

# Informatica

2<sup>e</sup> semester: les 5

GUIs –  
lineaire datastructuren –  
computerarchitectuur

Jan Lemeire

**Informatica 2<sup>e</sup> semester**  
*februari – mei 2021*



Vrije Universiteit Brussel

# Vandaag

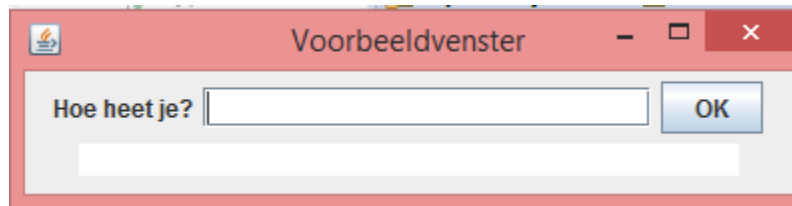
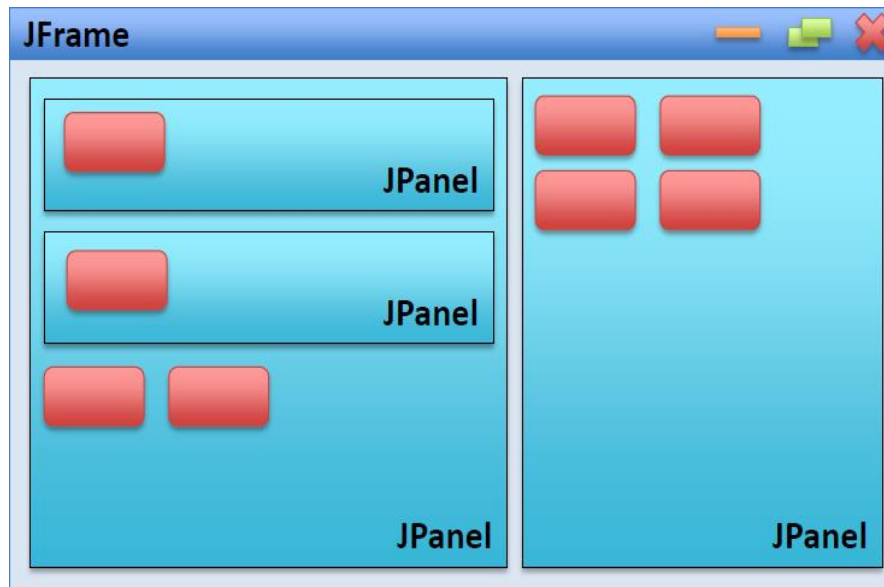
- 1. GUIs met Java (boek I)**
- 2. Klasse-oefening**
- 3. Stapel (hfst 3)**
- 4. Wachtrij (hfst 3)**
- 5. Interface-oefening**
- 6. Project**
- 7. Gelinkte lijst (hfst 6)**
- 8. Deel III/4: computerarchitectuur**

## **1.4 Graphical User Interface (GUI)**

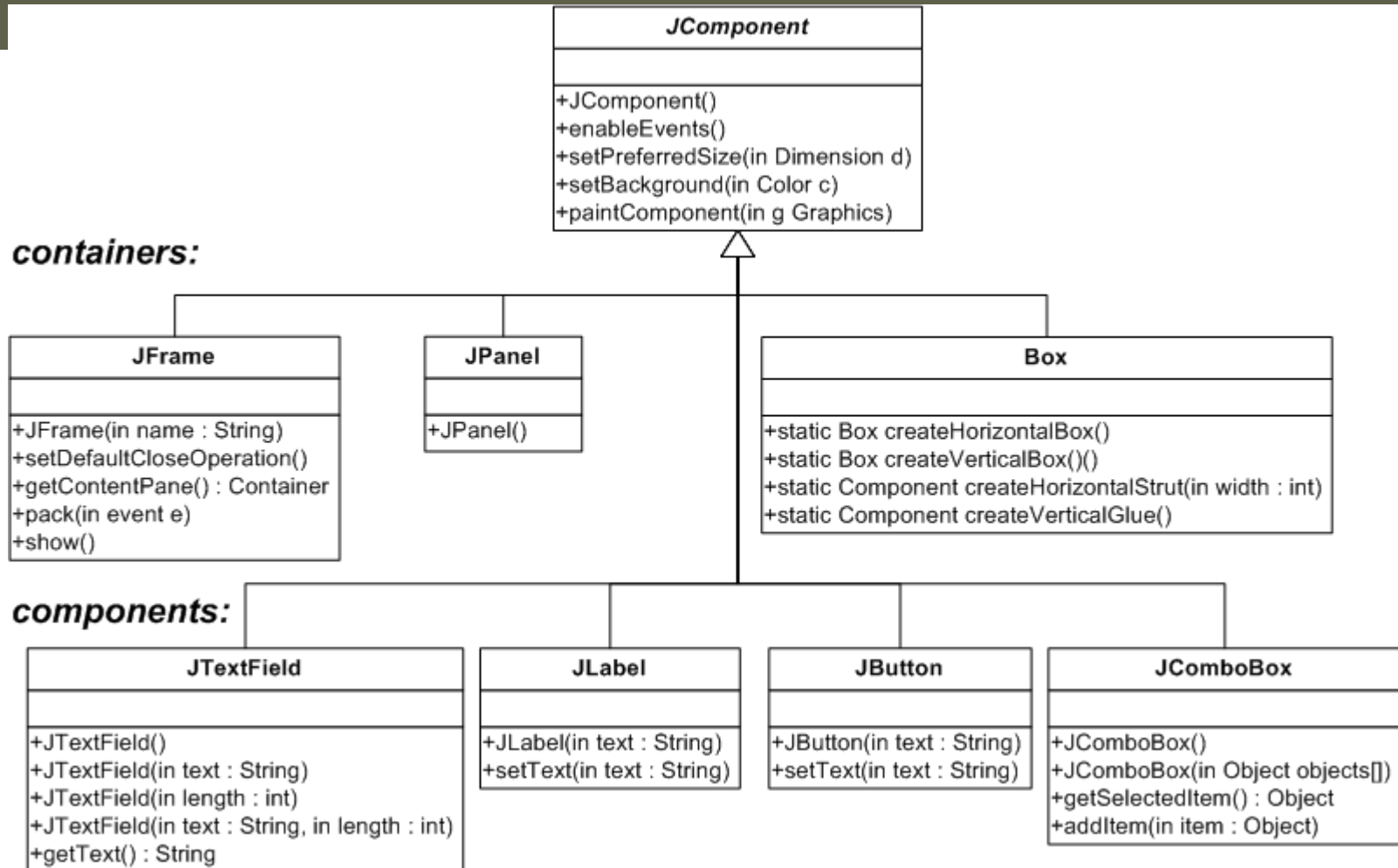
# GUI

- ◆ GUI = Graphical User Interface
- ◆ Gebruiken van de javaklassen van Swing
  - ✦ Online documentatie
- ◆ **Gebaseerd op de *kracht* van object-georiënteerde talen**
  - ✦ Encapsulatie
  - ✦ Overerving (Inheritance)
  - ✦ Abstractie/polymorphisme
    - interfaces

