

Object-georiënteerd Programmeren II

Prof. dr. Kris Luyten

Agenda

1. **Ons OO Mantra:** afspraken, regels en vuistregels
2. **Defensief programmeren:**
Contracten
3. **Ontwerp van en interacties tussen klassen:** GRASP, Design Patterns
4. **Exception Handling** (zelfstudie!)
5. Programmeren!

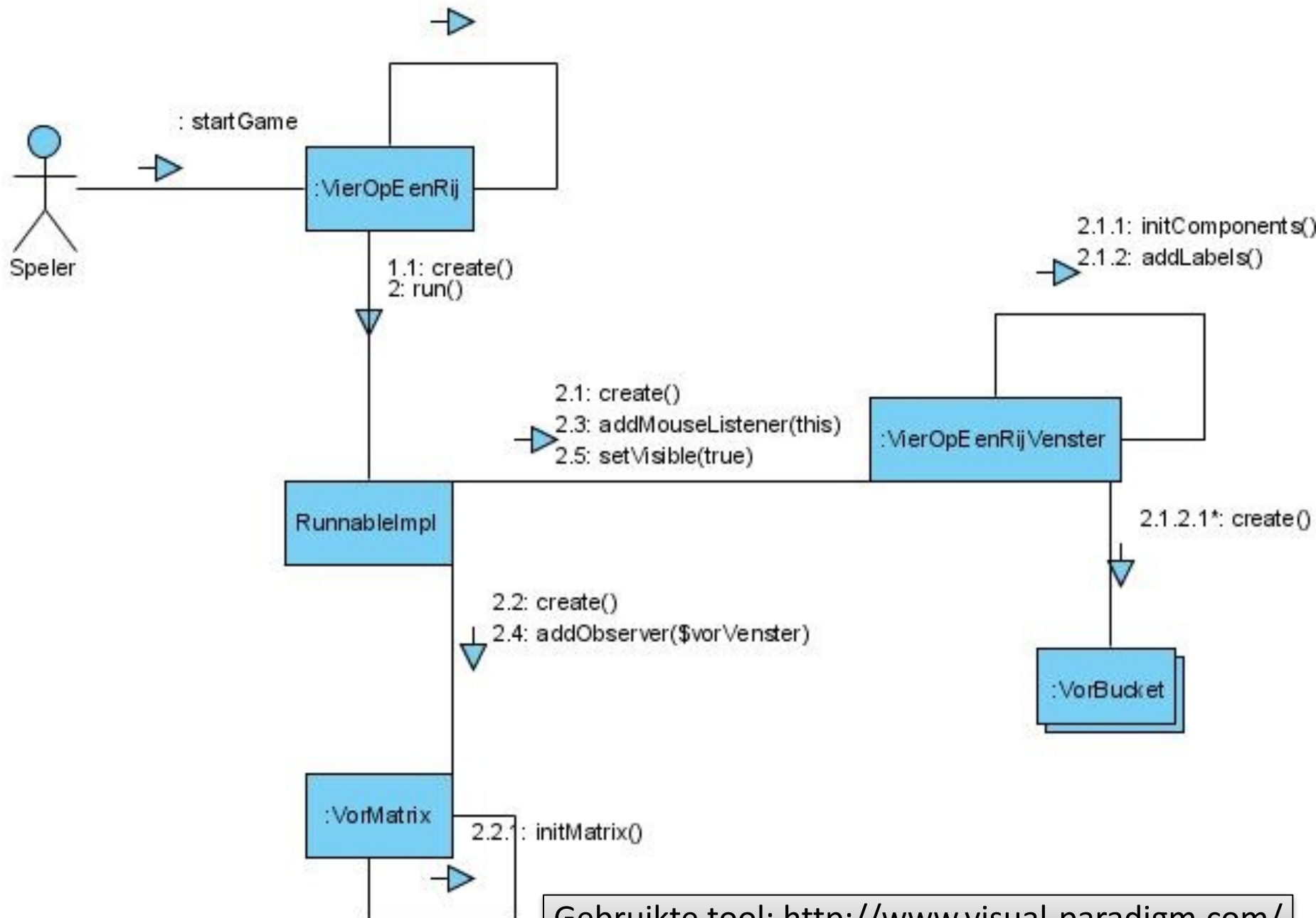
Intermezzo

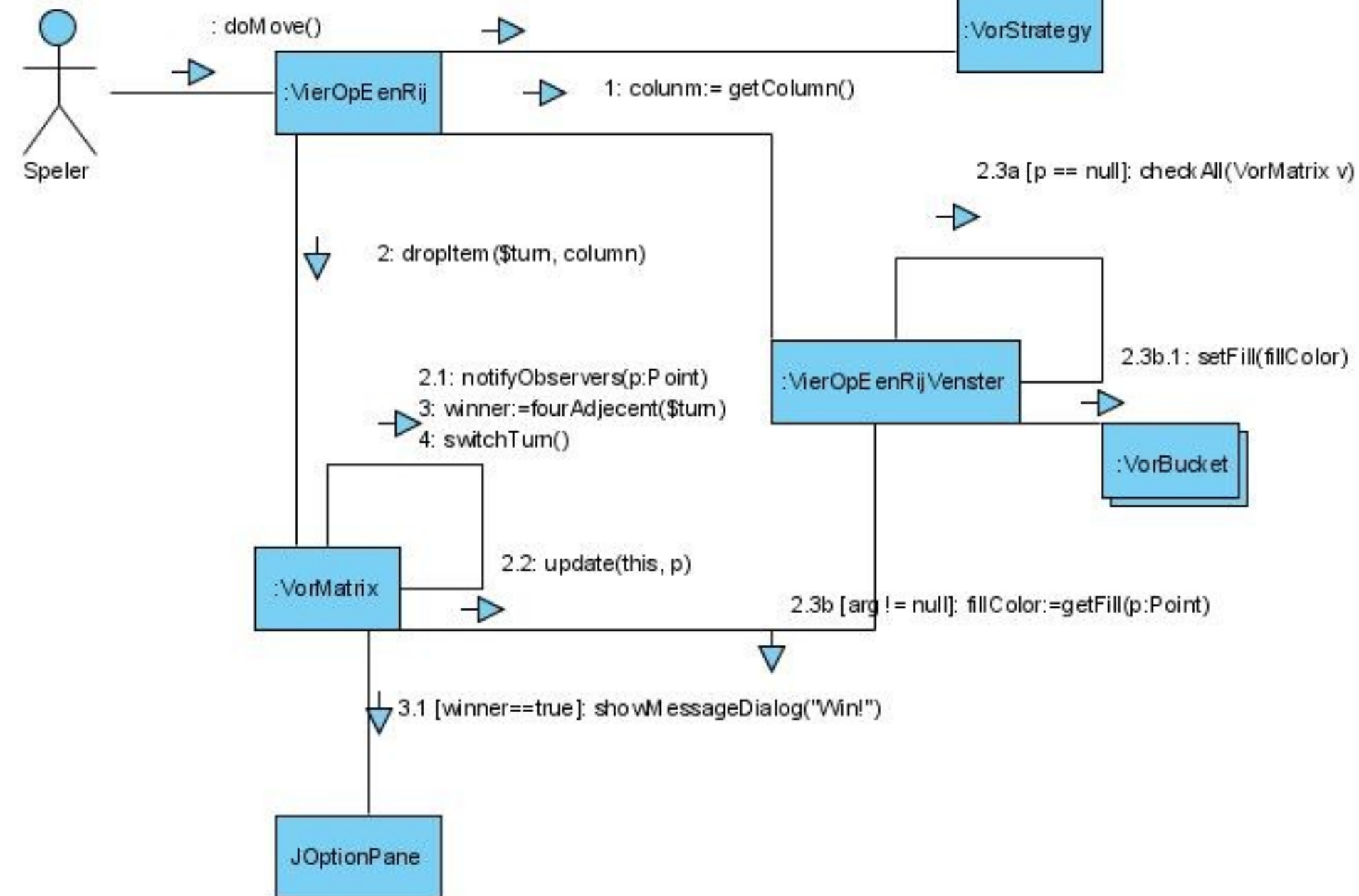
Vier op een Rij

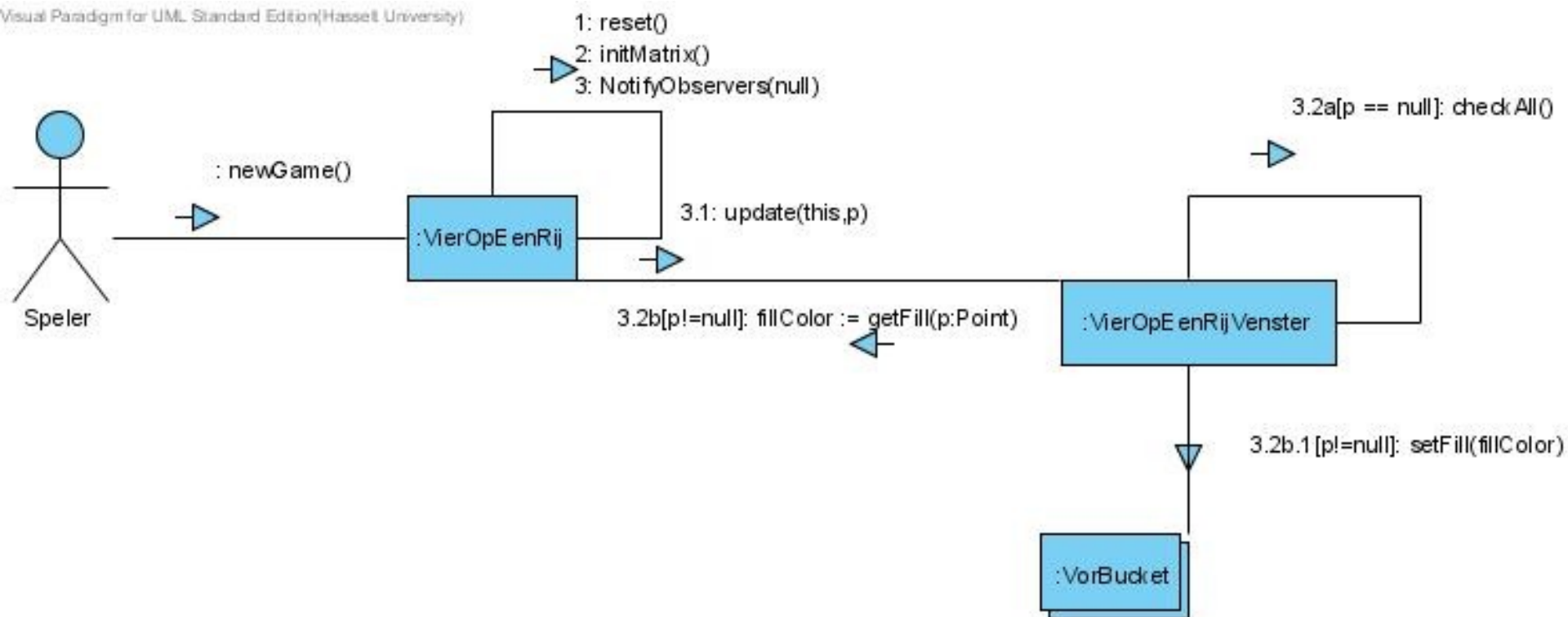
Vier op een Rij

- Twee spelers (geel en rood)
- Ieder om beurt kan een schijfje in een kolom laten vallen
- De eerste speler die 4 aaneensluitende schijfjes van dezelfde kleur heeft wint
