



# Java Essentials

## Hoofdstuk 8

### Klassen definiëren

#### **DE HOGESCHOOL MET HET NETWERK**

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt  
[www.pxl.be](http://www.pxl.be) - [www.pxl.be/facebook](https://www.pxl.be/facebook)



# Inhoud

1. Inleiding
2. De declaratie van de klasse
3. De klasse-omschrijving (body)
  1. Eigenschappen
  2. Methoden
  3. Constructors
  4. Instance members en class members
  5. De klasse Math



# 1. Inleiding

Zelf klassen maken: vb Rectangle

Eigenschappen: height  
width  
x  
y

Methoden: 

setHeight()  
setWidth()  
setPosition()  
getHeight()  
getWidth()  
getArea()  
getPerimeter()



## 2. De declaratie van de klasse

```
package ...;
```

```
public class Rectangle {  
    ...  
}
```

Dit is de 'meest eenvoudige' klasse-declaratie. Hier kunnen ook nog andere (optionele) componenten staan

Geen `main()`-methode. Een programma bestaat meestal uit meerdere klassen waarvan er maar 1 de methode `main()` heeft.



## Opdracht 1: *Een klasse maken*

Maak een klasse met de naam **Rectangle.java** in het pakket `be.pxl.h8.opdracht`



```
package be.pxl.h8.opdracht;  
public class Rectangle {  
    ...  
}
```

*(de klasse Rectangle gaan we steeds verder uitbreiden met eigenschappen en methoden)*

## Opdracht 2: *Een hoofdprogramma maken*

- Maak een hoofdprogramma met de naam **RectangleApp**
- Maak een object van de klasse *Rectangle* (m.b.v. *new*).

```
public class RectangleApp {  
    public static void main(String[] args) {  
        System.out.println("This program uses a rectangle");  
        Rectangle rect = new Rectangle();  
    }  
}
```

*(RectangleApp gaan we ook steeds verder uitbreiden:  
met code die gebruik maakt van de klasse Rectangle)*



# 3. De klasse-omschrijving (body)

- Body: tussen accolades

- Bevat:

- Eigenschappen (member-variabelen)
- Methoden (member-methods)
- Constructors

= Members  
van de klasse

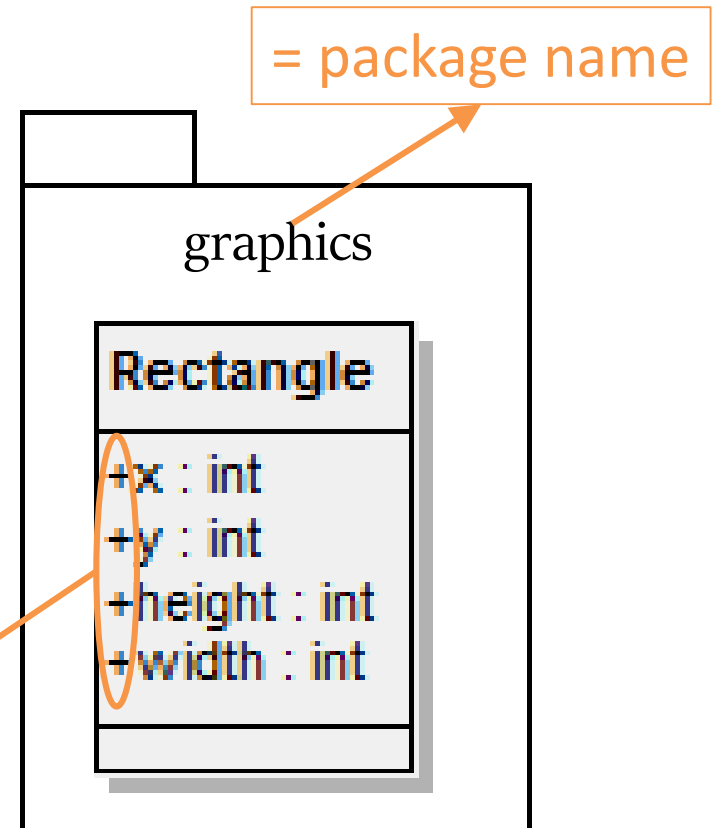


## 3.1 Eigenschappen

```
public class Rectangle {  
    public int x;  
    public int y;  
    public int height;  
    public int width;  
}
```

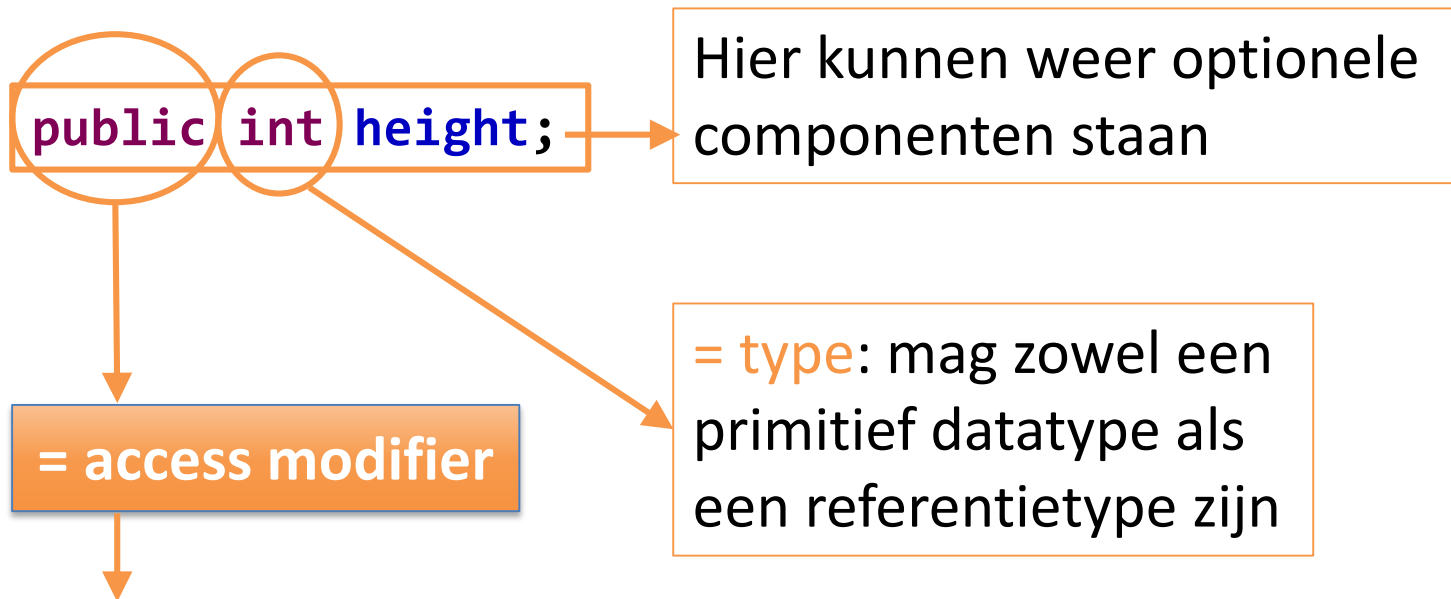
= Member-variabelen  
(zijn in dit voorbeeld  
instantievariabelen)

= public



= UML-schema van  
de klasse Rectangle





`public`: d.w.z. vanuit elke klasse toegankelijk  
Kan ook `private` zijn = beter  (zie verder in dit hoofdstuk).  
Verder kan je hier ook nog andere toegangs-niveaus schrijven

Indien de variabelen niet expliciet geïnitialiseerd worden, krijgen ze automatisch de waarde `0`, `null` of `false` naargelang het datatype.

