# polymorfisme

### Voorbeeld

### Te gebruiken bij:

-

### Eigenschappen:

-

### Info

#### Overzicht

Afbeelding met tekst, schermopname, Lettertype, diagram

Automatisch gegenereerde beschrijving

#### Aantekening

Laten we een voorbeeld bekijken om het beter te begrijpen. Stel dat we een classhiërarchie hebben met een basis-klasse genaamd Shape en afgeleide klassen zoals Circle en Rectangle. Elk van deze klassen heeft een methode genaamd calculateArea() om de oppervlakte van de vorm te berekenen.

java

public abstract class Shape {

public abstract double calculateArea();

}

public class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

public double calculateArea() {

return Math.PI \* radius \* radius;

}

}

public class Rectangle extends Shape {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

public double calculateArea() {

return length \* width;

}

}

Nu kunnen we polymorfisme toepassen door een variabele van het type Shape te declareren en deze te gebruiken om objecten van de klassen Circle en Rectangle te verwijzen:

java

Shape shape1 = new Circle(5.0);

Shape shape2 = new Rectangle(4.0, 6.0);

double area1 = shape1.calculateArea(); // Roep calculateArea() aan voor een cirkel

double area2 = shape2.calculateArea(); // Roep calculateArea() aan voor een rechthoek

In dit voorbeeld kunnen we zien dat we dezelfde calculateArea()-methode kunnen aanroepen op verschillende objecten van de klassen Circle en Rectangle, ondanks dat de variabelen shape1 en shape2 van het type Shape zijn. Dit is mogelijk omdat de afgeleide klassen Circle en Rectangle de methode calculateArea() hebben geïmplementeerd vanuit de basis-klasse Shape.