 - Horizontaal, verticaal en diagonaal

1: start()

Gebruikte tool: <http://www.visual-paradigm.com/>

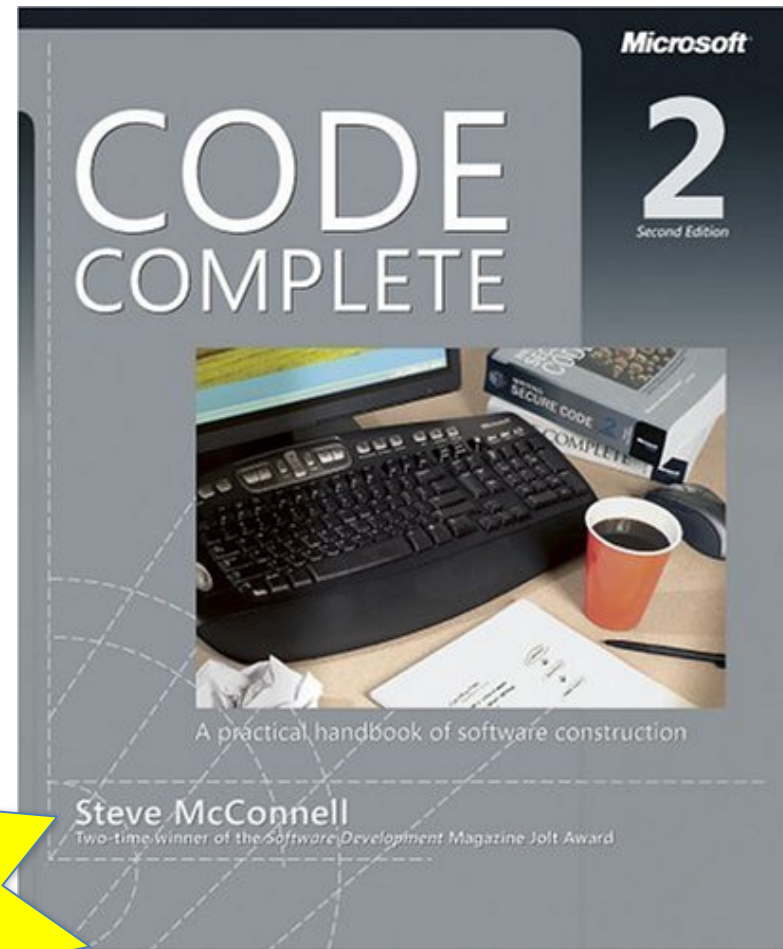




OO Mantra

Vuistregels


***Code Complete – A
Practical Handbook
of Software
Construction, 2nd
edition, Steve
McConnell***














Vuistregels

1. Geef voorkeur aan de maximale privacy die werkt voor je applicatie
private > protected > public
2. Alle methods die public zijn hebben eenzelfde niveau van abstractie
zie high cohesion
3. Data members zijn altijd verborgen
Toegang nodig? Voorzie get en set methods

