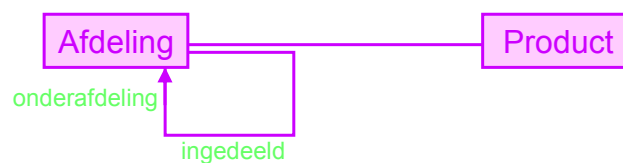
## OO Methodologiën UML

- richting van de relatie

- relaties zijn meestal niet tweezijdig.  
Soms zegt de rol onvoldoende



- Reflexatieve relatie



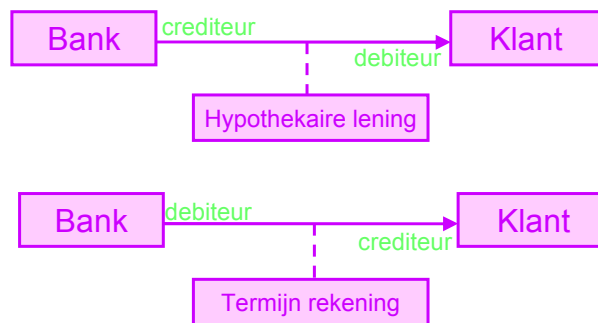
112



## OO Methodologiën UML

- Associatie classes

- Even relatie is soms complex en heeft die attributen. Dan gebruik je een class om de relatie te instantiëren.



113

## OO Methodologiën UML

- Aggregatie

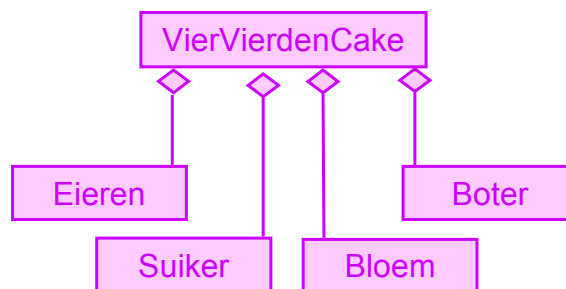
- benadrukt dat de "onderdelen" de "container" kunnen overleven.



114

# OO Methodologiën UML

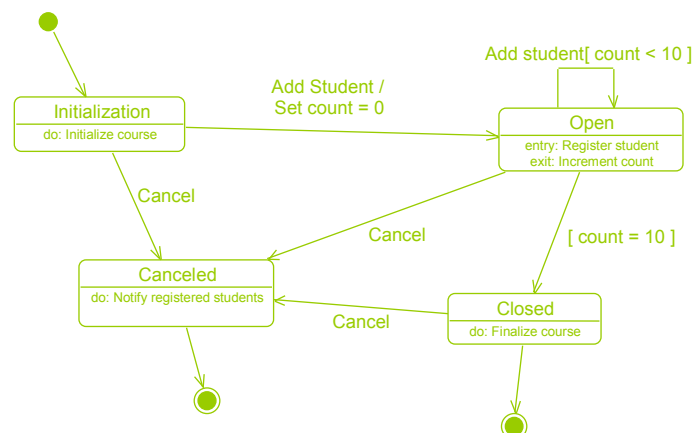
- Compositie
  - benadrukt dat de "onderdelen" de "container" niet overleven.



