

# Informatica

## Les 3

### Elektronica – Interfaces – Overschrijven

Jan Lemeire

**Informatica 2<sup>e</sup> semester**

*februari – mei 2020*

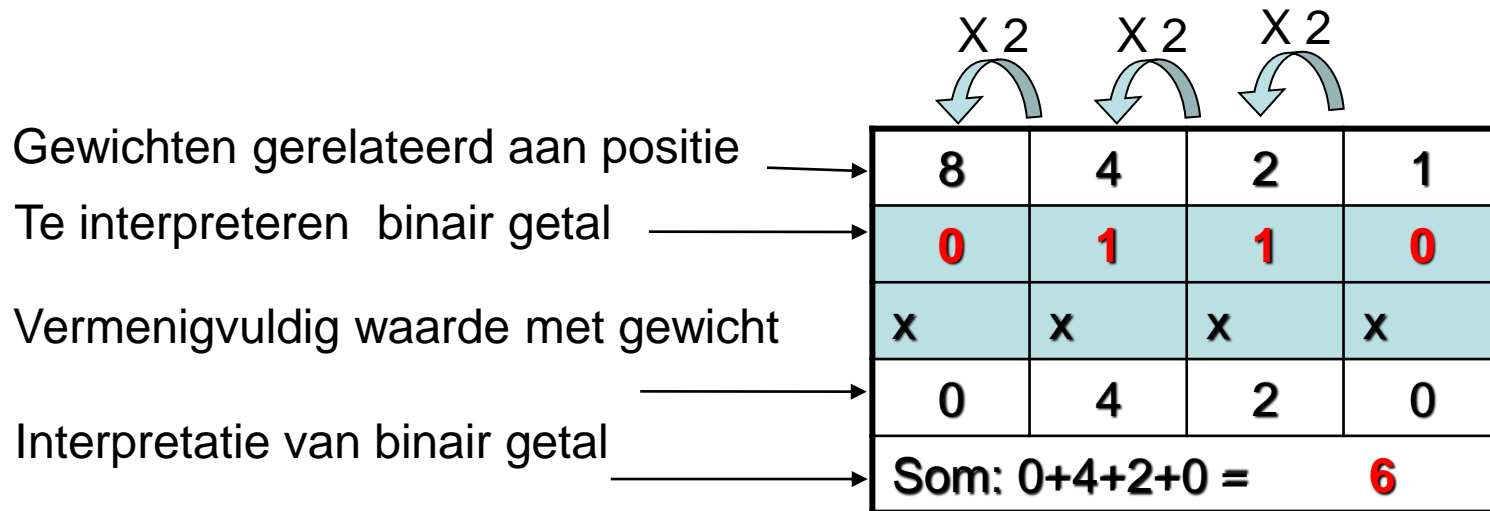


Vrije Universiteit Brussel

# Vandaag

1. **Deel III: hoofdstuk 1 – binair talstelsel**
2. **Deel III: hoofdstuk 2**
3. **Bibliotheekklassen Set & Map**
4. **Interfaces**
5. **Oefening**
6. **Overerving en overschrijven**

# Binair talstelsel



128	64	32	16	8	4	2	1	Decimale Interpretatie
1	0	0	0	0	0	1	0	130
0	1	0	1	0	1	0	1	85
0	0	1	0	1	0	0	1	41
a7	a6	a5	a4	a3	a2	a1	a0	$\sum_{i=0}^n a_i \cdot 2^i$

# Hexadecimaal talstelsel

Om de notatie van binaire getallen te verkorten worden ook de **hexadecimale notatie** gebruikt. Een hexadecimaal cijfer beschrijft 4 bits en wordt voorgesteld door 0, 1, ..., 9, A, ..., E.

Binair	Hex.	Binair	Hex.
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Vb: omzetting van 1110001101

1	1	1	0	0	0	1	1	0	1
3			8			D			

Om aan te duiden dat een getal in het hexadecimaal talstelsel is (bvb tijdens programmeren), voegen we de prefix '0x' toe: 0x38D.

*0x11 is dus gelijk aan 17*

# Waarmaken van Leibniz's droom



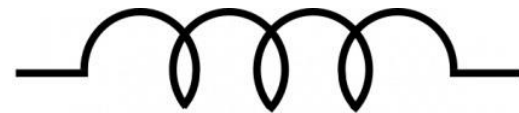


## Hoofdstuk 2: De elektronische relais

# Hoe elektrisch maken?

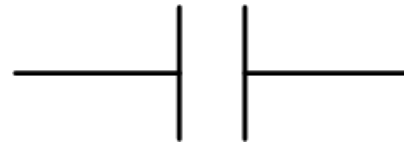
## 1. Geleider

1. Weerstand
2. Spoel
3. Verbinden van componenten



## 2. Isolator (*dielectric*)

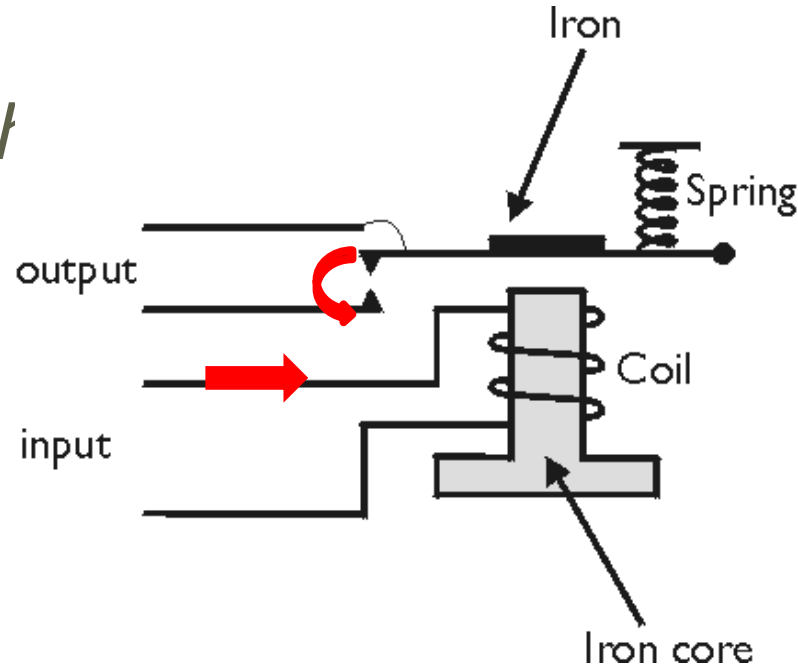
1. Condensator
2. Scheiden van componenten



***Extra component nodig...***

# Nodig: de relais

## *Electro-mech*



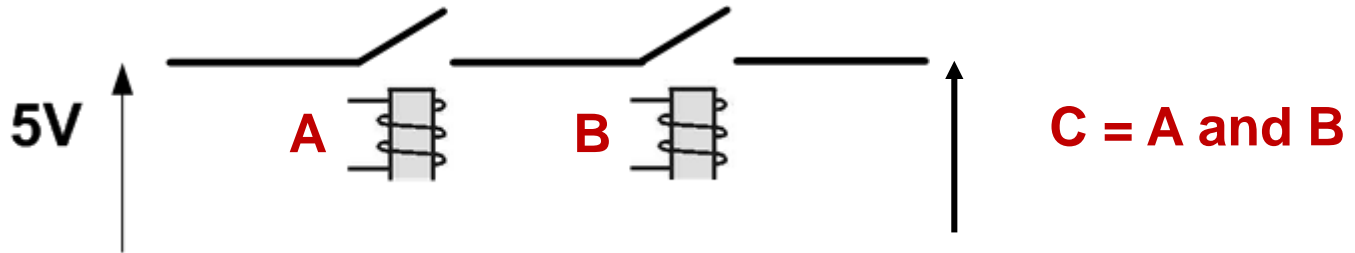
Naast de geleider, weerstand, capaciteit en spoel hebben we een component nodig met de werking zoals de elektromechanische relais. De relais zorgt ervoor dat een signaal een ander signaal kan beïnvloeden.