# GUI-componenten



# Swing's klassehiërarchie



```
import javax.swing.*;

public class MyPanel1 extends JPanel {

    // GUI-componenten
    private JTextField inputVeld;
    private JButton okKnop;
    private JTextArea outputVeld;

    public MyPanel1()
    {
        inputVeld = new JTextField(20); // groote veld (# symbolen)
        okKnop = new JButton("OK"); // inhoud knopje
        outputVeld = new JTextArea(1,30); // dimensies veld (rijen,
kolommen)

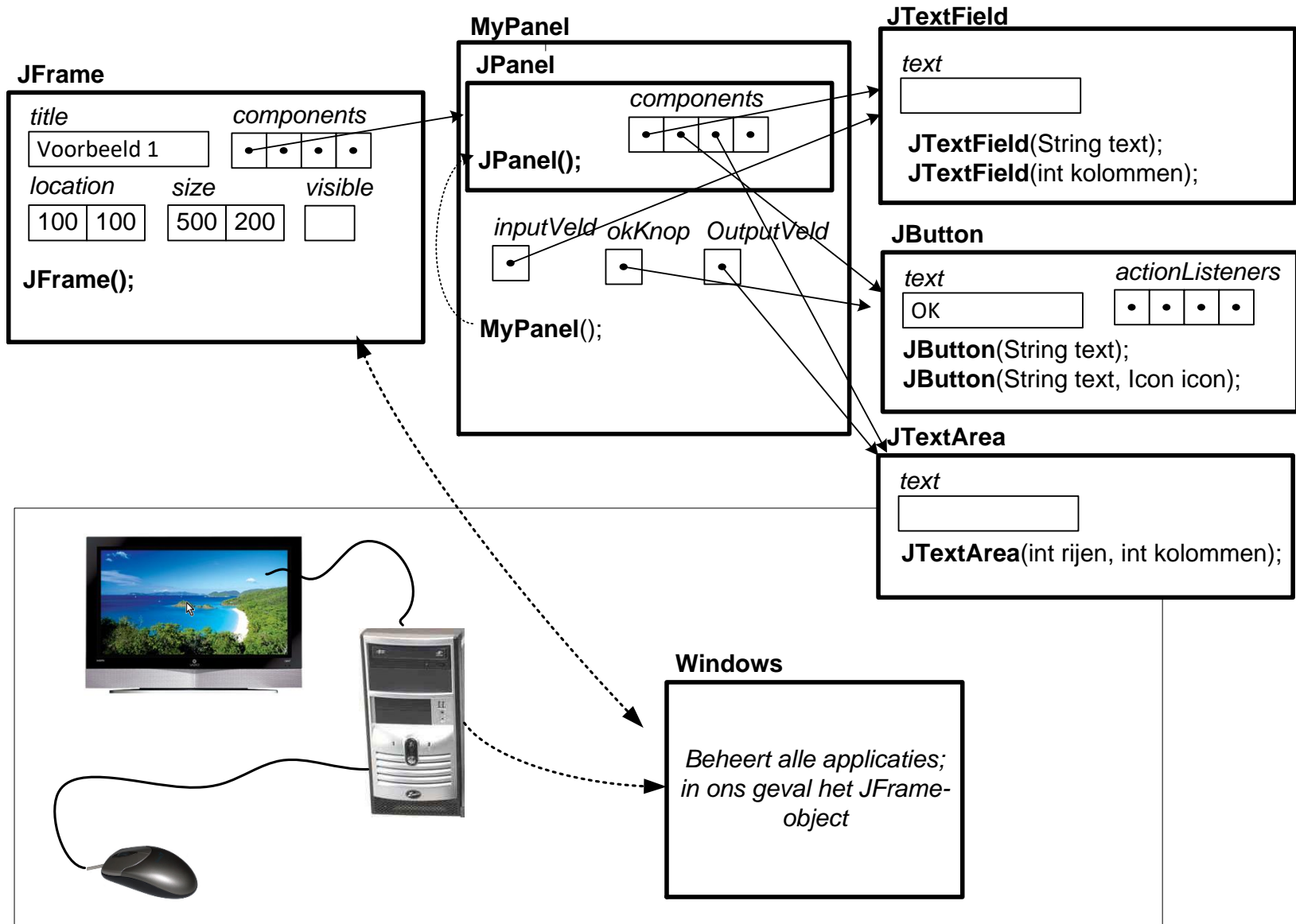
        // toevoegen van veldjes aan de GUI
        this.add(new JLabel("Hoe heet je?")); // JLabel: tekst op je GUI
        this.add(inputVeld);
        this.add(okKnop);
        this.add(outputVeld);
    }
}
```

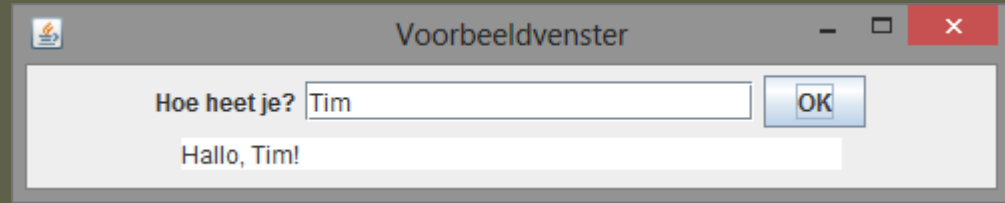
```
public static void main(String[] args)
{
    JFrame frame = new JFrame();
    frame.setSize(400, 100); // dimensies van het venster
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // zorgt dat
het programma afsluit bij het klikken op de 'X'-knop
    frame.setTitle("Voorbeeldvenster"); // titel van het venster
    frame.setLocation(100, 100); // positie van de linkerbovenhoek van
het venster op het bureaublad

    JPanel hoofdpaneel = new JPanel1(); // maak JPanel1-object aan
    frame.add(hoofdpaneel); // stop het in de Frame

    frame.setVisible(true); // het venster wordt geactiveerd
}
```







```
public class MyPanel2 extends JPanel implements ActionListener {

    // GUI-componenten
    private JTextField inputVeld;
    private JButton okKnop;
    private JTextArea outputVeld;

    public MyPanel2()
    {
        ...

        okKnop.addActionListener(this); // associatie ActionListener

        ...
    }

    // Dit wordt uitgevoerd wanneer er op de knop gedrukt wordt
    public void actionPerformed(ActionEvent e) {
        outputVeld.setText("Hallo, " + inputVeld.getText() + "!")
    }
}
```

# Interface ActionListener

```
interface ActionListener {  
    public void actionPerformed(ActionEvent e);  
}
```

## interface ActionListener

```
void actionPerformed(ActionEvent e);
```

## ActionEvent

*command*

>Kopieer>

*parameters*

ACTION\_PERFORMED...

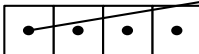
*oproepen  
actionPerformed() van  
alle actionListeners  
met ActionEvent*

## JFrame

*title*

Voorbeeld 1

*components*



*location*

100 100

*size*

500 200

*visible*

☐

JFrame();

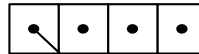
## MyPanel

JPanel

JPanel();

```
void actionPerformed(ActionEvent e)
{
  ...
}
```

*components*

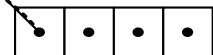


JButton

*text*

OK

*actionListeners*



void addActionListener();

