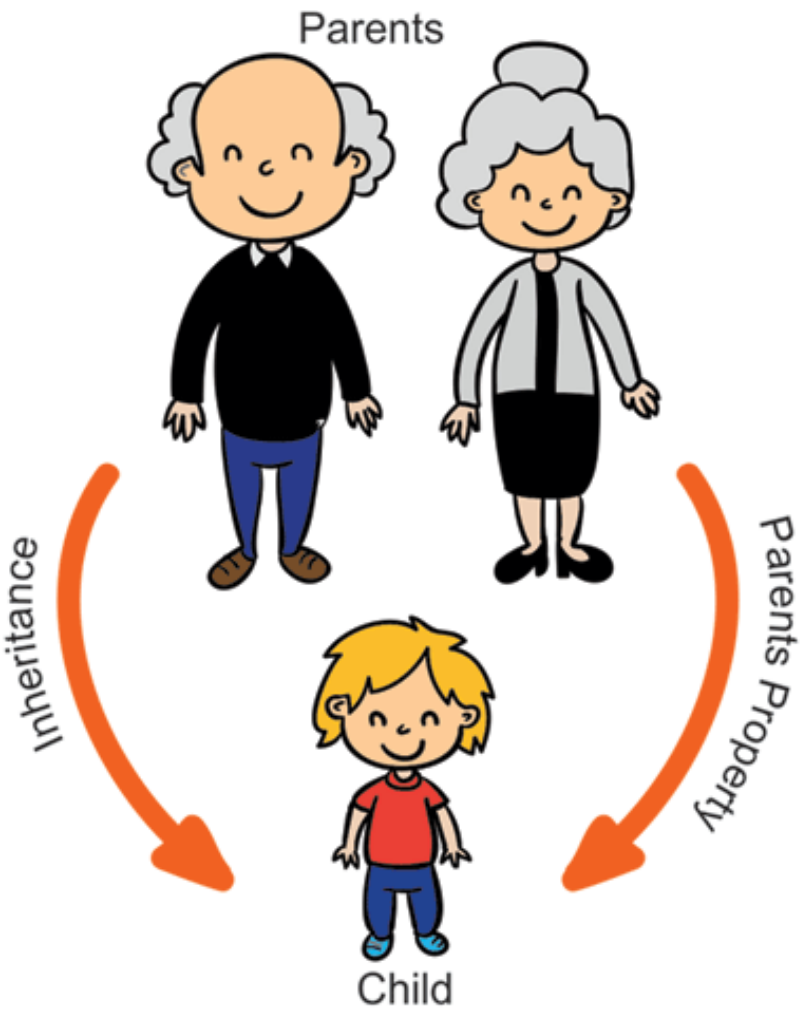
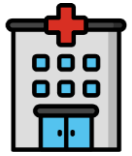
OOP

1. Typed languages
2. Object/Class
3. Aggregation
4. Encapsulation
5. Inheritance
6. Interface Polymorphism

Inheritance





In a hospital, we have the following people:

PERSON

A person has a name, an address a date of birth

EMPLOYEE

An employee **is a** person
But has salary and year of arrival in the hospital

PATIENT

A patient **is a** person
But has a health history

DOCTOR

A doctor **is a** employee
But has a medical specially

MANAGER

A manager **is a** employee
But has skills in management

```
class Person {
  name: string;
  address: string;
  dateBirth: string;
}

class Employee {
  name: string;
  address: string;
  dateBirth: string;

  salary: number;
  yearArrival: number;
}
class Doctor {
  name: string;
  address: string;
  dateBirth: string;

  salary: number;
  yearArrival: number;

  medicalSpecialities: string[];
}
```