Het inputsignaal bepaalt of de output 'aan' of 'uit' is. Met deze component kunnen we elektrische schakelingen maken die binair kunnen rekenen. Gebaseerd op de elektromechanische relais bouwde de duitser Zuse een volledig werkende computer die gebruikt werd tijdens de tweede wereldoorlog.

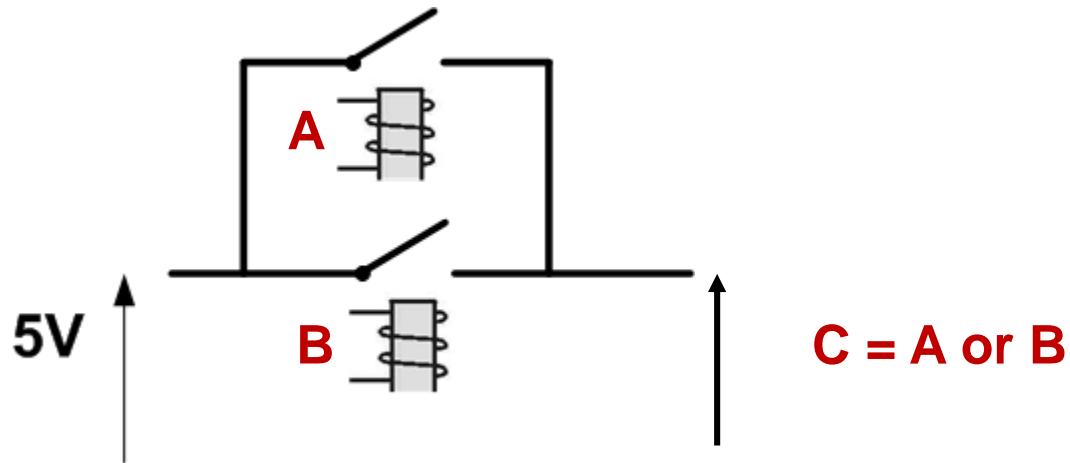


# Binair rekenen

and



or



*Met deze componenten kan je alles berekenen!*

# Complexe functies

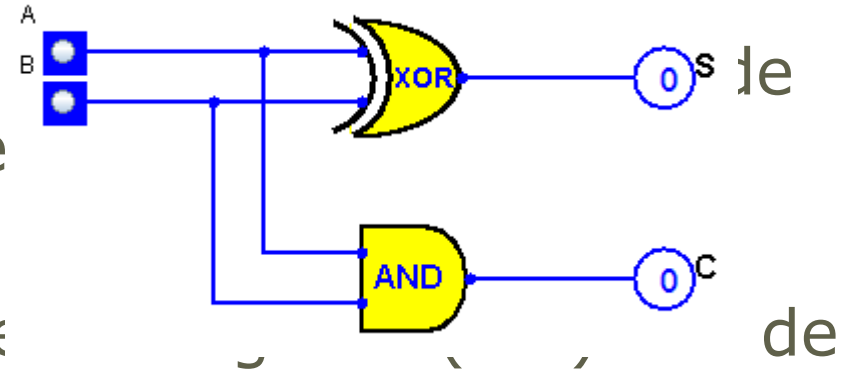
- ◆ Algemeen: van  $n$  inputs naar  $m$  outputs
  - ✦ 2-naar- $m$  kan je ontbinden als  $m$  keer 2-naar-1
  - ✦  $n$ -naar-1 kan je ontbinden door 2 bij 2 samen te nemen
    - Bvb  $\{a, b, c, d\} \Rightarrow x$  door  $\{a, b\} \Rightarrow e$ ,  $\{c, d\} \Rightarrow f$  en  $\{e, f\} \Rightarrow x$
  - ✦ Beide samen geeft  $n$ -naar- $m$ 
    - *is misschien wel niet het efficiëntste*



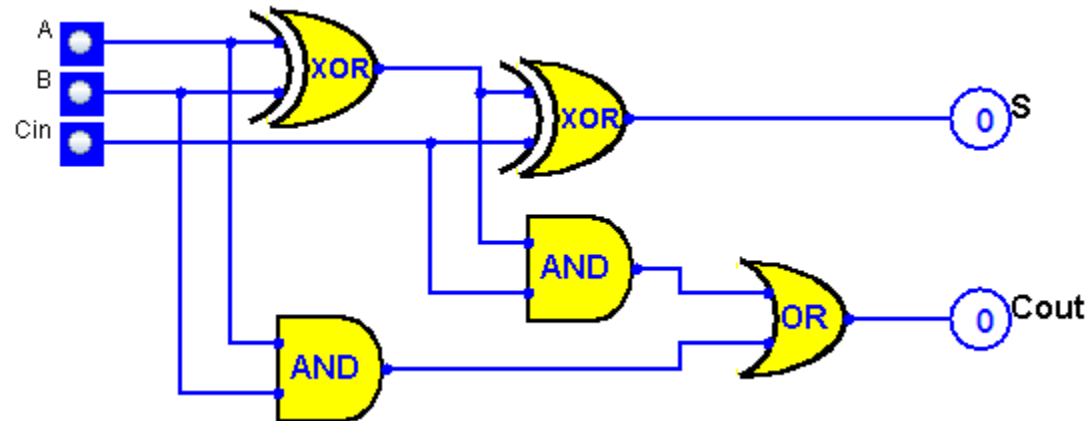
# Som van bits

## Half adder

- ◆ Som  $S$  van 2 bits  $A$  en  $B$ .  
bit voor de volgende orde
- ◆ Rekening houdend met de  
vorige orde.