*cursorpositie*

## Windows

*Bepaal component op  
positie (x, y) en bericht  
van event*

*Cursorbeweging,  
positie  
en kliks*

*muisklik*

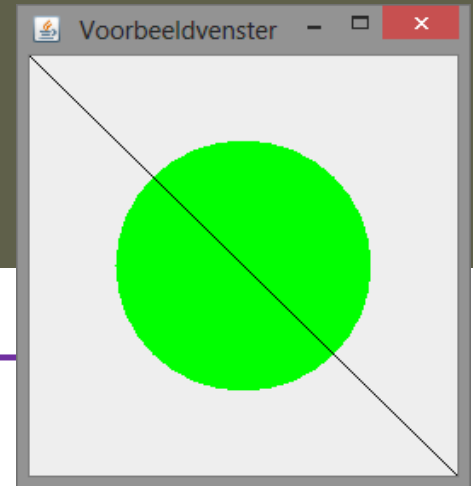
*Kliksignaal en beweging*

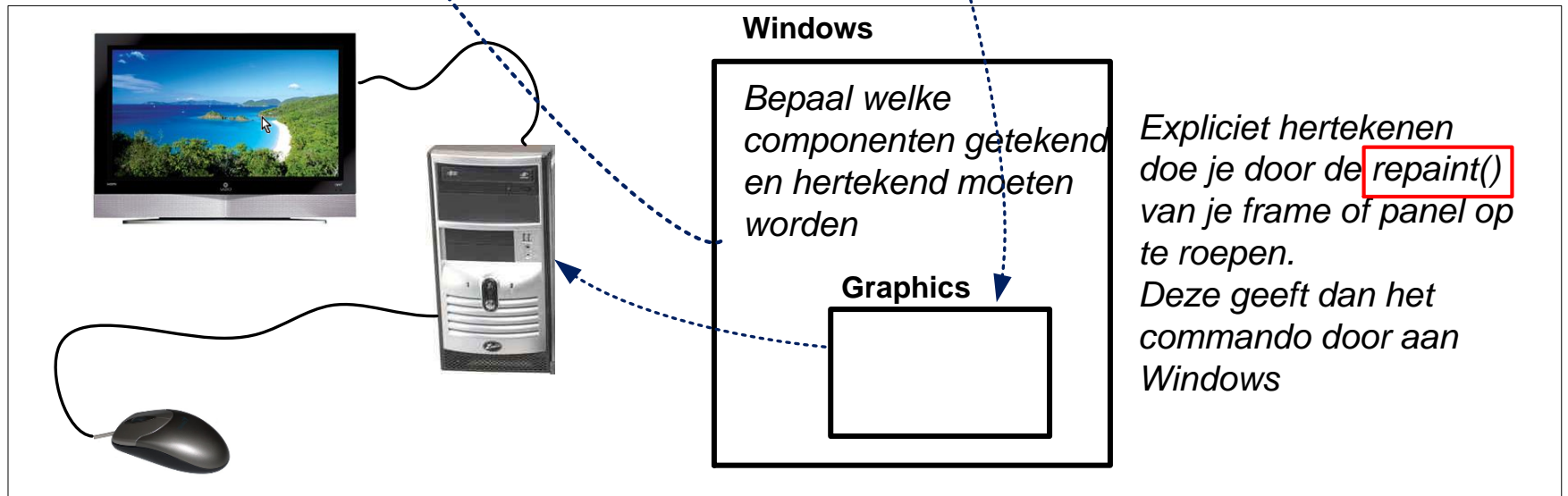
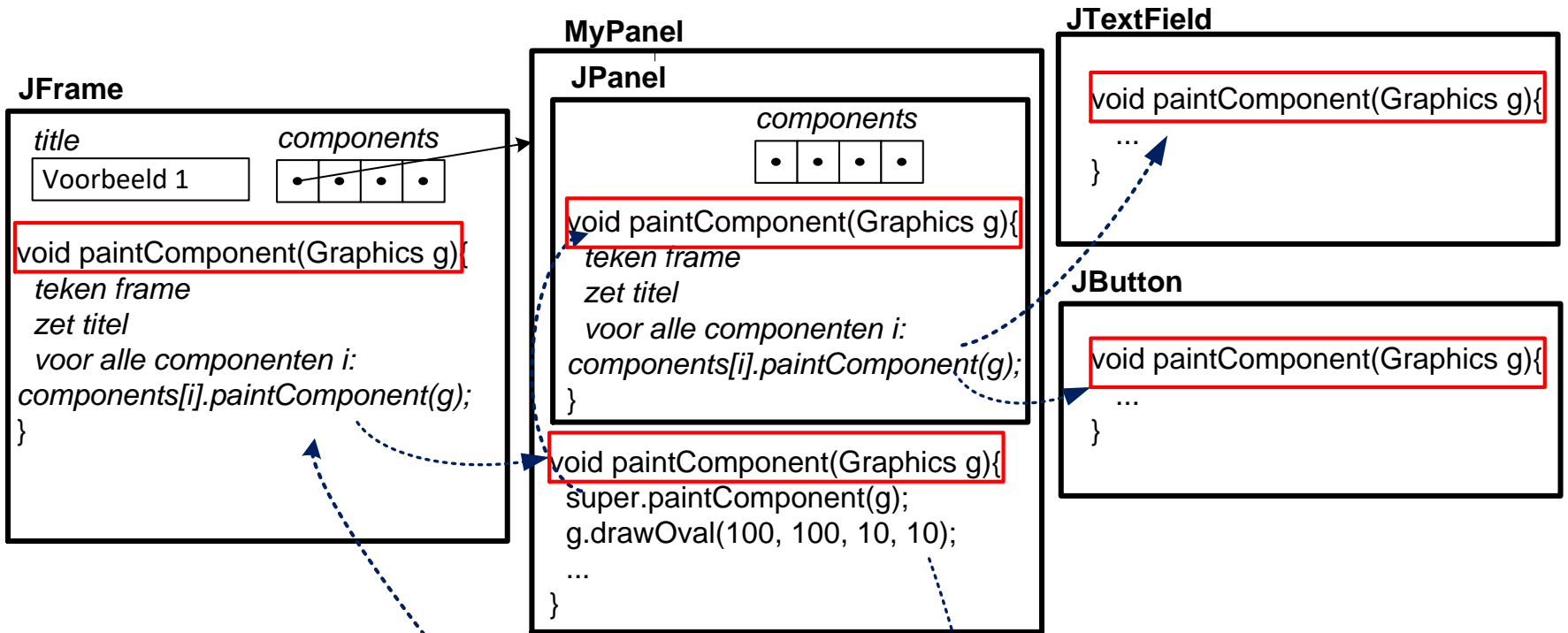


**Event handling achter de schermen van windows**

# Graphics

```
public class MyPanel3 extends JPanel {  
    public void paintComponent(Graphics g)  
    {  
        g.setColor(Color.GREEN);  
        g.fillOval(50,50,150,150);  
        g.setColor(Color.BLACK);  
        g.drawLine(0,0,250,250);  
    }  
  
    public static void main(String[] args)  
    {  
        ...  
    }  
}
```





**Je roept paintComponent(Graphics g) dus nooit zelf de op!**

Hertekenen doe je door repaint() op te roepen.

**Je tekent enkel binnen de paintComponent(Graphics g)!**

Je mag wel een andere methode oproepen en het Graphics-object doorgeven.

## Method Summary

<b>abstract void</b>	<a href="#"><u>clearRect</u></a> (int x, int y, int width, int height) Clears the specified rectangle by filling it with the background color of the current drawing surface.
<b>void</b>	<a href="#"><u>draw3DRect</u></a> (int x, int y, int width, int height, boolean raised) Draws a 3-D highlighted outline of the specified rectangle.
<b>abstract void</b>	<a href="#"><u>drawArc</u></a> (int x, int y, int width, int height, int startAngle, int arcAngle) Draws the outline of a circular or elliptical arc covering the specified rectangle.
<b>void</b>	<a href="#"><u>drawChars</u></a> (char[] data, int offset, int length, int x, int y) Draws the text given by the specified character array, using this graphics context's current font and color.
<b>abstract boolean</b>	<a href="#"><u>drawImage</u></a> ( <a href="#"><u>Image</u></a> img, int x, int y, <a href="#"><u>ImageObserver</u></a> observer) Draws as much of the specified image as is currently available.
<b>abstract void</b>	<a href="#"><u>drawLine</u></a> (int x1, int y1, int x2, int y2) Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
<b>abstract void</b>	<a href="#"><u>drawOval</u></a> (int x, int y, int width, int height) Draws the outline of an oval.
<b>abstract void</b>	<a href="#"><u>drawPolygon</u></a> (int[] xPoints, int[] yPoints, int nPoints) Draws a closed polygon defined by arrays of x and y coordinates.
<b>abstract void</b>	<a href="#"><u>drawPolyline</u></a> (int[] xPoints, int[] yPoints, int nPoints) Draws a sequence of connected lines defined by arrays of x and y coordinates.
<b>void</b>	<a href="#"><u>drawRect</u></a> (int x, int y, int width, int height) Draws the outline of the specified rectangle.
<b>abstract void</b>	<a href="#"><u>drawRoundRect</u></a> (int x, int y, int width, int height, int arcWidth, int arcHeight) Draws an outlined round-cornered rectangle using this graphics context's current color.



