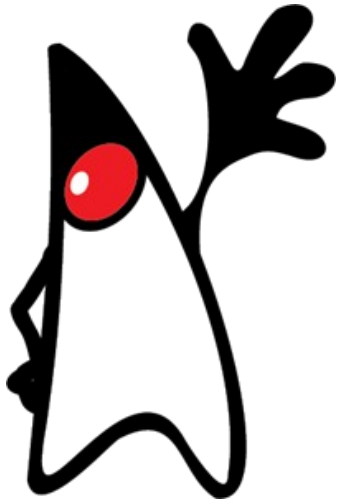
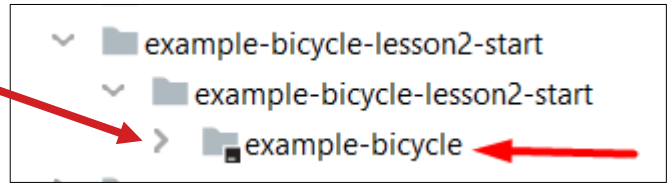
Classes part 1
in practice



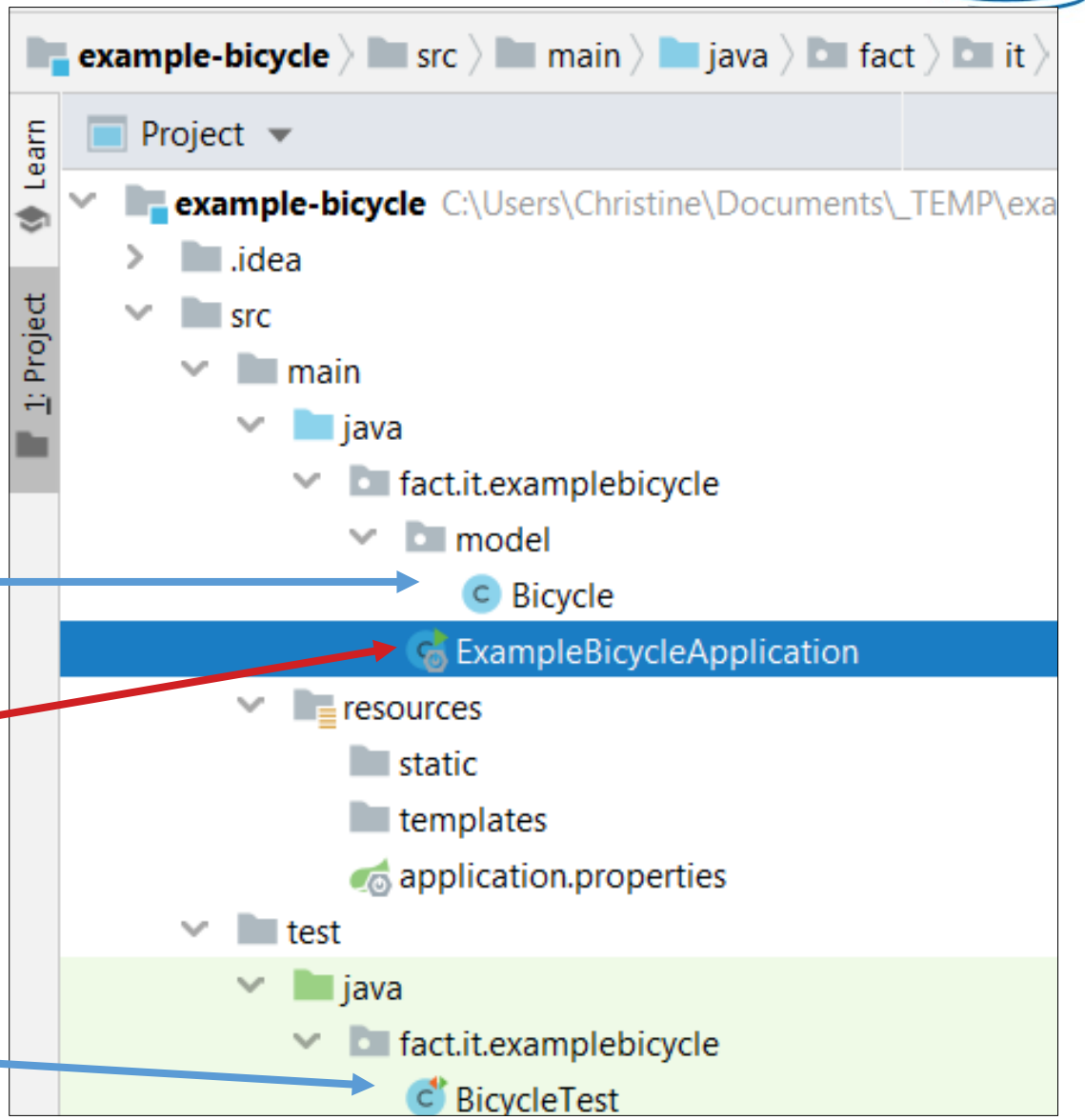
Write a class in IntelliJ

- Download the zip-file “example-bicycle-lesson2-start" on your local drive C:/Java (no shared drive).
- Unzip the file (choose *Extract Here* if possible)
- Open IntelliJ
- Choose File>Open…
- Click on the project *example-bicycle (with the special icon*)



Write a class in IntelliJ

- You can see this project structure:



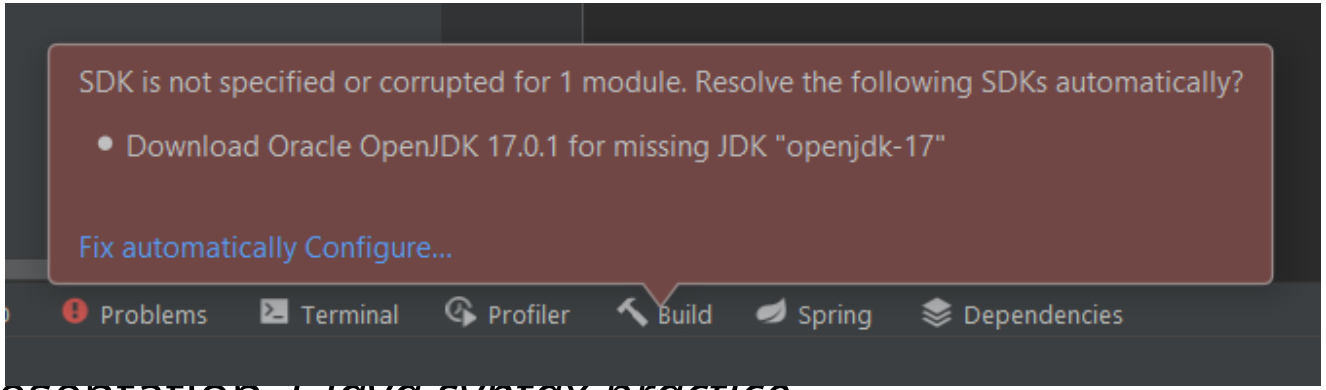
- *your class*
- *the application: here you write the code to create an object and use its methods*
- *test-class*

Example class Bicycle: attributes

- Click on the class (*cycle* and add the following (red) code:

```
public class Bicycle {
  private String type;
  private int year;
  private double rentalPrice;
```

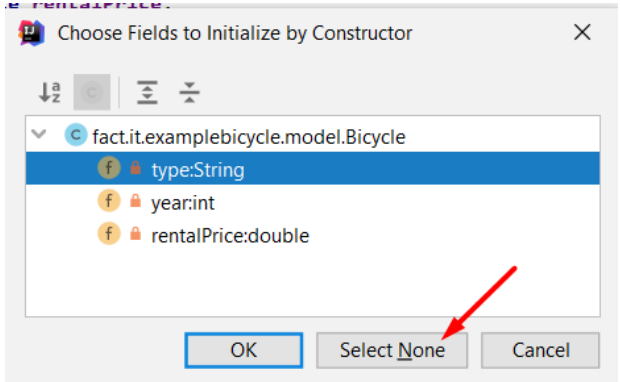
- If the code “String” turns red in your code and/or the following pop-up appears:



  - => see presentation *1 Java syntax practice*

Example class Bicycle: constructor

- Click in the code on the right mouse button (somewhere after the attributes and before the end brace)
- Choose Generate…
- Choose Constructor
- Click “Select None”
- Add 1 line of code




```
public Bicycle() {
   type = "not defined";
}
```

Every letter, every character, every space must be correct for the tests to succeed. So take a very close look at what is asked for in the assignment.

