

Informatica

Les 6

Basis- & slimme algoritmen

Jan Lemeire

Informatica 2^e semester

februari – mei 2021



Vrije Universiteit Brussel

Vandaag

- 1. Iets met functie**
- 2. Interfaces en abstracte klassen**
- 3. Newton's algoritme**
- 4. Discrete simulaties & Dijkstra**
- 5. Slimme algoritmen**

Basisalgoritmen

Hoofdstuk 4

Oefening iets met een functie

```

public class IetsMetFunctie {
    /** PROGRAMMA */
    public static void main(String[] args) {
        double a = -10, b = 10;

        double xx = vindXX(a, b);
        System.out.println("XX voor deze functie is "+xx);
    }
    public static double f(double x){
        //      return 2 *(x - 3);
        //      return x * x  + 12 *(x - 3);
        //      return - x * x * x / 8 + x * x  + 20 *(x - 3);
    }
    public static double vindXX(double a, double b){
        final double PRECISIE = 0.001; // configuratieparameter

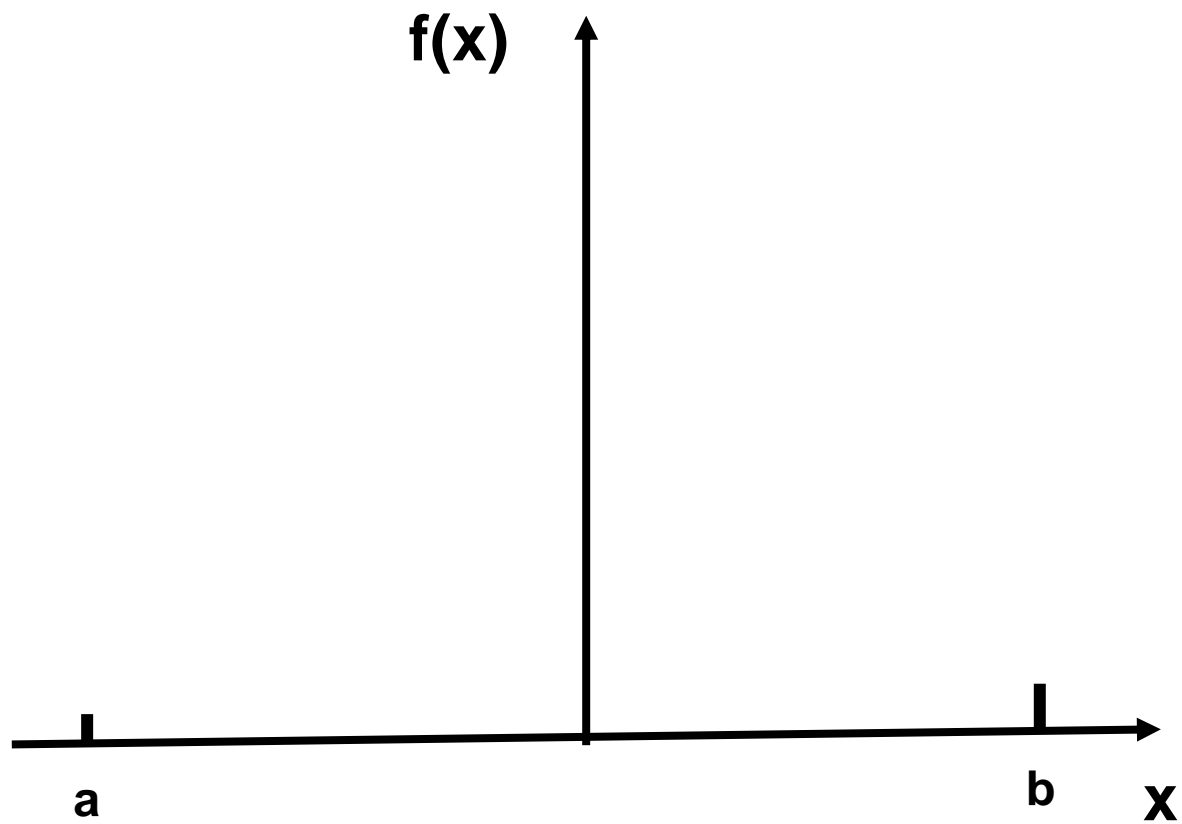
        if ( !(f(a) < 0 && f(b) > 0))
            throw new IllegalArgumentException("Geef betere beginpunten!");

        double m = (a + b) / 2;

        while (Math.abs( f(m)) > PRECISIE){
            System.out.print(m+", ");
            if (f(m) < 0)
                a = m;
            else
                b = m;
            m = (a + b) / 2;
        }
        System.out.println(m);
        return m;
    }
}