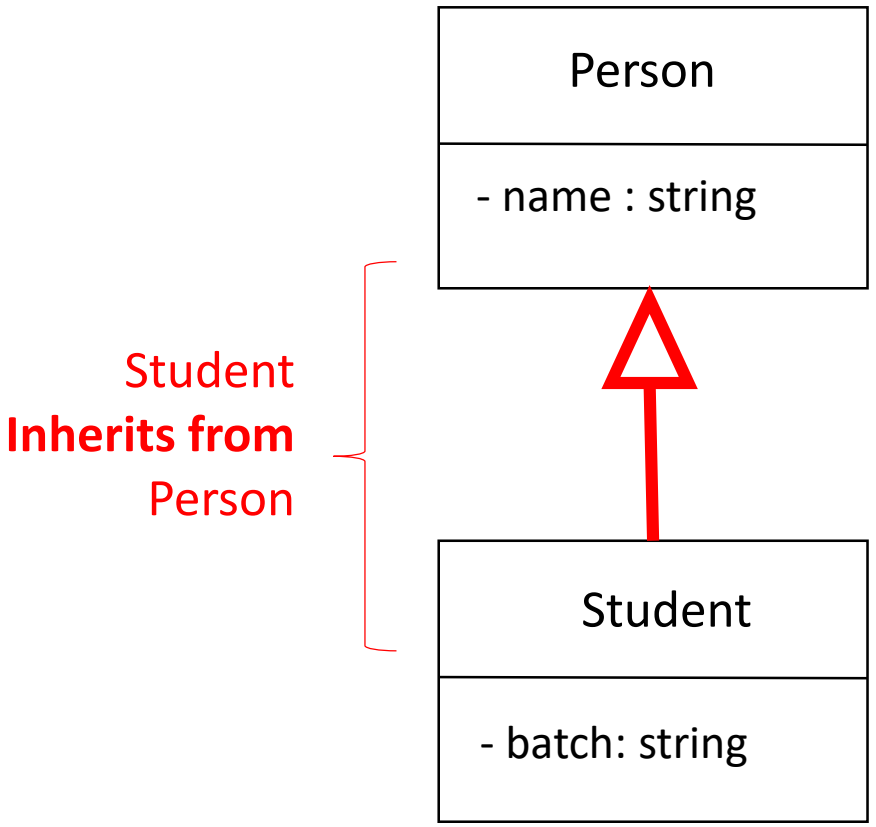
SOLUTION

ONE

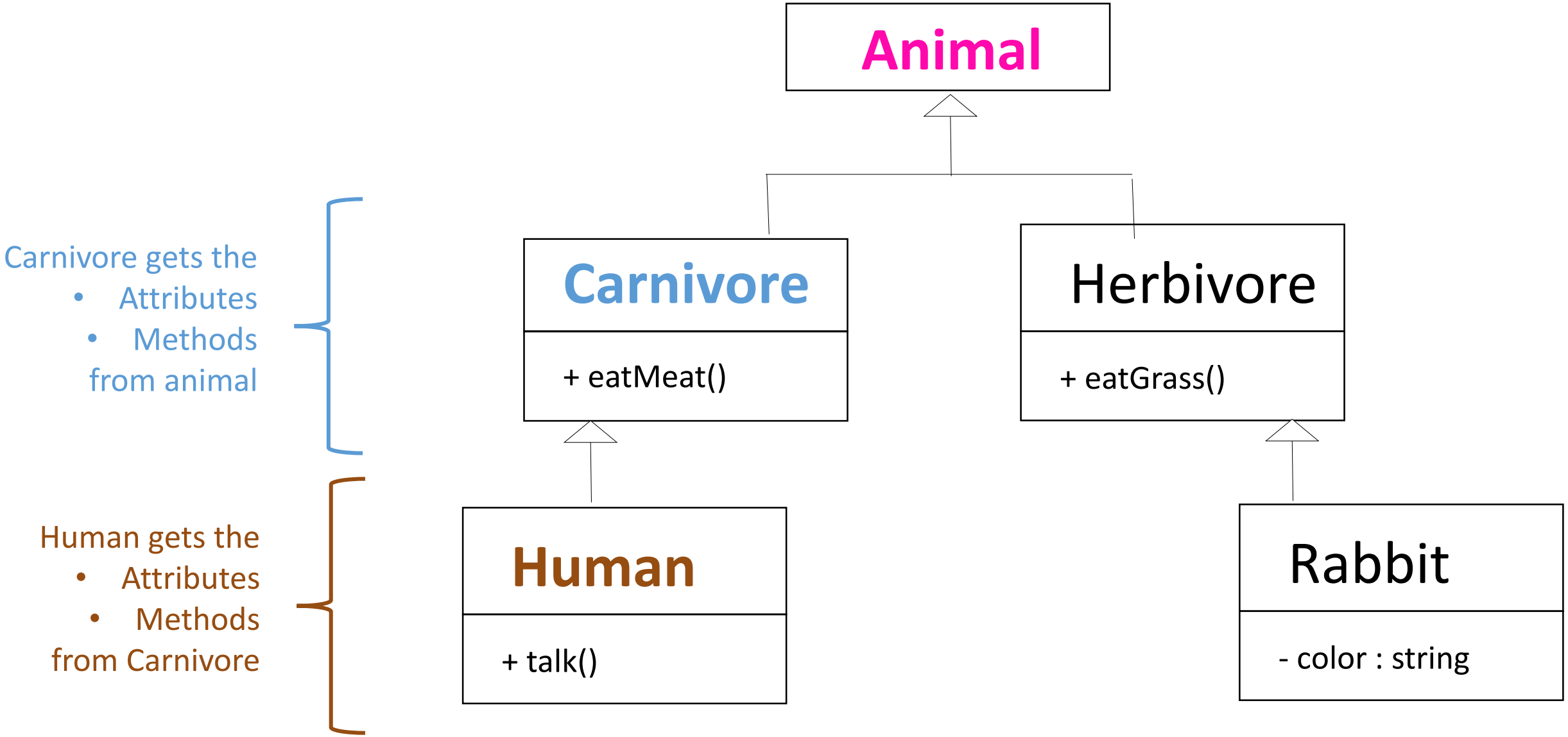
What do you think about this first solution ?

A class can inherit from another class

UML class diagram



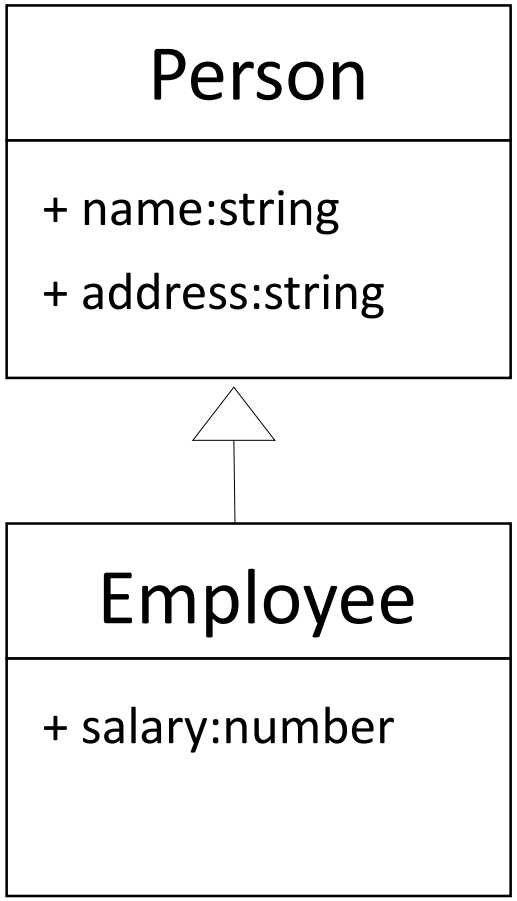
A class can inherit from another class



DEMO - 1

Children class get attributes from super class

SUPER CLASS



CHILDREN CLASS

```
class Person {
  name = "aa";
  address = "bb";
}
```

EXTENDS IS THE KEYWORD TO INHERIT !

```
class Employee extends Person {
  salary = 1000;
}

let ronan = new Employee();

console.log(ronan);
```

WHAT IT WILL PRINT ?

