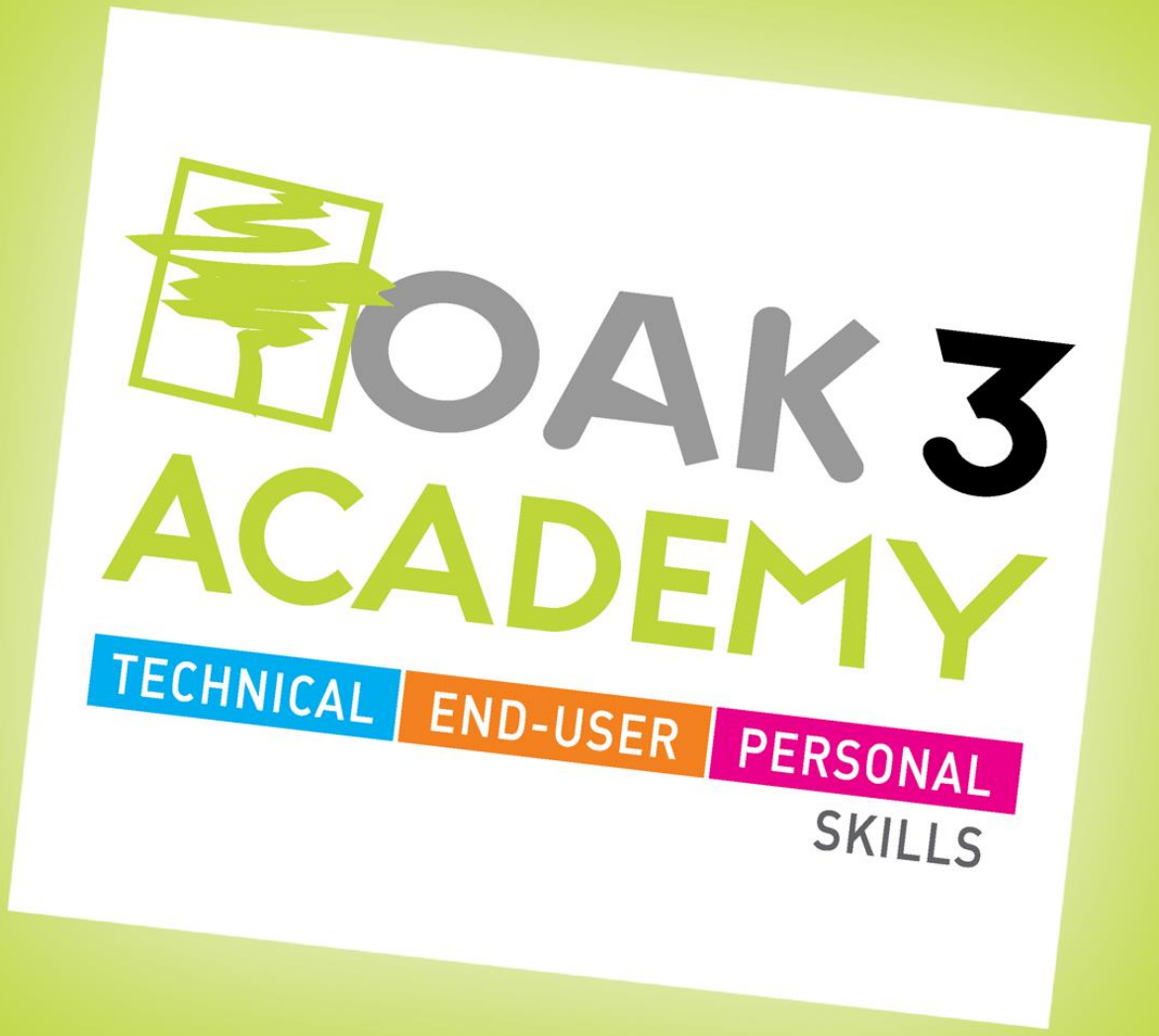


Interfaces

Inhoud

1. Inleiding
2. Een interface definiëren
3. Een interface implementeren in een klasse
4. Standaardmethoden
5. Statische methoden
6. Interface als datatype
7. Samenvatting

1. Inleiding

Definitie

- Interface is een soort abstracte klasse die alleen abstracte methoden en constanten (final eigenschappen) bevat.
- Sinds Java 8 kan een interface ook default methoden en static methoden bevatten.

