CYDEO

# Abstraction

# OOP Principles

- There are 4 Object Oriented Programming (OOP) principles:
  - Encapsulation
  - Inheritance
  - Abstraction
  - Polymorphism

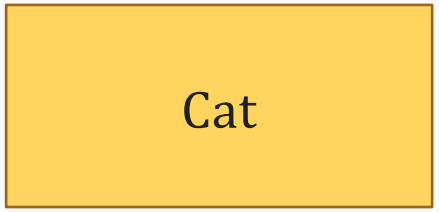
# Abstraction

- Process of hiding implementation details from the user
- Only the functionality will be provided to the user
- Focusing on the essential qualities of something rather than one specific example. (Ignoring the irrelevant & unimportant)
- User will have the information on what the object does instead of how it does

# Abstraction


eat():

eats dog food



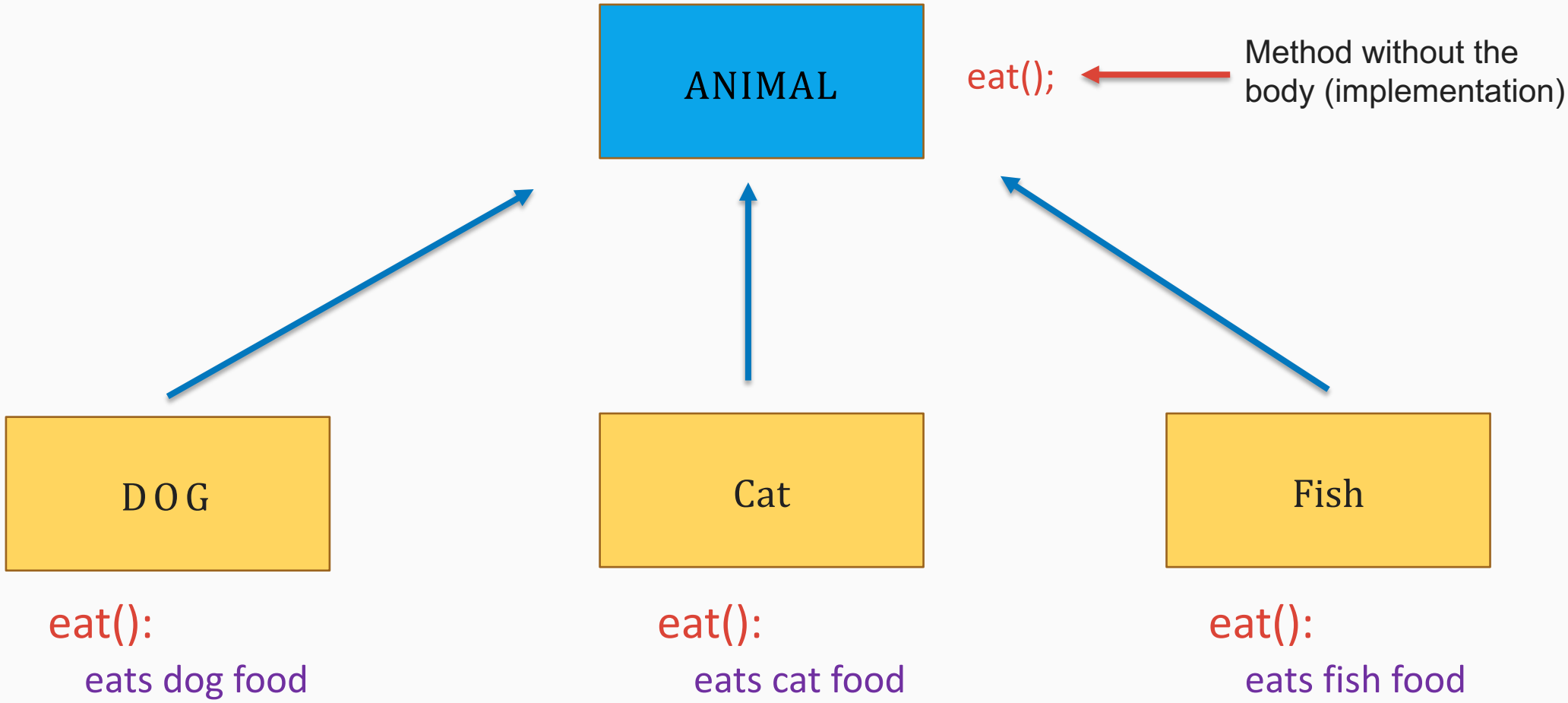

eat():

