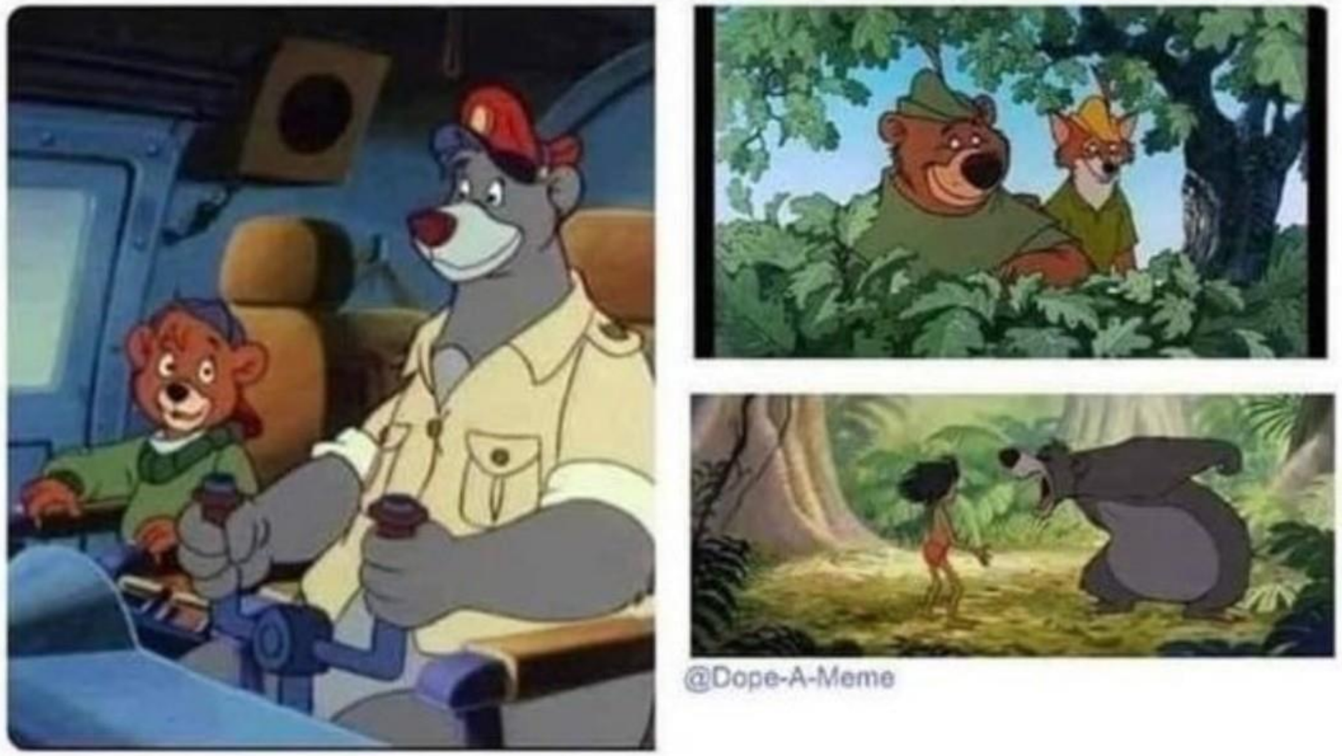
# Object Oriented Programming Objects

## Putting it all Together

And so, Dr. Baloo finds himself leaping from life to life, hoping each time that his next leap... will be the leap home.



## Phil Harris

- **Born**: June 24, 1904, Linton, IN
- **Died**: August 11, 1995, Rancho Mirage, CA

## Baloo

What defines him? (properties)

## What can Baloo do? (methods)

## Time to build a bear

## How can Baloo become other characters? (inheritance)

## Primary Concept of OOP

- **A**bstraction
- **P**olymorphism
- **I**nheritance
- **E**ncapsulation

## Abstraction

Handle and simplify complexity by hiding unnecessary details.

