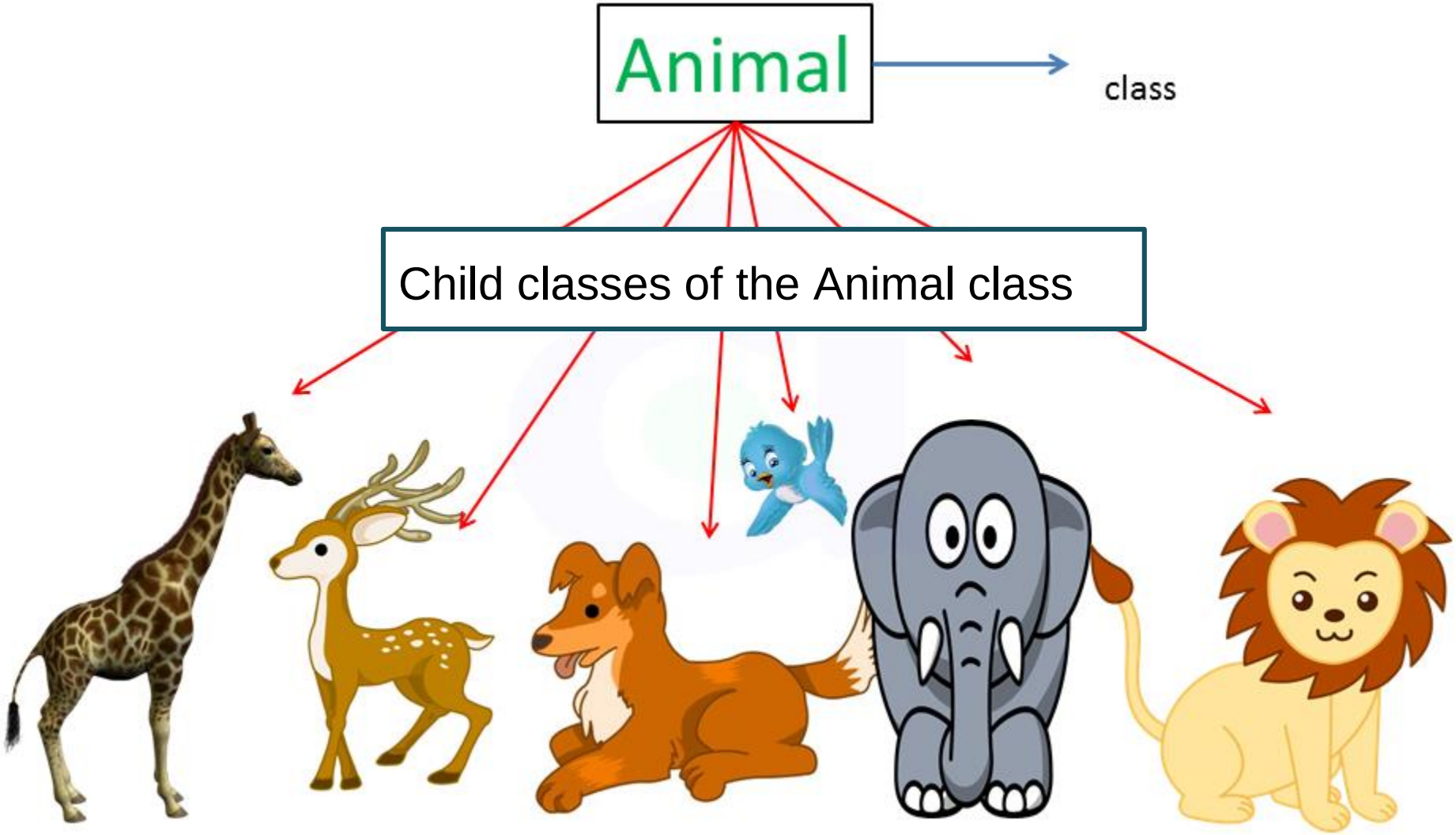
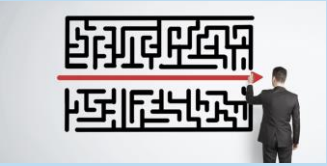
OOP - Inheritance