eats cat food

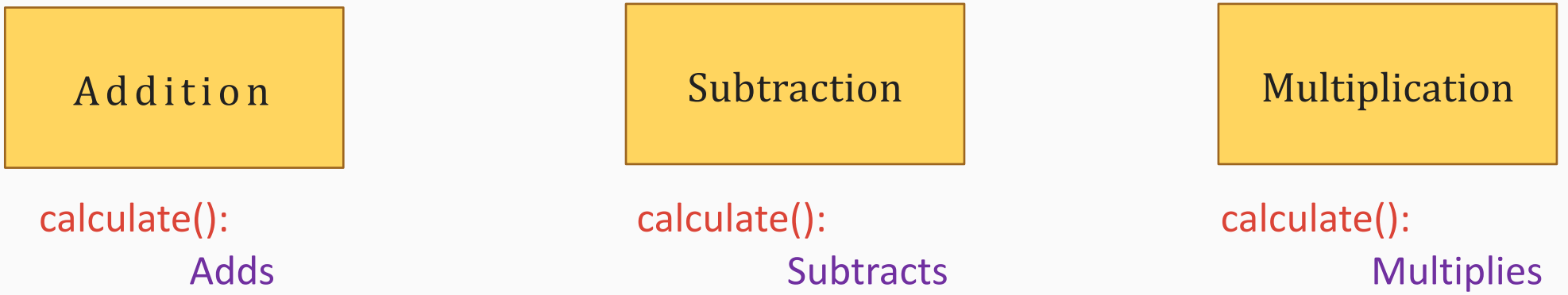

eat():