```

Absolute waarde



$$F(r) = 0.840376 \frac{2r+1}{(r+1)^2} - 0.221538 \frac{8r^2-3r-1}{(r+1)^3} - 0.0173983 \frac{54r^3-145r^2+12r+3}{(r+1)^4} + 0.0214648 \times$$

$$\frac{32r^4-203r^3+161r^2-5r-1}{(r+1)^5} + 0.0026529 \frac{5+30r-2010r^2+5376r^3-2835r^4+250r^5}{(r+1)^6}$$

Voor welke r -waarden wordt $F(r)$ nul?

- Analytisch niet altijd oplosbaar...
- Voor waarden van r de functie uitrekenen

Abstractie: Functie

Old school

```
/** PROGRAMMA */
public static void main(String[] args) {
    double a = -10, b = 10;

    double nulpunt = vindNulpunt(a, b, 1);
    System.out.println("XX voor deze functie is "+nulpunt);
}

public static double f(double x, int functie){
    switch (functie) {
        case(1): return 2 *(x - 3);
        case(2): return x * x + 12 *(x - 3);
        case(3): return - x * x * x / 8 + x * x + 20 *(x - 3);
        default: throw new IllegalArgumentException("Functie "+functie+" is geen geldige parameter.");
    }
}

public static double vindNulpunt(double a, double b, int functie){
    final double PRECISIE = 0.000001; // configuratieparameter

    if ( !(f(a, functie) < 0 && f(b, functie) > 0))
        throw new IllegalArgumentException("Geef betere beginpunten!");

    double m = (a + b) / 2;
    int ctr=0;
    while( Math.abs( f(m, functie)) > PRECISIE){
        System.out.print(m+", ");
        if (f(m, functie) < 0)
            a = m;
        else
            b = m;
        m = (a + b) / 2;
    }
}
```

Functie als object

Functie1

```
public double f(double x)
{
    return 2 * (x - 3);
}
```



```
public static double vindNulpunt(double a, double b, Functie functie) {
    if ( ! functie.f(a) < 0 && functie.f(b) > 0 )
        throw new IllegalArgumentException("Geef betere beginpunten!");

    double m = (a + b) / 2;

    while( Math.abs( functie.f(m) ) > 0.01) {
        System.out.print(m+", ");
        if (functie.f(m) < 0)
            a = m;
        else
            b = m;
        m = (a + b) / 2;
    }
    System.out.println(m);
    return m;
}
```

```
public interface Functie {
    /** geeft functiewaarde in
    opgegeven punt */
    double f(double x);
}
```

Interfaces

7. Een **abstracte methode** heeft *geen implementatie* (dêel 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.

Voordeel van abstractie

Librarycode (niet aanpasbaar)

interface Functie

```
double f(double x);
```

```
double vindNulpunt(a, b, Functie functie)
```

```
{  
    ...  
    Code gebruik makend van functie en  
    enkel de methoden van interface Functie  
    ...  
}
```

Mijn code

class Functie1

```
public double f(double x) {  
    return 2 *(x - 3);  
}
```

```
public static void main(String[] args)
```

```
{  
    Functie functie = new Functie1();  
    double x = vindNulpunt(a, b, functie);  
}
```

Functie: iets abstracts

- ◆ Nulpunt zoeken via een Functie object
- ◆ *De methode werkt immers voor alle functies*
- ◆ Definiëren wat een *functie* is:
 - ✦ Methode: `double f(double x)`
- ➔ Hiërarchie van functies

