SEB BP2: MODULE 4:

Dit is alweer het laatste stuk theorie voor dit gedeelte van het semester. De nieuwe stof is in een razend tempo aan je voorbijgegaan. Wanneer nodig kun je het beste delen van de vier modules nog een keer doornemen.

In deze laatste module gaan we in op het complexere overerven tussen classes. Ook kijken we naar hoe we interfaces tussen classes het beste kunnen bouwen. Dit is eigenlijk het enige nieuwe wat we in deze module behandelen. Er zijn nog verschillende oefenopgaves opgenomen die je kunt doornemen als oefening. Want nog steeds geldt, ook voor object georiënteerd programmeren, oefening baart kunst.

THEORIE

SCREENCAST OVER ONDERWERP 9: COMPLEXE OVERERVINGSSTRUCTUUR

http://www.youtube.com/playlist?list=PLpd9jJvk1Pjk9_ObEgzxyHz_WfiY8BzxE

OPTIONEEL: PLURALSIGHT

Herhaling: Class InheritanceL

https://app.pluralsight.com/player?course=java-fundamentals-language&author=jim-wilson&name=java-fundamentals-language-m8&clip=0&mode=live

Eventueel als herhaling: UML Class Diagrams:

https://app.pluralsight.com/player?course=uml-introduction&author=mike-erickson&name=uml-introduction-m3-structural-diagrams&clip=1&mode=live

OPTIONEEL: BOEK

HOOFDSTUK 10

10.5, pagina 386 t/m 389

OPGAVE SWITCH KLASSENDIAGRAM

Maak een klassendiagram met de klassen Licht, Knop, Switch en LichtSwitch. Je hoeft alleen de methoden handelInteractieAf en doeKnopActie op te nemen in het diagram. Geef wel duidelijk aan welke methoden abstract zijn en welke niet.

OPGAVE FOUTIEVE TOESTAND

Wanneer je op de LichtKnop drukt en het licht aanzet dan komt de LichtSwitch in een foutieve toestand (d.w.z. als je naar de switch kijkt, krijg je de indruk dat het licht niet brandt (hij staat immers “uit”), terwijl het licht wel aan is). Zorg ervoor dat de LichtSwitch aan gaat als het licht via de LichtKnop aangezet wordt. Probeer zo effectief mogelijk gebruik te maken van de klassen die al gemaakt zijn. Gebruik de startcode die op #OnderwijsOnline gegeven is (hieruit is alles dat met Geluid te maken heeft verwijderd).

OPGAVE PERSONENLIJST KLASSENDIAGRAM

Gegeven onderstaande drie klassen

```
public class Persoon {
     protected String naam, voornaam;

     public Persoon(String naam, String voornaam) {
          this.naam = naam;
          this.voornaam = voornaam;
     }

     public String getNaam() {
          return naam;
     }

     public void setNaam(String naam) {
          this.naam = naam;
     }

     public String getVoornaam() {
          return voornaam;
     }

     public void setVoornaam(String voornaam) {
          this.voornaam = voornaam;
     }
}


public class Docent extends Persoon {
     protected String code;

     public Docent(String naam, String voornaam, String code) {
          super(naam, voornaam);
          this.code = code;
     }

     public String getCode() {
          return code;
     }

     public void setCode(String code) {
          this.code = code;
     }
}


public class Student extends Persoon {
     protected int nummer;
     protected Docent SLBer;

     public Student(String naam, String voornaam, int nummer, Docent SLBer)
{
          super(naam, voornaam);
          this.nummer = nummer;
```

```
        this.SLBer = SLBer;
    }

    public int getNummer() {
        return nummer;
    }

    public void setNummer(int nummer) {
        this.nummer = nummer;
    }

    public Persoon getSLBer() {
        return SLBer;
    }

    public void setSLBer(Docent sLBer) {
        SLBer = sLBer;
    }
}
```

OPGAVE

Maak een klassendiagram van deze code. De methoden uit de klassen hoef je er niet in op te nemen.

OEFENINGEN

OPGAVE PERSONENLIJST IMPLEMENTATIE

Gegeven onderstaande code en het klassendiagram dat je in de voorbereiding gemaakt hebt.

```
public class Persoon {
    protected String naam, voornaam;

    public Persoon(String naam, String voornaam) {
        this.naam = naam;
        this.voornaam = voornaam;
    }

    public String getNaam() {
        return naam;
    }

    public void setNaam(String naam) {
        this.naam = naam;
    }

    public String getVoornaam() {
        return voornaam;
    }

    public void setVoornaam(String voornaam) {
        this.voornaam = voornaam;
    }
```

```
}

public class Docent extends Persoon {
     protected String code;

     public Docent(String naam, String voornaam, String code) {
          super(naam, voornaam);
          this.code = code;
     }

     public String getCode() {
          return code;
     }

     public void setCode(String code) {
          this.code = code;
     }
}


public class Student extends Persoon {
     protected int nummer;
     protected Docent SLBer;

     public Student(String naam, String voornaam, int nummer, Docent SLBer)
{
          super(naam, voornaam);
          this.nummer = nummer;
          this.SLBer = SLBer;
     }

     public int getNummer() {
          return nummer;
     }

     public void setNummer(int nummer) {
          this.nummer = nummer;
     }

     public Docent getSLBer() {
          return SLBer;
     }

     public void setSLBer(Docent sLBer) {
          SLBer = sLBer;
     }
}
```

OPGAVE A

In de klasse Student is een veld opgenomen met de naam SLBer en het type Docent. Dit veld wijst naar de docent die de SLBer is van de betreffende student. Nu zijn er bij ICA ook SLB'ers die geen docent zijn, dus we kunnen dit ontwerp niet handhaven. Kun je het type van het veld SLBer nu beter veranderen naar:

1. String of
2. Persoon?

Beargumenteer beide opties.

