INHOUDSOPGAVE

OVERZICHT THEORIE

Klassen, objecten, constructors

- Klassen en objecten (naam, eigenschappen en gedrag): Liang, 9.1 - 9.3.
- Constructors (objecten declareren en instantiëren): Liang, 9.4 - 9.5.
- All-args constructor, default constructor, overloading: Liang 9.4.
- Static versus non-static: Liang, 9.7.

Encapsulation en constructor chaining

- Encapsulation (inkapseling): Liang, 9.9, 10.2.
- Visibility modifiers `public` en `private`: Liang, 9.8.
- Keyword `this`: Liang, 9.14.
- Getters en setters: Liang, 9.9.
- Constanten: Liang, 2.7
- Constructor chaining: Liang, 9.14.
- Arrays van objecten: Liang, 9.11.

Klassen als attribuut

- Klassen als attribuut: Liang, 10.4.
- Unidirectionele relatie.

Overerving en toString()

- Overerving (inheritance), keyword `extends`, superklasse, subklasse: Liang, 11.2.
- Keyword `super`: Liang, 11.3
- Visibility modifier `protected`: Liang, 9.8.
- `toString()` methode: Liang, 11.6.

Polymorfisme en abstracte klassen, wrapper classes

- Polymorfisme (polymorphism): Liang, 11.7.
- Abstracte klassen en methoden: Liang, 13.2.
- Overriding: Liang, 11.4 - 11.5.
- `LocalDate` klasse.

ArrayList, instanceof

- Wrapper classes (`Integer`, `Double`, `Boolean`): Liang, 10.7 – 10.8.
- `ArrayList` klasse: Liang, 11.11.
- `foreach` loop: Liang, 7.2.7
- ArrayList als attribuut.
- `instanceof` en typecasting: Liang, 11.9.

Interfaces, StringBuilder

- Interfaces: Liang, 13.5, 13.8.
- `Comparable` interface: Liang, 13.6.
- `StringBuilder` klasse: Liang, 10.11.

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

class Fruit

objects Apple

Banana

Mango

1 KLASSEN, OBJECTEN, CONSTRUCTORS

In dit document beschrijven we per onderdeel wat de lesstof is en wat de bijbehorende opdrachten zijn. De lesstof wordt aan de hand van het voorbeeld 'Meetkunde' toegelicht. OOP (Object Oriented Programming) is een aanpak waarbij klassen van objecten gemaakt worden. Een *klasse* is een logische eenheid van attributen en methodes, die betrekking hebben op de objecten van de klasse.
Pijlers van OOP zijn: *inkapseling* (encapsulation), *abstractie* (abstraction), *overerving* (inheritance) en *polymorfisme* (polymorphism).

1. Ga naar de cursus Java Object Oriented Programming en bekijk de films van de Introduction
2. Bekijk eventueel ook Concepts of OOP
3. Bekijk ook Using-classes-as-blueprints
4. Lees Liang 9.2

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

1.1 Samenvatting

1. Object Oriented Programming heeft als doel code onderhoudbaar en herbruikbaar te maken.

A Class is like an object constructor, or a "blueprint" for creating objects.

2. In de code wordt gebruik gemaakt van *objecten*. Objecten representeren dingen in de werkelijkheid. Een object heeft eigenschappen en vertoont bepaald gedrag. Door middel van *abstractie* kiezen we voor de eigenschappen en het gedrag die van belang zijn voor het programma dat we schrijven.
3. Objecten van hetzelfde type worden gedefinieerd door een klasse, een soort blueprint voor de objecten. Een object is een *instantie* van een klasse.
4. In Java kunnen we zelf klassen (*classes*) aanmaken. In deze cursus worden klassen nagenoeg altijd aangemaakt, om zaken omtrent een 'ding' vast te leggen. De klasse heeft een naam, kent attributen/eigenschappen. Klassen kunnen ook gedrag (*methodes*) hebben. Het is het handigste om klassen zoveel mogelijk onafhankelijk van de omgeving te definiëren. Verder zijn klassen verantwoordelijk voor hun eigen inhoud.
5. Klassen die een 'ding' (persoon, afdeling, bestelling, enzovoorts) vertegenwoordigen worden in de package `model` geplaatst.
6. Objecten worden geïnstantieerd met behulp van *constructors* en het keyword new.
7. Statische attributen en methodes bestaan los van de instantie. Ze kunnen worden aangeroepen door middel van `Klasse.attribuut` of `Klasse.methode`.

1.2 Een klasse maken

To create an object of Main, specify the class name, followed by the object name, and use the keyword new:

1. Start *IntelliJ* en maak een nieuw project.
2. Kies een folder, waar je al je *IntelliJ* projecten gaat zetten. Noem het project `Meetkunde`.
3. Zorg dat links de projectstructuur zichtbaar is en zorg dat `src` geselecteerd is.

4. Maak een nieuw package aan onder src en noem dit `model`.
   Vanaf nu zetten we alle klassen, die objecten definiëren bij elkaar in een package `model`.
5. Zorg dat `model` geselecteerd is.
6. Maak een nieuwe Java klasse aan en noem dit `Cirkel`.
7. Nu zie je het volgende:

```
package model;

public class Cirkel {
}
```

↑ Je ziet hier zowel de naam van het package als die van de klasse terugkomen.

We geven eerst de attributen aan. De keuze voor attributen is een onderdeel van de *abstraction* in OOP. Een cirkel heeft een middelpunt met een x-coördinaat en een y-coördinaat, en een straal. Een cirkel krijgt ook een kleur.

Java'da bir sınıf (class) yazarken, o şeyin (örneğin bir çemberin) önemli özelliklerini seçeriz. Bu seçme işlemine soyutlama (abstraction) denir.

8. Zorg dat je het volgende op je scherm krijgt:

```
package model;

public class Cirkel {
    public double straal;
    public double middelpuntX;
    public double middelpuntY;
    public String kleur;
}
```

Gerçek hayatta bir çemberin:

Middelpunt (merkez) vardır:

x ve y koordinatlarıyla belirtilir.

Straal (yarıçap) vardır.

Kleur (renk) olabilir.

↑ Dit betekent dat ieder object van de klasse `Cirkel` drie attributen van het type double heeft en een attribuut van het type String.

Als de straal van een cirkel bekend is, dan kan de cirkel zelf de omtrek en de oppervlakte berekenen. Hieronder volgt hoe dat gaat.

9. Voeg de twee methodes toe volgens de code hieronder:

Çember nesnesinin zaten bir straal (yarıçap) bilgisi var.

O zaman bu nesne kendi alanını ve çevresini hesaplayabilir.

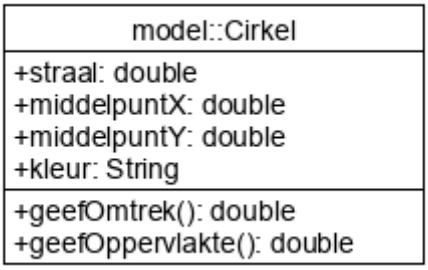
Bunu yapması için, sınıfına (class'a) methodlar ekleriz.

```
package model;

public class Cirkel {
    public double straal;
    public double middelpuntX;
    public double middelpuntY;
    public String kleur;

    public double geefOmtrek() {
        return 2 * Math.PI * straal;
    }

    public double geefOppervlakte() {
        return Math.PI * straal * straal;
    }
}
```

↑ De omtrek van een cirkel is 2π * straal en de oppervlakte van een cirkel is π*(straal)^2. Beide methodes retourneren het resultaat als een waarde van het type `double`.

↑ We zetten in de klasse de code klaar om daarmee objecten van het type Cirkel te kunnen maken met herbruikbare methoden. Zodra een cirkel object aangemaakt is met straal 3, dan kunnen de methoden aangeroepen worden om de omtrek en oppervlakte van die cirkel uit te rekenen.

We gebruiken UML klassendiagrammen voor een overzicht van de klasse. Het klassendiagram van de klasse Cirkel ziet er zo uit:



De bovenste cel bevat de naam van het package (`model`) en de naam van de klasse (`Cirkel`), gescheiden door twee keer een dubbele punt.
De tweede cel bevat de attributen (straal, middelpuntX , middelpuntY en kleur). De plus betekent dat de attribuut public is (overal zichtbaar en dus beschikbaar). Vervolgens zie je de naam van de attribuut en het type.
De derde cel bevat de methoden (geefOmtrek en geefOppervlakte). De plus betekent weer dat de methode public is. Vervolgens zie je de naam van de methode en na de dubbele punt het geretourneerde type.

1.3 Een klasse gebruiken

1. Lees Liang 9.4 en 9.5.
2. Zorg dat links de projectstructuur zichtbaar is en zorg dat **src** geselecteerd is.
3. Maak een nieuw package aan onder **src** en noem dit `controller`.

4. Zorg dat `controller` geselecteerd is.
5. Maak een nieuwe Java klasse aan en noem dit ***MeetkundeLauncher***.
6. Maak de main-methode aan.
7. Nu zie je het volgende:

```
package controller;

public class MeetkundeLauncher {
    public static void main(String[] args) {

    }
}
```

Je kunt de klasse `Cirkel` nu gebruiken in je launcher.

8. Type `Cirkel mijnEersteCirkel = new Cirkel();`
   Je scherm ziet er als volgt uit:

```
package controller;

import model.Cirkel;

public class MeetkundeLauncher {
    public static void main(String[] args) {
        Cirkel mijnEersteCirkel = new Cirkel();
    }
}
```

↑ De klasse `Cirkel` wordt automatisch geïmporteerd, omdat deze klasse nodig is voor het kunnen uitvoeren van de code.
↑ Cirkel `mijnEersteCirkel` vertelt de Java Compiler dat je een object van de klasse `Cirkel` declareert met de naam `mijnEersteCirkel`. `mijnEersteCirkel` is de referentie variabele (reference variable).
↑ Dit object bestaat nog niet. Het object wordt geïnstantieerd door het Java keyword `new`.
↑ Als de hele regel is uitgevoerd heb je een nieuw object van de klasse `Cirkel`, dat je via de variabele `mijnEersteCirkel` kunt benaderen.

Type het volgende onder de net ingevoerde regel: `mijnEersteCirkel` gevolgd door een punt. Je ziet dan het volgende:

**Cirkel mijnEersteCirkel satırı, Java derleyicisine şu bilgiyi verir: Cirkel sınıfından bir nesne tanımlanıyor, bu nesnenin ismi mijnEersteCirkel.
mijnEersteCirkel burada bir referans değişkenidir (yani bellekteki Cirkel nesnesine ulaşmak için kullanılır).
Ama bu satırda nesne henüz oluşturulmuş değildir.
Nesne, Java'nın new anahtar kelimesiyle oluşturulur (instantiate edilir).
Satırın tamamı çalıştırıldığında, Cirkel sınıfından yeni bir nesne oluşturulmuş olur ve bu nesneye mijnEersteCirkel değişkeni ile erişebilirsin**

↑ Alle attributen en methodes in de klasse `Cirkel` zijn nu beschikbaar. Je kunt de attributen aanpassen en lezen en de methodes aanroepen.

Zorg dat je main-methode er als volgt uitziet:

```
public static void main(String[] args) {
    Cirkel mijnEersteCirkel = new Cirkel();
    mijnEersteCirkel.straal = 3;
    mijnEersteCirkel.middelpuntX = 3;
    mijnEersteCirkel.middelpuntY = -2;
    mijnEersteCirkel.kleur = "groen";
    System.out.println(mijnEersteCirkel.geefOmtrek());
    System.out.println(mijnEersteCirkel.geefOppervlakte());
}
```

↑ Na de declaratie en initialisatie van het Cirkel object wordt aan de attributen van het object een waarde toegekend.

↑ In de laatste twee regels worden de methodes `geefOmtrek()` en `geefOppervlakte()` aangeroepen op `mijnEersteCirkel`. Omdat de `straal` van de cirkel op 3 gezet is, kunnen de methoden nu hun werk doen met `straal` = 3.

Run je programma. Je uitvoer ziet er zo uit:

```
18.84955592153876
28.274333882308138
```

Probeer enkele andere waarden en bekijk de uitvoer.

1.4 Constructors

1. Bekijk op ***LinkedIn Learning:*** building objects with a constructor en using concrete instances

In deze oefening voeg je drie constructors toe met de volgende *signatures* (dit wordt *overloading* genoemd):

Constructor (Yapıcı Metot)
Java'da bir sınıftan (örneğin Cirkel) nesne oluşturmak için constructor denilen özel bir metot kullanılır.
Constructor nedir?
→ Nesne (object) oluşturulurken ilk çalışan metottur.
→ Genellikle başlangıç değerlerini ayarlamak için kullanılır.

- Een *all-args constructor*: `Cirkel(double straal, double middelpuntX, double middelpuntY, String kleur)` 1. All-args constructor:Yani: tüm bilgiler kullanıcı tarafından girilecek.
- Een *default* constructor: `Cirkel()` Default constructor:Yani: hiçbir bilgi verilmez, her şeyin varsayılan değeri kullanılır.
- Een constructor met alleen de *straal*: `Cirkel(double straal)` 3. Yalnızca yarıçap (straal) constructor'ı

Gebruik daarbij de volgende default waarden:
`straal = 1`, `middelpuntX = 0`, `middelpuntY = 0` en `kleur = "wit"`.

TIP: Je kunt heel gemakkelijk constructors toevoegen door te kiezen voor Code > Generate... > Constructor. Selecteer de attributen, die je wilt gebruiken in de constructor en *IntelliJ* doet de rest.

---

2. Voeg de volgende code toe aan je klasse Cirkel:

```
public Cirkel(double straal, double middelpuntX, double middelpuntY, String
kleur) {
    this.straal = straal;
    this.middelpuntX = middelpuntX;
    this.middelpuntY = middelpuntY;
    this.kleur = kleur;
}

public Cirkel(double straal) {
    this.straal = straal;
    this.middelpuntX = 0;
    this.middelpuntY = 0;
    this.kleur = "wit";
}

public Cirkel() {
    this.straal = 1;
    this.middelpuntX = 0;
    this.middelpuntY = 0;
    this.kleur = "wit";
}
```

Bir sınıftan (class) yeni bir nesne (object) oluşturmak için kullanılan özel bir metottur. Ama dikkat:
Bir constructor'ın dönüş tipi yoktur (bu yüzden void yazılmaz).
Constructor'ın adı sınıfla aynı olmalıdır.

Aynı isimle (yani Cirkel) birden fazla constructor yazabilirsin, yeter ki parametreleri farklı olsun.
Bu özelliğe overloading denir.

↑ Een constructor heeft geen retourwaarde, dus er staat ook niet `void`.
↑ Elke constructor heeft altijd exact dezelfde naam als de klasse.
↑ Er kunnen meer constructors zijn, zolang ze verschillende *signatures* hebben (*overloading*).
↑ In de all-args constructor komen de namen `straal`, `middelpuntX`, `middelpuntY` en `kleur` twee keer voor. Een keer als attribuut van de klasse `Cirkel` en een keer als parameter van de constructor. Om verwarring te voorkomen kent Java het keyword `this`. Daarmee wordt verwezen naar de eigen klasse. `this.straal` verwijst daarom naar het attribuut van de klasse. De code `this.straal = straal` kent daarmee de waarde die is meegegeven als parameter toe aan het attribuut.

