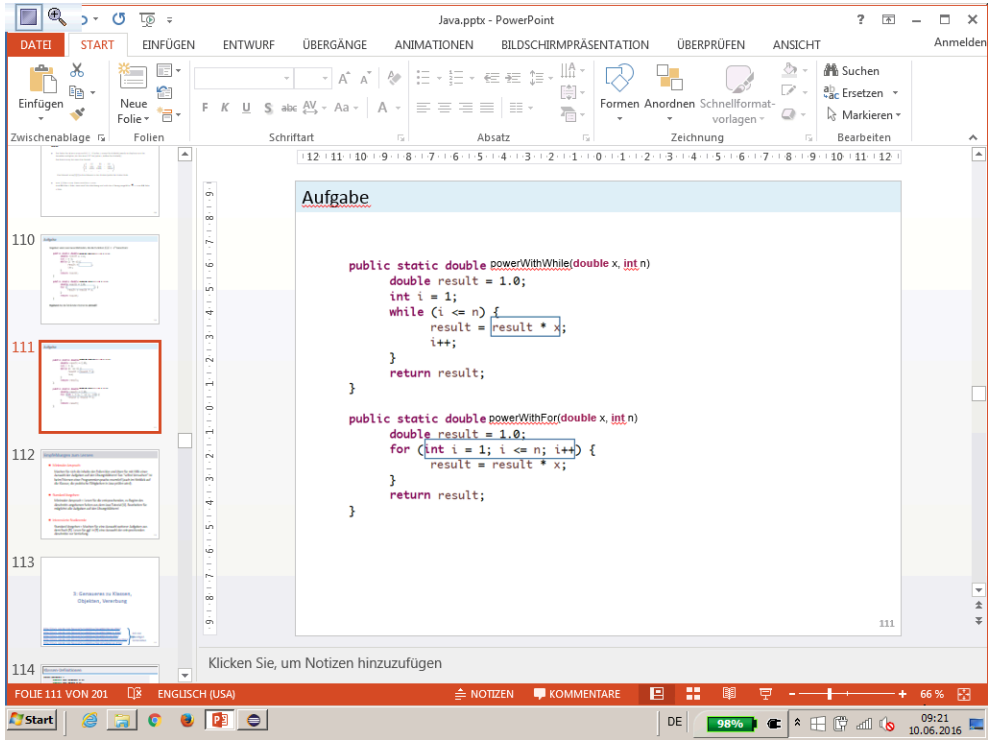
Script **generated by TTT**

Title: groh: profile1 (10.06.2016)

Date: Fri Jun 10 09:21:31 CEST 2016

Duration: 84:58 min

Pages: 34



Klassen-Definitionen

```
class Bicycle {
       public int cadence = 0;
       public int speed = 0;
       public int gear = 1;

       public Bicycle(int startCadence, int startSpeed, int startGear) {
            gear = startGear;
            cadence = startCadence;
            speed = startSpeed;
       }

       public void changeCadence(int newValue) {
            cadence = newValue;
       }

       public void chang
            gear = newVa
       }

       public void speed
            speed = spee
       }

       public void apply
            speed = spee
       }
}
```

```
public class MountainBike extends Bicycle {
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear)
    {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```


Klassen-Definitionen

```
class Bicycle {
       public int cadence = 0;
       public int speed = 0;
       public int gear = 1;

       public Bicycle(int startCadence, int startSpeed, int startGear) {
            gear = startGear;
            cadence = startCadence;
            speed = startSpeed;
       }

       public void changeCadence(int newValue) {
            cadence = newValue;
       }

       public void chang
            gear = newVa
       }

       public void speed
            speed = spee
       }

       public void apply
            speed = spee
       }
}
```

```
public class MountainBike extends Bicycle {
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear)
    {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

Klassen-Definitionen – Generelle Form

Klassen-Definitionen – Generelle Form:

```
modifier class MyClass extends MySuperClass
                implements YourInterface1, ...,
                           YourInterfaceN
{
    // Attribute, Konstruktoren, Methoden
}
```

- wobei (Access) *modifier* : Bestimmte Kombinationen von { `public, final, abstract` } und {protected, private, static} (aber nur für innere Klassen)
- wobei Namen der Klasse, Oberklasse und der Interfaces frei wählbar

```
class Bicycle {
    public int cadence = 0;
    public int speed = 0;
    public int gear = 1;

    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public void changeCadence(int newValue) {
        cadence = newValue;
    }

    public void changeGear(int newValue) {
        gear = newValue;
    }

    public void speedUp(int increment) {
        speed = speed + increment;
    }

    public void applyBrake(int decrement) {
        speed = speed - decrement;
    }
}
```

```
public class MountainBike extends Bicycle {
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear)
    {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

115

Klassen-Definitionen – Generelle Form

Klassen-Definitionen – Generelle Form:

```
modifier class MyClass extends MySuperClass
                implements YourInterface1, ...,
                           YourInterfaceN
{
    // Attribute, Konstruktoren, Methoden
}
```

- wobei (Access) *modifier* : Bestimmte Kombinationen von { `public, final, abstract` } und {protected, private, static} (aber nur für innere Klassen)
- wobei Namen der Klasse, Oberklasse und der Interfaces frei wählbar

```
class Bicycle {
    public int cadence = 0;
    public int speed = 0;
    public int gear = 1;

    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public void changeCadence(int newValue) {
        cadence = newValue;
    }

    public void changeGear(int newValue) {
        gear = newValue;
    }

    public void speedUp(int increment) {
        speed = speed + increment;
    }

    public void applyBrake(int decrement) {
        speed = speed - decrement;
    }
}
```

```
public class MountainBike extends Bicycle {
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear)
    {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

115

Klassen-Definitionen – Generelle Form

Klassen-Definitionen – Generelle Form:

```
modifier class MyClass extends MySuperClass
                implements YourInterface1, ...,
                           YourInterfaceN
{
    // Attribute, Konstruktoren, Methoden
}
```