OPGAVE B

Implementeer in elke klasse de methode toString (zie ook eerdere opgaves uit vorige modules). Deze methode moet de naam en de waarde van elk veld uit de klasse teruggeven als String. De toString uit Docent en Student moet ook de naam en de waarde van elk veld uit de klasse Persoon teruggeven.

Genereer deze methoden met eclipse (Source > Generate toString()) en test de methoden in het hoofdprogramma.

OPGAVE C

Gegeven het onderstaande programma:

```
import java.util.ArrayList;

public class PersonenLijst {
     private ArrayList<Persoon> lijst;

     public PersonenLijst() {
          lijst = new ArrayList<>();
     }

     public ArrayList<Student> getSLBStudenten(Docent SLBer) {
     }

     public static void main(String[] args) {
          PersonenLijst p = new PersonenLijst();
          Docent piet=new Docent("Piet","Jansen","jnsnp");
          Student marie=new Student("Marie","Pieters",31415,piet);
         Student jan=new Student("Jan","de Vries",92653,null); // nog geen
SLB'er
         p.lijst.add(piet);
         p.lijst.add(marie);
         p.lijst.add(jan);
        // voeg zelf nog enkele docenten en studenten toe
         ArrayList<Student> studentenVanPiet=p.getSLBStudenten(piet);

     }
}
```

Implementeer de methode getSLBStudenten (regel 10). Deze methode krijgt een instantie van Docent mee en retourneert een lijst van alle SLB-studenten. Test deze methode in main door de voorbeeldcode hierboven verder uit te breiden en de arraylist aan het einde te printen.

Hint: bij de implementatie kun je gebruik maken van de instanceof operator (Google maar eens “instanceof operator”) en van casten (zou bekend moeten zijn anders Google “Java casten”).

OPGAVE PERSONENLIJST ZONDER CAST

Casten is meestal onwenselijk. Door de klasse Persoon abstract te maken en een abstracte methode getSLBer te maken kunnen we van deze cast afkomen.

OPGAVE A

Implementeer deze oplossing en omschrijf zo exact mogelijk wat de consequentie is van deze beslissing.

OPGAVE B

Is deze oplossing wenselijk, of kunnen we beter gebruik maken van een cast. Beargumenteer het antwoord.

INTERFACES

THEORIE

SCREENCAST OVER INTERFACES

http://www.youtube.com/playlist?list=PLpd9jJvk1PjnR8YVs3ZsZtJoGbtJLSUV

OPTIONEEL: PLURALSIGHT

Creating abstract relationsships with interfaces:

https://app.pluralsight.com/player?course=java-fundamentals-language&author=jim-wilson&name=java-fundamentals-language-m12&clip=0&mode=live

OPTIONEEL: BOEK

HOOFDSTUK 10

10.6, 10.7 en 10.9 pagina 389 t/m 398

OPGAVE HET WOORD INTERFACE

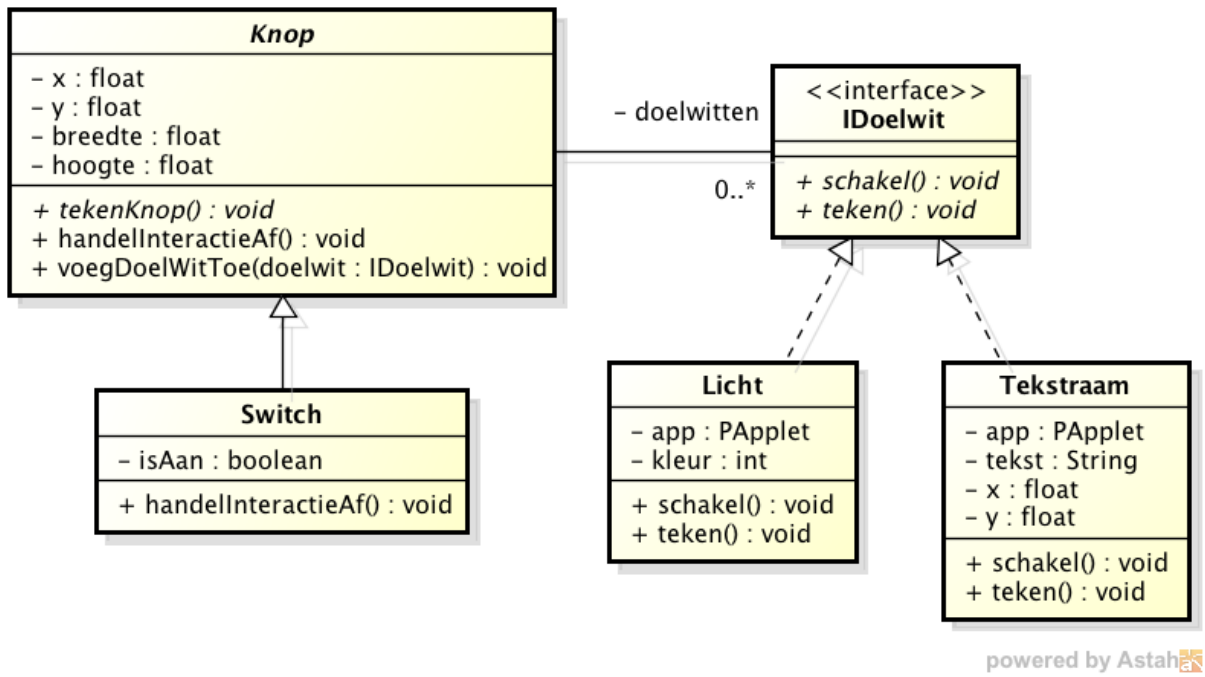
Bij het maken van (objectgeörienteerde) programma's komt de term "interface" geregeld voorbij. Geef drie mogelijke betekenissen van het woord "interface".