## 1.4.4. Animaties en Timers

### ⇒ Oefeningen (WPOs)

**EINDE BOEK I in de HOC**  
(wel nog 1 oefening volgende week)

# Klasse-oefening

4. Een **object** is een *instantiatie* van een klasse [`new`], waarbij elk object eigen waarden voor elk attribuut heeft. Bij het instantiëren wordt de constructor van de klasse uitgevoerd.
- De **default constructor** heeft geen parameters en een lege implementatie. Als er geen andere constructor bestaat, wordt de default constructor automatisch door Java aangemaakt.
  - Met `super (...)` op de *eerste lijn van je constructor* roep je één van de constructors van de superklasse op. Indien afwezig, wordt de default constructor opgeroepen. Deze moet dan wel bestaan voor de superklasse.
  - Op de *eerste lijn van een constructor* kan met `this (...)` een andere constructor van dezelfde klasse opgeroepen worden.

```

1. public class KlasseOefening2 {
2.     /** PROGRAMMA */
3.     public static void main(String[] args) {
4.         K11 o1 = new K11(5);
5.         K12 o2 = new K12(7);
6.         K13 o3 = new K13();
7.         o1.f();
8.         o2.f();
9.         o3.f();
10.    }
11. }
12. // welke klassen hebben de default constructor? .....
13. class K11 {
14.     int a;
15.     K11(){ // op welke lijn(en) wordt dit opgeroepen? .....
16.         this.a=5;
17.     }
18.     K11(int a){ // op welke lijn(en) wordt dit opgeroepen? .....
19.         this.a=a;
20.     }
21.     void f(){ // op welke lijn(en) wordt dit opgeroepen? .....
22.     }
23. }
24. class K12 extends K11 {
25.     K12(int x){ // op welke lijn(en) wordt dit opgeroepen? .....
26.         super(x);
27.     }
28.     void f(){ // op welke lijn(en) wordt dit opgeroepen? .....
29.     }
30. }
31. class K13 extends K11 {
32.     int b;
33. }

```

*Voer de code van de main() stap-voor-stap uit en geef aan welke code uitgevoerd wordt*

# Oefening interfaces

7. Een **abstracte methode** heeft *geen implementatie* (deel tussen accolades), enkel een header [abstract]. De eerste lijn eindigt met een punt-komma.
8. Een **interface** is een klasse met *enkel abstracte methodes* [interface].
  - Een interface heeft geen constructor en geen attributen (al zijn statische attributen wel mogelijk).
  - Een klasse mag meerdere interfaces als superklasse hebben [implements]: “de klasse implementeert de interface”.
  - Een concrete klasse moet alle abstracte methodes implementeren. Anders blijft het een abstracte klasse en kan je er geen objecten van maken.

# Oefening op interfaces

```
// a. Welke concrete klassen zijn correct (zijn echt concreet)?  
// b. Hoe maak ik ze concreet?  
// c. Welke methoden zou ik mogen weglaten opdat het toch nog concrete klassen blijven?
```

```
interface I1 {  
    void f();  
    void g(int x);  
}  
interface I2 {  
    int h();  
    int k(int a);  
}  
  
class C11 implements I1 {  
    public void f(){  
        ...  
    }  
    public void g(){  
        ...  
    }  
    public int h(){  
        ...  
    }  
}  
class C12 extends C11 implements I2 {  
    public void f(){  
        ...  
    }  
    public void g(int x){  
        ...  
    }  
}
```

# Stapel Stack

# De stapel of 'stack'

- ◆ Twee operaties: dingen opleggen en afnemen

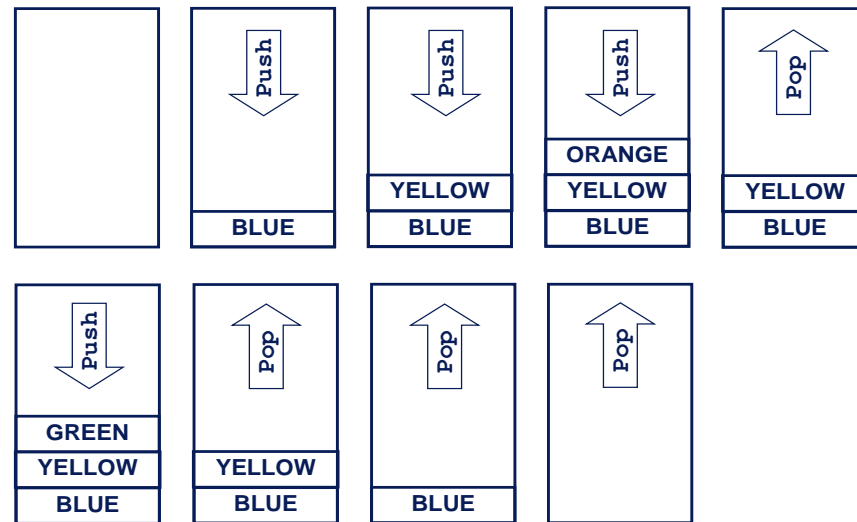


Fig.1.5. Last-in, first-out stack

- ◆ *last-in, first-out* (lifo) structuur

# 'interface' van stack

## ◆ Wat willen we er mee doen?

- ✦ Pop
- ✦ Push
- ✦ isEmpty
- ✦ isFull

## ◆ Headers van de methods (interface):

- ✦ `void push(int element)`
- ✦ `int pop()`
- ✦ `boolean isEmpty()`
- ✦ `boolean isFull()`



```
import java.nio.BufferOverflowException;
import java.nio.BufferUnderflowException;

public class Stack {
    private int[] data;
    private int pointer=0; // wijst naar het eerste vrije element

    public Stack(int capacity){
        data = new int[capacity];
    }

    public void push(int element){
        if (pointer == data.length) // check of vol
            throw new BufferOverflowException();
        data[pointer] = element;
        pointer++;
    }

    public int pop(){
        if (pointer == 0) // check of leeg
            throw new BufferUnderflowException();
        pointer--;
        return data[pointer];
    }

    public boolean isEmpty(){
        return pointer == 0;
    }

    public boolean isFull(){
        return pointer == data.length;
    }
}
```

# Vraagjes

- ◆ Je zou ook een pointer kunnen bijhouden naar het bovenste element van de stack, waarbij je met -1 aanduidt dat hij leeg is.
  - ✦ Wat verandert er aan de code?
- ◆ Voeg een *size()* methode toe die aangeeft hoeveel elementen er op de stack zijn.
- ◆ Nu werkt ie enkel met integers. Hoe generiek maken, t.t.z. bruikbaar voor elk type?
  - ✦ zie volgende datastructuur

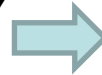
# Waarom niet gewoon een array gebruiken?