3. Pas de code in de launcher als volgt aan:

```
Cirkel mijnAllArgsCirkel = new Cirkel(3, 1, 4, "groen");
System.out.println(mijnAllArgsCirkel.geefOmtrek());
System.out.println(mijnAllArgsCirkel.geefOppervlakte());

Cirkel mijnDefaultCirkel = new Cirkel();
System.out.println(mijnDefaultCirkel.geefOmtrek());
System.out.println(mijnDefaultCirkel.geefOppervlakte());

Cirkel mijnStraalCirkel = new Cirkel(6);
System.out.println(mijnStraalCirkel.geefOmtrek());
System.out.println(mijnStraalCirkel.geefOppervlakte());
```

4. Run je programma. Je uitvoer ziet er zo uit:

```
18.84955592153876
28.274333882308138
6.283185307179586
3.141592653589793
37.69911184307752
113.09733552923255
```

5. Probeer enkele andere waarden en bekijk de uitvoer.

1.5 Static versus non-static

1. Bekijk: difference between class and instance members
2. Lees Liang 9.7.
3. Voeg de volgende code toe aan de klasse `Cirkel`:

```
public static String geefDefinitie() {
    return "Een cirkel is een verzameling punten, die allemaal dezelfde
afstand tot een middelpunt hebben.";
}
```

↑ Let op het keyword **`static`**. De methode is een klasse methode. Voor alle objecten van de klasse is de methode hetzelfde. De methode is niet afhankelijk van specifieke object eigenschappen.

4. Voeg de volgende code als eerste regel toe aan de klasse `MeetkundeLauncher`:
   `System.out.println(Cirkel.geefDefinitie());`
   zodat je code er als volgt uitziet:

Java'da bir şey static olarak tanımlanırsa, bütün sınıfa (class) aittir, yani:
O özelliğe ya da metoda nesne oluşturmadan doğrudan sınıf üzerinden ulaşabilirsin.

Eğer static yoksa, o özellik ya da metot:
yalnızca bir nesne üzerinden kullanılabilir.
her nesne için farklı bir değer taşıyabilir (örneğin her dairenin (cirkel) farklı bir rengi olabilir).

✅ geefDefinitie() metodu static olduğu için nesne oluşturmadan çağrılabiliyor.

❌ mijnAllArgsCirkel.geefDefinitie() gibi bir kullanım hata verir, çünkü geefDefinitie() nesneye ait değil, sınıfa ait bir metottur.

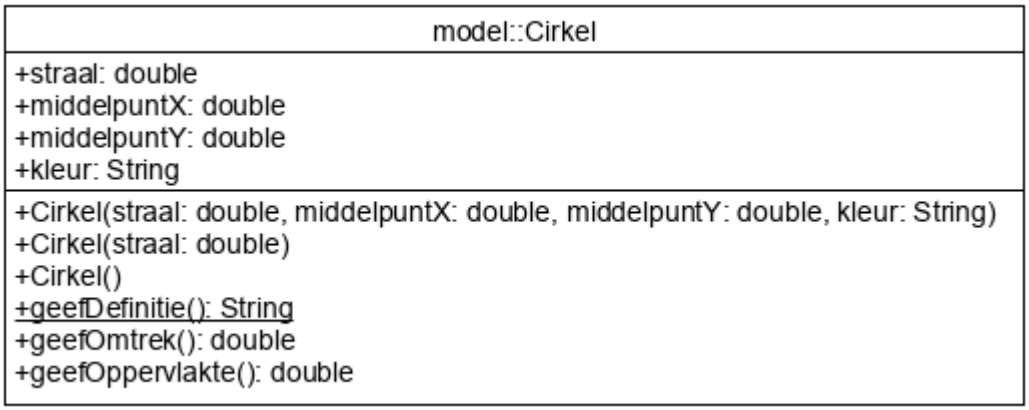
```
System.out.println(Cirkel.geefDefinitie());
Cirkel mijnAllArgsCirkel = new Cirkel(3, 1, 4, "groen");
System.out.println(mijnAllArgsCirkel.geefOmtrek());
System.out.println(mijnAllArgsCirkel.geefOppervlakte());

Cirkel mijnDefaultCirkel = new Cirkel();
System.out.println(mijnDefaultCirkel.geefOmtrek());
System.out.println(mijnDefaultCirkel.geefOppervlakte());

Cirkel mijnStraalCirkel = new Cirkel(6);
System.out.println(mijnStraalCirkel.geefOmtrek());
System.out.println(mijnStraalCirkel.geefOppervlakte());
```

↑ De methode `geefDefinitie()` is beschikbaar zonder dat er een instantie van de klasse `Cirkel` is gemaakt! De methode wordt aangeroepen via de klasse, dus als `Cirkel.geefDefinitie()`. De methode is daarentegen niet beschikbaar voor de instanties, dat wil zeggen, `mijnAllArgsCirkel.geefDefinitie()` kan niet.

5. Run dit programma.

1.6 Klassendiagram

Het klassendiagram van de klasse `Cirkel` ziet er inmiddels zo uit:



+ betekent public
Onderstreept betekent static
Je ziet nu ook dat er bij een methode de *signature* getoond wordt, dus de naam en het type van de parameter(s).

1.7 Opdrachten

1. Doe opdracht **Klassen-Objecten-1-Bedrijf**. Deze opdracht vind je in het document ***Opdracht Bedrijf***. Dit document bevat alle opdrachten die betrekking hebben op de case Bedrijf. Alle opdrachten zijn van het niveau basis.
2. Doe opdracht **Klassen-Objecten-2-Voetbalscores** (niveau: basis).

Encapsulation, bir sınıfın iç yapısını (özellikle değişkenlerini ve bazı metodlarını) dışarıdan doğrudan erişime kapatma işlemidir. Bu, yazdığın programın daha güvenli, kontrol edilebilir ve modüler olmasını sağlar.

2 ENCAPSULATION EN CONSTRUCTOR CHAINING

Getter: Bilgiyi okumak için

Setter: Bilgiyi değiştirmek için

2.1 Samenvatting

1. *Encapsulation* (inkapselen) is het verbergen van de eigen attributen en/of methodes, waarvan de klasse niet wil dat men ze direct benadert.
2. De *visibility modifier* `private` zorgt dat deze attributen en methodes niet meer direct benaderbaar zijn.
3. In plaats daarvan kunnen er voor attributen *getters* (voor het opvragen van een attribuut) en *setters* (voor het aanpassen van een attribuut) gedefinieerd worden. Dit geeft de mogelijkheid om voor sommige attributen geen *getter* of *setter* te definiëren of om extra controles in te bouwen in de *getter* of de *setter*.
4. *Constructor chaining* is het aanroepen van een specifiekere constructor door een generiekere constructor. Dit gebeurt met behulp van het keyword `this()`.
5. Objecten kunnen ook in een array geplaatst worden en via de index benaderd worden.

2.2 Encapsulation

Constructor chaining, bir kurucunun (constructor) içinde başka bir constructor'ı çağırmak demektir. Böylece kod tekrarını önleyebilir ve varsayılan değerleri bir yerde tanımlayabilirsin.

1. Bekijk: what is encapsulation
2. Ga verder met: discovering access modifiers
3. En bekijk ook nog: implementing encapsulation
4. Lees Liang 9.8, 9.9 en 10.2
5. Indien nodig kun je het project ***OOP Meetkunde Klassen-Objecten*** van DLO halen. Dit bevat de klassen `MeetkundeLauncher` en `Cirkel` zoals je die in het vorige onderdeel hebt afgesloten.

Er zijn twee manieren om een bestaand project te openen:

1. Via het startscherm
   a) Je opent een bestaand project door op het startscherm van *IntelliJ* de optie Open te selecteren.
   b) Selecteer dan de **map** waar het project staat (daar staat een – mogelijk verborgen – folder met de naam **.idea** en een bestand met de naam **[projectnaam].iml**. Je ziet dat mappen, die een project bevatten een apart icoon hebben.
   c) Klik op Ok en *IntelliJ* laadt dan het hele project in.
2. Vanuit *IntelliJ* zelf
   a) Selecteer File > Open… in het menu.
   b) Ga dan verder met stap *1b)* hierboven.

6. Verander de code in de klasse `Cirkel` als volgt:

```
private double straal;
private double middelpuntX;
private double middelpuntY;
private String kleur;
```

↑ `private` wil zeggen dat het attribuut of de methode alleen beschikbaar is binnen de klasse zelf. Het is dus niet meer vanuit de klasse `MeetkundeLauncher` beschikbaar.

7. Voeg getters en setters in de klasse `Cirkel` toe.

TIP: Je kunt heel gemakkelijk getters en setters toevoegen door te kiezen voor Code > Generate... > Getter and setter. Selecteer de attributen, waarvan je de getters en setters wilt maken en *IntelliJ* doet de rest.
TIP: Je kunt ook alles in één keer doen door te kiezen voor Refactor > Encapsulate Fields... Je ziet dan een uitgebreider scherm, waarbij je – behalve de getters en setters – ook kunt aangeven of je de attributen `private` wilt maken.

---

8. Pas de main-methode in de launcher als volgt aan:

```
public static void main(String[] args) {
    Cirkel mijnDefaultCirkel = new Cirkel();
    System.out.println(mijnDefaultCirkel.getStraal());
    System.out.println(mijnDefaultCirkel.geefOmtrek());
    System.out.println(mijnDefaultCirkel.geefOppervlakte());
    mijnDefaultCirkel.setStraal(3);
    System.out.println(mijnDefaultCirkel.geefOmtrek());
    System.out.println(mijnDefaultCirkel.geefOppervlakte());
}
```

9. Run je programma. Je uitvoer ziet er zo uit:

```
1.0
6.283185307179586
3.141592653589793
18.84955592153876
28.274333882308138
```

10. Je kunt een setter gebruiken om te zorgen dat er geen ongewenste dingen gedaan worden. Zo is een cirkel met een straal die kleiner dan of gelijk aan 0 is niet mogelijk. Dit kun je regelen met een setter. Verander de methode `setStraal()` als volgt:

```
public void setStraal(double straal) {
    if (straal <= 0) {
        System.out.println("De straal moet positief zijn. De straal wordt
op 1 gezet");
        this.straal = 1.0;
    } else {
        this.straal = straal;
    }
}
```

11. Probeer nu de setter uit, door in de main-methode de volgende regel toe te voegen:

```
mijnDefaultCirkel.setStraal(-3);
```

Je krijgt nu de foutmelding te zien.

12. Het is echter nog steeds mogelijk om in de constructor een cirkel aan te maken met een negatieve straal. Je kunt dit oplossen door ook in de constructor de `setStraal()` methode aan te roepen. Verander de code

```
this.straal = straal;
```

in **twee** constructors naar

```
setStraal(straal);
```

13. Probeer enkele andere waarden en bekijk de uitvoer.

2.3 Constanten

1. Voeg de volgende regel toe aan de klasse `Cirkel` (boven de andere attributen):
   `private final static double GRENSWAARDE_GROOT_FIGUUR = 100.0;`
   zodat je attributen er als volgt uitzien:

```
private final static double GRENSWAARDE_GROOT_FIGUUR = 100.0;
private double straal;
private double middelpuntX;
private double middelpuntY;
private String kleur;
```

↑ `final` wil zeggen dat de variabele niet meer veranderd kan worden en dus een constante is
↑ `static` wil zeggen dat de variabele voor alle objecten hetzelfde is
↑ Een *klasseconstante* is altijd `final static`

2. Voeg de volgende methode toe aan de klasse `Cirkel`:

```
public String vertelOverGrootte() {
    if ( geefOppervlakte() > GRENSWAARDE_GROOT_FIGUUR ) {
        return "Ik ben groot!!!";
    } else {
        return "Ik ben klein!!!";
    }
}
```

Pas de main-methode als volgt aan:

```
public static void main(String[] args) {
    Cirkel mijnDefaultCirkel = new Cirkel();
    System.out.println(mijnDefaultCirkel.getStraal());
    System.out.println(mijnDefaultCirkel.geefOppervlakte());
    System.out.println(mijnDefaultCirkel.vertelOverGrootte());

     mijnDefaultCirkel.setStraal(3);
    System.out.println(mijnDefaultCirkel.geefOppervlakte());
    System.out.println(mijnDefaultCirkel.vertelOverGrootte());

     mijnDefaultCirkel.setStraal(6);
    System.out.println(mijnDefaultCirkel.geefOppervlakte());
    System.out.println(mijnDefaultCirkel.vertelOverGrootte());
}
```

3. Run je programma. Je uitvoer ziet er zo uit:

```
1.0
3.141592653589793
Ik ben klein!!!
28.274333882308138
Ik ben klein!!!
113.09733552923255
Ik ben groot!!!
```

4. Probeer enkele andere waarden en bekijk de uitvoer.

2.4 Constructor chaining

1. Lees Liang 9.14. Lees de **tip** in 9.14.2 aandachtig!
2. Verander de code van de constructor met signature `Cirkel(double straal)` als volgt:

```
public Cirkel(double straal) {
    this(straal, 0, 0, "wit");
}
```

↑ Merk op dat `this` weer verwijst naar de klasse. `this(straal, 0, 0, "wit")` verwijst nu naar de all-args constructor binnen de klasse. De default waarden voor `middelpuntX`, `middelpuntY` en `kleur` worden meegegeven.

3. Verander de code van de default constructor – met signature `Cirkel()` – als volgt:

```
public Cirkel() {
    this(1);
}
```

↑ `this(1)` verwijst nu naar de constructor met signature `Cirkel (double straal)` binnen de klasse. De default waarde voor `straal` wordt meegegeven.
↑ De meest generieke constructor (de default constructor) roept de meer specifieke constructor aan, die vervolgens de meest specifieke constructor (de all-args constructor) aanroept.
↑ Merk op dat iedere default waarde maar op één plaats wordt bepaald.

4. Pas de code in de launcher als volgt aan (de code kun je kopiëren van 1.4 stap 3):

```
Cirkel mijnAllArgsCirkel = new Cirkel(3, 1, 4, "groen");
System.out.println(mijnAllArgsCirkel.geefOmtrek());
System.out.println(mijnAllArgsCirkel.geefOppervlakte());

Cirkel mijnDefaultCirkel = new Cirkel();
System.out.println(mijnDefaultCirkel.geefOmtrek());
System.out.println(mijnDefaultCirkel.geefOppervlakte());

Cirkel mijnStraalCirkel = new Cirkel(6);
System.out.println(mijnStraalCirkel.geefOmtrek());
System.out.println(mijnStraalCirkel.geefOppervlakte());
```

5. Run je programma. Je uitvoer ziet er zo uit:

```
18.84955592153876
28.274333882308138
6.283185307179586
3.141592653589793
37.69911184307752
113.09733552923255
```

↑ Merk op dat de uitvoer hetzelfde is als bij ***1.4.4***

6. Probeer enkele andere waarden en bekijk de uitvoer.

2.5 Arrays van objecten

1. Lees Liang 9.11.
2. Verander de code van de main-methode in de launcher als volgt:

```
Cirkel[] mijnCirkelArray = new Cirkel[3];
mijnCirkelArray[0] = new Cirkel(3, 1, 4, "groen");
mijnCirkelArray[1] = new Cirkel();
mijnCirkelArray[2] = new Cirkel(6);
for (int arrayTeller = 0; arrayTeller < mijnCirkelArray.length;
arrayTeller++) {
    System.out.println(mijnCirkelArray[arrayTeller].geefOmtrek());
    System.out.println(mijnCirkelArray[arrayTeller].geefOppervlakte());
}
```

↑ De syntax voor het declareren en instantiëren van een array is hetzelfde.
↑ De drie objecten van de klasse `Cirkel` worden in een array geplaatst.
↑ Met een for-loop worden de drie cirkels benaderd.
↑ `mijnCirkelArray[arrayTeller]` is dus een object van de klasse `Cirkel`

3. Run je programma. Je uitvoer ziet er zo uit:

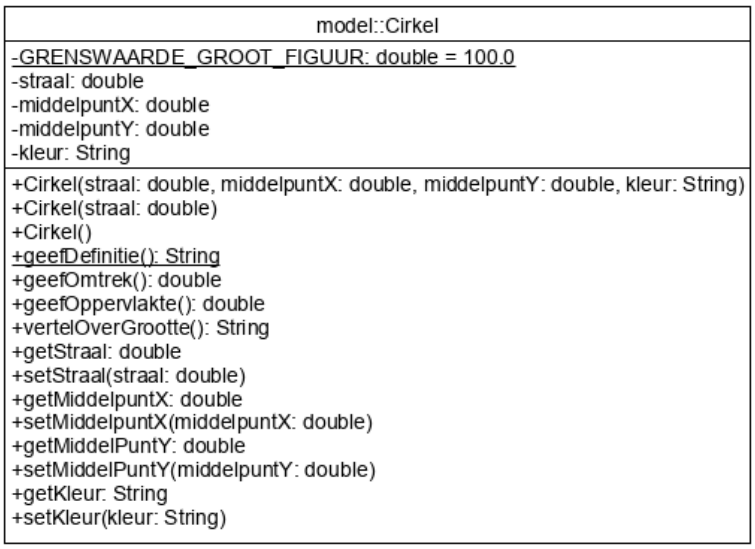
```
18.84955592153876
28.274333882308138
6.283185307179586
3.141592653589793
37.69911184307752
113.09733552923255
```

↑ Merk op dat de uitvoer hetzelfde is als bij ***1.4.4*** en ***2.4.5***.