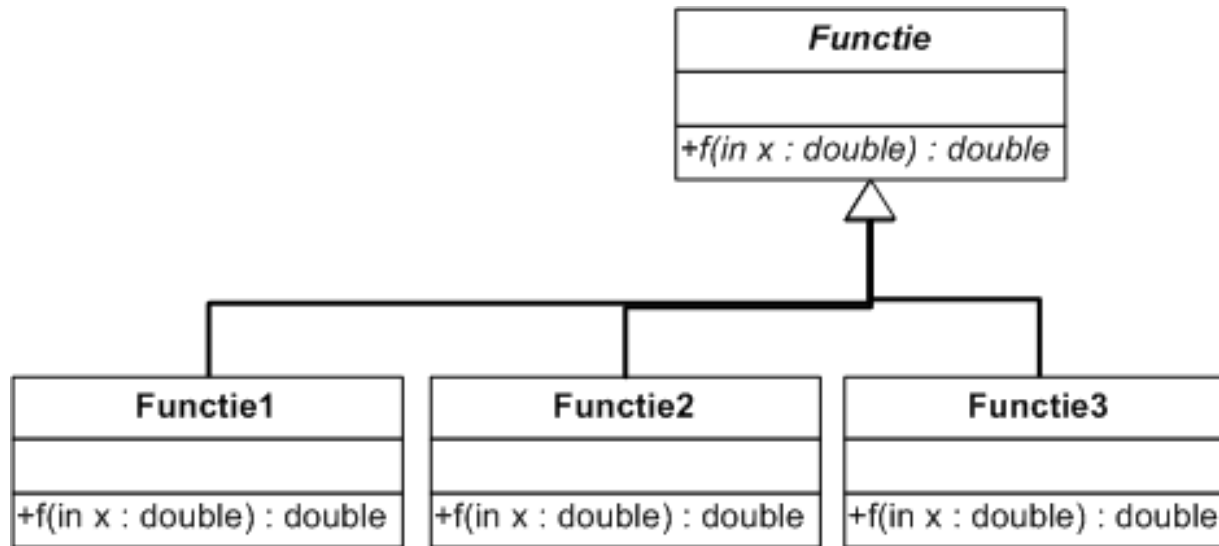
```
public interface Functie {  
    /** geeft functiewaarde in opgegeven punt */  
    double f(double x);  
}
```

```
class Functie1 implements Functie{  
    public double f(double x) {  
        return 2 * (x - 3);  
    }  
}
```

```
class Functie2 implements Functie{  
    public double f(double x) {  
        return x * x + 12 * (x - 3);  
    }  
}
```

```
class Functie3 implements Functie{  
    public double f(double x) {  
        return - x * x * x / 8 + x * x + 20 * (x - 3);  
    }  
}
```

