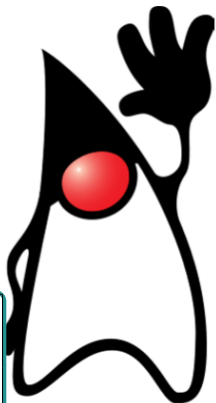


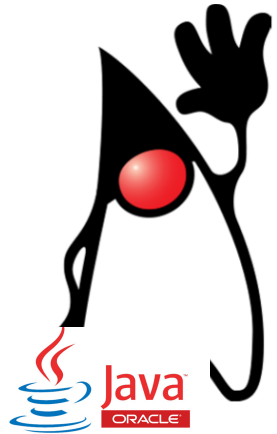
Klassen, Objecten en methoden in Java



Klasgroep	1EO-ICT
Opleiding	Bachelor Elektronica-ICT
Lokaal	Groot Auditorium
Tijdstip	maandag lestijd 3
Docent	Katja Verbeeck
Contact	katja.verbeeck@odisee.be
Handboek	hfst 4 - p 103 - p127

Inhoud

- 1 Introductie
- 2 Klassen
- 3 Objecten
- 4 Methoden in java
 - Methoden die een resultaat teruggeven
 - Methoden met parameters
 - Constructoren



Wat is een klasse?

Elk Java programma bestaat uit de definitie van enkele klassen in combinatie met een main methode.

Een **klasse** zelf is een blauwdruk, i.e. een abstracte beschrijving van een nieuw type, en drukt uit hoe een element van dat type (wat men een **object** noemt) gebouwd moet worden. Het legt tevens vast welke **data** het object bevat en welke **operaties** op die data toegelaten zijn. De operaties worden beschreven aan de hand van **methoden**, de data aan de hand van variabelen. Beide (methoden en variabelen) worden **members** van de klasse genoemd. De data members worden ook **data velden**, **fields** of **instance variabelen** genoemd.

Algemene vorm van een klasse

```
public class NaamVanDeKlasse{  
    // Een klasse bestaat uit data members en methoden  
  
    // declaratie van de instance variabelen  
    type var1;  
    type var2;  
    ...  
  
    // declaratie van de methoden  
    public returnType naamMethode1(parameters){  
        // body van methode1  
    }  
    public returnType naamMethode2(parameters){  
        // body van methode2  
    }  
    ...  
}
```

Merk op

- Een klasse stelt een type voor en moet dus een logisch samenhangende entiteit voorstellen.
- Merk op, een main methode is niet vereist in elke klasse! De main methode is het startpunt van elk programma, maar dit kan nu uit meerdere klassen bestaan.

Voorbeeld

Stel je wil een nieuw type voor voertuigen definiëren.

Eerste vraag : uit welke nuttige data bestaat een voertuig ? Of nog, welke data geeft een goede beschrijving van een voertuig. Is het van toepassing voor elk voertuig dat je wil voorstellen?

Goede kandidaten :

- het aantal passagiers dat het voertuig kan vervoeren
- de brandstofcapaciteit dat het voertuig kan opnemen (aantal liters)
- het verbruik : het aantal kilometers dat per liter brandstof kan afgelegd worden (km/l)

Tweede vraag : welke operaties op deze data moeten voorzien worden ?
(zie verder : methoden)

Een nieuw type voor Voertuigen

```
public class Voertuig{  
    int passagiers; // max aantal passagiers  
    int capaciteit; // max aantal liters  
        brandstof  
    double verbruik; // verbruik  
}
```

Merk op, op dit moment is alleen het nieuwe type zelf gedefinieerd. Er zijn nog geen objecten (elementen van het type voertuig) aangemaakt!