Example class Bicycle: getters


```
//getters

public String getType() {
 return type;
}

public int getYear() {
 return year;
}

public double getRentalPrice() {
 return rentalPrice;
}
```

- Click on the right mouse button just before the end brace (outside the constructor!)
- Choose Generate…
- Choose Getter
- Select all attributes
- Click OK

Example class Bicycle: setters


```
//setters
public void setType(String type) {
 this.type = type;
}

public void setYear(int year) {
 this.year = year;
}  …

public void setRentalPrice(double
rentalPrice) {
 this.rentalPrice = rentalPrice;
}
```

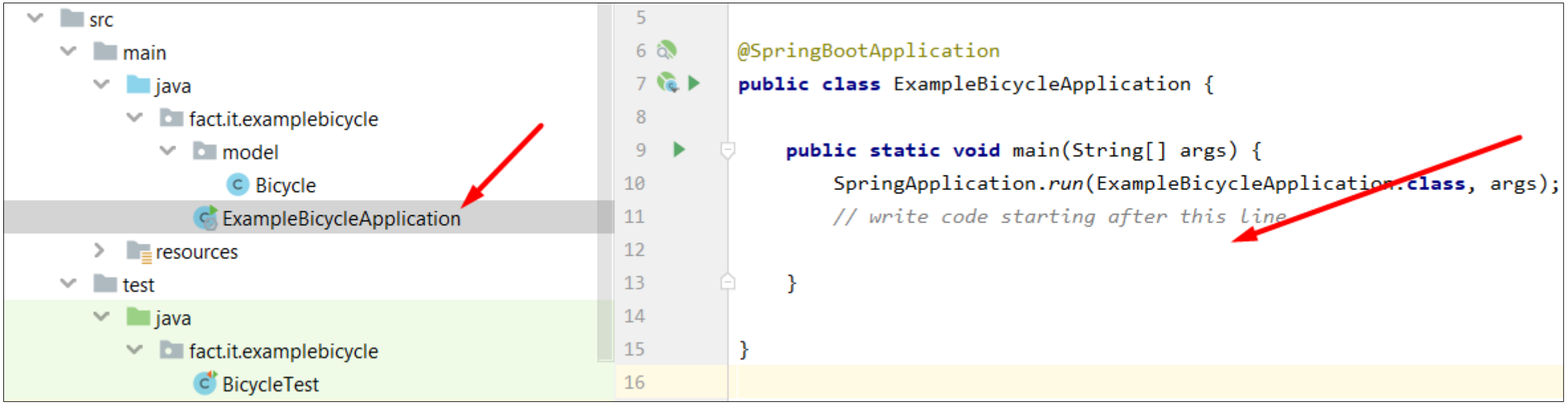
- Click the right mouse button just before the end brace
- Choose Generate…
- Choose Setter
- Select all attributes
- Click OK