Klasses

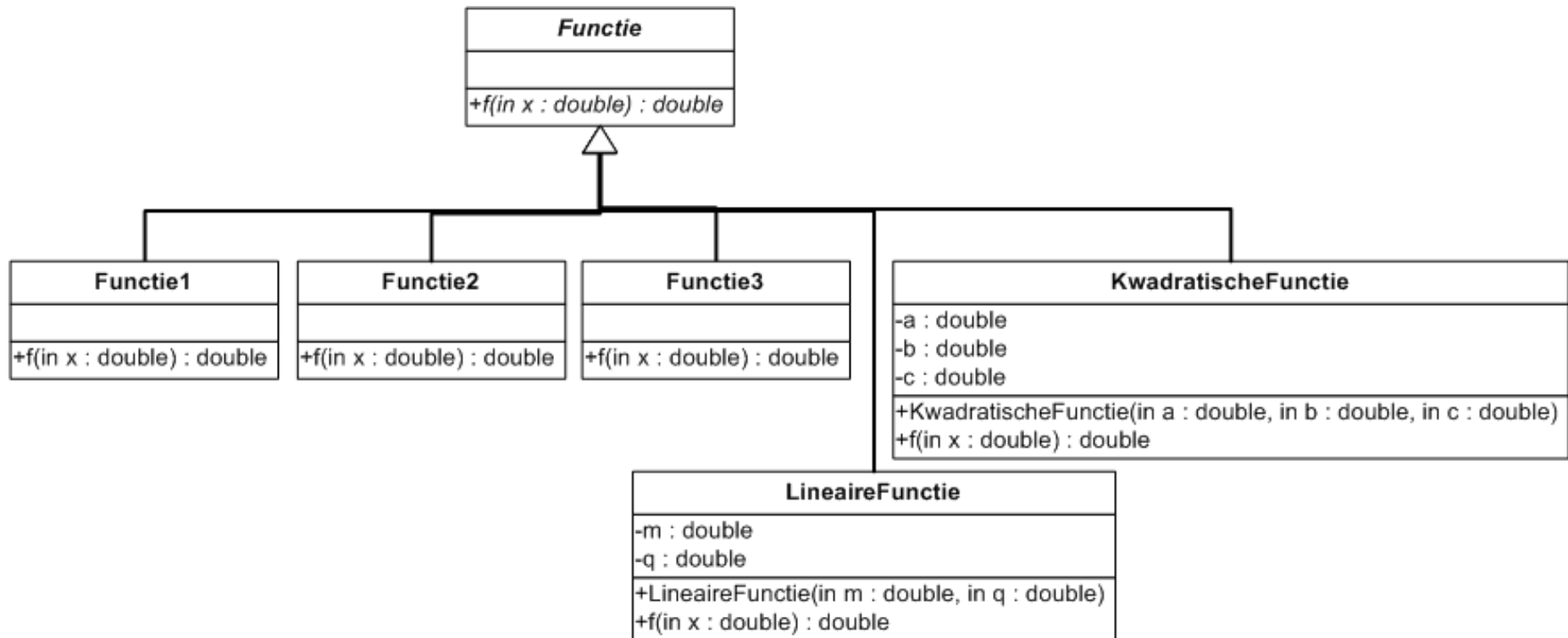


Gebruik

- ◆ Constructor: hier zonder parameters
- ◆ Default constructor: moet je niet definiëren
 - ✦ Als je een andere constructor definieert, vervalt deze. Met de andere constructor geef je aan dat je parameters moet meegeven.

```
Functie functie = new Functie2();  
double nulpunt = vindNulpunt(a, b, functie);
```


