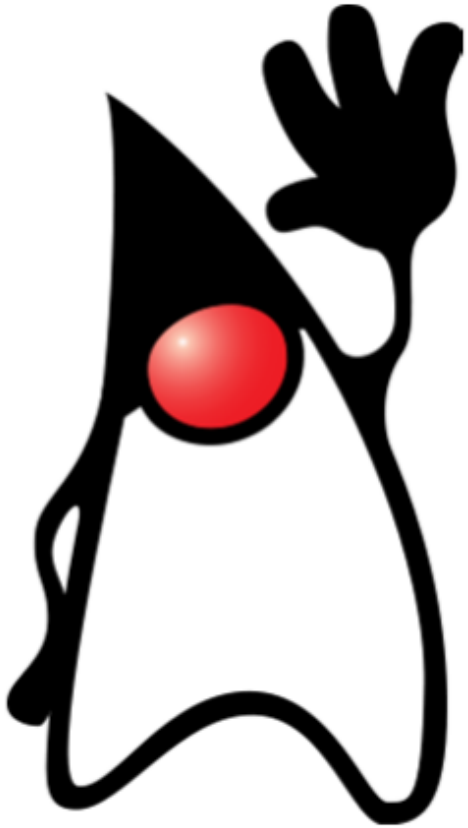
Programming Fundamentals

Code organiseren via klassen en methoden

Klasgroep	1EO-ICT
Opleiding	Bachelor Elektronica-ICT
Theorie	DI: 8:45 - 9:45
	WOE: 11:45 - 12:45
(Werk)Labo	DI: 9:45 - 12:45 + 13:30 - 14:30
	WOE: 13:30 - 15:30 + 15:30 - 17:30
Docent	Katja Verbeeck
Contact	katja.verbeeck@odisee.be



We kunnen niet al onze code in de main methode blijven schrijven!

- Deze zou veel te lang worden en onoverzichtelijk!
- Bovendien kunnen we nu wel via lussen code laten herhalen, maar het zou ook goed zijn dat we onze eigen stukken code kunnen hergebruiken wanneer nodig (zoals we doen met code uit de API).
- Dit kunnen we doen door functies (procedures) te maken en deze aan te roepen in de main methode. In een functionele / procedurele talen noemen we dit inderdaad functies of procedures. In een OO taal spreekt men van methoden.
- Net zoals data kan een methode statisch gemaakt worden via het keyword static (zoals de main methode). Dit doe je echter maar in heel specifieke gevallen.

Voorbeeld : een kleine RekenMachine

Schrijf een Java applicatie die 2 gehele getallen vraagt en een gehele operator en vervolgens de bewerking uitvoert. Wanneer de gebruiker een foutieve operator ingeeft blijf je vragen totdat er een correcte invoer gebeurt. Wanneer het resultaat op het scherm verschijnt kan de gebruiker aangeven dat hij nogmaals een bewerking wil ingeven door 'j' (van ja) in te typen. Blijf het uitvoeren van bewerkingen herhalen totdat de gebruiker een ander karakter dan 'j' intypt.

Voorbeeld : klasse RekenMachine

```
public static void main(String[] args) {
  boolean stop = false;
  Scanner scan = new Scanner(System.in);

  do { // getallen opvragen
    System.out.println("Geef een eerste getal:");
    int getal1 = scan.nextInt();
    System.out.println("Geef een tweede getal:");
    int getal2 = scan.nextInt();

    // bewerking opvragen
    System.out.print("Welke bewerking wil je
        uitvoeren? ");
    char operator = scan.next().charAt(0);
    while (operator != '+'&& operator != '-'
          && operator != '*'&& operator != '/'
          && operator != '%') {
        System.out.println("Gelieve een echte
```

```
        System.out.println("Gelieve een echte
            bewerking in te geven ! ");
        operator = scan.next().charAt(0);
    }

     // resultaat berekenen
    int res = -1;
    switch (operator) {
        case '+': res = getal1 + getal2; break;
        case '-': res = getal1 - getal2; break;
        case '*': res = getal1 * getal2; break;
        case '/': res = getal1 / getal2; break;
        case '%': res = getal1 % getal2; break;
        default: break;
    }
```

```
        // resultaat naar het scherm schrijven en vragen
            of er verder gerekend moet worden
        System.out.println(getal1 + " " + operator + " "
            + getal2 + " = " + res);
        System.out.println("Nog eentje? Druk op j : ");
        stop = ('j' != scan.next().charAt(0));
    } while (!stop);
    System.out.println("Einde");
    }
}
```