Generalize a specific code solution into a reusable code pattern

# Polymorphism

- Two types of polymorphism
  - Overload – same method name with different parameters
  - Override – replace an inherited property or method with new functionality

## Inheritance

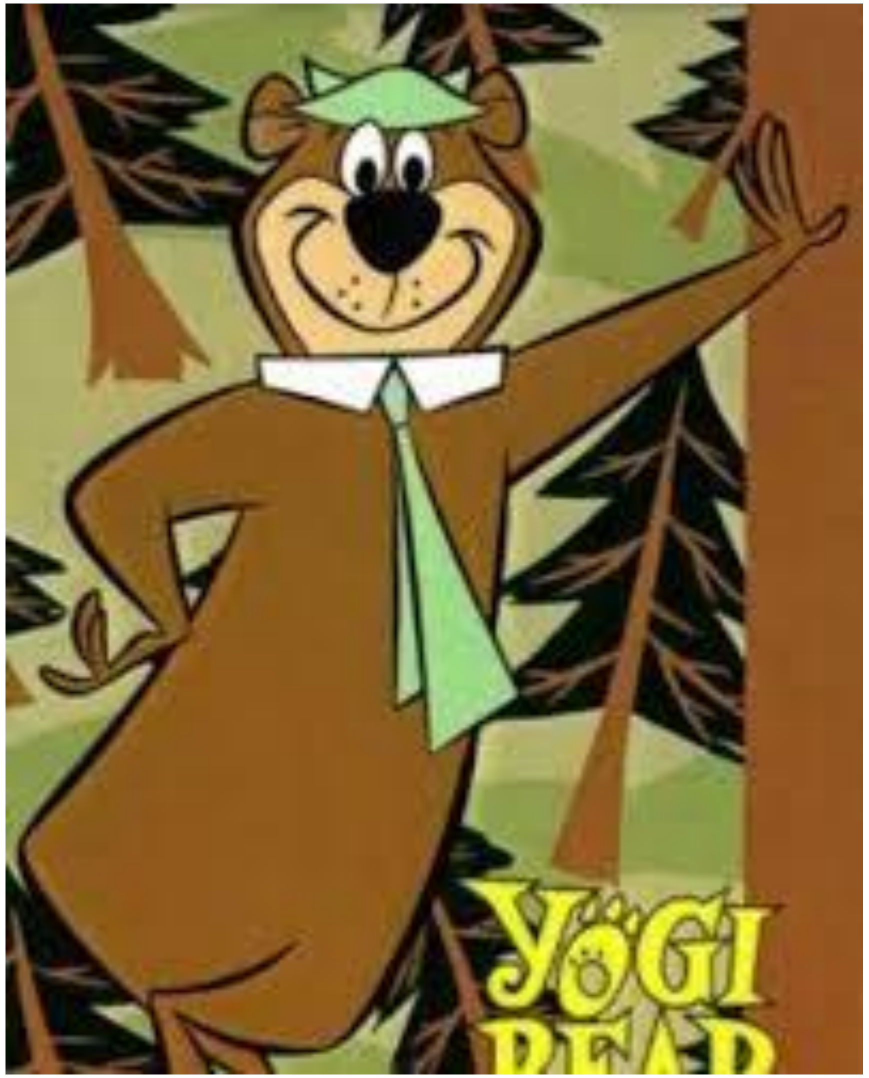
- Allows classes to “share” common properties and methods
- Different classes can be used with the same interface

## Encapsulation

- Properties and code are kept private
- Can only be accessed through public methods