OPGAVE MEERDERE INTERFACES IMPLEMENTEREN

Zoals in de screencast wordt verteld, is het in Java niet toegestaan om van meerdere klasses te erven ("multiple inheritance" bestaat in Java niet), maar mag je wel meerdere interfaces implementeren en daarnaast zelfs nog van een klasse erven. Waarom is multiple inheritance wel een probleem, maar het implementeren van meerdere interfaces niet?

OPGAVE MEERDERE DOELWITTEN

We starten met een programma dat lijkt op het programma uit de screencast, maar met een paar wijzigingen. Je vindt de startcode op onderwijsonline en het klassediagram staat hieronder.



Een knop kan nu meerdere IDoelwitten bevatten. Zodra er op de knop geklikt wordt, moeten alle gekoppelde doelwitten de methode schakel aanroepen.

De klasse Tekstraam plaats een tekst op het scherm als de methode schakel aangeroepen wordt.

OPDRACHT A

Implementeer deze klassen en laat zien dat het programma werkt door een instantie van Licht en Tekstraam aan een instantie van Switch te koppelen in het hoofdprogramma. Je kunt om te testen de gegeven code in de klasse “KnoppenApp” gebruiken: hierin wordt een switch gemaakt waaraan een instantie van licht en een instantie van doelwit worden gekoppeld.

OPGAVE B

Voeg een tweede switch toe waaraan dezelfde instantie van licht is gekoppeld als aan de switch uit opgave A. Wat voor probleem kan nu optreden?

OPGAVE VAN IF-ELSE NAAR INTERFACE

Gegeven een applicatie waarmee rechthoeken en cirkels getekend geplaatst en verwijderd kunnen worden.

```
import java.util.ArrayList;
import processing.core.PApplet;

@SuppressWarnings("serial")
public class TekenApp extends PApplet {

     public static void main(String[] args) {
          PApplet.main("week5.ifelsenaarinterface.TekenApp");
     }

     private ArrayList<Figuur> figurenLijst = new ArrayList<>();
     private String huidigGereedschap = "selecteer";
```

```
    public void setup() {
        size(400, 400);
    }

    public void draw() {
        background(0);
        for (Figuur figuur : figurenLijst) {
            figuur.teken(this);
        }
    }

    public void mousePressed() {
        switch (huidigGereedschap) {
            case "selecteer":
                System.out.println("s");
                break;
            case "rechthoek":
                Rechthoek r = new Rechthoek( mouseX, mouseY, 50, 50);
                figurenLijst.add(r);
                break;
            case "cirkel":
                Cirkel c = new Cirkel(mouseX, mouseY, 50);
                figurenLijst.add(c);
                break;
            case "gum":
                for (int i = figurenLijst.size() - 1; i >= 0; i--) {
                    Figuur fig = figurenLijst.get(i);
                    if (fig.isMuisBinnen(mouseX, mouseY)) {
                        figurenLijst.remove(i);
                    }
                }
                break;
        }
    }

    public void keyReleased() {
        switch (key) {
            case 's':
                huidigGereedschap = "selecteer";
                break;
            case 'r':
                huidigGereedschap = "rechthoek";
                break;
            case 'c':
                huidigGereedschap = "cirkel";
                break;
            case 'g':
                huidigGereedschap = "gum";
                break;
        }
    }
}
```

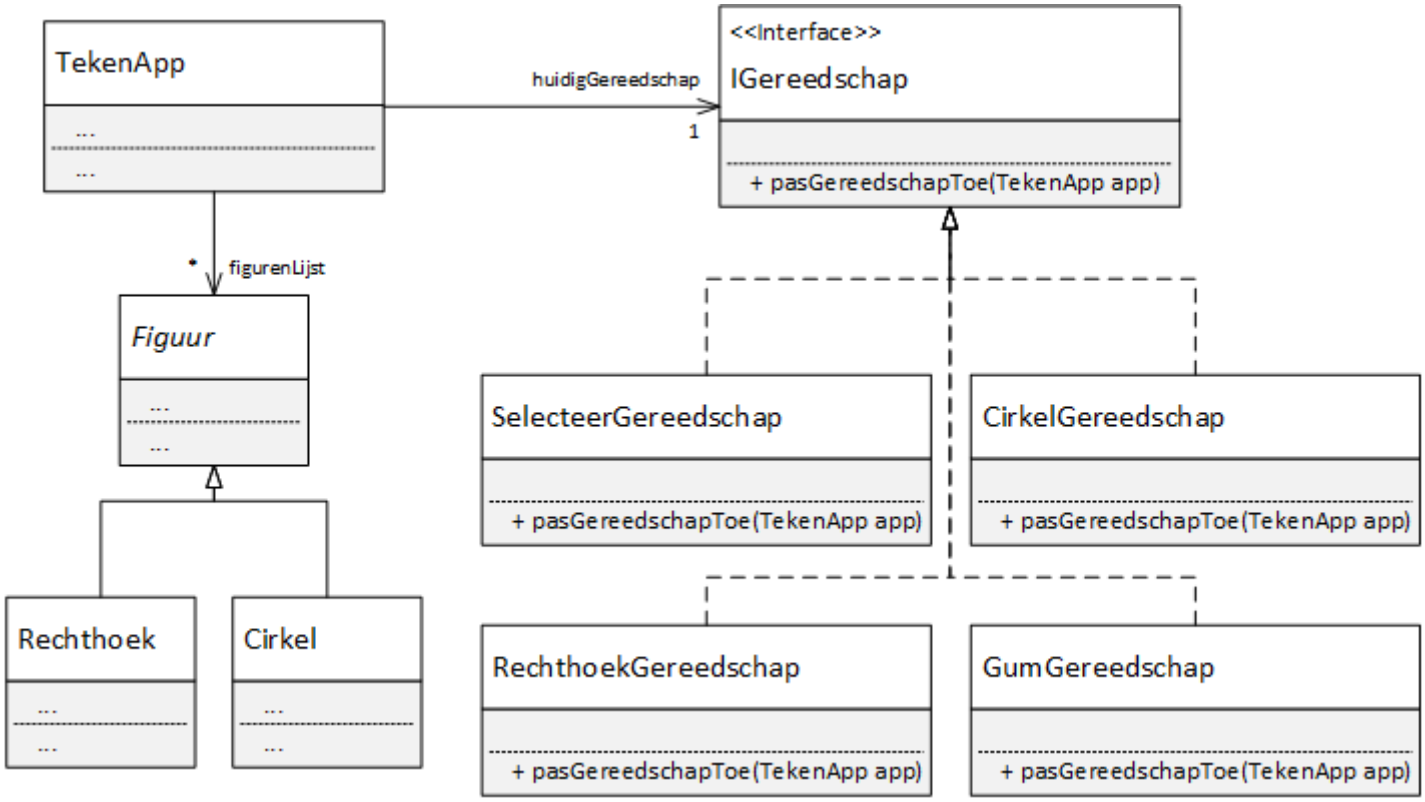
Hoewel deze code werkt, is het uitbreiden ervan lastig. Het probleem zit hem in het switch-statement in de methode mousePressed().