Gebruik methoden !

eigenlijk wil je de main methode verkorten als volgt :

```
public static void main(String[] args) {
    boolean stop = false;
    Scanner scan = new Scanner(System.in);

    do {
        int getal1 = vraagGetal("Geef een eerste getal: ");
        int getal2 = vraagGetal("Geef een tweede getal: ");
        char operator = vraagBewerking("Welke bewerking wil
            je uitvoeren? ");
        int resultaat = doeBewerking(getal1, getal2,
            operator);
        System.out.println(getal1 + " " + operator + " " +
            getal2 + " = " + resultaat);
        System.out.println("Nog eentje? Druk op j : ");
        stop = ('j' != scan.next().charAt(0));
    } while (!stop);
    System.out.println("Einde");
}
```

vraagGetal methode

Als ze rechtstreeks wordt aangeroepen in de main methode moet ze **static** zijn !

```
public static int vraagGetal(String vraag) {
    Scanner scan = new Scanner(System.in);
    System.out.print(vraag);
    int getal = scan.nextInt();
    return getal;
}
```

vraagBewerking methode

ook hier **static**

```
public static char vraagBewerking(String vraag) {
    Scanner scan = new Scanner(System.in);
    System.out.print(vraag);
    char operator = scan.next().charAt(0);
    while (operator != '+' && operator != '-'
           && operator != '*' && operator != '/'
           && operator != '%') {
        System.out.println("Gelieve een echte
            bewerking in te geven ! ");
        operator = scan.next().charAt(0);
    }
    return operator;
}
```

doeBewerking methode

ook hier **static**

```
public static int doeBewerking(int term1, int term2,
    char operatie) {
    int res = -1;
    switch (operatie) {
      case '+': res = term1 + term2; break;
      case '-': res = term1 - term2; break;
      case '*': res = term1 * term2; break;
      case '/': res = term1 / term2; break;
      case '%': res = term1 % term2; break;
      default: break;
   }
   return res;
}
```

Methoden

- 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
- Splits in nuttige deelfunctionaliteiten als de methode te lang wordt !
- Wanneer de methode geen resultaat teruggeeft dan geef je **void** terug.

Algemene vorm van een methode

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

Definitie van een methode

```
returnType methodeNaam (parameters)
```

Methodedefinities uit de API

	Definities uit de	klasse Math	en String
modifier	returntype	naam	input types
public static	int	**round**	(float f)
public static	double	**ceil**	(double d)
public static	double	**floor**	(double d)
public	char	**charAt**	(int i)
public	boolean	**startsWith**	(String s)
public	String	**toUpperCase**	()
public	String	**toLowerCase**	()
public	String	**substring**	(int i)
public	String	**substring**	(int i, int j)

Statische methoden

- deze worden vooraf gegaan door het keyword static
- dit keyword zegt eigenlijk : "ik behoor tot de klasse en ben dus niet afhankelijk van dynamische of veranderlijke data-velden"
- Je hoeft dan ook geen object aan te maken bij een methode aanroep, je roept ze aan rechtstreeks via de klassenaam of wanneer ze in dezelfde klasse gedefinieerd zijn gewoon via de naam. Dus hier geeft dit : *RekenMachine.vraagBewerking()* maar omdat de main zich zelf ook in de klasse RekenMachine bevindt is gewoon *vraagBewerking*() ook goed!

Je kan niet al je methoden static maken !