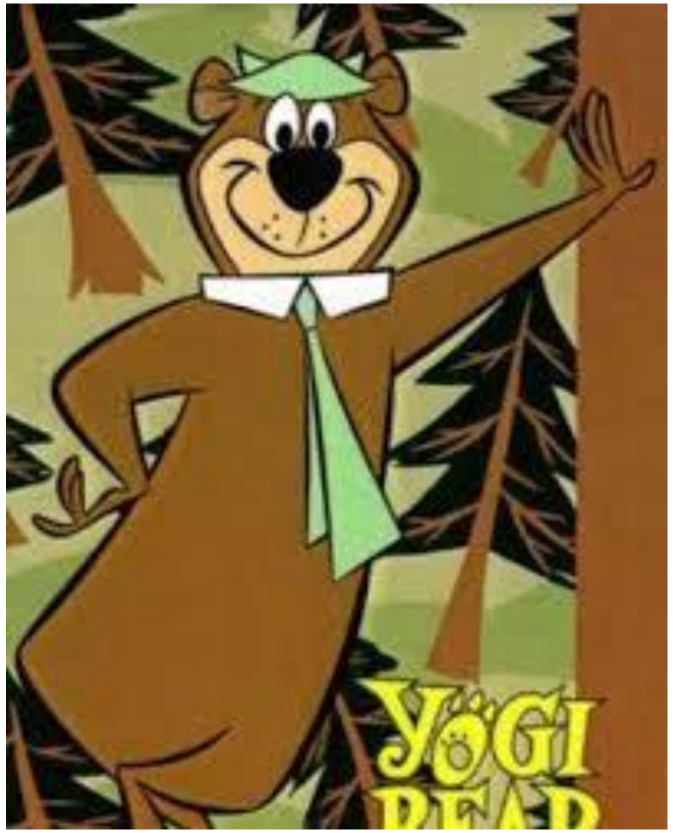
## Time to build some bears

## What if we add another different bear?



## What defines Yogi?

## What about adding more bears?



# Current Object Model

We are repeating code in each object

```
public class Baloo{}
```

```
public class Yogi{}
```

```
public class
PapaBerenstainBear{}
```

What is their commonality?

## Abstraction to the Rescue!

Interface, Abstract Class, Parent Class, or some combination?

## Interface

Can be thought of as a template of requirements with no code

Cannot be instantiated

Cannot contain a constructor

All properties and methods are public

All properties and methods must be implemented by the inherited object

## Abstract Class

Can be thought of as a base class

Can have defined constructor(s)