Het aanmaken van voertuigen

Voor het aanmaken van objecten maak je gebruik van de **new** operator en de constructor van een klasse.

```
Voertuig miniBus = new Voertuig();
```

miniBus heeft een fysieke plaats in het geheugen gekregen en kan dus effectief zijn eigen set van waarden bijhouden. Je kan de dot(.) operator gebruiken om waarden toe te kennen aan de instance variabelen van miniBus.

```
miniBus.passagiers = 7;  
miniBus.capaciteit = 80;  
miniBus.verbruik = 6.0;
```


Een programma met 1 voertuig object (en 2 klassen !)

```
public class DemoVoertuig {  
    // dit programma zal voertuigen aanmaken en gebruiken  
    public static void main(String[] args){  
        Voertuig miniBus = new Voertuig();  
        double afstand;  
        //waarden toekennen aan de data  
        miniBus.passagiers = 7;  
        miniBus.capaciteit = 80;  
        miniBus.verbruik = 6.0;  
  
        //bereken de afstand die afgelegd kan worden met een volle tank  
        afstand = miniBus.capaciteit * miniBus.verbruik;  
        System.out.println("De miniBus kan " + miniBus.passagiers + " personen  
            vervoeren en een afstand van " + Math.round(afstand * 100)/100.0 +  
            "kilometers afleggen met een volle tank ");  
    }  
}
```

Een programma met 2 voertuigen (en 2 klassen)

```
public class Demo2Voertuig {
// dit programma zal voertuigen aanmaken en gebruiken
    public static void main(String[] args){
        Voertuig miniBus = new Voertuig();
        Voertuig _2pk = new Voertuig();
        double afstand1, afstand2;
        //velden van de miniBus
        miniBus.passagiers = 7;
        miniBus.capaciteit = 80;
        miniBus.verbruik = 6.0;

        //velden van de miniBus
        _2pk.passagiers = 4;
        _2pk.capaciteit = 30;
        _2pk.verbruik = 10.0;

        //bereken de afstand die afgelegd kan worden met een volle tank
        afstand1 = miniBus.capaciteit * miniBus.verbruik;
        afstand2 = _2pk.capaciteit * _2pk.verbruik;
        System.out.println("De miniBus kan " + miniBus.passagiers + " personen
            vervoeren en een afstand van " + Math.round(afstand1 * 100)/100.0 + "
            kilometers afleggen met een volle tank ");
        System.out.println("Het _2pk'tje kan " + _2pk.passagiers + " personen
            vervoeren en een afstand van " + Math.round(afstand2 * 100)/100.0 + "
            kilometers afleggen met een volle tank ");
    }
}
```

Data-abstractie : Elk object heeft zijn eigen data

miniBus



passagiers	7
Capaciteit	80
verbruik	6.0

_2pk

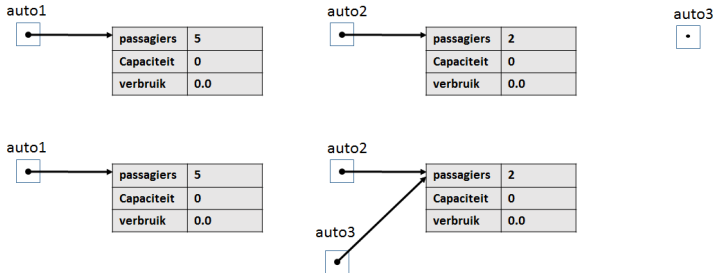


passagiers	4
Capaciteit	30
verbruik	10.0

Kopiëren van objecten

```
public class Voertuigen2 {  
  
    public static void main(String[] args){  
        Voertuig auto1 = new Voertuig();  
        Voertuig auto2, auto3 ; // declaratie  
        auto2 = new Voertuig(); // ken een plaats uit het  
            geheugen toe  
        System.out.println(auto1 + " " + auto2); // auto3  
            heeft nog geen geheugenplaats !  
        auto1.passagiers = 5;  
        auto2.passagiers = 2;  
  
        auto3 = auto2; // laat auto3 naar dezelfde  
            geheugenplaats wijzen!  
        System.out.println(auto3);  
        System.out.println(auto3.passagiers); // -> 2  
    }  
}
```

Elk object heeft zijn eigen data



Methoden

- Methoden zijn subroutines (functies) die de data van je klasse kan manipuleren.
- Vaak is het ook de enige manier om (veilig) toegang te verlenen tot die data.
- Elke methode voert 1 welbepaalde taak uit.
- Een methode kan input krijgen via parameters.
- Een methode kan output teruggeven via een return
- Elke methode is volledig beschreven via zijn definitie, deze beschrijving geeft alle nodige informatie om een methode aan te roepen en het resultaat op te vangen

Algemene vorm van een methode

```
returnType methodeNaam (parameters) {  
    // body van de methode  
}
```

