Exercises about classes and inheritance

Solve them in Visual Studio.

Exercise 13.01: People management

- The namespace of your project is “Persons”.
- You make a class “Person”.
  - 6 variables.
    - name (string).
    - birthDate (DateTime).
    - street (string).
    - houseNumber (string).
    - zipcode (string).
    - city (string).
  - 7 properties.
    - Name (string).
    - BirthDate (DateTime).
    - Street (string).
    - HouseNumber (string).
    - ZipCode (string).
    - City (string).
    - Address (string) (Dynamic property).
      - This consists of more than one variable.
      - street + space + houseNumber + jump to new line + zipcode + space + city.
  - You have to determine by yourself what needs a get and / or a set.
  - One constructor with 6 parameters.
    - name (string).
    - birthDate (DateTime).
    - street (string).

Notes

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

        - houseNumber (string).
        - zipcode (string).
        - city (string).
    - Method ShowInformation.
        - This shows all the information of a person.
- You make a class “Employee”.
    - This class inherits from Person.
    - You add 2 variables.
        - company (string).
        - wage (double).
    - You add 2 properties.
        - Company (string).
        - Wage (double).
    - You make a constructor with 8 parameters.
        - This constructor inherits from the constructor of Person.
    - Method ShowInformation.
        - This shows all the information of an employee.
        - Don’t repeat code.

You will later add a property to the class Person.

Method ShowInformation should work without editing it in the class Employee.

- You make a class “Student”.
  - This class inherits from Person.
  - You add 1 variable.
    - school (string).
  - You add 1 property.
    - School (string).
  - You make a constructor with 7 parameters.
    - This constructor inherits from the constructor of Person.
  - Method ShowInformation.
    - This shows all the information of a student.
    - Don’t repeat code.

	*You will later add a property to the class Person.* *Method ShowInformation should work without editing it in the class Employee.*

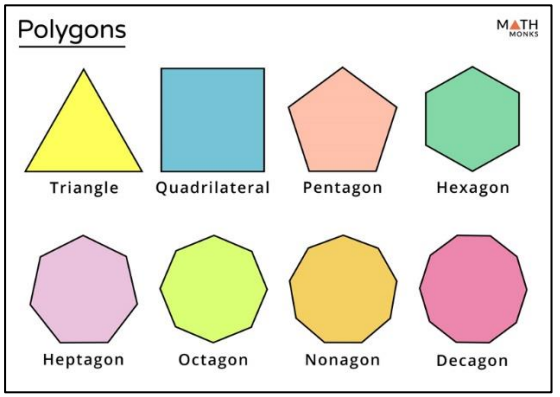
- You make a test program to test if all works fine.

	*For testing, you will add a property to the class Person. You can invent one.* *This property will be added in the ShowInformation.* *The method ShowInformation should work without editing it in the class Employee and Student.* *If this works, than change the constructor of a Person, by adding the new property.* *What must change in Employee and Student?*

Notes

Exercise 13.02: The Polygon Wood Cemetery

- The namespace of your project is “Mathematics”.



- You make a class “Polygon”.
  - One variable “colour”
  - One property “Colour”.
  - One constructor that sets the colour.
- You make a class “Rectangle”.
  - Inherits from Polygon.
  - Two extra variables / properties.
    - Width.
    - Height.
  - One constructor to create a rectangle with a Width and a Height, with a certain colour.
  - Two methods.
    - Area (and its functionality to calculate the area).
      - Width * Height.
    - Circumference (and its functionality to calculate the circumference).
      - 2 * Width + 2 * Height.

Notes

- You make a class “Square”.
  - Inherits from Rectangle.
    - Attention: A square is a special rectangle.
  - One constructor to create a Square with only one size (Width) and a certain colour.
    - This sets the Width and Height for the rectangle.

I ask you to make Square to be inherited from Rectangle. That is good for this exercise, but in general this is not real inheritance.

Inheritance should be about functionality. Doing stuff. Actions. Methods. The methods here are calculations, not actions.

The principle is correct, but working with an interface is here a better option. I ask you to work with classes (not with interface).