Eigenschappen (variabelen) gebruiken:

*objectName.propertyName*

```
public class RectangleApp {  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle();  
        System.out.println(rect.height);  
    }  
}
```



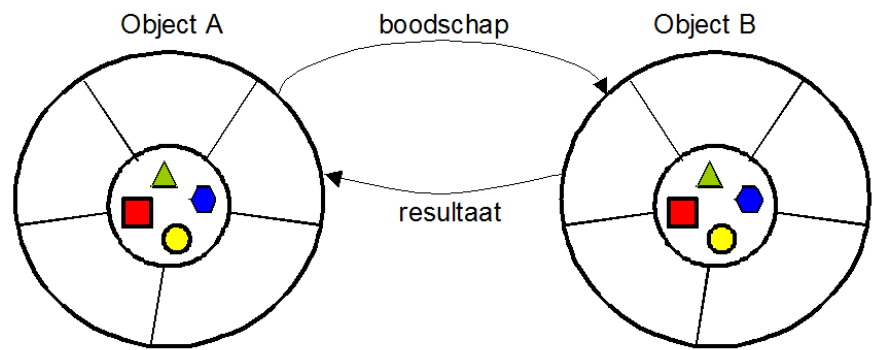
## Opdracht 3: *Member-variabelen toevoegen*

- Voeg de variabelen *height*, *width*, *x* en *y* toe aan de klasse-definitie.
- Druk de hoogte, breedte en de positie van de rechthoek af op het scherm.
- Ken in het hoofdprogramma expliciet een waarde toe aan de variabelen.
- Druk de hoogte, breedte en de positie van de rechthoek af op het scherm.
- Maak een 2<sup>e</sup> rechthoek en geef deze andere afmetingen en een andere positie. Druk de gegevens van deze 2<sup>e</sup> rechthoek af.
- Verander het toegangsniveau van de variabelen eens in `private`. Werkt dit?



## 3.2 Methoden

- Naam voor stuk code (werkwoord)
- D.m.v. methoden kunnen andere objecten boodschappen sturen naar dit object en eventueel een resultaat terugkrijgen.
- Heeft 1 taak
- Herbruikbaar
- Aanpasbaar
- lowerCamelCasing
- Kan andere methodes gebruiken



*Voorbeelden:*

## Rectangle:

- *rect1* (object) is een *Rectangle* (class)
- *height* en *width* zijn data (eigenschappen) van *rect1/Rectangle*
- *getArea()* is een methode (taak) om de oppervlakte van *rect1/Rectangle* te berekenen

## Persoon

- *jan* (object) is een *Persoon* (class)
- *geboortedatum* is data van *jan/Persoon*
- *berekenLeeftijd()* is een methode (taak) van *jan/Persoon*



## 3.2.1 Declaratie van methoden

```
[public | private] returntype methodName(parameters) {  
    ...  
}
```

- Vet gedrukt = verplicht
- Hier zijn weer meer componenten mogelijk

*Voorbeeld:*

```
public void doMethod(...) {  
    ...  
}
```

Er wordt geen waarde geretourneerd

De methode is toegankelijk vanuit andere klassen

↔ *private*: enkel toegankelijk vanuit de klasse zelf



## 3.2.2 De body van de methode

= member method

```
public class MyClass {  
  
    ...  
    public void doMethod() {  
        int number = 6;  
        System.out.println(number);  
    }  
}
```

Binnen codeblok  
kunnen variabelen  
gedeclareerd worden



lokale variabelen

moeten geïnitieerd worden



Membervariabelen (moeten niet geïnitieerd worden)



Een lokale variabele mag dezelfde naam hebben als een member-variabele → member-variabele wordt als het ware verborgen door de lokale variabele

= shadowing

```
public class MyClass {  
    private int number = 5;  
  
    public void doMethod() {  
        int number = 6;  
        System.out.println(number);      → 6  
        System.out.println(this.number); → 5  
    }  
}
```

Om de member-variabele toch te benaderen  
→ gebruik maken van het *this*-object  
(= referentie naar het huidig object)



## 3.2.3 Gegevens doorgeven aan een methode

```
public class Rectangle {
```

```
    public int x;
```

```
    public int y;
```

```
    ...
```

```
    public void setPosition(int xpos, int ypos) {
```

```
        x = xpos;
```

```
        y = ypos;
```

```
    }
```

```
    ...
```

```
}
```

= parameters

waarde 7 wordt gekopieerd naar xpos

waarde 9 wordt gekopieerd naar ypos

Bereik  
van  
*xpos* en  
*ypos*

```
public class RectangleApp {
```

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

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

```
        rect.setPosition(7, 9);
```

```
    }
```

```
}
```

= argumenten



```
public void setPosition(int xpos, int ypos) {  
    x = xpos;  
    y = ypos;  
}
```

= call-by-value

```
rect.setPosition(7, 9);
```

Voor primitieve datatypes  
wordt de waarde doorgegeven.



```
public class Rectangle {
```

```
    public int x;
```

```
    public int y;
```

```
    ...
```

```
    public void setPosition(int xpos, int ypos) {
```

```
        x = xpos;
```

```
        y = ypos;
```

```
        xpos = 0;
```

```
        ypos = 0;
```

```
    }
```

```
    ...
```

```
}
```

```
public class RectangleApp {
```

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

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

```
        int a = 5;
```

```
        int b = 7;
```

```
        rect.setPosition(a, b);
```

```
        System.out.println(a); → 5
```

```
        System.out.println(b); → 7
```

