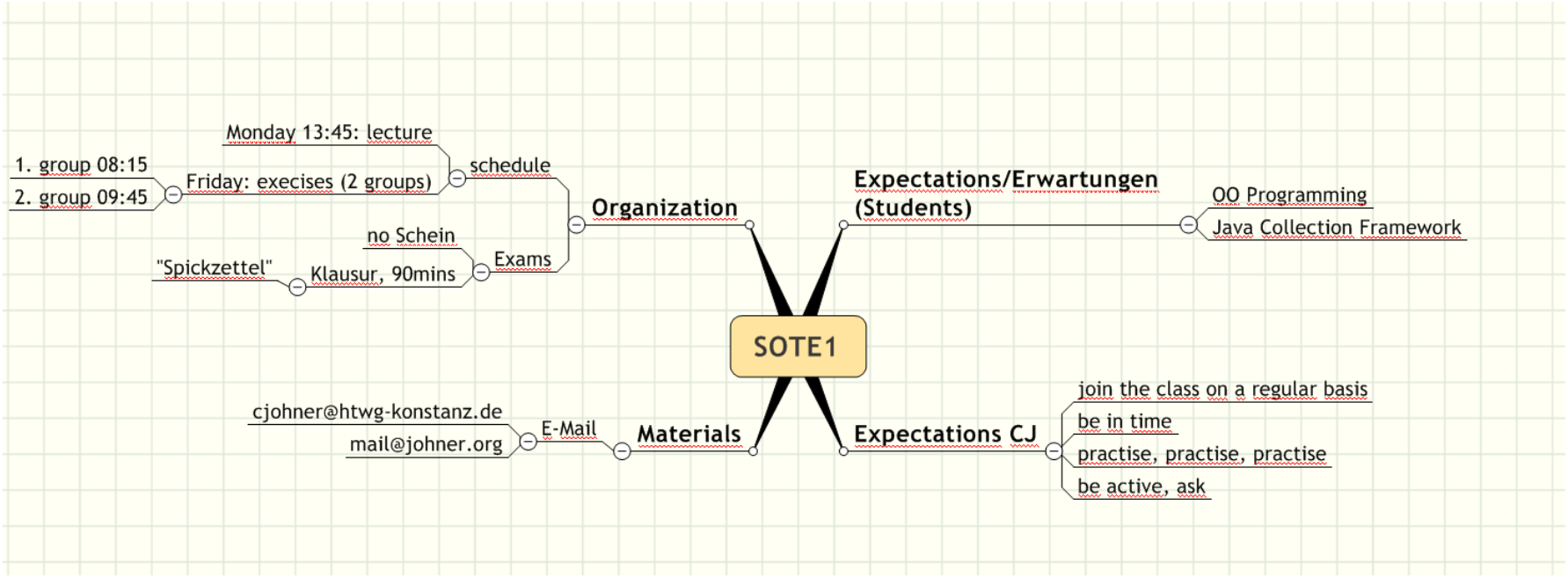
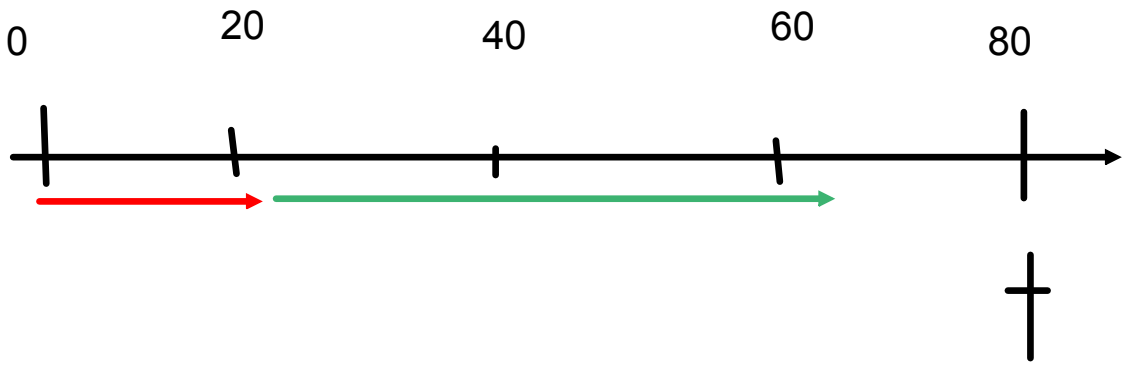

SOTE1
Organization
Monday 13:45: lecture
schedule
1. group 08:15
2. group 09:45
Friday: execises (2 groups)
Exams
no Schein
"Spickzettel"
Klausur, 90mins
Expectations/Erwartungen (Students)
OO Programming
Java Collection Framework
Materials
E-Mail
cjohner@htwg-konstanz.de
mail@johner.org
Expectations CJ
join the class on a regular basis
be in time
practise, practise, practise
be active, ask