DEMO - 2

Children constructor must call the super constructor

```
class Person {
  name: string;
  address: string;
  constructor(name: string, address: string) {
    this.name = name;
    this.address = address;
  }
}

class Employee extends Person {
  salary: number;

  constructor(name: string, address: string, salary: number) {
    super(name, address);
    this.salary = salary;  }
}

let ronan = new Employee("ronan", "paris", 400);
console.log(ronan);
```

super()

Is a call
To the
Super
Constructor !!

ACTIVITY 1

15 MIN

1 - Change the change of **Hopital.ts**
To avoid duplication of attributes

To do this:

- ✓ Employee must extend Person
- ✓ Doctor must extend Employee

```
class Person {
  name: string;
  address: string;
  dateBirth: string;
  constructor(name: string, address: string, dateBirth: string) {
    this.name = name;
    this.address = address;
    this.dateBirth = dateBirth;
  }
}

class Employee {
  name: string;
  address: string;
  dateBirth: string;
  salary: number;

  constructor(
    name: string,
    address: string,
```

2 - Add the Manager :

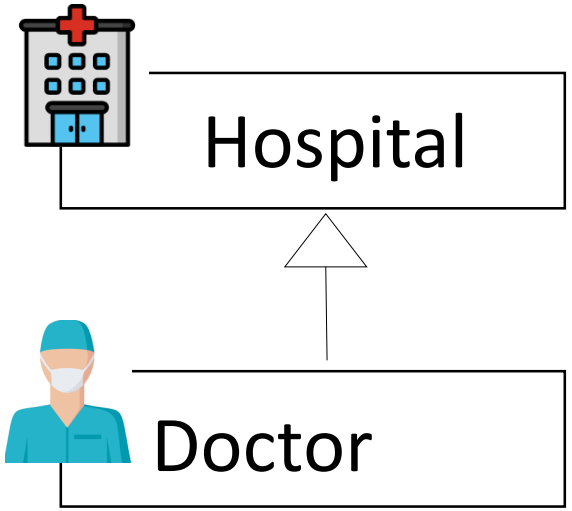
A manager **is an** employee
But has skills in management

MANAGER

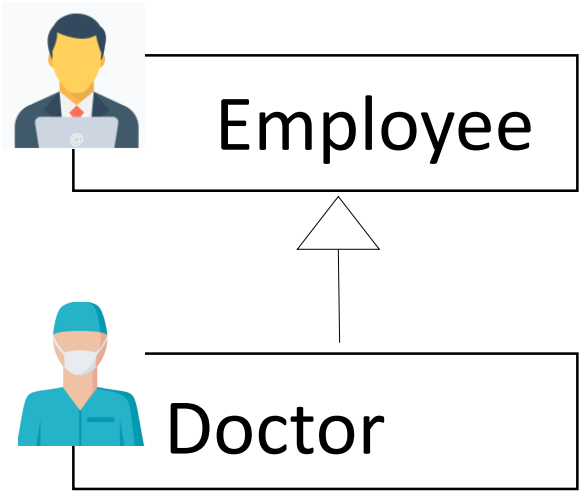
3 - Test your code !! (create some objects)

You cannot extend everything !

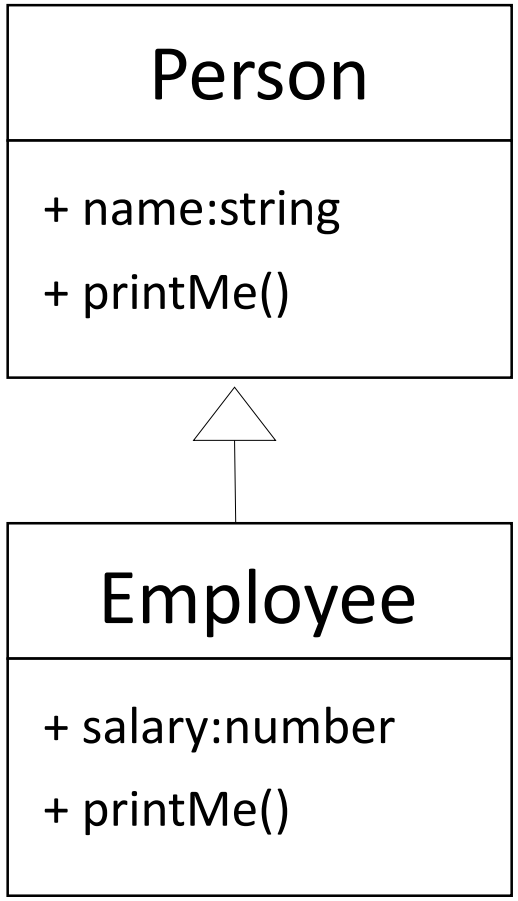
A doctor is not a hospital



A doctor is an employee



super can also be used to access to parent method




```
class Person {
  name = "aa";

  printMe() {
      console.log("name : " + this.name);
  }
}

class Employee extends Person {
  salary = 1000;

   printMe() {
      super.printMe();
      console.log(“salary : " + this.salary);
  }
}

let ronan = new Employee();
ronan.pritnMe();
```

WHAT IT WILL PRINT ?