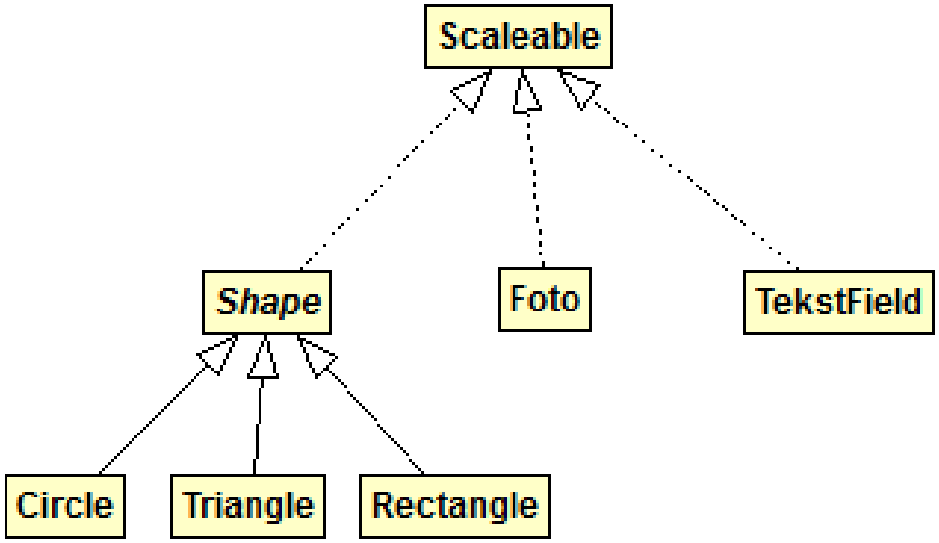
Doel

Aan bepaalde klassen methoden opleggen die moeten geïmplementeerd worden

Voorbeeld: grafisch programma

=> hoe er zeker van zijn dat alle grafische objecten herschaald kunnen worden

=> gebruik maken van interface Scaleable



=> door interface mogelijk om verschillende objecten toch een gemeenschappelijk gedrag te geven

2. Een interface definiëren

2.1 Declaratie van de interface

```
public abstract interface InterfaceName extends SuperInt1,SuperInt2
{
      …
}
```

Vet gedrukt = verplicht

Interface kan afgeleid zijn van meerdere superinterfaces

Voorbeeld:

```
public interface Scaleable {
      …
}
```

Toegangsniveau enkel public of package

2.1 De beschrijving van de interface

= declaratie van alle methoden en constanten

```
public interface Scaleable{
    // constanten
    public static final int DOUBLE = 200;
    public static final int HALF = 50;
    public static final int QUARTER = 25;

    // methoden
    public abstract void scale(int factor) ;

}
```

2.1 De beschrijving van de interface

= declaratie van alle methoden en constanten

```
public interface Scaleable{
    // constanten
    public static final int DOUBLE = 200;
    public static final int HALF = 50;
    public static final int QUARTER = 25;


    // methoden
    public abstract void scale(int factor) ;

}
```

alleen finale eigenschappen

2.1 De beschrijving van de interface

= declaratie van alle methoden en constanten

```
public interface Scaleable{

    // constanten
    public static final int DOUBLE = 200;
    public static final int HALF = 50;
    public static final int QUARTER = 25;

    // methoden
    public abstract void scale(int factor) ;

}
```

Abstracte methode:

- wel definitie
- geen implementatie

OAK 3 ACADEMY
TECHNICAL END-USER PERSONAL SKILLS

2.1 De beschrijving van de interface

= declaratie van alle methoden en constanten

```
public interface Scaleable{
    // constanten
    public static final int DOUBLE = 200;
    public static final int HALF = 50;
    public static final int QUARTER = 25;

    // methoden
    public abstract void scale(int factor) ;

}
```

Alle methoden zijn impliciet `public` en `abstract`

Impliciete gegevens mogen weggelaten worden

Alle variabelen zijn impliciet `public`, `static` en `final`

2.1 De beschrijving van de interface

= declaratie van alle methoden en constanten

```
public interface Scaleable{
    // constanten
    int DOUBLE = 200;
    int HALF = 50;
    int QUARTER = 25;

    // methoden
    void scale(int factor) ;

}
```

3. Een interface implementeren in een klasse

Bij declaratie van de klasse wordt aangegeven welke interfaces geïmplementeerd worden.

```
public class ClassName implements InterfaceName {
…
}
```

Voorbeeld

```
public class Rectangle extends Shape implements Scaleable
{
…
}
```

→ Alle methoden gedefinieerd in de interface Scaleable **moeten** geïmplementeerd worden in de klasse Rectangle.

