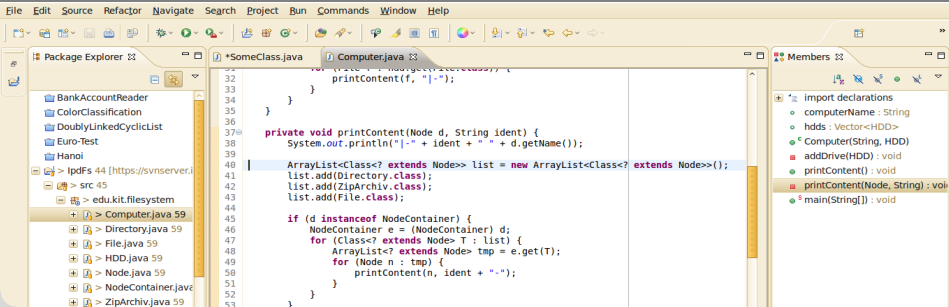


Programmieren-Tutorium Nr. 10 bei Martin Thoma

JUnit, Vererbung, toString(), Interfaces

Martin Thoma | 10. Dezember 2012

FAKULTÄT FÜR INFORMATIK



- 1 Einleitung
- 2 JUnit
- 3 Vererbung
- 4 toString()
- 5 Interfaces
- 6 Nachbesprechung ÜB 3
- 7 Abspann

Quiz: Vererbung

```
_____ Animal.java _____  
public class Animal {  
    private String sound;  
  
    public void roar() {  
        System.out.println(sound);  
    }  
}
```

```
_____ Jungle.java _____  
public class Jungle {  
    public static void main(String[] args) {  
        Animal tigger = new Tiger();  
        Animal felix = new Cat();  
        Cat ninja = new Cat();  
        Tiger diego = new Tiger();  
  
        tigger.roar();  
        felix.roar();  
        ninja.roar();  
        diego.roar();  
        diego.sound = "Hust hust";  
        diego.roar();  
    }  
}
```

```
_____ Tiger.java _____  
public class Tiger extends Animal {  
    String sound = "ROAR";  
}
```

```
_____ Cat.java _____  
public class Cat extends Animal {  
    public String sound;  
  
    public Cat() {  
        String sound = "Maunz";  
    }  
  
    @Override  
    public void roar() {  
        System.out.println("Cat:" + sound);  
    }  
}
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe?

Quiz: Antwort

```
Animal.java
public class Animal {
    private String sound;

    public void roar() {
        System.out.println(sound);
    }
}
```

```
Jungle.java
public class Jungle {
    public static void main(String[] args) {
        Animal tigger = new Tiger();
        Animal felix = new Cat();
        Cat ninja = new Cat();
        Tiger diego = new Tiger();

        tigger.roar();
        felix.roar();
        ninja.roar();
        diego.roar();
        diego.sound = "Hust hust";
        diego.roar();
    }
}
```

```
Tiger.java
public class Tiger extends Animal {
    String sound = "ROAR";
}
```

```
Cat.java
public class Cat extends Animal {
    public String sound;

