Communication and Multimedia Design | Hogeschool van Amsterdam

tt 25/26

OO puntnotatie, OO libraries en API's

L. Benvenuti

Creating Tomorrow

Object Oriented Programming

Ontstaan in de softwarecrisis

1. Simulaties mogelijk te maken
2. Hergebruik van code mogelijk te maken
   (en dus softwareontwikkeling goedkoper te maken)

Vergelijk software met een fiets – bestaat uit onderdelen. OO Software ook.

Rollenspel!

Dobbelsteen

```
private worp: Number;
```

```
public Dobbelsteen (){

vraag Laura een dobbelsteen en werp hem

worp = 6 * Math.ceil( Math.random() )
}


public werp(){

worp = 6 * Math.ceil( Math.random() )
}

public getWorp(){

            return worp;
}
```

Pion

private kleur:

in*[‘rood’, ‘blauw’, ‘geel’ or ‘groen’]*

private veld:

in [‘quicksand’, ‘tree’, ‘vine’,
‘rock’, ‘bottle’ or ‘fruit’]

```
public Pion(String myKleur) {

    kleur = myKleur;
    veld = startveld bij de kleur;
  }


public getVeld() {

    return veld;
  }


public zet(Number stappen) {

    zet het aantal stappen op het bord;
     veld =
           in [‘quicksand’, ‘tree’, ‘vine’,
               ‘rock’, ‘fruit’]
  }
```

Simulatie van een bordspel

De code bestaat uit 3 soorten objecten: **Dobbelstenen, Pionnen en CodeMonkeyIsland** spellen

```
let rafi = new Dobbelsteen();

let jaïr = new Pion('blauw'),
let jurrien = new Pion('rood'),
let tân= new Pion('geel'),
let jamie = new Pion('groen'),

let myPionnen = [jaïr , jurrien , tân , jamie ];

let christian = new CodeMonkey(myPionnen, rafi );
```

CodeMonkey

```
public  pionnen: [jaïr , jurrien , tân , jamie ]
public  dobbelsteen: rafi
private pionAanZet:
private stappen:
private veld:
```

```
public CodeMonkey( Pion [] myPionnen,
                              Dobbelsteen myDobbelsteen) {
                pionnen = myPionnen;
                dobbelsteen = myDobbelsteen;
                pionAanZet = pionnen[0];
}

public speel(){ //bij simulatie in de les: herhaal dit
                pionAanZet = volgendePion met de klok mee;
                dobbelsteen.werp();

                stappen = dobbelsteen.getWorp();
                pionAanZet.zet(stappen);
                veld = pionAanZet.getVeld();

                if(this.veld == 'quicksand') this.pionAanZet.zet(0);
                if(this.veld == 'tree')        this.pionAanZet.zet(3);
                if(this.veld == 'vine')        this.pionAanZet.zet(6);
                if(this.veld == 'rock')        this.pionAanZet.zet(-2);
                if(this.veld == 'bottle')    pionAanZet.zet(-1);
                if(this.veld == 'fruit')       fruit();
}

private fruit(){
                let resultaat = 0;

                pionnen.forEach( pion => {
                                if (pion.getVeld() == 'tree') {
                                    resultaat = resultaat + 1;
                                }
                };

                .pionAanZet.zet(resultaat);
}
```

Object Oriented Programming

Code Monkeylsland spelen zonder te kijken naar het bord

CodeMonkey

```
public  pionnen: [camil, sep, jesse, kim ]
public  dobbelsteen: binc
private pionAanZet: kim
private stappen: 3
private veld: fruit
```

```
public CodeMonkey( Pion [] myPionnen,
                              Dobbelsteen myDobbelsteen) {
                pionnen = myPionnen;
                dobbelsteen = myDobbelsteen;
                pionAanZet = pionnen[0];
}

public speel(){ //bij simulatie in de les: herhaal dit
                pionAanZet = volgendePion met de klok mee;
                dobbelsteen.werp();

                stappen = dobbelsteen.getWorp();
                pionAanZet.zet(stappen);
                veld = pionAanZet.getVeld();

                if(this.veld == 'quicksand') this.pionAanZet.zet(0);
                if(this.veld == 'tree')        this.pionAanZet.zet(3);
                if(this.veld == 'vine')        this.pionAanZet.zet(6);
                if(this.veld == 'rock')        this.pionAanZet.zet(-2);
                if(this.veld == 'bottle')    pionAanZet.zet(-1);
                if(this.veld == 'fruit')       fruit();
}

private fruit(){
                let resultaat = 0;

                pionnen.forEach( pion => {
                                if (pion.getVeld() == 'tree') {
                                    resultaat = resultaat + 1;
                                }
                };

                .pionAanZet.zet(resultaat);
}
```