4. Probeer enkele andere waarden en bekijk de uitvoer.

2.6 Klassendiagram

Het klassendiagram van de klasse `Cirkel` ziet er inmiddels zo uit:



+ betekent public
- betekent private
ALL_CAPS betekent constante
Onderstreept betekent static
Let op: in het algemeen worden de getters en setters niet getoond in het klassendiagram. Dat zullen we vanaf nu niet meer doen.

2.7 Opdrachten

1. Bekijk: exploring encapsulation in java string class
2. Doe Opdracht **Encapsulation-1 Bedrijf.**
3. Doe Opdracht **Encapsulation-2 SimOnly** (niveau: basis).
4. Mocht je nog zin en tijd hebben, doe dan Opdracht **Encapsulation-3 Kofferslot** (niveau: gevorderd).

Bir sınıf, başka bir sınıfın içinde özellik (attribute) olarak kullanılabilir.

3 KLASSEN ALS ATTRIBUUT

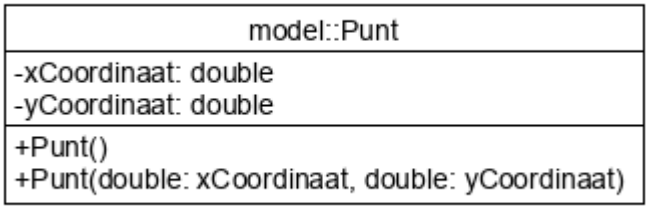
İki sınıf arasındaki ilişki farklı şekillerde sınıflandırılabilir, en yaygın olanı tek yönlü ilişki (unidirectionele relatie)'dir.

3.1 Samenvatting

1. Klassen kunnen worden gebruikt als attribuut in een andere klasse.
2. De relatie tussen twee klassen kan op verscheidene manieren worden geclassificeerd, de meest gebruikte manier is de *unidirectionele relatie*.

3.2 Klassen als attribuut

1. Lees Liang 10.3 en 10.4.
2. Indien nodig kun je het project ***OOP Meetkunde Encapsulation*** van ***DLO*** halen. Dit bevat de klassen `MeetkundeLauncher` en `Cirkel` zoals je die in het vorige onderdeel hebt afgesloten.
3. Maak binnen het package `model` een nieuwe Java klasse aan met de naam `Punt`.
4. Vul de klasse volgens het volgende klassendiagram.
5. Gebruik de defaultwaarde 0 voor `xCoordinaat` en `yCoordinaat`
6. Gebruik constructor chaining
7. Voeg de getters en de setters toe
   (De hele code vind je aan het einde van dit hoofdstuk.)



8. Hieronder wordt de klasse `Cirkel` opnieuw opgebouwd met een attribuut van de klasse `Punt` in plaats van de attributen `middelpuntX` en `middelpuntY`.
9. Verwijder:
10. de attributen middelpuntX en middelpuntY.
11. de all-args constructor en de constructor met alleen de straal.
12. de getters en setters van middelpuntX en middelpuntY.
13. Voeg een attribuut `middelpunt` van de klasse `Punt` toe aan de klasse `Cirkel`, zoals hieronder:

```
private final static double GRENSWAARDE_GROOT_FIGUUR = 100.0;
private double straal;
private Punt middelpunt;
private String kleur;
```

↑ De klasse `Cirkel` heeft nu een attribuut met de naam `middelpunt` van de klasse `Punt`.

14. Voeg de getter en setter voor het attribuut `middelpunt` toe. Dit is wat je krijgt:

```
public Punt getMiddelpunt() {
    return middelpunt;
}

public void setMiddelpunt(Punt middelpunt) {
    this.middelpunt = middelpunt;
}
```

↑ Merk op dat het retourtype van `getMiddelpunt()` een object van de klasse `Punt` is
↑ De parameter van `setMiddelpunt()` is ook een object van de klasse `Punt`

15. Voeg de all-args constructor toe:

```
public Cirkel(double straal, Punt middelpunt, String kleur) {
    this.straal = straal;
    this.middelpunt = middelpunt;
    this.kleur= kleur;
}
```

Let op de signature van de constructor. Er zijn drie parameters, die overeenkomen met de attributen van de klasse `Cirkel`.

16. Voeg de volgende constructor `Cirkel(double straal)` toe en pas constructor chaining toe:

```
public Cirkel(double straal) {
    this(straal, new Punt(), "wit");
}
```

↑ Net zoals in ***paragraaf 2.4*** verwijst deze constructor naar de all-args constructor.
↑ De all-args constructor verwacht een parameter van de klasse `Punt`. Deze wordt verkregen door een nieuwe instantie aan te maken en mee te geven als parameter.
↑ De default waarde voor het attribuut `Cirkel.middelpunt` wordt bepaald door de default constructor van de klasse `Punt`.
↑ De default constructor voor de klasse `Cirkel` hoeft niet te worden aangepast. Deze roept nog steeds de specifiekere constructor aan met de default waarde 1 voor de straal.

17. Controleer of je de drie constructors hebt

Ook de code in de launcher moet worden aangepast.

18. Pas de code in de main-methode aan, zoals hieronder (als het goed is, hoef je alleen de all-args constructor in de code aan te passen):

```
Cirkel[] mijnCirkelArray = new Cirkel[3];
mijnCirkelArray[0] = new Cirkel(3, new Punt(1,4), "groen");
mijnCirkelArray[1] = new Cirkel();
mijnCirkelArray[2] = new Cirkel(6);

for (int arrayTeller = 0; arrayTeller < mijnCirkelArray.length;
arrayTeller++) {
    System.out.println(mijnCirkelArray[arrayTeller].geefOmtrek());
    System.out.println(mijnCirkelArray[arrayTeller].geefOppervlakte());
}
```

↑ De regel `mijnCirkelArray[0] = new Cirkel(3, new Punt(1,4), “groen”)` maakt in de constructoraanroep ook een object aan van de klasse `Punt` met de coördinaten (1, 4).

↑ Dit object wordt dan meegegeven als tweede parameter in de all-args-constructor.

19. Run je programma. Je uitvoer ziet er zo uit:

```
18.84955592153876
28.274333882308138
6.283185307179586
3.141592653589793
37.69911184307752
113.09733552923255
```

↑ Merk op dat de uitvoer hetzelfde is als in de eerdere oefeningen.

20. Probeer enkele andere waarden en bekijk de uitvoer

21. Voeg de volgende twee regels code toe aan je for-loop

```
System.out.println(mijnCirkelArray[arrayTeller].getMiddelpunt().
 getxCoordinaat());
System.out.println(mijnCirkelArray[arrayTeller].getMiddelpunt().
 getyCoordinaat());
```

↑ `mijnCirkelArray[arrayTeller]` verwijst naar een object van de klasse `Cirkel`.

↑ `mijnCirkelArray[arrayTeller].getMiddelpunt()` is de getter binnen de klasse `Cirkel`, die een object van de klasse `Punt` teruggeeft.

↑ `mijnCirkelArray[arrayTeller].getMiddelpunt().getxCoordinaat()` is de getter binnen de klasse `Punt`, die de waarde van het attribuut `xCoordinaat` teruggeeft.

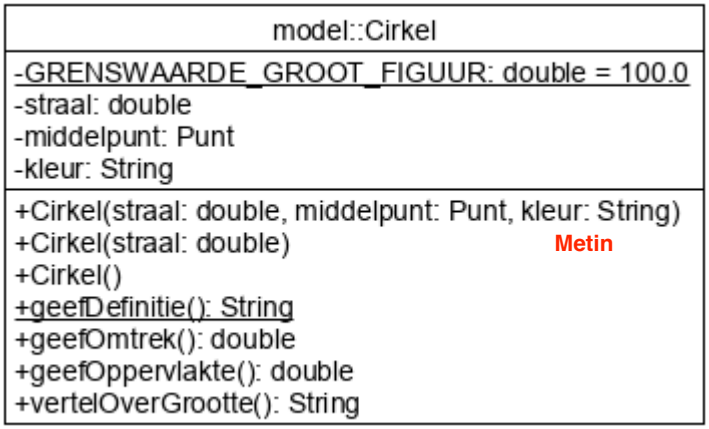
22. Run je programma. Je uitvoer ziet er zo uit:

```
18.84955592153876
28.274333882308138
1.0
4.0
6.283185307179586
3.141592653589793
0.0
0.0
37.69911184307752
113.09733552923255
0.0
0.0
```

23. Probeer enkele andere waarden en bekijk de uitvoer

3.3 Klassendiagram

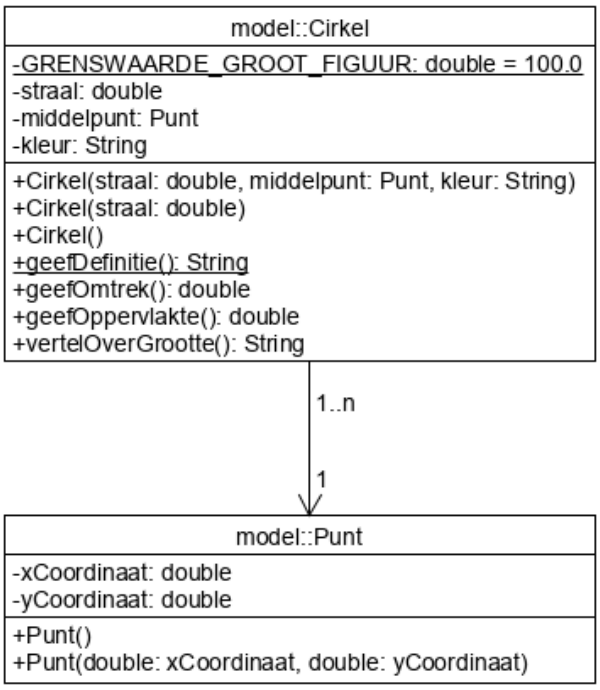
Het klassendiagram van de klasse `Cirkel` ziet er inmiddels zo uit:



+ betekent public
- betekent private
ALL_CAPS betekent constante
Onderstreept betekent static
De getters en setters zijn weggelaten.

3.4 De relatie tussen Cirkel en Punt

Het volgende klassendiagram geeft de relatie weer tussen de klasse `Cirkel` en de klasse `Punt`.



Een Cirkel bevat één Punt:
Her Cirkel nesnesi, tek bir Punt nesnesine sahiptir. Yani Cirkel sınıfının içinde Punt tipinde bir alan (field) vardır.

Een Punt kan bij één of meer Cirkels horen:
Bir Punt nesnesi birden fazla Cirkel için orta nokta olabilir. Ancak Punt nesnesinin kaç tane Cirkel ile ilişkili olduğu önemli değil çünkü...

De Cirkel weet van de Punt af:
Cirkel nesnesi, hangi Punt nesnesini kullandığını bilir. Yani Cirkel sınıfında Punt türünde bir referans vardır.

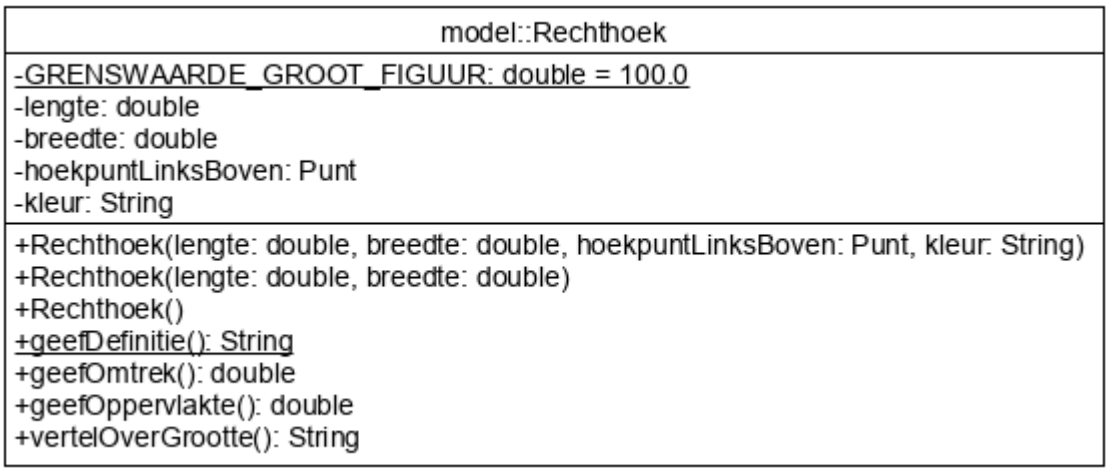
De Punt weet niets van de Cirkel waartoe deze behoort:
Ancak Punt nesnesi Cirkel hakkında hiçbir şey bilmez. Punt sınıfında Cirkel ile ilgili herhangi bir referans veya bilgi yoktur.

Deze relatie heet unidirectioneel:
Bu ilişkiye tek yönlü ilişki (unidirectional association) denir.
Cirkel → Punt doğru bir ok (açık ok) ile gösterilir.
Punt ise Cirkel hakkında hiçbir şey bilmez, bağlantısı yoktur.

- Een `Cirkel` bevat één `Punt`;
- Een `Punt` kan bij één of meer `Cirkels` horen;
- De `Cirkel` weet van de `Punt` af;
- De `Punt` weet niets van de `Cirkel` waartoe deze behoort;
- Deze relatie heet *unidirectioneel* en wordt weergegeven door een open pijl aan de kant van de class waarnaar verwezen wordt.

3.5 De klasse Rechthoek

24. Maak een nieuwe klasse aan volgens het onderstaande klassendiagram. Gebruik de volgende extra informatie.
25. De default waarde voor `lengte` is 2
26. De default waarde voor `breedte` is 1
27. De default waarde voor `hoekpuntLinksBoven` wordt bepaald door de klasse `Punt` zelf
28. De default waarde voor `kleur` = “`wit`”
29. Pas constructor chaining toe
30. Een rechthoek is een vierhoek met vier rechte hoeken
31. De omtrek is tweemaal de `lengte` plus twee maal de `breedte` (of tweemaal de som van de `lengte` en de `breedte`)
32. De oppervlakte is de `lengte` maal de `breedte`
33. De methode `vertelOverGrootte()` is identiek aan die voor de klasse `Cirkel`
34. Vergeet niet de getters en de setters aan te maken.
    (De hele code vind je aan het einde van dit hoofdstuk.)