Extra klassen



```

class LineaireFunctie implements Functie{
    double m, q;
    public LineaireFunctie(double m, double q){
        this.m = m;
        this.q = q;
    }
    public double f(double x) {
        return m * x + q;
    }
}

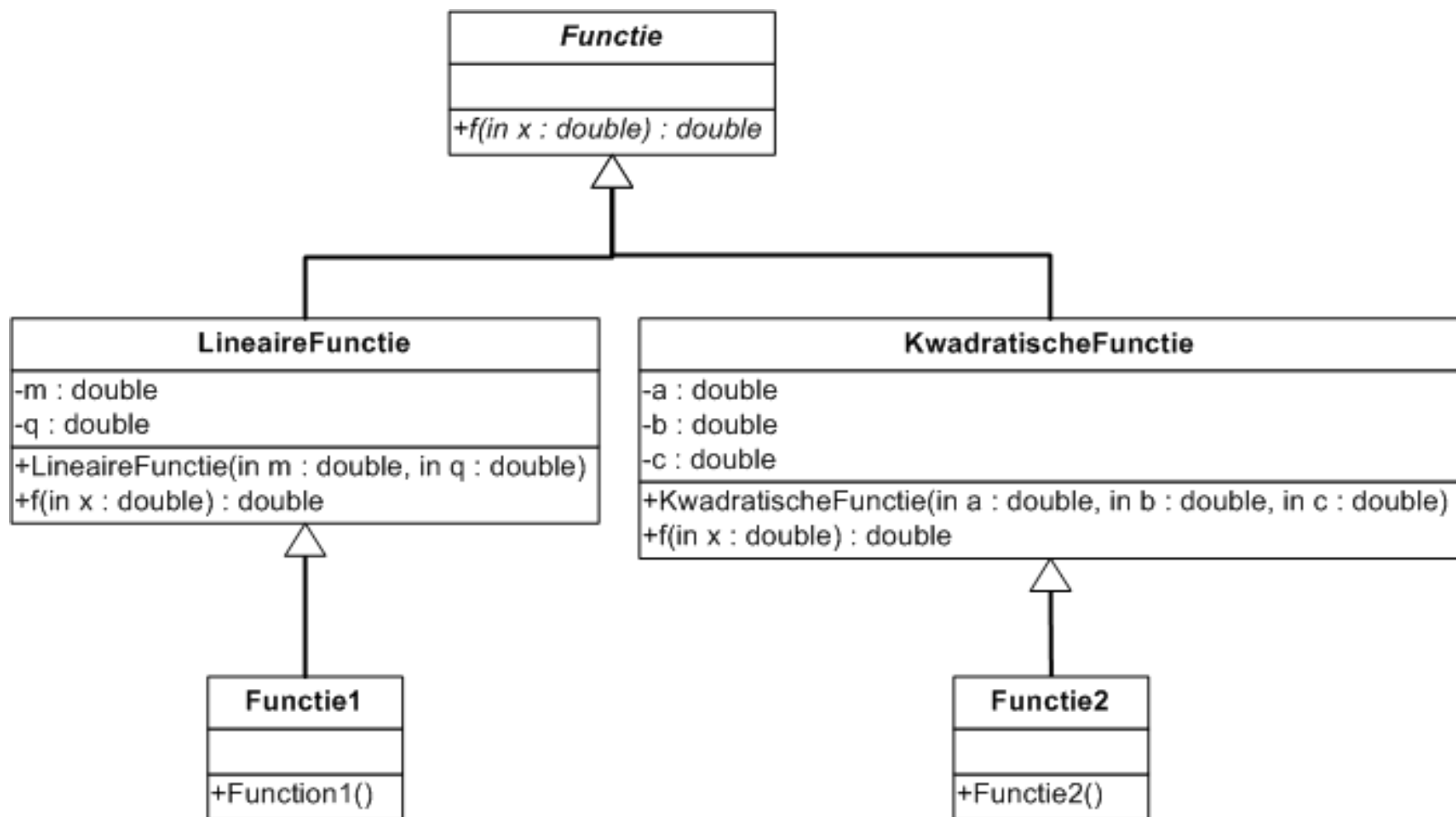
class KwadratischeFunctie implements Functie{
    double a, b, c;
    public KwadratischeFunctie(double a, double b, double c){
        this.a = a;
        this.b = b;
        this.c = c;
    }
    public double f(double x) {
        return a * x * x + b * x + c;
    }
}

```

```

Functie functie = new KwadratischeFunctie(1, 12, -36);
double nulpunt = vindNulpunt(a, b, functie);

```



Functie met specifieke parameters

```
class LineaireFunctie implements Functie{
    double m, q;
    public LineaireFunctie(double m, double q) {
        this.m = m;
        this.q = q;
    }
    public double f(double x) {
        return m * x + q;
    }
}
```

```
class Functie1 extends LineaireFunctie{
    Functie1() {
        super(2, -6);
    }
}
```

```
class KwadratischeFunctie implements Functie{
    double a, b, c;
    public KwadratischeFunctie(double a, double b, double c){
        this.a = a;
        this.b = b;
        this.c = c;
    }
    public double f(double x) {
        return a * x * x + b * x + c;
    }
}
```

```
class Functie2 extends KwadratischeFunctie{
    Functie2(){
        super(1, 12, -36);
    }
}
```

Lambda-expressies

- ◆ Sinds Java 8 (versienummer 1.8) kan je functies op een eenvoudigere manier meegeven, namelijk als **lambda-expressies**.