Create and use objects

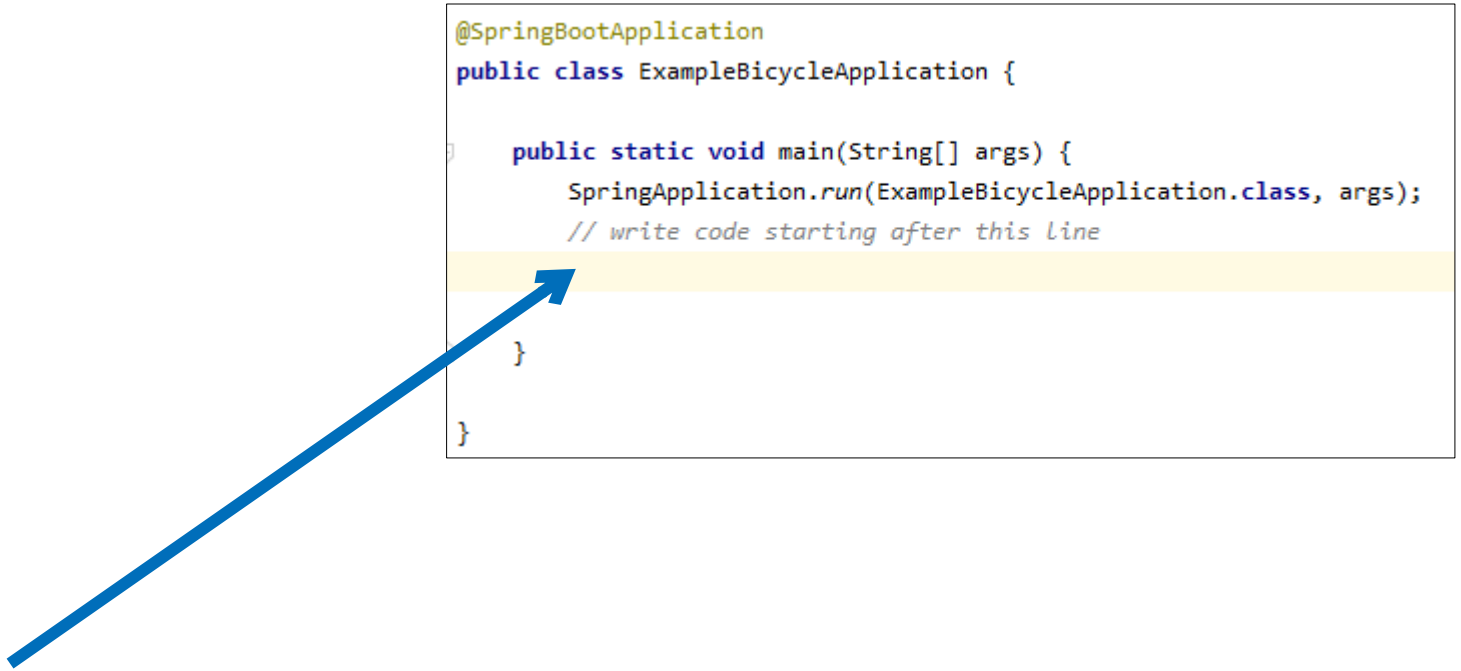
If you have created a class, you can also create an object of your own and use its methods.

How?

- In the main() method of your java application (cf. python application)



Example: ExampleBicycleApplication.java



Put your cursor here (just before the **2** braces…) and type the following code:

```
Bicycle myBicycle = new Bicycle();
myBicycle.setType("women's bicycle large");
myBicycle.setYear(2016);
myBicycle.setRentalPrice(4.5);
```

Using a Bicycle Object

- Complete further

```
public static void main(String[] args) {
    SpringApplication.run(ExampleBicycleApplication.class, args);
    // write code starting after this line
    Bicycle myBicycle = new Bicycle();
    myBicycle.setType("women's bicycle large");
    myBicycle.setYear(2016);
    myBicycle.setRentalPrice(4.5);

    System.out.println("You created a Bicycle-object with the following values:");
    System.out.println("The type of your bicycle is: " + myBicycle.getType());
    System.out.println("The year of your bicycle is: " + myBicycle.getYear());
    System.out.println("The rental price of your bicycle is: " + myBicycle.getRentalPrice());

    System.exit(0);
}
```

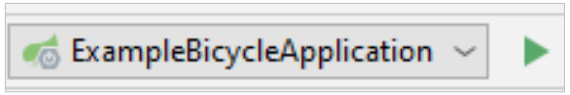
An application does not stop automatically. It does if you add **System.exit(0)** at the bottom…