Zodra er een nieuw gereedschap toegevoegd moet worden, moet het gedrag van dit gereedschap toegevoegd worden op een onoverzichtelijk plek. Daarnaast moet de naam van dit gereedschap op meerdere plekken gebruikt, zonder dat de compiler kan controleren of die naam goed geschreven is.

In deze opgave ga je met behulp van een interface dit switch-statement wegwerken.

OPDRACHT A

Hieronder is een klassendiagram te zien waarin een oplossing geboden wordt. Schrijf de code voor de Interface IGereedschap en voor alle gereedschappen die deze interface implementeren. Verplaatst de code uit mousePressed naar de juiste klassen.



OPDRACHT B

Verander in het hoofdprogramma de code uit keyReleased() en mousPressed zodanig dat er optimaal gebruik wordt gemaakt van alle gereedschapsklasse.

Als het goed is bestaat de code in mousePressed() na aanpassing uit slechts één regel. Overigens hoef je het switch-statement uit keyReleased() alleen aan te passen. Weg werken van dit statement is niet nodig

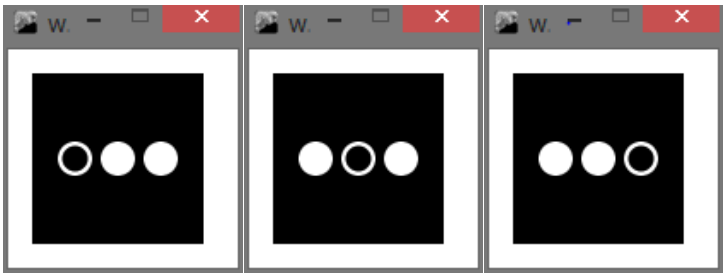
OEFENINGEN

OPDRACHT RADIOKNOPPEN

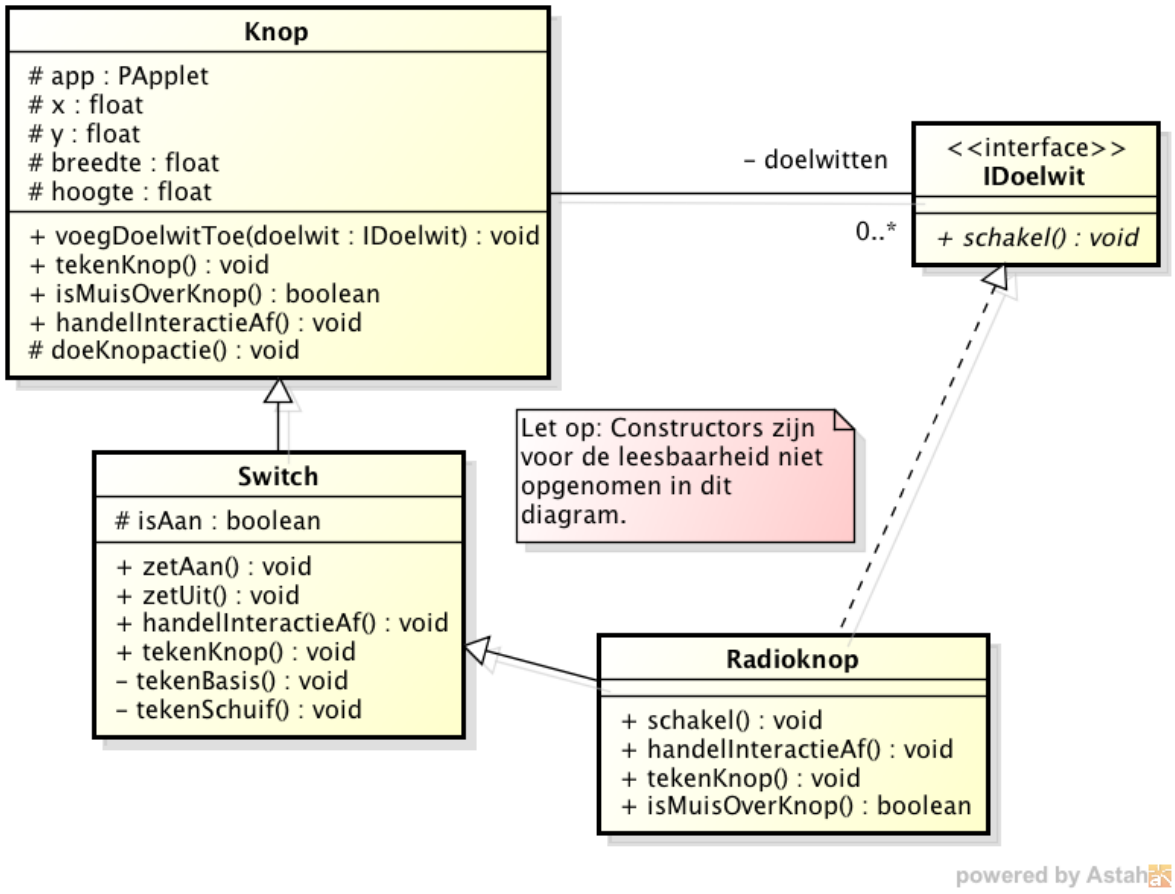
In deze oefening gaan we de implementatie van een interface combineren met een associatie van dezelfde interface binnen één klasse.