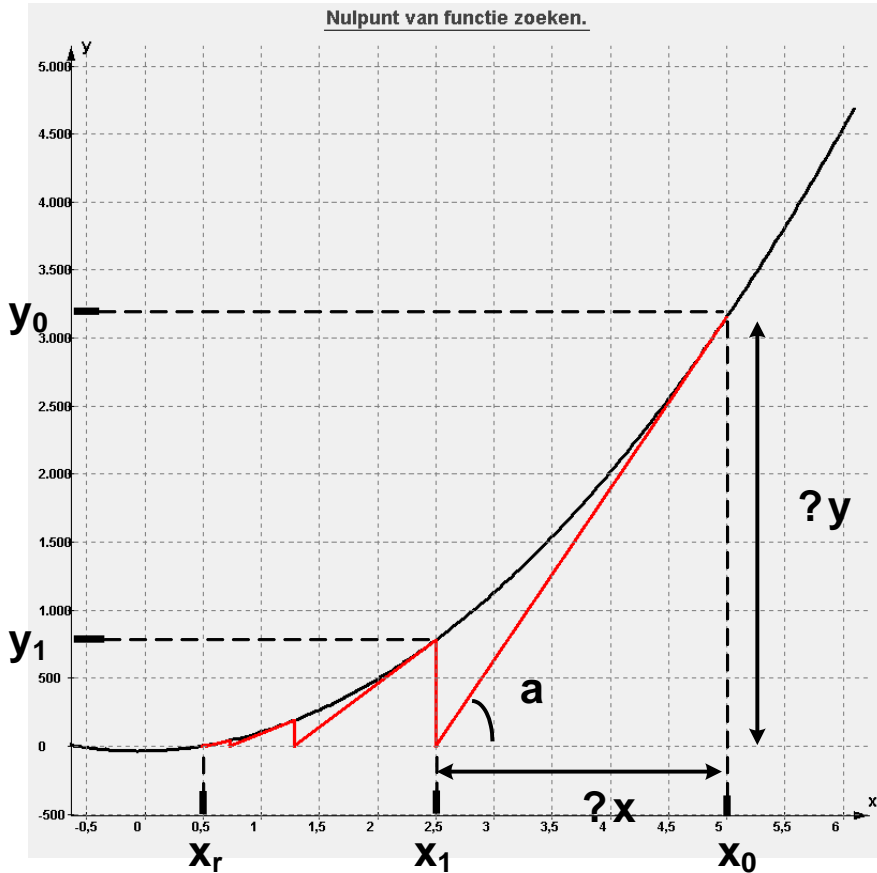
```
double nulpunt = vindNulpunt(a, b, x -> 2*x + x - 3);
```

- ◆ Je definieert ze met een pijl '->': voor de pijl zet je de inputparameters (hier: x), na de pijl zet je de berekening van de output. De Javacompiler checkt dat de gegeven lambda-expressie voldoet aan de gevraagde interface *Functie*, wat hier het geval is!

Newton's algoritme

Nulpunten zoeken: Newton's algoritme

Nulpunt van functie zoeken.



◆ (x_0, y_0) : eerste gok

$$\frac{\Delta y}{\Delta x} = \frac{\partial f}{\partial x} \Leftrightarrow \Delta x = \frac{\Delta y}{\frac{\partial f}{\partial x}} \Leftrightarrow x_1 = x_0 - \frac{\Delta y}{\frac{\partial f}{\partial x}}$$

Afgeleide van functie kennen

```
interface Functie {  
    /** geeft functiewaarde in opgegeven punt */  
    double f(double x);  
}
```

```
interface FunctieMetAfgeleide extends Functie {  
    /** geeft waarde van de afgeleide in opgegeven punt */  
    double afgeleide(double x);  
}
```

```

static int nbrIteraties=0;
public static double vindNulpuntMetNewton(FunctieMetAfgeleide functie,
int eersteGok){
    final int MAX_ITERATIES = 150;
    final double PRECISIE = 0.001;

    double nulpunt = eersteGok;
    double fx = functie.f(nulpunt);
    nbrIteraties=0;
    while (Math.abs(fx) > PRECISIE && nbrIteraties < MAX_ITERATIES){
        double dfx = functie.afgeleide(nulpunt);
        if (dfx == 0)
            throw new RuntimeException("Nulpunt met Newton:
Afgeleide in "+nulpunt+" is nul waardoor Newton faalt.");
        System.out.println("[ "+nbrIteraties+" ] Current =
"+nulpunt+" fx="+fx+" dfx="+dfx+" => new = "+(nulpunt - fx/dfx));
        nulpunt = nulpunt - fx/dfx;
        fx = functie.f(nulpunt);
        nbrIteraties++;
    }
    if (nbrIteraties >= MAX_ITERATIES)
        throw new RuntimeException("Nulpunt met Newton: convergeert
niet, te veel iteraties, beste punt tot nu toe heeft waarde
"+functie.f(nulpunt)+"!");
    System.out.println("Newton nulpunt = "+nulpunt+" met fx="+fx);
    return nulpunt;
}

