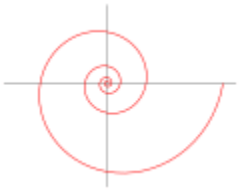
# Applikationsudvikling II

**Lecture 1**

Andrea Valente
aval@sdu.dk
https://portal.findresearcher.sdu.dk/en/persons/aval

# Structure of the course, materials

- 4 themes:
  - OOP
  - GUIs
  - Data handling
  - Development of large applications
- Tools:
  - c#, visual studio
  - https://creately.com/diagram-type/class-diagram/
  - or https://yuml.me/
  - …



- Books / materials
  - Beginning C# Object-Oriented Programming (2015)
  - [parts of] Hands-On Object-Oriented Programming with C#

  - C# https://learnxinyminutes.com/docs/csharp/
  - C# intro/refresh https://www.w3schools.com/cs/index.php
  - OOP https://www.w3schools.com/cs/cs_oop.php

# Structure of the course: exam

- Individual oral exam, based on questions
  - 15 min per student

- You will be given a few open questions 1 week before the exam
- Each student individually will prepare a short presentation, using examples from the course or new material
- At the oral: you will randomly pick 1 of the questions
  - You will then use 5-8 minutes for presentation
  - And the rest for questions

# Overview of the lectures (tentative)

- See study plan spring2023_ver3.docx

# *How are things? ;)*

- C# ?
  - Other languages?
- Collections ? Generics ?
- Classes ?
  - UML ? Design Patterns ? Refactoring ?
- WPF ?
- SQL ?
  - Other DBs? NoSQL ?
- Typical kind of programs/projects you work with?

# Who am I (?)

- Andrea*(s)* Valente
  - https://orcid.org/0000-0002-6295-9511
- **Computer Science Master**: computer graphics specialization, but thesis on type systems for object-based languages
- **Ph.D.** on formal languages, theory and implementation of calculi for mobility
- **Post-doc**, **research** and **teaching**: AAUE, then SDU Odense, now SDU Kolding

# Interests

- Computational Thinking/learning
- Programming
- Graphics/Games
- Formal systems

In the past 20 years I have taught courses in:

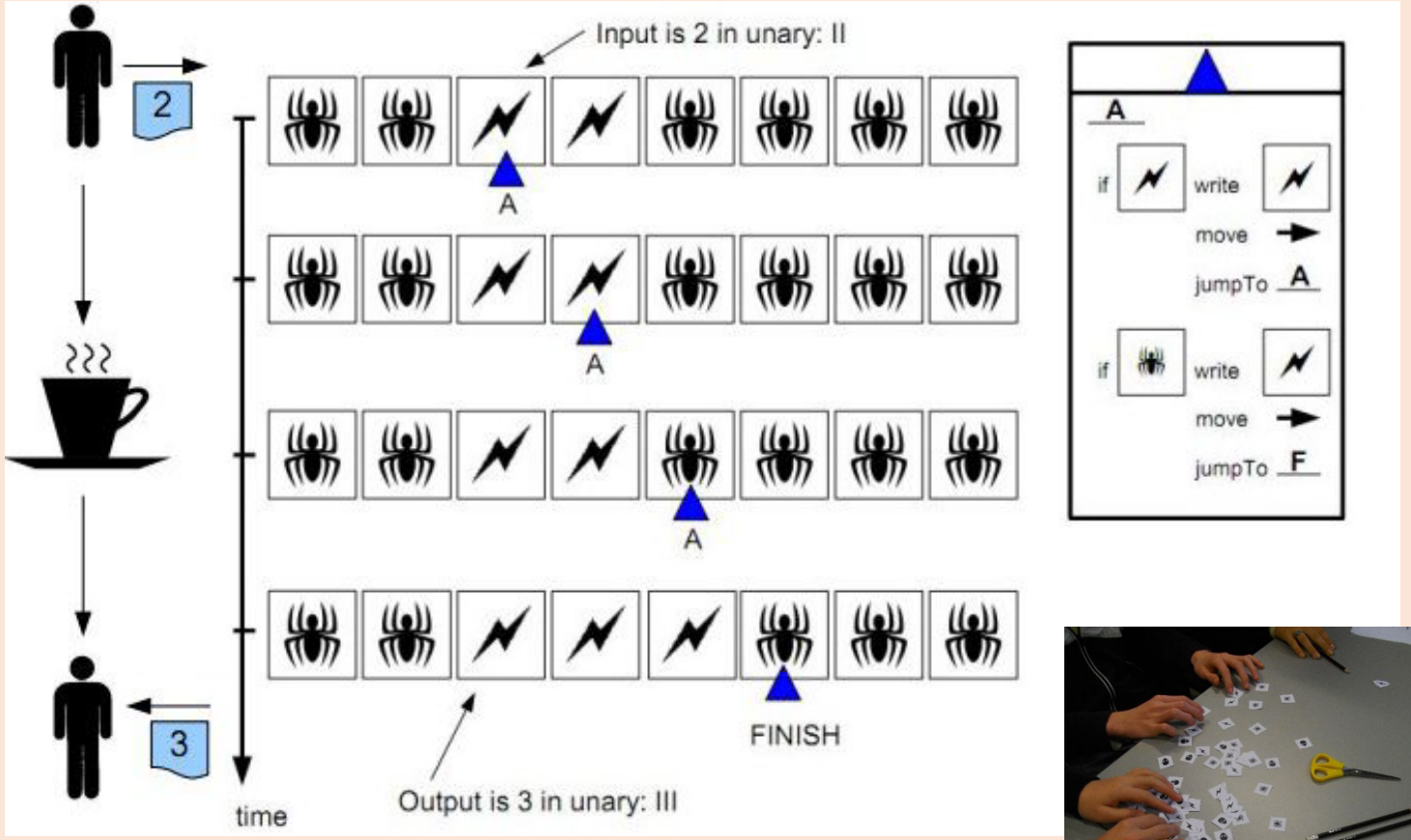
programming, HCI, math/calculus, image processing, web dev, database, OOP and software design, …