1. Code van stack komt overal in je programma terecht
2. Moeilijk om fouten met stack op te sporen
3. Je geeft niet aan wat je met de array wilt doen
4. Een collega-programmeur kan de array of pointer aanpassen
  - ➡ Consistentie om zeep helpen

Voorbeeld van het **Encapsulatieprincipe**

# Je programma wordt dan:

```
...  
int[] data = new int[capacity];  
int pointer=0;  
...
```



```
Stack stack = new Stack(3);
```

```
if (pointer == data.length)  
    throw new BufferOverflowException();  
data[pointer] = element;  
pointer++;
```



```
stack.push(element);
```

```
...  
if (pointer == 0)  
    throw new BufferUnderflowException();  
pointer--;  
element = data[pointer];  
...
```



```
element = stack.pop();
```

# Encapsulatie

# Pijlers van object-georiënteerde programmeertalen

p. 7

## I. Encapsulatie

- 2.3 ArrayList p. 11
- 3.1 Stapel-datastructuur p. 12
- 6.2 Java's LinkedList p. 55



## II. Overerving (inheritance)

- 1.1.2 Studentvoorbeeld p. 19
- 1.3 Vriendenvoorbeeld p. 36
- 1.4.1 MyPanel p. 41
- 1.4.3 PainComponent overschrijven p. 44
- 4.3 FunctieMetAfgeleide-interface p. 25

## III. Polymorfisme en abstractie

- 1.2.4 Set p. 32
- 1.2.5 Map p. 33
- 1.4.2 EventListener p. 43
- 1.6.4 Abstracte klassen p. 58
- 4.2 Functie-interface p. 21
- 5.2.2 Backtracking & Breadth-first p. 35
- Addendum bij hoofdstuk 5 (zie website, is optioneel)
  - Abstract zoekalgoritme
  - Vergelijking van zoekalgoritmes

# Encapsulatie (Data hiding)

- ◆ Data en operaties zitten samen (in het object)
- ◆ Data maak je niet toegankelijk voor de buitenwereld, enkel operaties
  - ✦ Daarom *get()* en *set()*-methodes voor attributen
- ◆ De maker van de klasse heeft controle over wat de gebruiker met de data doet
- ◆ Als gebruiker weet je dat de maker ervoor gezorgd heeft de consistentie bewaard blijft
  - ✦ “je hoeft niet bang te zijn iets verkeerd te doen”

# Wachtrij

# Queue



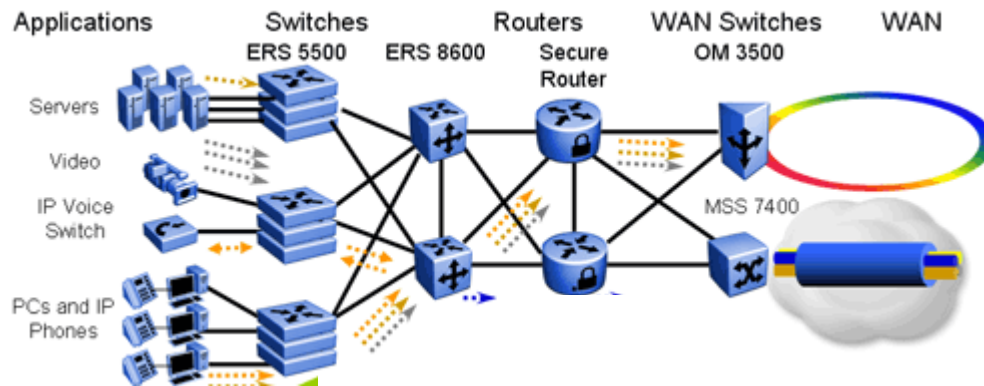
# Wachtrij of Fifo-queue



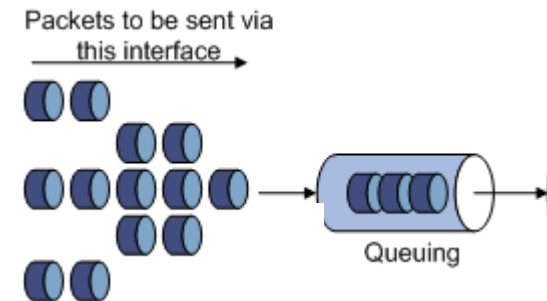
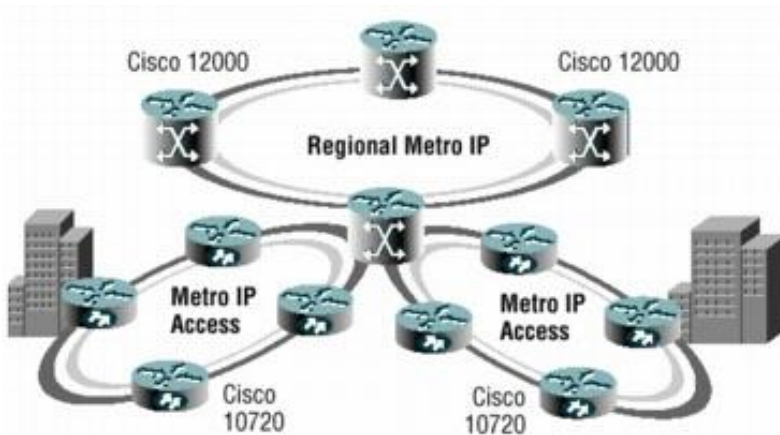
"What would the lady at the back like?"

◆ *Fifo = First-in, first out*

# Internetknopen



- ◆ **Switch:** knooppunt
- ◆ **Router:** bepaalt ook route van pakketje



Als queue vol: **packetdrop**

- Verlies van gegevens
- Moeten opnieuw verstuurd worden

# Fifo-queue

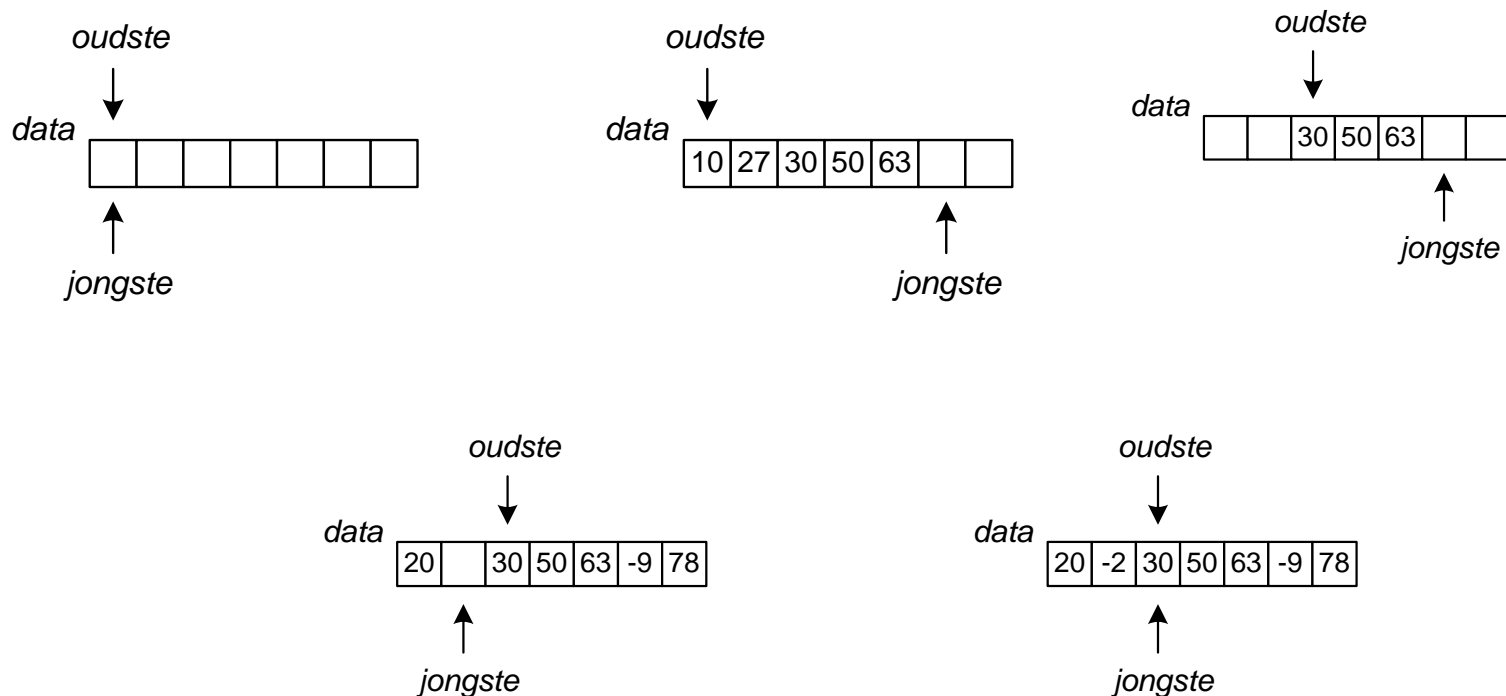
## ◆ Basisoperaties:

- ✦ add()

- ✦ get()

# Implementatie met array

## ◆ 2 pointers nodig



## ◆ en ... boolean voor te weten of vol/leeg

```

public class FIFOQueue<T>{
    private T[] data;
    private int oudste=0; // pointer naar oudste element (als niet leeg)
    private int jongste=0; // pointer naar de plaats voor het volgend element
    boolean full = false; // als queue vol is
    public FIFOQueue(int capacity){
        data = (T[]) new Object[capacity];
    }
    public void add(T waarde){
        if (full)
            throw new BufferOverflowException();
        data[jongste] = waarde;
        jongste++;
        if (jongste == data.length)
            jongste = 0;
        if (jongste == oudste)
            full = true;
    }
    public T get(){
        if (isEmpty())
            throw new BufferUnderflowException();
        T waarde = data[oudste];
        data[oudste] = null;
        oudste++;
        if (oudste == data.length)
            oudste = 0;
        if (full)
            full = false;
    }
}

```

```

public boolean isEmpty(){
    return !full && jongste == oudste;
}
public boolean isFull(){
    return full;
}

```

# Priorityqueue

- ◆ Een wachtrij waarbij niet het oudste element wordt teruggegeven, maar het element met de hoogste prioriteit
- ◆ Java heeft hiervoor een klasse `PriorityQueue<E>` in zijn bibliotheek.
  - ✦ De `poll()`-methode neemt het element met de hoogste prioriteit.
- ◆ De prioriteit van de elementen is gebaseerd op een orderrelatie die gedefinieerd is op de elementen.
  - ✦ Zoals we gaan zien in 7.2 en 8.5 kan dit in Java via de natuurlijke ordening of via een specifieke ordening.

**Project**

# Projectonderwerp

- ◆ Voldoende programmatorische complexiteit
  - ✦ Klassen & objecten
- ◆ Spelelement wordt meestal gekozen
  - ✦ Creativiteit!
- ◆ Mag:
  - ✦ Artificiële Intelligentie (A.I.): 'slim' algoritme
    - Hoofdstuk 5

***Groepjes van 2 - 3***



# Project: doelstellingen

- ◆ Programmeren
- ◆ Programmatorische complexiteit, gestructureerd programmeren
- ◆ Projectwerk, teamwork
- ◆ Creativiteit, ingenieuziteit
- ◆ Plezier
- ◆ Ontwikkelen probleemoplossende vaardigheden

# Project: uitvoering

- ◆ Voor paasvakantie: keuze groep & onderwerp



# Evaluatie

- ◆ Programmatorische complexiteit
  - ◆ Correctheid
  - ◆ Logische opdeling van code in klassen
  - ◆ Gestructureerd programmeren
    - ✦ 1 regel: **geen redundantie**
  - ◆ 'Properheid' van code:
    - ✦ Gebruik van zinvolle namen voor variabelen en functies
    - ✦ Logische structuur van code
    - ✦ documenteer vooral niet-triviale dingen
  - ◆ Creativiteit
  - ◆ Mondelinge verdediging
  - ◆ Bonus voor excellentie, als het 'af' is
- Constantes
  - Functies/methodes
    - Parameterizatie

# Laatste tips

- ◆ Ingenieur = efficiëntie
- ◆ Anderzijds: excellentie vergt net dat tikkeltje extra
- ◆ Probleemoplossende vaardigheden (debuggen)
- ◆ Als je probleem niet kan oplossen: VRAAG
  - ✦ Blijf er niet bij zitten!!
- ◆ Referenties en gebruikte code opgeven!
  - ✦ **Anders: plagiaat**

# Twee voorstellen

## ◆ Denkspel met Booleaanse functies

✦ Zoals Circuit Scramble (niet meer te downloaden...)

<https://apkpure.com/nl/circuit-scramble-computer-logic-puzzles/com.Suborbital.CircuitScramble#com.Suborbital.CircuitScramble-2>

## ◆ Engine waarbij speler het gedrag van de personages moet programmeren

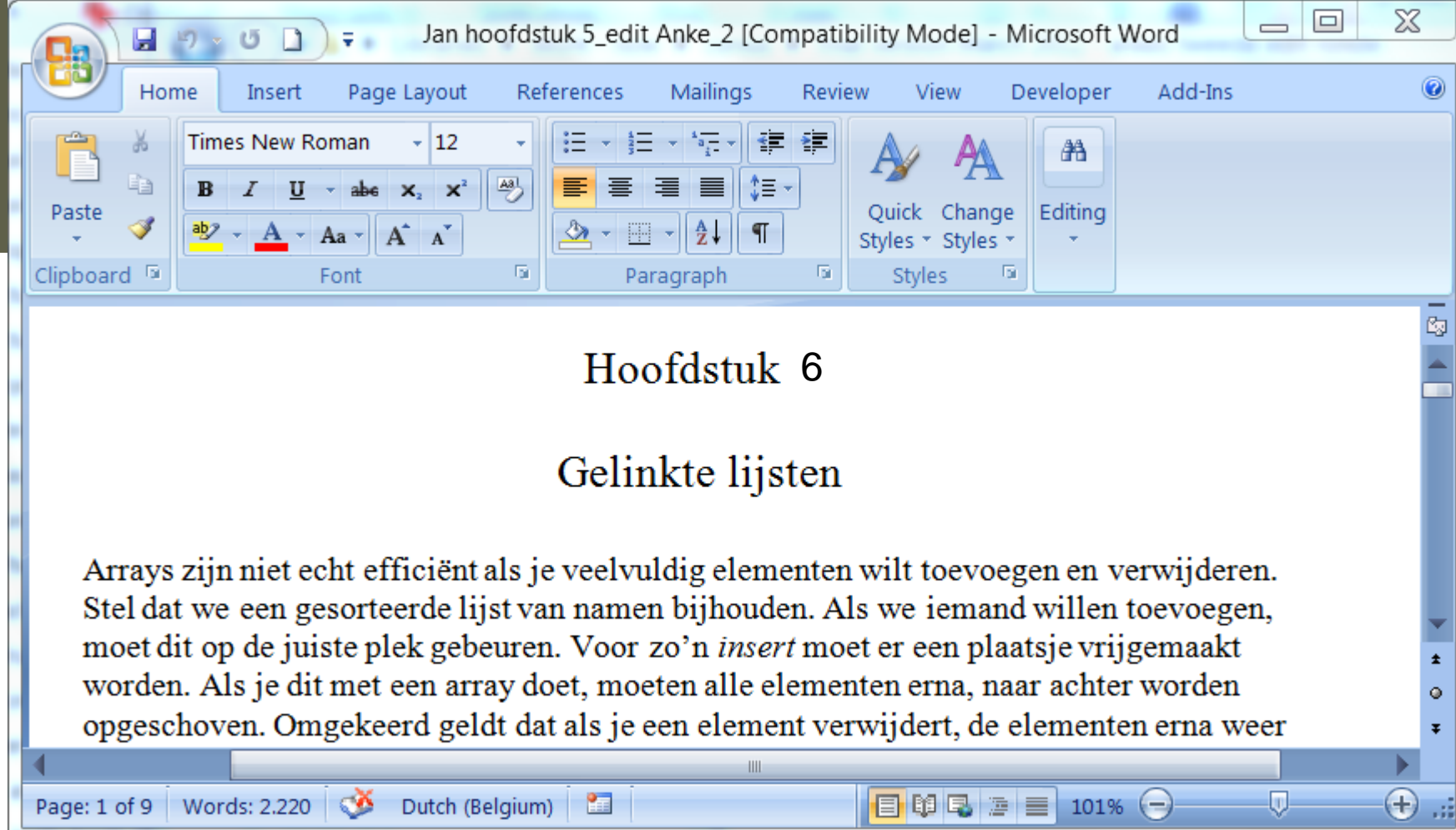
Zoals PACS, ons spel gebaseerd op PACMAN