- You make a class “RightTriangle”.
  - Inherits from Polygon.
  - Two extra properties.
    - Base. (one side of the right angle)
    - Height. (other side of the right angle)
  - One constructor to create a triangle with a Base and a Height, with a certain colour.
  - Two methods. (Can also be properties)
    - Area (and its functionality to calculate the area).
      - (Base * Height) / 2.
    - Circumference.

Do you have enough information on how to calculate this?

Look it up with internet. Think Pythagoras.

$$a^2 + b^2 = c^2$$

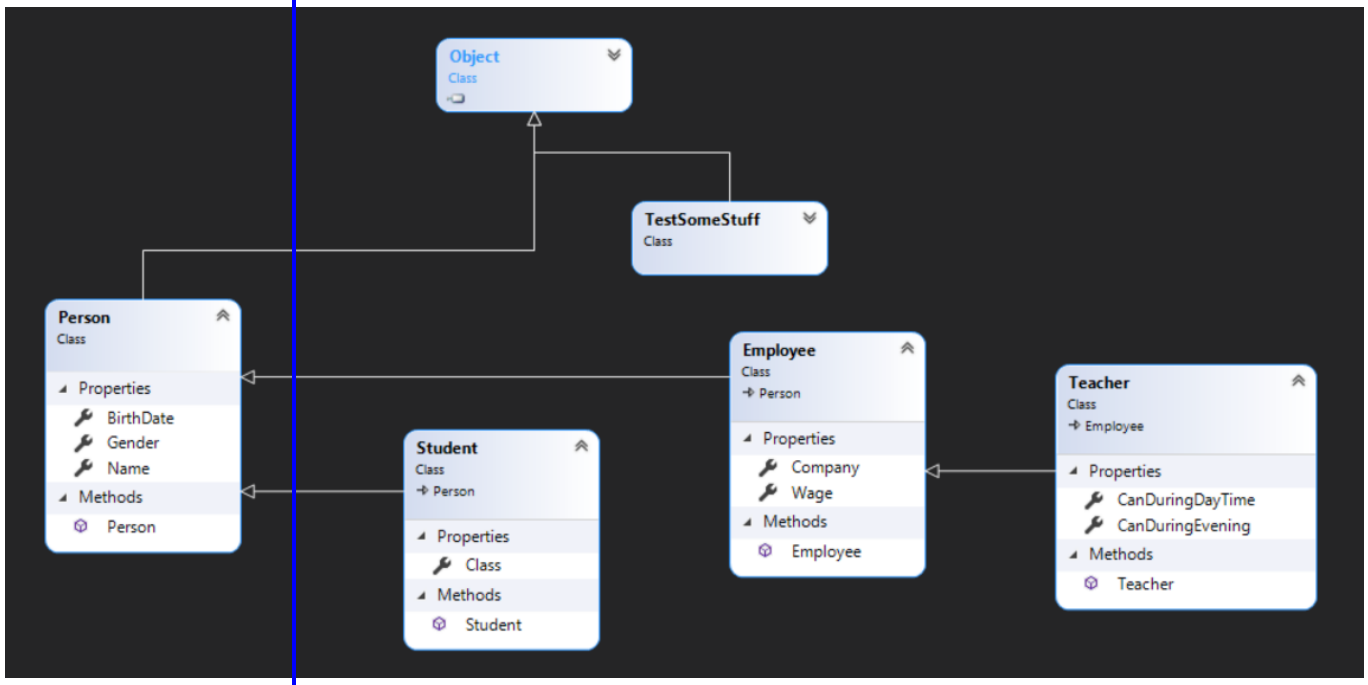
Variant 1

- Can you add a Circle to it?

Notes

Exercise 13.03: Draw a diagram in the class

- Create the classes thru a class diagram.
  - You don't type them by yourself.
- The picture is shown below.
  - Your classes will be generated by the tool.



Notes

Exercise 13.04: What's eating Gilbert Grape?