```

Mogelijke problemen

1. Geen of trage convergentie

- ✦ Gradiënt geeft niet noodzakelijk de juiste richting aan

2. Crash

- ✦ Afgeleide die nul is

3. Verticale afgeleide [Ruben Simons 2019]

- ✦ X-waarde blijft dan dezelfde, we blijven 'hangen in hetzelfde punt'

4. Geen nulpunt voor functie

- ✦ Vraag: *kan het algoritme garanderen dat functie geen nulpunten heeft?*

Discrete Simulations

Discrete simulatie

- ◆ We benaderen de continuë wereld, afgeleiden en integralen benaderen we met Δ 's
- ◆ Wereld veranderen we elke Δt

$$\Delta v_x = a_x \cdot \Delta t$$

$$\Delta v_y = a_y \cdot \Delta t$$

$$\Delta x = v_x \cdot \Delta t$$

$$\Delta y = v_y \cdot \Delta t$$

```

void simulationLoop(double kogelgewicht, double hoek, double vuurkracht){
    final int TIME_STEP = 1;
    final double G = 9.81, FRICTION=0.0001, UNCERTAINTY=0.1;
    Random rand = new Random();

    Kogel kogel = new Kogel(kogelgewicht);
    kogel.vx = vuurkracht * Math.cos(hoek) / kogelgewicht;
    kogel.vy = vuurkracht * Math.sin(hoek) / kogelgewicht;

    int time = 1;
    boolean ended = false;

    while(!ended){
        // forward time
        time += TIME_STEP;

        // update velocity
        double friction = FRICTION * kogel.vy * kogel.vy;
        kogel.vy += - TIMESTEP * G + (kogel.vy < 0 ? friction : -
friction) + (rand.nextDouble() - 0.5) * UNCERTAINTY;

        friction = FRICTION * kogel.vx * kogel.vx;
        kogel.vx += (kogel.vx < 0 ? friction : -friction) +
(rand.nextDouble() - 0.5) * UNCERTAINTY;

        // update position
        kogel.x += kogel.vx * TIMESTEP;
        kogel.y += kogel.vy * TIMESTEP;

        // check collisions
        if (kogel.y < 0)
            ended = true;
    }
}

```

```

class Kogel{
    double massa;
    double x, y, vx, vy;
    Kogel(double massa){
        this.massa = massa;
    }
}

```

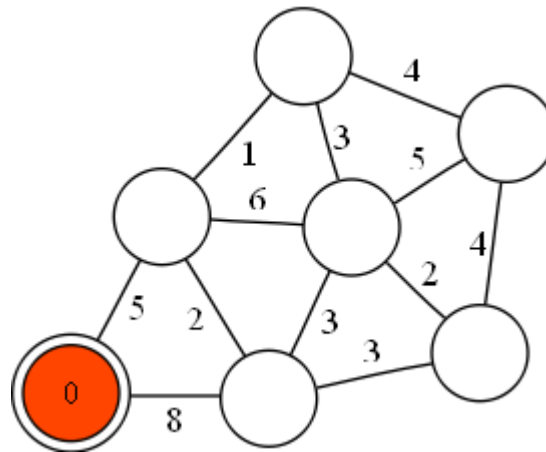
Uitbreiden

- ◆ Alle fysische dingen: MovingObject klassen
- ◆ Met subklassen voor specifiek gedrag