➔ Het wijzigen van de waarde van *xpos* in de methode heeft geen invloed op de waarde van *a*. De waarde van de variabele *a* wordt immers doorgegeven (gekopieerd) naar de variabele *xpos*.



```

public class MyClass {
    public void doMethod(StringBuilder strB) {
        strB.append("World");
    }
}

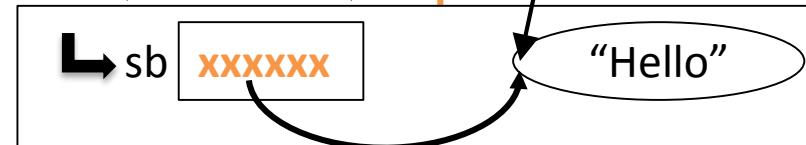
```

Voor referentietypes wordt de referentie  
(adres **xxxxxx**) naar het object doorgegeven.

```

StringBuilder sb = new StringBuilder("Hello");
MyClass object = new MyClass();
object.doMethod(sb);
System.out.println(sb); → HelloWorld

```



= call-by-reference



## Opdracht 4: *Methoden met parameters toevoegen*

- Voeg aan de klasse `Rectangle` de nodige methoden toe om de hoogte en de breedte van de rechthoek in te stellen.
- Voeg methoden toe om de positie van de rechthoek in te stellen:  
    `setPosition()`  
    `setX()`  
    `setY()`
- Voeg een methode toe om de grootte van de rechthoek met een bepaalde waarde te laten toenemen:  
    `grow(int dw, int dh)`



## 3.2.4 Waarden teruggeven via een methode (return)

```
return expression;
```

```
public class Rectangle {  
    ...  
    public double getArea() {  
        return width * height;  
    }  
}
```

Diagram: An arrow points from the word `double` to a box labeled "returntype".

- *return*: meestal op het einde van een methode.
- Indien er geen waarde wordt teruggegeven, kan het *return*-statement weggelaten worden.
- Indien het *return*-statement in het midden van een methode staat, wordt de methode op de plaats van het *return*-statement beëindigd.

```
public static void main(String[] args) {  
    ...  
    double area = rect.getArea();  
    System.out.println("Area: " + area);  
}
```



## Opdracht 5:

## *Methoden toevoegen die waarden teruggeven*

- Voeg aan de klasse Rectangle de nodige methoden toe om de oppervlakte en de omtrek te berekenen:

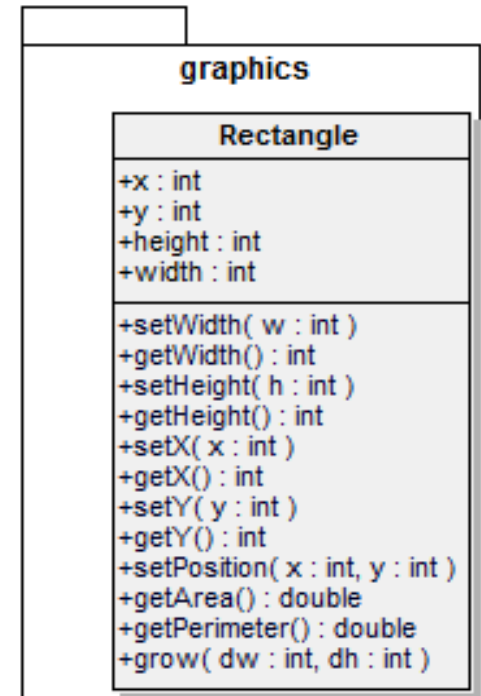
```
getArea()  
getPerimeter()
```

- Druk in het hoofdprogramma de oppervlakte en omtrek van de rechthoeken op het scherm af.



- Voeg methoden toe om de hoogte, breedte en de positie van de rechthoek op te vragen:

```
getWidth()  
getHeight()  
getX()  
getY()
```



Gebruik deze methoden om de hoogte, positie in het hoofdprogramma af te drukken.



# Methode met een variabel aantal parameters

Vind je terug in de cursus  
Hoofdstuk 7, punt 7.8

```
returntype method(type... params);
```

wordt door de compiler omgezet in:

```
returntype method(type[] params);
```

array



```
public class Statistics {  
    public static void main(String[] args) {  
        Statistics stat = new Statistics();  
        stat.printAverage(4, 7, 9);  
        stat.printAverage(3, 8, 6, 9, 4, 7);  
    }  
}
```

Er wordt een  
variabel aantal  
argumenten  
doorgegeven

(Deze methode-call roept hier vanuit de main-methode  
een methode aan uit **dezelfde** klasse (*Statistics*))

```
public void printAverage(int... values) {  
    int total = 0;  
    for (int el : values) {  
        total += el;  
    }  
    System.out.println((double)total /  
                        values.length);  
}
```

Kan enkel gebruikt worden als  
**laatste** parameter in de  
parameterlijst.



```
public class Statistics {  
    public static void main(String[] args) {  
        Statistics stat = new Statistics();  
        stat.printAverage(3, 8, 6, 9, 4, 7);  
        int[] values = { 3, 8, 6, 9, 4, 7 };  
        stat.printAverage(values);  
    }
```

doet  
hetzelfde  
als

```
    public void printAverage(int... values) {  
        int total = 0;  
        for (int el : values) {  
            total += el;  
        }  
        System.out.println((double)total /  
                             values.length);  
    }
```



```
public static void main(String[] args) {  
    stat.printAverage(3, 8, 6, 9, 4, 7);  
    int[] values = { 3, 8, 6, 9, 4, 7 };  
    stat.printAverage(values);  
}  
public void printAverage(int... values) {  
    ...  
}
```

```
public static void main(String[] args) {  
    stat.printAverage(3, 8, 6, 9, 4, 7);  
    int[] values = { 3, 8, 6, 9, 4, 7 };  
    stat.printAverage(values);  
}  
public void printAverage(int[] values) {  
    ...  
}
```

Deze combinatie  
is niet mogelijk.



Parameter kan dezelfde naam hebben als member-variabele  
→ *this* gebruiken

```
public class Rectangle {  
    public int x;  
    public int y;  
    ...  
    public void setPosition(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    ...  
}
```

## *Private of public* membervariabelen?

```
public int height;  
public int width;
```

```
public void setHeight(int height){  
    this.height = height;  
}
```

```
public class RectangleApp {  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle();  
        rect.height = 5;  
    }
```

➡ De hoogte van de rechthoek kan eender welke integer waarde krijgen. Wat als je de mogelijke waarden wil beperken?



```
private int height;  
private int width;
```

```
public int getHeight() {  
    return height;  
}
```

= Getter

```
public void setHeight(int height){  
    if (height < 0){  
        this.height = -height;  
    } else {  
        this.height = height;  
    }  
}
```

= Setter

We kunnen bijvoorbeeld beslissen om negatieve getallen te vervangen door hun absolute waarde (of te vervangen door 0)



- Het is een slechte gewoonte om membervariabelen *public* te maken → *private* (= afschermen van de buitenwereld)
- Je maakt deze variabelen toegankelijk via aangepaste methoden
- Via de *public* methode kan je dan controle uitoefenen op de waarde van de *private* membervariabelen.