Cannot be instantiated by itself

Must be inherited by a class

Can inherit one abstract class

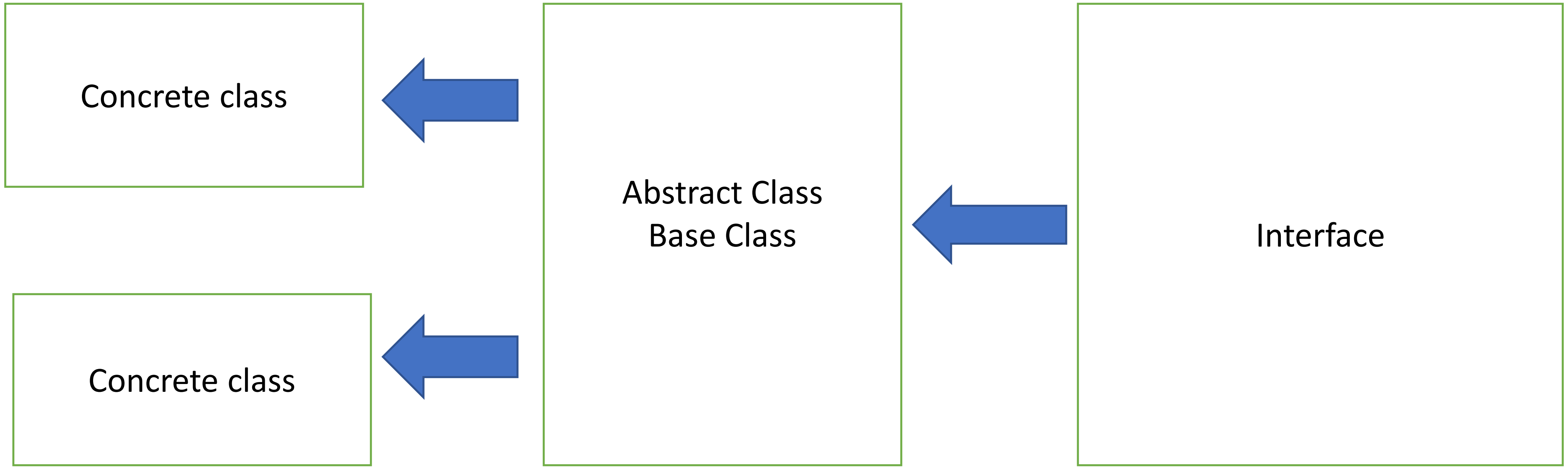
Can inherit multiple interfaces

## Class

Can inherit one class or abstract class

Can inherit multiple interfaces

Must implement any interface or abstract properties and methods


Concrete class
Concrete class
Abstract Class
Base Class
Interface

Only contains public properties and methods

Represents a contract/template where all items must be implemented

Interface

```
getName();
getHeight();
getFurColor();
```

**Abstract Class**

Can contain code implementations of properties and methods

Properties and methods can be private

Any property or method marked abstract must implemented by the inherited member

Abstract Class

```
//properties
Private String name;
Private int height;
Private String furColor;

//constructor (optional)
Public Baloo(String name, int height, String
furColor){
        omitted for space
}

//methods
Public getName(){};
Public getHeight(){};
Public getFurColor(){};
```