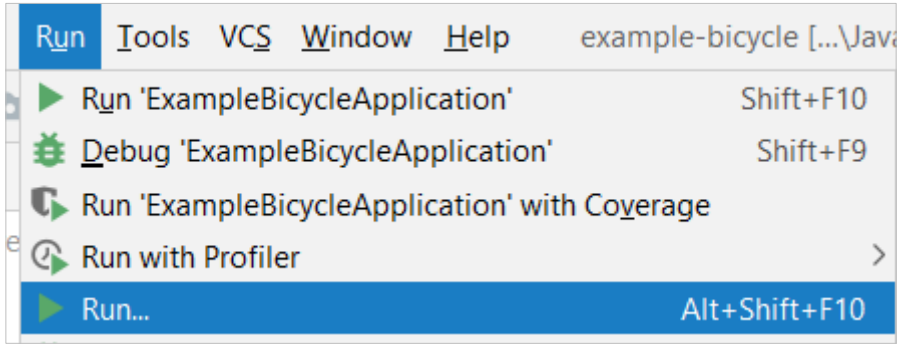
Please note that **println** contains the letter **l** and not the number **1**.

System.out.println() is automatically generated by entering 'sout' + Tab

Using a Bicycle Object

- There are 3 ways to run your application
  - On the right hand side of your screen choose "*ExampleBicycleApplication*" and click on the green arrow





  - From the menu Run or
  - Alt+Shift+F10: choose ExampleBicycleApplication

- When the application is executed you see the output window at the bottom of your IntelliJ-window. Scroll to see this output:

```
You created a Bicycle-object with the following values:
The type of your bicycle is: women's bicycle large
The year of your bicycle is: 2016
The rental price of your bicycle is: 4.5
```

Example class Bicycle: other methods

We still have to add these extra methods in the class “Bicycle”

```
public void increasePrice() {
 rentalPrice += 0.5;
}

public double getPricePerYear() {
 double pricePerYear = rentalPrice *
12;
 if (year < 2014) {
 pricePerYear *= 0.95;
     }
 return pricePerYear;
}
```

TIP: use **Ctrl+Alt+L** to format your code nicely…

Using a Bicycle Object

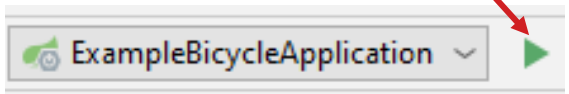
- Complete further in the application *ExampleBicycleApplication*

```
        System.out.println("You created a Bicycle-object with the following values:");
        System.out.println("The type of your bicycle is: " + myBicycle.getType());
        System.out.println("The year of your bicycle is: " + myBicycle.getYear());
        System.out.println("The rental price of your bicycle is: " + myBicycle.getRentalPrice());

        myBicycle.increasePrice();
        System.out.println("After increasing the price, the rental price is now: " +
myBicycle.getRentalPrice());
        System.out.println("And the price per year is: " + myBicycle.getPricePerYear());

        System.exit(0);
}
```

- Run the *ExampleBicycleApplication* again




```
You created a Bicycle-object with the following values:
The type of your bicycle is: women's bicycle large
The year of your bicycle is: 2016
The rental price of your bicycle is: 4.5
After increasing the price, the rental price is now: 5.0
And the price per year is: 60.0
```

Unit tests

= a method of testing software modules or pieces of source code (units) separately. In **unit testing,** one or more tests will be developed for each unit. Different test cases will then be run through. Ideally, all test cases are independent of other tests.

All unit tests are ready in the sample project but in “comment lines”. E.g.

