

Juniorprogrammierer.de

Java 14: Interface & JUnit

2024/25 – Sascha Stojanovic

Agenda

- Keyword "abstract" again
- Interface I + II
- Exercises
- JUnit
- Example Unittest
- Exercise

Keyword "abstract" again

- You can only declare abstract methods in an abstract class
- **You cant create an object of an Abstract class**, so these classes are used to slimmen their child-classes
- Next to classes **you can also define abstract-methods**
 - **The have no content**
 - **All children need them but they have different content**
 - Child without parental abstract class definition shows and error

```
Inheritance > J Vehicle.java > ...  
3 public abstract class Vehicle {  
29 }  
30  
31 // Define an abstract method  
32 public abstract void startEngine();  
33 }  
34
```



```
1 package Inheritance;  
2  
3 public class Motorbike extends Vehicle{  
4  
5     public Motorbike(){  
6         super(numberWheels:2);  
7     }  
8  
9     public void honk(){  
10        super.honk();  
11        System.out.println(x:"Miiib miiib");  
12    }  
13  
14    public void startEngine(){  
15        System.out.println(x:"Start with kickstarter");  
16    }  
17 }
```

Interface I

- In modern Software engineering you usually don't work with parental abstract classes
- Instead you use interfaces which are automatically abstract classes and only consists of abstract methods
- Instead of the keyword “class” you implement the interface with a keyword “interface”
- Child classes don't use the keyword “extends” but “implements” to inherit the interfaces methods

```
// Interface
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
    public void sleep() {
        // The body of sleep() is provided here
        System.out.println("Zzz");
    }
}

class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

Interface II

- The implementing class must override all methods of the interface
- All attributes of an interface are public, static and final
- An interface cannot have a constructor
- Java does not support multiple inheritances but can implement several interfaces
 - Separate the interfaces with a “,”

Exercise

- Create a new Interface based on the known Vehicle.java in folder “IntExercise” called VehicleInterface.java
- Create in the same folder the Car.java and implement the corresponding interface

JUnit

- How to install: <https://code.visualstudio.com/docs/java/java-testing>
- Unittests are classbased and should test the class independent of other method involved in this class
- Unittests should have a positive and a negative Test
- Unittest should be coded according to “First”-Principle
 - F => Fast
 - I => Independent
 - R => Repeatable
 - S => Self-Validating
 - T => Timely

Example Unittest

The screenshot displays an IDE with two open Java files. The left pane shows the Explorer view with a folder named 'Unittests' containing 'SimpleCalculator.java' and 'SimpleCalculatorTest.java'. The right pane shows the code editor with the following content:

```
SimpleCalculator.java
1 package Unittests;
2
3 public class SimpleCalculator {
4
5     public String greet() {
6         return "Hello";
7     }
8
9     public int add (int a, int b) {
10         return a + b;
11     }
12 }
13
```

```
SimpleCalculatorTest.java
1 package Unittests;
2
3 import org.junit.*;
4
5 public class SimpleCalculatorTest {
6
7     @Test
8     public void testGreet(){
9         SimpleCalculator sc = new SimpleCalculator();
10         Assert.assertEquals("Hello", sc.greet());
11     }
12
13     @Test
14     public void testCalc(){
15         SimpleCalculator sc = new SimpleCalculator();
16         Assert.assertEquals(4, sc.add(a:3, b:1));
17         //Assert.assertNotEquals(0, 0);
18         //Assert.assertTrue(false);
19     }
20 }
```


Exercise

- Create a "Calculator.java"
 - It contains a int subtract, int multiply and int divide methods
 - Int divide throws a `IllegalArgumentException` if divided by zero
- Write all methods and a positive and negative test where it makes sense