Vuistregels

▼  VorMatrix :: Observable

-  VorMatrix()
-  checkColumns(FILL color) : boolean
-  checkDownDiagonals(FILL color) : boolean
-  checkRows(FILL color) : boolean
-  checkUpDiagonals(FILL color) : boolean
-  dropltem(FILL color, int column)
-  fourAdjacent(FILL color) : boolean
-  getFill(int column, int row) : FILL
-  initMatrix()
-  reset()
-  \$vorMatrix : FILL[][]

Vuistregels

4. Waar mogelijk, verwijder implementatie-details uit de interface

```
public class VierOpEenRij implements java.awt.event.MouseListener {  
    private static VierOpEenRij $main;  
    private VierOpEenRijVenster $vorVenster;  
    private VierOpEenRij.FILL $vorMatrix[][] = new VierOpEenRij.FILL[VierOpEenRij.ROWS][VierOpEenRij.COLS];  
    public static int ROWS = 6;  
    public static int COLS = 7;  
    public static enum FILL { RED , YELLOW , EMPTY };  
    private FILL $turn;  
    private VorStrategy $vsStrategy;
```

Vuistregels

4. Waar mogelijk, verwijder implementatie-details uit de interface

```
public class VierOpEenRij implements java.awt.event.MouseListener {  
    private static VierOpEenRij $main;  
    private VierOpEenRijVenster $vorVenster;  
    private VorMatrix $vorMatrix;  
    public static int ROWS = 6;  
    public static int COLS = 7;  
    public static enum FILL { RED , YELLOW , EMPTY };  
    private FILL $turn;  
    private VorStrategy $vsStrategy;
```

Vuistregels

5. Friend classes zijn helemaal niet zo vriendelijk
6. Een method hoort niet in de publieke interface omdat deze zelf publieke methods gebruikt
7. Leesbaarheid is belangrijker dan snelheid van coderen
8. De klasse moet bruikbaar zijn met enkel kennis van zijn publieke interface (niet door kennis over de implementatie)
9. Wees voorzichtig met een klasse die sterk verbonden is met een of meerdere andere klassen
zie low coupling

FOUT!

```
public class VorMatrix extends java.util.Observable{
    private VierOpEenRij.FILL $vorMatrix[][] = new
VierOpEenRij.FILL[VierOpEenRij.COLS][VierOpEenRij.ROWS];

    public VorMatrix(){ ... }
    public void initMatrix(){ ... }
    public void reset(){ ... }
    public void dropItem(VierOpEenRij.FILL color, int column) throws
        ColumnFullException{ ... }
    public VierOpEenRij.FILL getFill(int column, int row){ ... }
    public boolean fourAdjacent(VierOpEenRij.FILL color){ ... }
    public $vorMatrix[][] getDataModel() { ... }
    private boolean checkRows(VierOpEenRij.FILL color){ ... }
    private boolean checkColumns(VierOpEenRij.FILL color){ ... }
    private boolean checkDownDiagonals(VierOpEenRij.FILL color){ ... }
    private boolean checkUpDiagonals(VierOpEenRij.FILL color){ .... }
```

FOUT!



```
public class VorMatrix extends java.util.Observable{
    private VierOpEenRij.FILL $vorMatrix[][] = new
    VierOpEenRij.FILL[VierOpEenRij.COLS][VierOpEenRij.ROWS];

    public VorMatrix(){ ... }
    public void initMatrix(){ ... }
    public void reset(){ ... }
    public void dropItem(VierOpEenRij.FILL color, int column) throws
        ColumnFullException{ ... }
    public VierOpEenRij.FILL getFill(int column, int row){ ... }
    public boolean fourAdjacent(VierOpEenRij.FILL color){ ... }
    public $vorMatrix[][] getDataModel(){ ... }
    private boolean checkRows(VierOpEenRij.FILL color){ ... }
    private boolean checkColumns(VierOpEenRij.FILL color){ ... }
    private boolean checkDownDiagonals(VierOpEenRij.FILL color){ ... }
    private boolean checkUpDiagonals(VierOpEenRij.FILL color){ .... }
```


Beter

```
public class VorMatrix extends java.util.Observable{  
    private VierOpEenRij.FILL $vorMatrix[][] = new  
    VierOpEenRij.FILL[VierOpEenRij.COLS][VierOpEenRij.ROWS];
```

```
    public VorMatrix(){ ... }  
    private void initMatrix(){ ... }  
    public void reset(){ ... }  
    public void dropItem(VierOpEenRij.FILL color, int column) throws  
        ColumnFullException{ ... }  
    public VierOpEenRij.FILL getFill(int column, int row){ ... }  
    public boolean fourAdjacent(VierOpEenRij.FILL color){ ... }
```



```
    private boolean checkRows(VierOpEenRij.FILL color){ ... }  
    private boolean checkColumns(VierOpEenRij.FILL color){ ... }  
    private boolean checkDownDiagonals(VierOpEenRij.FILL color){ ... }  
    private boolean checkUpDiagonals(VierOpEenRij.FILL color){ .... }
```

OOP Mantra

1. Verkies associaties boven overerving.
2. Een method bevat niet meer dan 15 regels code.
3. Een method is ofwel een inspector, ofwel een mutator, maar nooit beide.
4. Elke method van enige betekenis heeft een contract

5. Geef voorkeur aan maximale privacy
(private > protected > public)
6. Streef naar high cohesion
7. Data members zijn altijd verborgen
8. Verwijder implementatie-details uit
de interface
9. Friend classes zijn niet vriendelijk

10. Een method hoort niet in de
publieke interface omdat deze zelf
publieke methods gebruikt
11. Leesbaarheid is belangrijker dan
snelheid
12. De klasse moet bruikbaar zijn met
enkel kennis van zijn publieke
interface
13. Streef naar Low Coupling

Defensief programmeren

Design by Contract

De **interface** van een klasse
is een ***contract*** tussen de
ontwikkelaar-aanbieder en
ontwikkelaar-gebruiker

Design van klassen is als...

...het opstellen van een contract met je collega software ontwikkelaar:

1. wat bied je aan (*public methods*)
2. wat garandeer je (*assertions*)
3. welke problemen moet je melden (*exceptions*)

Een contract, dat wil zeggen dat:

- Je **voor elke method**
 1. beschrijft in *welke omstandigheden* die mag opgeroepen worden: **pre-conditions**
 2. indien aan de pre-conditions voldaan is, beschrijft wat de method zal *afleveren of verwezenlijken*: **post-conditions**
 3. beschrijft welke *uitzonderlijke omstandigheden* er kunnen voorkomen: **exceptions**
- Realiseerbaar met OO programmeertaal + documentatie
- Alles wat in een contract staat, moet begrijpbaar zijn *zonder* kennis van de implementatie van de method!