Definitie van een methode

```
returnType methodeNaam (parameters)
```

Definities uit de API

Definitie		
char	charAt	(int i)
boolean	startsWith	(String s)
String	toUpperCase	()
String	toLowerCase	()
String	substring	(int i)
String	substring	(int i, int j)

Een methode voor klasse Voertuig

```
public class VoertuigMM{  
    int passagiers; // max aantal passagiers  
    int capaciteit; // max aantal liters brandstof  
    double verbruik; // verbruik  
  
    // de klasse kan beter zelf de afstand bepalen en  
    // teruggeven als resultaat  
    public double afstand () {  
        double afstand = Math.round(capaciteit *  
            verbruik * 100) / 100.0;  
        return afstand;  
    }  
}
```

Merk op : afstand() berekent een double en dus is het logisch dat dit resultaat teruggegeven kan worden aan de oproeper. Die kan dan zelf beslissen wat ermee moet gebeuren.

Procedurele abstractie

Merk op, eens de methode afstand bestaat hoeft een gebruiker niet meer na te denken over hoe de afstand berekend moet worden. De logica van deze oplossing zit in de body van de methode en is voor de gebruiker iets abstracts geworden.

Wanneer `miniBus.afstand()` opgeroepen wordt, krijgt de afstand methode de controle en zal intern in de methode de data van de `miniBus` gebruikt worden om de berekening te doen. Na afloop van de methode krijgt het programma dat de oproep deed de controle terug.

Een methodeoproep gebeurt altijd relatief tov. een object, zodat de methode afgehandeld kan worden met de data van dat betreffende object. De oproep `_2pk.afstand()` zal dus dezelfde berekening uitvoeren maar nu met de interne data van het `_2pk` object.

return

return is een voorbehouden Java keyword waarmee een methode de controle terug kan geven aan de oproeper en dus de methode beëindigt. **return** geeft je methode een exit-point.

```
public class MyReturn {  
    public static void main(String[] args){  
        int i;  
        for(i=1; i < 100; i++){  
            if (i == 5) return;  
            System.out.println(i);  
        }  
        System.out.println("Wordt dit nog  
            uitgeprint?");  
    }  
}
```

return met een waarde

return kan ook een waarde teruggeven aan de oproeper. Wanneer dit gebeurt moet deze waarde van hetzelfde type zijn als het returntype in de methodedefinitie van de methode.

Wanneer de methode geen waarde teruggeeft dan staat het returntype van de methode op `void`.

Een methode hoeft geen `return` te bevatten. In dit geval worden alle expressies van de body van de methode uitgevoerd tot aan de afsluitende accolade. Deze methode heeft altijd returntype `void`.

Een programma met ...

```
public class DemoVoertuigMM{
// dit programma zal voertuigen aanmaken en gebruiken
    public static void main(String[] args){
        VoertuigMM miniBus = new VoertuigMM();
        VoertuigMM _2pk = new VoertuigMM();
        //velden van de miniBus
        miniBus.passagiers = 7;
        miniBus.capaciteit = 80;
        miniBus.verbruik = 6.0;
        //velden van het _2pk'tje
        _2pk.passagiers = 4;
        _2pk.capaciteit = 30;
        _2pk.verbruik = 10.0;
        //bereken de afstand die afgelegd kan worden met een volle tank
        System.out.println("De miniBus kan " + miniBus.passagiers + " personen
            vervoeren en een afstand van " + miniBus.afstand() + "kilometers
            afleggen met een volle tank ");
        System.out.println("Het _2pk'tje kan " + _2pk.passagiers + " personen
            vervoeren en een afstand van " + _2pk.afstand() + " kilometers
            afleggen met een volle tank ");
    }
}
```

Ander programma met ...

```
public class Demo2VoertuigMM{
// dit programma zal voertuigen aanmaken en gebruiken
    public static void main(String[] args){
        VoertuigMM miniBus = new VoertuigMM();
        VoertuigMM _2pk = new VoertuigMM();
        //velden van de miniBus
        miniBus.passagiers = 7;
        miniBus.capaciteit = 80;
        miniBus.verbuik = 6.0;
        //velden van het _2pk'tje
        _2pk.passagiers = 4;
        _2pk.capaciteit = 30;
        _2pk.verbuik = 10.0;
        // Welke auto kan de langste afstand afleggen?
        double afstand_minibus = miniBus.afstand(); // vang het resultaat op !
        double afstand_2pk = _2pk.afstand();

        if(afstand_minibus > afstand_2pk)
            System.out.println("De miniBus kan kan meer kilometers afleggen met een
                               volle tank ");
    }
}
```

Methoden met parameters

Via **parameters** kan je input geven aan een methode. Deze parameters kunnen in de body van de methode gebruikt worden zoals een lokale variabele.