## Full adder

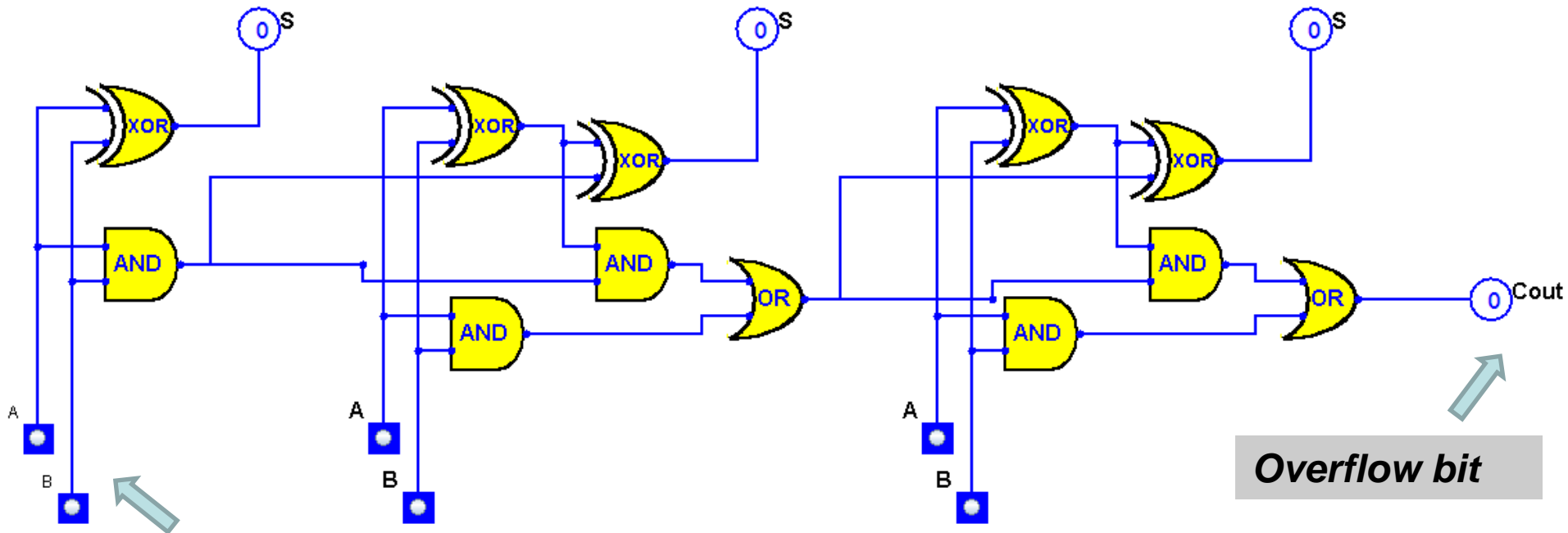




# Som van getallen

- ◆ Twee getallen (A en B) van 3 bits bij elkaar tellen (S)

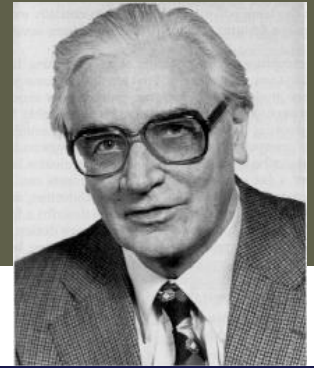
## Ripple carry adder



*Least-significant bits (bits van laagste orde)*

Zie <http://parallel.vub.ac.be/education/java/applets.html>

# WOII: de Duitsers



**Konrad Zuse**

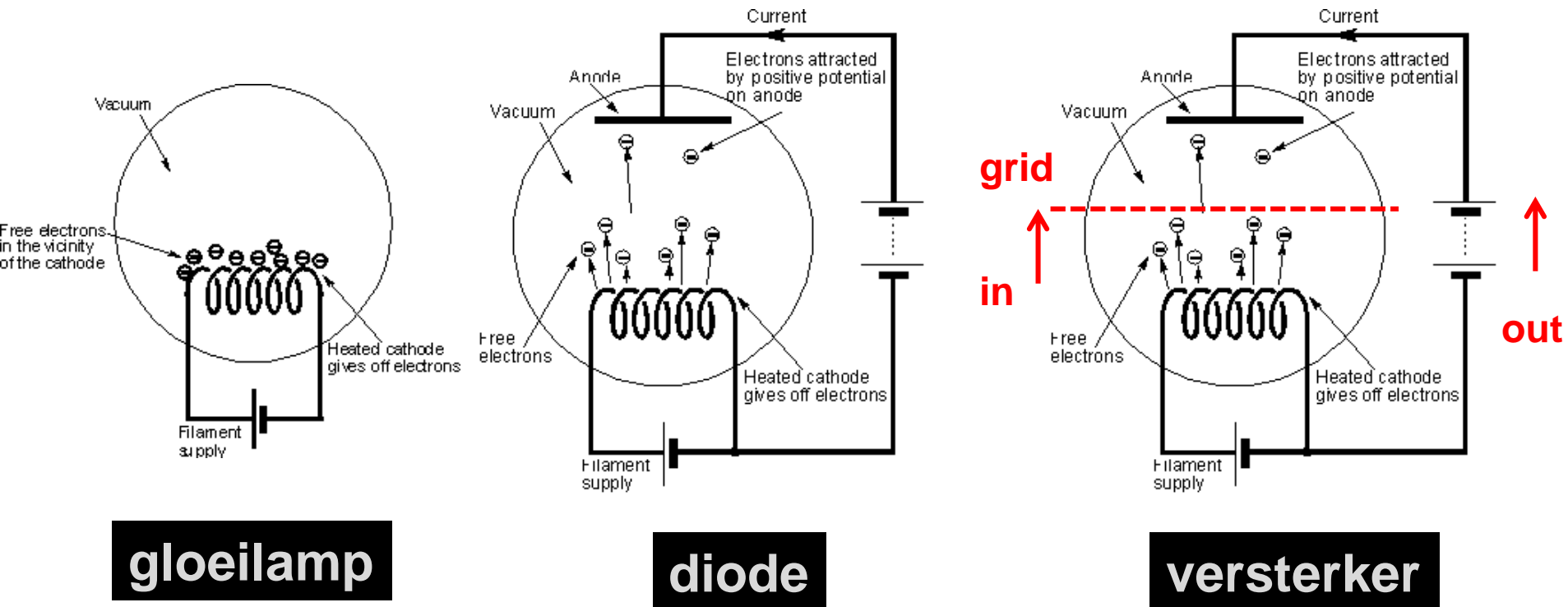
- ◆ Een werkende electro-mechanische computer



*Door de mechanische componenten beperkt tot vijftien à twintig operaties per seconde*