Een contract, dat wil zeggen dat:

- Je **voor elke method**
 1. beschrijft in *welke omstandigheden* die mag opgeroepen worden: **pre-condities**
 2. indien aan de pre-condities voldaan is, beschrijft wat de method zal *afleveren of verwezenlijken*: **post-condities**
 3. beschrijft welke *uitzonderlijke omstandigheden* er kunnen voorkomen: **exceptions**
- Realiseerbaar met OO programmeertaal + documentatie
- Vanaf nu **Verplicht** voor elke method

```

/**
 * Drop a colored item in a given column
 * @param color the color of the dropped item
 * @param column the column where the item is dropped
 * @throws ColumnFullException when all items are already colored in this column
 * @pre color is not null, not Empty and in Fill
 * @pre column < vm.length
 * @pre there exists a row where: vm[column][$row] = FILL.EMPTY
 * @post vm[column][$row] == color, where vm[column][$row+1] = FILL.EMPTY or $row =
vm.height
 */

```

```

public void dropItem(VierOpEenRij.FILL color, int column) throws ColumnFullException{
    try{
        int row = 0;
        while($vorMatrix[column][row] != VierOpEenRij.FILL.EMPTY && row < VierOpEenRij.ROWS)
            row++;
        if($vorMatrix[column][row] != VierOpEenRij.FILL.EMPTY)
            throw new ColumnFullException(column);
        $vorMatrix[column][row] = color;
        this.setChanged();
        notifyObservers(new java.awt.Point(column,row));
    }catch(ArrayIndexOutOfBoundsException aiobe){
        throw new ColumnFullException(column);
    }
}

```

/**

* Drop a colored item in a given column

* @param color the color of the dropped item

* @param column the column where the item is dropped

* @throws ColumnFullException when all items are already colored in this column

* @pre color is not null, not Empty and in Fill

* @pre column < vm.length

~~* @pre there exists a row where: vm[column][\$row] = FILL.EMPTY~~

~~* @post vm[column][\$row] == color, where vm[column][\$row+1] = FILL.EMPTY or \$row = vm.height~~

*/

public void dropItem(VierOpEenhouding vm, int column) throws ColumnFullException{

try{

int row = 0;

while(\$vorMatrix[column]

row++;

if(\$vorMatrix[column]

throw new Column

\$vorMatrix[column][r

this.setChanged();

notifyObservers(new j

}catch(ArrayIndexOutOfBoundsException aobe){

throw new ColumnFullException(column);

}

}

gebruik public getter methods waar
mogelijk:
getFill(column,\$row) == color

Ontwerp van & interacties tussen klassen

Wat moet waar?

Welke methods
in
welke klassen?

Encapsulatie

Encapsulatie

==

Data + Methods

Encapsulatie in OO

==

Data + *Methods* + ***Access
Modifiers***

== Information
Hiding

G.R.A.S.P.

General Responsibility Assignment
Software Patterns

Doel

- **Verantwoordelijkheden** over klassen verdelen
- = **essentiële stap** tijdens ontwerp en coderen
- GRASP = **hulpmiddel** om dit op een *gestructureerde manier* te doen

Verantwoordelijkheden?

- **doen**

- iets zelf doen (een object aanmaken, iets berekenen)
- activiteiten uitvoeren en controleren/coördineren in andere objecten

- **weten**

- weet hebben van *private* data
- weet hebben van gerelateerde objecten
- weet hebben van dingen die berekend of afgeleid kunnen worden

GRASP

1. High Cohesion
2. Low Coupling
3. Information Expert
4. Creator
5. Controller





“heuristieken”!

GRASP

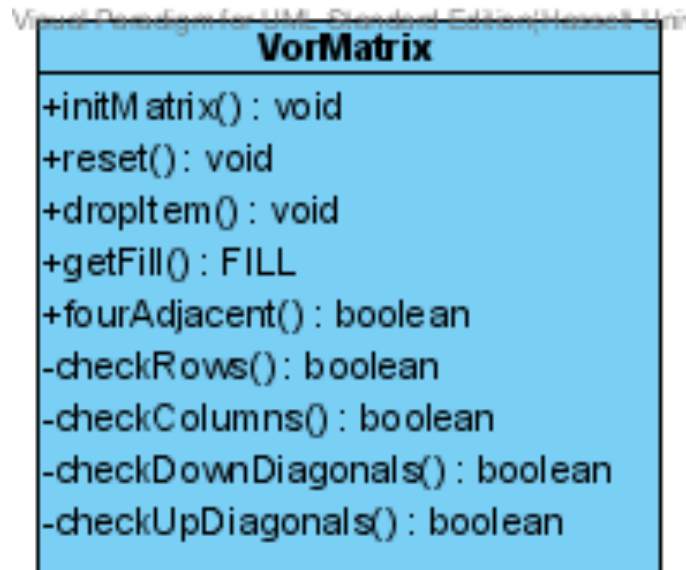
1. **High Cohesion**
2. Low Coupling
3. Information Expert
4. Creator
5. Controller

1. High Cohesion

- **Cohesion** geeft aan: *“hoe sterk gerelateerd en geconcentreerd de verantwoordelijkheden van een klasse zijn”*
- **High** cohesion
 - een klasse heeft sterk **gerelateerde** verantwoordelijkheden 
- **Low** cohesion
 - een klasse heeft sterk **uiteenlopende** verantwoordelijkheden 

1. High Cohesion



VorMatrix is verantwoordelijk voor alle functies met betrekking tot het bord en de spelregels die daarop gelden



GRASP

1. **High Cohesion**
2. **Low Coupling**
3. Information Expert
4. Creator
5. Controller

2. Low Coupling

- **Coupling** geeft aan: *“hoe sterk een klasse verbonden is met, kennis heeft van, of afhangt van andere klassen”*
- **Low** coupling 
 - een klasse hangt slechts af van **enkele andere klassen**
- **High** coupling 
 - een klasse hangt af van **vele andere klassen**.