35. Test je klasse door de volgende code in de main-methode uit te voeren:

```
System.out.println(Rechthoek.geefDefinitie());
Rechthoek[] mijnRechthoekArray = new Rechthoek[3];
mijnRechthoekArray[0] = new Rechthoek(4, 3, new Punt(2, 5), "blauw");
mijnRechthoekArray[1] = new Rechthoek();
mijnRechthoekArray[2] = new Rechthoek(25, 10);

for (int arrayTeller = 0; arrayTeller < mijnRechthoekArray.length;
arrayTeller++) {
    System.out.println(mijnRechthoekArray[arrayTeller].geefOmtrek());
    System.out.println(mijnRechthoekArray[arrayTeller].geefOppervlakte());
    System.out.println(mijnRechthoekArray[arrayTeller].
     getHoekpuntLinksBoven().getxCoordinaat());
    System.out.println(mijnRechthoekArray[arrayTeller].
     getHoekpuntLinksBoven().getyCoordinaat());
    System.out.println(mijnRechthoekArray[arrayTeller].
     vertelOverGrootte());
}
```

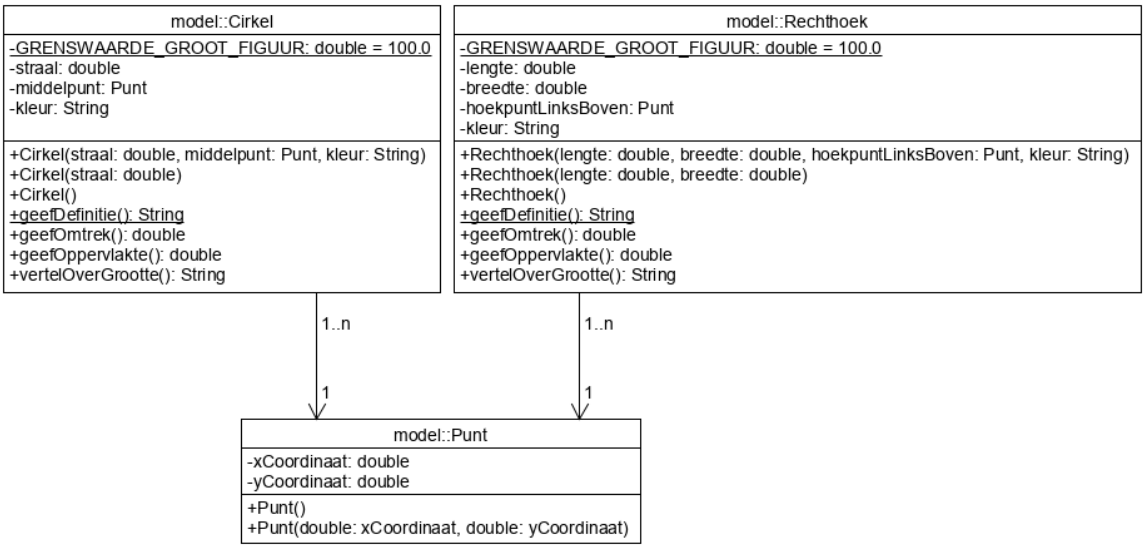
36. Je uitvoer ziet er zo uit:

```
Een rechthoek is een vierhoek met vier rechte hoeken.
14.0
12.0
2.0
5.0
Ik ben klein!!!
6.0
2.0
0.0
0.0
Ik ben klein!!!
70.0
250.0
0.0
0.0
Ik ben groot!!!
```

37. Probeer enkele andere waarden en bekijk de uitvoer.

3.6 Klassendiagram

Het totale klassendiagram ziet er nu zo uit:



3.7 De code voor de klasse Punt

```
package model;

public class Punt {
    private double xCoordinaat;
    private double yCoordinaat;

    public Punt() {
        this(0,0);
    }

    public Punt (double x, double y) {
        this.xCoordinaat = x;
        this.yCoordinaat = y;
    }

    public double getxCoordinaat() {
        return xCoordinaat;
    }

    public void setxCoordinaat(double xCoordinaat) {
        this.xCoordinaat = xCoordinaat;
    }

    public double getyCoordinaat() {
        return yCoordinaat;
    }

    public void setyCoordinaat(double yCoordinaat) {
        this.yCoordinaat = yCoordinaat;
    }
}
```

3.8 De code voor de klasse Rechthoek

```
package model;

public class Rechthoek {
private final static double GRENSWAARDE_GROOT_FIGUUR = 100.0;
    private double lengte;
    private double breedte;
    private Punt hoekpuntLinksBoven;
    private String kleur;

    public Rechthoek(double lengte, double breedte, Punt
hoekpuntLinksBoven,     String kleur) {
        this.lengte = lengte;
        this.breedte = breedte;
        this.hoekpuntLinksBoven = hoekpuntLinksBoven;
        this.kleur = kleur;
    }

    public Rechthoek(double lengte, double breedte) {
        this(lengte, breedte, new Punt(), "wit");
    }

    public Rechthoek() {
        this(2, 1);
    }

    public static String geefDefinitie() {
        return "Een rechthoek is een vierhoek met vier rechte hoeken.";
    }

    public double geefOmtrek() {
        return 2 * (lengte + breedte);
    }

    public double geefOppervlakte() {
        return lengte * breedte;
    }

    public String vertelOverGrootte() {
        if (geefOppervlakte() > GRENSWAARDE_GROOT_FIGUUR) {
            return "Ik ben groot!!!";
        } else {
            return "Ik ben klein!!!";
        }
    }

    public double getLengte() {
        return lengte;
    }

    public void setLengte(double lengte) {
        this.lengte = lengte;
    }

    public double getBreedte() {
        return breedte;
    }
```

```
    public void setBreedte(double breedte) {
        this.breedte = breedte;
    }

    public Punt getHoekpuntLinksBoven() {
        return hoekpuntLinksBoven;
    }
    public void setHoekpuntLinksBoven(Punt hoekpuntLinksBoven) {
        this.hoekpuntLinksBoven = hoekpuntLinksBoven;
    }
    public String getKleur() {
        return kleur;
    }
    public void setKleur(String kleur) {
        this.kleur = kleur;
    }
}
```

3.9 Opdrachten

1. Doe Opdracht **Klassen als attribuut-1 Bedrijf**.
2. Doe Opdracht **Klassen als attribuut-2 BSA monitor** (niveau: basis).
3. Mocht je nog zin en tijd hebben, doe dan Opdracht **Klassen als attribuut-3 Voetbalscores deel 2** (niveau: basis).

Bir alt sınıf (subklasse), başka bir üst sınıftan (superklasse) özellikleri (özellikler = değişkenler ve metodlar) miras alır (erft). Yani, alt sınıf, üst sınıfın özelliklerini ve davranışlarını kullanabilir.


```
class Dier {
   // Superklasse
}

class Hond extends Dier {
   // Subklasse
}
```

protected ve private farkı nedir?

protected: Bu değişken veya metod, alt sınıflar tarafından ve aynı package içindeki diğer sınıflar tarafından erişilebilir.

private: Bu durumda, sadece aynı sınıf içinde erişilebilir. Alt sınıflar erişmek istiyorsa, getter metodları kullanılmalıdır.

4 OVERERVING EN TOSTRING()

4.1 Samenvatting

Alt sınıf, üst sınıftaki özellik veya metodu çağırmak isterse super kelimesi kullanılır.

1. Overerving is wanneer een klasse (de *subklasse*) attributen en methodes erft van een andere klasse (de *superklasse*)
2. Overerving van een superklasse naar een subklasse gaat via het keyword `extends`
3. De visibility modifier `protected` kan gebruikt worden in de superklasse. De attributen zijn dan alleen zichtbaar voor de subklassen en voor andere klassen in dezelfde package. Het is ook mogelijk om de attributen `private` te houden. De toegang tot de attributen van een superklasse gaat dan altijd via een getter, ook in de subklassen.
4. Subklassen benaderen attributen en methodes van de superklasse via het keyword *super*
5. Subklassen kunnen methoden van de superklasse herdefiniëren (*override*)
6. De `toString()` methode wordt gebruikt om de relevante informatie van een object in de klasse te beschrijven

toString() metodu nedir?
Bu metot, bir nesnenin açıklayıcı bir metinle temsil edilmesini sağlar. Genellikle nesnenin bilgilerini içeren bir string döndürülür.

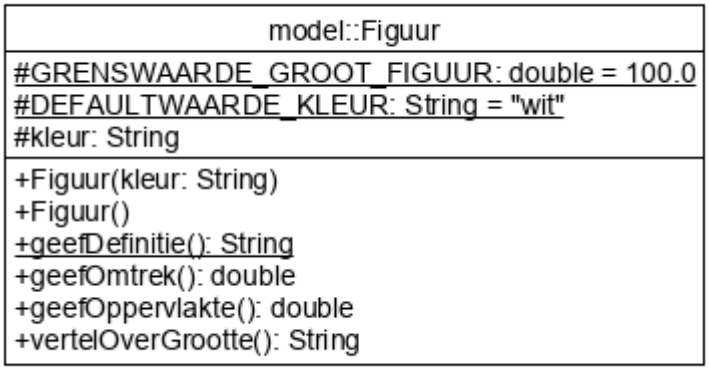
7. De methode `String.format()` kan worden gebruikt om teksten te formatteren. De formattering gaat op dezelfde manier als bij de `System.out.printf()` methode

Metinleri biçimlendirmek için kullanılır. Aynı biçimde System.out.printf() de kullanılabilir.

4.2 Overerving (inheritance): de klasse Figuur

1. Bekijk: what is inheritance
2. Ga verder met: different types of inheritance
3. Bekijk ook nog: using inheritance to reduce code duplication
4. Lees Liang 11.2 en 11.3
5. Indien nodig kun je het project ***OOP Meetkunde Klassen als attribuut*** van ***DLO*** halen. Dit bevat de klassen zoals je die in het vorige onderdeel hebt afgesloten
6. Maak een nieuwe klasse `Figuur` volgens het volgende klassendiagram. Gebruik de volgende extra informatie:
7. De defaultwaarde voor `kleur` is “wit” en aangezien de defaultwaarde op meer dan één plaats wordt gebruikt, wordt er een constante van gemaakt
8. Gebruik *constructor chaining*
9. Een figuur is een verzameling punten
10. De omtrek is onbekend, dus geef de waarde 0 terug
11. De oppervlakte is onbekend, dus geef de waarde 0 terug
12. De methode `vertelOverGrootte()` is identiek aan die voor de klasse `Cirkel` en de klasse `Rechthoek`
13. Maak de getter en setter voor `kleur`.
    (De hele code vind je aan het einde van dit hoofdstuk.)

Let op: de methode *geefDefinitie()* is statisch. Deze is daarom niet te overriden in de subklassen. Je kunt een statische methode wel herdefiniëren binnen de subklassen.



+ betekent public
- betekent private (komt hier niet voor)
betekent protected (beschikbaar voor de klasse en alle subklassen)
ALL_CAPS betekent constante
Onderstreept betekent static

↑ De drie attributen zijn `protected` en dus beschikbaar voor de subklassen;
↑ De methodes `geefOmtrek()` en `geefOppervlakte()` zijn `public`. Daar wordt echter in de subklassen nog verder aan gesleuteld;
↑ De methodes `Figuur(kleur: String)`, `Figuur()` en `geefDefinitie()` zijn `public`, omdat die iets specifiek over de klasse `Figuur` zeggen;
↑ De methode `vertelOverGrootte()` is `public`. De methode verdwijnt namelijk uit de subklassen.

4.3 Overerving (inheritance): de subklasse Cirkel

1. Wijzig de regel met de definitie van de klasse `Cirkel` als volgt:

```
public class Cirkel extends Figuur{
```

↑ De code `extends Figuur` zegt dat de klasse `Cirkel` nu een subklasse van de klasse `Figuur` is. Alle attributen en methoden, die `public` of `protected` zijn binnen de klasse `Figuur` zijn daarmee ook beschikbaar voor de klasse `Cirkel`

2. Verwijder de regel
```
private final static double GRENSWAARDE_GROOT_FIGUUR = 100.0;
```
de constante wordt nu overgeërfd van de klasse `Figuur`

3. Verwijder de regel
```
private String kleur;
```
het attribuut wordt ook overgeërfd van de klasse `Figuur`

4. Pas de all-args constructor als volg aan:

```
public Cirkel(double straal, Punt middelpunt, String kleur) {
    super(kleur);
    this.straal = straal;
    this.middelpunt = middelpunt;
}
```

↑ `kleur` is een attribuut van de klasse `Figuur`
↑ `super` verwijst naar de bovenliggende (super-) klasse.
↑ `super(kleur)` roept de all-args constructor aan van de klasse `Figuur`.

5. Pas de 'straal'-constructor als volg aan:

```
public Cirkel(double straal) {
    this(straal, new Punt(), DEFAULTWAARDE_KLEUR);
}
```

↑ De klasse `Figuur` bepaalt de defaultwaarde voor `kleur`. Deze wordt gebruikt in de constructor chaining.

6. Voeg het Javadoc keyword `@Override` toe aan de methodes `geefOmtrek()` en `geefOppervlakte()`, zoals hieronder.

```
@Override
public double geefOmtrek() {
    return 2 * Math.PI * straal;
}

@Override
public double geefOppervlakte() {
    return Math.PI * straal * straal;
}
```

↑ `@Override` zegt dat de oorspronkelijke methodes binnen de klasse `Figuur` niet gebruikt worden binnen de subklasse `Cirkel`. In plaats daarvan worden de eigen methodes gebruikt. Merk op dat het keyword niet verplicht is, maar bedoeld is om de leesbaarheid te verhogen. Ook zal IntelliJ je dan beter kunnen helpen bij een eventuele vergissing.

7. Verwijder de hele methode `vertelOverGrootte()` uit de klasse `Cirkel`.
   De methode binnen de klasse `Figuur` wordt nu gebruikt en het is niet nodig om een eigen methode te gebruiken

8. Verwijder de hele methodes `getKleur()` en `setKleur()`. Ook deze worden overgeërfd van de klasse `Figuur`

9. Ga in de main-methode naar de regel
   `System.out.println(mijnCirkelArray[arrayTeller].getMiddelpunt().getyCoordinaat());`
   en voeg een lege regel toe

10. Type `System.out.println(mijnCirkelArray[arrayTeller]` en dan een punt. Je ziet het volgende:

```
for (int arrayTeller = 0; arrayTeller < 3; arrayTeller++) {
    System.out.println(mijnCirkelArray[arrayTeller].geefOmtrek());
    System.out.println(mijnCirkelArray[arrayTeller].geefOppervlakte());
    System.out.println(mijnCirkelArray[arrayTeller].getMiddelpunt().getxCoordinaat());
    System.out.println(mijnCirkelArray[arrayTeller].getMiddelpunt().getyCoordinaat());
    System.out.println(mijnCirkelArray[arrayTeller].);
}

/*System.out.println(Rechthoek.geefDefinitie())
Rechthoek[] mijnRechthoekArray = new Rechthoek
mijnRechthoekArray[0] = new Rechthoek(4, 3, ne
mijnRechthoekArray[1] = new Rechthoek();
mijnRechthoekArray[2] = new Rechthoek(25, 10);
for (int arrayTeller = 0; arrayTeller < 3; arra
    System.out.println(mijnRechthoekArray[array
    System.out.println(mijnRechthoekArray[array
    System.out.println(mijnRechthoekArray[array
    System.out.println(mijnRechthoekArray[array
    System.out.println(mijnRechthoekArray[array
}*/
```

```
vertelOverGrootte()                    String
getMiddelpunt()                        Punt
geefOmtrek()                           double
geefOppervlakte()                      double
getStraal()                            double
getKleur()                             String
equals(Object obj)                     boolean
hashCode()                             int
toString()                             String
getClass()          Class<? extends Cirkel>
setMiddelpunt(Punt middelpunt)         void
Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards
```

↑ Je ziet dat de methode `vertelOverGrootte()` beschikbaar is in de klasse `Cirkel`. Ook `getKleur()` en `setKleur()` zijn beschikbaar.