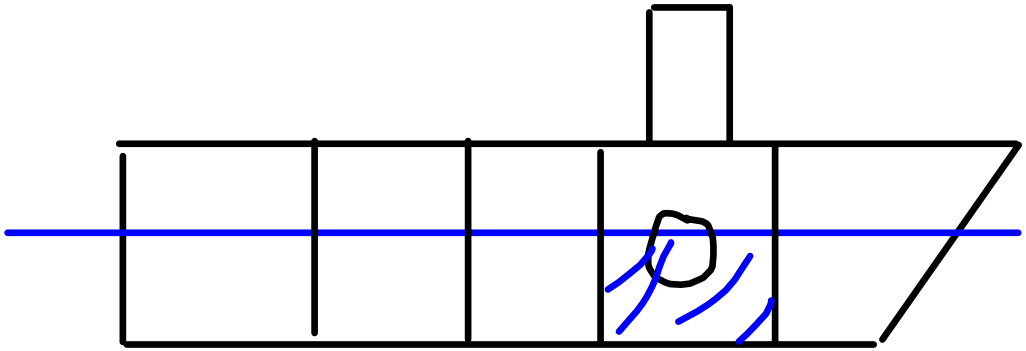

0
20
40
60
80

What's inside a Java class

- Package
- Imports
- class name (with extends, implements)
- Constructors
- Variable
  - > Instance Variables
  - > Class Variables (static)
  - > Constants (final)
- Methods
- inner classes

Packages: What are they good for?

- Better organization
- Encapsulation (private / protected methods only can be accesses from inside the class / package)
- Components can be re-used, better tested, easier to exchange,



Three Types of Inner Classes

1. Member Classes
2. Local inner classes
3. Anonymous inner classes

ad 1. Member classes

OuterClass.java

```
1  package innerClasses;
2
3  public class OuterClass {
4
5      public void doSomething(){
6          System.out.println("Our outer class does something");
7      }
8
9      public class InnerClass {
10         public void doSomething(){
11             System.out.println("Our inner class does something");
12         }
13
14         public class VeryInnerClass {
15             public void doSomething(){
16                 System.out.println("Our very inner class does something");
17             }
18
19         }
20     }
21 }
```

StartKlasse.java

```
1  package innerClasses;
2
3
4  public class StartKlasse {
5      public static void main(String[] args) {
6          OuterClass outer = new OuterClass();
7          outer.doSomething();
8
9          OuterClass.InnerClass innerClass = new OuterClass().new InnerClass();
10         innerClass.doSomething();
11
12         OuterClass.InnerClass innerClass2 = outer.new InnerClass();
13         innerClass2.doSomething();
14
15         OuterClass.InnerClass.VeryInnerClass veryInner = innerClass2.new VeryInnerClass();
16         veryInner.doSomething();
17
18     }
19 }
20
```

Problems Javadoc Declaration Console PIT Mutations Coverage

```
<terminated> StartKlasse (4) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (06.10.2017, 08:56:04)
Our outer class does something
Our inner class does something
Our inner class does something
Our very inner class does something
```

inner classes can be private or public

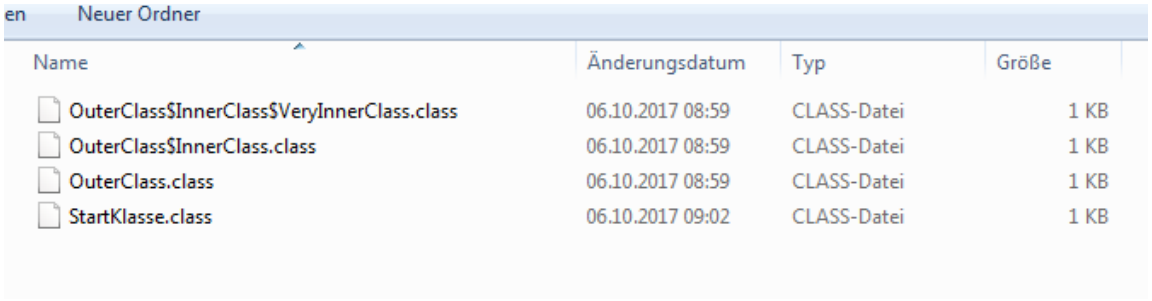
Instantiation of inner classes with ". new"

```java
package innerClasses;

public class OuterClass {

    public void doSomething(){
        System.out.println("Our outer class does something");
    }

    public class InnerClass {
        public void doSomething(){
            System.out.println("Our inner class does something");
            VeryInnerClass veryInner = new VeryInnerClass();
            veryInner.doSomething();
        }

        private class VeryInnerClass {
            public void doSomething(){
                System.out.println("Our very inner class does something");
            }

        }
    }
}
```