- Create a class "Animal".
  - Do the necessary stuff for making this.
  - It must have a method "Eat", that shows "Animal is eating".
- Create a class "Mammal".
  - Do the necessary stuff for making this.
  - It inherits from Animal.
  - It must have a method "Eat", that shows "Mammal is eating".
- Create a class "Cat".
  - Do the necessary stuff for making this.
  - It inherits from Mammal.
  - It must have a method "Eat", that shows "Cat is eating mouse".
- Create a class "Mouse".
  - Do the necessary stuff for making this.
  - It inherits from Mammal.
  - It must have a method "Eat", that shows "Mouse is eating cheese".

Notes

Exercise 13.05: Making a point here

- Create an abstract class “Shape”.
  - It has an empty constructor.
  - It has an abstract property “Name”.
  - It has a method “Area” that returns 0.
  - It has a method “Volume” that returns 0.
- Create a class “Point” that inherits from “Shape”.
  - It has 2 properties X and Y (the coordinates of a point).
  - The property Name returns “Point”.
  - It has a constructor with no parameters, where X and Y coordinates are 0.
  - It has a constructor with 2 parameters (X and Y coordinates).
  - You override the method “ToString” to show information about the point.
    - Show it in this format “[X, Y]”.
- Create a class “Circle” that inherits form “Point”.

Pay attention here.

This example is a very bad example. And I know it.

In real programming a Circle should not be a subclass of a Point. I’ve added this example to explain a principle, not an example that makes sense to use inheritance.

Whatever you do, I will succeed in breaking your solution. And that is what I want to explain here.

Your code will be correct, but the solution is not. This is a typical example of bad design. I’m well aware.

A Cirlce is not a subclass of a Point, but I ask you to program it that way. Inheritance is about functionality. In this exercise there are no methods, except the constructors that inherit.

  - It has 2 fields. The centre and the radius.
  - It has 3 properties “Centre” (this is a point), Radius and Name.

Notes

- Radius can't be negative. A negative value becomes 0.
- The property Name returns "Circle".

- It has 3 constructors.
  - One with no parameters. This is a point on coordinates 0 and 0 with radius 0.
  - One with 3 parameters.
    - X coordinate.
    - Y coordinate.
    - Radius.
  - One with 2 parameters.
    - A point.
    - Radius.
- You override the method Area that returns the Area of a circle.
- You create a method that calculates the circumference.
- You create a method that calculates the diameter.
- You override the method "ToString" to show information about the circle.
  - You show the centre, the radius, the Area and the diameter.

- Create a class "Cylinder" that inherits form "Circle".

Pay attention here.

This example is a very bad example. And I know it.

In real programming a Cylinder should not be a subclass of a Circle. I've added this example to explain a principle, not an example that makes sense to use inheritance.

Whatever you do, I will succeed in breaking your solution. And that is what I want to explain here.

Your code will be correct, but the solution is not. This is a typical example of bad design. I'm well aware.

A Cylinder is not a subclass of a Circle, but I ask you to program it that way. Inheritance is about functionality. In this exercise there are no methods, except the constructors that inherit.

Notes

- It has 2 fields. The base and the height.
- It has 3 properties “Base” (this is a circle), Height and Name.
  - Height can’t be negative. A negative value becomes 0.
  - The property Name returns “Cylinder”.
- It has 4 constructors.
  - One with no parameters. This is a circle on coordinates 0 and 0 with radius 0 and the height is 0.
  - One with 4 parameters.
    - X coordinate.
    - Y coordinate.
    - Radius.
    - Height.
  - One with 3 parameters.
    - A point.
    - Radius.
    - Height.
  - One with 2 parameters.
    - A circle.
    - Height.
- You override the method Area that returns the Area of a cylinder.
- You create a method that calculates the volume.
- You create a method that calculates the area.
- You override the method “ToString” to show information about the cylinder.
  - You show the centre, the radius, the Area and the diameter and the volume.

Notes

Exercise 13.06: Playing with Playing cards

- Create the classes thru a class diagram.
  - You don't type them by yourself.
- It is not the goal to implement stuff.
  - Just make the Properties and Methods that you think you will need.
    - Think how you want to structure your classes.
- Every playing card does have a colour (suit).
  - Hearts (red).
  - Diamonds (red).
  - Clubs (black).
  - Spades (black).
- Every playing card does have a value (face).
  - Ace, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen and King
- Every playing card is faced up (visible) or faced down (hidden)
- You have a deck of playing cards.
  - Create a deck with given suits and faces.
  - Count the cards in the deck.
  - Shuffle the deck.
  - Deal a hand. (a number of cards from the deck)
  - Draw a card.
- You have a hand of playing cards.
  - A number of cards out of the deck are given to a hand.
  - Count the cards in the hand.
  - Add a card.
  - Does the hand contains a card?
  - Remove a card.