- wobei (Access) *modifier* : Bestimmte Kombinationen von { `public, final, abstract` } und {protected, private, static} (aber nur für innere Klassen)
- wobei Namen der Klasse, Oberklasse und der Interfaces frei wählbar

```
class Bicycle {
    public int cadence = 0;
    public int speed = 0;
    public int gear = 1;

    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public void changeCadence(int newValue) {
        cadence = newValue;
    }

    public void changeGear(int newValue) {
        gear = newValue;
    }

    public void speedUp(int increment) {
        speed = speed + increment;
    }

    public void applyBrake(int decrement) {
        speed = speed - decrement;
    }
}
```

```
public class MountainBike extends Bicycle {
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear)
    {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

115

Definition von Konstruktoren

Konstruktoren Deklaration – Generelle Form:

```
modifier MyClass ( parameters )
throwsClauses {
    statements
}
```

- wobei (Access) *modifier* : Bestimmte Kombinationen von { `public, protected, private`}
- *parameters*: *(kommt gleich)*
- *throwsClauses*: *(kommt bei Exceptions)*

```
class Bicycle {
    public int cadence = 0;
    public int speed = 0;
    public int gear = 1;

    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public void changeCadence(int newValue) {
        cadence = newValue;
    }

    public void changeGear(int newValue) {
        gear = newValue;
    }

    public void speedUp(int increment) {
        speed = speed + increment;
    }

    public void applyBrake(int decrement) {
        speed = speed - decrement;
    }
}
```

```
public class MountainBike extends Bicycle {
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear)
    {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

118

Konstruktoren (Constructors)

- können vollständige und konsistente Initialisierung der Objekte sicherstellen
- (auch generell für Methoden): Mehrere Varianten anbieten --> verschiedene Grade an Details für verschiedene Nutzer der Klasse (<---> Abstraktion (API), Information Hiding)
- Unterklassen-Konstruktoren: Zugriff auf Oberklassen-Konstruktor mit **super** und Erweiterung nach Bedarf

```
class Bicycle {
  int cadence;
  int speed;
  int gear;

  Bicycle(int c, int s, int g) {
    cadence = c;
    speed = s;
    gear = g;
  }

  Bicycle(int g) {
    cadence = 0;
    speed = 0;
    gear = g;
  }
}
```

```
class Tandem extends Bicycle {
  int numberOfDrivers;

  Tandem(int c, int s, int g, int n) {
    super(c, s, g);
    numberOfDrivers = n;
  }
}
```


Mini-Aufgabe – http://pingo.upb.de

```
public class SomeClass {

        public int someAttribute;
        public int someOtherAttribute;

        public SomeClass(int initialValue){
                System.out.println("first constructor called");
                someAttribute = initialValue;
        }

        public SomeClass(int initialValue, int initialValueOther){
                System.out.println("second constructor called");
                someAttribute = initialValue;
                someOtherAttribute = initialValueOther;
        }

}
```

```
public class TestClass {

        public static void main(String[] args) {
                SomeClass someObject = new SomeClass(7);
                int x = someObject.someAttribute;
                System.out.println(x);
                int y = someObject.someOtherAttribute;
                System.out.println(y);
        }

}
```

Was ist die Ausgabe von main?

A	first constructor called 7 0
B	second constructor called 7 7
C	first constructor called 7 undefined
D	second constructor called 7 0

Methodenaufruf und Parameter-Übergabe

Übergabe einer Liste von Parametern an Methoden / Konstruktoren

```
int doSomething(int primitiveParameter1,
                SomeClass referenceParameter)
{
       int someInt = 17 + 9;
       primitiveParameter1 = 0;
       referenceParameter = null;
       return someInt;
}
```


Methodenaufruf und Parameter-Übergabe

Übergabe einer Liste von Parametern an Methoden / Konstruktoren

```
int doSomething(int primitiveParameter1,
                SomeClass referenceParameter)
{
       int someInt = 17 + 9;
       primitiveParameter1 = 0;
       referenceParameter = null;
       return someInt;
}
```

Methodenaufruf und Parameter-Übergabe

Übergabe einer Liste von Parametern an Methoden / Konstruktoren

```
int doSomething(int primitiveParameter1,
                SomeClass referenceParameter)
{
       int someInt = 17 + 9;
       primitiveParameter1 = 0;
       referenceParameter = null;
       return someInt;
}
```

Übergabe von Parametern primitiven Typs: durch Kopieren des Werts (call by value)

```
int x = 1;
SomeClass someObject = new SomeClass();
int y = doSomething(x, someObject);
// after this statement, x still has value 1.
```


Methodenaufruf und Parameter-Übergabe

Übergabe einer Liste von Parametern an Methoden / Konstruktoren

```
int doSomething(int primitiveParameter1,
                SomeClass referenceParameter)
{
       int someInt = 17 + 9;
       primitiveParameter1 = 0;
       referenceParameter = null;
       return someInt;
}
```

Übergabe von Parametern primitiven Typs: durch Kopieren des Werts (call by value)

```
int x = 1;
SomeClass someObject = new SomeClass();
int y = doSomething(x, someObject);
// after this statement, x still has value 1.
```

127

Methodenaufruf und Parameter-Übergabe

Übergabe einer Liste von Parametern an Methoden / Konstruktoren

```
int doSomething(int primitiveParameter1,
                SomeClass referenceParameter)
{
       int someInt = 17 + 9;
       primitiveParameter1 = 0;
       referenceParameter = null;
       return someInt;
}
```

Übergabe von Parametern primitiven Typs: durch Kopieren des Werts (call by value)

```
int x = 1;
SomeClass someObject = new SomeClass();
int y = doSomething(x, someObject);
// after this statement, x still has value 1.
```

127

Methodenaufruf und Parameter-Übergabe

Übergabe einer Liste von Parametern an Methoden / Konstruktoren

```
int doSomething(int primitiveParameter1,
                SomeClass referenceParameter)
{
       int someInt = 17 + 9;
       primitiveParameter1 = 0;
       referenceParameter = null;
       return someInt;
}
```

Übergabe von Parametern primitiven Typs: durch Kopieren des Werts (call by value)

```
int x = 1;
SomeClass someObject = new SomeClass();
int y = doSomething(x, someObject);
// after this statement, x still has value 1.
```

127

Methodenaufruf und Parameter-Übergabe

Warum ist das so?