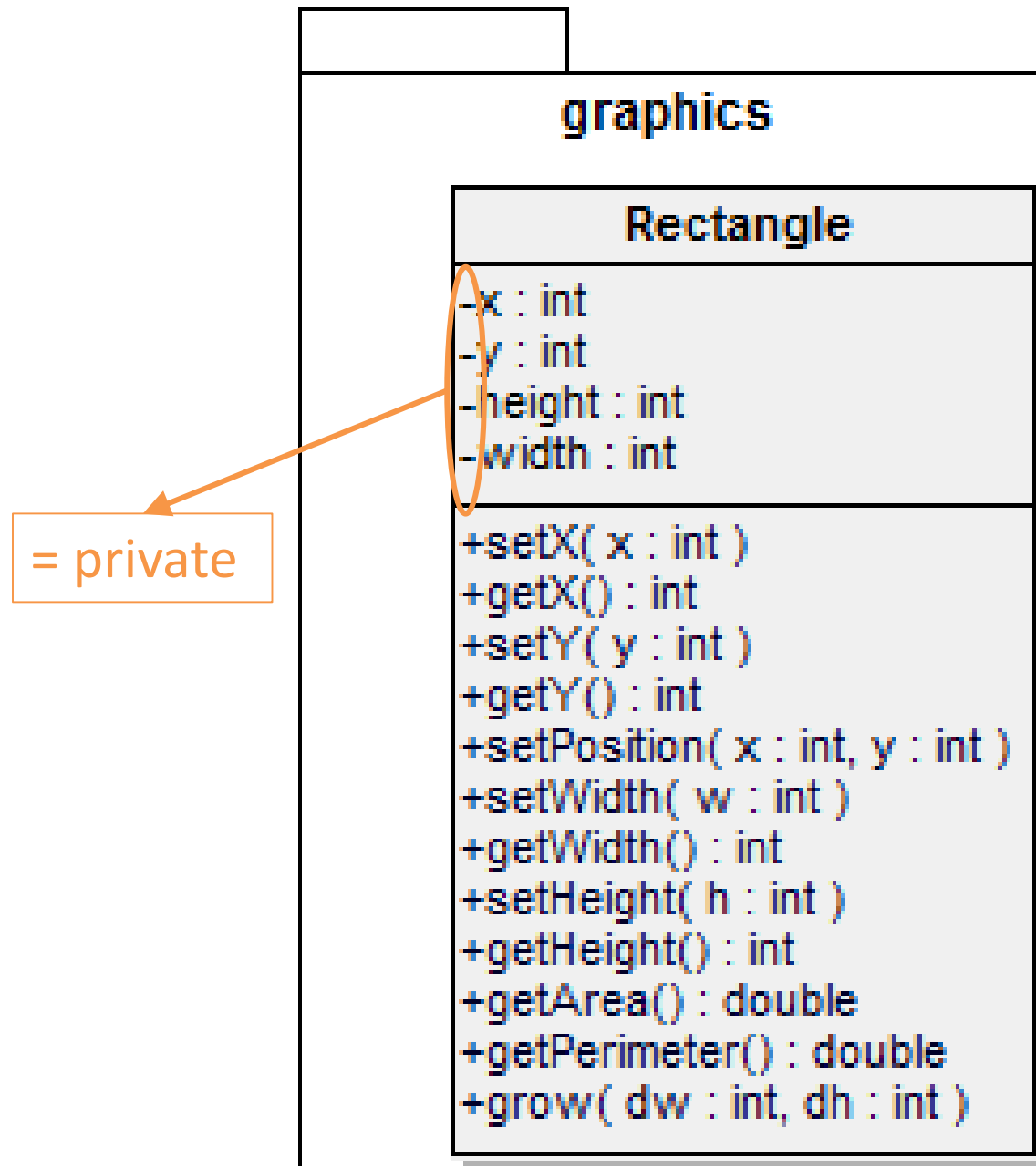
= encapsulation  
(inkapseling;  
data hiding)



## Opdracht 6: *Controle op variabelen via methoden*

- Pas de methoden voor het zetten van de hoogte en de breedte aan zodat de absolute waarde genomen wordt. Zorg ook dat de methode *grow()* de waarden niet negatief maakt.
- Maak de eigenschappen *hoogte*, *breedte*, *x* en *y* van de klasse *Rectangle* privaat en gebruik enkel de overeenkomstige methoden om de eigenschappen in te stellen en op te vragen.
- Maak in het hoofdprogramma een rechthoek met een negatieve hoogte en breedte en controleer het resultaat.





## 3.2.5 Method overloading

= methoden met dezelfde naam maar met verschillende (types/aantal) parameters



```
private int height;  
private int width;  
  
public void setHeight(int height){  
    if (height < 0){  
        this.height = -height;  
    } else {  
        this.height = height;  
    }  
}  
  
public void setHeight(short height){  
    if (height < 0){  
        this.height = -height;  
    } else {  
        this.height = height;  
    }  
}
```

De compiler kiest de juiste methode op basis van de parameters die worden doorgegeven.

De inhoud van de body mag voor beide methodes verschillend zijn.



```
public class RectangleApp {  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle();  
        int intHeight = 10;  
        short shortHeight = 20;  
        rect.setHeight(intHeight);  
        rect.setHeight(shortHeight);  
    }  
}
```

```
public void setHeight(int height) {...}
```

```
public void setHeight(short height) {...}
```