<http://parallel.vub.ac.be/education/modula2/projectII/index.html>

*Deze games wil ik graag mee begeleiden*

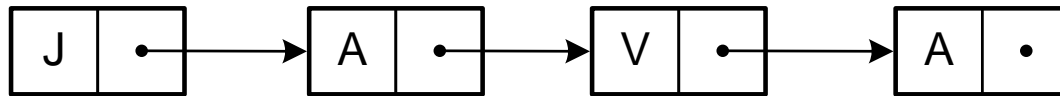
# **Gelinkte lijst**

## **Linked list**



# Linked List

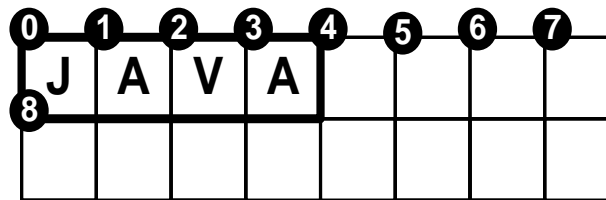
◆ Flexibel toevoegen en verwijderen



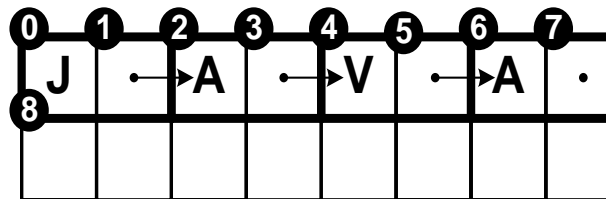


# Geheugengebruik

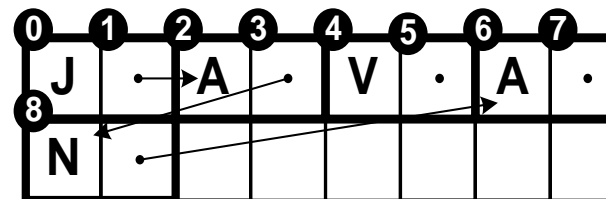
## ◆ Array



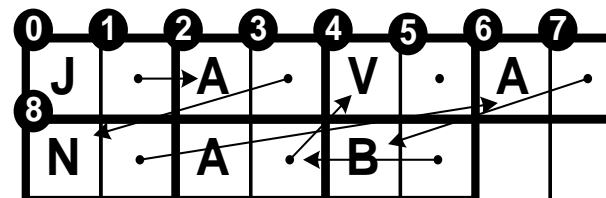
## ◆ Linked List



*Geordend in 't begin*



*Door elkaar na  
enkele veranderingen*



```

public class LinkedList <T> {
    class Link<T>{
        T data;
        Link<T> next;

        Link(T data){
            this.data = data;
            this.next = null;
        }
        Link(T data, Link<T> next){
            this.data = data;
            this.next = next;
        }
    }

    Link<T> first;
    public LinkedList(){
        first = null;
    }
}

```

# Java's LinkedList

## Constructor Summary

[LinkedList](#)()

Constructs an empty list.

[LinkedList](#)([Collection](#)<? extends [E](#)> c)

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

## Method Summary

boolean [add](#)([E](#) e)

Appends the specified element to the end of this list.

# Toevoegen van een bepaald element aan het einde

```
public void append(T data) {  
    Link<T> el = new Link<T>(data);  
    if (first == null) {  
        first = el;  
    } else {  
        // zoek laatste link  
        Link<T> last = first;  
        while (last.next != null)  
            last = last.next;  
        last.next = el;  
    }  
}
```

# Printen van lijst

```
public String toString(){
    Link<T> link = first;
    String str = "[";
    boolean eerste = true;
    while (link != null){
        if (eerste)
            eerste = false;
        else
            str+=", ";
        str += link.data;
        link = link.next; // ga naar volgende
    }
    str+="]";
    return str;
}
```

# Vinden van een bepaald element

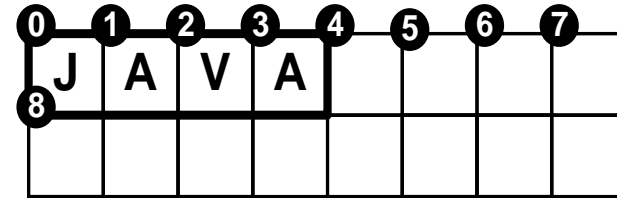
```
public T get(int index) {  
    int i=0;  
    Link<T> link = first;  
    while (link != null && i < index) {  
        link = link.next;  
        i++;  
    }  
    if (link == null)  
        return null;  
    else  
        return link.data;  
}
```

- ◆ *Nadeel*: heel de lijst moet doorlopen worden
- ◆ Bij arrays kan je onmiddellijk naar de index springen!

# Vinden van het $i^{\text{de}}$ element

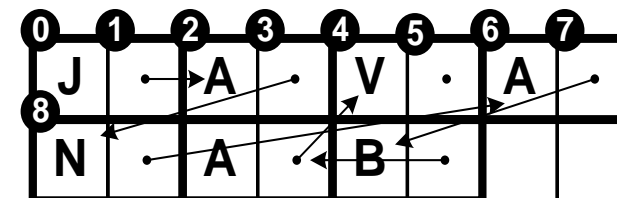
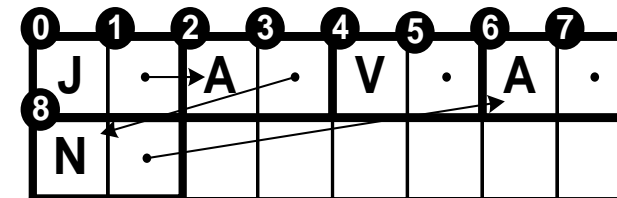
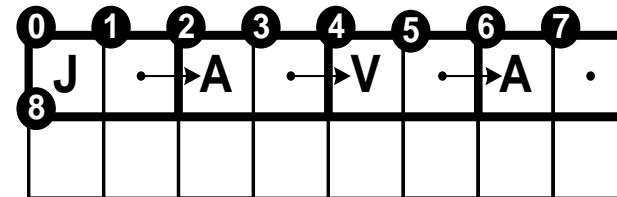
## ◆ Array

= beginvakje +  $i$



## ◆ Linked List

*Je bent van in het begin  
niet zeker waar het  
 $i$ -de element zich bevindt  
in het geheugen, zeker nadat  
elementen zijn toegevoegd  
en verwijderd*