**Abstract Class**

Can contain code implementations of properties and methods

Properties and methods can be private

Any property or method marked abstract must implemented by the inherited member

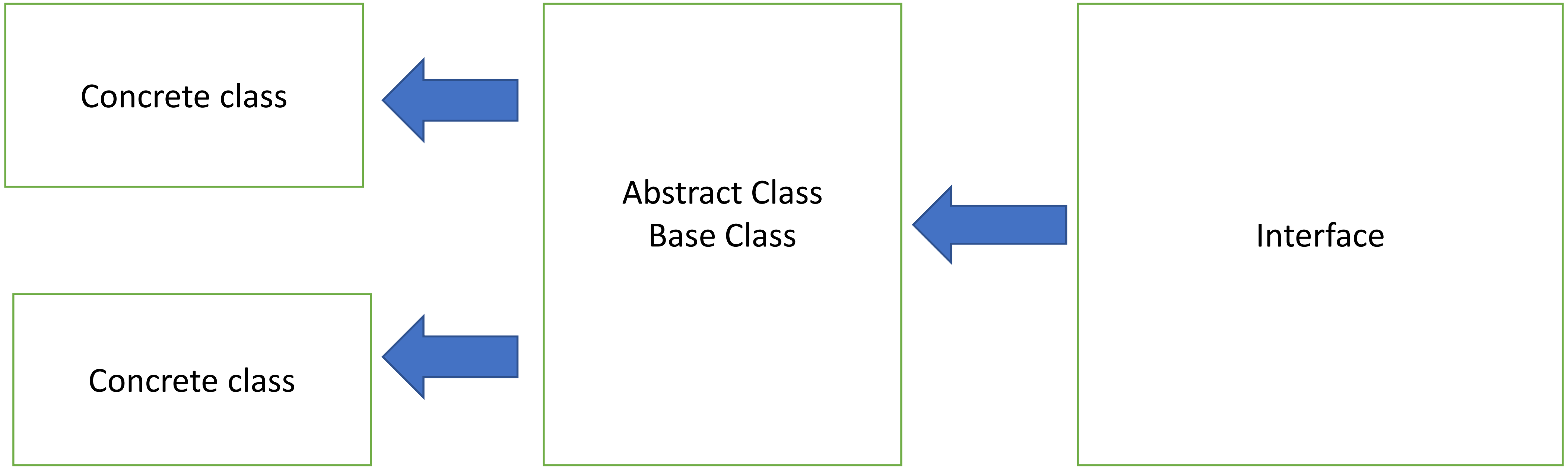
Abstract Class
vs
**Base Class**

**Base Class**

Ordinary Class used as Parent class

AKA POCO – plain old class object

Can be instantiated and stand alone


Concrete class
Concrete class
Abstract Class
Base Class
Interface

# More Inheritance

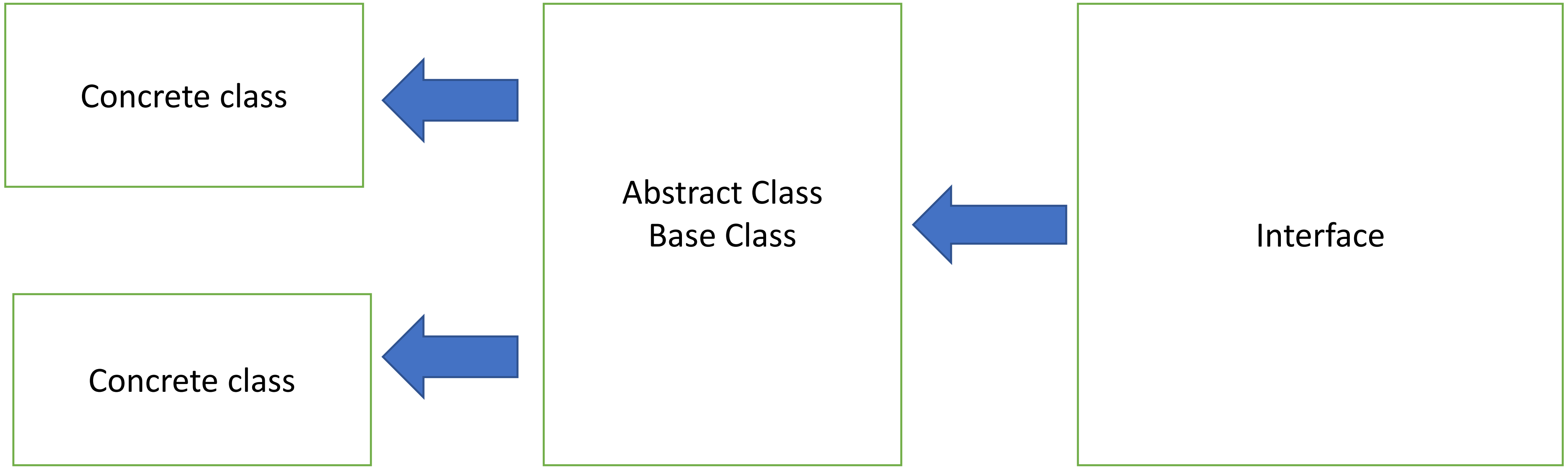
Stop the ride I want to get off!