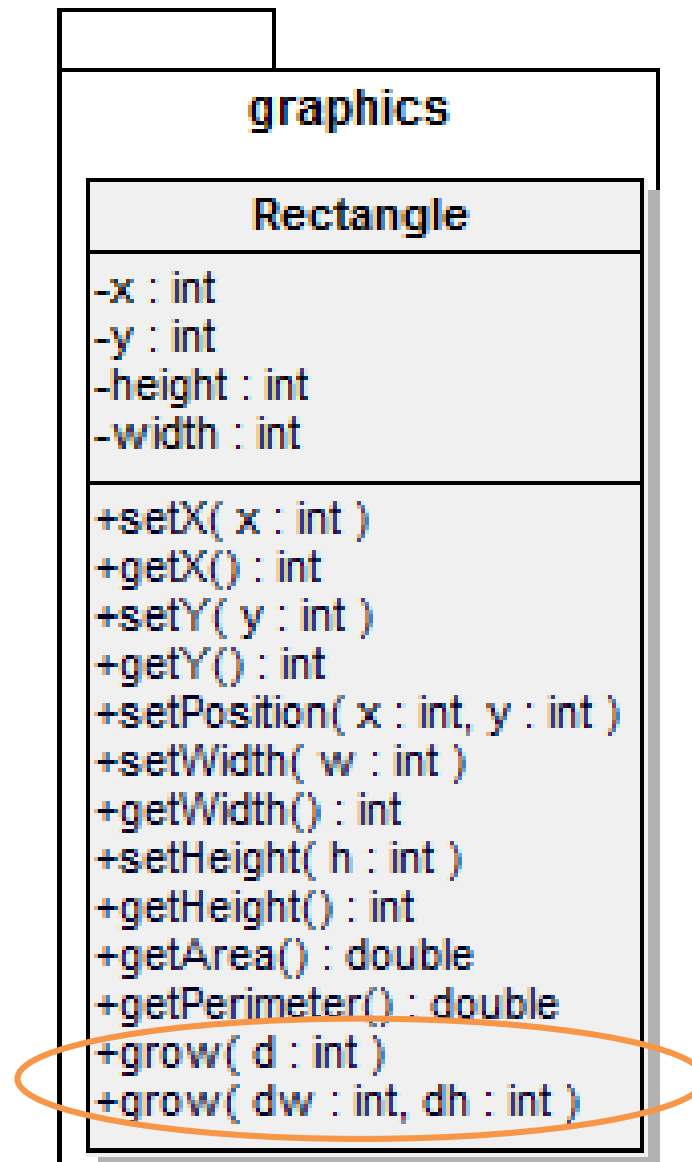
= Method-  
overloading



## Opdracht 7: *Method overloading*

- Voeg een tweede methode *grow()* toe die slechts één parameter heeft die zowel voor de hoogte als de breedte geldt.
- Test de methode uit in het hoofdprogramma.





## 3.3 Constructors

- Iedere klasse heeft een constructor(-method).
- Wordt uitgevoerd bij het creëren van het object met de new-operator.
- Wordt gebruikt om het object te initialiseren.
- Lijkt op een gewone methode maar heeft geen return-type en de naam komt overeen met de klassenaam.
- De methode kan maar 1x aangeroepen worden, nl. tijdens het aanmaken van het object.



```
public class Rectangle {  
    private int width;  
    private int height;  
    ...  
}
```

= Constructor-  
overloading

```
public Rectangle() {  
}
```

= default constructor  
(= parameterloze constructor)

```
public Rectangle(int width, int height) {  
    this.width = width;  
    this.height = height;  
}
```

= constructor met parameters

```
public class RectangleApp {  
    public static void main(String[] args) {  
        Rectangle rect1 = new Rectangle();  
        Rectangle rect2 = new Rectangle(4, 7);  
    }  
}
```



## Default constructor

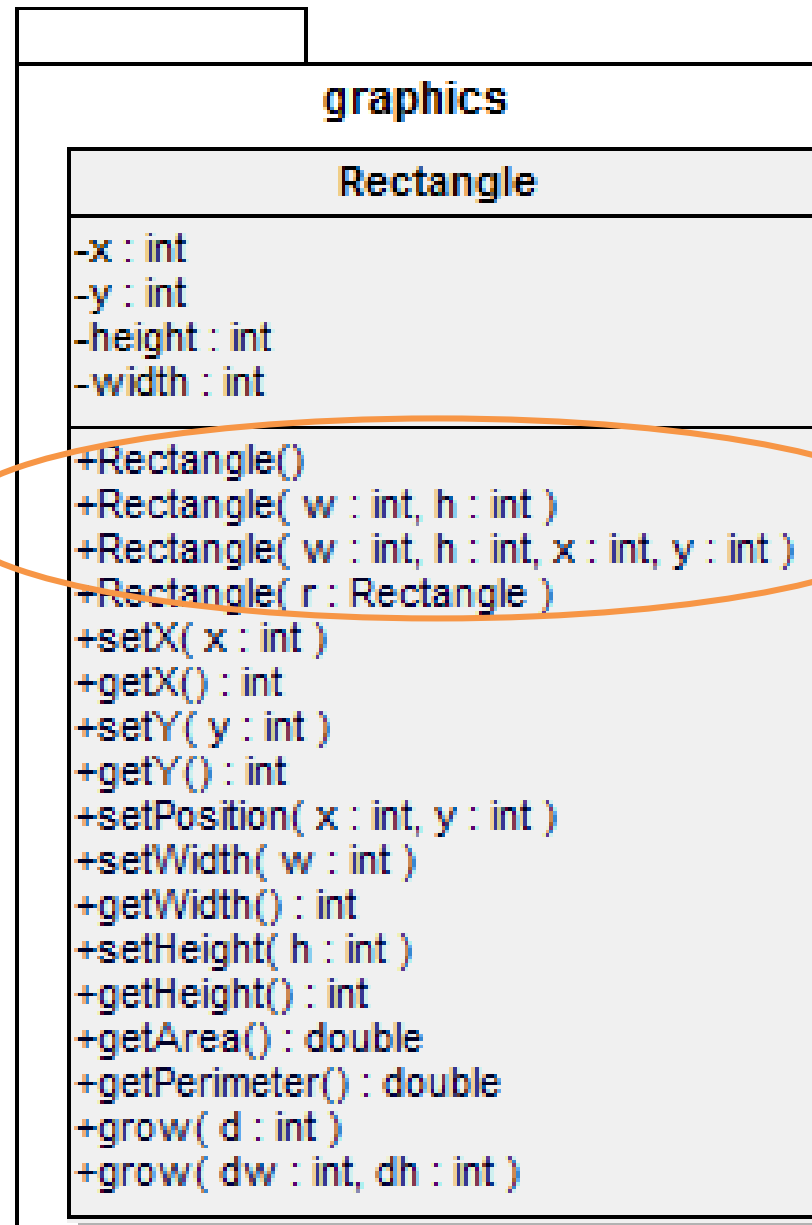
- Indien een constructor niet expliciet gedefinieerd wordt, krijgt de klasse een default constructor zonder parameters (*default constructor moet dus niet geschreven worden*)
- Zodra er een constructor met parameters gespecificeerd wordt, vervalt deze default constructor en moet dan eventueel expliciet gedeclareerd worden (*indien deze aangeroepen wordt vanuit het programma*)



## Opdracht 8: *Constructors toevoegen*

- Voeg een constructor toe met 2 parameters voor de initiële hoogte en breedte. Maak een rechthoek met deze constructor en druk de hoogte en breedte van de rechthoek af op het scherm.
- Voeg een constructor toe zonder parameters.
- Voeg een 3<sup>e</sup> constructor toe die naast de hoogte en de breedte ook nog de x-positie en de y-positie van de rechthoek als parameter neemt. Maak in het hoofdprogramma een rechthoek met deze constructor en druk de gegevens af op het scherm.
- Voeg een constructor toe die een rechthoek maakt o.b.v. een bestaande rechthoek. Geef als parameter een referentie naar de bestaande rechthoek mee.