Voorbeelden van “coupling”

Mogelijke koppelingen van klasse A naar klasse B:

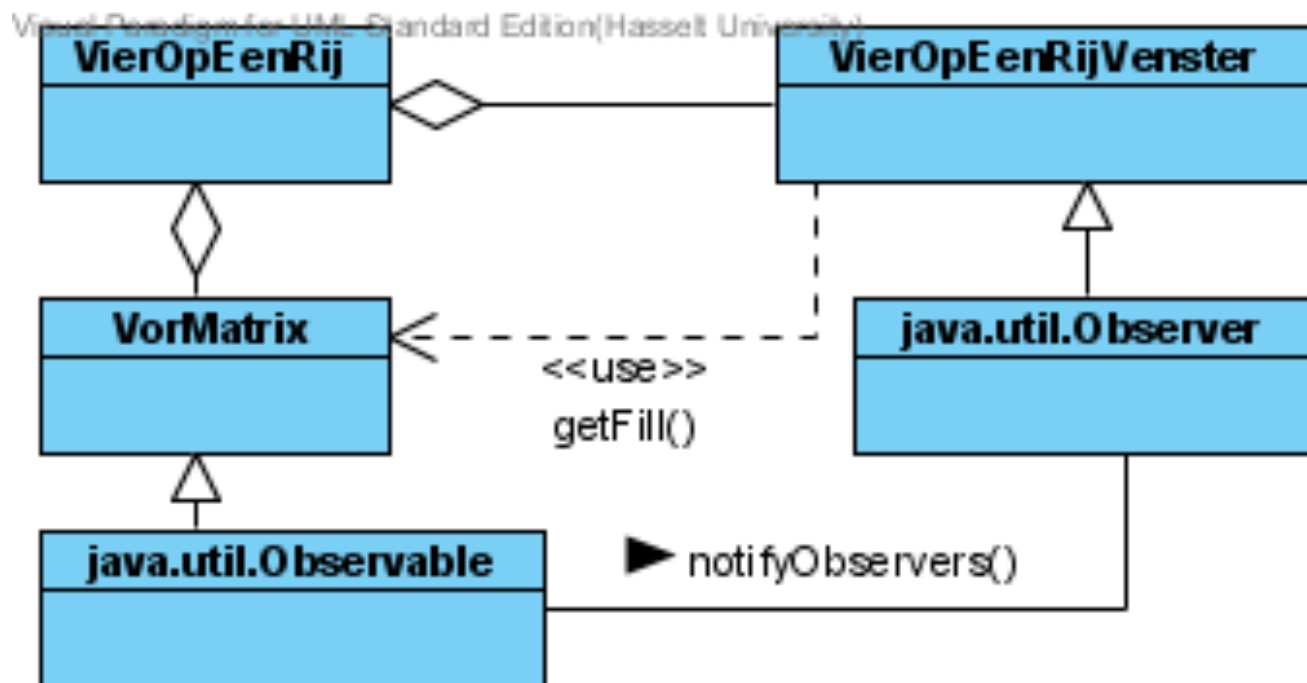
- A heeft een **member** die verwijst naar een object van klasse B
- een A object **roept methods aan** op een B object
- A heeft een **method die verwijst naar** een object van klasse B (bv. als parameter, lokale variabele, of return-waarde)
- A is een direct of indirect **afgeleide klasse** van B

High Coupling: probleem?

- veranderingen in gerelateerde klassen vereisen ook lokale wijzigingen
- klasse is moeilijker te begrijpen op zichzelf
- moeilijker te hergebruiken vanwege afhankelijkheid aan andere klassen

2. Low Coupling

Beperkte afhankelijkheid: **VorMatrix** moet niets van de GUI **VierOpEenRijVenster** weten, buiten het feit dat het een Observer is

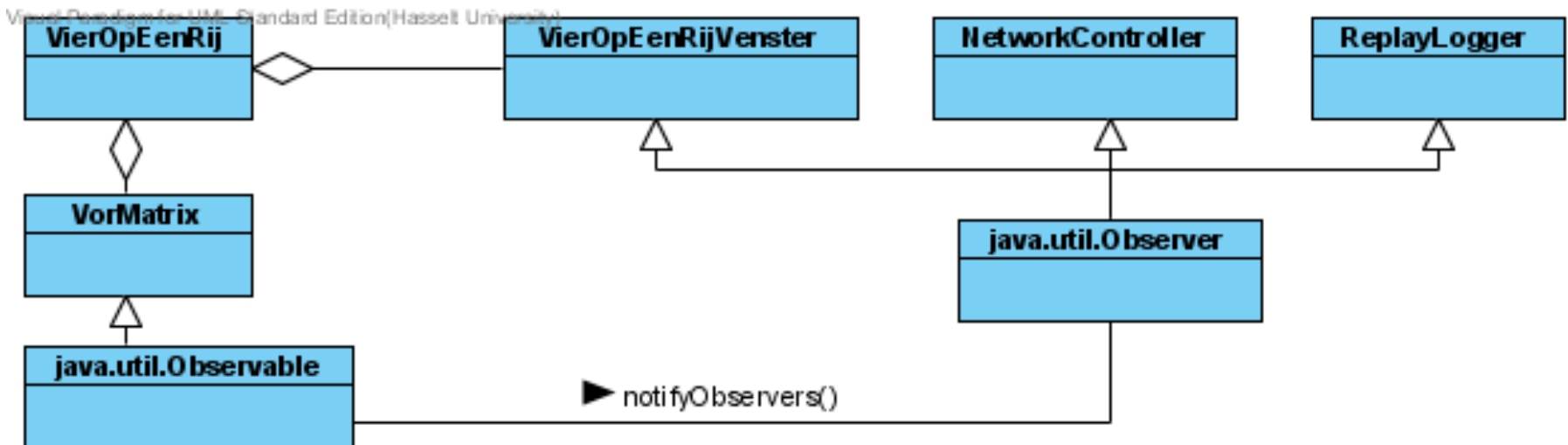


2. Low Coupling

Herbruikbaarheid/Uitbreidbaarheid

– Makkelijk uitbreidbaar met andere Observers, bv.

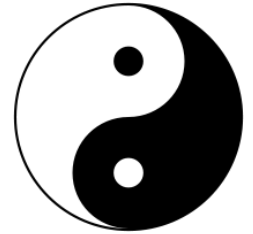
- Network Controller
- ReplayLogger
- ...



GRASP

1. **High Cohesion**
2. **Low Coupling**
3. **Information Expert**
4. **Creator**
5. **Controller**

*hangen vaak
samen*



*tenzij deze
problemen
veroorzaken met
HC & LC*

GRASP

1. High Cohesion
2. Low Coupling
3. **Information Expert**
4. Creator
5. Controller

3. Information Expert

- *“geef de verantwoordelijkheid aan de klasse die over de benodigde informatie beschikt”*
- Voordelen
 - *information hiding* blijft behouden
 - leidt meestal tot low coupling en high cohesion

GRASP

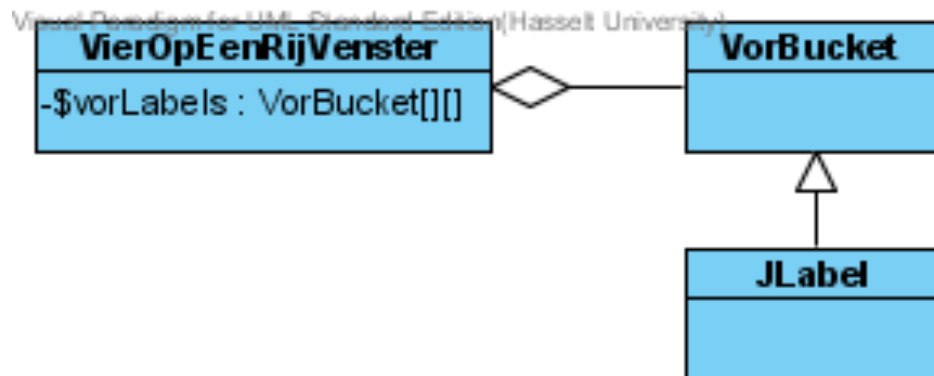
1. High Cohesion
2. Low Coupling
3. Information Expert
4. **Creator**
5. Controller