ACTIVITY 2

15 MIN

Update the code of ColorPoint:

getInfo() must return the existing info related to Point + color

Example :

```
let p2 = new ColoredPoint(10, 20, "red");
Let info = p2.getInfo()
```

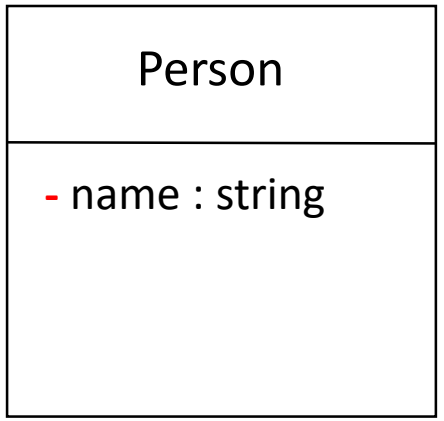
Shall return:

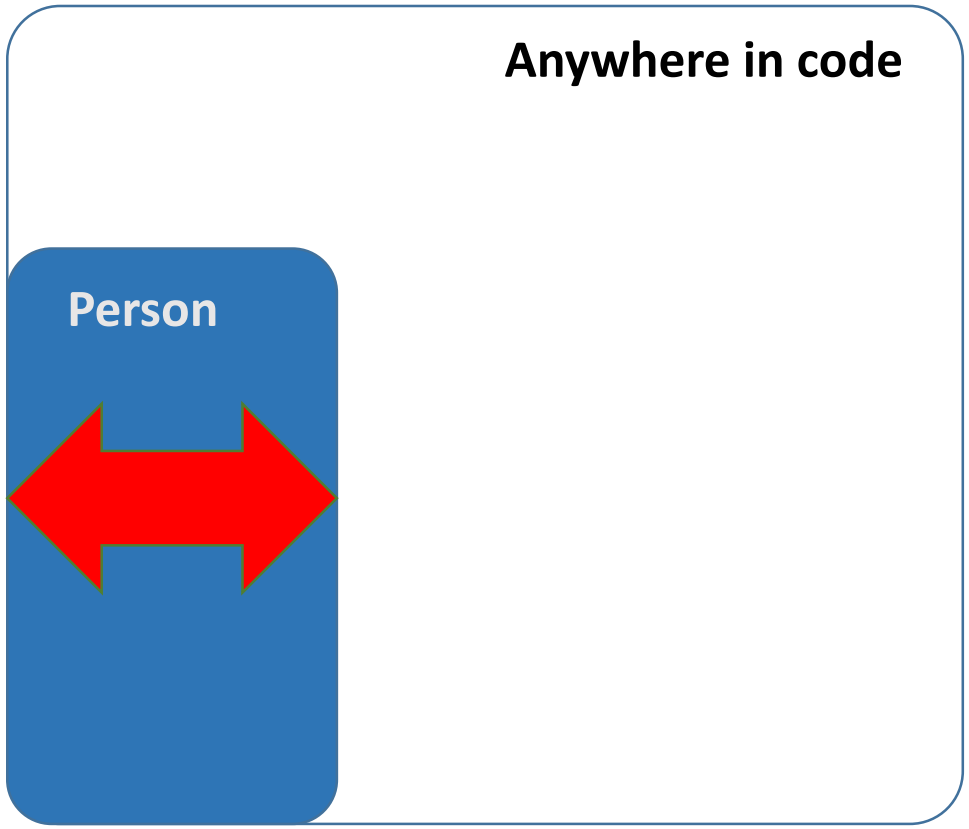
position x=10 y=20 and color=red

```
import { Point } from "./Point";

export class ColoredPoint extends Point {
  private color: string;

  constructor(x: number, y: number, color: string) {
    super(x, y);
    this.color = color;
  }

  /**
   * @returns information about the colored point
   */
  getInfo(): string {
    return ""; // TODO : must get the info from point and add t
  }
```

Visibility : PRIVATE

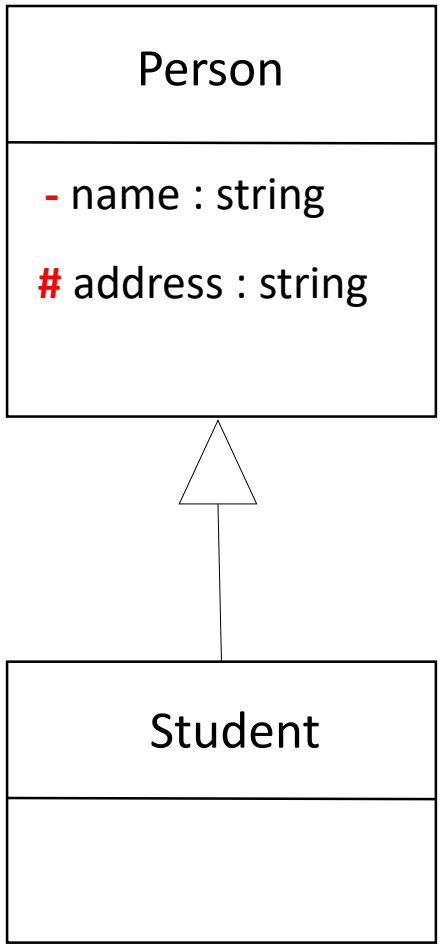
PRIVATE attributes are visible only inside the CLASS