Deze variabele krijgt pas een waarde toegekend bij oproep van deze methode. Men zegt dan dat de **formele parameter** de waarde van de **actuele parameter** krijgt bij een methode-oproep.

Een methode kan meerdere parameters als input krijgen. (terwijl er max 1 returnwaarde kan zijn)

Wanneer de oproep de controle teruggeeft aan de oproeper, verdwijnt de binding tussen formele en actuele parameter.

Een methode met parameters voor de klasse Voertuig

```
public class VoertuigParam{
    int passagiers; // max aantal passagiers
    int capaciteit; // max aantal liters brandstof
    double verbruik; // verbruik

    // bereken de afstand met een vole tank
    public double afstand () {
        double afstand = Math.round(capaciteit * verbruik * 100) / 100.0;
        return afstand;
    }

    // gegeven de af te leggen weg, bereken de nodige brandstofhoeveelheid
    public double berekenBrandstof(double km){
        return (Math.round(km / verbruik * 100) / 100.0);
    }
}
```


Een programma met ...

```
public class DemoVoertuigParam{
// dit programma zal voertuigen aanmaken en gebruiken
    public static void main(String[] args){
        VoertuigParam miniBus = new VoertuigParam();
        VoertuigParam _2pk = new VoertuigParam();
        //velden van de miniBus
        miniBus.passagiers = 7;
        miniBus.capaciteit = 80;
        miniBus.verbruik = 6.0;
        //velden van het _2pk'tje
        _2pk.passagiers = 4;
        _2pk.capaciteit = 30;
        _2pk.verbruik = 10.0;
        //bereken het aantal liter benzine nodig om 250 km af te leggen
        System.out.println("De miniBus heeft " + miniBus.berekenBrandstof(250.0) +
            " liter brandstof nodig om 250 kilometers af te leggen.");
        System.out.println("Het _2pk'tje heeft " + _2pk.berekenBrandstof(250.0) +
            " liter nodig om 250 kilometers af te leggen. ");
    }
}
```

Constructor methoden

Een constructor methode **initialiseert een object**. Deze methode heeft dezelfde naam als de naam van de klasse. Een constructormethode heeft wel **geen returntype**.

Men maakt onderscheid tussen **de standaard (of default) constructor** en **geparameteriseerde constructoren**.

De default constructor wordt automatisch voor je aangemaakt wanneer je een nieuwe klasse implementeert. Deze zal al je data velden initialiseren met hun default waarden (0, 0.0, false and null)

Een geparametriseerde constructor schrijf je zelf! Deze kan als input enkele parameters nemen, waarmee je de datavelden van de klasse zelf kan initialiseren. Let wel, wanneer je zelf expliciet een constructor schrijft zal er geen default constructor meer voor je gegenereerd meer worden!

Default constructor

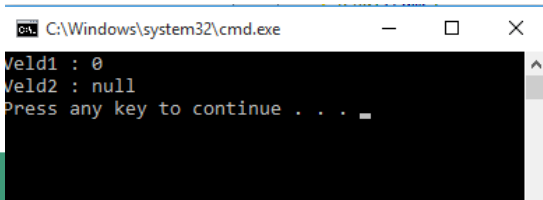
```
public class EenKlasse {  
    int veld1;  
    String veld2;  
  
    public void printAlleVelden(){  
        System.out.println("Veld1 : " + veld1);  
        System.out.println("Veld2 : " + veld2);  
    }  
}
```

Default constructor

```
public class EenKlasseDemo {  
  
    public static void main(String[] args){  
        EenKlasse voorbeeld = new EenKlasse();  
        voorbeeld.printAlleVelden();  
  
    }  
  
}
```