Object Oriented Programming: roundup

- Objecten zijn kleine onafhankelijke systemen

- Wij herkennen ze aan de “.” (punt) notatie, als in myArray.length of console.log()

- Object Oriented Programming maakt hergebruik mogelijk
  Alle Pionnen, Dobbelstenen hebben dezelfde code:
  1x code schrijven, maximaal hergebruik

- Ieder object heeft zijn eigen verantwoordelijkheid:
  - voor het beheer van eigen data
  - Voor het beantwoorden van “verzoeken” van andere objecten

- Door met elkaar te communiceren, voeren objecten complexe handelingen uit

Object Oriented Programming

- Hergebruik van code door developers:

  Libraries en reference guides

- Gebruik van systemen door users en/of door andere systemen:

  API’s

(1) Hergebruik van code: libraries en reference guides

- Stel je voor, dat je een ganzenbord wil programmeren, of een mens-erger-je-niet

- Dan wil je Pion-objecten en Dobbelsteen-objecten hergebruiken

-> Dobbelsteen-objecten zijn verantwoordelijk voor het genereren van toevalsgetallen tussen 1 en 6

-> Pion-objecten zijn verantwoordelijk voor het beheer van de positie op het bord

- Welke informatie heb je daarvoor nodig?

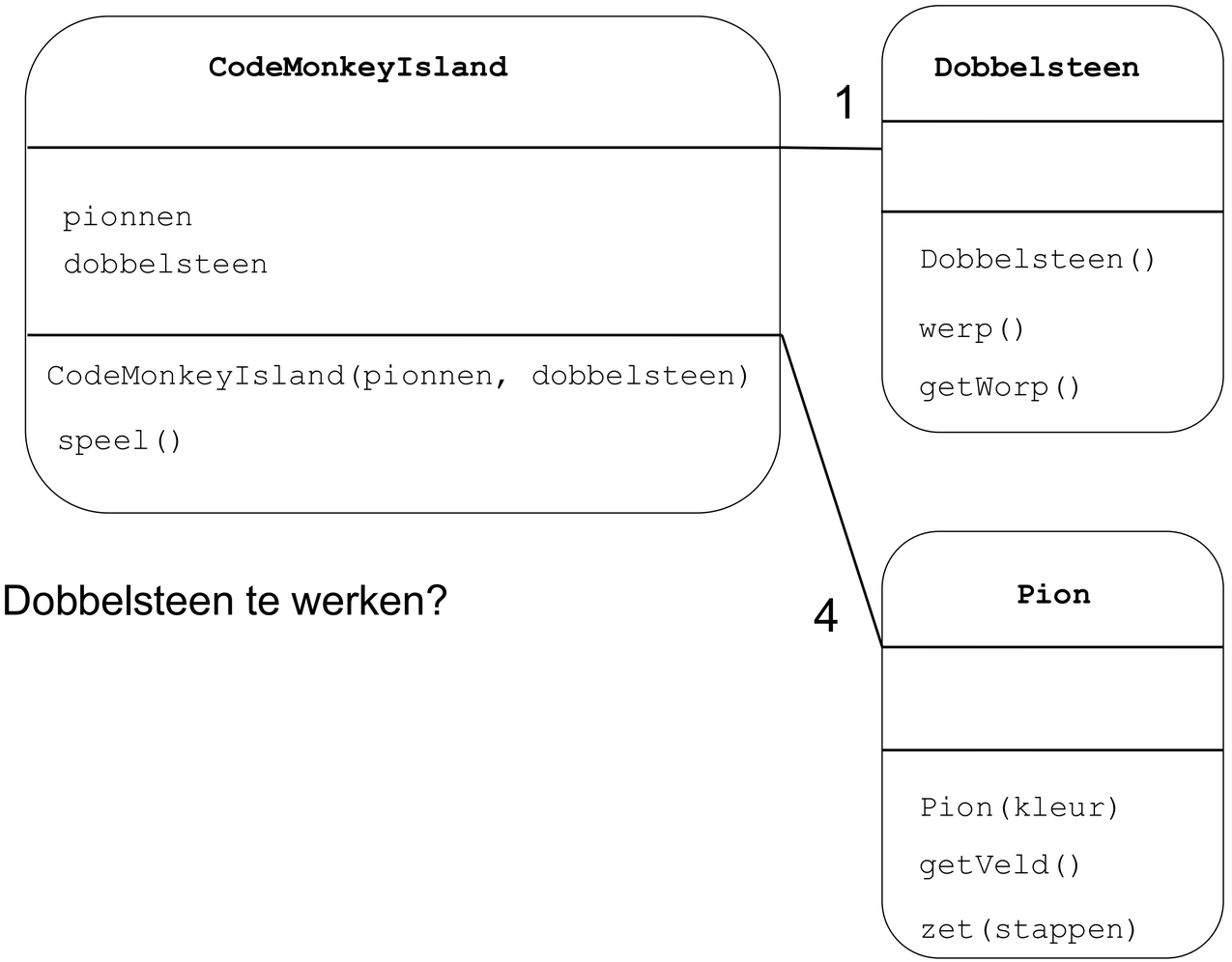
- De informatie die de developer nodig heeft om code te schrijven die deze objecten gebruikt.
  vind je in reference guides als MDN, d3

