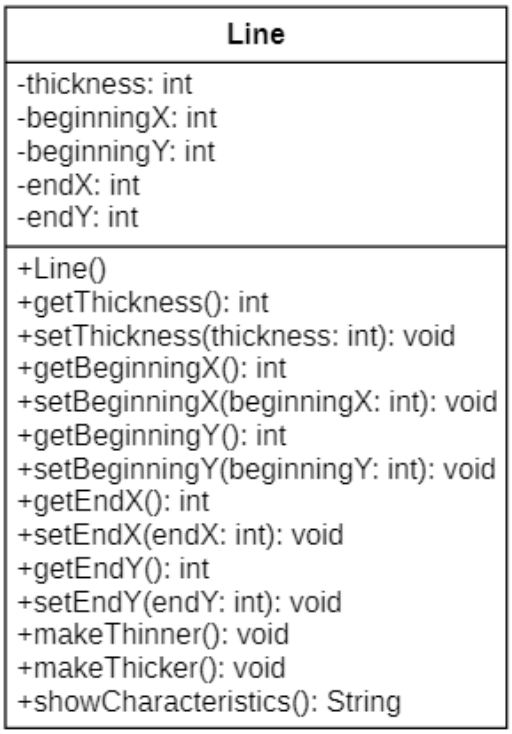
Exercises Classes Part 2

For all exercises:

- Use the start folders on Canvas
- Create the class(es) in the package *model* according to the given UML Diagram. No more and no less. **You are not allowed to create additional or other attributes or methods than the one you find in the UML Diagram**...
- After you have written these class(es), run the tests
- If all tests are successful, create objects of the class in the main method of your application and experiment with the different methods. Don't forget to import the class you have created…

Exercise 1: Line

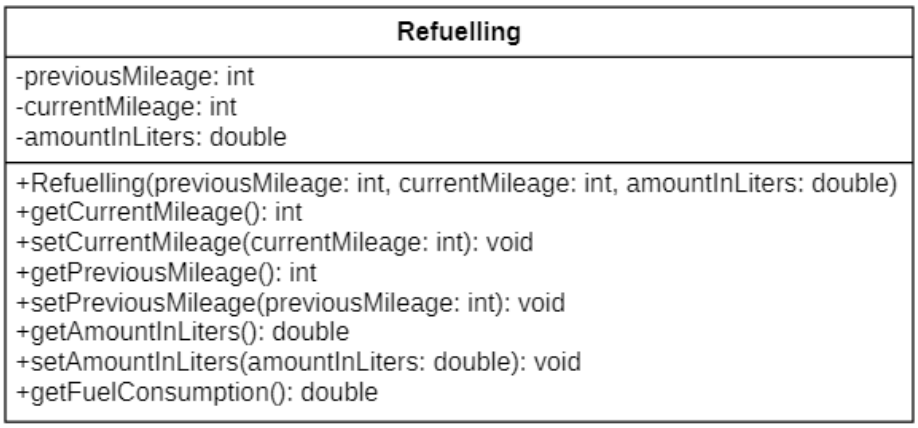
Create a class **Line** according to the UML diagram below:



a) In the *no-arg constructor* (constructor without parameters), you choose a starting point and an end point yourself. You set the thickness equal to 1.
b) In the method *makeThinner*() you reduce the thickness of the line by 1
c) In the method *makeThicker*() you increase the thickness of the line by 1
d) The method *showCharacteristics*() returns the characteristics of the line in the form of a String: *E.g*.
   `Line starts at (5,5), line ends at (5,35) and the thickness is: 19`
e) Test your exercise with the given unit tests.
f) Lastly, implement a Scanner so that users can interact with your application. Use the user input to create a Line-object and then print out the output in the same format as above. Have users input beginning X & Y, end X & Y and the thickness.

Exercise 2: Refuelling

Create a class **Refuelling** according to the UML diagram below:



a) The method *getFuelConsumption*() returns the consumption per 100 km.

   For example: if you have driven 500 km since the last refuelling and you are now refuelling 30 liters, your consumption is 6 liters/100 km.

b) Test your exercise with the given unit tests.

c) In the main method of your application, create a Refuelling-object with the previousMileage = 12150, the currentMileage= 12975 and the amountInLiters = 55.8.

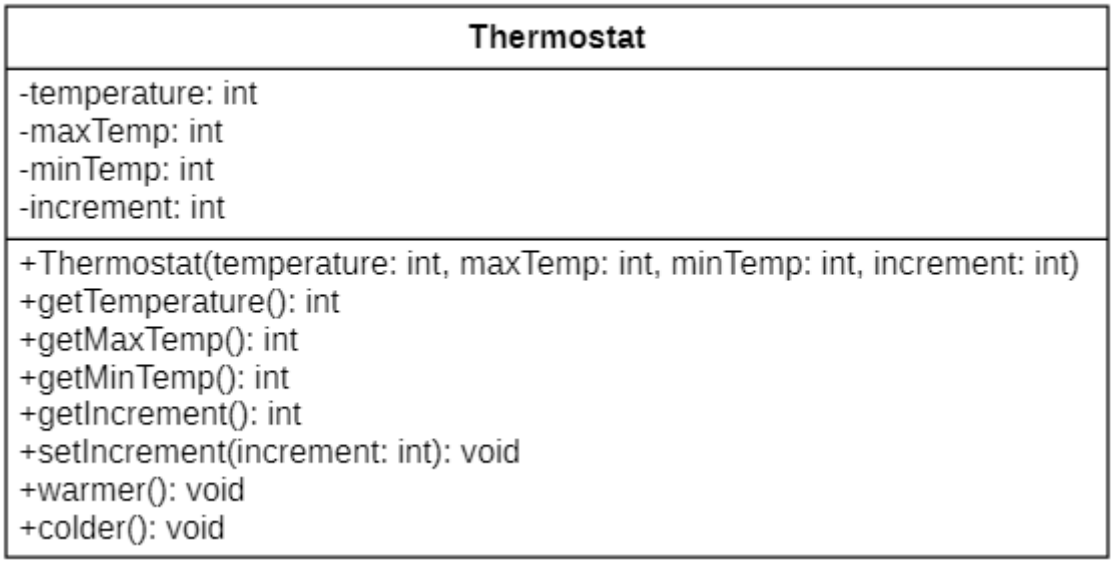
   Make sure you then show the consumption in your output window as follows:

```
Your fuel consumption was 6.763636363636364 liters.
```