Notes

Exercise 13.07: The Doors (Jim Morrison)

- Create a class that represent a door.
- A door can be:
  - Opened.
  - Closed.
  - Locked.
  - Unlocked.
    - These are methods.
- You can show the state of a door.
  - Opened door.
  - Closed door that is locked.
  - Closed door that is unlocked.
    - The states are properties.
    - Use the ToString method to show the states.
- Pay attention:
  - You can not open a door that is locked.
  - You can not lock a door that is open.
  - You can not unlock an unlocked door.
  - You can not lock a locked door.
  - You can not open an open door.
  - You can not close a closed door.
- On every time that a consumer of your class tries to do something wrong, you must send a nice error message.
  - Use the throw error message functionality if the user of the class tries to do something wrong.
  - Use the try … catch … functionality in your test application to show the received error message.
- Make sure you have tested all scenarios in your test application.

Notes

Exercise 13.08: Falling without air resistance (in love)

- Create a class that represent a falling object.
- Objects on Earth fall down due to gravity.
  - These properties are important.
    - The falling distance (d).
    - The falling time (t).
    - The falling speed / velocity (v).
- At Earth the gravitational acceleration is 9,81 m/s^2.
- There are three formulas to take into account.

$$velocity\ (v) = gravitational\ acceleration\ (g) * time(t)$$

$$distance\ (d) = \frac{1}{2} * gravitational\ acceleration\ (g) * time(t)^2$$

$$time\ (t) = \sqrt{\frac{2 * distance\ (d)}{gravitational\ acceleration\ \ (g)}}$$

- The 2 last formulas are in fact the same.
  - One is given the distance.
  - The other is geven the time.
-
- This means:
  - If you know how long a thing is falling, than the speed and the distance can be calculated.
  - If you know what the speed is at the moment, you can calculate the distance that it is falling and how long it is falling.
  - If you know the distance that is has fallen, you can calculate how long it is falling and what speed it has.
- Make a class that can answer these questions:
  - A thing is falling from 100 meter.

Notes

COPY PASTE)

        - What is the speed and the time when it touches the ground?
    - A thing is falling at a speed of 20 meter per second.
        - What is the distance it is falling at that moment?
    - How long does it take for a falling object to reach a speed of 500 kilometer per hour?
- Make sure you can show and enter the speed in kilometer per hour and in meter per second.
- Distances are always given in meters.
- Times are always given in seconds.
- Every calculated number is rounded with 6 decimals.

An example

An object is falling during 20 seconds.

- The speed is 9,81 m/s^2 * 20 s → 196,2 m/s.
- The speed is 196.2 m/s * 3,6 km/h → 706.32 km/h.
- The distance is (9,81 m/s^2) * 400 s^2 / 2 → 1962 m.

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

Exercise 13.09: Traffic light (this has three lamps)

Create a traffic light class, that will exist of three lamps.

- Create a class that represent a lamp.
  - A light has a constructor.
    - Lamp is switched off on creation.
  - A light has a method SetOnOff.
    - Input parameter is a boolean.
      - True: Lamp is on.
      - False: Lamp is off.
  - A light can be switched with a method Switch.
    - Going from on to off.
    - Going from off to on.
  - A light has a dynamic property IsOn.
    - True it is on.
    - False it is off.
- Create a class that represent a traffic light.
  - Is has three lamps.
    - Red, Orange and Green.
  - A traffic light has a constructor.
    - The traffic light is not active at that moment.
  - When you activate a traffic light.
    - Orange lamp is on. Other lamps are off.
  - When you deactivate a traffic light.
    - All lamps are off and stay off, till you activate the traffic light again.
  - When on an activated traffic light the green lamp is switched on, all other lamps are swithed off.
  - When on an activated traffic light the red lamp is switched on, all other lamps are swithed off.
  - When on an activated traffic light the orange lamp is switched on, all other lamps are swithed off.