11. Selecteer de methode `vertelOverGrootte()`
12. Plaats de hele code over de rechthoek in comments
13. Run je programma. De uitvoer ziet er als volgt uit:

```
18.84955592153876
28.274333882308138
1.0
4.0
Ik ben klein!!!
6.283185307179586
3.141592653589793
0.0
0.0
Ik ben klein!!!
37.69911184307752
113.09733552923255
0.0
0.0
Ik ben groot!!!
```

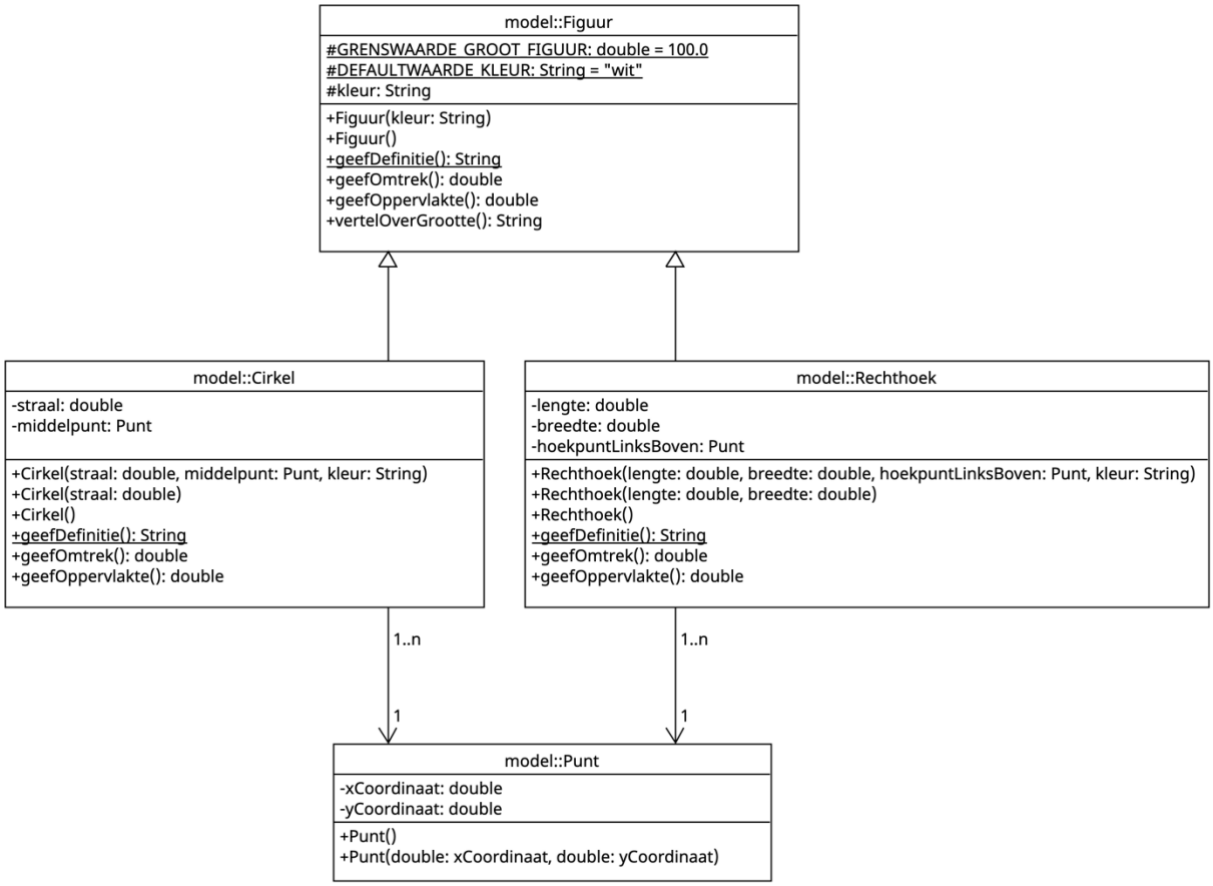
↑ De methode `vertelOverGrootte()` in de klasse `Figuur` roept de methode `geefOppervlakte()` aan. Doordat deze 'overridden' wordt in de klasse `Cirkel`, wordt niet de methode `geefOppervlakte()` van de klasse `Figuur` gebruikt maar die van de klasse `Cirkel`.

4.4 Overerving (inheritance): de subklasse Rechthoek

1. Pas de klasse `Rechthoek` aan, zodat deze óók een subklasse van de klasse `Figuur` wordt:
2. Voeg `extends Figuur` toe
3. Verwijder de attributen `GRENSWAARDE_GROOT_FIGUUR` en `kleur`
4. Wijzig de all-args constructor:
   roep `super(kleur)` aan
   verwijder `this.kleur = kleur`
5. Pas de constructor aan, die de defaultwaarde van `kleur` meegeeft
6. Override de methodes `geefOmtrek()` en `geefOppervlakte()`
7. Verwijder de methode `vertelOverGrootte()`
8. Verwijder de getter en setter van `kleur`
9. Uncomment de code over de rechthoek en run je programma.
   Je uitvoer is hetzelfde als in ***3.5.13***

4.5 Klassendiagram

Subklassen worden weergegeven met een gesloten pijl. Het totale klassendiagram ziet er zo uit:



+	betekent	public
-	betekent	private
#	betekent	protected
ALL_CAPS	betekent	constante
Onderstreept	betekent	static
Open pijl	betekent	unidirectioneel
Gesloten pijl	betekent	subklasse

4.6 De methode toString()

1. Lees Liang 11.6
2. Voeg aan de main-methode de volgende regel code toe.

```
Punt testPunt = new Punt(2.0, 3.5);
System.out.println(testPunt);
```

↑ Met deze code onderzoeken we wat het resultaat is van het printen van een variabele van het type Punt.

3. Je ziet een resultaat als hieronder:

```
model.Punt@36baf30c
```

↑ Het printen van een variabele van het type `Punt` resulteert niet in concrete informatie over het punt zelf. Het is algemene informatie, nl. de naam van de package waarin de klasse `Punt` staat, gevolgd door het vermelden van de klassenaam en tenslotte het geheugen adres van de variabele.

↑ Dat er iets geprint wordt, komt doordat de klasse `Punt` een methode erft van een superklasse genaamd `Object`. Dit is de methode `toString()`.

↑ Alle klassen in Java zijn automatisch subklasse van de superklasse Object en erven alle methoden van deze klasse. Voorbeelden van deze methoden zijn `equals()`, `hashcode()`, `toString()`.

4. Vervang de for-loop voor de array van cirkels in de main-methode als volgt:

```
for (int arrayTeller = 0; arrayTeller < mijnCirkelArray.length;
arrayTeller++) {
   System.out.println(mijnCirkelArray[arrayTeller]);
   System.out.println(mijnCirkelArray[arrayTeller].vertelOverGrootte());
}
```

5. Je ziet een resultaat als hieronder:

```
model.Cirkel@7a81197d
Ik ben klein!!!
model.Cirkel@5ca881b5
Ik ben klein!!!
model.Cirkel@24d46ca6
Ik ben groot!!!
```

↑ Ook hier resulteert het printen van een variabele van het type Cirkel in dezelfde generieke informatie, nl. de packagenaam, de klassenaam en het geheugenadres.

6. We gaan de methode `toString()` in de klassen *overriden*, zodat er specifieke informatie geprint gaat worden. Voeg aan de klasse `Punt` een methode `toString()` toe, zoals hieronder:

```
@Override
public String toString() {
    return String.format("(%.2f, %.2f)", xCoordinaat, yCoordinaat);
}
```

↑ De methode `toString()` moet een `String` als *retourwaarde* hebben.

↑ De `String.format()` methode wordt gebruikt om strings te formatteren. De syntax is dezelfde als voor de `System.out.printf()`-methode die in ***Programming*** is geïntroduceerd. Hierboven worden de x- en y-coördinaten van het middelpunt met twee decimalen getoond.

↑ `@Override` geeft aan dat we de `toString()` methode van de *superklasse* `Object` *overriden*.

7. Run de main-methode nogmaals en kijk naar het resultaat van `System.out.println(testPunt)`. Je ziet het volgende:

```
(2.00, 3.50)
```

↑ Er staat nu specifieke informatie van de variabele `testPunt`. De waarden van de x- en y-coödinaat worden getoond in de vorm van een String van het formaat in de `toString()`-methode.

8. Voeg aan de klasse `Figuur` een methode `toString()` toe, zoals hieronder:

```
@Override
public String toString() {
    return "Kleur: " + kleur + "\nOmtrek: " + geefOmtrek() +
        "\nOppervlakte: " + geefOppervlakte();
}
```

↑ In de klasse `Figuur` definieren we een `toString()`-methode die de gemeenschappelijke informatie in de *superklasse* teruggeeft in de vorm van een String.

↑ Deze methode gaan we hergebruiken in de *subklassen*.

9. Voeg aan de klasse `Cirkel` een methode `toString()` toe, zoals hieronder:

```
@Override
public String toString() {
    return super.toString() + "\nStraal: " + straal + "\nMiddelpunt: " +
        middelpunt.toString();
}
```

↑ `super.toString()` roept de methode `toString()` van de superklasse `Figuur` aan. Deze geeft een `String` zoals die in stap 8 is bepaald. Daar worden de eigen attributen van de klasse `Cirkel` aan toegevoegd. Tot slot wordt de methode `toString()` van het `Punt` `middelpunt` aangeroepen.

10. Voeg aan de klasse `Rechthoek` een methode `toString()` toe, zoals hieronder:

```
@Override
public String toString() {
    return super.toString() + "\nLengte: " + lengte + "\nBreedte: " +
        breedte + "\nHoekpunt: " + hoekpuntLinksBoven.toString();
}
```

11. Vervang nu de for-loop van de array van rechthoeken in je main-methode als volgt:

```
for (int arrayTeller = 0; arrayTeller < mijnRechthoekArray.length;
arrayTeller++) {
   System.out.println(mijnRechthoekArray[arrayTeller]);
   System.out.println(mijnRechthoekArray[arrayTeller].vertelOverGrootte());
}
```

12. Run je programma.

4.7 De code voor de klasse Figuur

```
package model;

public class Figuur {
    protected final static double GRENSWAARDE_GROOT_FIGUUR = 100.0;
    protected final static String DEFAULTWAARDE_KLEUR = "wit";
    protected String kleur;

    public Figuur(String kleur) {
        this.kleur = kleur;
    }

    public Figuur() {
        this(DEFAULTWAARDE_KLEUR);
    }

    public static String geefDefinitie() {
        return "Een figuur is een verzameling punten.";
    }

    public double geefOmtrek() {
        return 0;
    }

    public double geefOppervlakte() {
        return 0;
    }

    public String vertelOverGrootte() {
        if (geefOppervlakte() > GRENSWAARDE_GROOT_FIGUUR) {
            return "Ik ben groot!!!";
        } else {
            return "Ik ben klein!!!";
        }
    }

    public String getKleur() {
        return kleur;
    }

    public void setKleur(String kleur) {
        this.kleur = kleur;
    }
}
```

4.8 Opdrachten

1. Doe Opdracht **Overerving-1 Bedrijf**.
2. Doe Opdracht **Overerving-2 Verkeersboetes** (niveau: basis).
3. Mocht je nog zin en tijd hebben, doe dan Opdracht **Overerving-3 Leseenheid** (niveau: basis).

"Bir üst sınıfın değişkeni, alt sınıftan bir nesneye referans verebilir."

Açıklama:

Nesne yönelimli programlamada polimorfizm, aynı türdeki bir referansın (örneğin, bir üst sınıf tipi) farklı alt sınıf nesnelerine işaret edebilmesi anlamına gelir.

Yani, bir superclass (üst sınıf) türündeki bir değişken, onun subclass (alt sınıf) türünden oluşturulan bir nesneyi tutabilir.

Dier dier = new Hond(); // Dier: üst sınıf, Hond: alt sınıf

5 POLYMORFISME EN ABSTRACTE KLASSEN

5.1 Samenvatting

1. Variabelen van een *superklasse* kunnen refereren naar een instantie (een object) van een *subklasse*. Dit heet *polymorfisme*.
2. *Abstracte klassen* en *abstracte methodes* zijn bedoeld om ze te declareren en ze op een andere plaats (bijvoorbeeld in een subklasse) te specificeren.
3. De `LocalDate` klasse wordt gebruikt om bewerkingen met data uit te voeren.

5.2 Polymorfisme

Abstracte klasse (Soyut sınıf): Kendisinden doğrudan nesne oluşturulamaz.

Abstracte methode (Soyut metot): Sadece ismi verilir, içeriği (gövdesi) alt sınıflarda yazılır.

```
abstract class Dier {
    abstract void maakGeluid(); // sadece tanım var, gövde yok
}
class Hond extends Dier {
    void maakGeluid() {
        System.out.println("Blaft"); // burada detay verildi
    }
}
```

1. Bekijk: what is polymorphism
2. Lees Liang 11.7 en 11.8
3. Indien nodig kun je het project ***OOP Meetkunde Inheritance*** van ***DLO*** halen. Dit bevat de klassen zoals je die in het vorige onderdeel hebt afgesloten.
4. Maak een nieuwe methode aan in je launcher **onder** je main-methode, zoals hieronder:

```
public static void toonInformatie(Figuur figuur) {
    System.out.println(figuur);
}
```

↑ De parameter van deze methode is van de klasse `Figuur`.

Polymorfisme: Üst sınıf değişkeni, alt sınıf nesnesine referans verebilir.

Abstracte klassen/methoden: Ortak yapı tanımlanır, detaylar alt sınıflarda yazılır.

LocalDate: Tarihlerle çalışmak için kullanılan Java sınıfıdır.

5. Verander je main-methode als volgt:

```
public static void main(String[] args) {
    Cirkel mijnCirkel = new Cirkel(3, new Punt(2, 5), "groen");
    toonInformatie(mijnCirkel);
    System.out.println();
    Rechthoek mijnRechthoek = new Rechthoek(3, 4, new Punt(-2, 6),
        "blauw");
    toonInformatie(mijnRechthoek);
}
```

↑ De methode *`toonInformatie()`* wordt aangeroepen met variabelen van de klassen `Cirkel` en `Rechthoek`. **Een methode die een parameter van een superklasse verwacht, kan altijd worden aangeroepen met een variabele van een subklasse. Dit noemen we** *polymorfisme*.

Bu metodun parametresi Figuur tipinde.

Figuur, hem Cirkel hem de Rechthoek sınıflarının superclass'ı (üst sınıfı) olmalı.

System.out.println(figuur); satırı sayesinde figuur nesnesinin bilgileri yazdırılır.

Bu satırın düzgün çalışabilmesi için Figuur sınıfında toString() metodunun tanımlanmış olması gerekir.

Cirkel ve Rechthoek sınıfları Figuur sınıfından türemiş (yani extends Figuur kullanılmış).

toonInformatie() metodu sadece Figuur tipinde parametre bekliyor. Ama siz bu metoda Cirkel ve Rechthoek nesnesi veriyorsunuz.

Bu Java'da geçerli çünkü Cirkel ve Rechthoek, Figuur sınıfının alt sınıfları (subclass).