115

# OO methodologiën UML

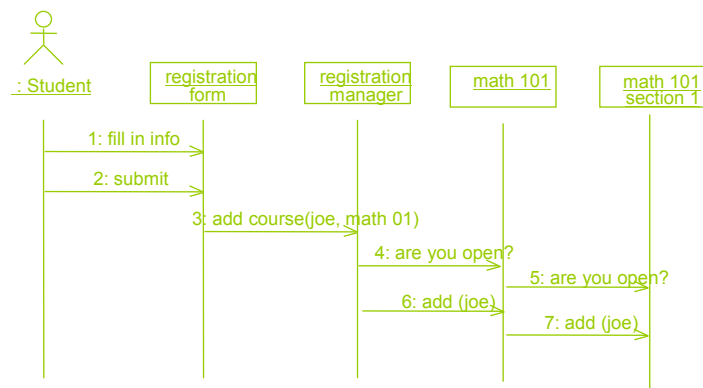
## State transition diagram



116

# OO methodologiën UML

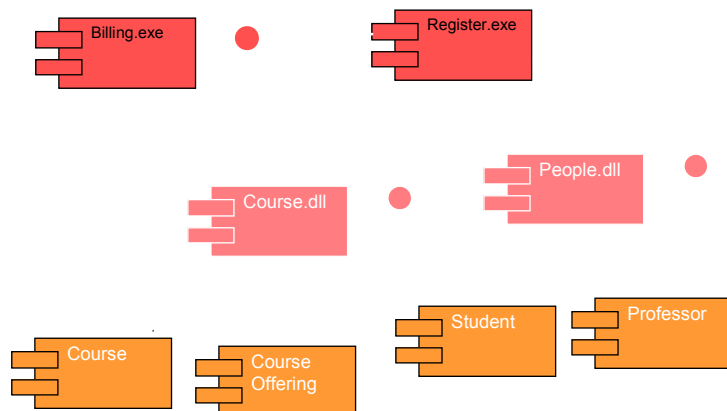
## Het sequence diagram



117

# OO methodologiën

## Het component diagram



118

## OO methodologiën UML

- Om UML te kunnen gebruiken hebben we een “manier van werken” nodig.
- Een manier van werken die goed aansluit bij UML is RUP, het Rational Unified Process.

119

## Het Rational Unified Process

- In essentie volgt RUP de volgende stappen:
  - Use Cases, zicht vanuit de buitenkant
  - Conceptueel model, zicht vanuit de binnenkant
  - System Sequence Diagrams, algemeen zicht op de interactie tussen buitenkant en binnenkant (per use case), “welke event voert de gebruiker uit het systeem”
  - Contracten voor alle systeemevents in alle SSD's
  - Interactiediagrammen voor elk systeem event, sequence diagrams of collaboration diagrams
  - Het class diagram met alle datamembers en member functies

120

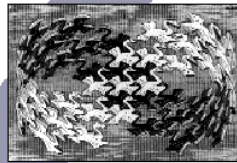
# Design Patterns

- Oplossingen voor veel voorkomende oo problemen

## 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

121

## Design patterns Soorten van patronen

- Creatiepatronen
  - Ze isoleren en verbergen de kennis over welke concrete classes het systeem gebruikt, hoe instanties van een bepaalde class gemaakt worden en waaruit ze samengesteld zijn.
- Structuurpatronen
  - Ze laten classes en objecten in een groter geheel efficiënt samenwerken.
- Gedragspatronen
  - Ze behandelen algoritmen en de toewijzing van verantwoordelijkheden aan objecten. Ze beschrijven patronen van objecten en classes, maar ook communicatiepatronen tussen objecten

122

## Design patterns

### singleton pattern, creatiepatroon

- Beheert de creatie van objecten, zodat er slechts één enkel object kan bestaan.
- Iedereen deelt dit ene object.
- Alternatief voor globale variabelen.
- Gebruikt overal waar iets moet beheerd worden.  
(ook in het echte leven heeft men best voor iets slechts een beheerder)

123

## Design patterns

### singleton pattern, creatiepatroon

```
• public class IconSizeManager // JAVA
{
    private int toolbarSize = 16;
    private static IconSizeManager singleton;
    private IconSizeManager() {}
    public static IconSizeManager getInstance()
    {
        if( singleton == null)
            singleton = new IconSizeManager();
        return singleton;
    }

    public synchronized int getToolbarSize()
    {
        return toolbarSize;
    }
    public synchronized void setToolbarSize(int size)
    {
        if(size == 16|| size == 24)
            toolbarSize=size;
    }
}
```

124

# Design patterns

## singleton pattern, creatiepatroon

```
• Class IconSizeManager `VB.NET
  Private iToolbarsize As Integer = 16
  Private Shared singleton As IconSizeManager

  Private Sub New ()
  End Sub

  Public Shared Function getInstance() As IconSizeManager
    If singleton Is Nothing Then
      singleton = new IconSizeManager()
    End If
    return singleton
  End Function

  Public Property ToolbarSize() As Integer
    Get
      Return iToolbarsize
    End Get
    Set
      if Value = 16 Or Value = 24 Then
        iToolbarSize=Value
      End If
    End Set
  End Property
End Class
```

125

## En nu?

- UML
  - VDAB cursus
- Java Basis
  - VDAB cursus
- Vb.net
  - VDAB cursus
- c#
  - VDAB cursus

126