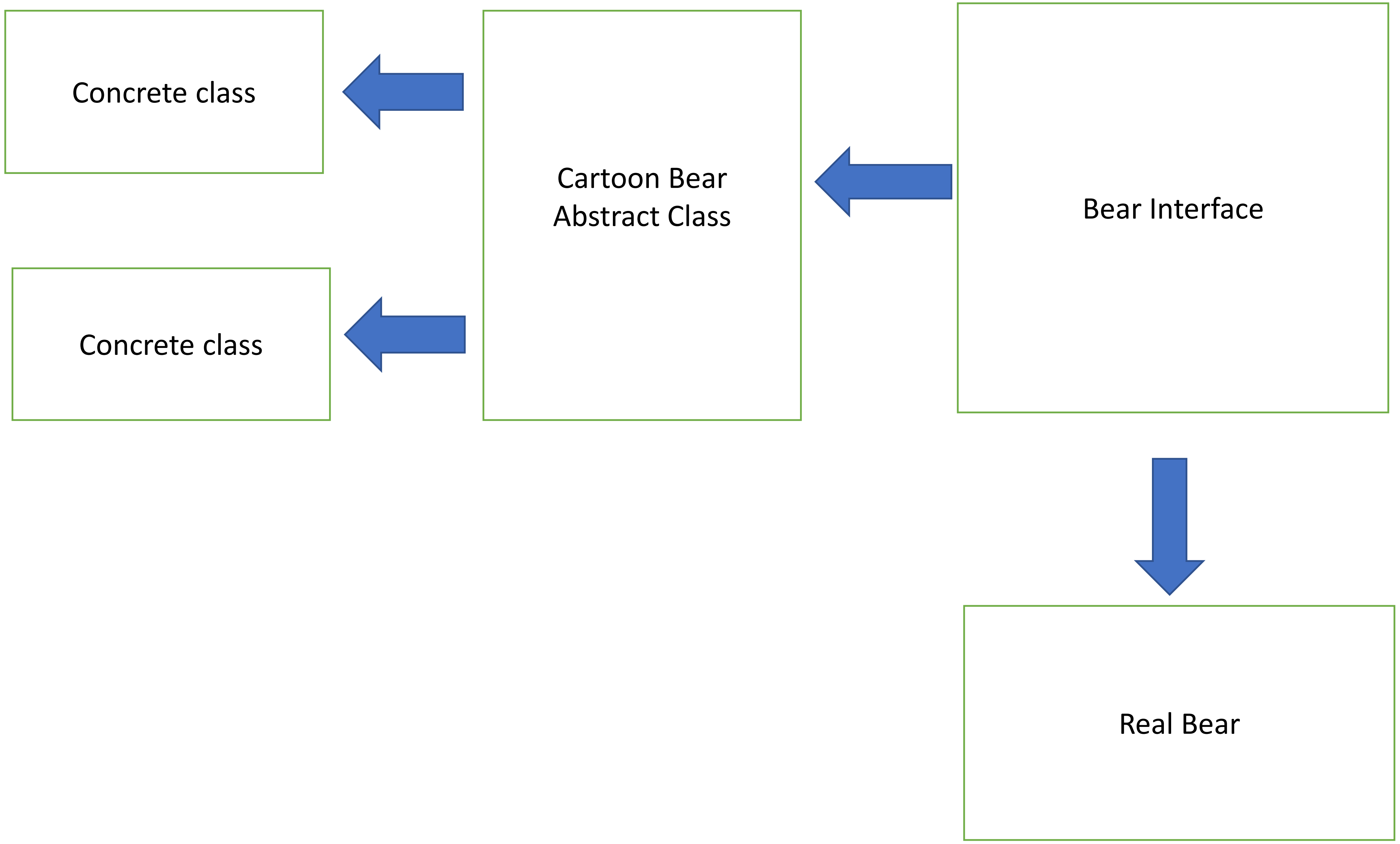
Baloo can be inherited to assume other bear characters.

They are all Baloo but have different characteristics

Yogi and Papa Berenstain always remain the same and are defined as final classes

# What about side kicks?


Concrete class
Concrete class
Abstract Class
Base Class
Interface


Concrete class
Concrete class
Cartoon Bear Abstract Class
Bear Interface
Real Bear