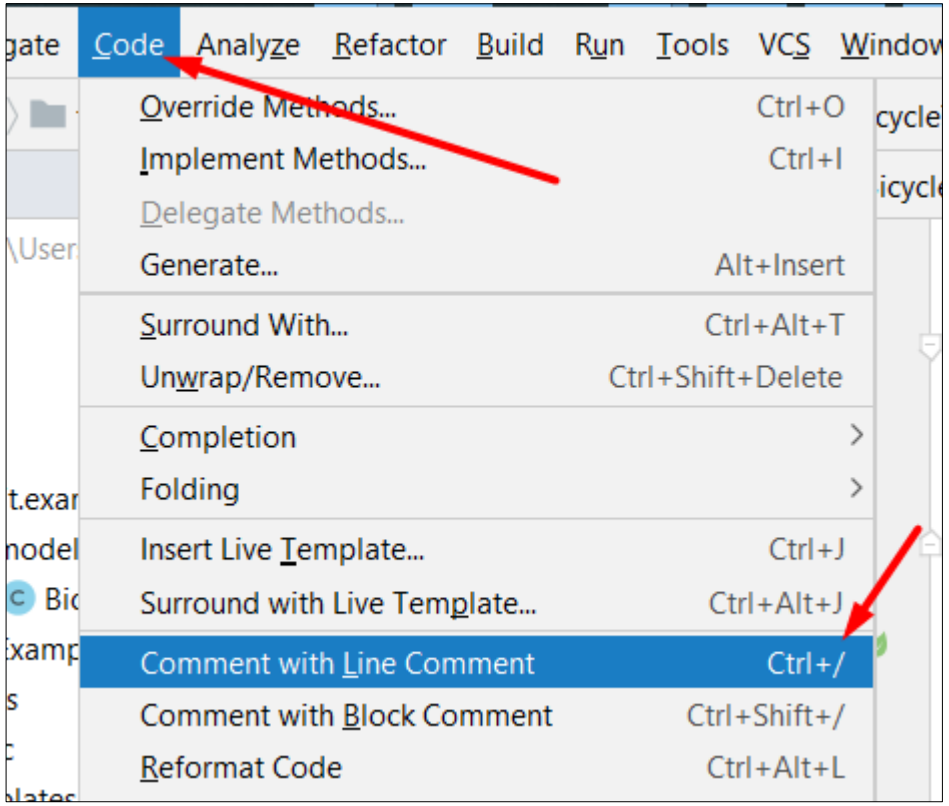
```
//    @Test
//    public void testConstructorAndGetters() {
//        Bicycle myBike = new Bicycle();
//        assertEquals("not defined", myBike.getType());
//        assertEquals(0, myBike.getYear());
//        assertEquals(0.0, myBike.getRentalPrice(), 0.01);
//    }
```

Unit tests

Your program can not run if the tests don’t work.

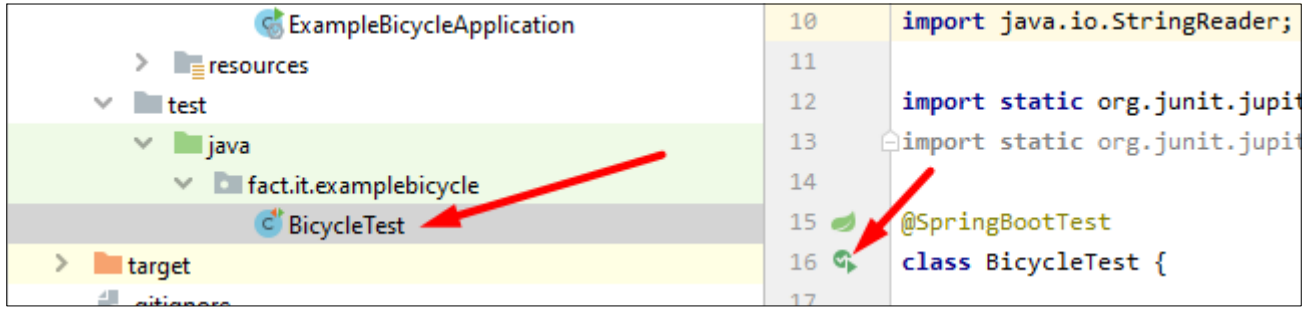
In order to test: select the comment lines and press Ctrl / or using the menu: Code>Comment with Line Comment

```
//    @Test
//    public void testConstructorAndGetters() {
//        Bicycle myBike = new Bicycle();
//        assertEquals("not defined", myBike.getType());
//        assertEquals(0, myBike.getYear());
//        assertEquals(0.0, myBike.getRentalPrice(), 0.01);
//    }
//
//
//    @Test
//    public void testSetType() {
//        Bicycle myBike = new Bicycle();
//        myBike.setType("Damesfiets large");
//        assertEquals("Damesfiets large", myBike.getType());
//
//    }
//
```