- *Erinnerung*: Variablen von Referenztyp zeigen auf ein Objekt dieses Typs (dieser Klasse) = Wert der Variablen ist diese Referenz
- Call by Value heisst: Kopien der Werte der Variablen werden übergeben. Diese Kopien können beliebig manipuliert werden. Sobald die Methode endet, werden die Kopien zerstört.
- Über die kopierten Referenzen auf Objekte, die ausserhalb der Methode weiterexistieren, können diese Objekte dennoch zugegriffen werden („.“-Operator) und bei Bedarf dauerhaft verändert werden. (Seiteneffekt)

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...		...
1149	**someObject**	<1150>
1150	someObject.xxx	23
1151	someObject.yyy	0
1152	someObject.zzz	7
...	...	...
5327	**referenceParameter**	<1150>
5328	...	...
5329	...	...

Methodenaufruf und Parameter-Übergabe

Warum ist das so?

- *Erinnerung*: Variablen von Referenztyp zeigen auf ein Objekt dieses Typs (dieser Klasse) = Wert der Variablen ist diese Referenz
- Call by Value heisst: Kopien der Werte der Variablen werden übergeben. Diese Kopien können beliebig manipuliert werden. Sobald die Methode endet, werden die Kopien zerstört.
- Über die kopierten Referenzen auf Objekte, die ausserhalb der Methode weiterexistieren, können diese Objekte dennoch zugegriffen werden („.“-Operator) und bei Bedarf dauerhaft verändert werden. (Seiteneffekt)

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...		...
1149	**someObject**	<1150>
1150	someObject.xxx	23
1151	someObject.yyy	0
1152	someObject.zzz	7
...	...	...
5327	**referenceParameter**	<1150>
5328	...	...
5329	...	...

129

Methodenaufruf und Parameter-Übergabe

Warum ist das so?

- *Erinnerung*: Variablen von Referenztyp zeigen auf ein Objekt dieses Typs (dieser Klasse) = Wert der Variablen ist diese Referenz
- Call by Value heisst: Kopien der Werte der Variablen werden übergeben. Diese Kopien können beliebig manipuliert werden. Sobald die Methode endet, werden die Kopien zerstört.
- Über die kopierten Referenzen auf Objekte, die ausserhalb der Methode weiterexistieren, können diese Objekte dennoch zugegriffen werden („.“-Operator) und bei Bedarf dauerhaft verändert werden. (Seiteneffekt)

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...		...
1149	**someObject**	<1150>
1150	someObject.xxx	23
1151	someObject.yyy	0
1152	someObject.zzz	7
...	...	...
5327	**referenceParameter**	null
5328	...	...
5329	...	...

poof!

130

Rückgabewerte von Methoden – `return`

- Methoden können Werte zurückgeben.
- Typ des Rückgabewertes muss in der Methodendefinition angegeben werden.
- In der Methode veranlasst `return` *expression*`;` die Rückgabe des Werts der *expression* und das Verlassen der Methode.
- Wenn Methode nichts zurück geben soll: Rückgabetyp ist `void`

```
long faculty(int n) {
  long result = 1;
  for (int i = 2; i <= n; i++) {
    result = result * i;
  }
  return result;
}
```

```
// Somewhere else...
long x = faculty(5);
System.out.println("Faculty of 5 is " + x + ".");
```

131

Rückgabewerte von Methoden – `return`

- Wenn Rückgabe Referenz auf Objekt (oder Array), das in Methode erzeugt wurde --> Objekt/Array existiert natürlich auch nach Beendigung der Methode weiter.

```
Bicycle goGetABike() {
  Bicycle someBike;
  if (checkForSufficientFunds()) {
    someBike = new Bicycle();
    return someBike;
  } else {
    return null;
  }
}


// Call the method from somewhere else...
Bicycle bike = goGetABike();
...
// go on and use the bike ....
```


Rückgabewerte von Methoden – `return`

- Wenn Rückgabe Referenz auf Objekt (oder Array), das in Methode erzeugt wurde --> Objekt/Array existiert natürlich auch nach Beendigung der Methode weiter.

```
Bicycle goGetABike() {
  Bicycle someBike;
  if (checkForSufficientFunds()) {
    someBike = new Bicycle();
    return someBike;
  } else {
    return null;
  }
}


// Call the method from somewhere else...
Bicycle bike = goGetABike();
...
// go on and use the bike ....
```

132

Aufrufen von Methoden; Referenz `this`

- Methoden können über den dot-"."-Operator mit Bezug auf die jeweiligen Objekte „auf diesen Objekten" (←→ Bezug auf Attribute des Objekts (←→ Sichtbarkeit)) aufgerufen werden.
- Schreibersparnis: Code der Methoden kann andere Methoden und Attribute des Objekts ohne Extra-Angabe des Bezugs aufrufen / referenzieren. Wenn doch nötig, können Objekte mit der Referenz `this` Bezug auf sich selbst nehmen.

```
public class Bicycle {
  public int cadence = 0;

  public void addToCadence(int amount) {
    cadence = cadence + amount  // also: this.cadence = this.cadence + amount;
  }

  public void someOtherMethod() {
    addToCadence(5);            // also: this.addToCadence(5)
  }
}
```

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();

bike1.addToCadence(10);   // bike1.cadence == 10;
bike2.addToCadence(9);    // bike2.cadence == 9;
bike1.someOtherMethod();  // bike1.cadence == 15;
```


Aufrufen von Methoden; Referenz `this`

- Methoden können über den dot-"."-Operator mit Bezug auf die jeweiligen Objekte „auf diesen Objekten" (←→ Bezug auf Attribute des Objekts (←→ Sichtbarkeit)) aufgerufen werden.
- Schreibersparnis: Code der Methoden kann andere Methoden und Attribute des Objekts ohne Extra-Angabe des Bezugs aufrufen / referenzieren. Wenn doch nötig, können Objekte mit der Referenz `this` Bezug auf sich selbst nehmen.