OOP

Reminder – Our Motivation to know OOP

- ★ Simplicity



- ★ Maintainability

- ★ Reusability

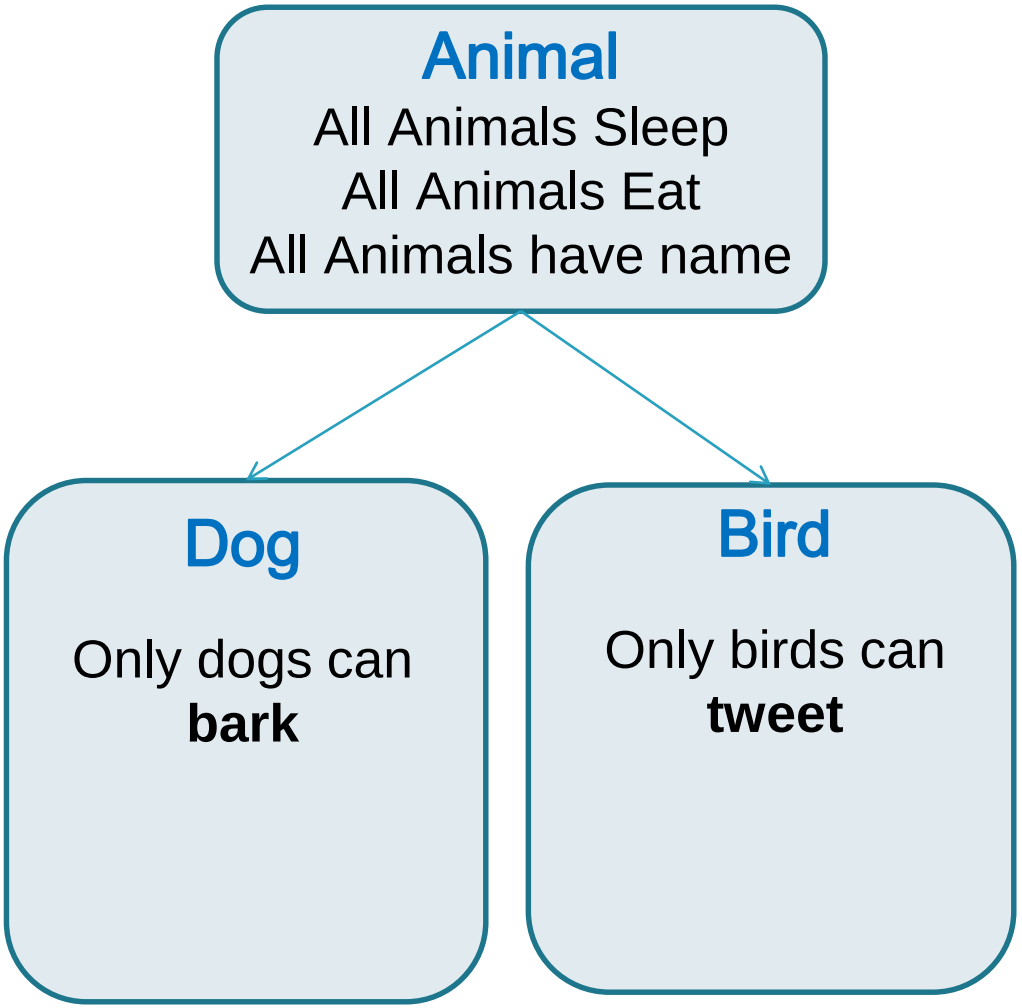
- ★ Flexibility

OOP - Inheritance

Motivation

If we think about the animal example, some attributes are similar in all animals, but some are different.

Barking, for example is a unique behavior, but it is similar for a sub group of animals – all the dogs.