    public Cat() {
        String sound = "Maunz";
    }

    @Override
    public void roar() {
        System.out.println("Cat:" + sound);
    }
}
```

- null
- Cat:null
- Cat:null
- null
- null

- Zeile 2 und 3: `sound` im Konstruktor von `Cat` ist eine lokale Variable, kein Attribut
- In Java werden nur Methoden vererbt
 - Klassen: Signatur und Implementierung
 - Interfaces: Nur Signatur

Mehr dazu später

JUnit ...

- ist ein Java-Paket
- ist ein Framework zum Testen von Java-Programmen
- ist SEHR verbreitet
- dient der Erstellung von Unit-Tests
- wurde von Erich Gamma und Kent Beck erstellt

JUnit ...

- ist ein Java-Paket
- ist ein Framework zum Testen von Java-Programmen
- ist SEHR verbreitet
- dient der Erstellung von Unit-Tests
- wurde von Erich Gamma und Kent Beck erstellt

JUnit ...

- ist ein Java-Paket
- ist ein Framework zum Testen von Java-Programmen
- ist SEHR verbreitet
- dient der Erstellung von Unit-Tests
- wurde von Erich Gamma und Kent Beck erstellt

JUnit ...

- ist ein Java-Paket
- ist ein Framework zum Testen von Java-Programmen
- ist SEHR verbreitet
- dient der Erstellung von Unit-Tests
- wurde von Erich Gamma und Kent Beck erstellt

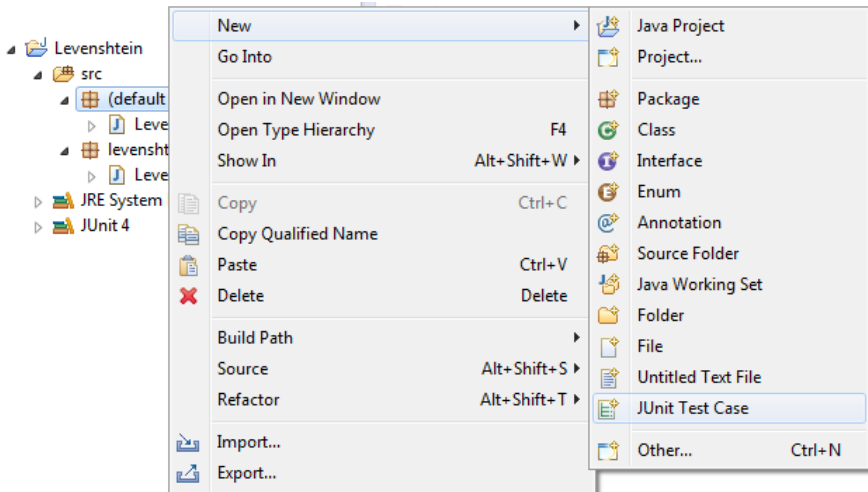
JUnit ...

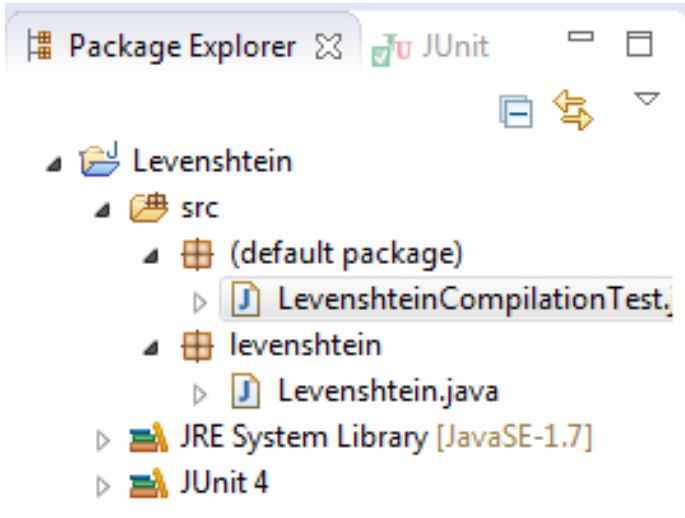
- ist ein Java-Paket
- ist ein Framework zum Testen von Java-Programmen
- ist SEHR verbreitet
- dient der Erstellung von Unit-Tests
- wurde von Erich Gamma und Kent Beck erstellt

LevenshteinCompilationTest.java

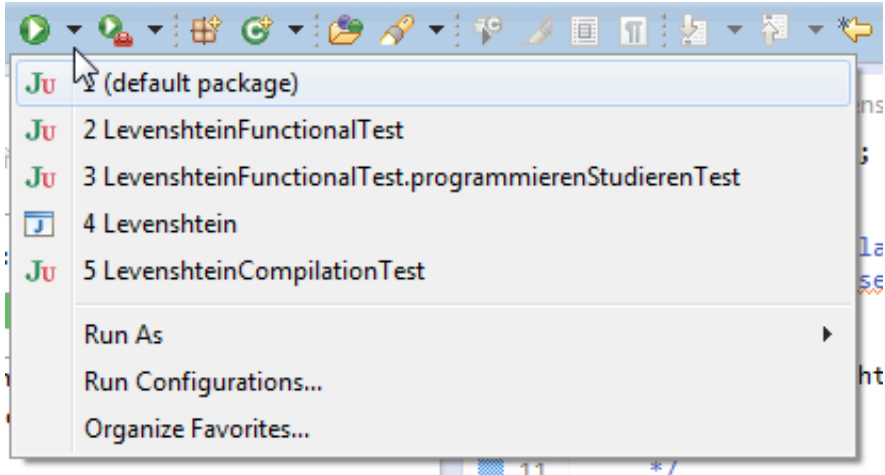
```
1 import levenshtein.Levenshtein;
2
3 import org.junit.Test;
4 import org.junit.runner.JUnitCore;
5 import static org.junit.Assert.assertEquals;
6
7 public class LevenshteinCompilationTest {
8
9     public LevenshteinCompilationTest() {
10    }
11
12    @Test(timeout = 10000)
13    public void runTest() {
14        Levenshtein levenshtein = new Levenshtein("nämlich", "dämlich");
15        assertEquals("Incorrect result comparing nämlich and dämlich", 1, levenshtein.getDistance());
16    }
17
18    public static void main(String[] args) throws Exception {
19        JUnitCore.main(LevenshteinCompilationTest.class.getName());
20    }
21 }
```

Eclipse: JUnit





Eclipse: JUnit



Package Explorer

JUnit

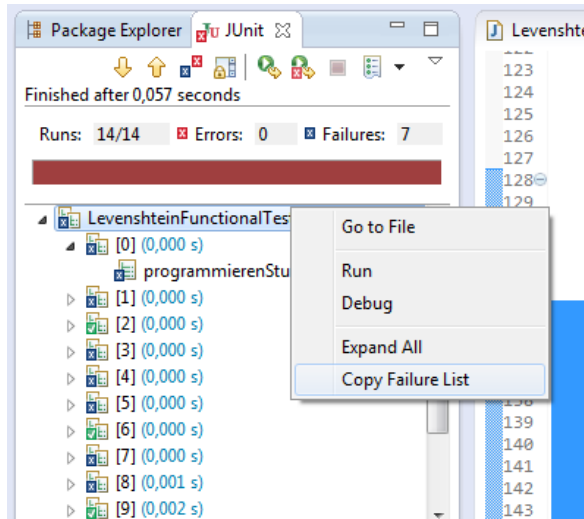


Finished after 0,047 seconds

Runs: 1/1 Errors: 0 Failures: 0



▶ LevenshteinCompilationTest [Runner: JUnit4TextRunner]



- ▷ [8] (0,000 s)
- ▷ [9] (0,002 s)
- ▷ [10] (0,003 s)

Failure Trace