Een constructor kan een andere constructor met de nodige parameters aanroepen: met de referentievariabele *this*

```
public class Rectangle {
```

```
...
```

```
public Rectangle() {
```

```
    this(0,0);
```

```
}
```

```
public Rectangle(int width, int height) {
```

```
    x = 0;
```

```
    y = 0;
```

```
    this.height = height;
```

```
    this.width = width;
```

```
}
```

```
}
```

*this* verwijst naar het eigen object (= referentievariabele)



## Voordeel:

- De eigenlijke code voor het initialiseren van het object moet maar op 1 plaats geschreven worden.
- Code wordt compacter en overzichtelijker
- Risico op fouten daalt: aanpassingen moeten immers gebeuren in 1 constructor i.p.v. in elke constructor.

Indien een constructor een andere constructor aanroept moet dit steeds gebeuren op de eerste regel



## Opdracht 9: *Het gebruik van this*

Laat de constructors met minder parameters de constructor met de meeste parameters aanroepen. Zorg er ook voor dat een rechthoek geen negatieve afmetingen krijgt via de constructor.



## 3.4 Instance members en class members

Instance members

= instance-variabelen en instance-methoden





## 3.4.1 Instance-variabelen

Instance-variabelen

= ieder object van bepaalde klasse heeft een eigen kopie van de variabelen met elk hun eigen waarde

Synoniemen: veld, eigenschap, objectvariabele, kenmerk, attribuut



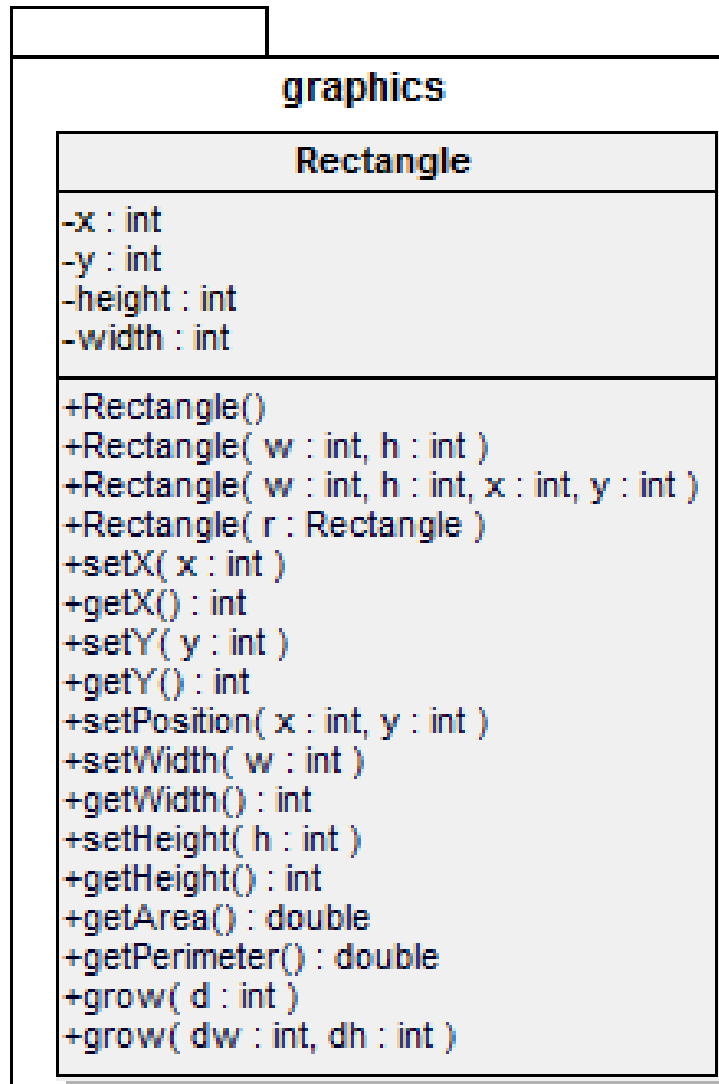
# Voorbeeld: klasse Rectangle

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
  
    ...  
}
```

- Elke rechthoek heeft zijn eigen variabelen met eigen waarden
- Eigenschappen: meestal private
- Wijzigen en opvragen via get- en set-methoden



# Voorbeeld: klasse Rectangle: UML-diagram



- **Instance-variabelen:**  
**private**
- **Constructors en**  
**methoden: public**

# Voorbeeld: klasse Rectangle

Gebruik van instance-variabelen in de class zelf

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
  
    ...  
    public void setX (int x) {  
        this.x = x;  
    }  
  
    public int getX () {  
        return x;  
    }  
}
```

- Lokale variabele en instance-variabele met zelfde naam?  
Gebruik 'this'



# Instance-variabelen: initialisatie

De initialisatie van instance-variabelen kan op 3 manieren:

1. Tijdens de declaratie (vooral voor primitieve datatypen)
2. In een initialisatieblok → zie volgende dia
3. In de constructor (waarden komen binnen als parameters)



# Voorbeeld: klasse Rectangle

## Initialisatieblok

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
  
    {  
        x = 0;  
        y = 0;  
        width = 5;  
        height = 10;  
    }  
  
    ...  
}
```

- De code van het initialisatieblok wordt uitgevoerd vóór de code van de constructor
- Initialisatieblok kan nuttig zijn als je bijvoorbeeld in je initialisatie een 'if' wil gebruiken



## 3.4.2 Klasse-variabelen

### Klasse-variabelen

- Zijn gemeenschappelijk voor alle objecten van dezelfde klasse.
- Van elke klasse-variabele is er slechts 1, die door alle objecten gedeeld wordt
- Worden gedefinieerd met `static`



# Voorbeeld: klasse Rectangle

Aantal hoeken toevoegen

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
    public static final int ANGLES = 4;  
  
    ...  
}
```

- Het aantal hoeken is voor elke rechthoek gelijk, Dus 1 variabele is voldoende.
- static → klasse-variabele
- final → constante
- public → mag publiekelijk geraadpleegd worden (kan omwille van 'final' toch niet aangepast worden)





# Gebruik van een klasse-variabele

Aantal hoeken

`ClassName.variabele`

→ Zo is het duidelijk dat de variabele bij een klasse hoort

Voorbeeld:

```
System.out.println(Rectangle.ANGLES);
```



# Voorbeeld: klasse Rectangle

Bijhouden hoeveel rechthoeken gecreëerd werden

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
    public static final int ANGLES = 4;  
    public static int count = 0;  
  
    ...  
}
```

- **count is voorlopig public**
- **Telkens een object gecreëerd wordt, moet het geteld worden**
  - in de constructor
  - OF via initialisatieblok

# Voorbeeld: klasse Rectangle

Bijhouden hoeveel rechthoeken gecreëerd werden  
→ teller ophogen in constructor

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
    public static final int ANGLES = 4;  
    public static int count = 0;  
  
    public Rectangle () {  
        this (0, 0);  
    }  
    public Rectangle (int width, int height) {  
        this.height = height;  
        this.width = width;  
        count ++;  
    }  
    ...  
}
```

- Doordat de constructors met minder parameters de constructor met de meeste parameters oproepen moet de code maar 1 keer toegevoegd worden!