*With languages like:* Python, JavaScript (etc), Java, C#, P5, …

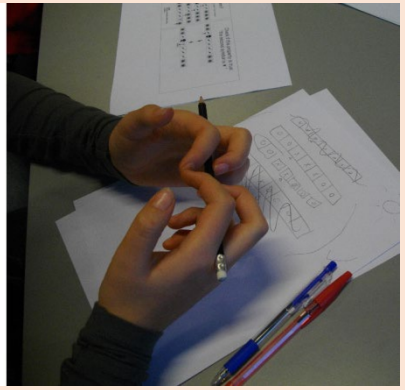
Research interests/projects …

# Paper Turing Machine

CS as a game -> sense making

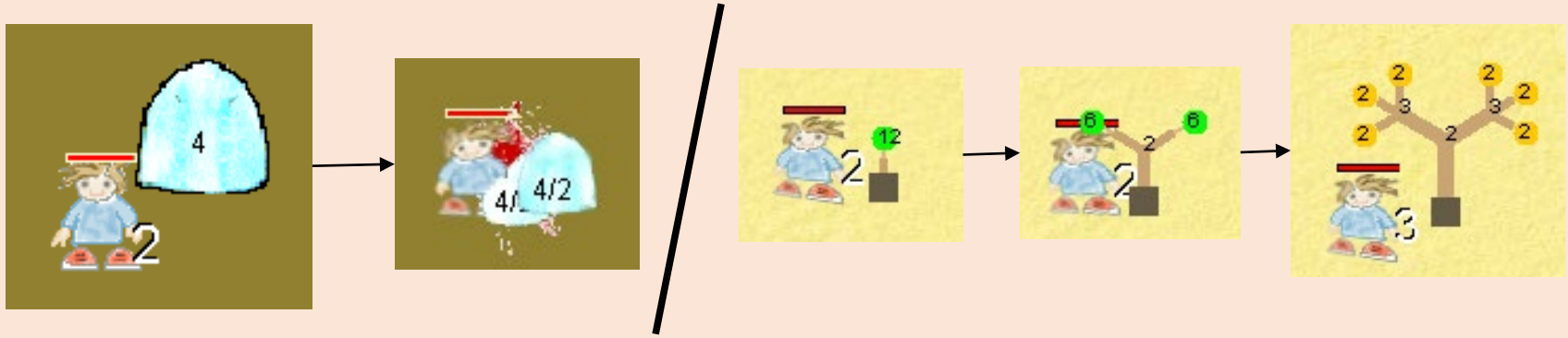




# Prime Slaughter

Computer game for playful math-learning

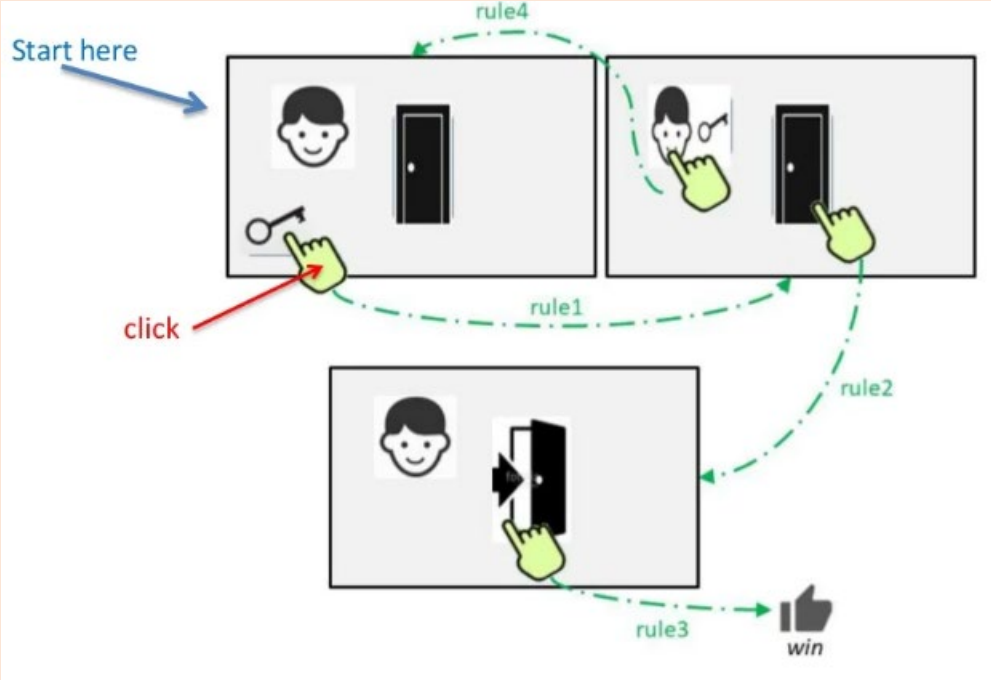



Transpose abstract mathematical concepts into playful visual interaction

- Prime factorization
- Numbers as monsters, swords and trees

A simplified version (browser-friendly) can be found here:
http://www.andval.net/tbpsSimple/index.html

# Stick and Click




- Presentation: https://www.slideshare.net/andrea270872/stick-click-the-road-towards-friendly-classroomcentered-interactive-digital-content-authoring
- The tool: http://www.andval.net/stick_n_click/stick&click.html

# Who are you?

*(round of introductions)*

name - *expectations about course (?)*

# Object-oriented Programming

**What is OOP?**

- “Object-oriented programming is an approach to software development in which the structure of the software is based on objects interacting with each other to accomplish a task.”
- “This interaction takes the form of messages passing back and forth between the objects. In response to a message, an object can perform an action.”