```
public class Bicycle {
  public int cadence = 0;

  public void addToCadence(int amount) {
    cadence = cadence + amount  // also: this.cadence = this.cadence + amount;
  }

  public void someOtherMethod() {
    addToCadence(5);            // also: this.addToCadence(5)
  }
}
```

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();

bike1.addToCadence(10);   // bike1.cadence == 10;
bike2.addToCadence(9);    // bike2.cadence == 9;
bike1.someOtherMethod();  // bike1.cadence == 15;
```

133

Aufrufen von Methoden; Referenz `this`

- Methoden können über den dot-"."-Operator mit Bezug auf die jeweiligen Objekte „auf diesen Objekten" (←→ Bezug auf Attribute des Objekts (←→ Sichtbarkeit)) aufgerufen werden.
- Schreibersparnis: Code der Methoden kann andere Methoden und Attribute des Objekts ohne Extra-Angabe des Bezugs aufrufen / referenzieren. Wenn doch nötig, können Objekte mit der Referenz `this` Bezug auf sich selbst nehmen.

```
public class Bicycle {
  public int cadence = 0;

  public void addToCadence(int amount) {
    cadence = cadence + amount  // also: this.cadence = this.cadence + amount;
  }

  public void someOtherMethod() {
    addToCadence(5);            // also: this.addToCadence(5)
  }
}
```

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();

bike1.addToCadence(10);   // bike1.cadence == 10;
bike2.addToCadence(9);    // bike2.cadence == 9;
bike1.someOtherMethod();  // bike1.cadence == 15;
```


Aufrufen von Methoden; Referenz `this`

- Methoden können über den dot-"."-Operator mit Bezug auf die jeweiligen Objekte „auf diesen Objekten" (←→ Bezug auf Attribute des Objekts (←→ Sichtbarkeit)) aufgerufen werden.
- Schreibersparnis: Code der Methoden kann andere Methoden und Attribute des Objekts ohne Extra-Angabe des Bezugs aufrufen / referenzieren. Wenn doch nötig, können Objekte mit der Referenz `this` Bezug auf sich selbst nehmen.

```
public class Bicycle {
  public int cadence = 0;

  public void addToCadence(int amount) {
    cadence = cadence + amount  // also: this.cadence = this.cadence + amount;
  }

  public void someOtherMethod() {
    addToCadence(5);            // also: this.addToCadence(5)
  }
}
```

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();

bike1.addToCadence(10);   // bike1.cadence == 10;
bike2.addToCadence(9);    // bike2.cadence == 9;
bike1.someOtherMethod();  // bike1.cadence == 15;
```