4. Creator

- *“geef klasse B de verantwoordelijkheid om objecten van klasse A aan te maken als één of meer van volgende uitspraken waar zijn:”*
 - B **aggregeert** A objecten
 - B **bevat** A objecten
 - B **maakt sterk gebruik van** A objecten
 - B **heeft initialisatiedata** om A objecten aan te maken (information expert m.b.t. aanmaak van A)
- als meerdere: verkies klasse die **aggregeert/bevat**

4. Creator

- Klasse ***VierOpEenRijVenster*** is verantwoordelijk voor het aanmaken van instanties van klasse ***VorBucket***
 - GUI klasse ***VierOpEenRijVenster*** bevat GUI elementen ***VorBucket***
 - ***VierOpEenRijVenster*** gebruikt ook veelvuldig objecten van type ***VorBucket***



GRASP

1. High Cohesion
2. Low Coupling
3. Information Expert
4. Creator
5. **Controller**

5. Controller

- *“geef de verantwoordelijkheid om ‘systeemgebeurtenissen’ af te handelen aan een klasse A”*
- Controller werkt coördinerend/controlerend
 - delegeert werk door naar andere objecten
 - zelf weinig verantwoordelijkheid
- ligt tussen user interface en applicatielogica

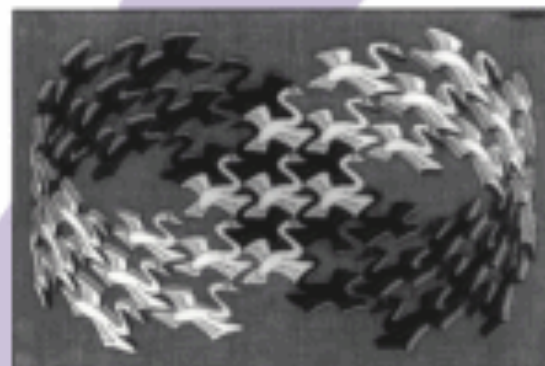


ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 MIT, Bucher / Costello Art - Boston - Holland. All rights reserved.

Foreword by Grady Booch

Definitie (naar *Christopher Alexander*, architect van gebouwen en steden & eredoctoer UHasselt):

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”



Vier essentiële onderdelen

1. de **naam** van het patroon,
2. het **probleem** waarop het patroon van toepassing is,
3. de **oplossing**: een beschrijving van de elementen van het ontwerp, hun relaties, hun verantwoordelijkheden en samenhang,
4. de **gevolgen**: meestal ruimte- en tijdsafwegingen, implementatiespecifieke en taalafhankelijke gevolgen, impact of flexibiliteit, uitbreidbaarheid en overdraagbaarheid;

Drie soorten

- Creational
- Structural
- Behavioral

Enkele basispatronen

- Singleton (creational pattern)
- Observer (behavioral pattern)
- Strategy (behavioral pattern)

Singleton Pattern

```
public class Singleton{
    private static Singleton $uniqueInstance = null;
    private static someType $singletonData;

    protected Singleton() { ... }

    public static Singleton instance() {
        if($uniqueInstance == null)
            $uniqueInstance = new Singleton();
        return $uniqueInstance;
    }

    public someType getSingletonData() {
        return $singletonData;
    }
}
```

Singleton
<u>-uniqueInstance: Singleton</u> <u>-singletonData: someType</u>
<u>+instance(): Singleton</u> <u>+someOperation()</u> <u>+getSingletonData(): someType</u>

Singleton Pattern

```
public class Singleton {  
    private static Singleton $uniqueInstance = null;  
    private static someType singletonData;
```

```
protected Singleton()
```

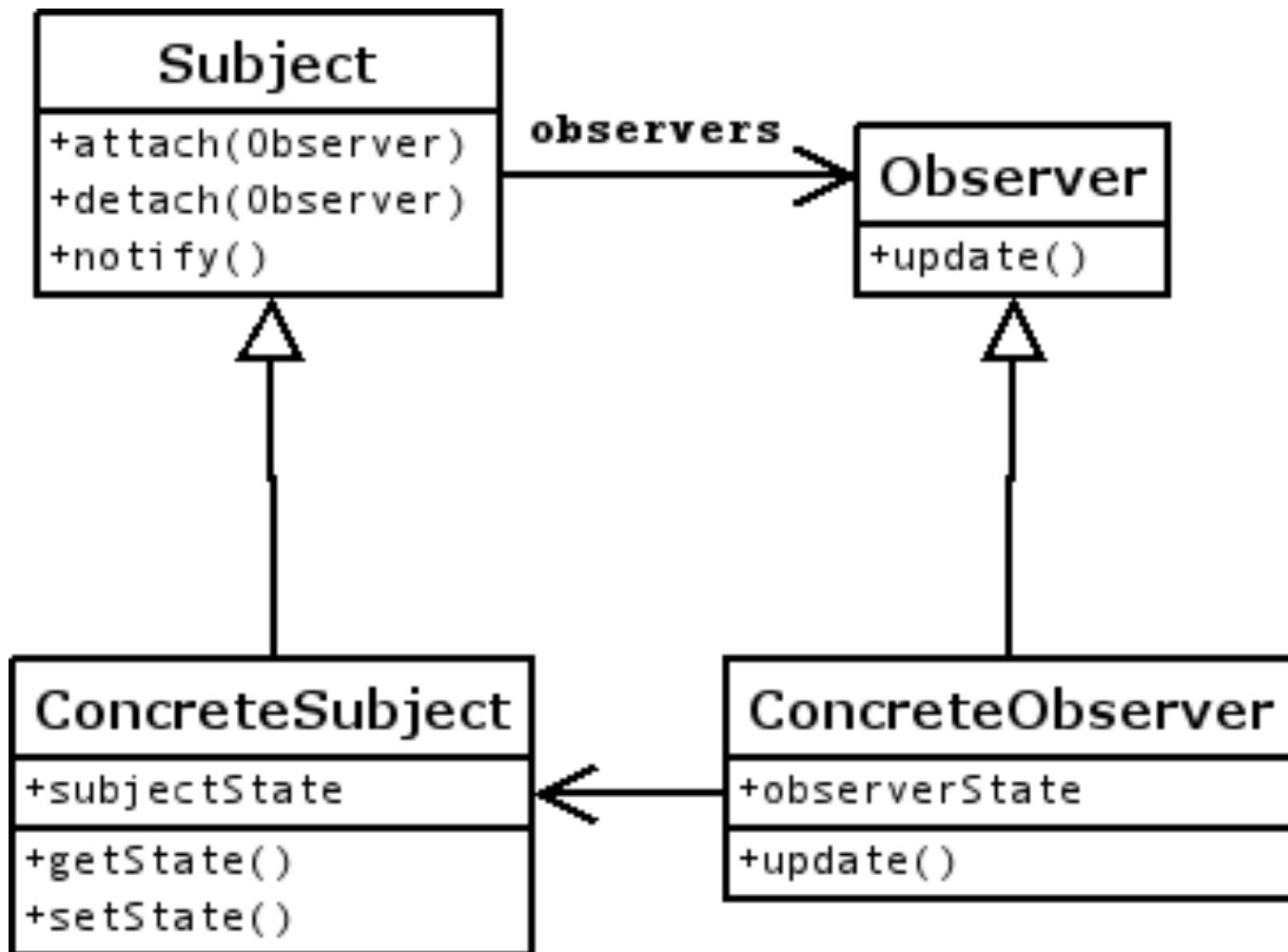
```
public static Singleton getInstance()  
{  
    if($uniqueInstance == null)  
        $uniqueInstance = new Singleton();  
    return $uniqueInstance;  
}
```

```
public someType getSingletonData() {  
    return $singletonData;  
}
```