→ De klasse Rectangle kan gebruik maken van alle constanten gedefinieerd in de interface Scaleable.

```
public class Rectangle extends Shape
implements Scaleable {

private int height, width;

public Rectangle (int x, int y,
                  int h, int w) {
    super(x, y);
    this.height = height;
    this.width = width;
}

//andere methoden

public void scale(int factor) {
    height = height * factor / 100;
    width = width * factor / 100;
}
```

Rectangle is een subklasse van de abstracte klasse Shape

implements

Rectangle maakt gebruik van de interface Scaleable

```
public class Rectangle extends Shape
implements Scaleable {

private int height, width;

public Rectangle (int x, int y,
                  int h, int w) {
    super(x, y);
    this.height = height;
    this.width = width;
}

//andere methoden

public void scale(int factor) {
    height = height * factor / 100;
    width = width * factor / 100;
}
```

Eigenschappen

x, y overgeërfd van Shape

height, width

```
public class Rectangle extends Shape
implements Scaleable {

private int height, width;

public Rectangle (int x, int y,
                  int h, int w) {
    super(x, y);
    this.height = height;
    this.width = width;
}

//andere methoden

public void scale(int factor) {
    height = height * factor / 100;
    width = width * factor / 100;
}
```

constructor

```
public class Rectangle extends Shape
implements Scaleable {

private int height, width;

public Rectangle (int x, int y,
                  int h, int w) {
    super(x, y);
    this.height = height;
    this.width = width;
}

//andere methoden

public void scale(int factor) {
    height = height * factor / 100;
    width = width * factor / 100;
}
```

Andere methoden

waaronder de implementatie van de abstracte methoden gedefinieerd in de abstracte klasse Shape

```
public class Rectangle extends Shape
implements Scaleable {

private int height, width;

public Rectangle (int x, int y,
                  int h, int w) {
    super(x, y);
    this.height = height;
    this.width = width;
}

//andere methoden

public void scale(int factor) {
    height = height * factor / 100;
    width = width * factor / 100;
}
```

Abstracte methodes uit interface Scaleable

moeten geïmplementeerd worden

Implementatie van meerdere interfaces

```
public class ClassName implements InterfaceName1,
InterfaceName2 {
…
}
```

Voorbeeld

```
public class Rectangle extends Shape implements
Scaleable, Drawable {
…
}
```

→ Alle methoden gedefinieerd in de interface Scaleable en Drawable moeten geïmplementeerd worden in de klasse Rectangle.

→ De klasse Rectangle kan gebruik maken van alle constanten gedefinieerd in de interface Scaleable en Drawable.

Implementatie van meerdere interfaces
= meervoudige overerving (=multiple inheritance)

Bij klassen enkelvoudige overerving!!!!!
= elke klasse heeft exact één directe superklasse

Voorbeeld

```
public class Dier{                    // superklasse is Object
}

public class Hond extends Dier{   // superklasse is Dier
}

class VreemdDing extends Dier, Auto{     // FOUT
}
```

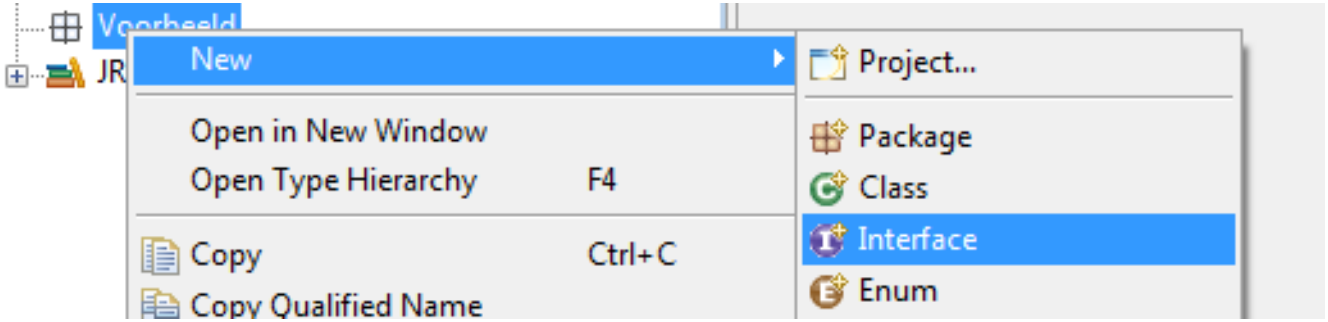
```
public interface Scaleable{
    // methoden
    void scale(int factor);

    // constanten
    int DOUBLE = 200;
    int HALF = 50;
    int QUARTER = 25;
}
```

```
public interface Drawable{
    // methoden
    void draw() ;

}
```

```
public class Rectangle extends Shape
implements Scaleable, Drawable {

private int height, width;

public Rectangle (int x, int y,
                  int h, int w) {
    super(x, y);
    this.height = height;
    this.width = width;
}

//andere methoden

public void scale(int factor){
    height = height * factor / 100;
    width = width * factor / 100;
}

public void draw(){
    System.out.printf("x:%d y:%d h:%d w:%d",
    super.getX(), super.getY(), height, width);
}
```