# De Vacuümbuis: elektronische relais

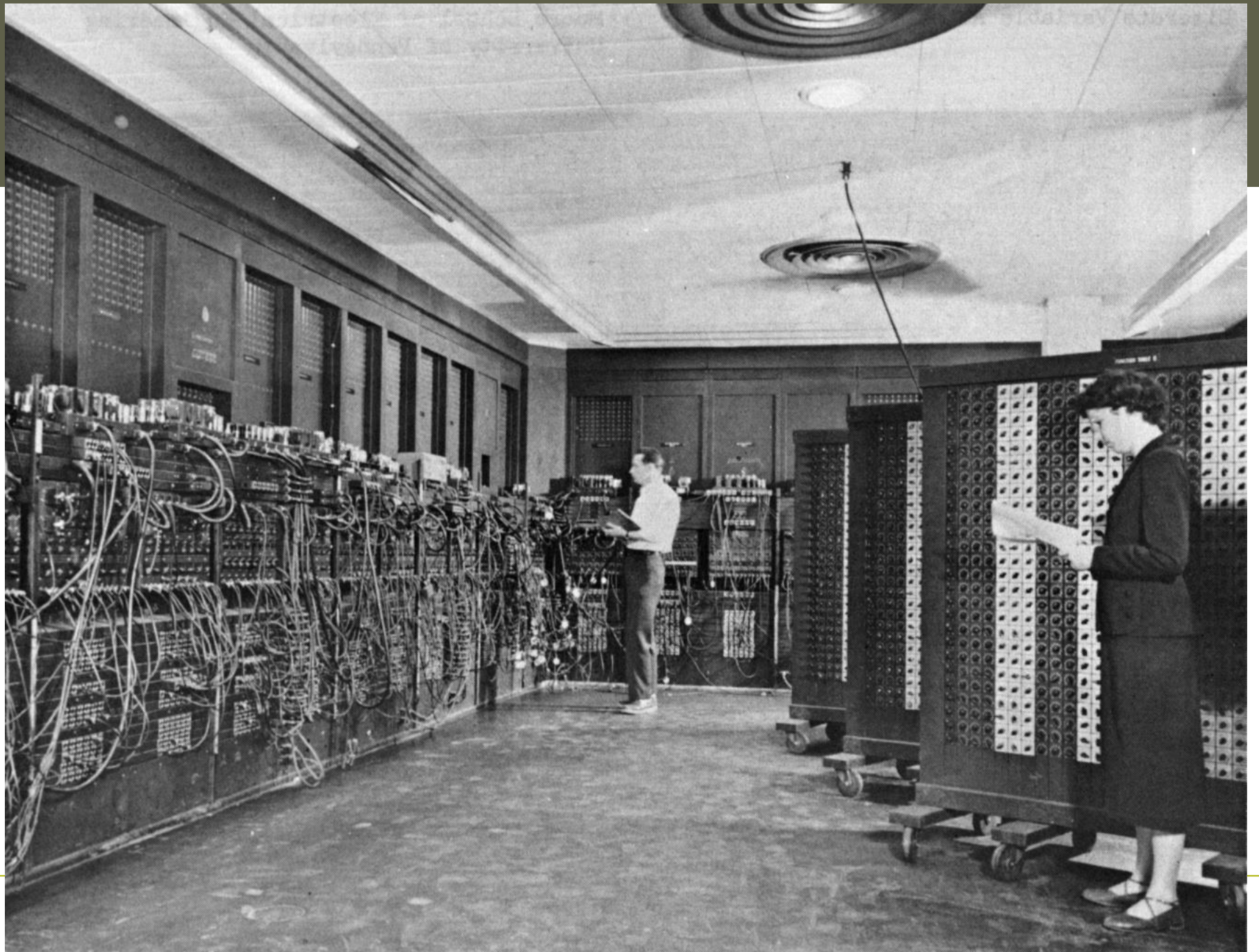
## Vrije elektronen in vacuüm



De spanning (*in*) van het grid van de vacuümbuis bepaalt de geleidbaarheid tussen anode en kathode. Door een grote spanning op *out* te zetten, zal een klein inputsignaal *versterkt* worden. Een signaal met een klein vermogen wordt omgezet in een signaal met een groot vermogen.

# Vacuümbuizen







# De eerste computer: ENIAC

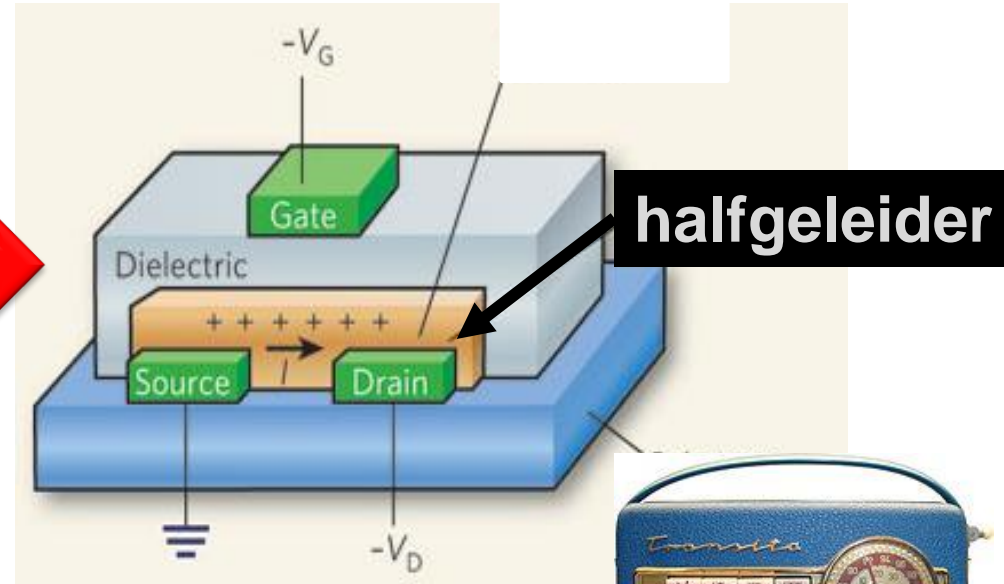
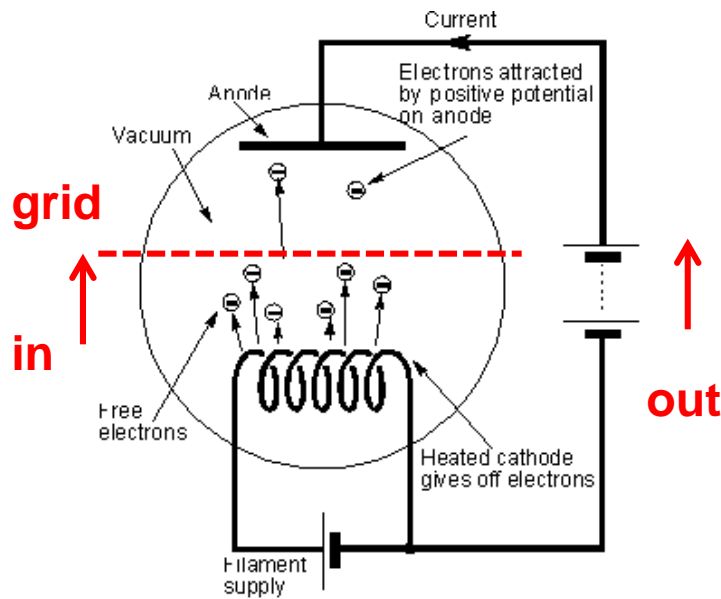


John Mauchly and John Eckert, 1945



De ENIAC maakte gebruik van vacuümbuizen voor het rekenen. Deze zijn echter duur, groot, onhandig en gaan gemakkelijk stuk.