- De main methode behoort meestal niet dat dezelfde klasse van je methoden. Een Java programma bestaat typisch uit meerdere klassen en die hoeven niet allemaal een main methode te bevatten.
- Een main methode is niet vereist in elke klasse! Je programma moet wel minstens 1 main methode bevatten, vermits dit het startpunt van je programma is.
- Meestal schrijf je een aparte klasse voor je main methode. Deze bevat typisch geen data, maar alleen de main methode en eventueel enkele statische hulpmethoden om de code overzichtelijk te houden.
- Het doel van methoden is om datavelden te kunnen manipuleren. Dat kan niet als je methode static is. De meeste methoden zijn dan ook objectmethoden en dus niet statisch !

Algemene vorm van een klasse

Een main methode mag maar hoeft niet opgenomen te worden!

```
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
   }
   ...
}
```

Objectmethoden

- Objectmethoden zijn methoden die de data van je objecten kan manipuleren.
- Vaak is het ook de enige manier om (veilig) toegang te verlenen tot die data.

RekenMachine met data en objectmethoden

```
public class RekenMachineObject {

    int getal1; // datavelden
    int getal2;
    char operator;

    public void stelDataIn(int get1, int get2){
        getal1 = get1;
        getal2 = get2;
    }

    public void stelOperatorIn(char op) {
        operator = op;
    }
```

vervolg

De methode doeBewerking heeft nu geen parameters nodig want deze ziet de datavelden

```
    public int doeBewerking() {
        int res = -1;
        switch (operator) {
            case '+': res = getal1 + getal2; break;
            case '-': res = getal1 - getal2; break;
            case '*': res = getal1 * getal2; break;
            case '/': res = getal1 / getal2; break;
            case '%': res = getal1 % getal2; break;
            default: break;
        }
        return res;
    }
    public String geefBewerking(){
        return getal1 + " " + operator + " " + getal2 + " = ";
    }
}
```

Klasse RekenMachineTest

De main methode kan nu perfect in een andere klasse staan !

```
class RekenMachineTest {
    public static void main(String[] args) {
        boolean stop = false;
        Scanner scan = new Scanner(System.in);
        RekenMachineObject eenMachine = new RekenMachineObject();

        do {
            int get1 = vraagGetal("Geef een eerste getal: "); // statische
                 methoden
            int get2 = vraagGetal("Geef een tweede getal: ");
            char op = vraagBewerking("Welke bewerking wil je uitvoeren? ");

            eenMachine.stelDataIn(get1, get2); // objectmethoden
            eenMachine.stelOperatorIn(op);
            int resultaat = eenMachine.doeBewerking();
            System.out.println(eenMachine.geefBewerking() + resultaat);

            System.out.println("Nog eentje? Druk op j : ");
            stop = ('j' != scan.next().charAt(0));
        } while (!stop);
        System.out.println("Einde");
    }
```

vervolg Klasse RekenMachineTest

samen met enkele statische methode die input/output verzorgen voor de main

```
    public static int vraagGetal(String vraag) {
        Scanner scan = new Scanner(System.in);
        System.out.print(vraag);
        int getal = scan.nextInt();
        return getal;
    }

    public static char vraagBewerking(String vraag) {
        Scanner scan = new Scanner(System.in);
        System.out.print(vraag);
        char operator = scan.next().charAt(0);
        while (operator != '+' && operator != '-'
                && operator != '*' && operator != '/'
                && operator != '%') {
            System.out.println("Gelieve een echte bewerking in te geven ! ");
            operator = scan.next().charAt(0);
        }
        return operator;
    }
}
```

Oproepen van statische versus objectmethoden

statische methodeoproepen

```
double resultaat;
resultaat = Math.pow(2,8);

int resultaat =
    Integer.parseInt("123");

String resultaat =
    Integer.toString(123);
```

objectmethodeoproepen

```
String obj = new
    String("hallo");

int resultaat;
resultaat =
    obj.indexOf('x');

char resultaat;
resultaat = obj.charAt(2);

String resultaat;
resultaat =
    obj.substring(1,4);
```