**Enkel wanneer het
echt echt echt echt
echt nodig is**

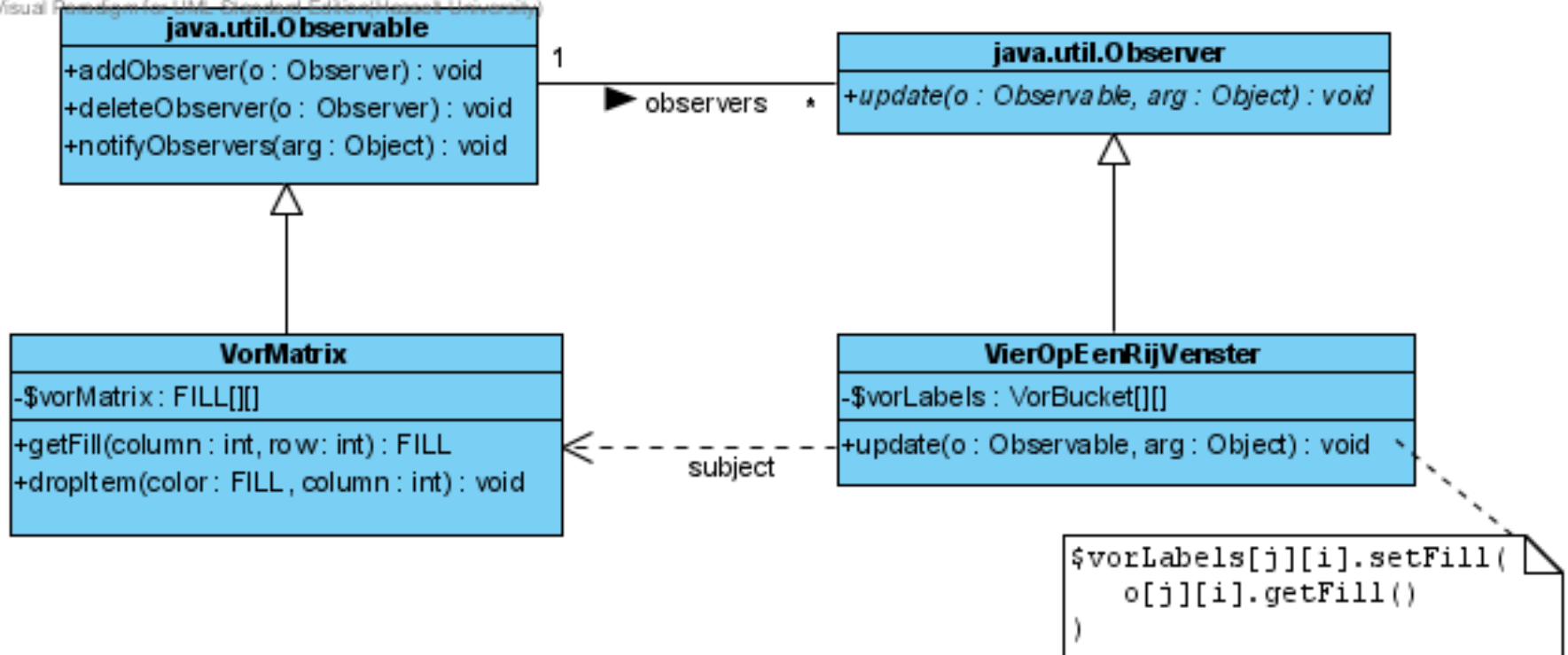
Singleton	
-uniqueInstance:	Singleton
-singletonData:	someType
+instance():	Singleton
+someOperation()	
+getSingletonData():	someType

Observer Pattern



Observer Pattern

Visual Design for UML - Student Edition (Hemel University)



Observer Pattern

```
public class ModelData
    extends Observable{

    private someType $data;

    public void registerObserver(
        Observer o){
        addObserver(o);
    }

    public void setData(someType data)
    {
        $data=data;
        notifyObservers();
    }
}
```

```
public class VisualRepresentation
    extends Canvas
    implements Observer{

    public void update(Observable o,
        Object arg){
        someType d =
            ((ModelData)o).getData;
        //redraw data in user interface
        ...
    }
}
```

Observer Pattern

```
ModelData I = new ModelData();  
VisualRepresentation vr = new VisualRepresentation();  
I.registerObserver(vr);
```