İşte bu yüzden bu örnek polymorfisme'in bir uygulamasıdır.

toonInformatie(Figuur figuur) metodu tek bir kere yazılır ama birçok farklı alt sınıf nesnesi ile çalışabilir.

Bu yapı sayesinde kod daha esnek, tekrar kullanılabilir ve bakımı kolay olur.

6. Run je programma, je uitvoer ziet er zo uit:

```
Kleur: groen
Omtrek: 18.84955592153876
Oppervlakte: 28.274333882308138
Straal: 3.0
Middelpunt: (2,00, 5,00)

Kleur: blauw
Omtrek: 14.0
Oppervlakte: 12.0
Lengte: 3.0
Breedte: 4.0
Hoekpunt: (-2,00, 6,00)
```

İlk kısımda, Cirkel sınıfının toString() metodu çalışıyor.

İkinci kısımda, Rechthoek sınıfının toString() metodu çalışıyor.

Bu durum, hangi nesne türünün hangi metodu çağıracağını derleme (compile) zamanında değil, çalışma (runtime) zamanında belirlenmesini ifade eder. Bu olaya Java'da:

✅ Dynamic Binding (Dinamik Bağlama) denir.

↑ De eerste helft van de uitvoer betreft de `toString()` methode van de klasse `Cirkel`. De tweede helft van de uitvoer betreft de `toString()methode` van de klasse `Rechthoek`. **De compiler bepaalt zelf welke `toString()`-methode gebruikt moet worden. Dit noemen we** *dynamic binding*.

Dit principe kan ook toegepast worden in arrays van een superklasse. Deze kan dan ook objecten van een subklasse opnemen.

7. Voeg de volgende regels toe in je main-methode :

```
Figuur[] figuren = new Figuur[3];
figuren[0] = mijnCirkel;
figuren[1] = mijnRechthoek;
figuren[2] = new Cirkel(10, new Punt(-1,-3), "karmozijn");
for (int figuurTeller = 0; figuurTeller < 3; figuurTeller++) {
    System.out.println(figuren[figuurTeller]);
    System.out.println();
}
```

↑ De array bevat objecten van de klasse `Figuur` en kan gevuld worden met objecten van een van de subklassen. Ook nu wordt de juiste `toString()`-methode aangeroepen.

Figuur[] tipi bir superklasse dizisi.

Bu diziye hem Cirkel hem Rechthoek gibi subklasse nesneleri konulabiliyor.

System.out.println(figuren[i]) satırı çalıştığında:

Java, her elemanın gerçek tipi neyse, onun toString() metodunu çağırıyor.

Bu yine polymorfisme ve dynamic binding örneğidir.

8. Run je programma, je uitvoer ziet er zo uit:

```
Kleur: groen
Omtrek: 18.84955592153876
Oppervlakte: 28.274333882308138
Straal: 3.0
Middelpunt: (2,00, 5,00)

Kleur: blauw
Omtrek: 14.0
Oppervlakte: 12.0
Lengte: 3.0
Breedte: 4.0
Hoekpunt: (-2,00, 6,00)

Kleur: karmozijn
Omtrek: 62.83185307179586
Oppervlakte: 314.1592653589793
Straal: 10.0
Middelpunt: (-1,00, -3,00)
```

5.3 Abstracte klassen

Figuur sınıfından doğrudan nesne oluşturulamaz.

Ancak Figuur'ü kalıtan (extends eden) sınıflar oluşturabiliriz (örneğin Cirkel ve Rechthoek gibi).

1. Lees Liang 13.2

2. Verander de declaratie van je klasse `Figuur` als volgt:

```
public abstract class Figuur {
```

3. Verander de methodes `geefOmtrek()` en `geefOppervlakte()` als volgt:

```
public abstract double geefOmtrek();
public abstract double geefOppervlakte();
```

↑ Merk op dat beide methodes geen inhoud (body) hebben. Ze worden hier alleen gedeclareerd.

↑ De implementatie van de methodes gebeurt in de subklassen.

Van een abstracte klasse kun je alleen nieuwe subklassen definiëren als je in die subklassen de abstracte methodes herdefinieert (`override`).

4. Run je programma. Aangezien je niets veranderd hebt aan de inhoud zal de uitvoer gelijk zijn aan dat van de vorige oefening.

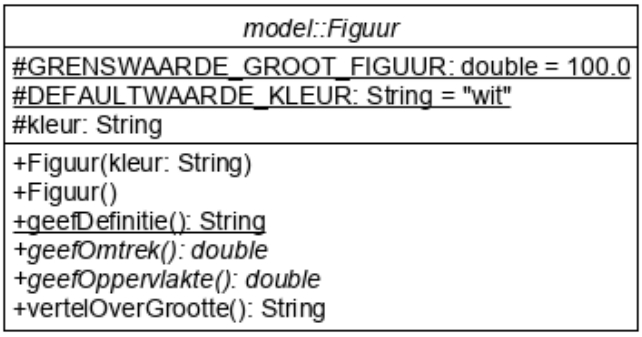
Bu metodların gövdesi (body) yok çünkü bu sınıf yalnızca tanım yapıyor, uygulamayı alt sınıflara bırakıyor.

Cirkel ve Rechthoek sınıfları, bu metodları kendi içlerinde override etmek zorunda.

Eğer override etmezlerse hata alırsın.

5.4 Klassendiagram

Het klassendiagram van de klasse `Figuur` ziet er zo uit:



+	betekent	public
-	betekent	private (komt hier niet voor)
#	betekent	protected
ALL_CAPS	betekent	constante
Onderstreept	betekent	static
Schuin gedrukt	betekent	abstract (hier zowel de klasse als de twee methodes)

5.5 De LocalDate-klasse

Gedurende de komende lessen zal gebruik gemaakt worden van een datum. Je krijgt hier alvast een paar voorbeelden van de *LocalDate-klasse*, die gebruikt wordt om een datum in op te slaan. Je kunt de volgende code in een apart project testen. Op de ***DLO*** staat de code in een project ***LocalDate***.

```
public static void main(String[] args) {
    // Geeft de dag van vandaag
    LocalDate vandaag = LocalDate.now();
    System.out.println(vandaag);

     // Een manier om 1 januari 2021 als LocalDate in te voeren.
    LocalDate beginVan2021 = LocalDate.parse("2021-01-01");
    System.out.println(beginVan2021);
    // Een andere manier om een datum in te voeren
    LocalDate koningsDag2021 = LocalDate.of(2021, 04, 27);
    System.out.println(koningsDag2021);

    // Een paar voorbeelden van methodes binnen de LocalDate klasse
    int maandVanKoningsDag2021 = koningsDag2021.getMonthValue();
    System.out.println(maandVanKoningsDag2021);

    String dagVanDeWeek = koningsDag2021.getDayOfWeek().toString();
    System.out.println(dagVanDeWeek);

    // Twaalf dagen na de Koningsdag van 2021
    System.out.println(koningsDag2021.plusDays(12));
    // Vijf maanden na de Koningsdag van 2021
    System.out.println(koningsDag2021.plusMonths(5));
    // Drie jaar na vandaag
    System.out.println(vandaag.plusYears(3));
    // Komt de Koningsdag van 2021 na vandaag?
    if (koningsDag2021.isAfter(vandaag)) {
     System.out.println("Vandaag is het voor 27 april 2021");
     } else if (koningsDag2021.equals(vandaag)) {
     System.out.println("Vandaag is het 27 april 2021");
     } else {
     System.out.println("Vandaag is het na 27 april 2021");
     }
     if (beginVan2021.isBefore(vandaag.minusMonths(3))) {
     System.out.println("Nieuwjaarsdag 2021 is meer dan drie maanden
geleden");
     }
 }
```

↑ De LocalDate klasse heeft statisch methoden, zoals *now()*, *parse()* en *of()*, om objecten van het type LocalDate mee te instantiëren.

↑ De LocalDate klasse heeft allerlei methoden om informatie van een datum te krijgen, zoals de maand of de dag van de week.

↑ De LocalDate klasse heeft methoden om bewerkingen op datums te doen, zoals dagen, maanden of jaren erbij optellen.

↑ De LocalDate klasse heeft methoden om datums met elkaar te vergelijken.

5.6 Opdrachten

1. Doe Opdracht **Polymorfisme-1 Bedrijf**.
2. Doe Opdracht **Polymorfisme-2 Vervoermiddelen** (niveau: basis).

ArrayList, Java'daki dinamik bir liste yapısıdır.
Normal Array sabit boyutludur (String[] array = new String[10] gibi), ama ArrayList gerektiğinde büyür veya küçülür.
ArrayList<String> lijst = new ArrayList<>();
lijst.add("Hallo");
lijst.add("Wereld");
ArrayList<String> ifadesindeki <String>, listenin sadece String türündeki nesneleri tutacağını belirtir.
Buna generics denir.

6 WRAPPER CLASSES, ARRAYLIST, INSTANCEOF

6.1 Samenvatting

1. *Wrapper classes* zijn klassen, die basistypes (zoals `int` en `double`) inpakken. Ze zorgen ervoor dat primitieve types zich kunnen gedragen als objecten.
2. Van `int` naar `Integer` heet *boxing*; van `Integer` naar `int` heet *unboxing*.
3. *ArrayList* is flexibeler dan *Array*, bijvoorbeeld omdat de lengte indien nodig dynamisch wordt aangepast bij het toevoegen en verwijderen van objecten
4. Een ArrayList geef je een datatype van de objecten mee door in de declaratie tussen zogeheten *vishaken* <> dit datatype te zetten, dus bijvoorbeeld `ArrayList<String>`
5. Het is gebruikelijk en handig om door een ArrayList te lopen met behulp van de zogeheten *for-each loop*
6. *Typecasting* en `instanceof` zijn twee hulpmiddelen om met de juiste klasse bezig te kunnen zijn.
7. `instanceof` is een boolean-operator die wordt gebruikt om de klasse van een bepaald object te controleren.
8. *Typecasting* wordt toegepast om vervolgens de specifieke attributen en methodes van die subklasse te kunnen benaderen.

6.2 Wrapper classes

1. Lees Liang 10.7 en 10.8
2. Voeg de volgende code toe onderaan je main-methode:

```
int integer1 = 3;
Integer integer2 = Integer.valueOf(3);
Integer integer3 = 3;
int integer4 = Integer.valueOf(3);
System.out.println(integer1 + "\n" + integer2.toString() + "\n" + integer3
+ "\n" + Integer.valueOf(integer4).toString());
```

↑ De eerste regel kent de waarde 3 toe aan de variabele van het primitieve type.
↑ De tweede regel dwingt af dat een object van het type `Integer` wordt aangemaakt. De constructor `new Integer(int)` wordt niet meer gebruikt en in plaats daarvan wordt de statische methode `valueOf(int)` gebruikt. Het maken van een `Integer` object van een `int` heet *boxing*.
↑ *IntelliJ* meldt dat in de tweede regel onnodige *boxing* plaatsvindt.
↑ In de derde regel wordt dit ondervangen door direct de waarde van de `int` 3 aan het `Integer` object toe te kennen, dit heet *autoboxing*.
↑ De vierde regel maakt een object van het type `Integer` aan en kent die als waarde toe aan de variabele van het primitieve type. Door *unboxing* kan dit toch toegekend worden en omdat dit weer automatisch gebeurt heet dit *auto-unboxing*. Ook hier meldt *IntelliJ* overigens dat er sprake is van onnodige *boxing*.

- ↑ De vijfde regel toont de waarden van `integer1` en `integer3` op het uitvoerscherm door automatisch de nodige acties te ondernemen. Bij het afdrukken van `integer2` en `integer4` wordt expliciet gemaakt wat de compiler automatisch doet bij `integer1` en `integer3`.
- ↑ Merk op dat `integer1` en `integer4` *geen* `toString()` methode kennen (het zijn primitieve datatypes) en `integer2` en `integer3` *wel* een `toString()` methode kennen (het zijn objecten van de klasse `Integer`).

3. Run je programma en merk op dat er vier keer een 3 wordt getoond.

6.3 ArrayList

1. Lees Liang 11.11
2. Indien nodig kun je het project ***OOP Meetkunde Polymorfisme*** van DLO halen. Dit bevat de klassen zoals je die in het vorige onderdeel hebt afgesloten.
3. Voeg de volgende code toe aan je main-methode:

```
ArrayList<Cirkel> mijnCirkels = new ArrayList<>();
mijnCirkels.add(new Cirkel(3, new Punt(1, 4), "groen"));
mijnCirkels.add(new Cirkel());
mijnCirkels.add(new Cirkel(6));

System.out.printf("Er zijn nu %d cirkels\n", mijnCirkels.size());
System.out.println("De straal van mijn tweede cirkel is: " +
    mijnCirkels.get(1).getStraal());
mijnCirkels.remove(2);
System.out.printf("Er zijn nu %d cirkels\n", mijnCirkels.size());

toonInformatie(mijnCirkels.get(1));
```

- ↑ `ArrayList<Cirkel> mijnCirkels = new ArrayList<>();` declareert een ArrayList met objecten van de klasse `Cirkel` met de naam `mijnCirkels`.
- ↑ De volgende drie regels vullen de ArrayList met drie objecten.
- ↑ `mijnCirkels.size()` geeft het aantal objecten in de ArrayList, dat zijn er nu drie.
- ↑ `mijnCirkels.get(1)` geeft het tweede object, dat is dus een object van de klasse Cirkel.
- ↑ `mijnCirkels.get(1).getStraal()` geeft de straal van dat object.
- ↑ `mijnCirkels.remove(2)` verwijdert het derde(!) object.
- ↑ `mijnCirkels.size()` geeft het aantal objecten in de ArrayList, dat zijn er nog maar twee.

4. Run je programma. Je uitvoer ziet er zo uit:

```
Er zijn nu 3 cirkels
De straal van mijn tweede cirkel is: 1.0
Er zijn nu 2 cirkels
Kleur: wit
Omtrek: 6.283185307179586
Oppervlakte: 3.141592653589793
Straal: 1.0
Middelpunt: (0,00, 0,00)
```

6.4 for-loop voor een ArrayList: for-each loop

1. Lees Liang 7.2.7
2. Verwijder de regel `toonInformatie(mijnCirkels.get(1))` in je main-methode
3. Voeg de volgende code toe aan je main-methode:

```
for (int listTeller = 0; listTeller < mijnCirkels.size(); listTeller++) {
    toonInformatie(mijnCirkels.get(listTeller));
    System.out.println();
}
```

↑ Begin je for-loop altijd met 0.
↑ De teller mag nooit gelijk zijn aan de `size()`. Bedenk dat `size()` == 2 wil zeggen dat de waarden 0 en 1 beschikbaar zijn! Vandaar dus '<'.

4. Run je programma. Je krijgt de informatie over de aanwezige twee cirkels te zien.
5. Verwijder de bovenstaande code en voeg de volgende code toe aan je main-methode:

```
for (Cirkel cirkel : mijnCirkels) {
    toonInformatie(cirkel);
    System.out.println();
}
```

↑ `cirkel` is de naam van de reference variabele van het type `Cirkel`, die langs alle objecten in de ArrayList `mijnCirkels` loopt
↑ Je kunt de zogeheten for-each loop als volgt lezen: Voor elke `cirkel` van het type `Cirkel` in de lijst `mijnCirkels` doe ….
↑ Let wel: je declareert net als in de gewone for-loop een hulpvariabele (hier dus `cirkel`) waarmee je stapsgewijs alle elementen van een array of lijst langsgaat. Bij een gewone for-loop is dat een *teller*, die je zelf moet ophogen. Bij de *for-each loop* ga je direct langs alle objecten in de lijst, het ophogen of 'verspringen' naar het volgende element wordt vanzelf gedaan.