Default constructor

```
public class EenKlasseDemo {  
  
    public static void main(String[] args){  
        EenKlasse voorbeeld = new EenKlasse();  
        voorbeeld.printAlleVelden();  
  
    }  
  
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt displays the output of a Java program: 'Veld1 : 0', 'Veld2 : null', and 'Press any key to continue . . .'. A cursor is visible on the line 'Press any key to continue . . .'. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Een default constructor die je zelf schrijft

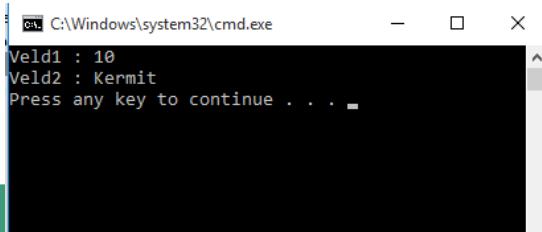
```
public class EenKlasse2 {  
    int veld1;  
    String veld2;  
  
    public EenKlasse2(){  
        veld1 = 10;  
        veld2 = "Kermit";  
    }  
  
    public void printAlleVelden(){  
        System.out.println("Veld1 : " + veld1);  
        System.out.println("Veld2 : " + veld2);  
    }  
}
```

Een default constructor die je zelf schrijft

```
public class EenKlasseDemo2 {  
  
    public static void main(String[] args){  
        EenKlasse2 voorbeeld = new EenKlasse2();  
        voorbeeld.printAlleVelden();  
  
    }  
  
}
```

Een default constructor die je zelf schrijft

```
public class EenKlasseDemo2 {  
  
    public static void main(String[] args){  
        EenKlasse2 voorbeeld = new EenKlasse2();  
        voorbeeld.printAlleVelden();  
  
    }  
  
}
```



A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Windows\system32\cmd.exe'. The window has standard minimize, maximize, and close buttons. The command prompt displays the following text: 'Veld1 : 10', 'Veld2 : Kermit', and 'Press any key to continue . . .'. A cursor is visible at the end of the last line.

Niet-Default of geparametriseerde constructor

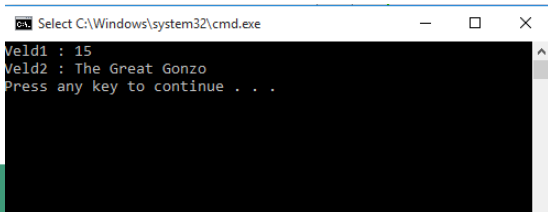
```
public class EenKlasse3 {  
    int veld1;  
    String veld2;  
  
    public EenKlasse3(int getal, String muppet){  
        veld1 = getal;  
        veld2 = muppet;  
    }  
  
    public void printAlleVelden(){  
        System.out.println("Veld1 : " + veld1);  
        System.out.println("Veld2 : " + veld2);  
    }  
}
```

Niet-Default of geparametriseerde constructor

```
public class EenKlasseDemo3 {  
  
    public static void main(String[] args){  
        EenKlasse3 voorbeeld = new EenKlasse3(15,"The Great Gonzo");  
        // EenKlasse3 voorbeeld2 = new EenKlasse3(); // compile error ! required:  
            int,String found: no arguments  
        voorbeeld.printAlleVelden();  
    }  
}
```

Niet-Default of geparametriseerde constructor

```
public class EenKlasseDemo3 {  
  
    public static void main(String[] args){  
        EenKlasse3 voorbeeld = new EenKlasse3(15,"The Great Gonzo");  
        // EenKlasse3 voorbeeld2 = new EenKlasse3(); // compile error ! required:  
        // int,String found: no arguments  
        voorbeeld.printAlleVelden();  
    }  
}
```



```
C:\> Select C:\Windows\system32\cmd.exe  
Veld1 : 15  
Veld2 : The Great Gonzo  
Press any key to continue . . .
```

Meerdere constructoren voor 1 klasse

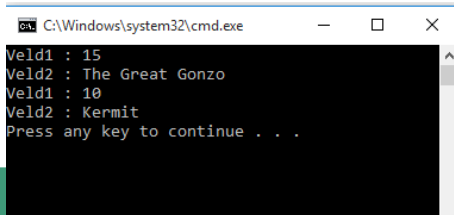
```
public class EenKlasse4 {  
    int veld1;  
    String veld2;  
  
    public EenKlasse4(){  
        veld1 = 10;  
        veld2 = "Kermit";  
    }  
  
    public EenKlasse4(int getal, String muppet){  
        veld1 = getal;  
        veld2 = muppet;  
    }  
  
    public void printAlleVelden(){  
        System.out.println("Veld1 : " + veld1);  
        System.out.println("Veld2 : " + veld2);  
    }  
}
```

Meerdere constructoren voor 1 klasse