133

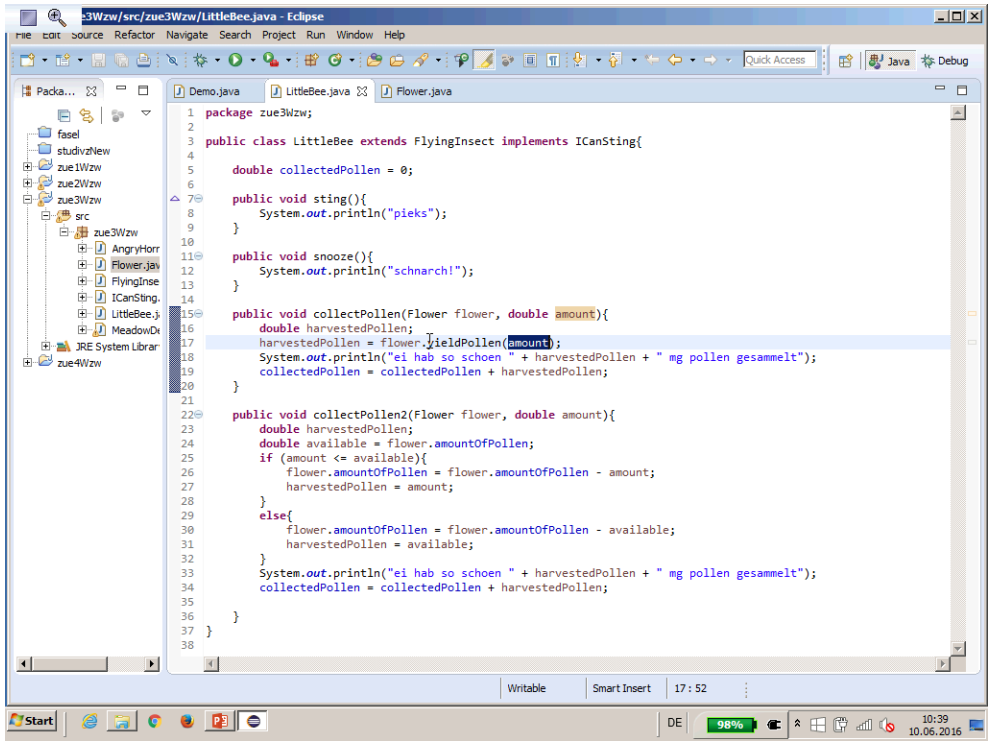
Aufrufen von Methoden; Referenz `this`

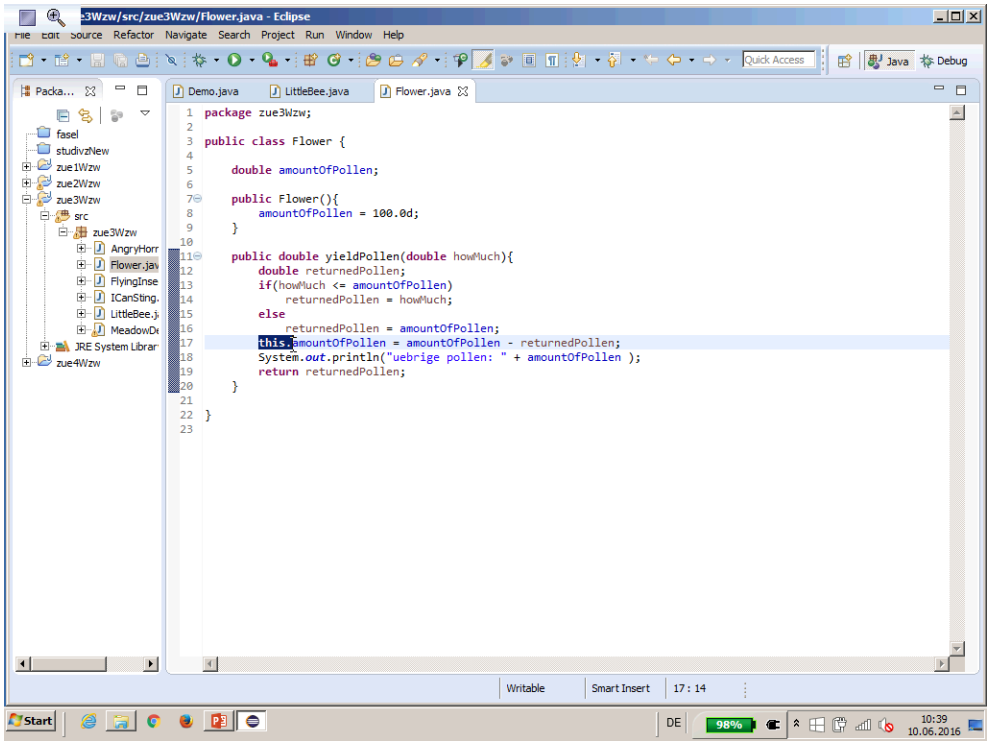
- Methoden können über den dot-"."-Operator mit Bezug auf die jeweiligen Objekte „auf diesen Objekten" (←→ Bezug auf Attribute des Objekts (←→ Sichtbarkeit)) aufgerufen werden.
- Schreibersparnis: Code der Methoden kann andere Methoden und Attribute des Objekts ohne Extra-Angabe des Bezugs aufrufen / referenzieren. Wenn doch nötig, können Objekte mit der Referenz `this` Bezug auf sich selbst nehmen.

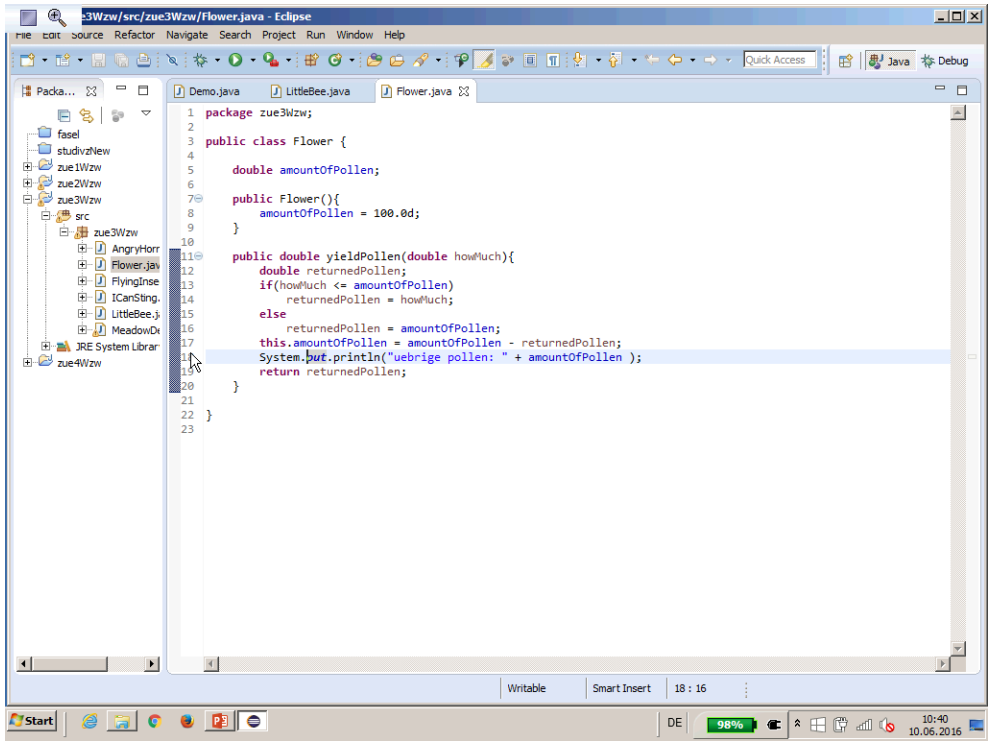
```
public class Bicycle {
  public int cadence = 0;

  public void addToCadence(int amount) {
    cadence = cadence + amount  // also: this.cadence = this.cadence + amount;
  }

  public void someOtherMethod() {
    addToCadence(5);            // also: this.addToCadence(5)
  }
}
```

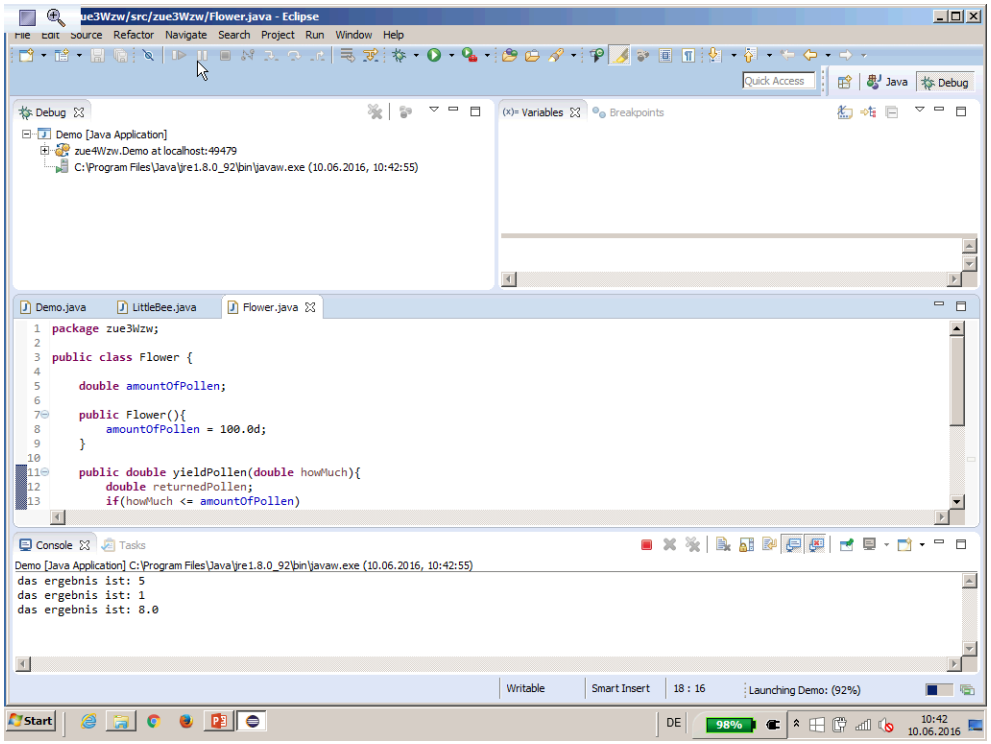
```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();

bike1.addToCadence(10);   // bike1.cadence == 10;
bike2.addToCadence(9);    // bike2.cadence == 9;
bike1.someOtherMethod();  // bike1.cadence == 15;
```