6. Run je programma. Je uitvoer is exact hetzelfde als hierboven.

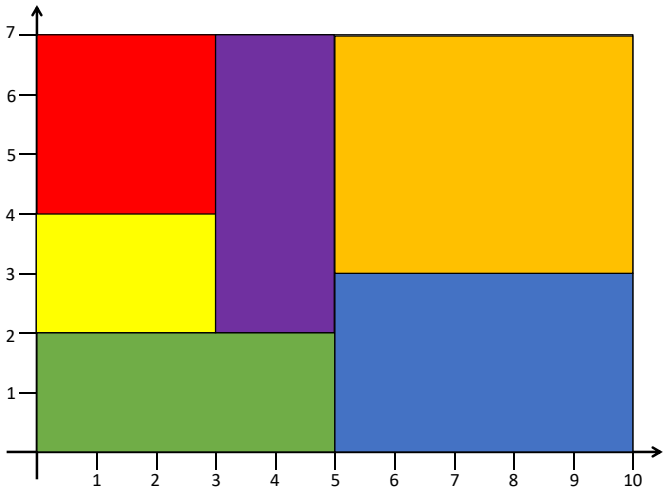
6.5 De klasse Canvas: attribuut van het type ArrayList

1. Maak een nieuwe klasse Canvas aan volgens de code hieronder:

```
public class Canvas {
    private double lengte;
    private double breedte;
    private ArrayList<Figuur> mijnFiguren;

    public Canvas(double lengte, double breedte) {
        this.lengte = lengte;
        this.breedte = breedte;
        mijnFiguren = new ArrayList<>();
    }

    public void voegFiguurToe(Figuur figuur) {
        mijnFiguren.add(figuur);
    }

    @Override
    public String toString() {
        String canvasString = "";
        for (Figuur figuur : mijnFiguren) {
            canvasString += figuur.toString() + "\n\n";
        }
        return canvasString;
    }
}
```

↑ In deze klasse wordt een ArrayList gemaakt van de abstracte klasse `Figuur`.
↑ De methode `voegFiguurToe()` kent ook alleen de abstracte klasse `Figuur`.
↑ Voorgaande betekent dat je alle objecten van subklasse typen in de ArrayList kunt plaatsen, dus cirkels en rechthoeken. Dit is wederom polymorfisme.
↑ De methode `voegFiguurToe()` geeft een vorm van encapsulatie. Alleen via deze methode heb je toegang tot de ArrayList `mijnFiguren`.
↑ De methode `toString()` roept de `toString()` methode aan van de objecten in de arraylist.

Het canvas wordt nu gevuld volgens de onderstaande afbeelding:



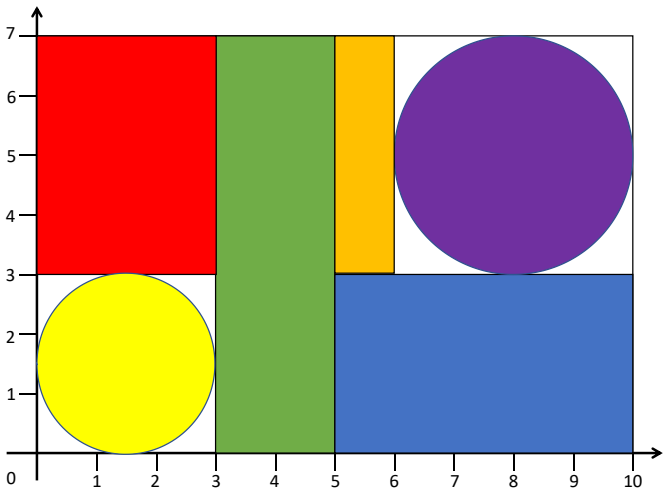
2. Voer de volgende code in je main-methode:

```
Canvas canvas = new Canvas(10, 7);
canvas.voegFiguurToe(new Rechthoek(3, 3, new Punt(0, 7), "rood"));
canvas.voegFiguurToe(new Rechthoek(3, 2, new Punt(0, 4), "geel"));
canvas.voegFiguurToe(new Rechthoek(5, 2, new Punt(0, 2), "groen"));
canvas.voegFiguurToe(new Rechthoek(5, 2, new Punt(3, 7), "paars"));
canvas.voegFiguurToe(new Rechthoek(5, 4, new Punt(5, 7), "oranje"));
canvas.voegFiguurToe(new Rechthoek(5, 3, new Punt(5, 3), "blauw"));
System.out.println(canvas);
```

3. Run je programma. Je uitvoer ziet er (deels) zo uit:

```
Kleur: groen
Omtrek: 14.0
Oppervlakte: 10.0
Lengte: 5.0
Breedte: 2.0
Hoekpunt: (0,00, 2,00)

Kleur: paars
Omtrek: 14.0
Oppervlakte: 10.0
Lengte: 5.0
Breedte: 2.0
Hoekpunt: (3,00, 7,00)

Kleur: oranje
Omtrek: 18.0
Oppervlakte: 20.0
Lengte: 5.0
Breedte: 4.0
Hoekpunt: (5,00, 7,00)

Kleur: blauw
Omtrek: 16.0
Oppervlakte: 15.0
Lengte: 5.0
Breedte: 3.0
Hoekpunt: (5,00, 3,00)
```

Hieronder vul je het volgende canvas.



4. Voer de volgende code in je main-methode en run je programma.

```
Canvas canvas2 = new Canvas(10, 7);
canvas2.voegFiguurToe(new Rechthoek(4, 3, new Punt(0, 7), "rood"));
canvas2.voegFiguurToe(new Cirkel(1.5, new Punt(1.5, 1.5), "geel"));
canvas2.voegFiguurToe(new Rechthoek(7, 2, new Punt(3, 2), "groen"));
canvas2.voegFiguurToe(new Rechthoek(4, 1, new Punt(5, 7), "oranje"));
canvas2.voegFiguurToe(new Cirkel(2, new Punt(8, 5), "paars"));
canvas2.voegFiguurToe(new Rechthoek(5, 3, new Punt(5, 3), "blauw"));
System.out.println(canvas2);
```

↑ De uitvoer toont voor iedere figuur de juiste informatie.

6.6 ArrayList als methode parameter

1. Voeg in je Launcher onder de methode *toonInformatie()* de volgende methode toe.

```
public static void toonInformatieAlleFiguren(ArrayList<Figuur> figuren) {
    for (Figuur figuur : figuren) {
        toonInformatie(figuur);
        System.out.println();
    }
}
```

↑ *figuren* is de naam van een variabele die naar een `ArrayList` van `Figuren` verwijst. Je kunt daardoor een for-each loop gebruiken om de afzonderlijke figuren in de `ArrayList` te benaderen.

2. In de volgende stap ga je de methode uitproberen met het attribuut `mijnFiguren` in de klasse `Canvas`. Die is daar `private`, dus je moet eerst een getter maken.
3. Probeer vervolgens de methode uit door in de main-methode onderin de volgende regel te plaatsen:

```
toonInformatieAlleFiguren(canvas.getMijnFiguren());
```

6.7 ArrayList als uitvoervariabele

1. Voeg in je klasse Canvas de volgende methode toe.

```
public ArrayList<Figuur> geefFigurenMetGrotereOppervlakte(double
grenswaarde){
    ArrayList<Figuur> gevraagdeFiguren = new ArrayList<>();
    for (Figuur figuur : mijnFiguren) {
        if (figuur.geefOppervlakte() > grenswaarde) {
            gevraagdeFiguren.add(figuur);
        }
    }
    return gevraagdeFiguren;
}
```

↑ Deze methode gaat door de *ArrayList* met figuren, die in een canvas zitten en vergelijkt de oppervlakte van ieder figuur met de meegegeven grenswaarde. Als de oppervlakte groter is, dan wordt de figuur toegevoegd aan de *ArrayList* `gevraagdeFiguren`. Wanneer de methode klaar is met de *for-each loop*, dan wordt de *ArrayList* geretourneerd.

2. Probeer de methode uit, door onderin de main-methode de volgende regel te plaatsen:

```
toonInformatieAlleFiguren(canvas.geefFigurenMetGrotereOppervlakte(10));
```

↑ Merk op dat we de uitvoer van de methode `geefFigurenMetGrotereOppervlakte(25)` meteen gebruiken als invoer voor de methode `toonInformatieAlleFiguren()`

6.8 instanceof en typecasting

Voordat een figuur aan een canvas kan worden toegevoegd, kun je eerst kijken of het wel binnen het canvas past. Om de situatie iets te vereenvoudigen wordt er hieronder alleen naar de grootte van het figuur gekeken en niet naar de positie:

- Een cirkel past in een rechthoekig canvas als de straal kleiner is of gelijk is aan de halve breedte.
- Een rechthoek past in een rechthoekig canvas als de lengte van de rechthoek kleiner dan of gelijk is aan de lengte van het canvas én als de breedte van de rechthoek kleiner dan of gelijk is aan de breedte van het canvas.

1. Lees Liang 11.9

2. Voer de volgende code in de klasse Canvas.

```
private boolean figuurPastAlsVormInCanvas(Figuur figuur) {
    boolean figuurPast = false;
    if (figuur instanceof Rechthoek) {
        if (((Rechthoek) figuur).getLengte() <= lengte &&
                ((Rechthoek) figuur).getBreedte() <= breedte){
            figuurPast = true;
        }
    } else {
        if (((Cirkel) figuur).getStraal() <= breedte / 2) {
            figuurPast = true;
        }
    }
    return figuurPast;
}
```

↑ De conditie `if (figuur instanceof Rechthoek)` geeft true als de variabele `figuur` van de subklasse `Rechthoek` is.

↑ Je wilt de getters van `Rechthoek` gebruiken, een `Figuur` heeft die methodes niet. Je **moet** het object nu *typecasten* naar `Rechthoek` d.m.v. `(Rechthoek)`. Dat gaat goed omdat je gecontroleerd hebt of het een `Rechthoek` is.

↑ Als `figuur` niet van de subklasse `Rechthoek` is, dan is het nu automatisch van de subklasse `Cirkel` en kunnen we `figuur` typecasten naar `Cirkel`.

3. Voeg een check toe in de methode `voegFiguurToe()` in de klasse `Canvas`, zoals hieronder:

```
public void voegFiguurToe(Figuur figuur) {
    if (figuurPastAlsVormInCanvas(figuur)) {
        mijnFiguren.add(figuur);
        System.out.println("Dit figuur is toegevoegd");
    } else {
        System.out.println("Dit figuur is te groot");
    }
}
```

4. Pas de code aan in de main-methode in de launcher om enkele figuren te testen.

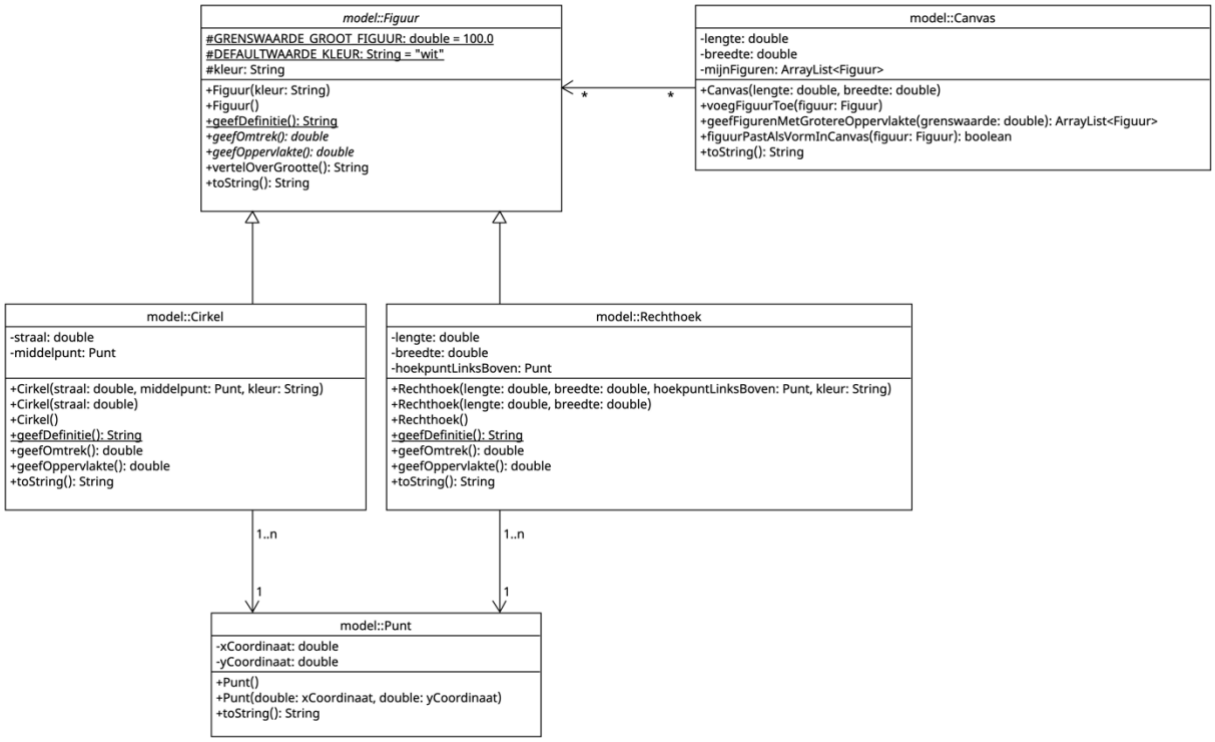
```
Canvas canvas2 = new Canvas(10, 7);
canvas2.voegFiguurToe(new Rechthoek(4, 3, new Punt(0, 7), "rood"));
canvas2.voegFiguurToe(new Cirkel(4, new Punt(1.5, 1.5), "geel"));
canvas2.voegFiguurToe(new Rechthoek(9, 8, new Punt(3, 2), "groen"));
canvas2.voegFiguurToe(new Rechthoek(4, 1, new Punt(5, 7), "oranje"));
canvas2.voegFiguurToe(new Cirkel(2, new Punt(8, 5), "paars"));
canvas2.voegFiguurToe(new Rechthoek(11, 3, new Punt(5, 3), "blauw"));
```

5. Run je programma. Je uitvoer ziet er zo uit:

```
Dit figuur is toegevoegd
Dit figuur is te groot
Dit figuur is te groot
Dit figuur is toegevoegd
Dit figuur is toegevoegd
Dit figuur is te groot
```

6.9 Klassendiagram

Het totale klassendiagram ziet er nu zo uit:



↑ De klasse `Canvas` bevat 0 of meer objecten van de klasse `Figuur`.
↑ Een object van de klasse `Figuur` kan in 0 of meer objecten van de klasse `Canvas` worden geplaatst.
↑ Een canvas weet van het bestaan van de figuren die het bevat.
↑ Een figuur weet niet in welke canvassen het zit.