Gekoppelde radioknoppen zijn knoppen, waarvan er maar één tegelijkertijd aan kan staan.

Zie onderstaande screenshots voor een voorbeeld. Je ziet hierop steeds drie radioknoppen.



In deze opdracht ga je de klasse Radioknop implementeren door deze te laten erven van Switch en de interface IDoelwit te laten implementeren. Zie onderstaand klassendiagram.



OPGAVE A

Leg uit welke mogelijkheden Radioknop krijgt door:

- te erven van Switch
- de interface IDoelwit te implementeren

OPGAVE B

Implementeer de klasse Radioknop volgens bovenstaande specificatie. Voor de methoden tekenKnop en isMuisOverKnop in deze klasse, kun je onderstaande code gebruiken

```
@Override
public void tekenKnop() {
     app.ellipseMode(PApplet.CENTER);
     app.noStroke();
     app.fill(255);
     app.ellipse(x, y, breedte, hoogte);
     if (isAan) {
          app.fill(0);
          app.ellipse(x, y, breedte - breedte/4, hoogte - hoogte/4);
     }
}

@Override
public boolean isMuisOverKnop() {
     if (PApplet.dist(app.mouseX, app.mouseY, x, y) < breedte/2) {
          return true;
     }
     else {
          return false;
     }
}
```

OPGAVE C

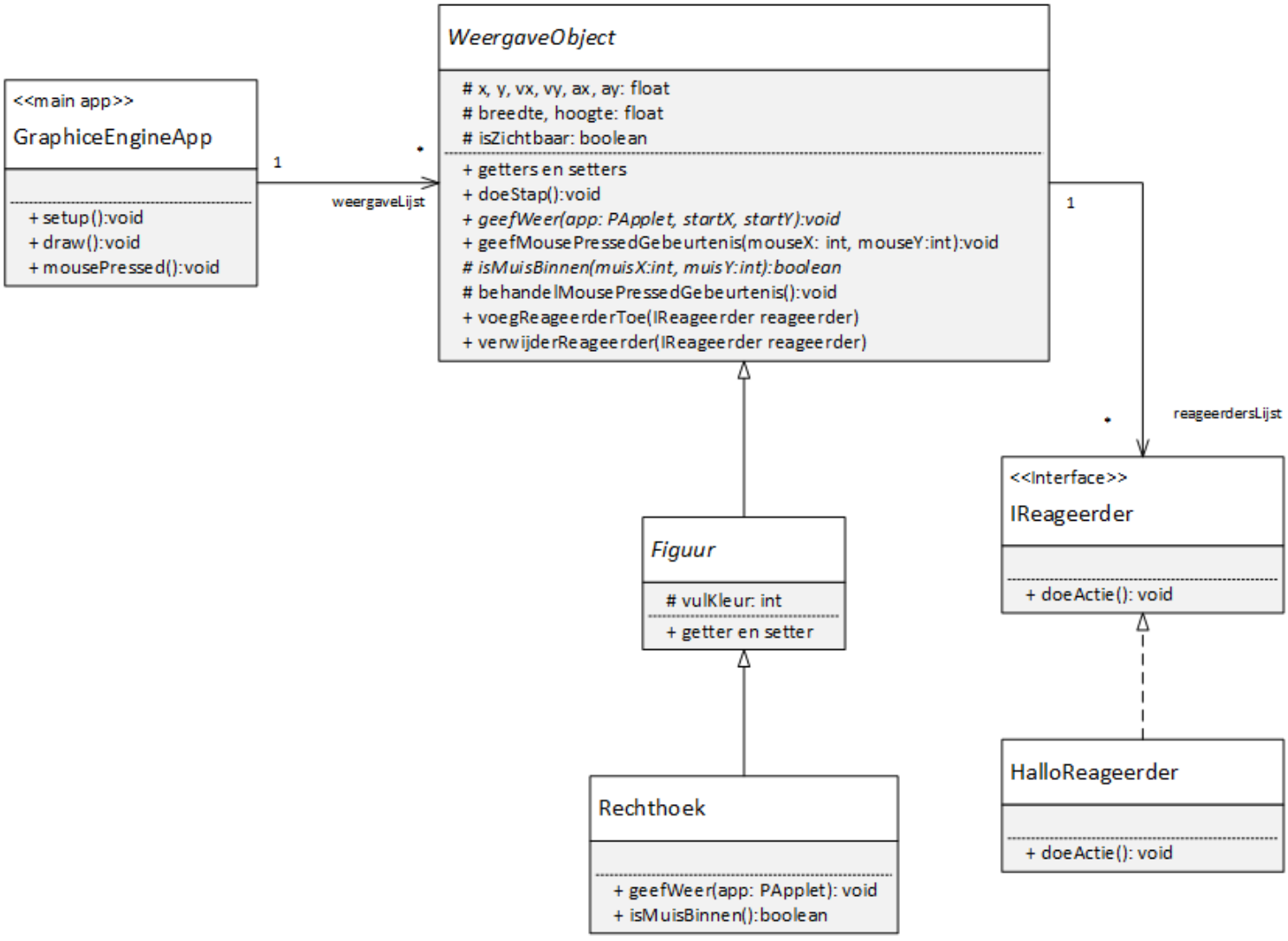
Test de klassen in het hoofdprogramma door een ArrayList met drie instanties van RadioKnop te maken. Uiteraard moet je voor elke radioknop de andere twee radioknoppen als doelwit toevoegen. Zorg dat er in totaal drie radioknoppen in het hoofdprogramma zijn die allen aan elkaar gekoppeld zijn (d.w.z. dat van elk van de drie radioknoppen de twee andere radioknoppen als doelwit zijn toegevoegd).