**We will look at real-world objects, then model them in our programs and use them to “simulate” some of the behavior of the real objects**

E.g. your problem might be about cars and how are used

- A car object consists of other objects that interact with each other to accomplish the task of getting you to the store. You put the key object in the ignition object and turn it. This in turn sends a message (through an electrical signal) to the starter object, which interacts with the engine object to start the car.
- As a driver, you are isolated from the logic of how the objects of the system work together to start the car. You just initiate the sequence of events by executing the start method of the ignition object with the key. You then wait for a response (message) of success or failure.

# OOP as a way of solving problems

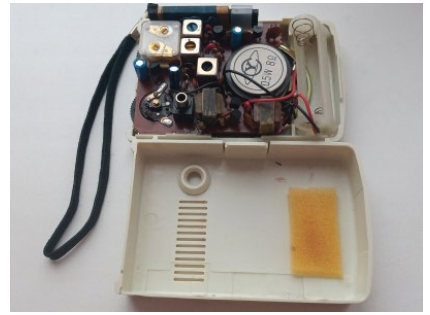
- In OOP, we try to think about our **software components** as small objects, and create relationships between them to solve a problem.
- You may have come across this approach with other programming concepts in the programming world, such as
  - procedural programming,
  - functional programming,
  - and other kinds of programming