Actuele versus formele parameters

uit de klasse Math

```
// methodedefinitie uit de API
public static double pow(double d1, double d2)

//oproep:
double macht = Math.pow(2,4);
```

de formele parameters : **d1, d2**
de actuele parameters = de waarden die d1 en d2 krijgen bij aanroep = 2 en 4

Actuele versus formele parameters

uit de klasse String

```
//methodedefinitie uit de API
public String substring(int i, int j)

//oproep:
String s = "Kermit";
String sub = s.substring(2,4);
```

de formele parameters : **i, j**
de actuele parameters = de waarden die i en j krijgen bij aanroep = 2 en 4

Actuele versus formele parameters

uit de klasse Math

```
//methodedefinitie uit de API
public static double random()

//oproep:
double d = Math.random();
```

de formele parameters : geen
de actuele parameters = geen want deze methode heeft geen parameters

Actuele versus formele parameters

uit de klasse Integer

```
//methodedefinitie uit de API
public static int parseInt(String s)

//oproep
String tekst = "123";
int i = Integer.parseInt(tekst);
```

de formele parameters : **s**
de actuele parameters = de waarde die s krijgt bij aanroep = "123"

Methode Overloading

Er zijn meerdere methoden met dezelfde naam, hoe kan dit?
$\rightarrow$ Een methode met dezelfde naam maar met een andere parameterlijst is voor de compiler gewoon een andere methode, m.a.w. een methode wordt niet gedefinieerd door haar naam alleen dus!

Vandaar de vele methoden in de API met dezelfde naam. Bijvoorbeeld in de klasse String vind je 4 methoden met de naam *indexOf* :

- public int indexOf(String str)
- public int indexOf(String str, int fromIndex)
- public int indexOf(int ch)
- public int indexOf(int ch, int fromIndex)

Methode overloading

Welke zijn geldige overloads van de methode :
public int berekenSom(int par1, int par2)

Goede Voorbeelden

- public int berekenSom(int par1, int par2, int par3)
- public double berekenSom(double par1, double par2)
- public double berekenSom(int par1, double par2)

Foute Voorbeelden

- public double berekenSom(int par1, int par2)
- public int berekenSom(int arg1, int arg2)

Meer of minder parameters + parameters van een ander type zijn toegelaten.
Je kan geen ander returntype teruggeven als de parameterlijst dezelfde blijft !
De naam van de parameters veranderen doet er niet toe, enkel het type is van belang!

Documenteren van methoden

javadoc

Javadoc is een Java programma dat toelaat de documentatie die je via tags annoteert bij je code omzet naar html pagina's in dezelfde stijl als de Java API.
De algemene tags : **@author** - **@version** voeg je toe bovenaan aan je programma.
De tags die je kan gebruiken om methodes te beschrijven zijn : **@param** - **@return** - **@see**. Deze plaats je vlak boven je methodedefinitie.
Let op, gebruik de juiste opbouw van de documentatie headers vb :

```
/**
* Voert een gegeven rekenkundige bewerking uit
* op term1 en term2
* @param operator De operator (+, -, *, / of %)
* @return Het resultaat van de berekening
*/
```