OPGAVE GRAPHICSENGINE

In deze opgave ga je een graphics engine ontwerpen. Hierbij komen een groot aantal technieken die je hiervoor hebt geleerd terug.

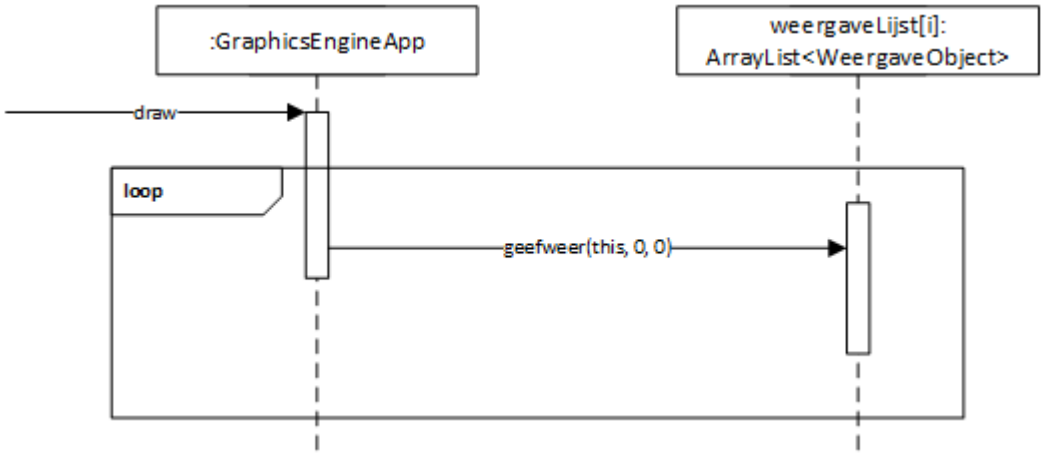
OMSCHRIJVING

Bekijk onderstaande klassendiagram.



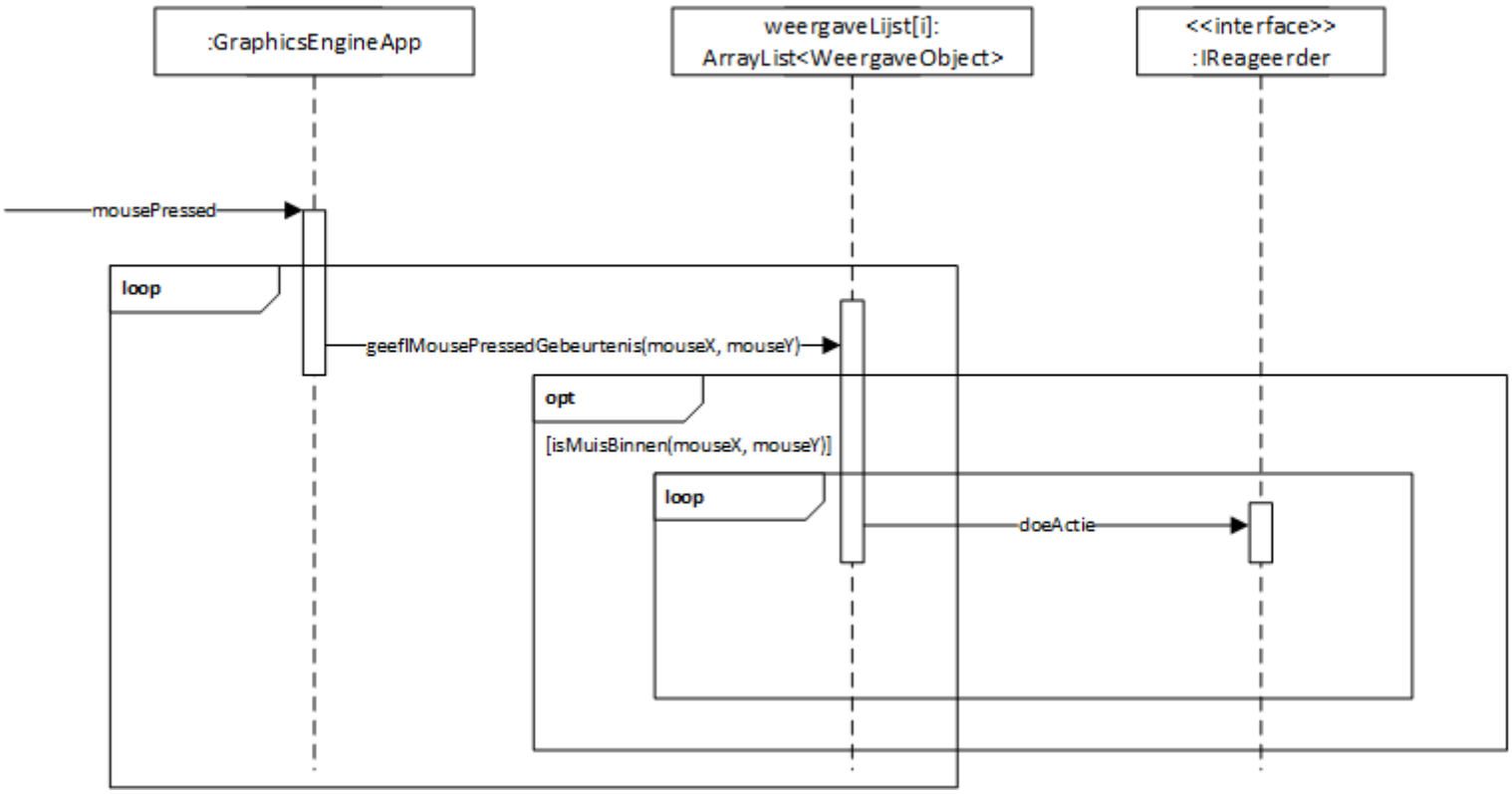
De GraphicsEngineApp houdt een lijst van Weergaveobject bij. In de draw methode van GraphicEngineApp wordt elk WeergaveObject getekend. Dit gebeurd door de methode geefWeer aan te roepen (die abstract is in WeergaveObject).