# Van vacuümbuis naar transistor

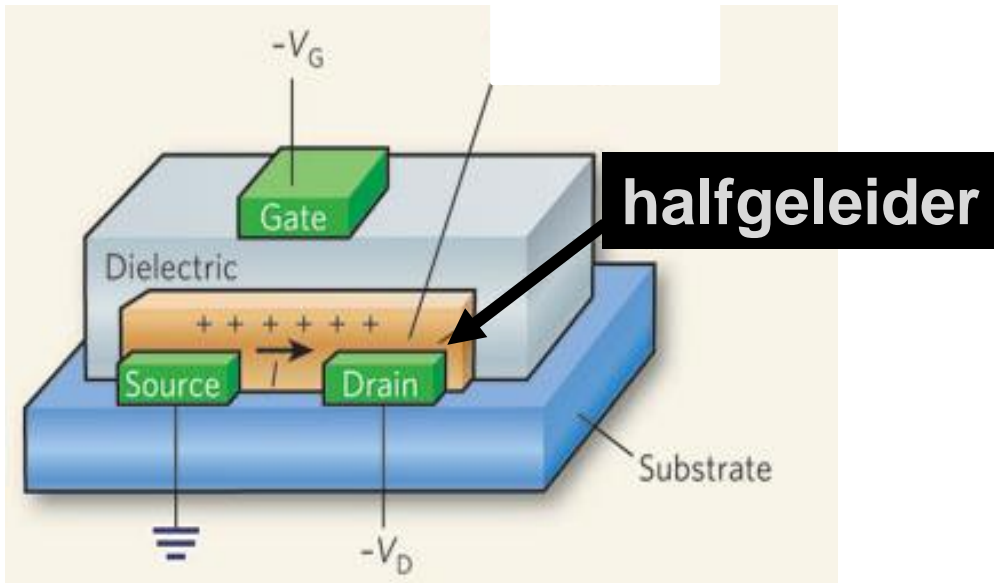


Maar de echte doorbraak kwam met de uitvinding van de transistor. 10 jaar (!) werd er in Bell Labs (van AT&T - American Telephone & Telegraph Company) actief gezocht naar een handige vervanger van de vacuümbuis. Het labo bestond uit praktische technici, theoretici (fysica, wiskunde, chemie) en creatieve geesten. De eerste succesvolle toepassing was de draagbare transistorradio.

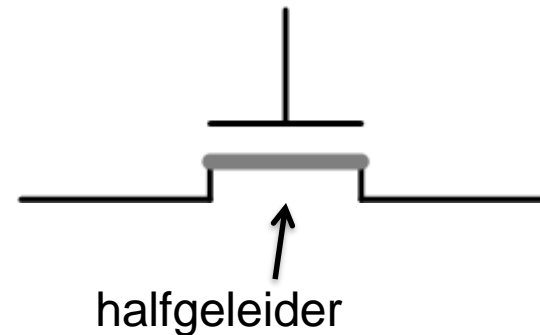
# Transistor (MOSFET)



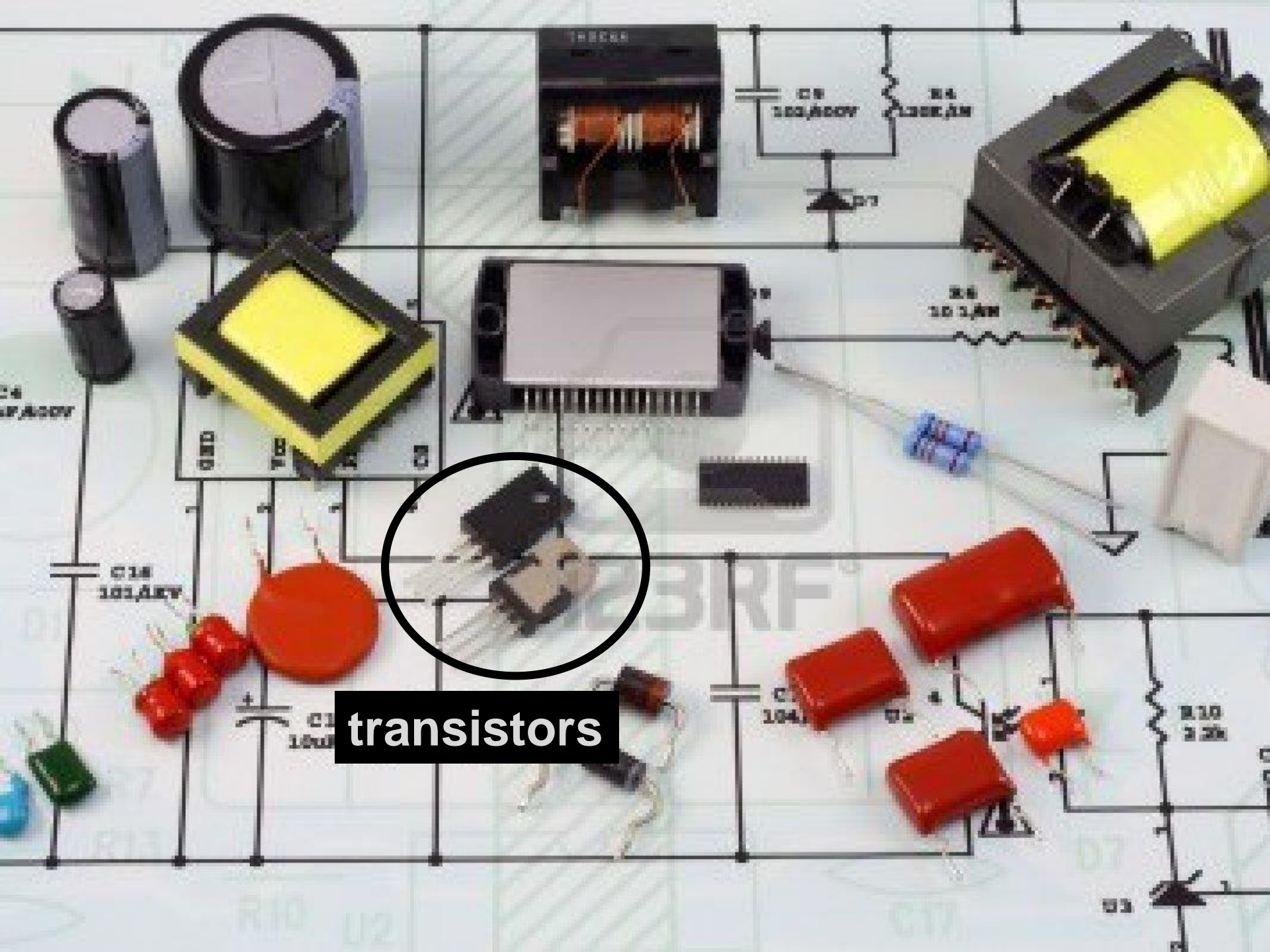
- ◆ De spanning  $V_G$  bepaalt de geleidbaarheid van de halfgeleider (semiconductor) tussen Source en Drain
  - ◆ Te gebruiken als versterker
  - ◆ Te gebruiken als relais, voor binaire operaties



<http://www-g.eng.cam.ac.uk/mmg/teaching/linearcircuits/mosfet.html>



De halfgeleider gedraagt zich als een isolator als  $V_G=0$  en als geleider bij  $V_G>1V$ . We hebben een elektronische relais!



transistors

# Analoge electronica

*Sommeren, integreren,  
afleiden*

<http://www.falstad.com/circuit/>

# 1.2.4 Verzameling

## Set

# De Set

## Definitie van documentatie:

*“A collection that contains no duplicate elements. More formally, sets contain no pair of elements  $e1$  and  $e2$  such that  $e1.equals(e2)$ , and at most one null element.”*

- ◆ Een wiskundige verzameling dus...
- ◆ In Java gedefinieerd als ***interface***

## Method Summary

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

Adds the specified element to this set if it is not already present.