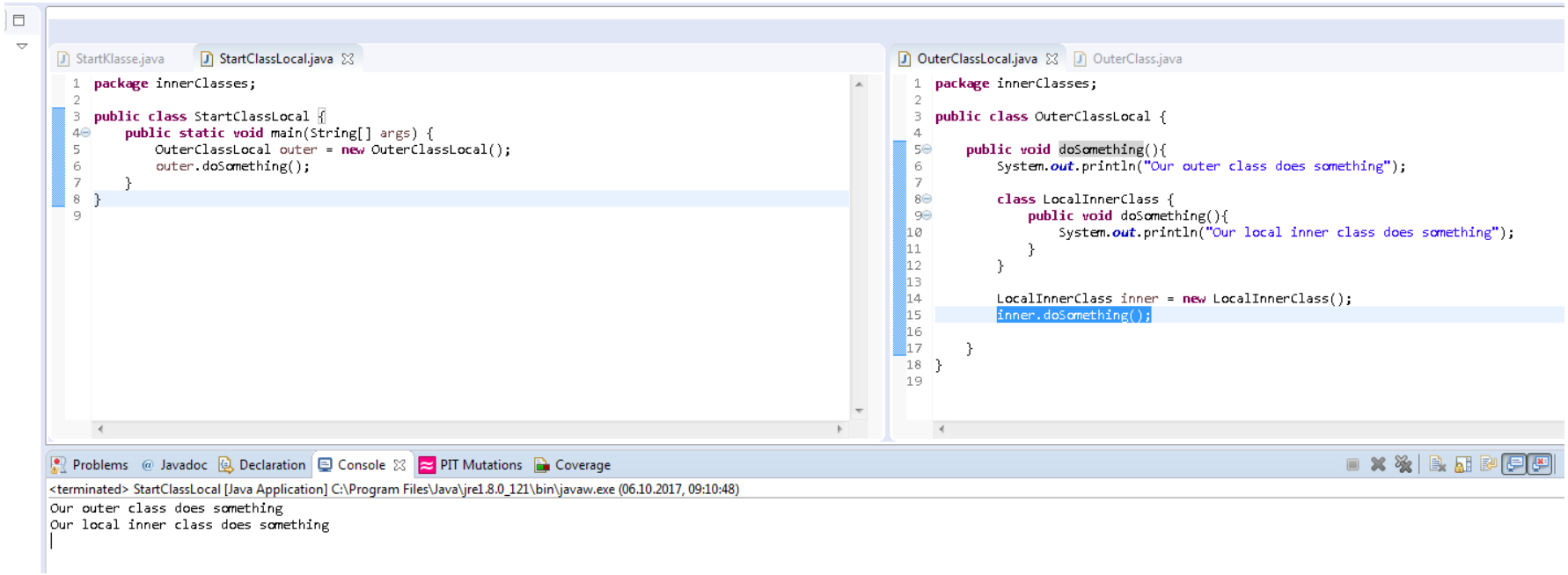
Instantiation of VeryInnerClass

private!

Class Files in the File System



ad 2. Local inner classes



Local inner classes are declared inside(!) of methods and can be used (instantiated) only there.

Local inner classes do not have any modifiers (private, public).

ad 3) Anonymous inner classes

```
JButton button = new JButton();
button.addEventListener(new EventListener() {
   //methods
});
```

Excurse Interfaces

Interfaces are a precondition fo component oriented programming. Interfaces represent contracts, that other classes sign by using the keyword "implements".

StartKlasse.ja... StartClassLoc... Moth

```
1  package schnittstellen;
2
3  public interface IGraphicsCard {
4
5      void display(String text);
6
7  }
```

Interface

StartKlasse.ja... StartClassLoc... Motherboard.ja... NVidia.java Al

```
1  package schnittstellen;
2
3  public class NVidia implements IGraphicsCard {
4
5      public void display(String text) {
6          System.out.println("Nvidia is displaying the text " + text);
7      }
8
9  }
10
```

Class that is implementing the interface ("signing the contract")

Motherboard.... NVidia.java IGraphicsCa... »3

```java
package schnittstellen;

public class Motherboard {
    private IGraphicsCard graphicsCard;

    public Motherboard(IGraphicsCard gc){
        graphicsCard = gc;
    }

    public void exececuteProgramm() {
        ///...

        graphicsCard.display("Hello");

    }

}
```

OuterClassLocal.java StartClass.java OuterClass.java

```java
package schnittstellen;

public class StartClass {
    public static void main(String[] args) {
        Motherboard motherboard = new Motherboard(new AMD());
        motherboard.exececuteProgramm();
    }
}
```

By using interfaces we were able to declare the class "Motherboard" completly independent from a specific implementation of a graphics card such as NVidia or AMD

This type of programming is call "Dependency Injection" (the dependency on a specific class is "injected" with the constructor).