```
java.lang.AssertionError: d(das gleiche, dasselbe) expected:<5> but was:<0>
at org.junit.Assert.fail(Assert.java:93)
at org.junit.Assert.failNotEquals(Assert.java:647)
at org.junit.Assert.assertEquals(Assert.java:128)
at org.junit.Assert.assertEquals(Assert.java:472)
at LevenshteinFunctionalTest.programmierenStudierenTest(LevenshteinFunctionalTest.java:10)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
at java.lang.reflect.Method.invoke(Unknown Source)
at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
```

Fehler

The import org.junit cannot be resolved

Lösung

- [Hier](#) junit-4.11.jar mit Hamcrest herunterladen
- Project » Properties » Java Build Path » Libraries » Add External JARs...
- junit-4.11.jar auswählen
- Auf OK klicken

Vererbung ...

- ist ein Schlüsselement der OOP
- ist in Java eingeschränkt: Eine Klasse erbt in Java von genau einer anderen Klasse
 - alle Klassen erben von `Object`
- dient der Spezialisierung

Vererbung . . .

- ist ein Schlüsselement der OOP
- ist in Java eingeschränkt: Eine Klasse erbt in Java von genau einer anderen Klasse
 - alle Klassen erben von **Object**
- dient der Spezialisierung

Vererbung . . .

- ist ein Schlüsselement der OOP
- ist in Java eingeschränkt: Eine Klasse erbt in Java von genau einer anderen Klasse
 - alle Klassen erben von **Object**
- dient der Spezialisierung

Vererbung . . .

- ist ein Schlüsselement der OOP
- ist in Java eingeschränkt: Eine Klasse erbt in Java von genau einer anderen Klasse
 - alle Klassen erben von **Object**
- dient der Spezialisierung

Wo kann Vererbung nützlich sein?

- Oberklasse Liste, Unterklassen SinglyLinkedList und DoubleLinkedList

- `contains()` ist gleich
- `append()` ist unterschiedlich
- `remove()` ist unterschiedlich

- Oberklasse Animal, Unterklassen Säugetier, Tiger, Schlange, Bär,

...

- Brettspiele:

- Klasse Spielbrett; Unterklassen: Schachbrett, Dame-Brett, Mensch-ärgere-dich-nicht
- Klasse Spielfigur; Unterklassen: Bauer, Dame, Springer, Turm
- Klasse Spiellogik; Unterklassen: DameLogik, SchachLogik

Wo kann Vererbung nützlich sein?

- Oberklasse Liste, Unterklassen SinglyLinkedList und DoubleLinkedList
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse Animal, Unterklassen Säugetier, Tiger, Schlange, Bär, ...
- Brettspiele:
 - Klasse Spielbrett; Unterklassen: Schachbrett, Dame-Brett, Mensch-ärgere-dich-nicht
 - Klasse Spielfigur; Unterklassen: Bauer, Dame, Springer, Turm
 - Klasse Spiellogik; Unterklassen: DameLogik, SchachLogik

Wo kann Vererbung nützlich sein?

- Oberklasse Liste, Unterklassen SinglyLinkedList und DoubleLinkedList
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse Animal, Unterklassen Säugetier, Tiger, Schlange, Bär, ...
- Brettspiele:
 - Klasse Spielbrett; Unterklassen: Schachbrett, Dame-Brett, Mensch-ärgere-dich-nicht
 - Klasse Spielfigur; Unterklassen: Bauer, Dame, Springer, Turm