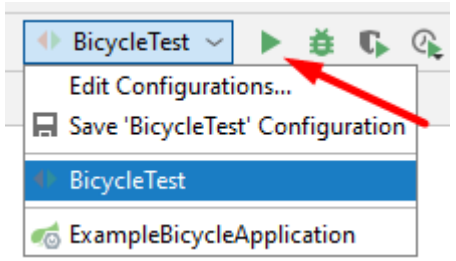


Unit tests

- 3 ways to run the tests:
  - Go to 'BicycleTest’ and click on the green icon next to the “class BicycleTest{“



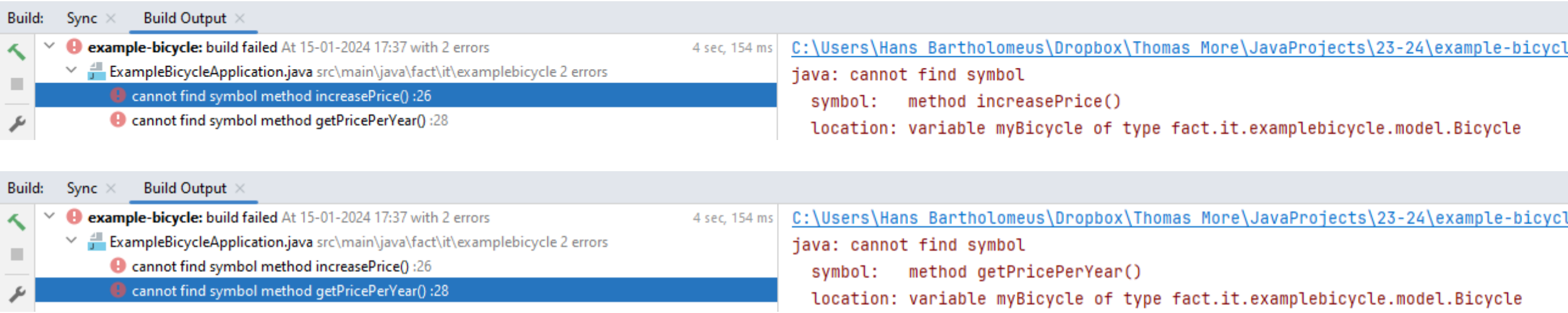
  - Go to 'BicycleTest’ and Ctrl+Shift+F10
  - Choose Run 'BicycleTest’



Unit tests

- If you haven't programmed all methods OR a method's name is different from what was requested in the exercise, the tests will not run.

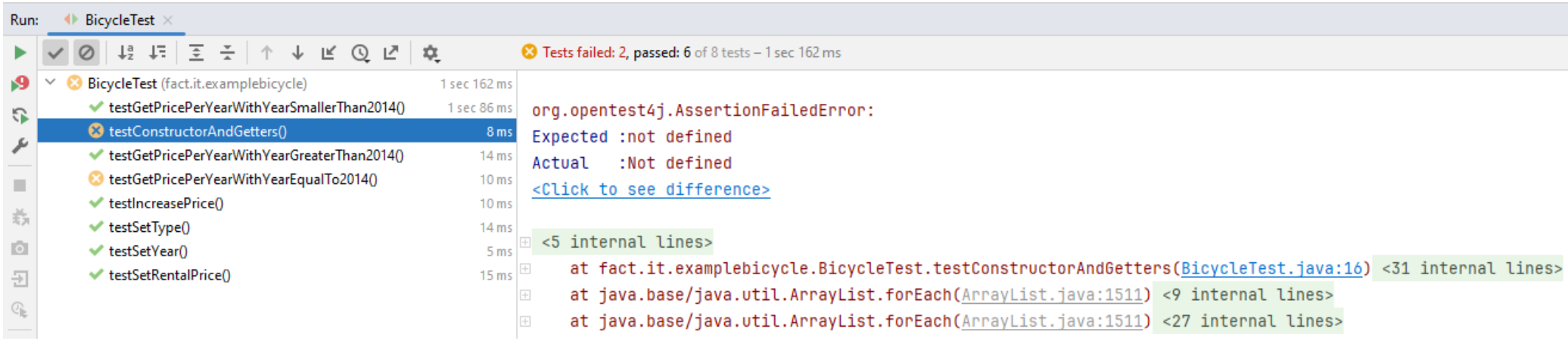
- E.g.
  - "increasePrice()" was not programmed
  - getPricePerYear() is called PricePerJaar() instead =>