Observer Pattern

```
public class ModelData
    extends Observable{

    private someType $data;

    public void registerObserver(
        Observer o){
        addObserver(o);
    }

    public void setData(someType data)
    {
        $data=data;
        notifyObservers();
    }
}
```

```
public class VisualRepresentation
    extends Canvas
    implements Observer{

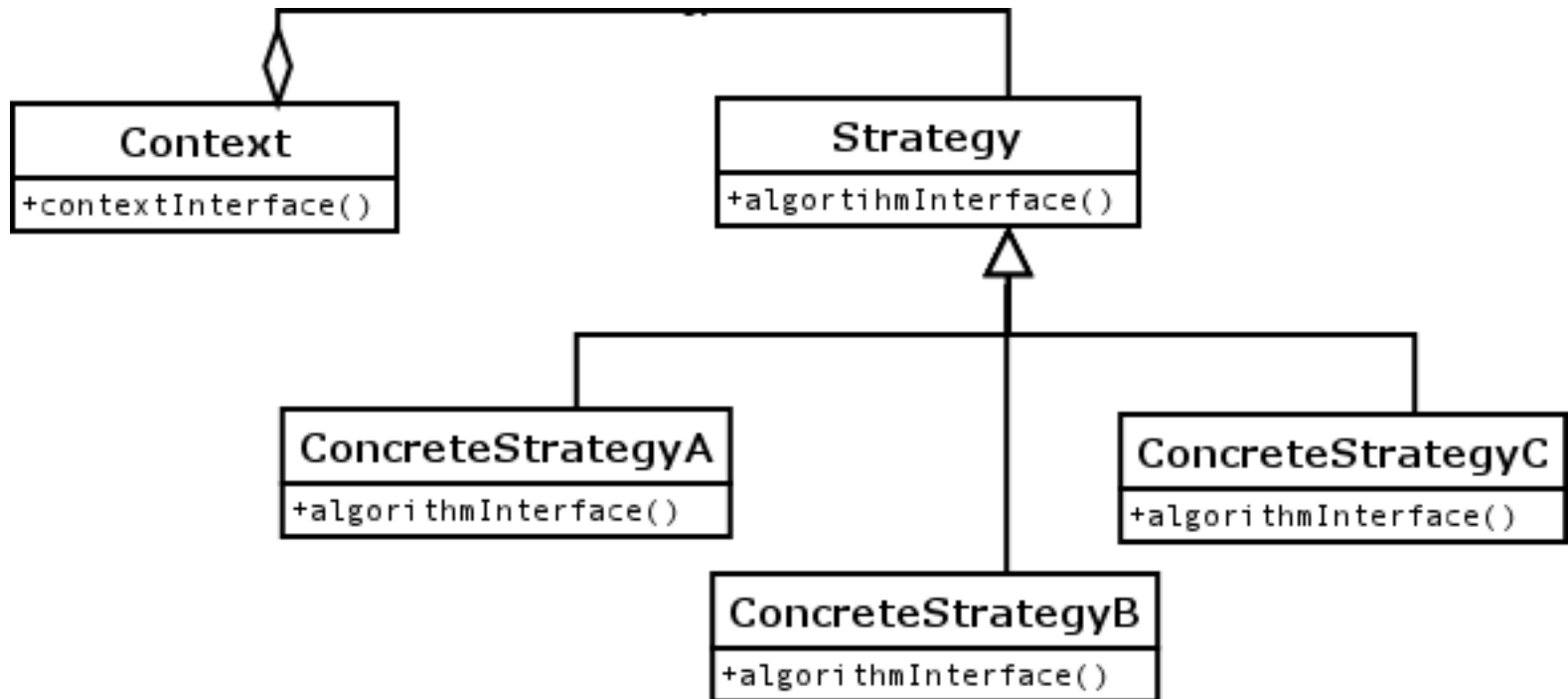
    public void update(Observable o,
        Object arg){

        someType o =
            ((ModelData)o).getData;
        //redraw data in user interface
        ...
    }
}
```

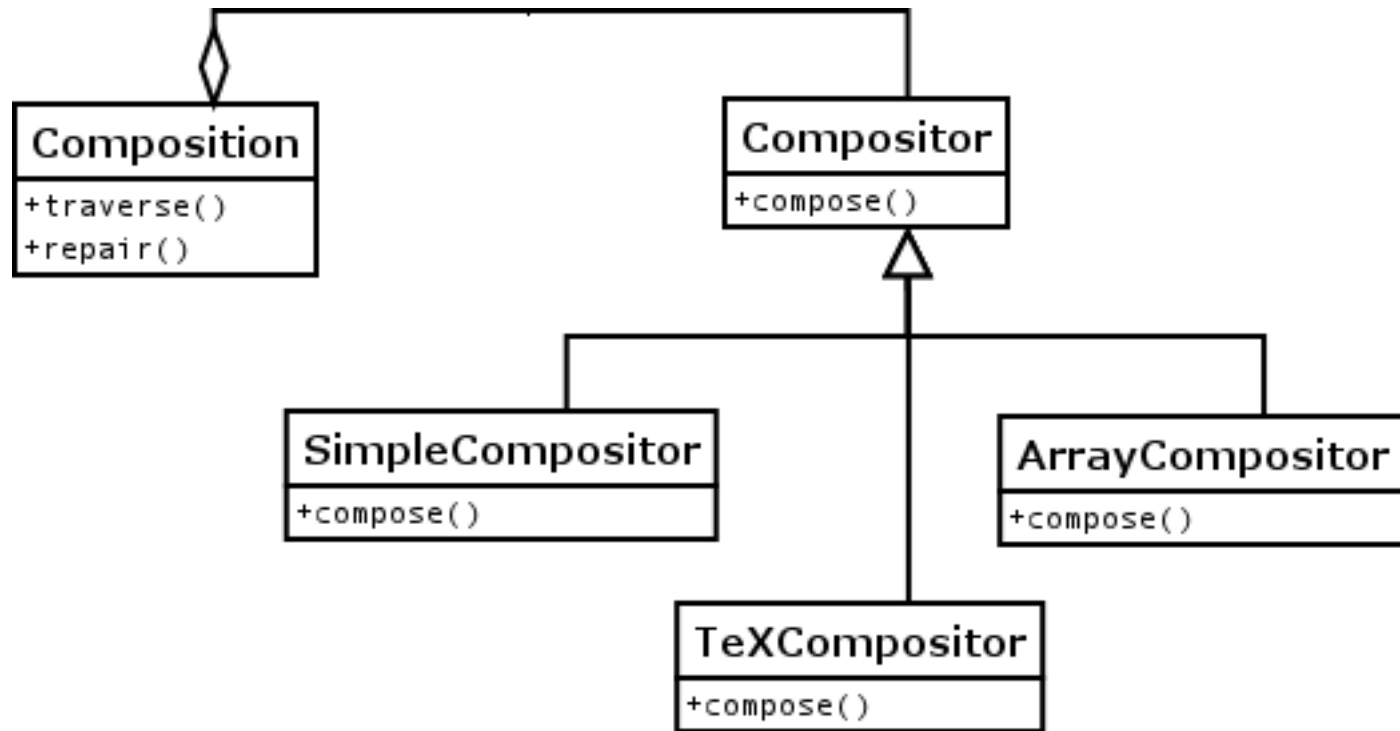


**Voor all Observers o
doe o.update(o,arg);**

Strategy Pattern



Strategy Pattern



Exception Handling

Exception Handling

Zelfstudie

Thinking in Java 3rd Edition, Chapter 9: Error
Handling with Exceptions

Oefeningen

Oefening 1

[Opdracht_2_1]..[Opdracht_2_5]

Oefeningen 1 tot en met 5 uit deel
over exceptions

Oefening 2

[Opdracht_2_6] Breid *Vier op een Rij* uit met een nieuw strategy object

class LoserStrategy implements VorStrategy

LoserStrategy zal enkel vier op een rij maken als er geen enkele andere mogelijkheid meer is.

[Opdracht_2_7] Pas de GRASP patronen toe bij deze implementatie. Je mag andere klassen maken indien je dat nodig vindt.

Oefening 3

[Opdracht_2_8] Voeg een tweede Observer toe voor de VorMatrix

public class LogGame

implements java.util.Observer

die bij elke verandering in het spel deze bij wegschrijft in een file (bij elke zet wordt het spelbord dus weggeschreven).