 - Klasse Spiellogik; Unterklassen: DameLogik, SchachLogik

Wo kann Vererbung nützlich sein?

- Oberklasse `Liste`, Unterklassen `SinglyLinkedList` und `DoubleLinkedList`
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse `Animal`, Unterklassen `Säugetier`, `Tiger`, `Schlange`, `Bär`,
...
- Brettspiele:
 - Klasse `Spielbrett`; Unterklassen: `Schachbrett`, `Dame-Brett`,
`Mensch-ärgere-dich-nicht`
 - Klasse `Spielfigur`; Unterklassen: `Bauer`, `Dame`, `Springer`, `Turm`
 - Klasse `Spiellogik`; Unterklassen: `DameLogik`, `SchachLogik`

Wo kann Vererbung nützlich sein?

- Oberklasse `Liste`, Unterklassen `SinglyLinkedList` und `DoubleLinkedList`
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse `Animal`, Unterklassen `Säugetier`, `Tiger`, `Schlange`, `Bär`,
...
- Brettspiele:
 - Klasse `Spielbrett`; Unterklassen: `Schachbrett`, `Dame-Brett`, `Mensch-ärgere-dich-nicht`
 - Klasse `Spielfigur`; Unterklassen: `Bauer`, `Dame`, `Springer`, `Turm`
 - Klasse `Spiellogik`; Unterklassen: `DameLogik`, `SchachLogik`

Wo kann Vererbung nützlich sein?

- Oberklasse Liste, Unterklassen SinglyLinkedList und DoubleLinkedList
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse Animal, Unterklassen Säugetier, Tiger, Schlange, Bär, ...
- Brettspiele:
 - Klasse Spielbrett; Unterklassen: Schachbrett, Dame-Brett, Mensch-ärgere-dich-nicht
 - Klasse Spielfigur; Unterklassen: Bauer, Dame, Springer, Turm
 - Klasse Spiellogik; Unterklassen: DameLogik, SchachLogik

Wo kann Vererbung nützlich sein?

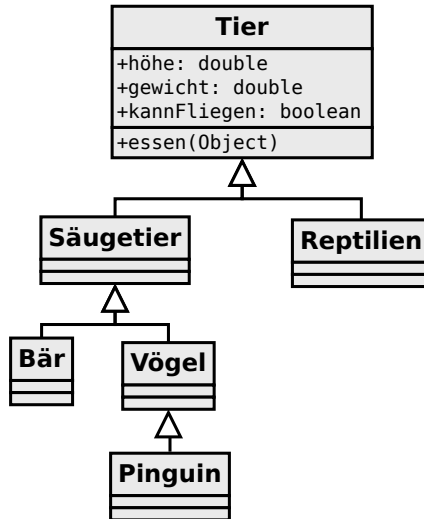
- Oberklasse Liste, Unterklassen SinglyLinkedList und DoubleLinkedList
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse Animal, Unterklassen Säugetier, Tiger, Schlange, Bär, ...
- Brettspiele:
 - Klasse Spielbrett; Unterklassen: Schachbrett, Dame-Brett, Mensch-ärgere-dich-nicht
 - Klasse Spielfigur; Unterklassen: Bauer, Dame, Springer, Turm
 - Klasse Spiellogik; Unterklassen: DameLogik, SchachLogik

Wo kann Vererbung nützlich sein?

- Oberklasse Liste, Unterklassen SinglyLinkedList und DoubleLinkedList
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse Animal, Unterklassen Säugetier, Tiger, Schlange, Bär, ...
- Brettspiele:
 - Klasse Spielbrett; Unterklassen: Schachbrett, Dame-Brett, Mensch-ärgere-dich-nicht
 - Klasse Spielfigur; Unterklassen: Bauer, Dame, Springer, Turm
 - Klasse Spiellogik; Unterklassen: DameLogik, SchachLogik

Wo kann Vererbung nützlich sein?

- Oberklasse Liste, Unterklassen SinglyLinkedList und DoubleLinkedList
 - `contains()` ist gleich
 - `append()` ist unterschiedlich
 - `remove()` ist unterschiedlich
- Oberklasse Animal, Unterklassen Säugetier, Tiger, Schlange, Bär, ...
- Brettspiele:
 - Klasse Spielbrett; Unterklassen: Schachbrett, Dame-Brett, Mensch-ärgere-dich-nicht
 - Klasse Spielfigur; Unterklassen: Bauer, Dame, Springer, Turm
 - Klasse Spiellogik; Unterklassen: DameLogik, SchachLogik



Vererbung: Beispiel in Java