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

Adds all of the elements in the specified collection to this set if they're not already present (optional operation).

void [clear](#)()

Removes all of the elements from this set.

boolean [contains](#)([Object](#) o)

Returns true if this set contains the specified element.

boolean [containsAll](#)([Collection](#)<?> c)

Returns true if this set contains all of the elements of the specified collection.

boolean [equals](#)([Object](#) o)

Compares the specified object with this set for equality.

boolean [isEmpty](#)()

Returns true if this set contains no elements.

boolean [remove](#)([Object](#) o)

Removes the specified element from this set if it is present (optional operation).

boolean [removeAll](#)([Collection](#)<?> c)

Removes from this set all of its elements that are contained in the specified collection.

boolean [retainAll](#)([Collection](#)<?> c)

Retains only the elements in this set that are contained in the specified collection.

int [size](#)()

Returns the number of elements in this set (its cardinality).



# Bewerkingen

bewerking		methode
Unie	$a \cup b$	AddAll ()
Doorsnede	$a \cap b$	RetainAll ()
Verschil	$a \setminus b$	RemoveAll ()
Symmetrische verschil	$(a \setminus b) \cup (b \setminus a)$	Combinatie van bovenstaande

# Interfaces

## Set.java

```
public interface Set<E> extends Collection<E> {  
  
    int size();  
    boolean isEmpty();  
    boolean contains(Object o);  
    Iterator<E> iterator();  
    Object[] toArray();  
    <T> T[] toArray(T[] a);  
    boolean add(E e);  
    boolean remove(Object o);  
    boolean containsAll(Collection<?> c);  
    boolean addAll(Collection<? extends E> c);  
    boolean retainAll(Collection<?> c);  
    boolean removeAll(Collection<?> c);  
    void clear();  
    boolean equals(Object o);  
    int hashCode();  
}
```

# Interfaces

- ◆ Alle methodes zijn *abstract*: enkel header, geen implementatie
- ◆ Klasse die interface implementeert moet *alle* methodes implementeren

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.

## **Filosofie:**

- ◆ een interface definieert hoe je met een object communiceert.
- ◆ Je hoeft de implementatie niet te kennen
- ◆ Inwisselbare implementatie, efficiëntste implementatie hangt af van situatie

# Pijlers van object-georiënteerde programmeertalen


## 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

# Set Object: implementatie

- ◆ Implementaties voldoen aan het “Set-kontrakt”
- ◆ Gebruik: *TreeSet* en *HashSet*
  - ✦ *Komen we op terug in hoofdstuk 8*

# 1.2.5 Mappen

# Mappen of *dictionaries*

- ◆ Linken object van type A aan objecten van type B.
  - ✦ Opzoeken gebeurt op A-objecten, dus A best gesorteerd
- ◆ Voorbeeld:

Ida, Jennifer	Oprolbare brug
Laurent, Maarten	Ultragrabber
Manuka, Stani	Moving Tower Ahmedabad
Mitichashvili, Tornike	piramide
Nguyen, Kim	trap Villa Savoye
Pelicaen, Erik	achthoekstructuur
Perez Sotomayor, Lucia	Origami



**student**

**key**

**project**

**value**



# Java map

## Interface Map<K,V>

### Met Type Parameters:

K - the type of *keys* maintained by this map

V - the type of mapped *values*

```
Map<Student, String> map;
```

# Map methodes

void [clear](#) ()

Removes all of the mappings from this map (optional operation).

boolean [containsKey](#) ([Object](#) key)

Returns true if this map contains a mapping for the specified key.

boolean [containsValue](#) ([Object](#) value)

Returns true if this map maps one or more keys to the specified value.

[Set](#)<[Map.Entry](#)<[K](#), [V](#) [entrySet](#) ()

>> Returns a [Set](#) view of the mappings contained in this map.

boolean [equals](#) ([Object](#) o)

Compares the specified object with this map for equality.

[V](#) [get](#) ([Object](#) key)

Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

int [hashCode](#) ()

Returns the hash code value for this map.

boolean [isEmpty](#) ()

Returns true if this map contains no key-value mappings.

[Set](#)<[K](#)> [keySet](#) ()

Returns a [Set](#) view of the keys contained in this map.

[V](#) [put](#) ([K](#) key, [V](#) value)

Associates the specified value with the specified key in this map (optional operation).

void [putAll](#) ([Map](#)<? extends [K](#), ? extends [V](#)> m)

Copies all of the mappings from the specified map to this map (optional operation).

[V](#) [remove](#) ([Object](#) key)

Removes the mapping for a key from this map if it is present (optional operation).

int [size](#) ()

Returns the number of key-value mappings in this map.

[Collection](#)<[V](#)> [values](#) ()

# Implementaties van Java map

## ◆ *TreeMap* en *HashMap*

✦ *Tree of Hashtabel (zien er later!)*

```
Map<String, String> map;
```

```
map = new TreeMap<String, String>();
```

*of*

```
map = new HashMap<String, String>();
```

## ◆ *Cf. TreeSet* en *HashSet* implementaties van de *Set* interface

# Klasse-oefening

```
class Trapezium {
    double lengte1, lengte2, breedte;
    Trapezium(double lengte1, double lengte2, double breedte){
        this.lengte1 = lengte1;
        this.lengte2 = lengte2;
        this.breedte = breedte;
    }
    double oppervlakte(){ return breedte * (lengte1 + lengte2)/2; }
}
class Rechthoek {
    double breedte, lengte;
    Rechthoek(double lengte, double breedte){
        this.breedte = breedte;
        this.lengte = lengte;
    }
}
class Vierkant extends Rechthoek {
    Vierkant(double zijde){
        super(zijde, zijde);
    }
}
```

```
/** Gegeven de volgende klassen: Trapezium, Rechthoek, Vierkant  
1. Maak een object aan van elke klasse. Kies je eigen waarden  
voor de parameters.  
2. Duidt aan welke constructors worden opgeroepen en welke  
superconstructors.  
3. Geef voor elk object de waarde van de attributen.  
4. Bereken de oppervlakte van de trapezium en print deze af.  
5. Voeg een methode toe aan Rechthoek die de oppervlakte  
berekent.  
6. Maak van de rechthoek een subklasse van het trapezium. Dan  
zijn de attributen 'breedte' en 'lengte' in feite niet meer  
nodig, die mag je dus schrappen. De oppervlaktemethode ook.  
7. Vierkant erft alle attributen over, maar heeft er in feite  
slechts 1 nodig. Schrap de 'extends Rechthoek', voeg het nodige  
attribuut toe en de methode voor oppervlakteberekening. */
```