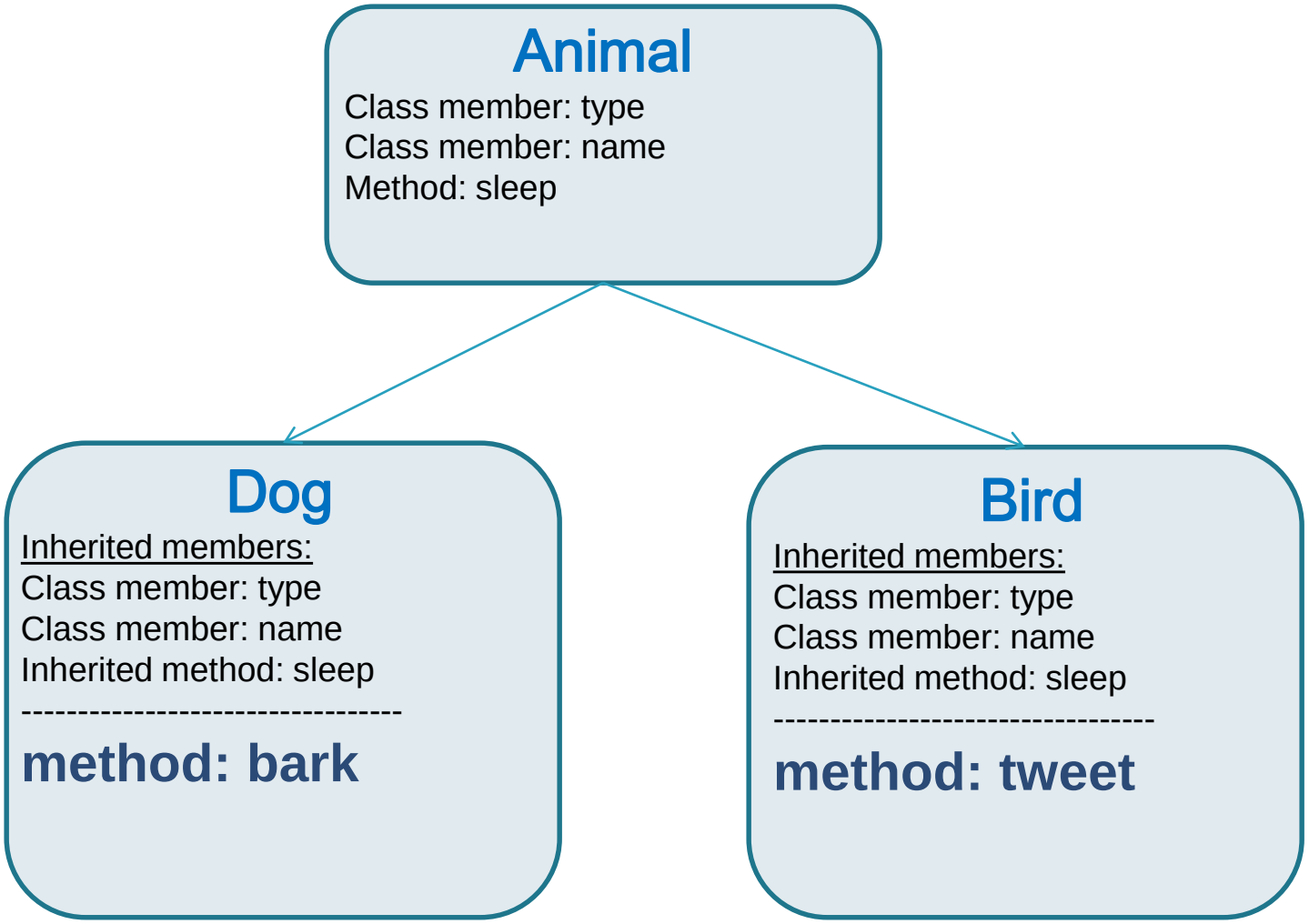
Our Animal Class

```
class Animal {
  constructor(name, type) {
    this.name = name;
    this.type = type;
  }

  sleep() {
    console.log("I fell asleep");
  }
}
```

Another Model

Inheritance



OOP - Inheritance

```
class Dog extends Animal {
  constructor(name, sound){
    super(name, "dog");
    this.bark_sound = sound;
  }

  bark () {
    console.log(this.name + " barks: " + this.bark_sound);
  }
}
```

Saved word.
Dog class inherits the Animal members and methods.

A new dog method.
A dog instance can bark
an animal instance cannot.