eats fish food

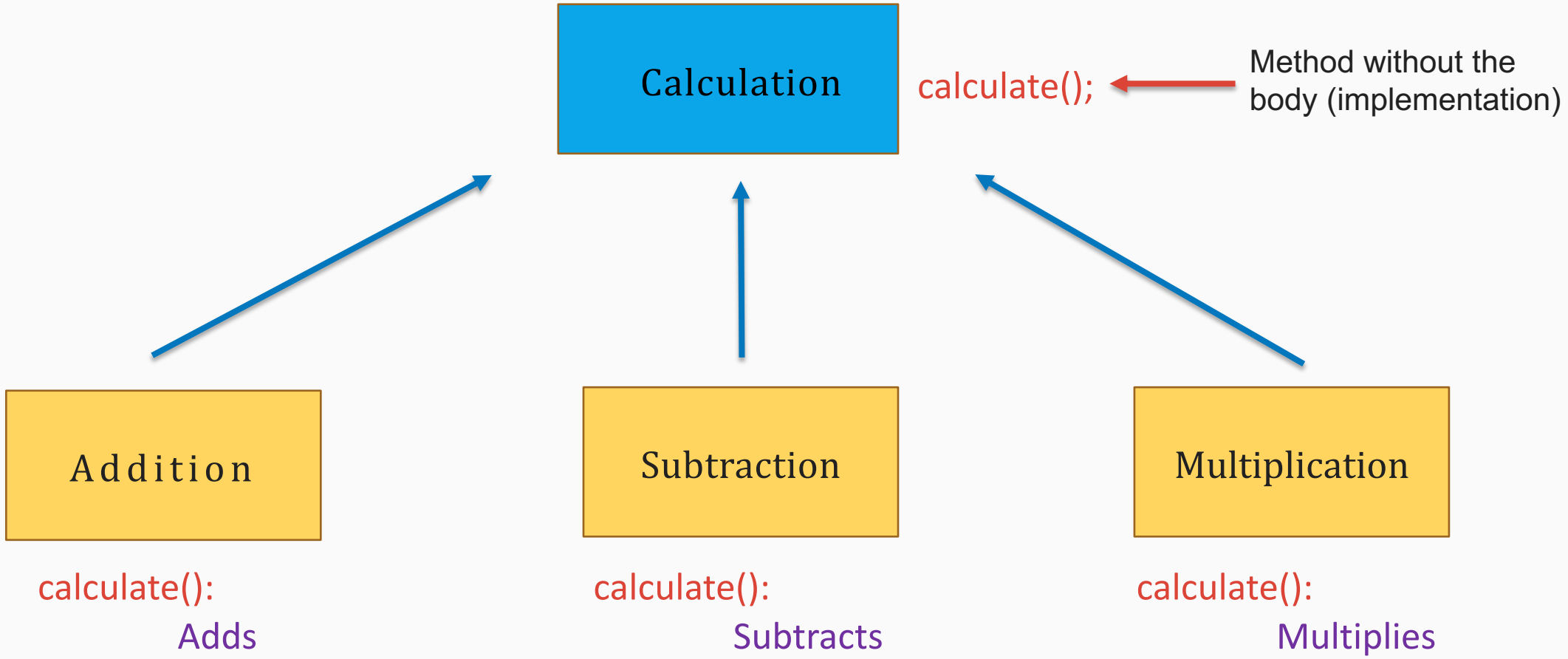
# Abstraction



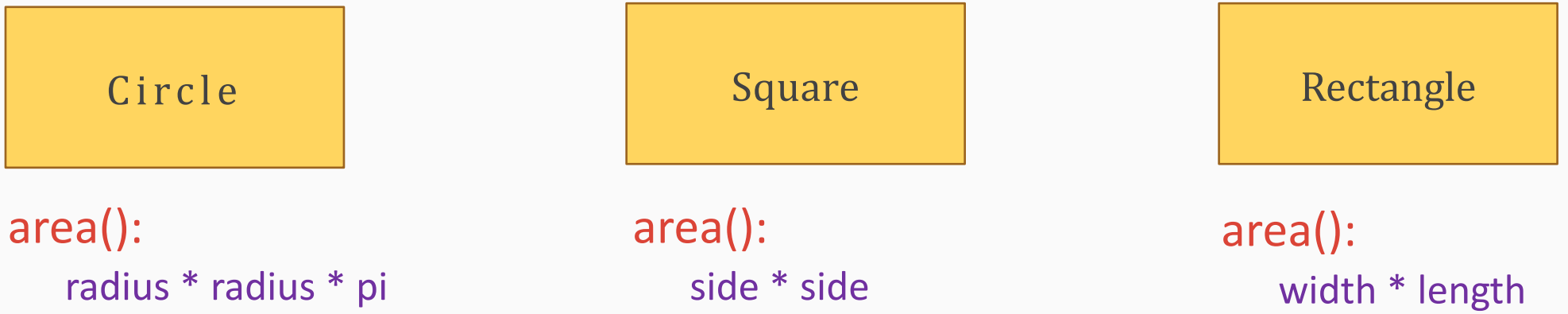
# Abstraction



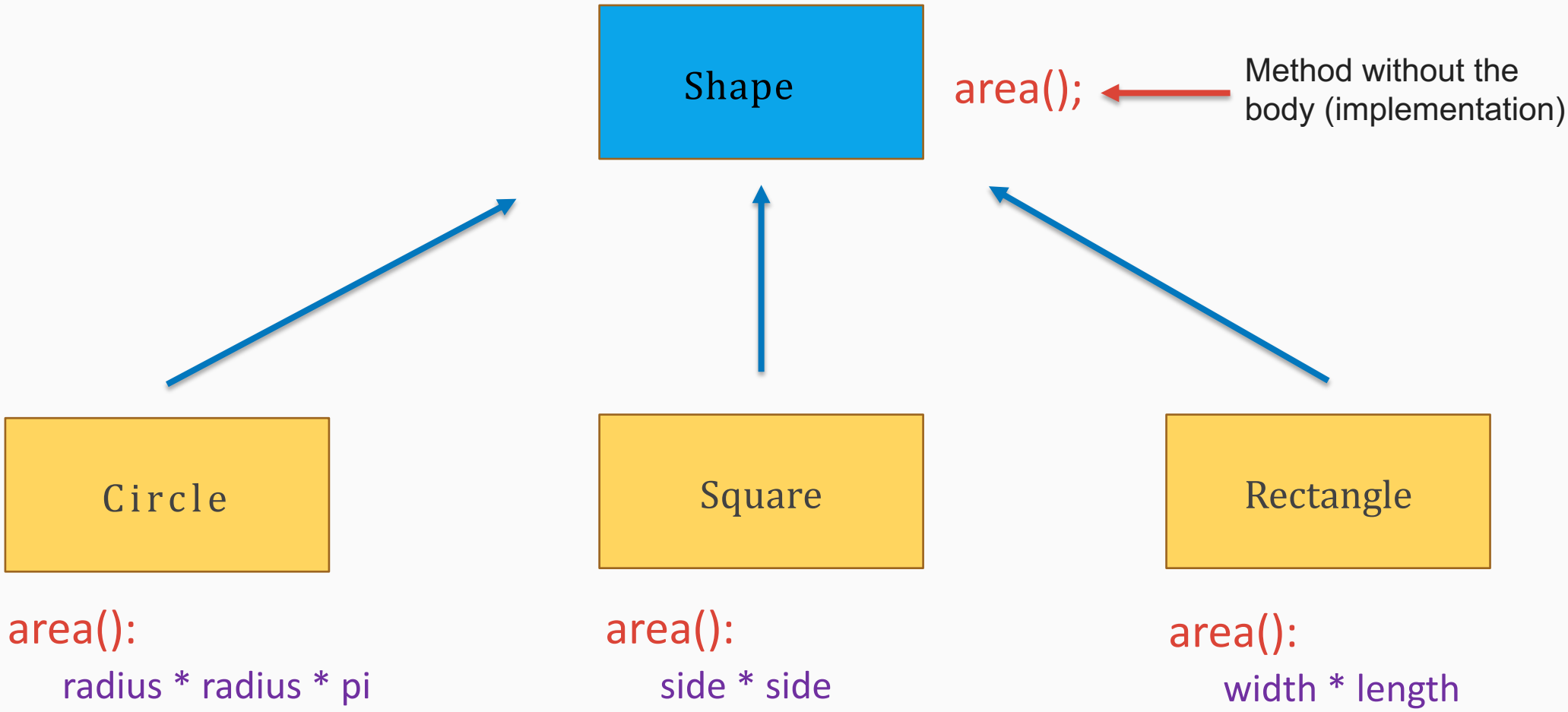
# Abstraction



# Abstraction



# Abstraction



# Abstract Method

- A method that does not have body, only signature
- A method that's meant to be overridden
- Abstract keyword is used to create abstract method

```
public abstract void eat();
```

Method without the body (implementation)

```
public abstract double area();
```

Method without the body (implementation)