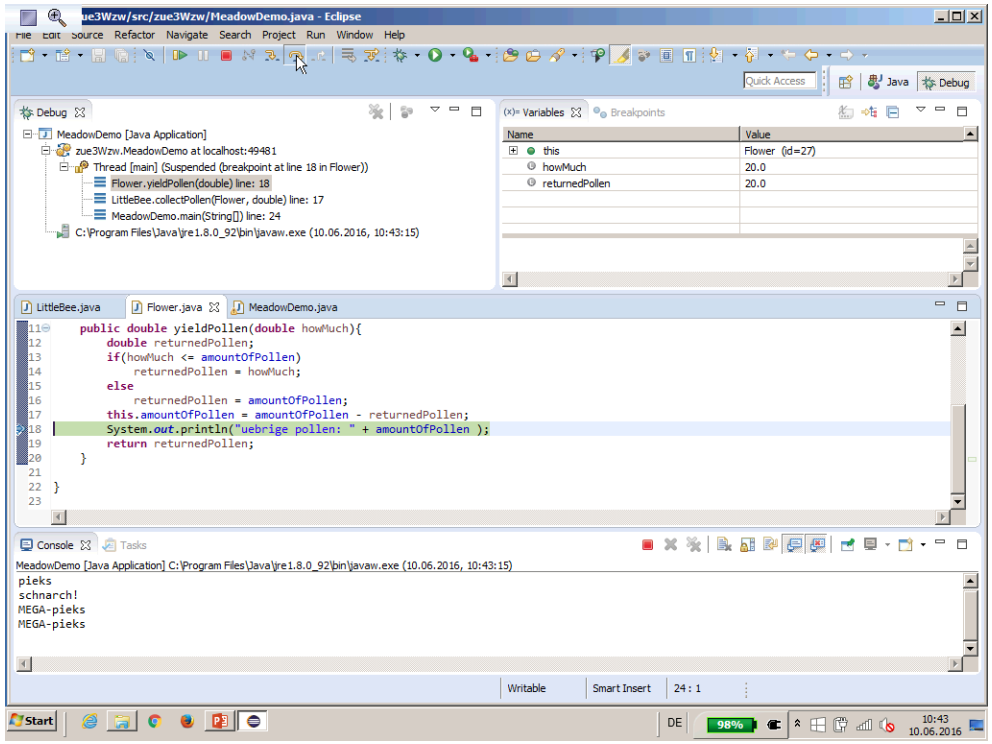
133

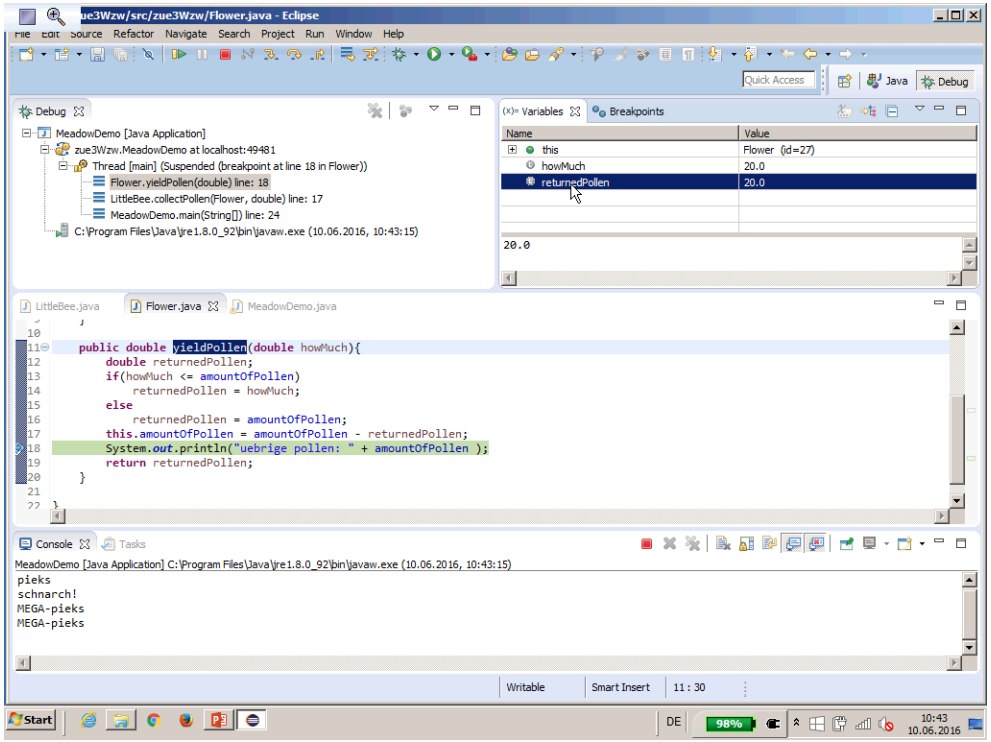


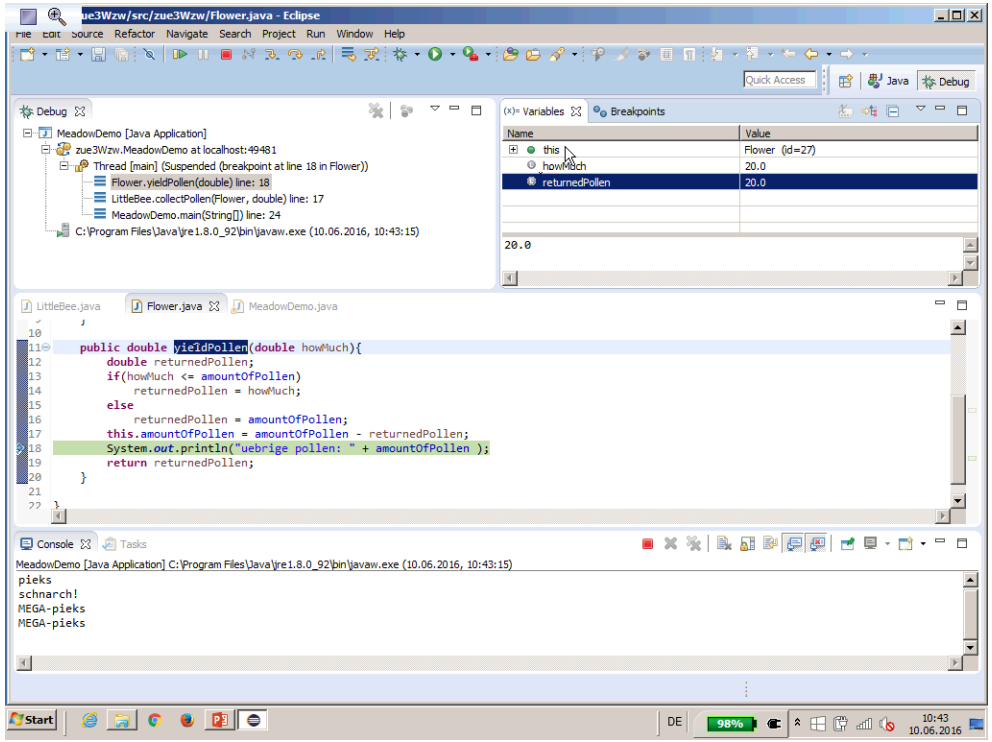

zue3Wzw/src/zue3Wzw/Flower.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Packa...
Demo.java
LittleBee.java
Flower.java
fasel
studivzNew
zue1Wzw
zue2Wzw
zue3Wzw
src
zue3Wzw
AngryHorr
Flower.jav
FlyingInse
ICanSting.
LittleBee.j
MeadowDe
JRE System Librar
zue4Wzw
1 package zue3Wzw;
2
3 public class Flower {
4
5 double amountOfPollen;
6
7 public Flower(){
8 amountOfPollen = 100.0d;
9 }
10
11 public double yieldPollen(double howMuch){
12 double returnedPollen;
13 if(howMuch <= amountOfPollen)
14 returnedPollen = howMuch;
15 else
16 returnedPollen = amountOfPollen;
17 this.amountOfPollen = amountOfPollen - returnedPollen;
18 System.out.println("uebrige pollen: " + amountOfPollen);
19 return returnedPollen;
20 }
21
22 }
23
Writable
Smart Insert