From: *Hands-on C# book, chpt 2*

# Characteristics of OOP

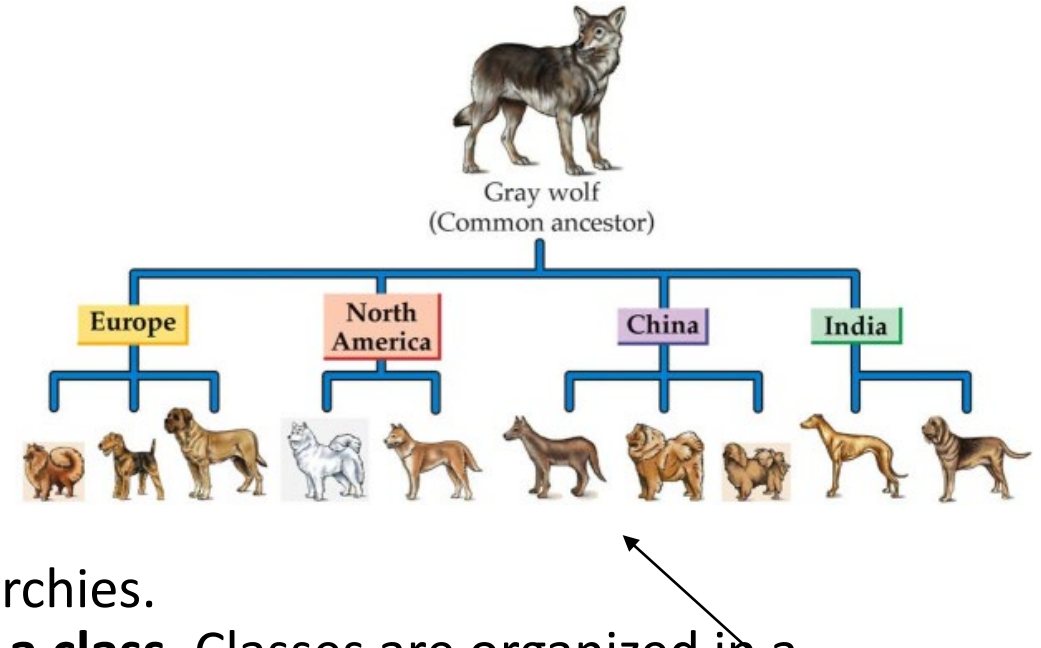
- **Objects**
  - an object is a structure for incorporating data and the procedures for working with that data
  - Objects have *properties* (data) and *methods* (to interact with them)
- **Abstraction**
  - The objects include only the information that is relevant in the context of the application. Extraneous information would be ignored.
- **Encapsulation**
  - It means that you can decide how much access to the data, inside an object, is granted. **You can decide that it is hidden** or visible to certain other objects.
  - by encapsulating data, you make the data of your system more secure and reliable.
  - You *know* (AKA *can decide*) how the data is being accessed and what operations are being performed on the data.
- Inheritance …
- Polymorphism …

Hidden data



Encapsulation:
only use buttons!

# Characteristics of OOP




- Objects …
- Abstraction …
- Encapsulation …
- **Inheritance**
  - Most real-life objects can be classified into hierarchies.
  - To define objects of a certain type, we will write **a class**. Classes are organized in a *hierarchy*.
  - **Inheritance** is a relation between classes. The word **inheritance** means receiving or deriving something from something else.
  - *For example, you can classify all dogs together as having certain common characteristics such as having four legs and fur. Their breeds further classify them into subgroups with common attributes such as size and demeanor.*
  - A class can inherit properties and/or methods from another class.
- Polymorphism …

# Characteristics of OOP

- Objects …
- Abstraction …
- Encapsulation …
- Inheritance …
- **Polymorphism**
  - *polymorph* means *many forms*
  - Polymorphism is the ability of **two different objects** to **respond to the same message** in their own unique/different way. In OOP you can create objects that respond to the same message in their own unique implementations:
    - you could send a print message to a printer object that would print the text on a printer, and you could send the same message to a screen object that would print the text to a window on your computer screen
  - Another kind of polymorphism is when you implement **different methods with the same name**. An object can then tell which method to implement depending on the context *(i.e. the number and type of arguments passed)* of the message.

# Break

- Questions so far?

# OOP in C#


*Bla bla* … but what about **programming with OOP**?!

Let's **design** and **implement** a Dog class. Let's say that:

- A dog has a name
- And we need a method (i.e. a function for dogs) to describe a dog

These are design decisions

Then we want the program to:

- create 2 dog objects
- name them "fufi" and "butch"
- and print a description of both objects

# Create a new project in Visual Studio

- You should have a class with only a **main** method inside
  - Usually this class is not a ”real” class, but an excuse to define a **main** method.
- The **main** method is *static* (i.e. **special**), it runs automatically as soon as the application is executed: *no need to call* the main method

- Before (i.e. above) the class with the main, we can write our own classes:
  - Here we can write the Dog class
  - Then in the **main** we can use the Dog class to create 2 dog objects :)

```
namespace Dog_example
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

Run and see what the program does…

```
namespace Dog_example
{
    2 references
    class Dog
    {
        public string name = "aDog"; // initial name of all dogs
        1 reference
        public string DescribeYourself()
        {
            return name;
        }
    }

    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Dog dog1 = new Dog();
            dog1.name = "fufi";
            Console.WriteLine(dog1.DescribeYourself());
        }
    }
}
```

Change the code:

- create a second dog,
- call him Butch,
- and print its description

# Inheritance: dogs are animals!

- Inheritance is a relationship between 2 classes
- Here, suppose we want to have a class Animal, and we decide that:
  - An animal has a weight
  - **Dogs are animals**

This suggests that Dogs are an extension of the *idea* of Animals.

- I can now implement a class Animal,
  and somehow declare that the class Dog takes after it!

- But… if a dog is also an animal, and all animals have a weight…
  then all dogs will have a weight too in my ”world”!