# Abstract Method Rules

- An abstract method can not be static
- An abstract method can not be final
- An abstract method can not have private access modifier
- An abstract method does not have body
- An abstract method can only be created in an abstract class or in an interface

# Abstract Class

- A class that's meant to be a parent (super) class
- Goal is to provide reusable variables and methods to sub classes
- Abstract keyword is used to create the abstract class
- An abstract class can not be instantiated

```
public abstract class Animal{


}
```

# Abstract class & Abstract Method

```java
public abstract class Animal{

    public String breed;
    public char gender;

    public Animal(String breed, char gender){
        this.breed = breed;
        this.gender = gender;
    }

    public abstract void eat();

}
```

Abstract class

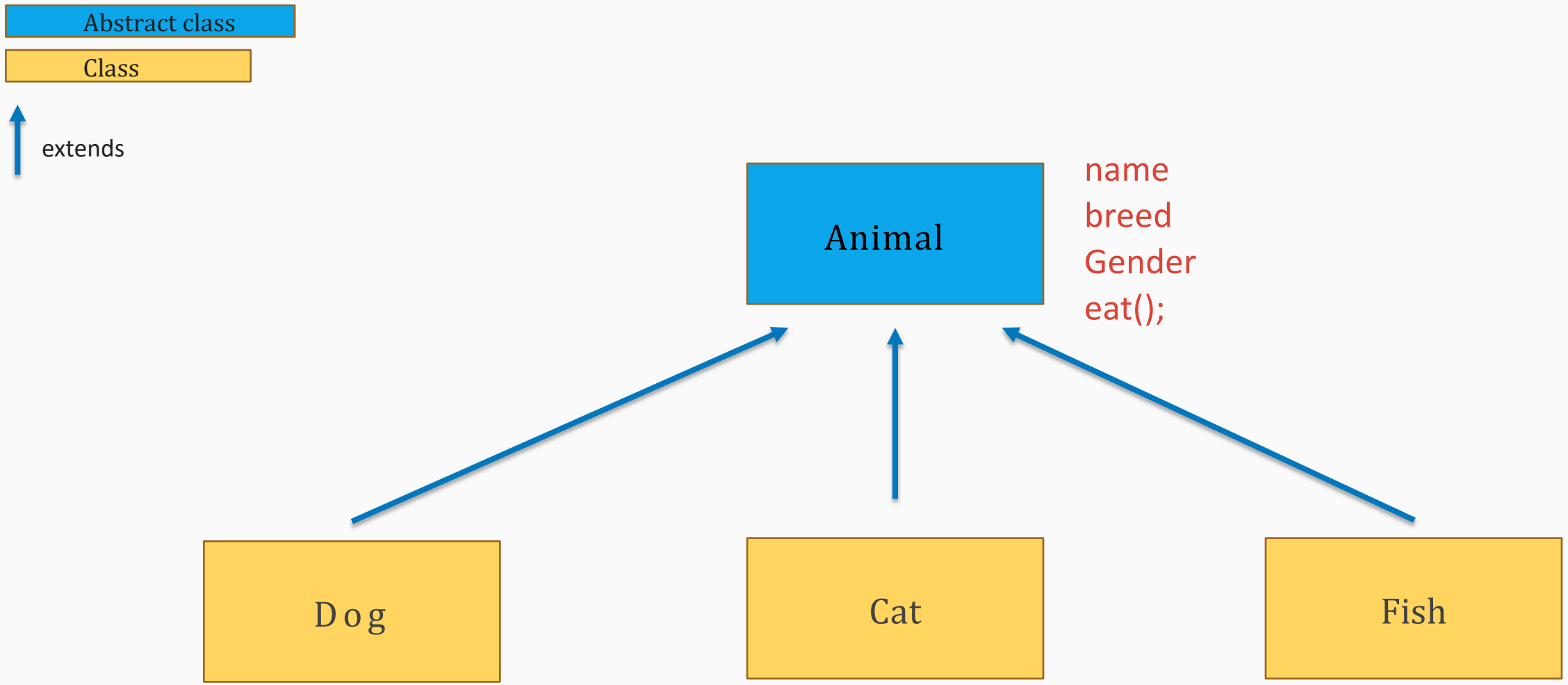
Super (parent) class is responsible to provide the variables and methods that are needed to the all the sub classes without worrying about the small details