d) Lastly, implement a Scanner so that users can interact with your application. Use the user input to create a Refuelling-object and then print out the output in the same format as above.

Exercise 3: Thermostat

Create the class **Thermostat** according to the UML diagram below:



Please note that the attributes do not have a setter, except for the attribute *increment*.

Each thermostat has a certain range of temperatures that can be reached. That range is determined by a minimum and a maximum temperature.

If you want to change the temperature, it is always done with a certain step size, determined by the *increment* attribute.

a) Ensure in the constructor that the minimum temperature is always lower than the maximum temperature. If these values are passed on to the constructor (through the parameters) differently, switch their values.
b) The passed temperature (= value of the parameter *temperature*) may also have to be corrected in the constructor. The temperature must always be between the minimum and maximum temperature. When an incorrect value is passed, you have to replace it with the average value of the interval = (minimum temperature + maximum temperature)/2
c) Also note that the value of the increment must always be positive. If the passed value is negative, it must be made positive. Keep in mind that this value can be set via the constructor as well as via the method setIncrement().
d) The methods *warmer*() and *colder*() allow the temperature to raise or lower by the increment value. The temperature cannot have a value outside the range set by the minimum and the maximum temperature. If this is the case, adjust the temperature to maxTemp or minTemp.

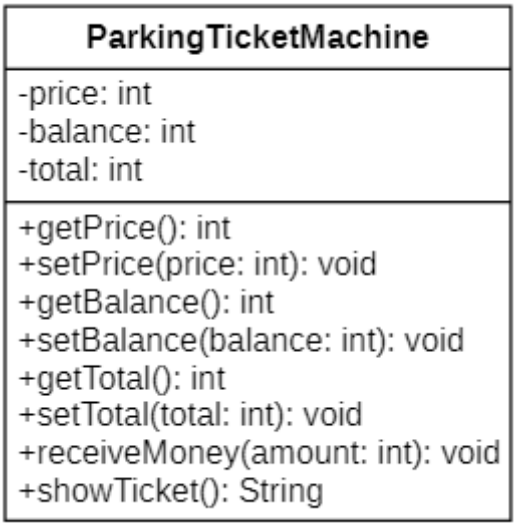
e) Test your exercise with the given unit tests.

f) In the main method of your application, create an object of the class Thermostat and use its methods to show its results in the output window.

g) Can you **overload** the methods warmer() and colder()? If so, how? Develop new methods in the Thermostat class and make sure they return a String. Test these methods in the main method of your application.

Exercise 4: Parking ticket machine

Create the class **ParkingTicketMachine** according to the UML diagram below:



You can see that the class has 3 attributes:
- *price*: the price of a parking ticket
- *balance*: the amount inserted by the user
- *total*: the total amount already received by the parking ticket machine

We do not (yet) generate a constructor.

Besides the getters and setters we add 2 more methods:
- *receiveMoney*: This method is called when someone puts money into the machine. The amount of money inserted is added to the balance of the ticket machine.
- *showTicket*: This method shows the ticket price. The String that is returned looks like this:

```
#-# Parking meter #-# Ticket = 10 euros #-#
```

a) Test your code with the given unit tests.

b) Some tests will not yet be successful. Therefor you have to add these functionalities to the *showTicket*-method:

- When a ticket is printed, it must be checked whether enough money has been inserted (balance >= price). If enough money has been inserted, the ticket will be shown. If this is not the case, the returned String will contain an error message indicating how much money the user still has to insert. This error message has to be as follows:

  ```
  You still need to insert at least 5 euros
  ```

- If the user has inserted enough money, the *balance* should be set to *(balance – price)* and the *price* has to be added to the *total*.

- If the balance is not 0 after updating it as required in the previous bullet point, you have to inform the user that he can request for change. The showTicket-method must then show the following message:

```
#-# Parking meter #-# Ticket = 10 euros #-# (change = 2 euros)
```

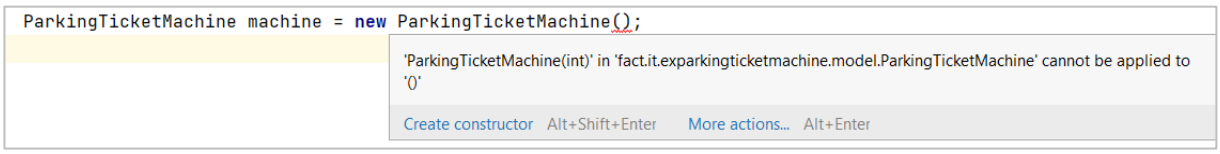
c) Continue programming until all unit tests are successful.

d) In the main method, create a ParkingTicketMachine-object.
Set the price to 10 and show the result of the *showTicket*() method. Run the application.

Note: you have not written a constructor and yet you can create an object =>
Java generates the *default constructor* during compilation

e) **Food for thought**:
Add to the class a constructor with 1 parameter (price).

Note that a compilation error is now given in the main method (and in the tests):



What causes this error? How do you solve it?