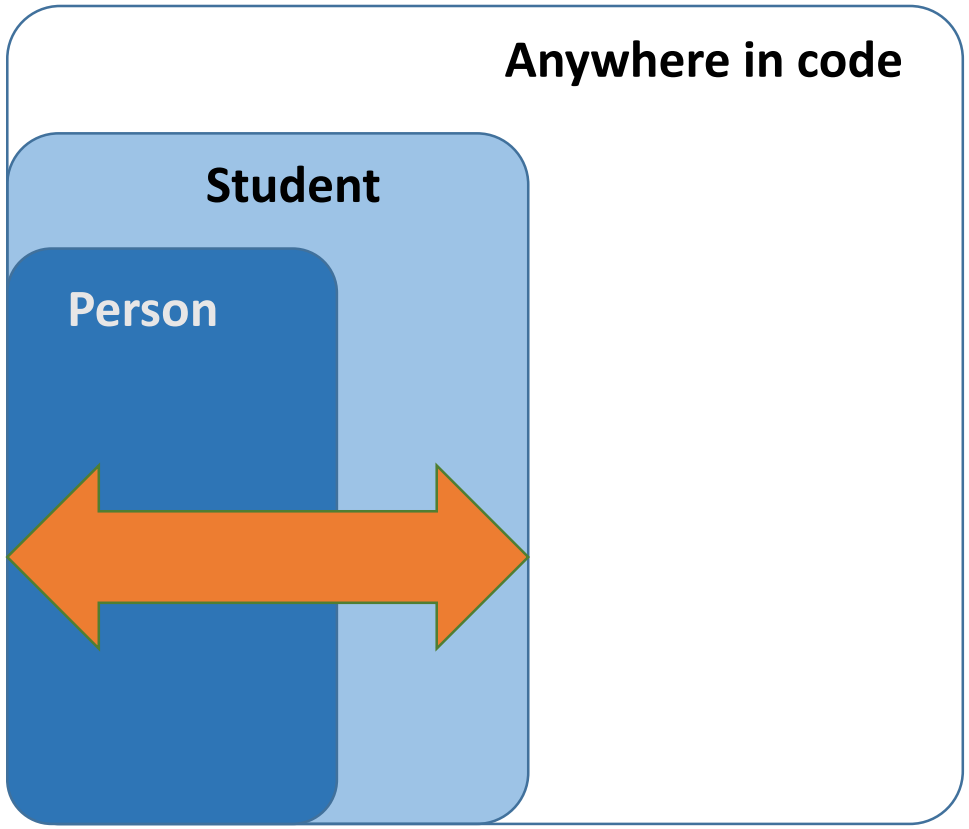




Visibility : PRIVATE > PROTECTED

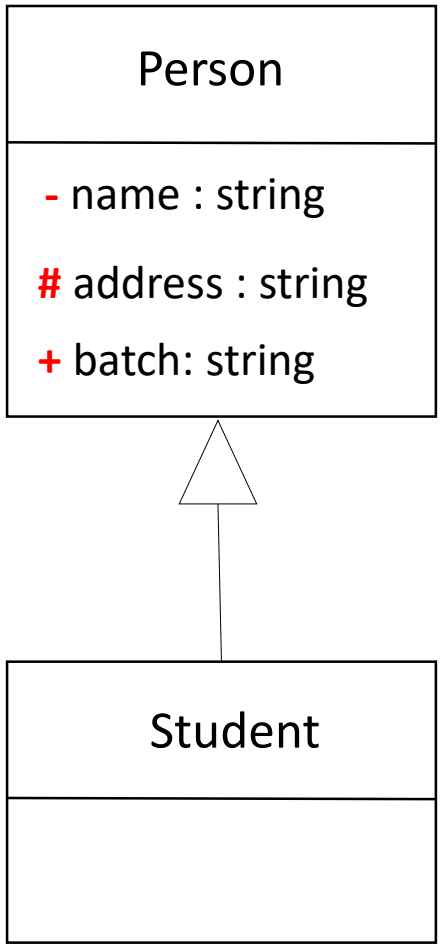
PROTECTED attributes are visible inside CLASS but also children CLASSES

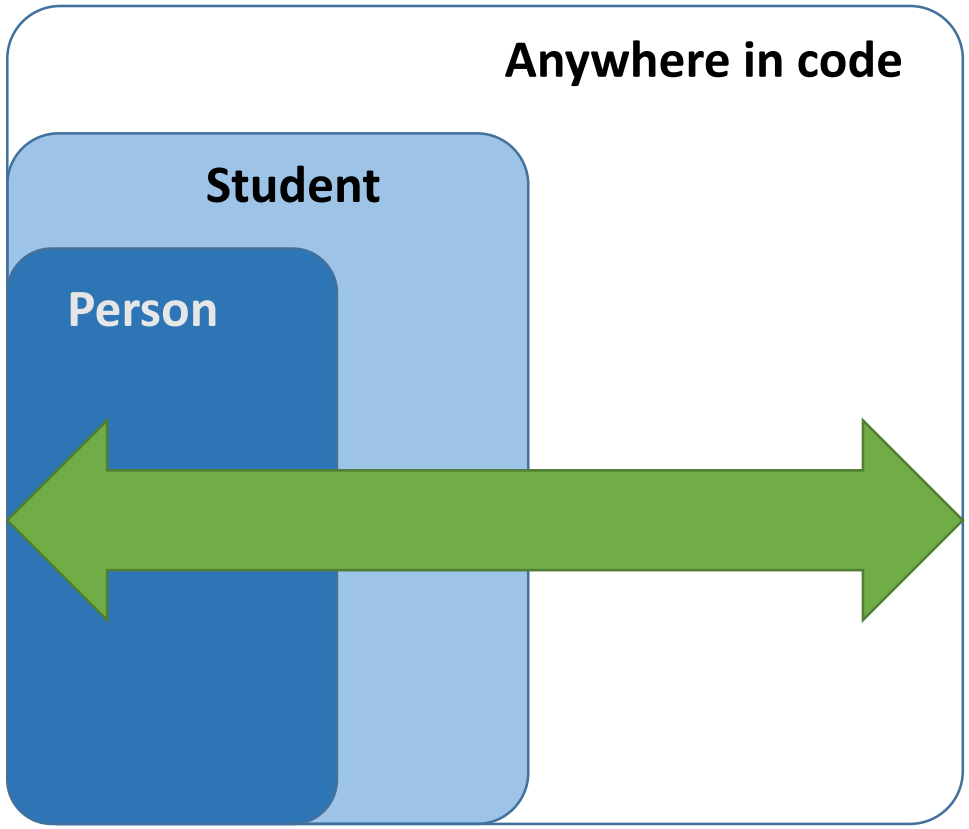




Visibility : PRIVATE > PROTECTED > PUBLIC

PUBLIC attributes are everyrywhere





ACTIVITY 3

15 MIN

This time the color is NOT an attribute
But it's a computed value !!!