```
class Animal
{
    public int weight = 0;
}

4 references
class Dog : Animal
{
    public string name = "aDog"; // initial name of all dogs
    2 references
    public string DescribeYourself()
    {
        return name;
    }
}
```

...

use the **weight!**

```
public string DescribeYourself()
{
    return name + " and weights " + weight + " kg";
}
```

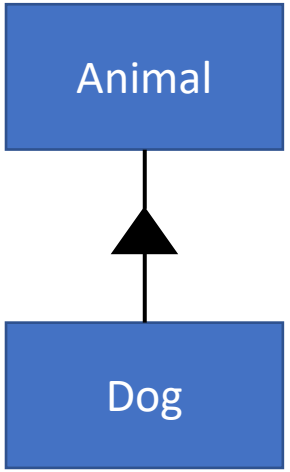
...

```
static void Main(string[] args)
{
    Dog dog1 = new Dog();
    dog1.name = "fufi";
    dog1.weight = 5;
    Console.WriteLine(dog1.DescribeYourself());

    Dog dog2 = new Dog();
    dog2.name = "butch";
    dog2.weight = 15;
    Console.WriteLine(dog2.DescribeYourself());
}
```

Look at the code in folder **code** Run it and see the output...

set the weights for the 2 dogs




*A Dog **is an** Animal*

*OR:*
*The class Dog **extends** the class Animal*

*OR:*
*Dog is a sub-class of the animal class.*
*Animal class is a super-class of the Dog class.*

# So far …

- We have 3 classes
  - The class with only the **main** method
  - A class representing animals, and an animal has a weight
  - A class for the dogs, and a dog has a name, and can describe itself

- There are relationship among these 3 classes:
  - The Dog class **extends** the class of the Animal (inheritance or hierarchy)
  - The class with the **main** *uses instances* of the Dog class (i.e. objects of Dog type) to perform some actions, calculations and/or printing some data. This relationship is called **association** between 2 classes.

# Break

- Questions so far?

# Now, let’s start from 0

- And re-design our program about dogs and animals -> **more classes!**