javadoc voor de RekenMachineTest

de statische methode vraagGetal

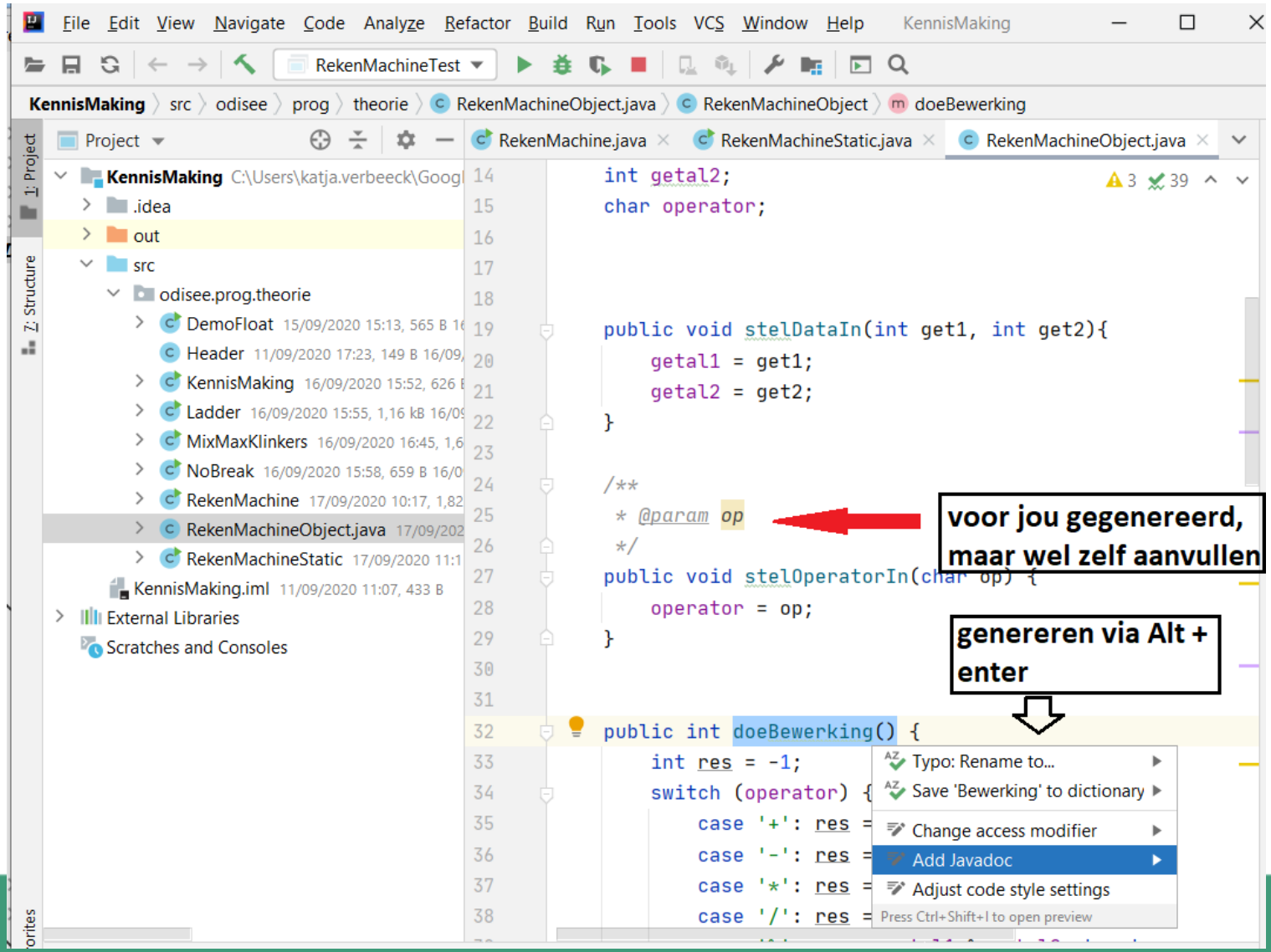
```
/**
* vraagGetal methode
* @param vraag : de vraag die moet uitgeprint worden
* @return het getal dat ingelezen wordt
* @see java.util.Scanner#nextInt()
*/

public static int vraagGetal(String vraag) {
  Scanner scan = new Scanner(System.in);
  ...
```