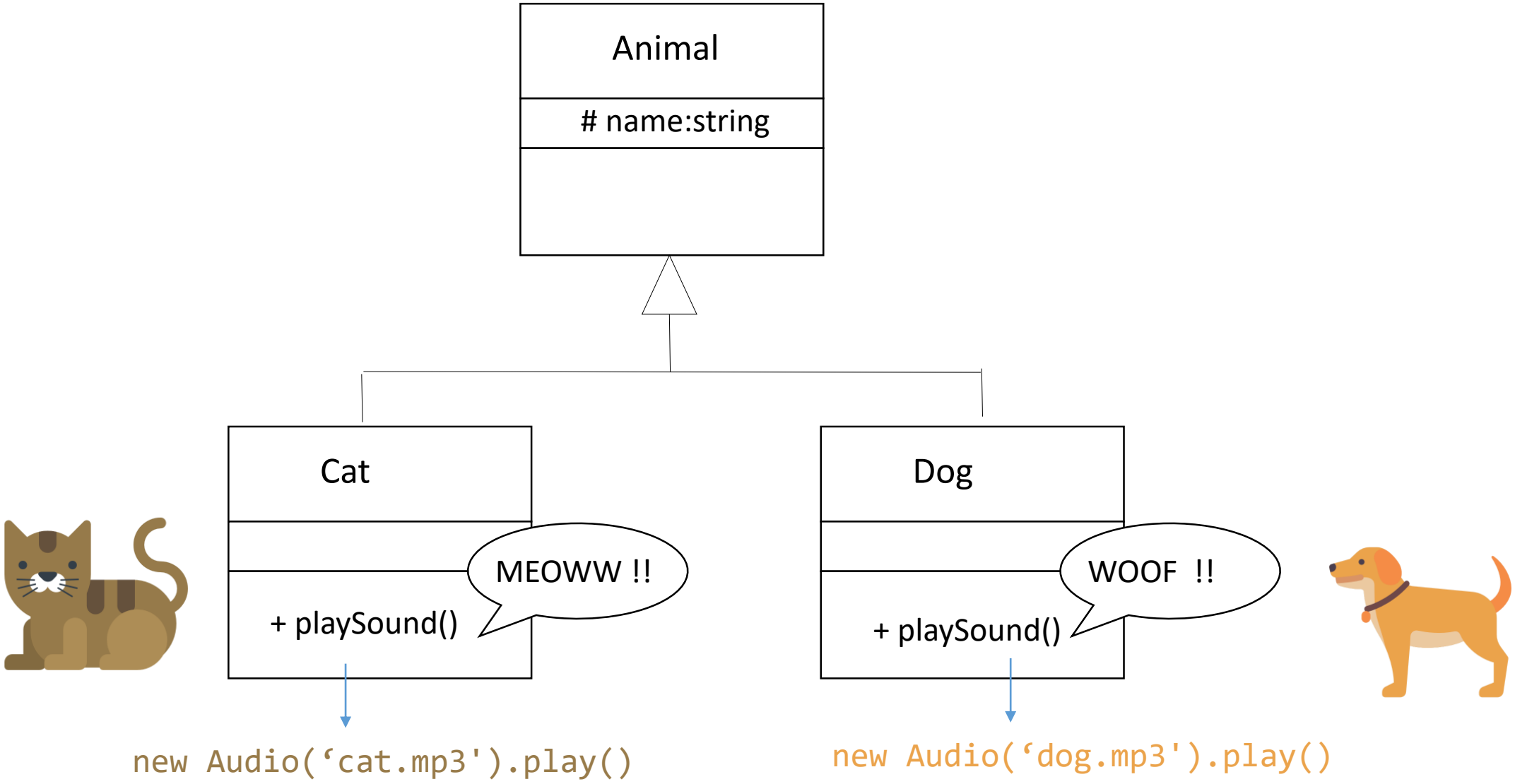
Code the method getColor: following the instructions

```
import { Point } from "./Point";

export class ColoredPoint extends Point {
  constructor(x: number, y: number) {
    super(x, y);
  }

  /**
   * Depending on the point position:
   * - if x= 0 => green
   * - else if y= 0 => red
   * - else => yellow
   * @returns the color of the point
   */
  getColor(): string {
    return "TODO";
  }
```