Abstract class

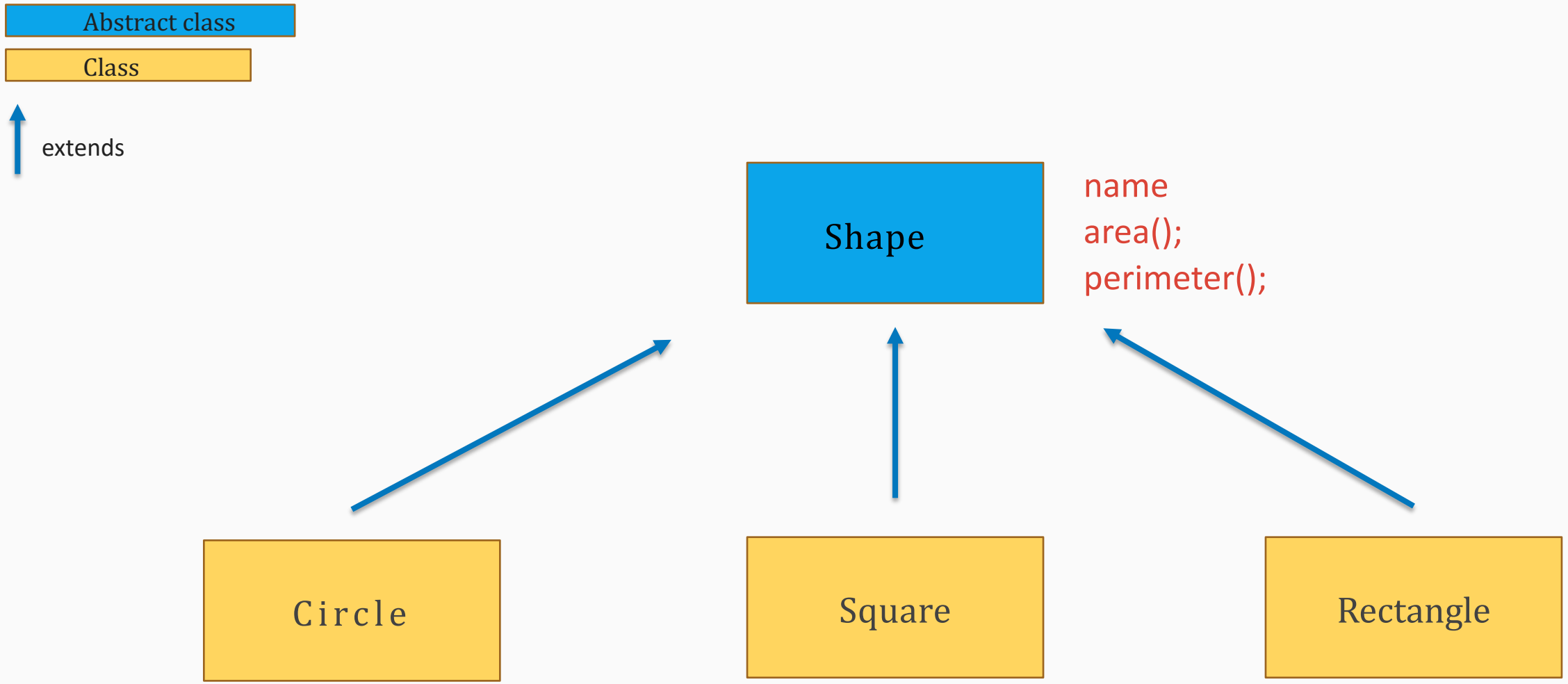
⚠ Sub (child) classes are responsible for providing the implementations that are needed

CYDEO

# Abstraction



# Abstraction



# Creating Object

- Abstract class is meant to be inherited only, not meant to be instantiated
- Abstract class is not a concrete class and object has to be concrete
- A sub class of abstract class is called concrete class, and it can be instantiated
- A Concrete class must implement all the inherited abstract methods

```
public abstract class Animal{

    public abstract void eat();

}

public class Dog extends Animal{

    @Override
    public void eat(){
        System.out.println("Dog eats dog foods");
    }


}
```