+	betekent	public
-	betekent	private
#	betekent	protected
ALL_CAPS	betekent	constante
Onderstreept	betekent	static
Schuingedrukt	betekent	abstract
Gesloten pijl	betekent	subklasse
Open pijl	betekent	unidirectioneel

6.10 Opdrachten

1. Doe Opdracht **ArrayList-1 Bedrijf**.
2. Doe Opdracht **ArrayList-2** (niveau: basis).

7 INTERFACES, STRINGBUILDER

7.1 Samenvatting

1. Een *interface* is een klasse-achtige constructie om gedrag voor klassen te specificeren
2. Een interface heeft alleen maar abstracte methoden
3. De onderliggende klassen moeten de methoden *implementeren*
4. Je kunt zelf een interface maken, je gebruikt dan het keyword *interface* in plaats van `class`
5. ***Java*** heeft een `Comparable<T>` interface die de methode `compareTo()` voorschrijft. De methode vergelijkt objecten onderling zodat je kunt bepalen welke groter is en wanneer ze gelijk zijn. De klasse Figuur implementeert `Comparable<Figuur>`, dit geef je aan met het keyword *implements*
6. De `ArrayList<>` is een implementatie van de `List<>` interface. Het is gebruikelijk om bij het declareren van een `ArrayList<>` als datatype de interface `List<>` te gebruiken. Hier is weer sprake van *polymorfisme*.
   Voorbeeld: `List<String> woordenlijst = new ArrayList<>()`
7. De *StringBuilder*-klasse wordt gebruikt om efficiënter Strings aan elkaar te plakken

7.2 De Comparable Interface

1. Lees Liang 13.5, 13.6 en 13.8
2. Indien nodig kun je het project ***OOP Meetkunde Polymorfisme-Abstracte Klasse*** van ***DLO*** halen. Dit bevat de klassen zoals je die in het vorige onderdeel hebt afgesloten.
3. Verander de klasse `Figuur` door de interface `Comparable` als volgt toe te voegen:

```
public abstract class Figuur implements Comparable<Figuur> {
```

↑ Let op: bij de Comparable interface moet je aangeven met welk type je wilt vergelijken. Dit geef je aan tussen de < > -haken.

↑ Standaard vergelijk je een object van een bepaalde klasse met een ander object van dezelfde klasse. Hier ga je in de klasse Figuur implementeren hoe je een object met een ander object van type Figuur moet vergelijken.

Java'nın hazır bir interface'i vardır: Comparable<T>. Bu interface, compareTo() adında bir method içerir. Bu method iki nesneyi karşılaştırır (hangisi büyük, hangisi küçük, ya da eşit).

Örnek: Figuur sınıfı, Comparable<Figuur> interface'ini uygular.

Interface, Java'da bir sınıfa (class) benzer bir yapıdır ve sınıfların hangi davranışlara (methodlara) sahip olması gerektiğini tanımlar.

Bir interface içinde yalnızca abstract (soyut) metodlar bulunur. Yani methodların sadece imzası (adı, parametreleri vs.) olur, gövdesi (içeriği) olmaz.

Interface kullanan sınıflar bu methodları zorunlu olarak kendileri tanımlamalıdır (override).

Kendi interface'ini yazmak mümkündür. Bunu yapmak için class yerine interface anahtar kelimesi kullanılır.

ArrayList<>, List<> interface'inin bir uygulamasıdır. Java'da genelde veri tipi olarak interface kullanmak tercih edilir.Bu sayede polimorfizm sağlanmış olur (birden fazla farklı nesne türü aynı interface türünden olabilir).

StringBuilder sınıfı, string'leri daha verimli şekilde birleştirmek için kullanılır.

String nesneleri Java'da sabittir (immutable), her ekleme yeni nesne oluşturur. StringBuilder bunu optimize eder.

Comparable<Figuur> demek, bu sınıfın kendisiyle aynı türden (Figuur) nesnelerle karşılaştırılabilir olduğunu belirtir.

compareTo() methodu içinde, iki Figuur nesnesinin nasıl karşılaştırılacağını sen tanımlarsın. Örneğin alana (area) göre sıralama gibi:

```
@Override
public int compareTo(Figuur andereFiguur) {
   return Double.compare(this.berekenOppervlakte(),
andereFiguur.berekenOppervlakte());
}
```

4. Voeg een methode `compareTo()` toe:

```
@Override
public int compareTo(Figuur anderFiguur) {
    if (this.geefOppervlakte() > anderFiguur.geefOppervlakte()) {
        return 1;
    } else if (this.geefOppervlakte() < anderFiguur.geefOppervlakte()) {
        return -1;
    } else {
        return 0;
    }
}
```

↑ Merk op dat compareTo() een `int` terug geeft
↑ De methode compareTo() moet een positief getal terug geven voor het geval dat dit object groter is dan het andere object. Je geeft een negatief getal terug als het kleiner is en je geeft 0 terug als beide objecten gelijk zijn. We kiezen vaak voor 1, -1 en 0, maar dat hoeft niet.
↑ Deze code implementeert het vergelijken van figuren op basis van de oppervlakte.

5. Voeg de volgende code (tijdelijk) toe aan de main-methode:

```
Figuur figuur1 = new Rechthoek(3, 3, new Punt(0, 7), "rood");
Figuur figuur2 = new Rechthoek(3, 2, new Punt(0, 4), "geel");
System.out.println(figuur1.compareTo(figuur2));
if (figuur1.compareTo(figuur2) > 0) {
    System.out.println("Figuur1 is groter dan figuur2");
}
```

6. Run het programma en bekijk de output. Haal daarna bovenstaande code weer weg.

7. Verander de `toString()` methode in de klasse `Canvas` als volgt:

```
@Override
public String toString() {
    String canvasString = "";
    Collections.sort(mijnFiguren);
    for (Figuur figuur : mijnFiguren) {
        canvasString += figuur.toString() + "\n\n";
    }
    return canvasString;
}
```

↑ Merk op dat de regel `Collections.sort(mijnFiguren)` de ArrayList `mijnFiguren` eerst sorteert. Daarna worden de items in oplopende volgorde afgedrukt
↑ Sorteren veronderstelt dat je objecten kunt vergelijken op groter, gelijk en kleiner. Om de lijst `mijnFiguren` te kunnen sorteren moet de klasse `Figuur` de `Comparable<>` interface implementeren. Dat is hierboven dus gebeurd d.m.v. `compareTo()` waarin de oppervlaktes met elkaar worden vergeleken. CompareTo geeft altijd een positief getal, een negatief getal of 0 terug.

8. Verander de code in je main-methode als volgt:

```
Canvas canvas = new Canvas(10, 7);
canvas.voegFiguurToe(new Rechthoek(3, 3, new Punt(0, 7), "rood"));
canvas.voegFiguurToe(new Rechthoek(3, 2, new Punt(0, 4), "geel"));
canvas.voegFiguurToe(new Rechthoek(5, 2, new Punt(0, 2), "groen"));
canvas.voegFiguurToe(new Rechthoek(5, 2, new Punt(3, 7), "paars"));
canvas.voegFiguurToe(new Rechthoek(5, 4, new Punt(5, 7), "oranje"));
canvas.voegFiguurToe(new Rechthoek(5, 3, new Punt(5, 3), "blauw"));
System.out.println(canvas);
```

9. Run je programma en merk op dat de figuren in oplopende volgorde van oppervlakte worden getoond.
   De meeste Java-klassen implementeren al de Comparable interface en hebben dus een implementatie van de `compareTo()` methode. We kunnen daar vaak handig gebruik van maken.
   Als je de figuren wilt sorteren op de naam van de kleur, dan moet je zorgen dat de figuren vergeleken worden op hun kleurnaam. Die is van het type `String` en die klasse heeft de `compareTo()` methode die vergelijkt op alfabetische volgorde.

10. Probeer de volgende print-statements:

```
System.out.println("ab".compareTo("ac"));
System.out.println("ab".compareTo("ab"));
System.out.println("ab".compareTo("aa"));
System.out.println("ab".compareTo("az"));
System.out.println("zo".compareTo("ab"));
```

Wat zie je als output? (-1, 0, 1, -24, 25) Waarom?

11. Verander de code in de `compareTo()` methode van de `Figuur` klasse als volgt:

```
@Override
public int compareTo(Figuur anderFiguur) {
    return this.getKleur().compareTo(anderFiguur.getKleur());
}
```

↑ Bedenk dat this.getKleur en anderFiguur.getKleur() Strings zijn
↑ Omdat de compareTo-methode van String op je juiste manier werkt en positieve of negatieve waarden of 0 teruggeeft, kun je deze waarden in je "eigen" Figuur-compareTo gewoon "doorsluizen".

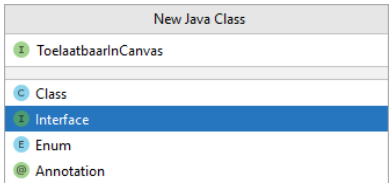
12. Run je programma nog een keer en merk op dat de figuren in alfabetisch volgorde van kleur worden getoond

7.3 De ToelaatbaarInCanvas Interface

De volgende oefening gaat om het maken van een eigen interface. We herschrijven de code van het onderdeel ***6.2 ArrayList***, zodat iedere *subklasse* van de klasse `Figuur` zelf kan berekenen of deze in een canvas met een bepaalde lengte en breedte past.

1. Ga met je muis naar de package `model` en kies met de rechtermuisknop voor New > Java Class.

2. Type `ToelaatbaarInCanvas` en kies voor de optie Interface, zoals in het volgende scherm:



3. Voeg dan een regel toe, zodat je code er als volgt uitziet:

```
package model;

public interface ToelaatbaarInCanvas {
    boolean pastInCanvas(double lengte, double breedte);
}
```

↑ Je hebt in de package `model` nu een *interface* toegevoegd, die een *abstracte methode* heeft. Een interface kan alleen maar abstracte methodes bevatten, daarom hoef je het keyword `abstract` in een interface niet te gebruiken. Iedere niet-abstracte klasse, die deze interface gebruikt, is verplicht om de methode te overriden. Bij een abstracte klasse wordt dit dus van een subklasse verwacht.

4. Pas de definitie van de klasse `Figuur` als volgt aan:

```
public class Figuur implements ToelaatbaarInCanvas, Comparable<Figuur>{
```

↑ Je hebt de interface toegevoegd aan de klasse `Figuur`. Omdat dit zelf een abstracte klasse is hoef je de methode `pastInCanvas()` niet te overriden. Dat moet je dan wel doen in de subklassen `Cirkel` en `Rechthoek`.

5. Voeg dus de volgende code toe, met de voor de klasse `Cirkel` juiste berekening (zie de klasse `Canvas` in onderdeel ***ArrayList***).

```
@Override
public boolean pastInCanvas(double lengte, double breedte) {
    return straal <= breedte / 2;
}
```

6. Voeg de volgende code toe, met de voor de klasse `Rechthoek` juiste berekening (zie de klasse `Canvas` in onderdeel ***ArrayList***).

```
@Override
public boolean pastInCanvas(double lengte, double breedte) {
    return this.lengte <= lengte && this.breedte <= breedte;
}
```

7. Je hebt de berekeningen niet meer in de klasse `Canvas` nodig, dus je kunt de methode `figuurPastAlsVormInCanvas()` verwijderen

8. Je moet nog wel controleren of een figuur past (dus toelaatbaar is), voordat je de figuur toevoegt aan een canvas. Verander de code in de methode `voegFiguurToe()` als volgt:

```
public void voegFiguurToe(Figuur figuur) {
    if (figuur.pastInCanvas(lengte, breedte)) {
        mijnFiguren.add(figuur);
        System.out.println("Je figuur is toegevoegd");
    } else {
        System.out.println("Je figuur past niet.");
    }
}
```

9. Je kunt je code nu testen door een te grote cirkel en een te grote rechthoek toe te voegen aan je canvas (met lengte 10 en breedte 7).

7.4 De StringBuilder klasse

1. Lees Liang 10.11

Hieronder staat de `toString()` methode in de klasse `Canvas`.

```
@Override
public String toString() {
    String canvasString = "";
    for (Figuur figuur : mijnFiguren) {
        canvasString += figuur.toString() + "\n\n";
    }
    return canvasString;
}
```

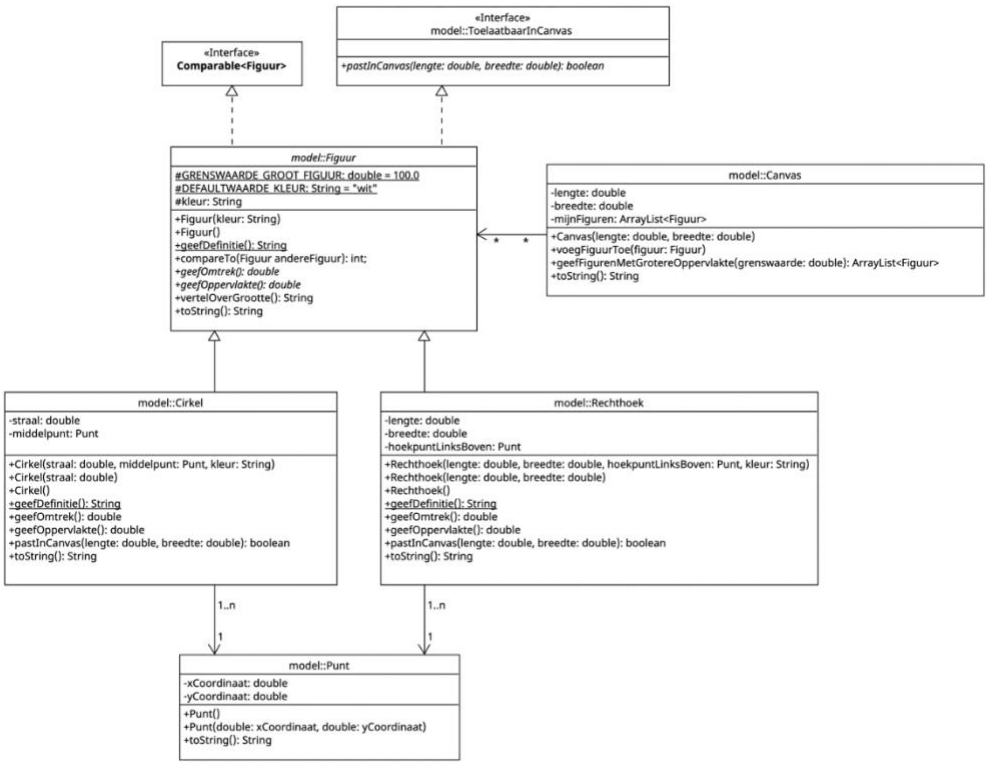
In de methode wordt de String `canvasString` via een *foreach-loop* iedere keer uitgebreid met de `toString()` van de afzonderlijke figuren door middel van de *String-concatenatie* +=. De `StringBuilder` klasse is hier geschikter voor. De compiler roept namelijk zelf iedere keer een nieuwe instantie van `StringBuilder` aan als twee `Strings` aan elkaar gekoppeld worden door +=. Het is daarom efficiënter om eenmalig zelf een instantie van de `StringBuilder` te maken.

2. Verander de `toString()` methode zoals hieronder.

```
@Override
public String toString() {
    StringBuilder canvasStringBuilder = new StringBuilder();
    for (Figuur figuur : mijnFiguren) {
        canvasStringBuilder.append(String.format("%s\n\n",
         figuur.toString()));
    }
    return canvasStringBuilder.toString();
}
```

↑ `canvasStringBuilder` is nu een object van de klasse `StringBuilder` en wordt geïnstantieerd.
↑ De methode `append()` wordt gebruikt om het `StringBuilder`-object uit te breiden.
↑ Tot slot wordt de `StringBuilder.toString()`-methode aangeroepen om een String te retourneren. Waarom?

7.5 Klassendiagram



+	betekent public
-	betekent private
#	betekent protected
ALL_CAPS	betekent constante
Onderstreept	betekent static
Schuingedrukt	betekent abstract
Gesloten pijl met dichte lijn	betekent subklasse
Gesloten pijl met stippellijn	betekent interface
Open pijl	betekent unidirectioneel

7.6 Opdrachten

1. Doe Opdracht **Interface-1 Bedrijf**.
2. Doe Opdracht **Interface-2 Vervoermiddelen Belasting** (niveau: basis).
3. Mocht je nog zin en tijd hebben, doe dan Opdracht **Interface-3 Wagenpark** (niveau: basis).
4. Als voorbereiding voor de assessment opdracht kun je Opdracht **Botenverhuur Bouw klassenstructuur** (niveau: basis) doen.