Dijkstra's kortste pad

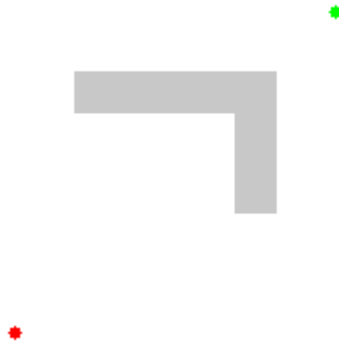
Probleemstelling

- ◆ **Gegeven:** graph waarbij elke link een gewicht heeft

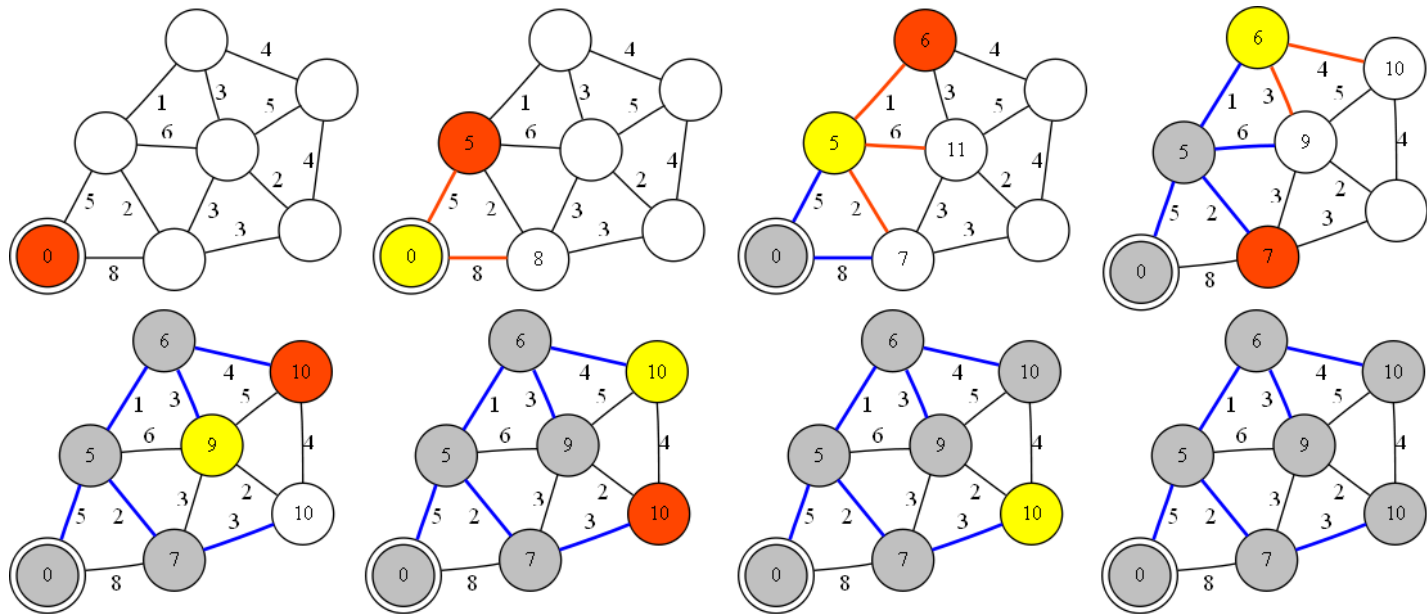


- ◆ **Gevraagd:** kortste pad (minimaliseren som van de gewichten) van elke node naar node 0

Toegepast op een labyrinth



Dijkstra's algoritme



Het algoritme maakt een priority queue van nodes aan, die gesorteerd wordt op afstand (kleinste eerst).

Het algoritme onthoudt per node:

- de tot-dan-toe gevonden afstand naar node 0. Initieel op oneindig gezet.
- de eerstvolgende node die hij moet nemen om naar node 0 te gaan (in blauw aangegeven in graaf).

Afstand van node 0 zet je op 0. De node voeg je toe aan de priority queue.

Doe zolang priority queue niet leeg is:

- Neem eerste element uit priority queue (kortste afstand tot node 0)
- Ga voor deze node al zijn burens af:
 - Tel de afstand tot buur bij je eigen afstand.
 - Kijk of deze afstand beter is dan de tot-dan-toe gevonden afstand van de buur. Indien dit zo is, update de tot-dan-toe gevonden afstand en geef aan dat de buur naar de node moet gaan.
- Als de buur nog niet in de priority queue is, voeg hem toe.