OOP - Inheritance

```
class Dog extends Animal {
  constructor(name, sound){
    super(name, "dog");
    this.bark_sound = sound;
  }

  bark () {
    console.log(this.name + " barks: " + this.bark_sound);
  }
}
```

We already know how to create an instance:

```
let bubu = new Dog("Bubu", "oof oof");
```

OOP - Inheritance

```
class Dog extends Animal {
  constructor(name, sound){
    super(name, "dog");
    this.bark_sound = sound;
  }


  bark () {
    console.log(this.name + " barks: " + this.bark_sound);
  }

}
```

Super

Super is a reference to the parent class.
In this case we use super to execute the parent constructor.

OOP - Inheritance

```
class Dog extends Animal {
  constructor(name, sound){
    super(name, "dog");
    this.bark_sound = sound;
  }


  bark () {
    console.log(this.name + " barks: " + this.bark_sound);
  }

}

let bubu = new Dog("Bubu", "oof oof");
```

Super

Calling the super constructor has to be the first line in the constructor!

Assigning Dog specific attributes

OOP - Inheritance

```
class Dog extends Animal {

  constructor name, sound {
    super(name, "dog");
    this.bark_sound = sound;
  }

  bark () {
    console.log(this.name + " barks: " + this.bark_sound);
  }
}
```

Passing the constructor both the parent arguments and the child arguments

Parent argument is sent to the parent constructor

Child argument is being assigned to the instance

OOP - Inheritance

```
class Dog extends Animal {

  constructor(name, sound){
    super(name, "dog");
    this.bark_sound = sound;
  }

  bark () {
    console.log(this.name + " barks: " + this.bark_sound);
  }
}
```

We already know that type will be dog, so we don't need to pass it as a parameter!

Exercise - Cat

Here is an Animal class:

```
class Animal {
   constructor(name, type) {
      this.name = name;
      this.type = type;
   }
}
```

Your task:

Create a `Cat` class that extends the `Animal` class. Additionally It has an attribute `isFriendly`.

It also has a method describe that will print:
" I am <cat name> and I <am/am not > a friendly cat "

Create a cat instance and invoke the describe method.

Dog and Bird Implementations

```
class Dog extends Animal {
  constructor(name, sound){
    super(name, "dog");
    this.bark_sound = sound;
  }

  bark () {
    console.log(`${this.name} barks: ${this.bark_sound}`);
  }
}
```

```
class Bird extends Animal {
  constructor(name){
    super(name, "bird");
  }
  tweet () {
    console.log("tweet tweet!!");
  }
}
```

Exercise:
In pairs explain to each other what does it mean for one class to extend another class. You can use the animal examples.

Instance

Let's create some animals!

```
let dambo = new Animal("Dambo", "elephant");
dambo.name;//Dambo
dambo.type;//elephant

let tweety = new Bird("tweety");
tweety.name;// tweety
tweety.type;//bird
tweety.tweet();//tweet tweet!!

let bubu = new Dog("Bubu", "oof oof");
bubu.name;//Bubu
bubu.type;//dog
bubu.bark();//Bubu barks: oof oof
```

Inherited properties

Bird method

Dog method

Parent vs Child Methods

```
class Animal {
  constructor(name, type) {
    this.name = name;
  }

  sleep() {
    console.log("I fell asleep");
  }
}
```

Parent method

```
class Bird extends Animal {
  constructor(name){
    super(name, "bird");
  }
  tweet () {
    console.log("tweet tweet!!");
  }
}
```

Child method

Instance

Let's create some animals!

```
let dambo = new Animal("Dambo", "elephant");
dambo.name;//Dambo
dambo.type;//elephant
dambo.sleep();

let tweety = new Bird("tweety");
tweety.name;// tweety
tweety.type;//bird
tweety.tweet();//tweet tweet!!
tweety.sleep();
```

Animal method

Inherited properties

Bird method

Animal method

Exercise - Coffee

Here is a Drink class:

```
class Drink {
  constructor(title, isHot){
    this.title = title;
    this.isHot = isHot;
  }
}
```

Your task:

1. Create a Coffee class that extends the Drink class. Additionally It has attributes for: number of coffee shots, with/without milk.
2. It also has a method that prints the instructions. For example, for an espresso it would print:

   "to make an espresso add 1 coffee shots and don't add milk".

   if the drink is hot it will also add a warning: "be careful it is hot".
3. Create an instance of the coffee class and call the method that prints the instruction.