# Abstract Class vs Regular class (Concrete)

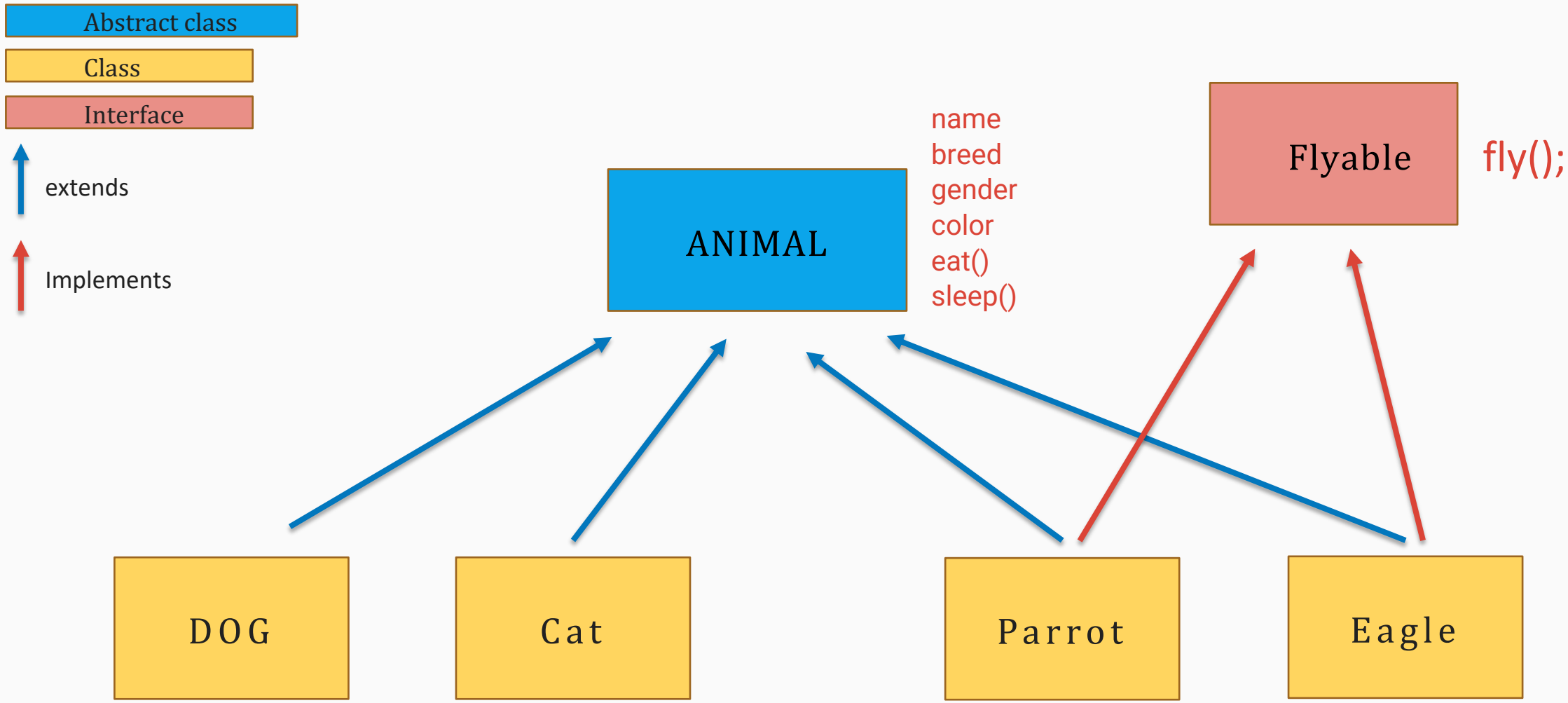
| Regular class | Abstract class |
| --- | --- |
| can have constructors, instances and statics | can have constructors, instances and statics |
| Regular class can be instantiated | Abstract class can not be instantiated |
| Regular class can not have abstract method | Abstract class can have abstract method |
| Regular class can be declared as final | Abstract class can not be declared as final |

# Interface

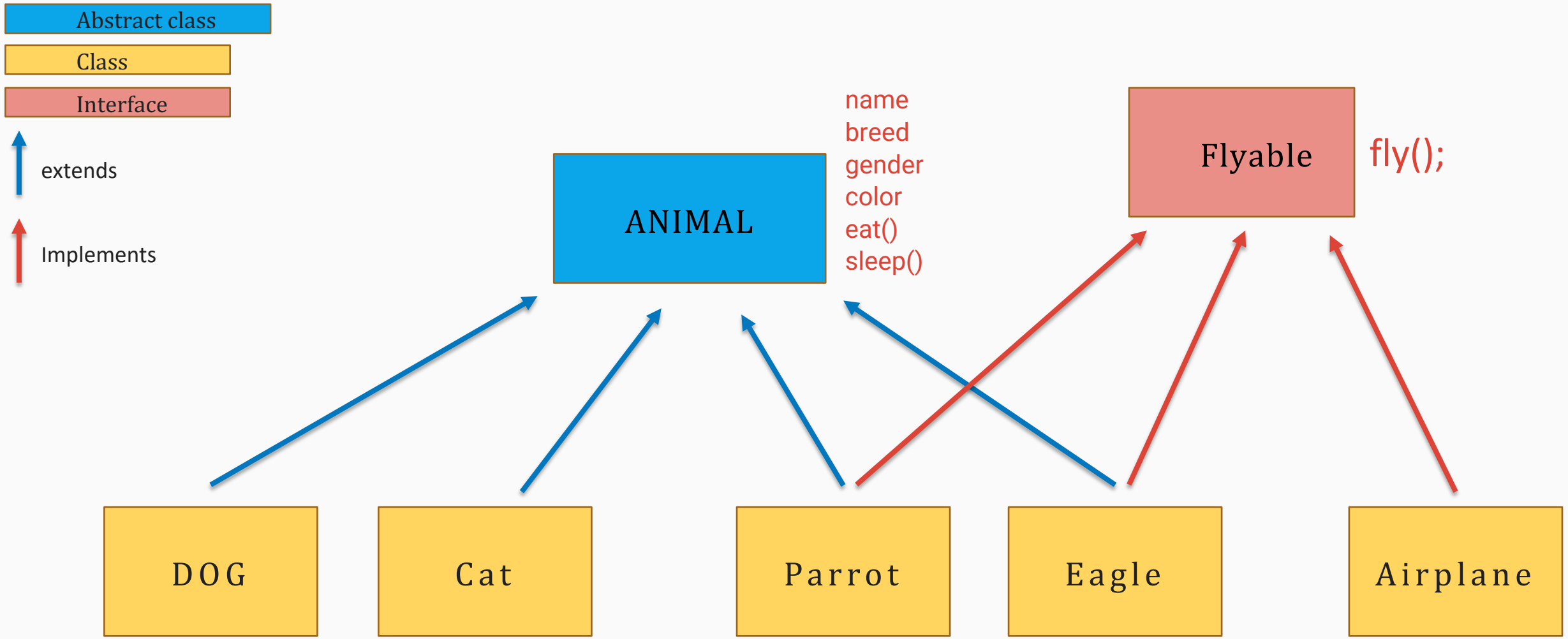
- It's a template, a blueprint of a class
- Interface keyword is used to create interface
- Specifies the behavior(s) that a class should implement
- Provides additional methods that subclass(es) need.
- We can achieve 100% abstraction using interfaces

```
public interface Flyable{

    public void abstract fly();

}
```