# 1.2.6 Generics

## TER INFO

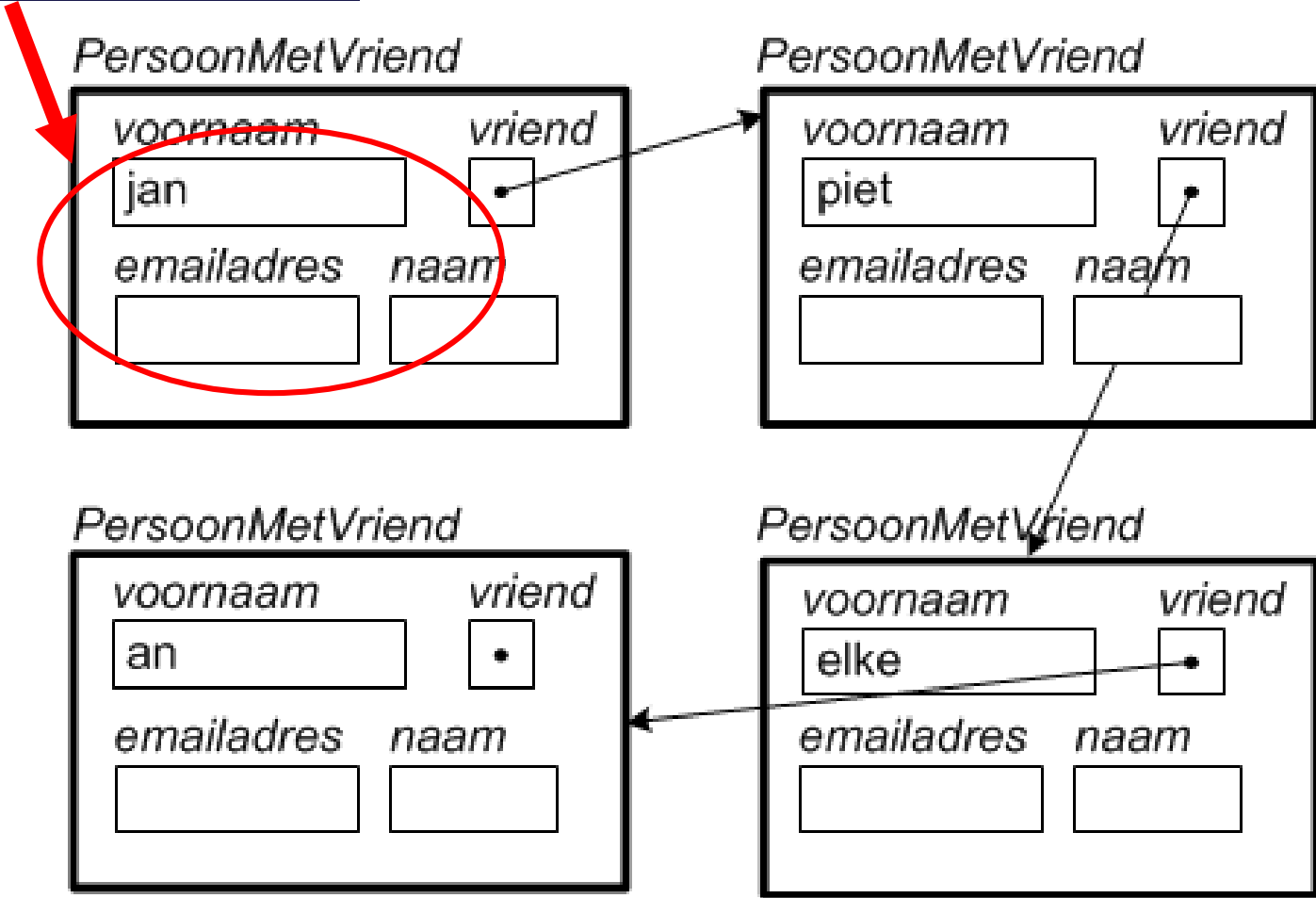
**1.3**

# **Vriendenvoorbeeld**



attributen van moederklasse

# Objecten



**Persoon**

```
String voornaam, naam;  
String emailadres;
```

```
void maakDefaultEmailadres(String  
domeinVanProvider);
```

**PersoonMetVriend**

```
PersoonMetVriend vriend;  
  
boolean kenJeDiePersoonViaVia  
(PersoonMetVriend persoon)
```

**Student**

```
String faculteit;  
int rolnummer, score;  
Vak[] vakken;  
int[] punten;
```

**Ontkenner**

```
boolean kenJeDiePersoonViaVia(persoon)
```

**Leugenaar**

```
boolean kenJeDiePersoonViaVia(persoon)
```

```
public class PersoonMetVriend extends Persoon {  
  
    //===== ATTRIBUTEN =====//  
    PersoonMetVriend vriend; // een persoon heeft maar 1 vriend  
  
    //===== CONSTRUCTOR =====//  
    PersoonMetVriend(String voornaam){  
        super(voornaam, ""); // oproep constructor van Persoon  
        vriend = null; // heeft bij default geen vriend  
    }  
  
    PersoonMetVriend(String voornaam, String naam){  
        super(voornaam, naam); // oproep constructor van Persoon  
        vriend = null; // heeft bij default geen vriend  
    }  
  
    //===== METHODES =====//  
    boolean kenJeDiePersoonViaVia(PersoonMetVriend persoon){  
        if (persoon == vriend)  
            return true;  
        else if (vriend != null)  
            return vriend.kenJeDiePersoonViaVia(persoon);  
        else  
            return false;  
    }  
  
    public String toString(){  
        return voornaam+"("+vriend+")";  
    }  
}
```

```

public class OefeningVrienden {

    //===== PROGRAMMA =====//
    public static void main(String[] args) {
        PersoonMetVriend jan = new PersoonMetVriend("jan");
        PersoonMetVriend piet = new PersoonMetVriend("piet");
        PersoonMetVriend elke = new PersoonMetVriend("elke");
        PersoonMetVriend an = new PersoonMetVriend("an");

        jan.vriend = piet;
        piet.vriend = elke;
        elke.vriend = an;

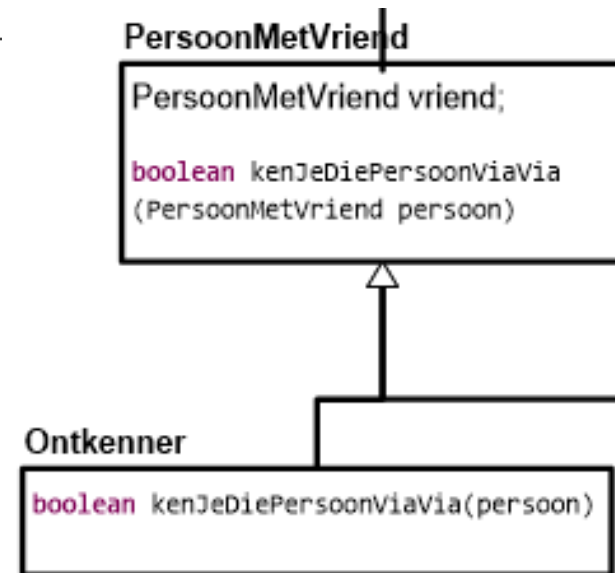
        boolean kentJanAn = jan.kenJeDiePersoonViaVia(an);
        System.out.println(jan+" kent "+an+"? "+kentJanAn);
    }
}