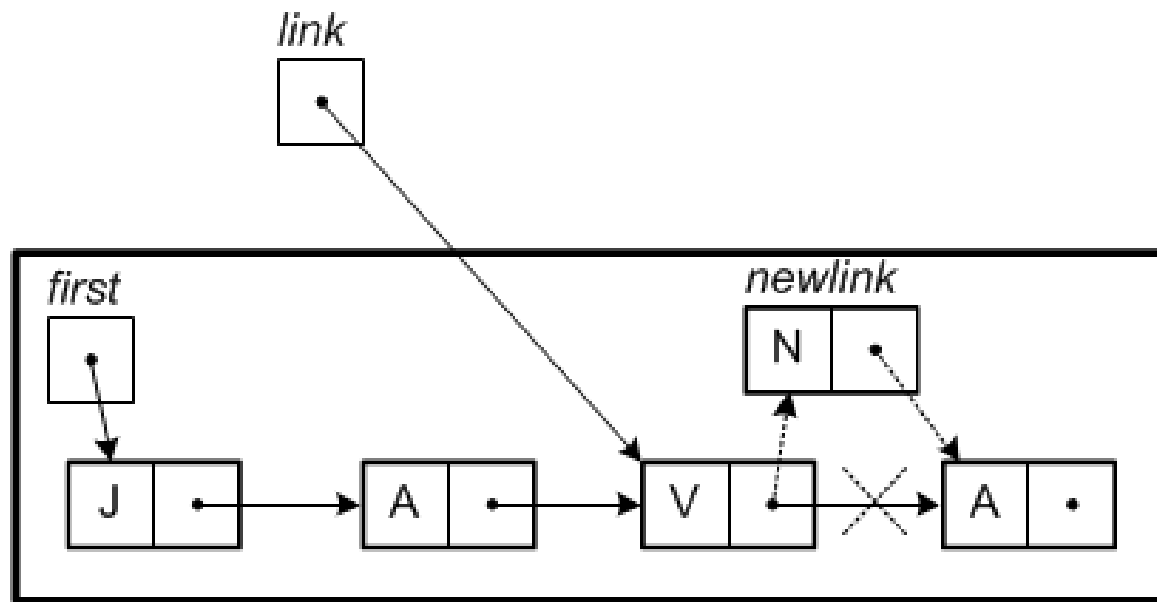


# Inlassen van een element

```
public void insert(T data, int index){
    Link<T> newlink = new Link<T>(data);
    if (index == 0){
        // wordt eerste element
        newlink.next = first;
        first = newlink;
    } else {
        int i=0;
        Link<T> link = first;
        while (link != null && i < index - 1){
            link = link.next;
            i++;
        }
        if (link == null)
            throw new IllegalArgumentException("Insert at
"+index+" impossible; list has only "+(i+1)+" elements.");
        newlink.next = link.next;
        link.next = newlink;
    }
}
```



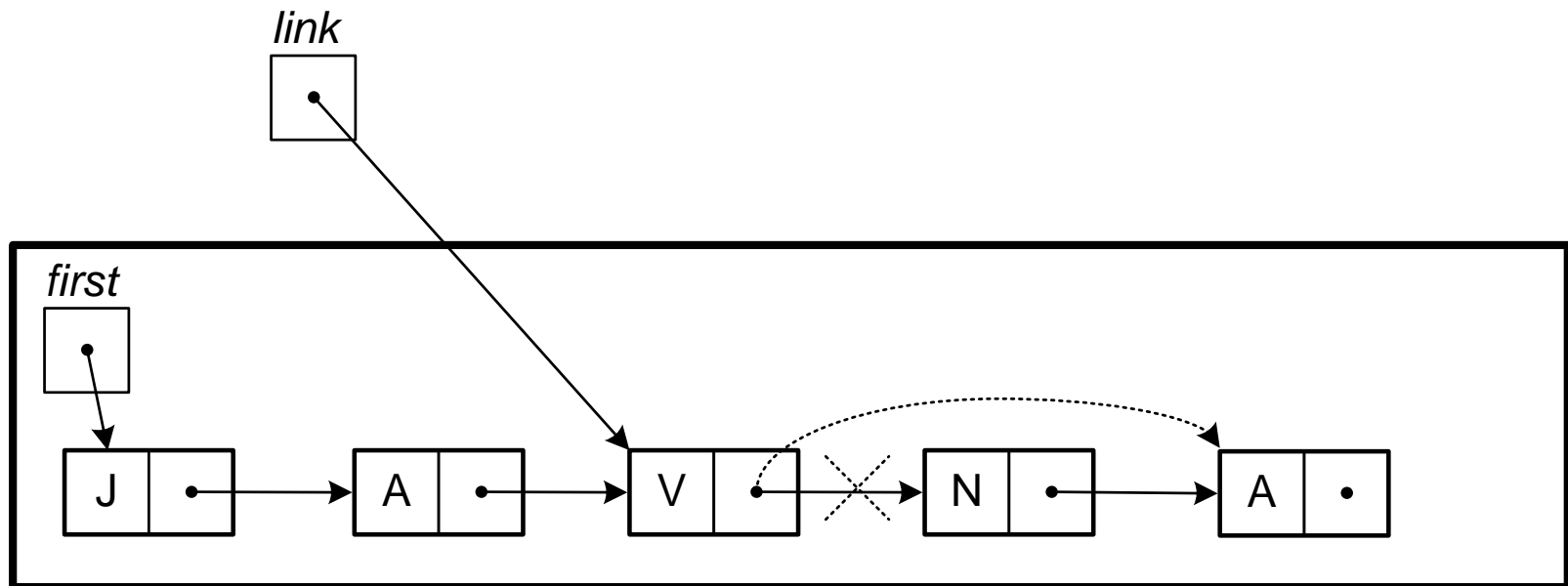
# Inlassen van een element



# Verwijderen van een element

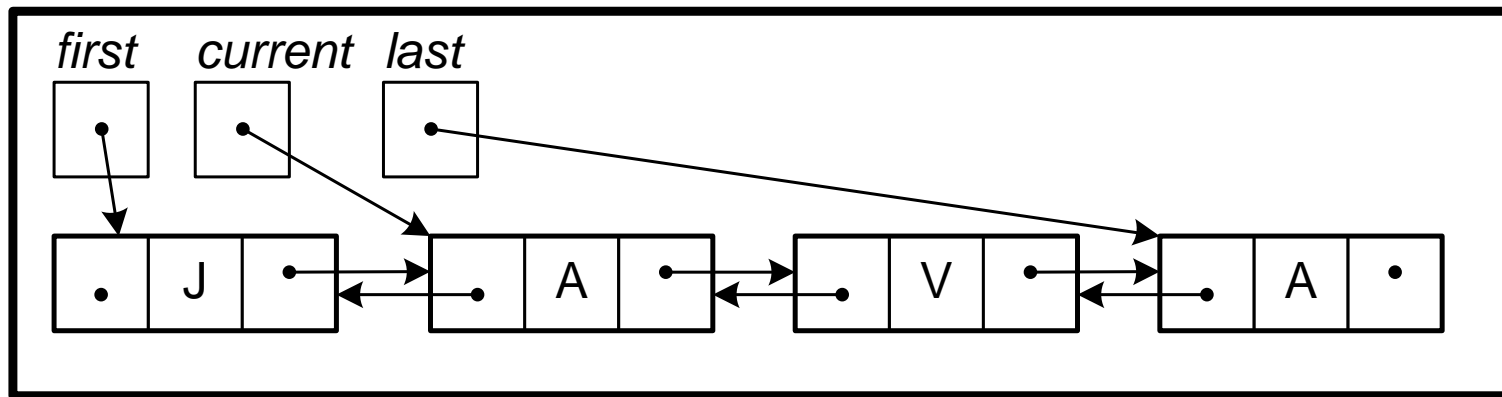
```
public void remove(int index) {
    if (index == 0) {
        if (first == null)
            throw new IllegalArgumentException("Removal
of first element impossible; list has no elements.");
        first = first.next;
    } else {
        int i=0;
        Link<T> link = first;
        while (link != null && i < index - 1){
            link = link.next;
            i++;
        }
        if (link == null || link.next == null)
            throw new IllegalArgumentException("Removal
of element "+index+" impossible; list has only "+(i+1)+"
elements.");
        link.next = link.next.next;
    }
}
```

# Verwijderen van een element



# Dubbele gelinkte lijst

*DoubleLinkedList*



## ◆ *Voordelen:*

- ◆ einde gemakkelijk te vinden
- ◆ houdt huidige positie bij

## ◆ *Toepassing:* word