```
_____ Animal.java _____  
public class Animal {  
    private String sound;  
  
    public void roar() {  
        System.out.println(sound);  
    }  
}
```

```
_____ Jungle.java _____  
public class Jungle {  
    public static void main(String[] args) {  
        Animal tigger = new Tiger();  
        Animal felix = new Cat();  
        Cat ninja = new Cat();  
        Tiger diego = new Tiger();  
  
        tigger.roar();  
        felix.roar();  
        ninja.roar();  
        diego.roar();  
        diego.sound = "Hust hust";  
        diego.roar();  
    }  
}
```

```
_____ Tiger.java _____  
public class Tiger extends Animal {  
    String sound = "ROAR";  
}
```

```
_____ Cat.java _____  
public class Cat extends Animal {  
    public String sound;  
  
    public Cat() {  
        String sound = "Maunz";  
    }  
  
    @Override  
    public void roar() {  
        System.out.println("Cat:" + sound);  
    }  
}
```

JLS 8.4.8

A class *C* inherits from its direct superclass and direct superinterfaces all abstract and non-abstract methods of the superclass and superinterfaces that are public, protected, or declared with default access in the same package as *C*, and are neither overridden (§8.4.8.1) nor hidden (§8.4.8.2) by a declaration in the class.

- Jedes Objekt hat eine Methode `toString()`
- Diese wird von `Object` vererbt
- und ~~kann~~ sollte überschrieben werden

- Jedes Objekt hat eine Methode `toString()`
- Diese wird von `Object` vererbt
- und ~~kann~~ sollte überschrieben werden

- Jedes Objekt hat eine Methode `toString()`
- Diese wird von `Object` vererbt
- und ~~kann~~ sollte überschrieben werden

Wie sollte toString() aussehen?

- Eine kurze textuelle Repräsentation des Objekts
- Soll von Menschen gelesen werden
- Per Standard:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Node.java