# Voorbeeld: klasse Rectangle

Bijhouden hoeveel rechthoeken gecreëerd werden  
→ teller ophogen via initialisatieblok

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
    public static final int ANGLES = 4;  
    public static int count = 0;  
  
    {  
        count++;  
    }  
    ...  
}
```

- De verhoging gebeurt voordat de code van de constructor opgeroepen wordt.



# Initialisatie van een klasse-variabele

De initialisatie van klasse-variabelen kan op 2 manieren:

1. Tijdens de declaratie (vooral voor primitieve datatypen)  
vb. **public static final int** *ANGLES* = 4;
2. In een static initialisatieblok  
voor meer complexe initialisaties van meer dan 1  
statement  
→ zie volgende dia



# Voorbeeld: klasse Rectangle

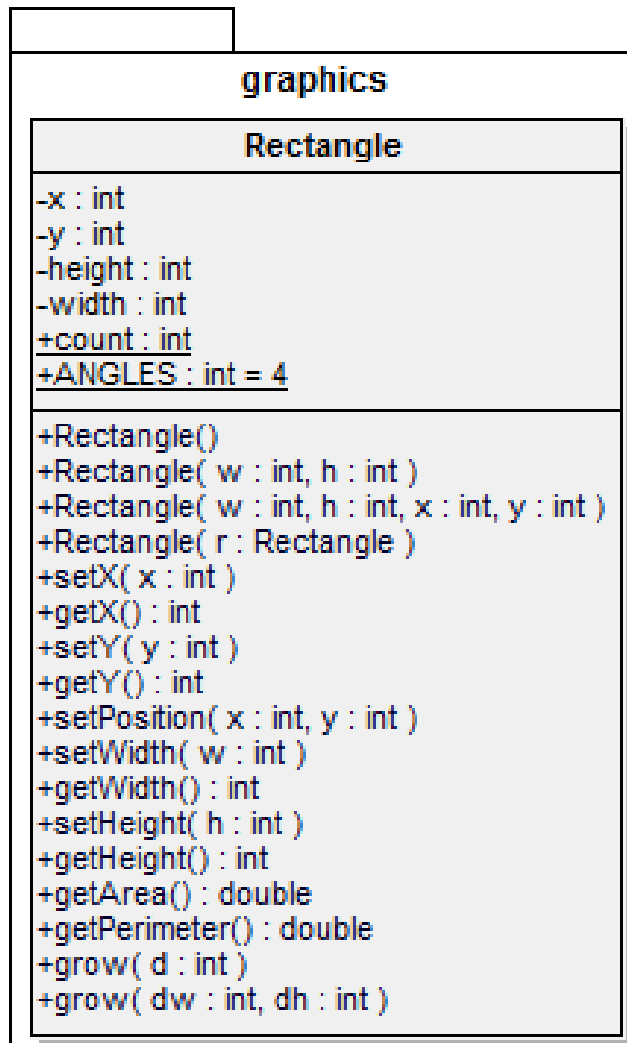
Initialisatie van de klasse-variabele via een static initialisatieblok

```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
    public static final int ANGLES = 4;  
    public static int count;  
  
    static {  
        count = 0;  
    }  
    ...  
}
```

- Het static initialisatieblok wordt 1 keer uitgevoerd bij de eerste keer dat de klasse gebruikt wordt.
- Meerdere static initialisatieblokken zijn toegelaten.



# Voorbeeld: klasse Rectangle : UML-diagram



- static variabele  
→ onderlijnd

# Opdracht 10

- Maak een klasse 'Klas' met als eigenschappen de naam van de klas en het aantal studenten.
- Maak 2 constructors: een klas die aangemaakt wordt via de default-constructor wordt '1TINx' met 0 studenten. De tweede constructor krijgt 2 waarden binnen die toegekend worden aan de 2 objectvariabelen. De eerste constructor roept de tweede op.
- Maak voor alle object-variabelen getters en setters aan.
- Voeg een klasse-variabele toe die het maximum aantal studenten bevat dat in een klas kan. Geef hieraan de waarde 40.
- Druk (in de main) het maximum aantal studenten af op het scherm.
- Doe de initialisatie van het maximum via een initialisatieblok i.p.v. tijdens de declaratie.
- Zorg dat bij creatie of wijziging van een klas-object rekening gehouden wordt met het maximum.
- Voeg een klasse-variabele toe die het aantal objecten van die klasse bevat. Verhoog deze variabele via een initialisatieblok.
- Maak een aantal klas-objecten aan en druk telkens het aantal klassen af op het scherm.
- Maak een array en voeg hierin al je Klas-objecten toe. Overloop alle objecten en druk van alle klassen naam en aantal studenten af.





### 3.4.3 Instance-methoden

#### Instance-methode

→ Moet opgeroepen worden op een concreet object van de klasse

Voorbeeld: `rec.setPosition(3, 6);`

- De methode kan gebruik maken van de instance-variabelen van dat specifieke object
- De methode kan ook gebruik maken van de klasse-variabelen
- De methode kan gebruik maken van lokale variabelen



## 3.4.4 Klasse-methoden

Klasse-methoden

= een methode die wordt opgeroepen op basis van de klasse-naam