- This time I want to start with a diagram about the classes and use the diagram to guide myself in the implementation

- *This is a typical situation when creating applications for a customer, who has ideas about what the application should to, but cannot understand code*

- I will use a **UML class diagram**,
  - show what the diagram can represent,
  - and how to convert the information in a diagram into an implementation.

# UML class diagrams

- We can *draw* them with **yuml.me** https://yuml.me/diagram/scruffy/class/draw

- **Try it out with:**

*[Animal;name:string|Describe():string]*

*[Mammal]*

*[Dog|Describe():string]*

*[Bat|Describe():string]*

*[Human|owns:array of Animal|Describe():string]*

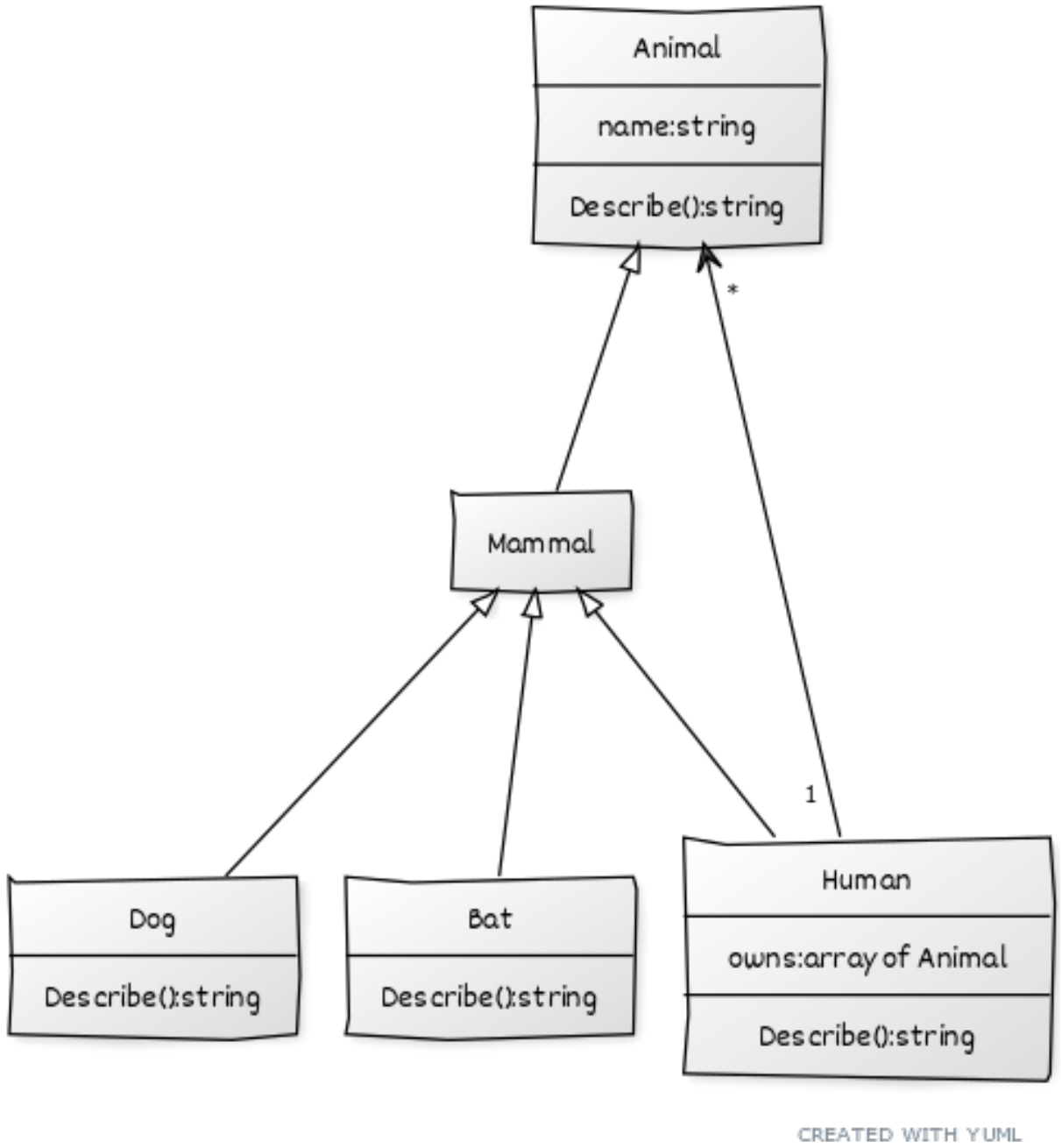
*[Animal]^[Mammal]*

*[Mammal]^[Dog]*

*[Mammal]^[Bat]*

*[Mammal]^[Human]*

*[Human]1->*[Animal]*

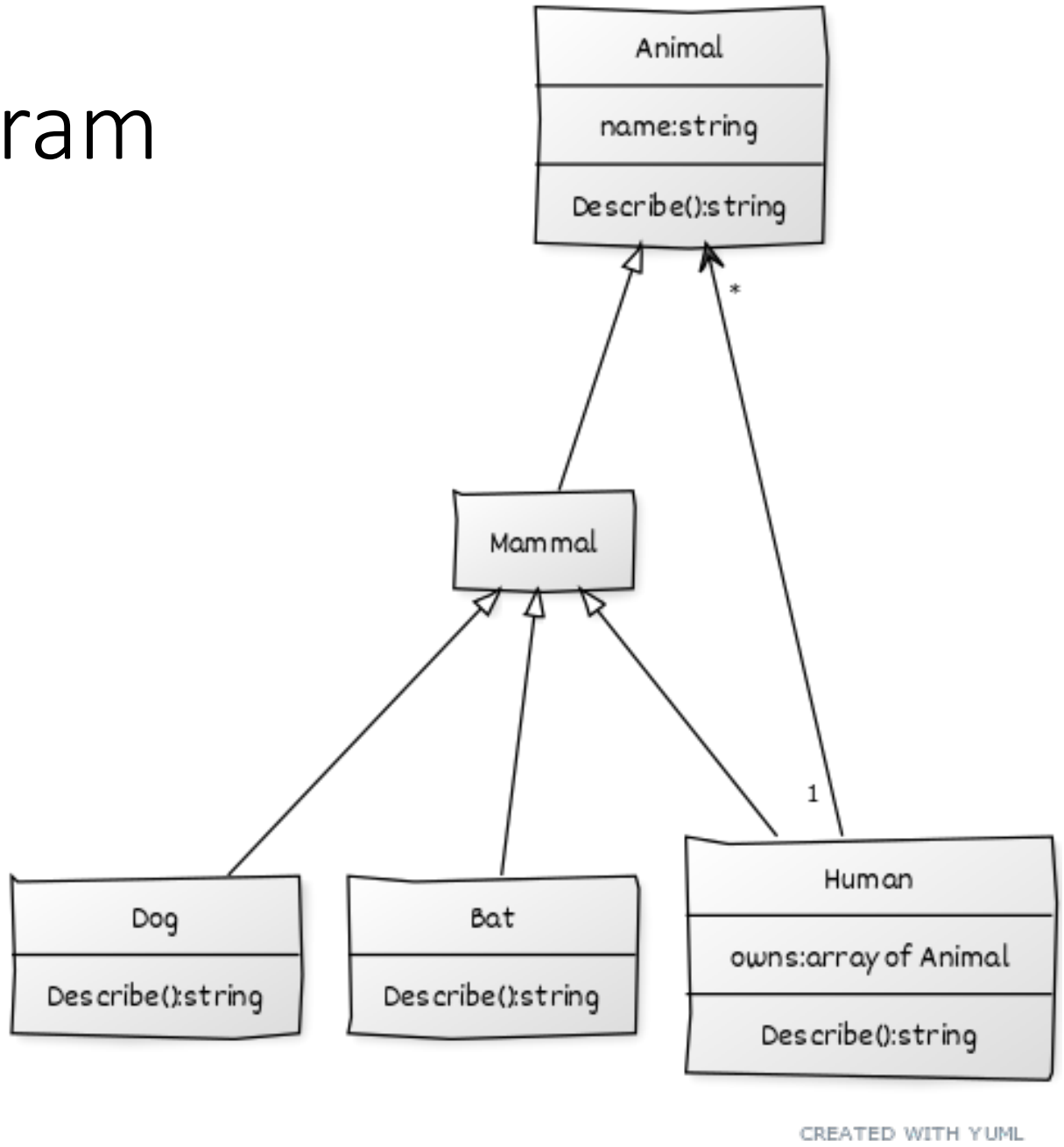


# Reading the class diagram

- How do we **read** this diagram?
- **Can you "translate" this diagram in natural language sentences?**

  …

  …



...

## Natural language sentences:

A mammal **is an** animal.

A human **is a** mammal.

…

An animal **has a** name.

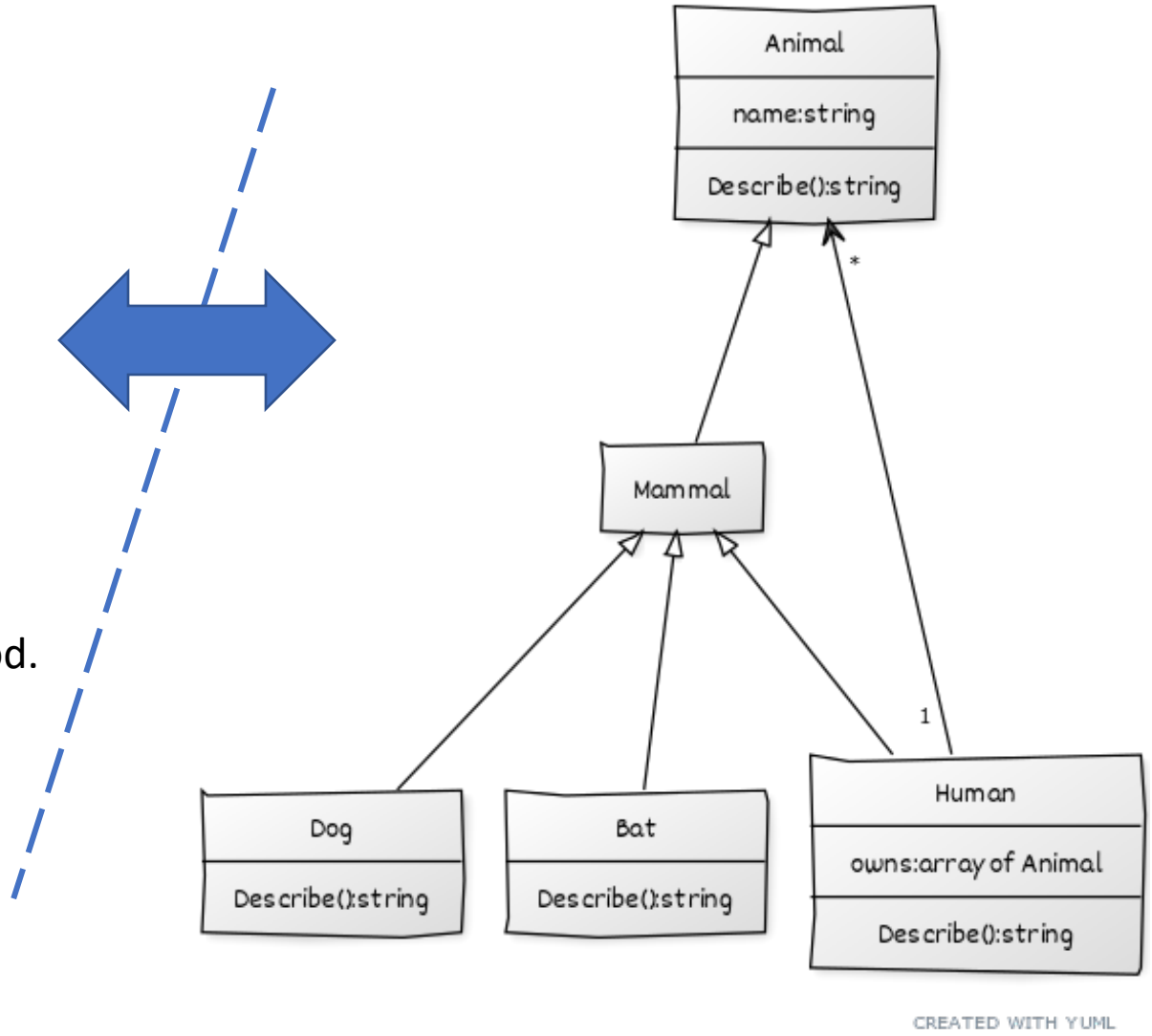
…

A dog can be **described** with the *Describe* method.

…

A human **has a list of** owned animals.

**BUT…** this means we can also go **from sentences to** class diagram!



# Implementation in C#

- **See the implementation** of these classes in **code\AnimalClasses**
- The 2 versions show the progress of the implementation…