```
1 public class Node {
2     public Bike element;
3     public Node next;
4
5     public Node(Bike element) {
6         this.element = element;
7     }
8
9     @Override
10    public String toString() {
11        return "Node[element=" + element + "]";
12    }
13 }
```

- auf Deutsch: Schnittstelle
- es werden nur Methodensignaturen vererbt
- die Implementierung muss komplett selbst durchgeführt werden!
- wird wie Klassen in einer eigenen „MeinInterface.java“ Datei gespeichert

Namenskonvention

Der Name einer Schnittstelle endet oft mit -able.

- auf Deutsch: Schnittstelle
- es werden nur Methodensignaturen vererbt
- die Implementierung muss komplett selbst durchgeführt werden!
- wird wie Klassen in einer eigenen „MeinInterface.java“ Datei gespeichert

Namenskonvention

Der Name einer Schnittstelle endet oft mit -able.

- auf Deutsch: Schnittstelle
- es werden nur Methodensignaturen vererbt
- die Implementierung muss komplett selbst durchgeführt werden!
- wird wie Klassen in einer eigenen „MeinInterface.java“ Datei gespeichert

Namenskonvention

Der Name einer Schnittstelle endet oft mit -able.

- auf Deutsch: Schnittstelle
- es werden nur Methodensignaturen vererbt
- die Implementierung muss komplett selbst durchgeführt werden!
- wird wie Klassen in einer eigenen „MeinInterface.java“ Datei gespeichert

Namenskonvention

Der Name einer Schnittstelle endet oft mit -able.

- auf Deutsch: Schnittstelle
- es werden nur Methodensignaturen vererbt
- die Implementierung muss komplett selbst durchgeführt werden!
- wird wie Klassen in einer eigenen „MeinInterface.java“ Datei gespeichert

Namenskonvention


Der Name einer Schnittstelle endet oft mit -able.

Bicycle.java

```
1 interface Bicycle {  
2  
3     // wheel revolutions per minute  
4     void changeCadence(int newValue);  
5  
6     void changeGear(int newValue);  
7  
8     void speedUp(int increment);  
9  
10    void applyBrakes(int decrement);  
11 }
```

ACMEBicycle.java

```
1 class ACMEBicycle implements Bicycle {  
2     // remainder of this class  
3     // implemented as before  
4 }
```

- **Comparable**: Vergleichen mit 
- **List**: Viele Listenoperationen
- **Iterable**: foreach
- **Serializable**: Speichern / verschicken übers Netzwerk
- **Runnable**: Multithreading

- JLS 7
- Galileo openbook

Lösungen sind [hier](#) zu finden.

- `char charAt(int index)`: Returns the char value at the specified index.
- `public boolean matches(String regex)` Tells whether or not this string matches the given regular expression.
- `String substring(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.

Eclipse-Tipp

Wenn Eclipse euch im Projektordner einen Fehler anzeigt, aber keine Datei fehlerhaft ist, solltet ihr mal einen Blick in `Window` `»` `Show View` `»` `Problem` werfen.

- `char charAt(int index)` : Returns the char value at the specified index.
- `public boolean matches(String regex)` Tells whether or not this string matches the given regular expression.
- `String substring(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.

Eclipse-Tipp

Wenn Eclipse euch im Projektordner einen Fehler anzeigt, aber keine Datei fehlerhaft ist, solltet ihr mal einen Blick in `Window` » `Show View` » `Problem` werfen.

- `char charAt(int index)` : Returns the char value at the specified index.
- `public boolean matches(String regex)` Tells whether or not this string matches the given regular expression.
- `String substring(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.

Eclipse-Tipp

Wenn Eclipse euch im Projektordner einen Fehler anzeigt, aber keine Datei fehlerhaft ist, solltet ihr mal einen Blick in `Window` » `Show View` » `Problem` werfen.

- `char charAt(int index)` : Returns the char value at the specified index.
- `public boolean matches(String regex)` Tells whether or not this string matches the given regular expression.
- `String substring(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.

Eclipse-Tipp

Wenn Eclipse euch im Projektordner einen Fehler anzeigt, aber keine Datei fehlerhaft ist, solltet ihr mal einen Blick in Window Show View Problem werfen.

- 5. 17.12.2012: Generics?, Video „Library“ zeigen
 - 24.12.2012: Heiligabend - [Kein Tutorium](#)
 - 31.12.2012: Silvester - Kein Tutorium
- 4. 07.01.2013
- 3. 14.01.2013
- 2. 21.01.2013
- 1. 28.01.2013: Abschlussprüfunsvorbereitung
- 0. 04.02.2013: Abschlussprüfunsvorbereitung
 - 10.02.2013: Ende der Vorlesungszeit des WS 2012/2013 ([Quelle](#))

Beware of physicist fathers



Bildquelle: http://evelintanm.blogspot.de/2011_12_01_archive.html