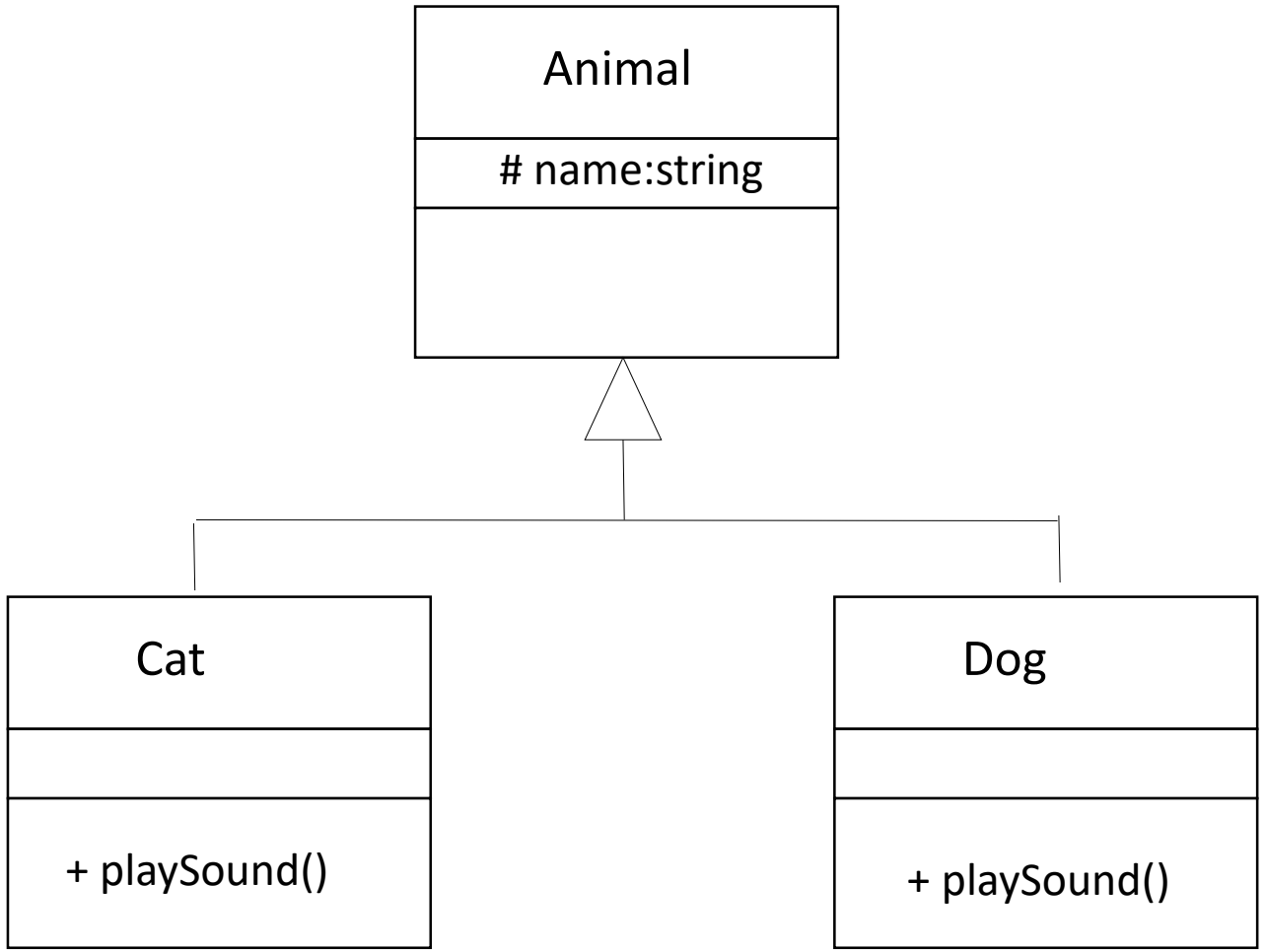
Abstract classes

We want to play the sounds of animals…

So we create a class **Animal** and 2 sub classes :

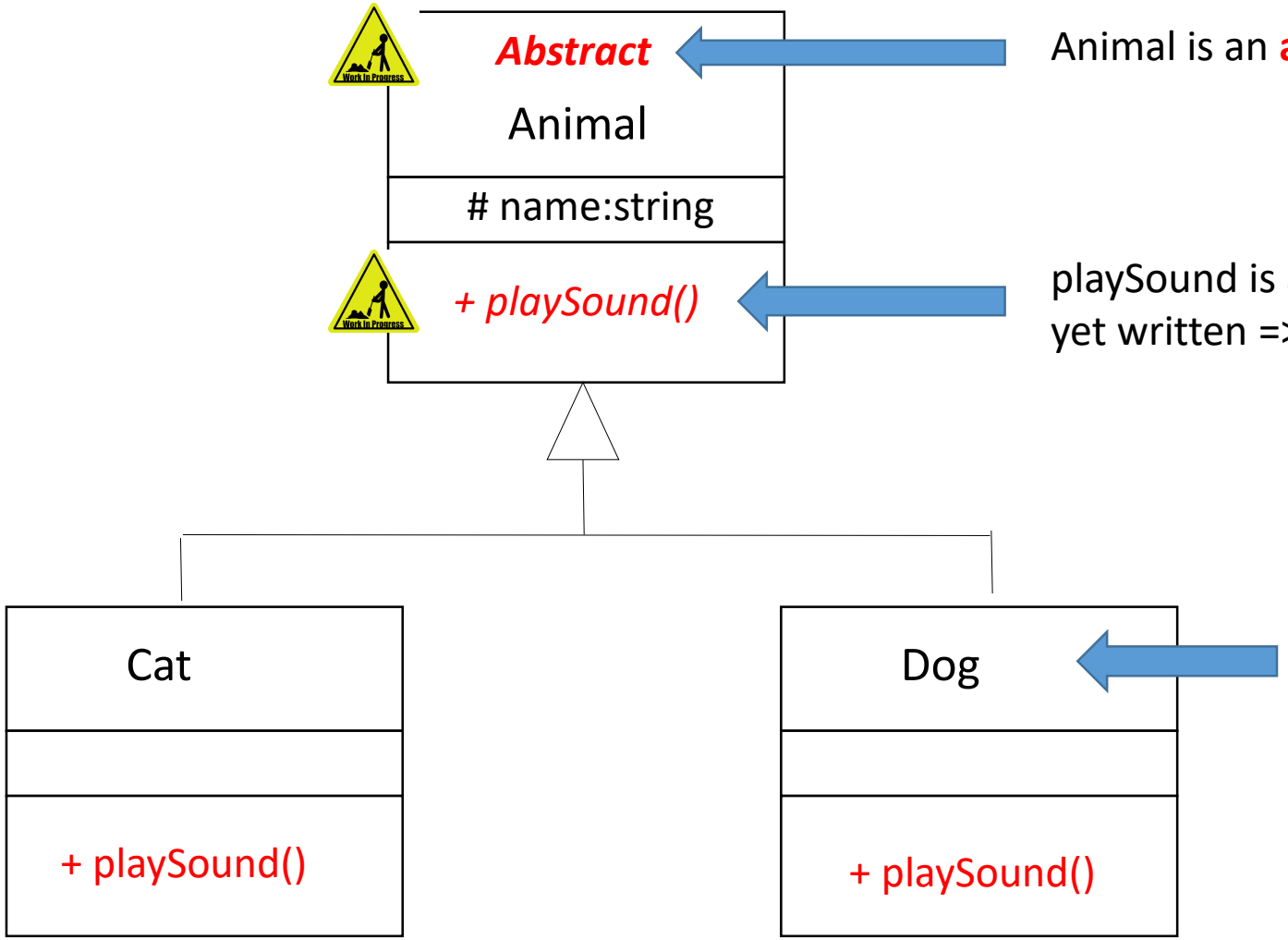


From a list of animals : can we play the sounds?




```
Let animals : Animal[]= [];

// code to add animals..

for(animal of animals) {
  animal.playSound();
}
```