```
ClassName.method( )
```

→ De methode kan enkel gebruik maken van de klasse-variabelen (en van lokale variabelen binnen de methode)



# Voorbeeld: klasse Rectangle

Klasse-methode die het aantal objecten teruggeeft

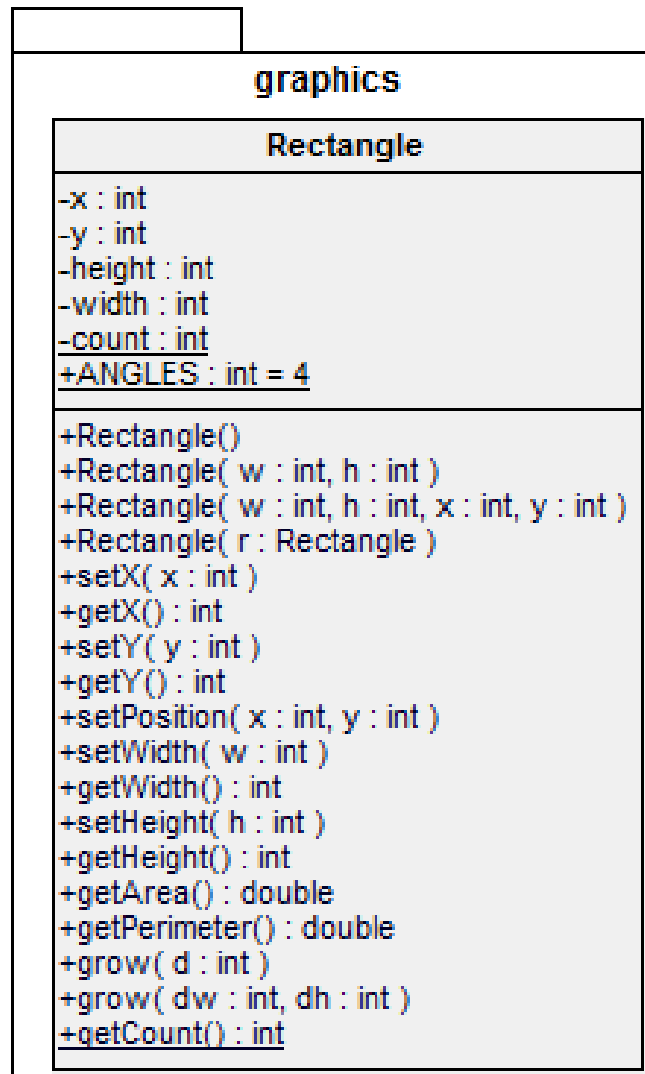
```
public class Rectangle {  
    private int width;  
    private int height;  
    private int x;  
    private int y;  
    public static final int ANGLES = 4;  
    private static int count = 0;  
  
    public static int getCount() {  
        return count;  
    }  
    ...  
}
```

- count is nu private!
- Methode getCount() is
  - public
  - static



# Voorbeeld: klasse Rectangle

Klasse-methode die het aantal objecten teruggeeft



- static methode **getCount()** is onderlijnd

# Gebruik van een klasse-methode

Opvragen van het aantal gecreëerde Rectangle-objecten

`ClassName.method( )`

In de main-methode:

```
int counter = Rectangle.getCount( );
```



# Opmerking1

De main-methode is een static method.

- We kunnen enkel klasse-variabelen gebruiken.
- We kunnen wel lokale variabelen creëren en gebruiken.



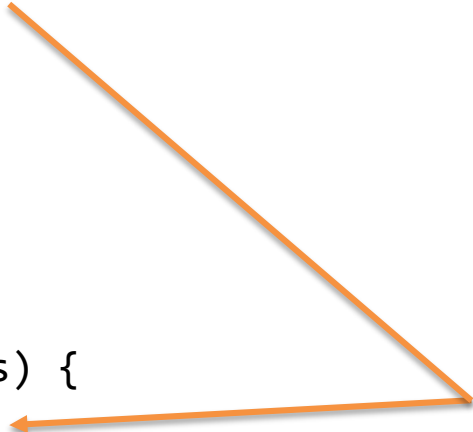
# Opmerking2

Je kan static members importeren.

→ Geen vermelding van de klasse-naam meer nodig.

Voorbeeld:

```
package graphics;  
import static graphics.Rectangle.*;  
  
public class RectangleApp {  
    public static void main (String [] args) {  
        System.out.println(getCount());  
        System.out.println(ANGLES);  
    }  
}
```



# Opmerking3

Binnen een klasse-methode is de `this`-referentie onbestaande.





## Opdracht 11

- Voeg aan de klasse 'Klas' een methode toe die het aantal Klas-objecten geeft.  
Maak de variabele die het aantal klassen bijhoudt private.
- Maak een variabele om het totaal aantal studenten bij te houden. Klasse-variabele of instance-variabele?  
Zorg ervoor dat dit totaal correct bijgehouden wordt.  
Maak eveneens een get-methode aan om dit totaal te kunnen opvragen.
- In de main-methode druk je het aantal aangemaakte klassen af en het totaal aantal studenten.
- Voeg 1 student toe aan een bepaalde klas en controleer of het totaal nog steeds klopt.



## 3.5 De klasse Math

Definitie utility-klasse

= klasse die hulpmethoden aanreikt

→ Bevatten enkel static members

→ Je kan er geen objecten van maken

De klasse Math is een utility-klasse met 2 klasse-variabelen:

Math.PI            het getal  $\pi$

Math.E            het getal e



## 3.5 De klasse Math

De klasse Math bevat dus 2 klasse-variabelen:

Math.PI	het getal $\pi$
Math.E	het getal e

En een groot aantal methoden voor wiskundige berekeningen:

- Goniometrische functies
- Machtsverheffing
- Worteltrekking
- Exponentiële functies
- ...

Voorbeeld

```
cosinus = Math.cos(angle);
```



## 3.5 De klasse Math

### Enkele functies

Methode	Omschrijving van de functie
<code>int abs(int x)</code> <code>long abs(long x)</code> <code>float abs(float x)</code> <code>double abs(double x)</code>	berekent de absolute waarde van het argument
<code>double sqrt(double x)</code>	berekent de vierkantswortel van het argument
<code>double pow(double x, double y)</code>	berekent $x^y$
<code>double random()</code>	genereert willekeurig een getal groter of gelijk aan 0 en strikt kleiner dan 1
<code>int round(float x)</code> <code>long round(double x)</code>	Berekent het dichtst bijzijnde geheel getal

**!! Uitleg van alle methoden → Java API-documentatie !!**



# Voorbeelden wiskundige functies

## 1. Verbeter, indien nodig:

### *Programma 1*

```
int x;  
double y;  
y = Math.sqrt(2);  
x = Math.sqrt(4);
```

### *Programma 2*

```
int x, y;  
x = Math.pow(2, 3);  
y = Math.round(4.7);
```

## 2. Wat is het verschil tussen `Math.abs(-1)` en `Math.abs(-1.0)`?



## Opdracht 12

Schrijf een programma om  $x^3$  te berekenen. Hierbij is  $x$  geheel en het resultaat moet ook geheel zijn.  $x$  moet via het toetsenbord ingegeven worden.

## Opdracht 13

Schrijf een programma om de diameter van een cirkel te berekenen als de oppervlakte van de cirkel gegeven is. Formule oppervlakte cirkel:  $\pi \cdot r^2$

$r$  = de straal van de cirkel

Rond het resultaat af op 2 decimalen.



## Opdracht 14

Herneem de opdracht met de klasse Klas.

Druk onderaan in de main-methode het gemiddeld aantal studenten per klas af (afgerond op 1 decimaal).