Zie onderstaand sequentiediagram. De parameters startX en startY kun je gebruiken als je het nulpunt van de PApplet waarop je de figuren tekent niet in de linkerbovenhoek wilt hebben. Deze twee parameters hebben we echt nodig als we deze applicatie gaan uitbreiden in de les.



Een WeergaveObject houdt een lijst van IReageerder bij. Er kunnen IReageerderobjecten worden toegevoegd en verwijderd.

Zodra er geklikt wordt op een WeergaveObject, voert dit object de doeActie uit van elke IReageerder die eraan gekoppeld is. Zie ook onderstaand sequentiediagram.



De HalloReageerder is een voorbeeld van een klasse die IReageerder implementeert. Deze Reageerder kan de String “hallo wereld” naar de console schrijven.

OPGAVE A

Maak een nieuw project in Eclipse en maak de klassen en de interface uit het klassendiagram. Veel methoden heb je al in eerdere projecten gemaakt. Kopieer en plak de code uit de oude projecten naar de nieuwe klassen en test of het programma werkt.

OPGAVE B

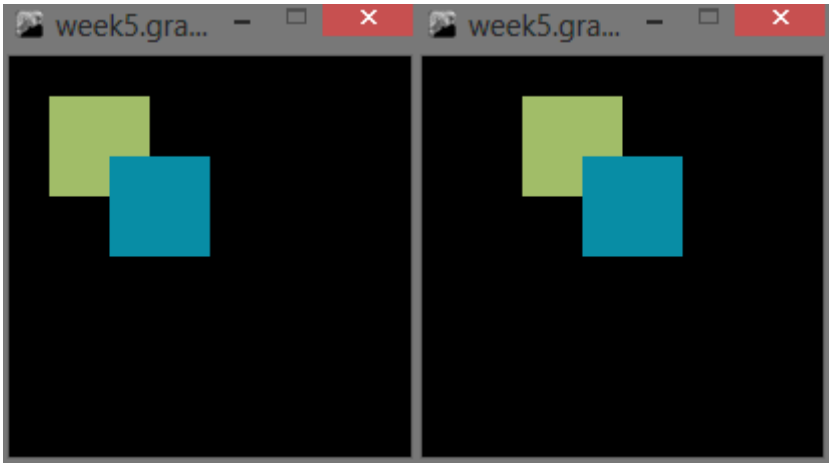
Implementeer de andere methoden op basis van bovenstaande beschrijving en test het programma door een instantie van Rechthoek te maken en deze te koppelen aan een HalloReageerder.

UITDAGING UITBREIDING GRAPHICSENGINE

OPGAVE A: CONTAINERS

De engine wordt interessanter wanneer je weergaveobjecten ook in andere weergaveobjecten kunt stoppen. Op die manier krijg je een ouder-kind relatie. Wanneer je de ouder bijvoorbeeld verplaatst, verplaatsen alle kinderen ook mee.

Een voorbeeld:

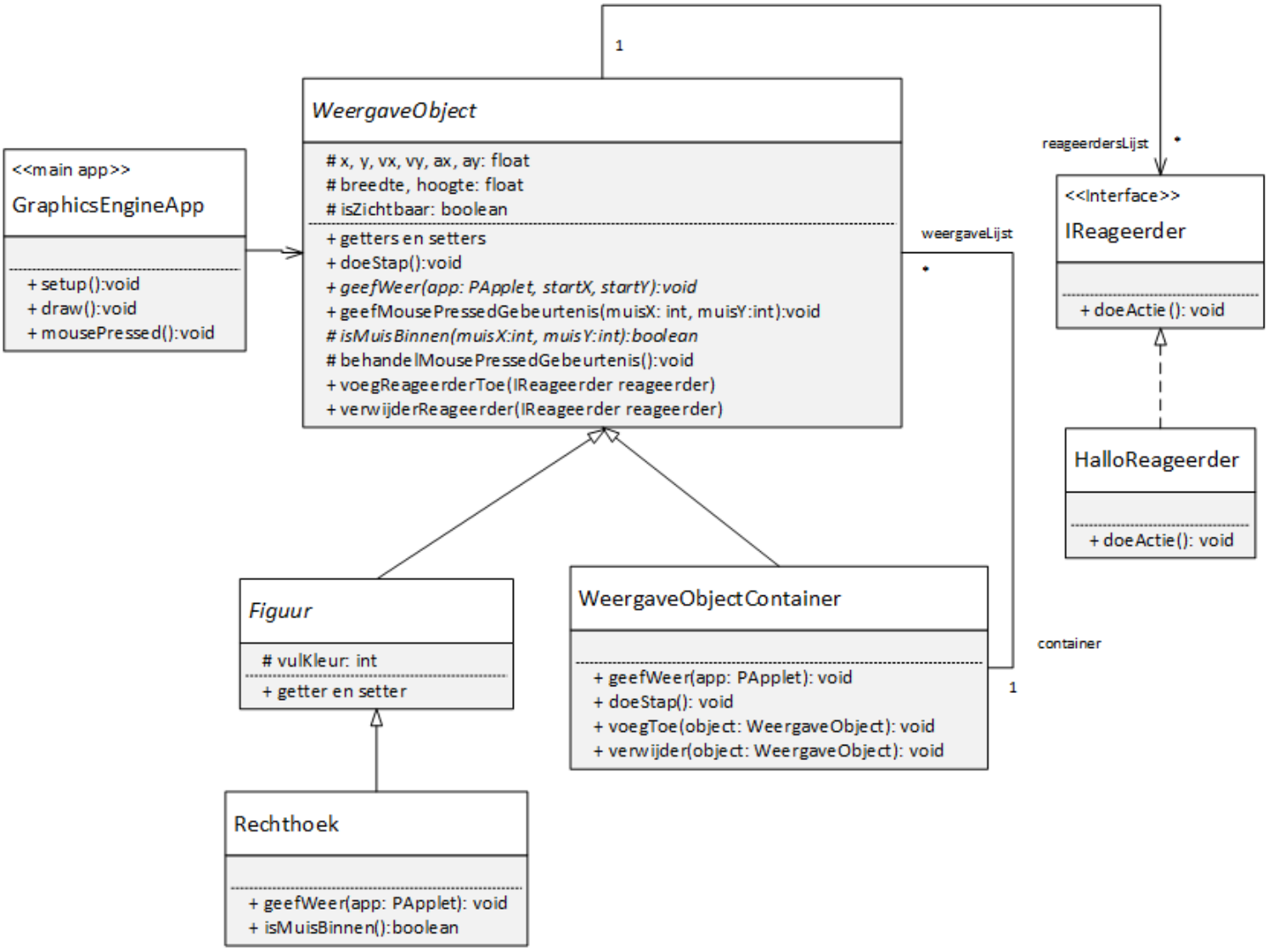


Er is één WeergaveObjectContainer die twee WeergaveObjecten bevat (twee Rechthoeken). In het linkerplaatje hebben de x- en y-coördinaten van de container de waarde 0. In het rechterplaatje is de x-coördinaat gelijk aan 30, waardoor beide rechthoeken ook 30 pixels naar rechts zijn verplaatst.

In deze opgave implementeer je deze mogelijkheid door een nieuwe klasse toe te voegen: WeergaveObjectContainer. Deze klasse kan meerdere instanties van Weergave Object bevatten. Het interessante is dat WeergaveObjectContainer tegelijkertijd een subklasse van WeergaveObject is.

Een WeergaveObjectContainer heeft zelf een breedte en een hoogte van 0.

Hieronder staat het klassendiagram met een mogelijk ontwerp.



OPGAVE A1

Denk eerst na over hoe je de WeergaveObjectContainer gaat testen. Schrijf deze test(en) in de GraphicsEngineApp. Zorg dat alle methoden uit WeergaveObjectContainer worden getest.

Hint: De methode geef weer heeft de parameters startX en startY. Deze kun je gebruiken om ervoor te zorgen dat de x- en de y-coördinaat van alle kinderen van de WeergaveObjectContainer worden aangepast afhankelijk van de x- en de y-coördinaat van de WeergaveObjectContainer zelf.

OPGAVE A2

Maak sequentiediagrammen van de methoden uit WeergaveObjectContainer.

OPGAVE A3

Implementeer de methoden uit WeergaveObjectContainer één voor één en ga pas verder als je zeker weet dat een methode correct werkt.

OPGAVE B: WEERGAVEOBJECTEN MEENEMEN IN REAGEERDERS

Het is jammer dat de een Reageerder niet de beschikking heeft over de informatie van het WeergaveObject dat zijn doeActie() methode aanroept.

OPGAVE B1

Pas de methode doeActie aan zodanig dat een Reageerder wel bij de informatie van een WeergaveObject kan.

OPGAVE B2

Test de nieuwe implementatie door een nieuwe Reageerder te maken die de snelheid van een WeergaveObject te veranderen zodra je op dit WeergaveObject hebt geklikt.

OPGAVE B3

Probeer ook de vulkleur aan te passen met behulp van een nieuwe Reageerder. Waar loop je nu tegenaan?

OPGAVE C (UITDAGING): ALGEMENE GEBEURTENISSEN

Het is jammer dat we alleen kunnen reageren op MousePressedGebeurtenissen. Breidt het Gebeurtenissensysteem zo uit, dat je op meerdere gebeurtenissen kunt reageren.

OPGAVE D (GROTE UITDAGING): BOUNDING BOX

Het is handig als de breedte en de hoogte van een WeergaveObjectCotainer een andere waarde dan 0 zou hebben. De breedte en de hoogte zou bepaald kunnen worden door de breedte en een hoogte van een rechthoek dat precies alle kinderen omsluit. Dit rechthoek wordt de bounding box genoemd van de WeergaveObjectContainer. Schrijf code waarmee de bounding box van een WeergaveObjectContainer bepaald kan worden.