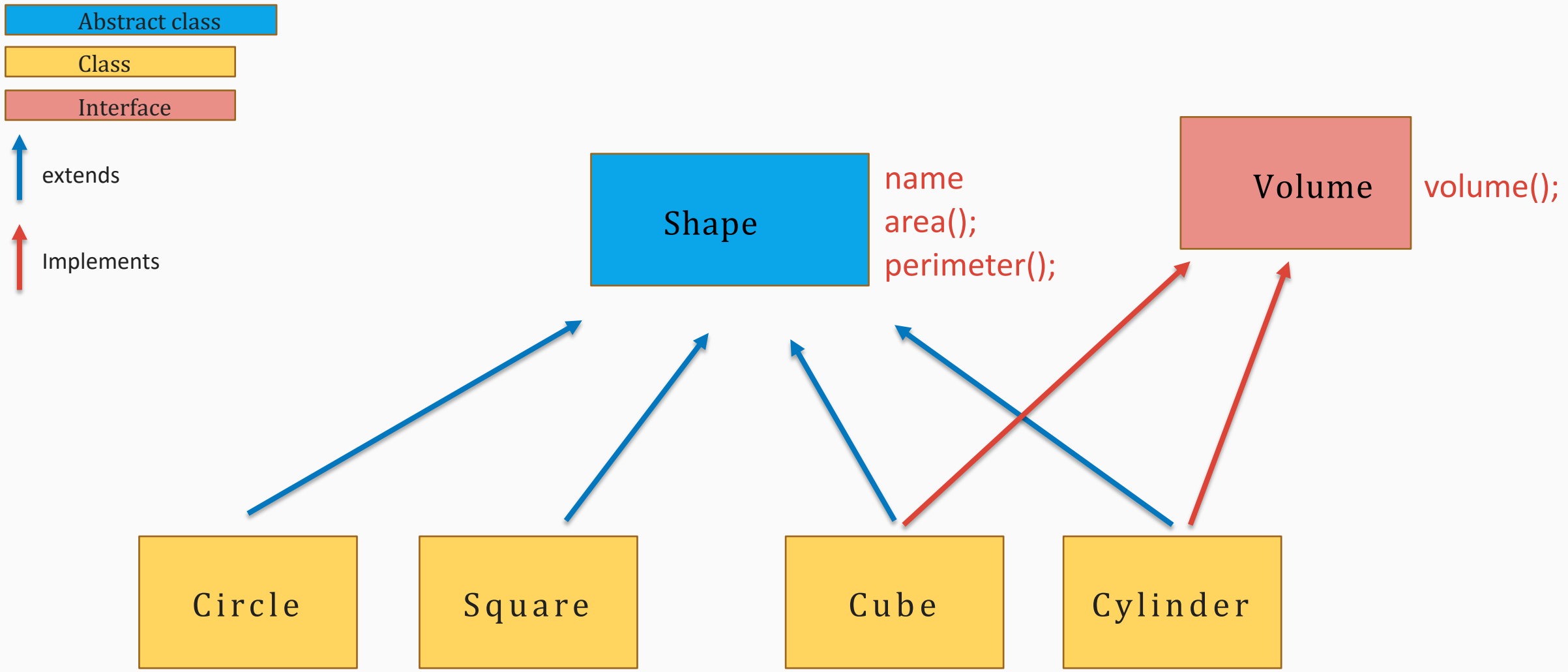
# Interface Example



# Interface Example



# Interface Example



# Properties of Interface

- Variables are static & final by default
- Interface can have static methods, abstract methods and default methods only
- Public is the only access modifier that can be used in interface and given by default
- Abstract keyword is given by default to the abstract methods of interface

```
public interface Interface1{

    int a = 100; // static & final by default

    void method1(); //abstract method

}
```

# What interface can have?

```java
public interface Interface1{

    int a = 100; // static & final

    void method1(); //abstract method

    static void method2(){ // static method
        System.out.println("Static Method");
    }

    default void method3(){ // default method
        System.out.println("Static Method");
    }


}
```