```
public class EenKlasseDemo4 {  
  
    public static void main(String[] args){  
        EenKlasse4 voorbeeld = new EenKlasse4(15,"The Great Gonzo");  
        EenKlasse4 voorbeeld2 = new EenKlasse4(); // geen compile error  
        voorbeeld.printAlleVelden();  
        voorbeeld2.printAlleVelden();  
    }  
}
```

Meerdere constructoren voor 1 klasse

```
public class EenKlasseDemo4 {  
  
    public static void main(String[] args){  
        EenKlasse4 voorbeeld = new EenKlasse4(15,"The Great Gonzo");  
        EenKlasse4 voorbeeld2 = new EenKlasse4(); // geen compile error  
        voorbeeld.printAlleVelden();  
        voorbeeld2.printAlleVelden();  
    }  
}
```



```
C:\Windows\system32\cmd.exe  
Veld1 : 15  
Veld2 : The Great Gonzo  
Veld1 : 10  
Veld2 : Kermit  
Press any key to continue . . .
```

Voertuigen aanmaken met een geparametriseerde constructor

```
public class VoertuigConstr{
    int passagiers; // max aantal passagiers
    int capaciteit; // max aantal liters brandstof
    double verbruik; // verbruik

    // default Constructor
    public VoertuigConstr(){
        passagiers = 0;
        capaciteit = 0;
        verbruik = 0.0;
    }

    // niet-default constructor
    public VoertuigConstr(int pas, int cap, double verb){
        passagiers = pas;
        capaciteit = cap;
        verbruik = verb;
    }

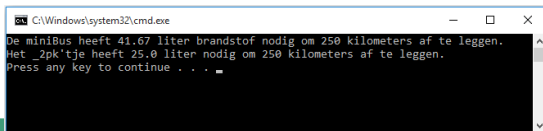
    // methoden
    ...
}
```

Voertuigen aanmaken met een geparametriseerde constructor

```
public class DemoVoertuigConstr{  
    // dit programma zal voertuigen aanmaken en gebruiken  
    public static void main(String[] args){  
        VoertuigConstr miniBus = new VoertuigConstr(7,80,6.0);  
        VoertuigConstr _2pk = new VoertuigConstr(4,30,10.0);  
        //bereken het aantal liter benzine nodig om 250 km af te leggen  
        System.out.println("De miniBus heeft " + miniBus.berekenBrandstof(250.0) +  
            " liter brandstof nodig om 250 kilometers af te leggen.");  
        System.out.println("Het _2pk'tje heeft " + _2pk.berekenBrandstof(250.0) +  
            " liter nodig om 250 kilometers af te leggen. ");  
    }  
}
```


Voertuigen aanmaken met een geparametriseerde constructor

```
public class DemoVoertuigConstr{  
    // dit programma zal voertuigen aanmaken en gebruiken  
    public static void main(String[] args){  
        VoertuigConstr miniBus = new VoertuigConstr(7,80,6.0);  
        VoertuigConstr _2pk = new VoertuigConstr(4,30,10.0);  
        //bereken het aantal liter benzine nodig om 250 km af te leggen  
        System.out.println("De miniBus heeft " + miniBus.berekenBrandstof(250.0) +  
            " liter brandstof nodig om 250 kilometers af te leggen.");  
        System.out.println("Het _2pk'tje heeft " + _2pk.berekenBrandstof(250.0) +  
            " liter nodig om 250 kilometers af te leggen. ");  
    }  
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of the Java program. The first line shows "De miniBus heeft 41.67 liter brandstof nodig om 250 kilometers af te leggen." followed by a line break. The second line shows "Het _2pk'tje heeft 25.0 liter nodig om 250 kilometers af te leggen." followed by a line break. The third line shows "Press any key to continue . . .".

```
C:\Windows\system32\cmd.exe  
De miniBus heeft 41.67 liter brandstof nodig om 250 kilometers af te leggen.  
Het _2pk'tje heeft 25.0 liter nodig om 250 kilometers af te leggen.  
Press any key to continue . . .
```