Unit tests

- If you coded all methods but you made a mistake in calculation or result:

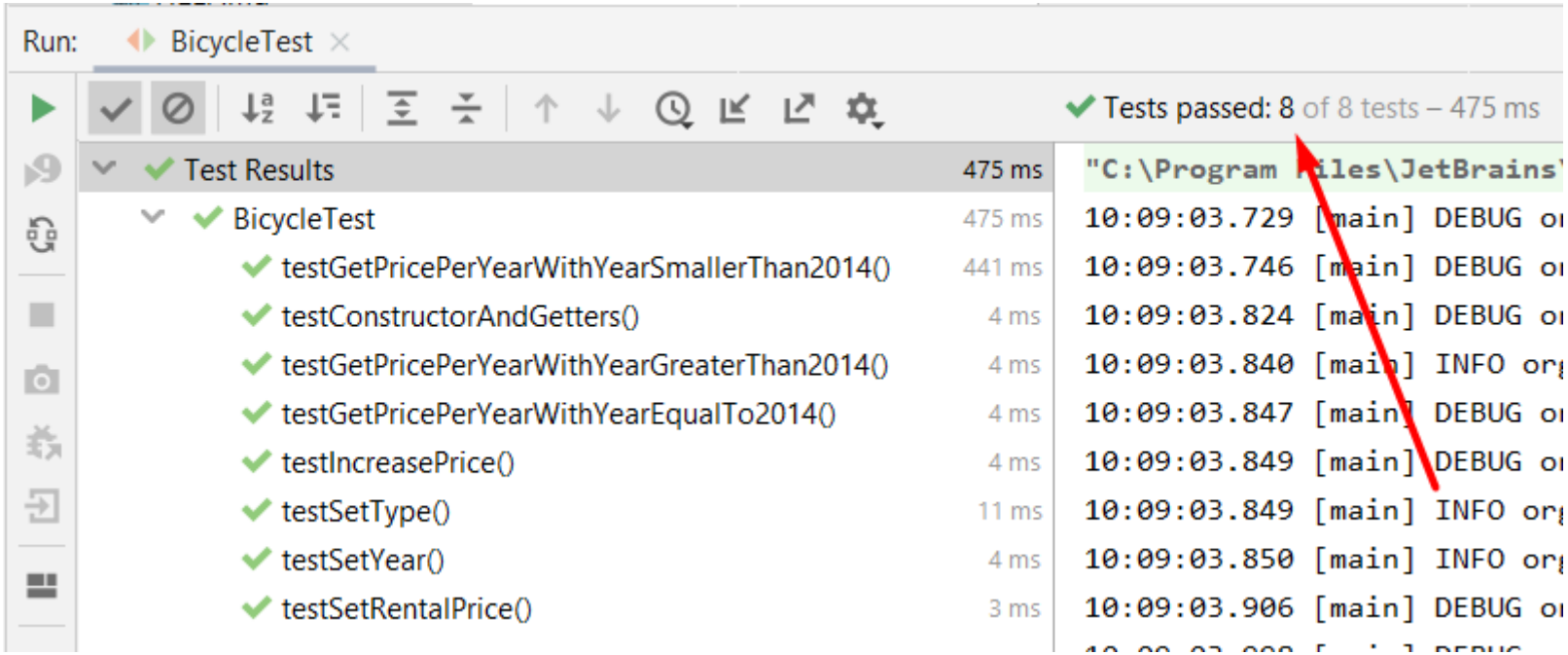
```
type= "Not defined";
```




```
if (year <= 2014) {
```

Unit tests

- If you coded all methods and they are all correct



Exercises on Canvas