Interfaces in Eclipse

RMK op package, new, interface

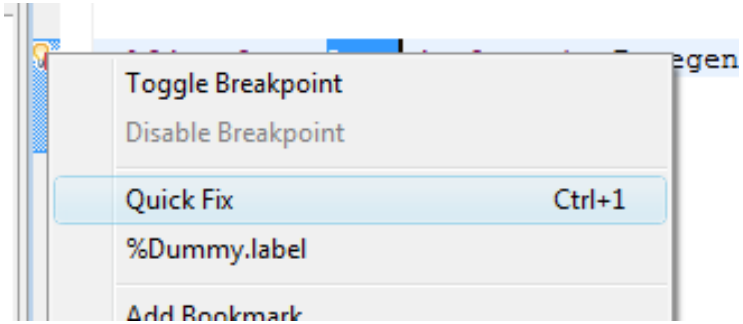


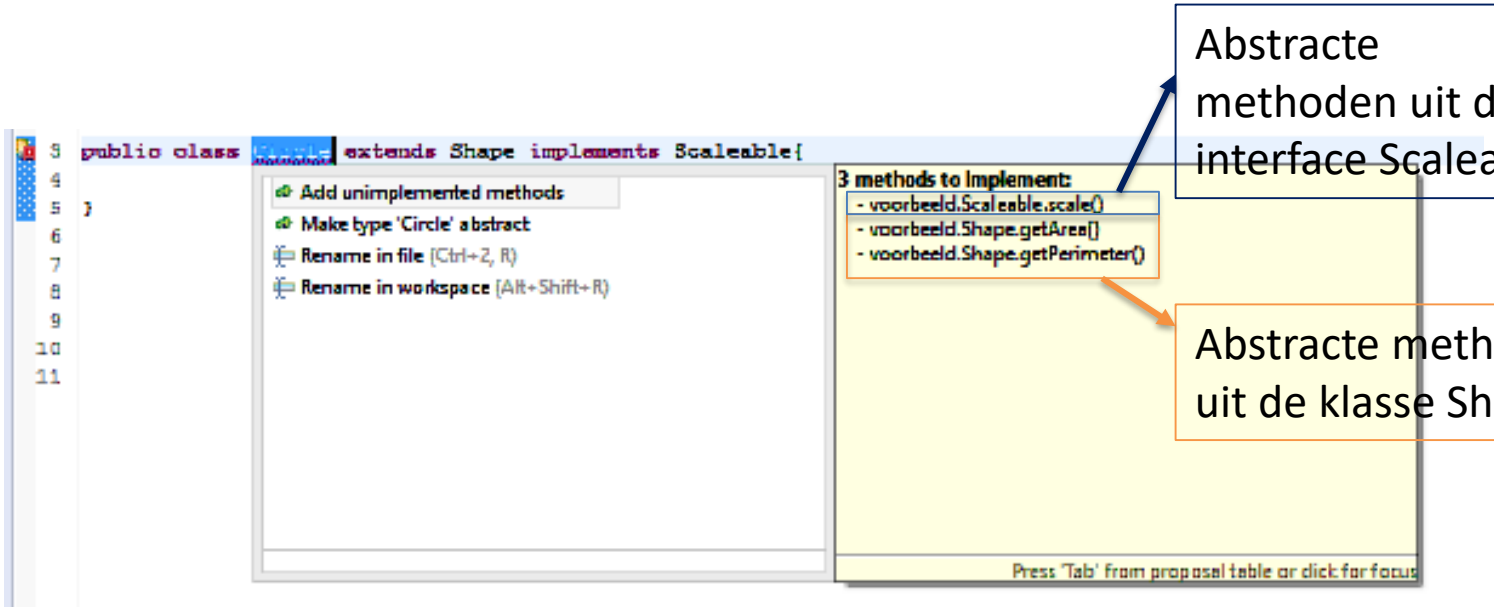
public class Circle extends Shape implements Scaleable

```
1  package voorbeeld;
2
3  public class Circle extends Shape implements Scaleable{
4
5  }
6
```