Notes

- This is an example of a composition.
  - A traffic light exists of three different lamps.

Variant 1

Rework the exercise, but you create a coloured lamp, that inherits everything from the class Lamp/

Use the new class in your traffic light.

Variant 2

Rework the class of a coloured lamp. Now add some intensity to it.

- This class has two constructors.
  - One with one parameter, the color and by default an intensity of 0. Meaning the lamp is switched off.
  - One with two parameters, the color and the intensity.
    - When an intensity is given outside the range of [0 .. 1], the default intensity of 0 is given.
  - You must be able to set the intensity with a property.
    - Intensity 0, lamp is out.
    - Other intensity, lamp is on.

Notes

Exercise 13.10: Counting can be very Abstract

We will create two classes that are a Counter. Both classes has a different part, but most code is similar.

Create an abstract counter class that will be the base of the two Counter classes.

- An abstract class Counter will be created.
- A class PositiveCounter will be created.
  - This counter can't go below 0.
- A class NormalCounter will be created.
  - This counter can go below 0.

Content of the abstract class Counter

- A counter can be incremented with 1.
  - Add 1 to the counter.
- A counter can be decremented with 1.
  - This is an abstract method.
- A counter can be switched on or off.
  - Switching off means the counter is set back to 0.
- You can ask to the counter if it is switched off.
- You can ask to the counter what the value is.
- You can reset a counter.
  - The value of the counter can be reset to 0.
- Set properties and methods to their correct Access Modifier.
  - Public, private or protected.

Content of the class NormalCounter

- A Counter can be decremented with 1.
  - Subtract 1 from the counter.
  - Implementation of the abstract method in the class Counter.

Notes

Content of the class PositiveCounter

- A Counter can be decremented with 1.
  - Subtract 1 from the counter, but only in the range of positive numbers.
    - You can't go lower than 0.
  - Implementation of the abstract method in the class Counter.

Write tests

- Prove that your code works.
- Create a list of 2 counters.
  - 1 normal counter.
  - 1 positive counter.
- Do some stuff with it.

Notes

Exercise 13.11: A Class of Interfacing

You will create two simple interfaces, and two simple classes.

- Create an interface that can switch a thing on or off.
- Create an interface that can define a colour of a thing.
- Create a class Lamp.
  - This lamp implements the interface to switch something on or off.
- Create a class ColourLamp.
  - This lamp inherits from Lamp and implements the interface to define a colour.
- Make sure you have a class diagram.
- Test everything you have created.
- Document everything you have created.
  - Use for this the class diagram.
  - Use for this the test routines.

Notes

Exercise 13.12: Make me some stuff (a stuffed animal can)

This is a very difficult exercise.

I must receive 2 things:

- *A document in how to use the classes you have created. This document can contains example code in how to use the classes.*
- *The classes itself.*

- You must invent and create an exercise by yourself.
- There must be inheritance in classes at least 3 levels deep.
  - Parent – Child – Child of child.
  - Inheritance must be implemented somewhere.
    - this and base must be used somewhere.
- Your namespace is "StudentTryout".
- Your code will be treated as a Black Box.
  - Meaning.
    - I will create a test program using your classes.
    - I will not look inside the classes to create the test program.
  - I will do this:
    - using StudentTryout;
    - And see what is there and I will try to use it.

Whatever I do, if the program fails it must be failing in my program, because I'm doing something stupid.

It never fails in your code. Everything that goes outside your classes and inside your classes behaves correctly like you have sent me the documentation.

- After that. I will look inside your code just for trying stuff out, because a lot is possible, and see if I could have broken your code.

Notes

Class Exercises in framework Karel the Robot

Solve them in Visual Studio.

Example 09999-e Karel the Robot.zip contains all demos given during the courses.

You can adapt, extend the existing code by solving those exercises. Create a new GitHub repository to save your progress. Make me contributor if it, when you want me to evaluate the exercises.

Exercise 13.13:

-

Notes