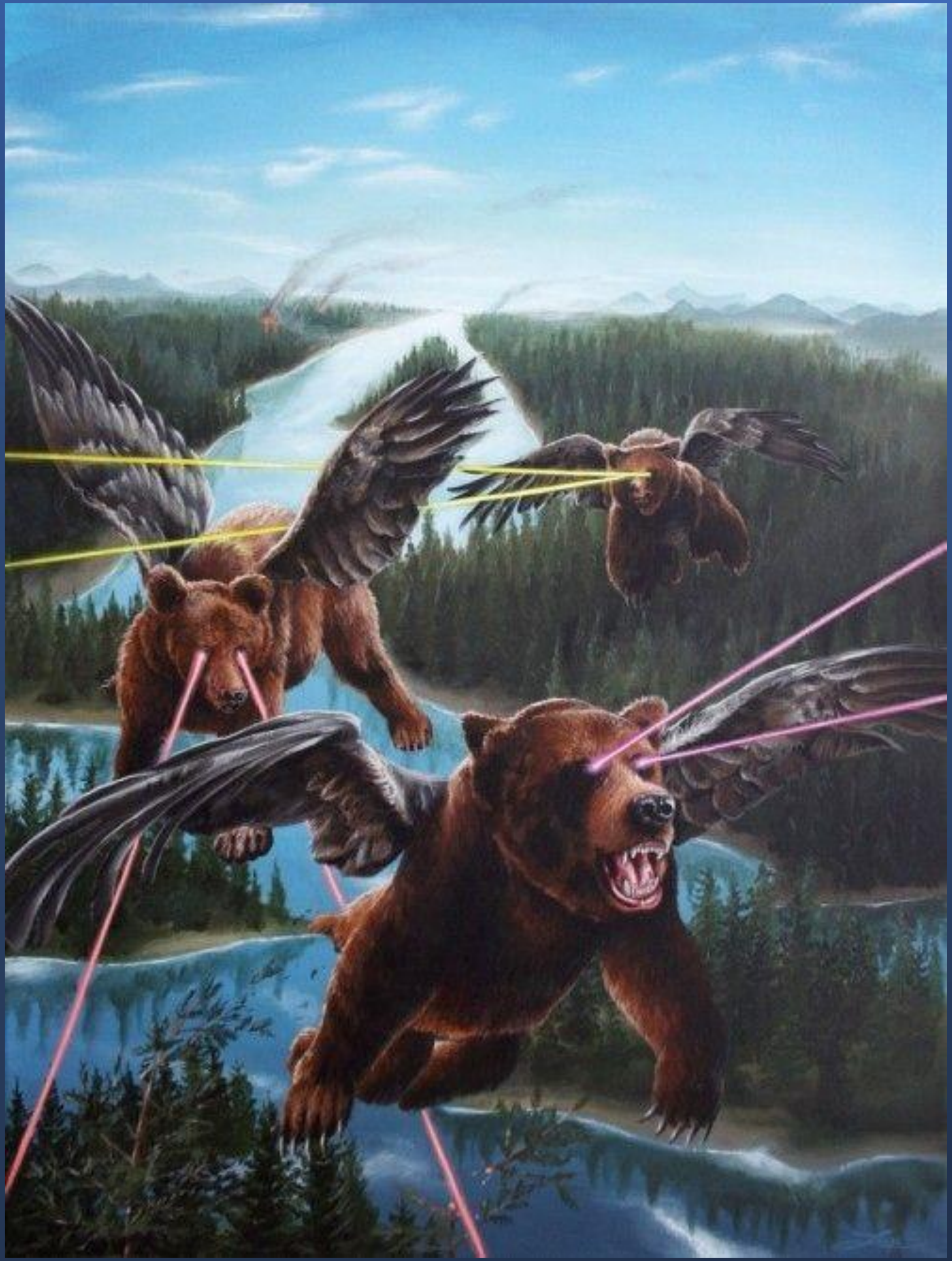
## Suppose we build Battle Bot

```
public Interface BattleBot{
            // properties
            // We want these private properties
            // int health;
            // int armor;
            // Weapon currentWeapon;
            // List<Weapon> weapons;

      //No Constructor

      //setters and getters
        public int getHealth();
        public int getArmor();
        public Weapon getCurrentWeapon();

      // methods
        public int strikeOpponent();
        public int takeHit(int opponentStrikeValue);
}
```

```
public Abstract Class BattleBot implements BattleBot{
        // properties
                // We want these private properties
        private int health;
        private int armor;
        private Weapon currentWeapon;
        private List<Weapon> weapons;

        Public BattleBot(int health, int armor, Weapon currentWeapon, List<Weapon> weapons){
              this.health = health,  ….  Condensed for space
        }

        //setters and getters
          public int getHealth();
          public int getArmor();
          public Weapon getCurrentWeapon();

        // methods
          public int strikeOpponent();
          public int takeHit(int opponentStrikeValue);
}
```

```
Public Baloo extends CartoonBear implements BattleBot {

        //What happens?
}
```

## Battle Bot Bear



CN
CARTOON NETWORK
We BARE BEARS