Let’s add playSound as an abstract method



Animal is an **abstract** class : it can contains **abstract methods**

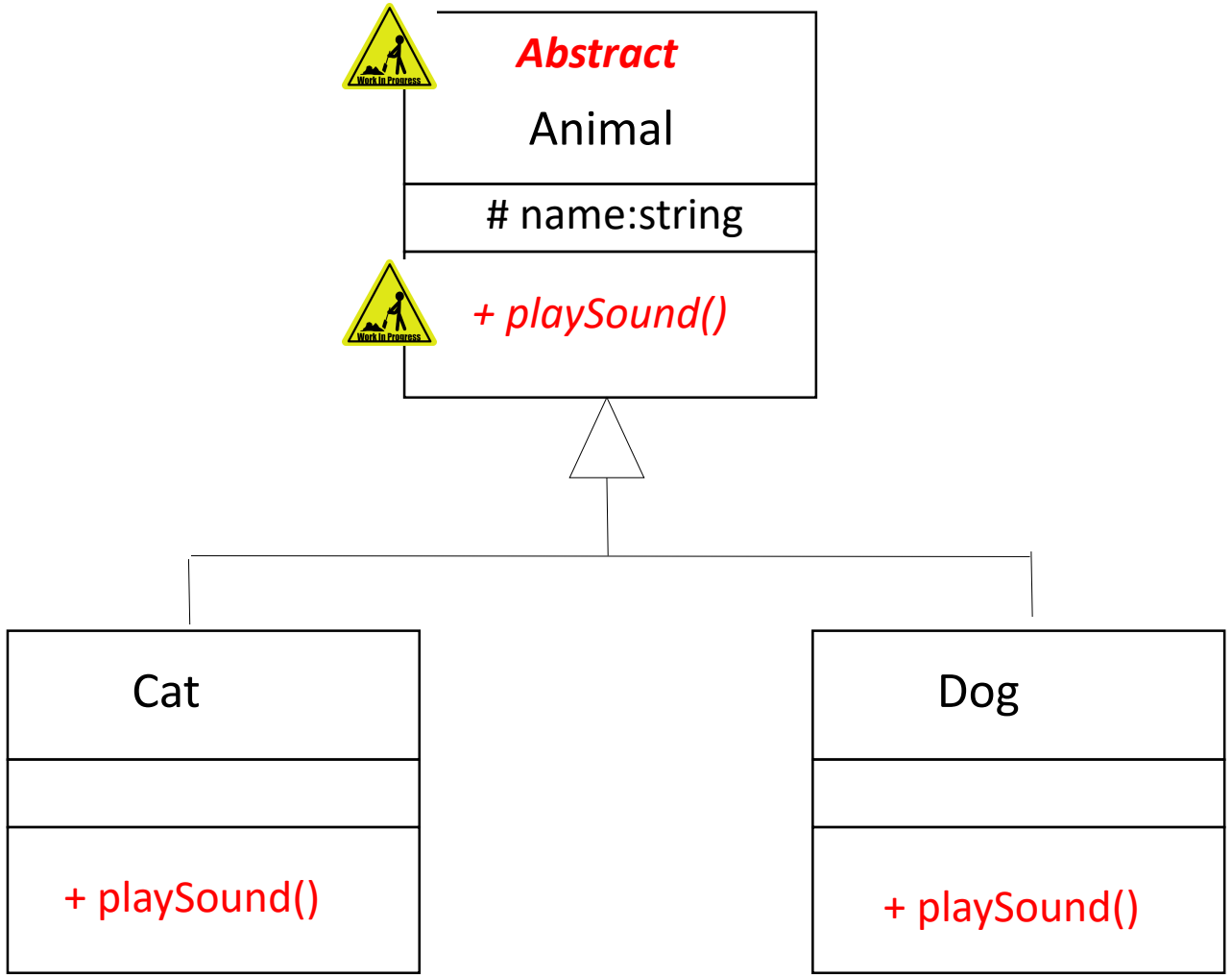
playSound is an **abstract** method: the body of the method is not yet written => the children will do it !

Class Dog **extends** the **abstract** class Animal

So : it MUST implement the method playSound()

DEMO - 3

Let’s code it




```
abstract class Animal {
  constructor(protected name: string) {}

  abstract playSound(): void;
}

class Cat extends Animal {
  constructor(name: string) {
    super(name);
  }

  playSound() {
    console.log("meooww !");
  }
}
```

ACTIVITY 4

15 MIN

Operation is an abstract class to perfrom
A operation between 2 numbers

Code 2 classes :
- One to add the 2 numbers
- One to multiply the 2 numbers

```
abstract class Operation {
  constructor(public number1: number, public number2: number) {}

  abstract doOperation(): number;
}

// TODO 1 : Create a class AddOperation, which extends Operation
//          doOpenration must return the sum of the 2 numbers
//
// TODO 2 : Create a class MultiplyOperation, which extends Operati
//          doOpenration must return the multiplication of the 2 nu
```

KNOW I KNOW :

- ✓ **How** to **inherit** from class and **why**
- ✓ The meaning of **Super** in constructor
- ✓ The meaning of **Super** in methods
- ✓ The **visibility** : private > protected > public
- ✓ What is an **abstract** class and **abstract methods**