Pion

```
 / Constructor
// myKleur: de kleur van de Pion
public Pion(String myKleur) {

   }

// methode getVeld() geeft informatie van het veld waarop
// de Pion staat
// return: een String met het symbool op het veld
public getVeld() {

   }


// methode zet() verplaatst de Pion op het bord
// stappen: het aantal stappen dat de Pion moet
// zetten
public zet(Number stappen) {

   }
```

Dobbelsteen

```
// Constructor
public Dobbelsteen() {
     }


// methode werp() werpt de Dobbelsteen
public werp(){

}


// methode getWorp() haalt de laatste worp van de
// Dobbelsteen op.
// return: de laatste worp van de Dobbelsteen
public getWorp(){

     }
```



Hergebruik van code

Wat moeten developers weten om met objecten van het type Dobbelsteen te werken?

Dobbelsteen

Representeert een zuivere, 6-zijdige dobbelsteen

Attributen

Methoden:

Dobbelsteen() – constructor, hiermee maak je een nieuwe Dobbelsten aan

werp() – simuleert het eenmaal werpen van een dobbelsteen

getWorp() – geeft de laatste worp terug

return: Number uit {1,2,3,4,5,6}

Reference Guide (als MDN, d3)
Doelgroep: andere programmeurs

Willen weten:
- welke objecten mag ik gebruiken?
- wat zijn hun verantwoordelijkheden?
- welke methoden mag ik aanroepen?
- welke parameters hebben die methoden?
- Wat geven die methoden terug?

CodeMonkeyIsland
Bewaakt de spelregels van het bordspel

Attributen
pionnen - array objecten van het type Pion
dobbelsteen - Dobbelsteen

Methoden
CodeMonkeyIsland() – consturctor, heeft
parameters: pionnen - array objecten van het type Pion
dobbelsteen - Dobbelsteen

speel() – methode die de ‘beurt’ van volgende speler uitvoert

Pion
Beeldt de toestand van de pion af op het bord

Attributen

Methoden:
Pion() – constructor, heeft
parameter: kleur - String uit {‘rood’, ’blauw’, ‘geel’, ‘groen}

getVeld() – geeft informatie terug over de toestand van de Pion
return: String

zet() – verandert de toestand van de Pion, heeft
parameter: stappen - Number.

Dobbelsteen
Representeert een zuivere 6-zijdige dobbelsteen

Attributen

Methoden:
Dobbelsteenr() - constructor

werp() – simuleert het eenmaal werpen van een dobbelsteen

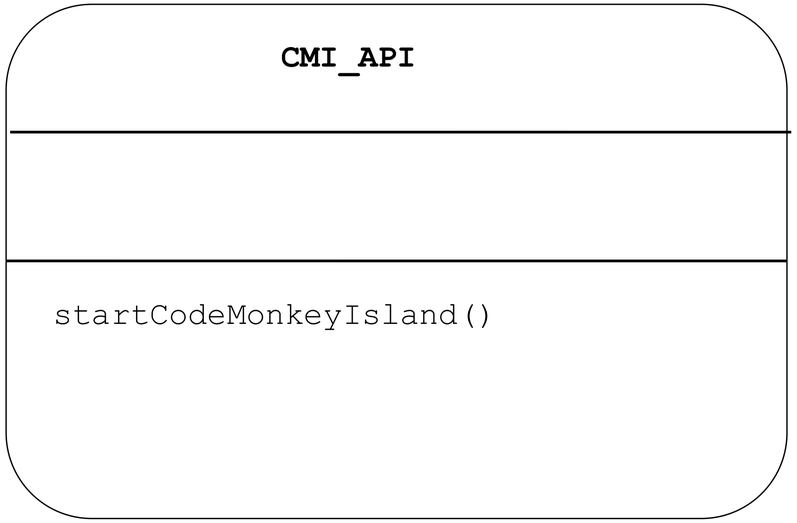
getWorp() — geeft de laatste worp terug
return: Number

(2) Gebruik van systemen voor derden: API's

- Stel, ie wil mensen toestaan om een spel CodeMonkeyIsland te starten, zonder dat zij kunnen programmeren

- Het script staat op slide 5:

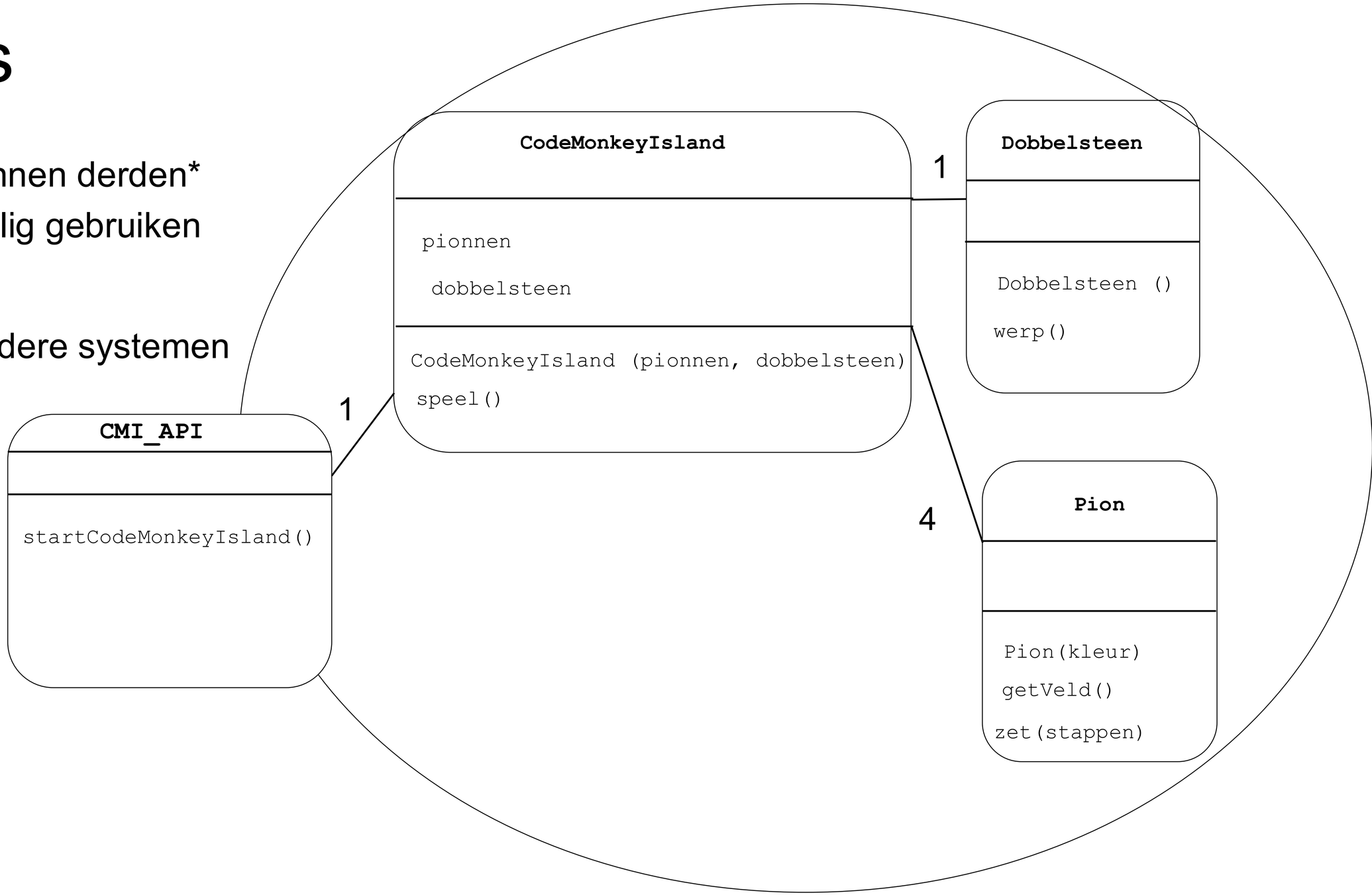
```
public startCodeMonkeyIsland(){

    let myDobbelsteen = new Dobbelsteen();

    let eersteSpeler = new Pion('blauw')
    let tweedeSpeler = new Pion('rood')
    let derdeSpeler = new Pion('geel')
    let vierdeSpeler = new Pion('groen')

    let myPionnen = [   eersteSpeler,
                        tweedeSpeler,
                        derdeSpeler,
                        vierdeSpeler
                    ]

    let spel = new CodeMonkey(myPionnen,
                              myDobbelsteen)
}
```



(2) API’s

Via een API kunnen derden*
het systeem veilig gebruiken

* mensen of andere systemen



Object Oriented Programming: roundup

- Ontstaan in de softwarecrisis om hergebruik van software mogelijk te maken
- Objecten zijn kleine systemen, te herkennen aan de “.” (punt)notatie
- Die zijn geschreven door andere programmeurs en worden aangeboden in libraries als d3
- Ieder object heeft zijn eigen verantwoordelijkheid
- API’s zijn objecten die als verantwoordelijkheid hebben: de communicatie met het systeem (door users en/of andere systemen) veilig te laten verlopen

Vragen?