- **Virtual** and **override** keywords: https://www.c-sharpcorner.com/UploadFile/2072a9/virtual-vs-override-vs-new-keywords-in-csharp/

- If you are not familiar with List<T> generic list, see here https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-8.0
- And if you would have used a non-generic ArrayList, read here: https://dotnetcoretutorials.com/2020/02/03/arraylist-vs-list/

# Verb-noun analysis: *how to*

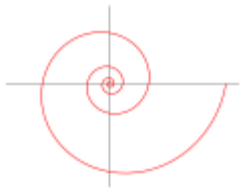
- This idea of going from the description of a problem to a class diagram is called **verb-noun analysis**
- You should:
  - Start with a natural-language description
  - Find all nouns (i.e. name of possible objects/classes)
  - Find all verbs (i.e. possible name of methods)
  - Look for possible associations: ”A has a B” sentences
  - Look for possible hierarchy: ”A is a B” sentences

- Ref: https://www.clearlaunch.com/programming-nouns-verbs/

# Questions about the verb-noun analysis?

- If it looks a bit complex, don’t worry ;)

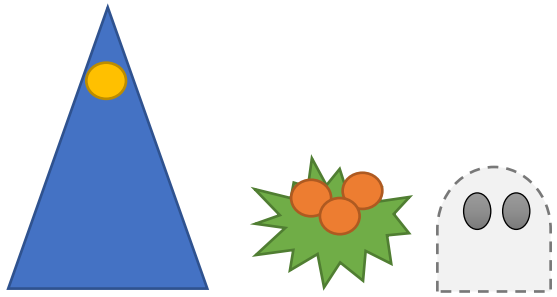
  Remember… spiral approach



- **Tasks for next time ->**

  *(but you can start here, in groups if you like)*

# A data modeling problem



- In a video game, a game level contains a list of monsters.
- There are 3 kinds of monsters: **large** monsters, **furry** monsters and **invisible** monsters.
- All monsters have a certain amount of health points (HPs) , and a monster can be hit via a method: **Hit(damagePoints)** so that the damage points decrease the health points of the monster. However:
  - large monsters take only half of the damage points