Fout: Circle heeft nog niet alle methodes uit Scaleable geïmplementeerd

RMK op  , Quick fix, Add unimplemented methods





4. Standaardmethoden

Probleem

Hoe methoden achteraf aan interface toevoegen zonder dat de implementerende klassen gedwongen worden om hiervan een implementatie te voorzien?

Oplossing

Sinds Java 8 gebruik van **standaard methoden (default methoden)**

= methoden die al een implementatie hebben

```
public interface Scaleable{

    // constanten
    public static final int DOUBLE = 200;
    public static final int HALF = 50;
    public static final int QUARTER = 25;

    // methoden
    public abstract void scale(int factor) ;

    public default void scaleDouble() {
        scale(DOUBLE);
    }

    public default void scaleHalf() {
        scale(HALF);
    }

}
```

Implementerende klassen/ afgeleide interfaces

- moeten met methoden scaleDouble() , scaleHalf() niets doen
  => implementatie wordt gewoon overgenomen

- kunnen methoden scaleDouble(), scaleHalf() herdefiniëren als abstracte methode
  => concrete subklassen moeten deze methode zelf implementeren

- kunnen methode scaleDouble(), scaleHalf() opnieuw implementeren (override)

5. Statische methoden

Sinds Java 8 ook statische methoden mogelijk in interfaces => hebben een implementatie

```
public interface MyInterface {
     // methoden
     public void method();

     public static void staticMethod() {
          System.out.println(“call of staticMethod()“);
     }
}
```

```
public class MyClass implements MyInterface {
     public void method() {
          System.out.println(“call of method()“);
     }
}
```

```
public class MyClassApp {
     public static void main (String [] args){
          Myclass test =  new MyClass();
          test.method();
          MyInterface.staticMethod();
     }
}
```

6. De interface als data-type

Tot nu toe: polymorfisme

= **variabele van een superklasse** kan verwijzen naar een **object van een klasse afgeleid van deze superklasse.**

```
Rectangle rect = new Rectangle();
Shape shape = new Rectangle();
```

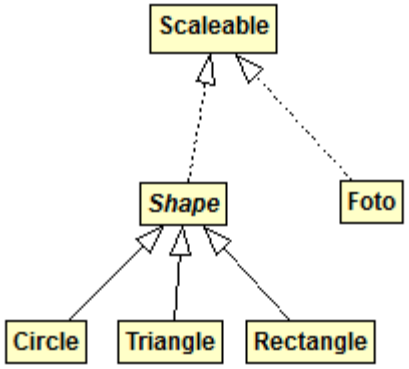
Nieuwe vorm van polymorfisme:

= **variabele van een interface** kan verwijzen naar een **object van een klasse die deze interface implementeert.**

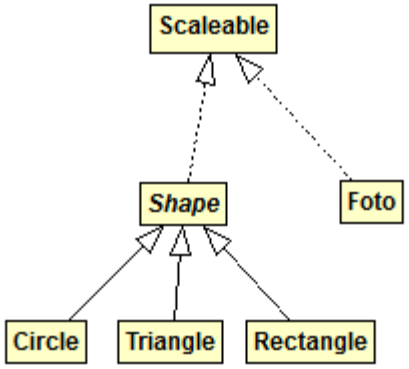
```
Scaleable scaleable = new Rectangle();
```

OAK 3 ACADEMY
TECHNICAL END-USER PERSONAL SKILLS

```
public class TekenProgramma {

    public static void main(String[] args) {
        Scaleable [] tek = new Scaleable[4];
        tek[0] = new Foto() ;
        tek[1] = new Circle(3, 2, 1) ;
        tek[2] = new Triangle() ;
        tek[3] = new Rectangle(5, 10, 2, 3) ;

        for(int i = 0; i < tekening.length; i++){
            tek[i].scale(50);
        }

        tek[3].setHeight(20);   // foutmelding, oplossing?

        if (tek[1] instanceof Scaleable) {
            System.out.println("dit object implementeert
                        de interface Scaleable");
        }
     }
}
```



```
public class TekenProgramma {

    public static void main(String[] args) {
        Scaleable [] tek = new Scaleable[4];
        tek[0] = new Foto() ;
        tek[1] = new Circle(3, 2, 1) ;
        tek[2] = new Triangle() ;
        tek[3] = new Rectangle(5, 10, 2, 3) ;

        for(int i = 0; i < tekening.length; i++){
            tek[i]. scale(50);
        }

        ((Rectangle) tek[3]).setHeight(20);  //casting

        if (tek[1] instanceof Scaleable) {
            System.out.println("dit object implementeert
                        de interface Scaleable");
        }
      }
}
```