17 : 14
Start
DE
98%
10:39
10.06.2016

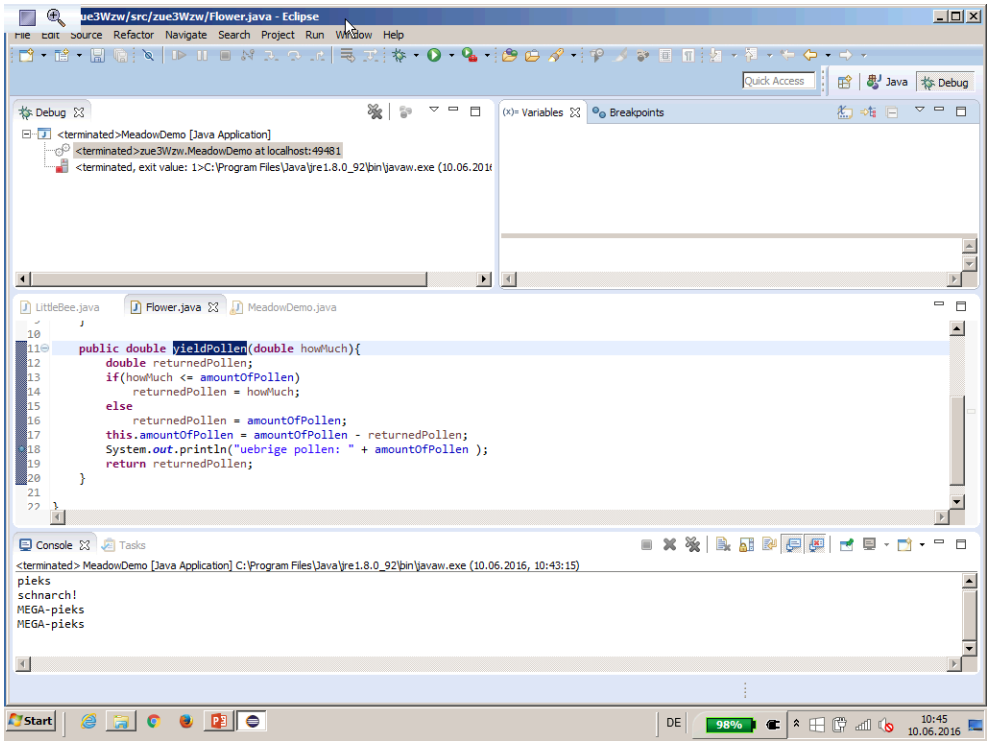

zue3Wzw/src/zue3Wzw/Flower.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Packa...
Demo.java
LittleBee.java
Flower.java
fasel
studivzNew
zue1Wzw
zue2Wzw
zue3Wzw
src
zue3Wzw
AngryHorr
Flower.jav
FlyingInse
ICanSting.
LittleBee.j
MeadowDe
JRE System Librar
zue4Wzw
1 package zue3Wzw;
2
3 public class Flower {
4
5 double amountOfPollen;
6
7 public Flower(){
8 amountOfPollen = 100.0d;
9 }
10
11 public double yieldPollen(double howMuch){
12 double returnedPollen;
13 if(howMuch <= amountOfPollen)
14 returnedPollen = howMuch;
15 else
16 returnedPollen = amountOfPollen;
17 this.amountOfPollen = amountOfPollen - returnedPollen;
18 System.out.println("uebrige pollen: " + amountOfPollen);
19 return returnedPollen;
20 }
21
22 }
23
Writable
Smart Insert
18 : 16
Start
DE
98%
10:40
10.06.2016