  - invisible monsters take only 1 damage points whatever the value of damagePoints, because the player cannot see them well enough to inflict the entire damage
  - furry monsters get all damage points because they are fluffy and sweet **:´(**

- In your **main** method do this:
  - Create 3 monsters, one for each type. Then Initialize the monster objects with some relevant values for HP
  - Prints a description of each monster
  - Call the Hit method on each monster, with random damagePoints, then prints the descriptions again

# The following code generates a random integer number, between 1 and 6

```
Random rnd = new Random();
Console.WriteLine( rnd.Next(1,6) );
Console.WriteLine( rnd.Next(1,6) );
Console.WriteLine( rnd.Next(1,6) );
Console.WriteLine( rnd.Next(1,6) );
```

# Your tasks…

Create a new project and write this silly game as a C# program.

1. Using the verb-noun analysis, write down which classes you need, in order to implement the game level. Consider attributes and methods.
2. Detail the relationships between classes, in particular inheritance and association (AKA the **has-a** relations).
3. Draw a UML class diagram of your classes. Use yuml.me and save the PNG diagram.
4. Now, implement your classes in C# (following the steps suggested in the lecture) and add the necessary instructions in the **main** method

**(EXTRA)** Consider that furry monsters can also be pink. **Pink furry monsters** are a special kind of furry monsters, and they are so weak that when they get hit, they get double damage points. Modify your program so that you also have pink furry monsters in the game.

**Run your game repeatedly, to test that all works as indented.**