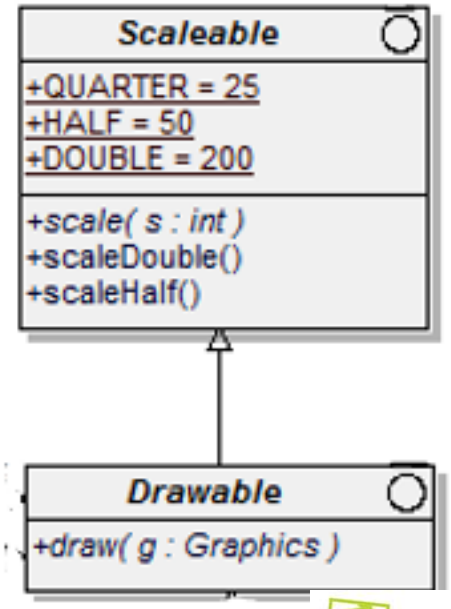


Opdracht

Interfaces

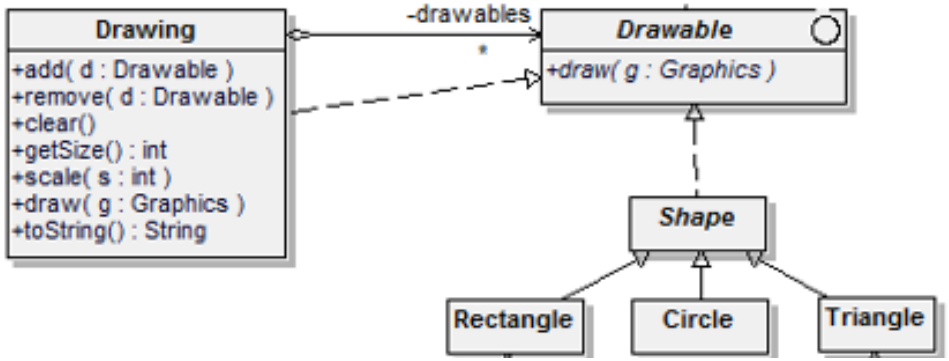
- Maak de Interface Scaleable aan.

- Maak de Interface Drawable aan,
  deze interface is afgeleid van de interface Scaleable.



- Implementeer de interface Drawable voor alle figuren.
  Opmerking : De oppervlakte van elke figuur moet met de opgegeven factor vergroten/verkleinen.
- Maak een programma DrawApp dat een array van Shapes ‘shape’ aanmaakt van grootte 3.
  Steek in deze array een Circle, een Rectangle en een Triangle.
  Wijzig de schaal van de figuren en druk de dimensies af.

- Wijzig de klasse Drawing als volgt



Opmerkingen

1. methode draw()

   roept methode draw() op voor alle elementen in de array drawables

2. methode scale()

   roept methode scale() op voor alle elementen in de array drawables

3. methode toString()

   roept de methode toString() op voor alle elementen in de array drawables en maakt hiervan een nieuwe String

- Maak in je programma DrawApp een tekening.
  - ✓ Plaats de elementen van de array ‘shape’ in je tekening.
  - ✓ Voeg vervolgens 50 cirkels toe aan je tekening (straal van 1 tem 50, middelpunt x = 0, y = 0.
  - ✓ Druk de gegevens van alle elementen in je tekening af.
  - ✓ Herschaal alle elementen in je tekening.
  - ✓ Druk daarna opnieuw alle elementen in je tekening af.

4. Samenvatting

- Interface is een verzameling van abstracte methoden die in een klasse die deze interface gebruikt geïmplementeerd moeten worden.

- Interface kan daarnaast ook constanten, default methoden en static methoden bevatten.

- Mbv interface nieuw soort datatype
  => objecten die heel verschillend zijn, kunnen toch op dezelfde wijze behandeld worden