```

# Ontkenner

*Maak van Elke een ontkenner*

```
public class Ontkenner extends PersoonMetVriend {  
  
    Ontkenner(String naam) {  
        super(naam);  
    }  
    boolean kenJeDiePersoonViaVia(PersoonMetVriend vriend){  
        return false;  
    }  
}
```



**Overerving (Inheritance)**  
**met overschrijven van methode**

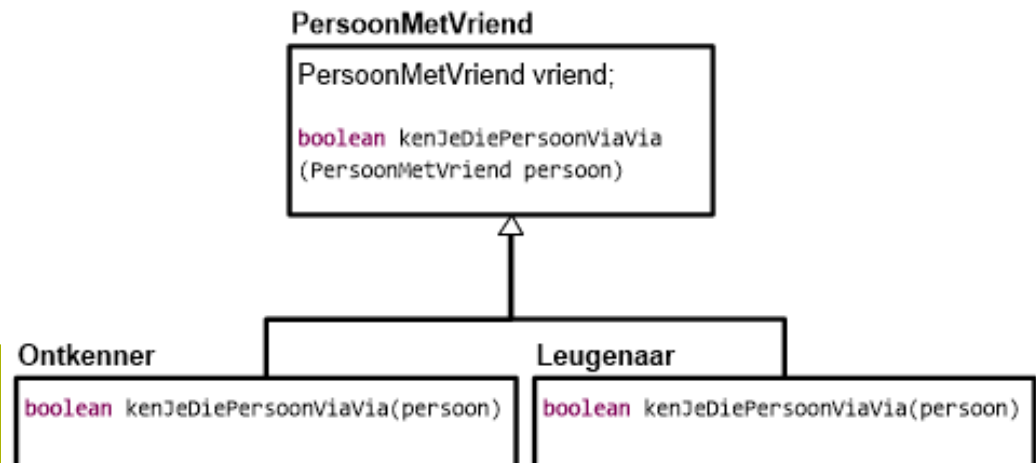
# Java's spelregels

6. Een klasse kan **erven** van 1 concrete of abstracte *superklasse* [*extends*].
  - De *subklasse* erft alle attributen en methodes van de superklasse over.
  - De *subklasse* mag attributen en/of methodes toevoegen.
  - In de constructor roep je met `super(x)` op de eerste lijn de constructor van de superklasse op. Als je die weg laat, wordt de default constructor opgeroepen (alsof er `super()` zou staan. Deze moet natuurlijk wel bestaan.
  - Methodes met dezelfde header als een methode van de superklasse *overschrijven* die methode, bij een object zal deze nieuwe methode opgeroepen worden ipv. die van de superklasse. De methode van de superklasse roep je op met `super.m()`.

# Leugenaar

*Maak van Piet een leugenaar*

```
public class Leugenaar extends PersoonMetVriend {  
  
    Leugenaar(String naam) {  
        super(naam);  
    }  
    boolean kenJeDiePersoonViaVia(PersoonMetVriend vriend){  
        return !super.kenJeDiePersoonViaVia(vriend);  
    }  
}
```



# Alan Turing





Winston Churchill



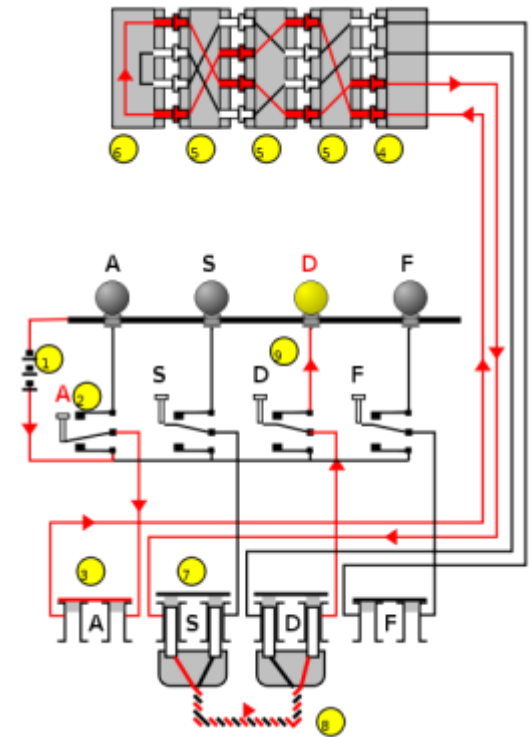
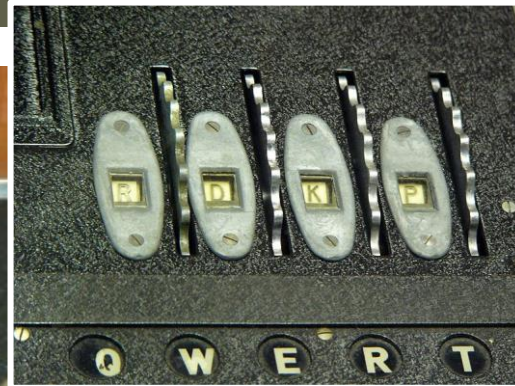
# Bletchley Park



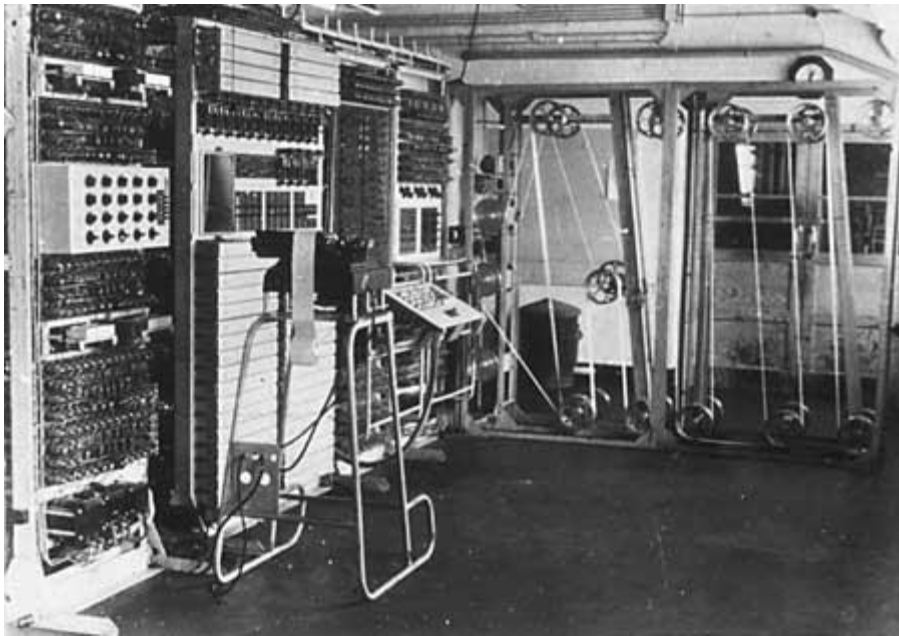
***Op zoek naar "cribs"***



# De Enigma code



# Colossus: the first operational electronic computer



**Alan Turing**  
**1912-1954**

Film 2014



THE TRUE ENIGMA  
WAS THE MAN WHO CRACKED  
THE CODE

BENEDICT CUMBERBATCH

KEIRA KNIGHTLEY

**THE IMITATION GAME**