javadoc voor de RekenMachineTest

de statische methode vraagBewerking

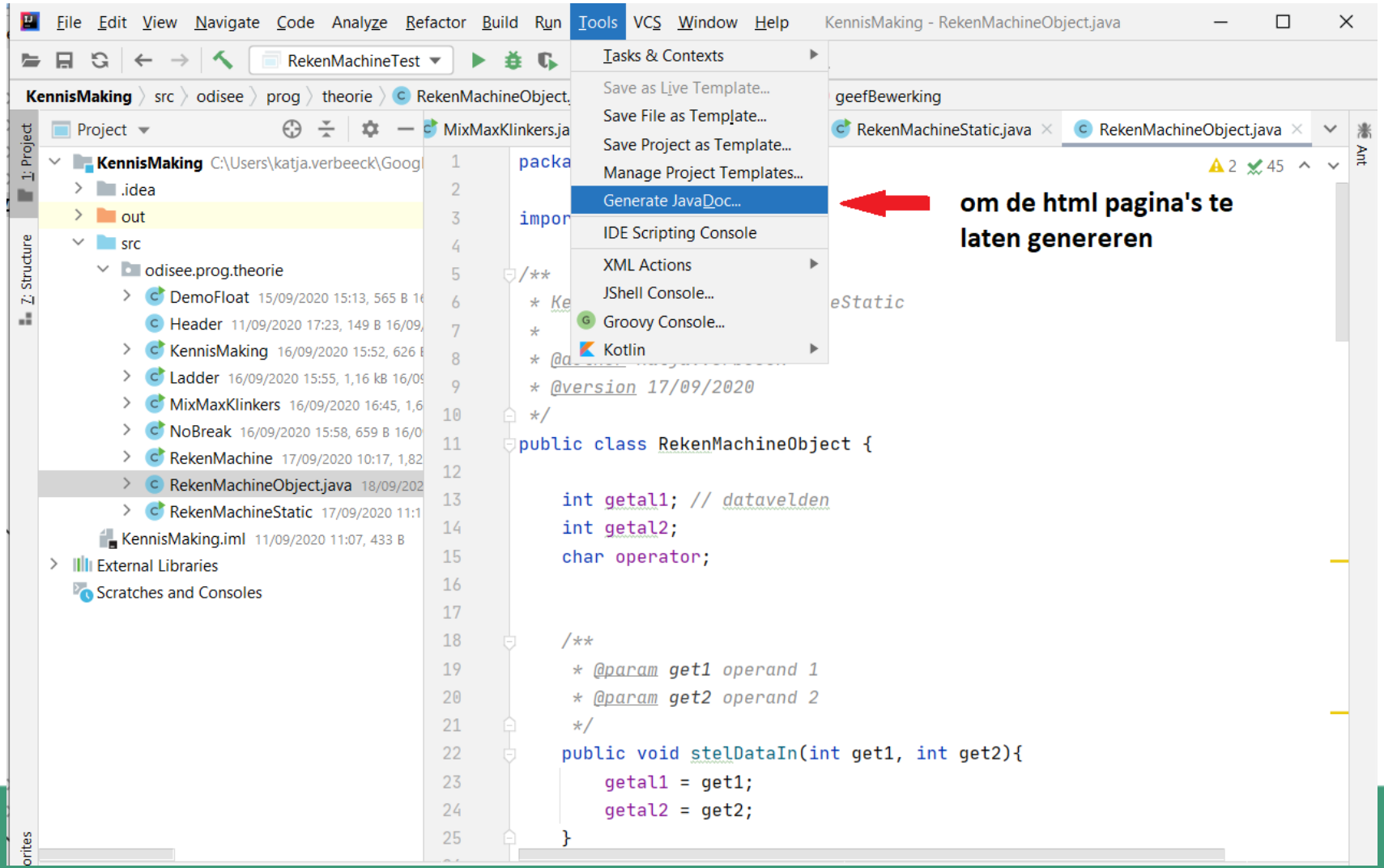
```
/**
* vraagBewerking methode
* @param vraag : de tekst die moet uitgeprint worden
* @return het ingelezen karakter
* @see String#charAt(int)
*/
public static char vraagBewerking(String vraag) {
  Scanner scan = new Scanner(System.in);
  ...
```

javadoc in Intellij



javadoc in Intellij

Via tools → Generate Javadoc



javadoc in Intellij