OOP - Inheritance

Test for an instance class

Instanceof

Tests whether the specified constructor appears anywhere in the inheritance hierarchy of the object

```
let dog = new Dog("Blacky", "arf arf");

dog instanceof Dog;
// true
dog instanceof Animal;
// true
dog instanceof Bird;
// false
```

OOP - Inheritance

Example – Count the zoo's animals

```
let animals = [new Dog("Blacky", "arf arf"), new Bird("tweety"), new
Animal("Bagheera", "panter")];


let dogsCount = 0, birdCount = 0, othersCount = 0;
animals.forEach(element => {
  if (element instanceof Dog){
     dogsCount++;
  } else if (element instanceof Bird){
     birdCount++;
  } else {
     othersCount++;
 }
console.log(`In the zoo we have ${dogCount} dog(s), ${birdCount} bird(s)
              and ${othersCount}` other animal(s)`);
   // In the zoo we have 1 dog(s), 1 bird(s) and 1 other animal(s)
});
```

Is the animal a dog?

Is the animal a bird?

Code Design

Exercise: Let's design a Private lessons school. For simplicity we will say that each student will learn one subject.
Think Object based: we want to think who are the main players?
What are the main functionalities?
Let's start with basic attributes!

School
teachers: Array < Teacher>
students: Array < Student >

School Person
name: string
subjects: Array <string>

Student
name: string
subjects: Array <string> (size 1)

Teacher
name: string
subjects: Array <string>
maxStudents: int

Code Design

Question: How will we know which student and which teacher belong to which course?

School
teachers: Array < Teacher>
students: Array < Student >

School Person
name: string
subjects: Array <string>

Student
name: string
subjects: Array <string> (size 1)

Teacher
name: string
subjects: Array <string>
maxStudents: int
students: Array <students>

Code Design

Let's add methods

School
teachers: Array < Teacher>
students: Array < Student >

addStudent()
addTeacher()
assignStudent()

School Person
name: string
subjects: Array <string>

addSubject(subject)

Student
name: string
subjects: Array <string> (size 1)

Teacher
name: string
subjects: Array <string>
maxStudents: int
students: Array <students>

canTeach(student)

Let's see code!

```
class SchoolPerson {
  constructor(name, subjects) {
    this.name = name;
    this.subjects = subjects || [];
  }

  addSubject(subject) {
    if (this.subjects.indexOf(subject) === -1) {
      this.subjects.push(subject);
    }
  }
}
```

Let's see code!

```
class Teacher extends SchoolPerson {
  constructor(name, maxStudents, subjects) {
    super(name, subjects);
    this.maxStudents = maxStudents || 5;
    this.students = [];
  }

  canTeach(student) {
    return this.students.length<this.maxStudents &&
         this.subjects.includes(student.subjects[0]);
  }
}

class Student extends SchoolPerson {
  constructor(name, subjects) {
    super(name, subjects);
  }
}
```

Let's see code!

```
class School {
  constructor() {
    this.teachers = [];
    this.students = [];
  }

  addStudent(student){
    this.students.push(student);
  }
  addTeacher(teacher){
    this.teachers.push(teacher);
  }

  assignStudent(student) {
    for (let i = 0; i < this.teachers.length; i++) {
      let teacher = this.teachers[i];
      if (teacher.canTeach(student)) {
        teacher.students.push(student);
      }
    }
  }
}
```


Questions?

OOP Inheritance Cheat Sheet

Define a Class

```
class Animal {
  constructor(name, type){
    this.name = name;
    this.type = type;
  }

  sleep(){
    console.log(this.name + " fell asleep");
  }
}
```

Define a child class

```
class Dog extends Animal {
  constructor(name, sound){
    super(name, "dog");
    this.bark_sound = sound;
  }
  bark () {
    console.log(this.name + " barks: " + this.bark_sound);
  }
}
```

Check instance parent

```
element instanceof Dog;
```