Variables: static & final variable only

Methods: static, abstract and default methods only

# What Interface can not have?

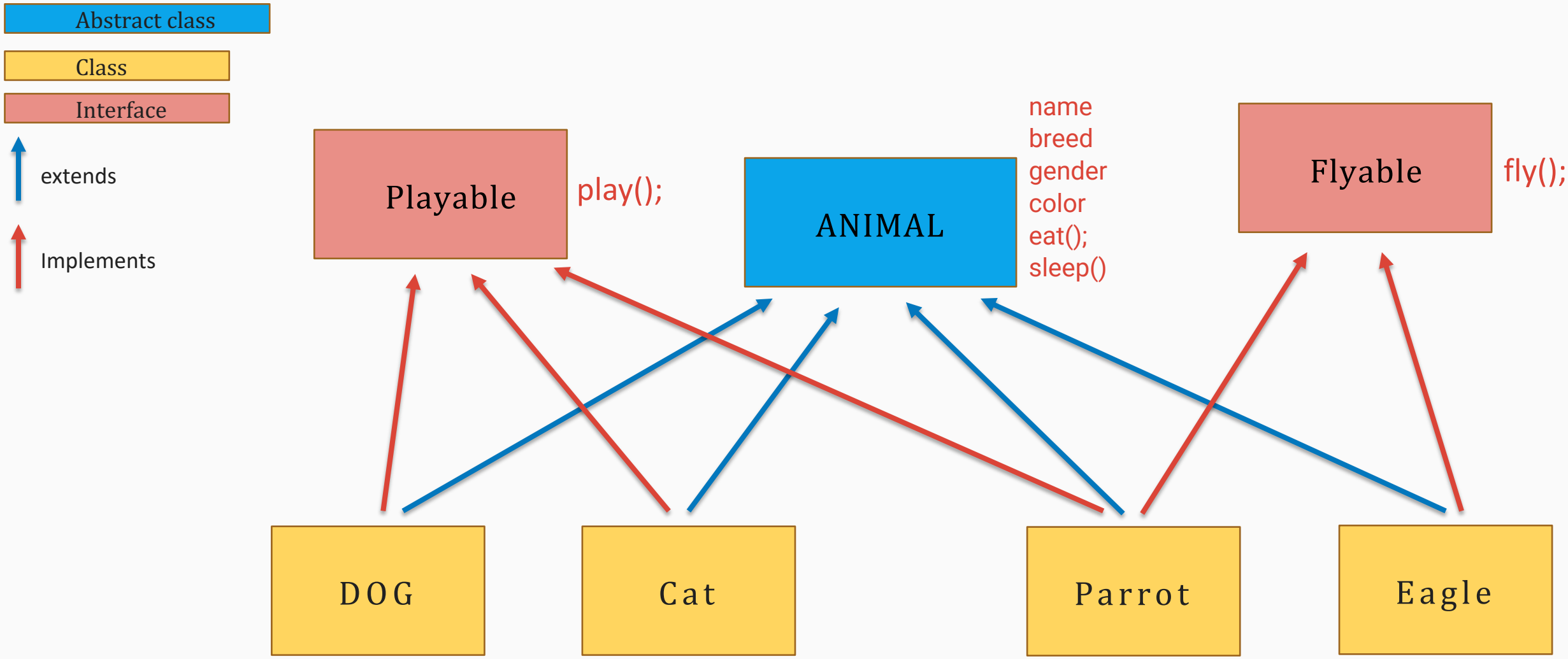
- An interface can not have instance variables
- An interface can not have instance methods
- An interface can not have constructors
- An interface can not have blocks
- We can not create objects from interface

# Implementing the Interface

- Class can extend only one superclass, but java allows a class to implement multiple interfaces
- Implements keyword is used to inherit from interface(s)
- When a class implements multiple interfaces, it must implement (override) all the abstract methods

```
public class MyClass implements Interface1, Interface2, Interface3{}
```

# Interface Example



# Implementing the interface

```java
public abstract class Animal{

    public String name;
    public String breed;
    public char gender;
    public String color;

    public abstract void eat();

}
```

```java
public interface Flyable{

    boolean canFly = true;

    void fly();

}
```

```java
public class Eagle extends Animal implements Flyable{

    @Override
    public void eat(){
        System.out.println("Eagle eats snake");
    }

    @Override
    public void eat(){
        System.out.println("Eagle can fly 50 kilometers/hour");
    }

}
```

# Implementing the interface

```
public abstract class Shape{

    public abstract double area();

    public abstract double perimeter();

}
```

```
public interface Volume{

    boolean hasVolume = true;

    double volume(); // abstract method

}
```

```
public class Cube extends Shape implements Volume{

    public double side;

    @Override
    public double area(){
        return 6 * side * side;
    }

    @Override
    public double perimeter(){
        return side * 12;
    }

    @Override
    public double volume(){
        return side * side * side;
    }

}
```

# Abstract Class vs Interface

| Abstract class | Interface |
| --- | --- |
| Can not be instantiated | Can not be instantiated |
| Multiple inheritance is not allowed | Multiple inheritance is allowed |
| Can have constructor | Can not have constructor |
| Can have instance, static and abstract methods | Can have static, abstract and default methods |
| Can have instance and static variables | Can only have static variable (final by default) |
| Can not be final | Can not be final |
| Can use other access modifiers than public | Can not use other access modifiers than public |