zue3Wzw/src/zue3Wzw/Flower.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
Demo [Java Application]
zue4Wzw.Demo at localhost:49479
C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:42:55)
Variables
Breakpoints
Demo.java
LittleBee.java
Flower.java
1 package zue3Wzw;
2
3 public class Flower {
4
5 double amountOfPollen;
6
7 public Flower(){
8 amountOfPollen = 100.0d;
9 }
10
11 public double yieldPollen(double howMuch){
12 double returnedPollen;
13 if(howMuch <= amountOfPollen)
Console
Tasks
Demo [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:42:55)
das ergebnis ist: 5
das ergebnis ist: 1
das ergebnis ist: 8.0
Writable
Smart Insert
18 : 16
Launching Demo: (92%)
Start
DE
98%
10:42
10.06.2016


ue3Wzw/src/zue3Wzw/MeadowDemo.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
MeadowDemo [Java Application]
zue3Wzw.MeadowDemo at localhost:49481
Thread [main] (Suspended (breakpoint at line 18 in Flower))
Flower.yieldPollen(double) line: 18
LittleBee.collectPollen(Flower, double) line: 17
MeadowDemo.main(String[]) line: 24
C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:43:15)
Variables
Breakpoints
Name
Value
this
Flower (id=27)
howMuch
20.0
returnedPollen
20.0
LittleBee.java
Flower.java
MeadowDemo.java
11 public double yieldPollen(double howMuch){
12 double returnedPollen;
13 if(howMuch <= amountOfPollen)
14 returnedPollen = howMuch;
15 else
16 returnedPollen = amountOfPollen;
17 this.amountOfPollen = amountOfPollen - returnedPollen;
18 System.out.println("uebrige pollen: " + amountOfPollen);
19 return returnedPollen;
20 }
21
22 }
23
Console
Tasks
MeadowDemo [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:43:15)
pieks
schnarch!
MEGA-pieks
MEGA-pieks
Writable
Smart Insert
24 : 1
Start
DE
98%
10:43
10.06.2016


ue3Wzw/src/zue3Wzw/Flower.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
MeadowDemo [Java Application]
zue3Wzw.MeadowDemo at localhost:49481
Thread [main] (Suspended (breakpoint at line 18 in Flower))
Flower.yieldPollen(double) line: 18
LittleBee.collectPollen(Flower, double) line: 17
MeadowDemo.main(String[]) line: 24
C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:43:15)
Variables
Breakpoints
Name
Value
this
Flower (id=27)
howMuch
20.0
returnedPollen
20.0
20.0
LittleBee.java
Flower.java
MeadowDemo.java
10
11 public double yieldPollen(double howMuch){
12 double returnedPollen;
13 if(howMuch <= amountOfPollen)
14 returnedPollen = howMuch;
15 else
16 returnedPollen = amountOfPollen;
17 this.amountOfPollen = amountOfPollen - returnedPollen;
18 System.out.println("uebrige pollen: " + amountOfPollen);
19 return returnedPollen;
20 }
21
22 }
Console
Tasks
MeadowDemo [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:43:15)
pieks
schnarch!
MEGA-pieks
MEGA-pieks
Writable
Smart Insert
11 : 30
Start
DE
98%
10:43
10.06.2016


ue3Wzw/src/zue3Wzw/Flower.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
MeadowDemo [Java Application]
zue3Wzw.MeadowDemo at localhost:49481
Thread [main] (Suspended (breakpoint at line 18 in Flower))
Flower.yieldPollen(double) line: 18
LittleBee.collectPollen(Flower, double) line: 17
MeadowDemo.main(String[]) line: 24
C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:43:15)
Variables
Breakpoints
Name
Value
this
Flower (id=27)
howMuch
20.0
returnedPollen
20.0
20.0
LittleBee.java
Flower.java
MeadowDemo.java
10
11 public double yieldPollen(double howMuch){
12 double returnedPollen;
13 if(howMuch <= amountOfPollen)
14 returnedPollen = howMuch;
15 else
16 returnedPollen = amountOfPollen;
17 this.amountOfPollen = amountOfPollen - returnedPollen;
18 System.out.println("uebrige pollen: " + amountOfPollen);
19 return returnedPollen;
20 }
21
22 }
Console
Tasks
MeadowDemo [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:43:15)
pieks
schnarch!
MEGA-pieks
MEGA-pieks
Start
DE
98%
10:43
10.06.2016


ue3Wzw/src/zue3Wzw/Flower.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Debug
<terminated>MeadowDemo [Java Application]
<terminated>zue3Wzw.MeadowDemo at localhost:49481
<terminated, exit value: 1>C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.201
Variables
Breakpoints
LittleBee.java
Flower.java
MeadowDemo.java
10
11 public double yieldPollen(double howMuch){
12 double returnedPollen;
13 if(howMuch <= amountOfPollen)
14 returnedPollen = howMuch;
15 else
16 returnedPollen = amountOfPollen;
17 this.amountOfPollen = amountOfPollen - returnedPollen;
18 System.out.println("uebrige pollen: " + amountOfPollen);
19 return returnedPollen;
20 }
21
22 }
Console
Tasks
<terminated> MeadowDemo [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (10.06.2016, 10:43:15)
pieks
schnarch!
MEGA-pieks
MEGA-pieks
Start
DE
98%
10:45
10.06.2016