OOP TA Session 5

Abstract Classes

Interfaces

Abstract Classes

- Cannot be instantiated
- What is it good for?
  - Subclasses will inherit common code and fields
  - Subclasses will have common type (more later)

Example 1

```
public abstract class AbstractClass {
    public void method() { }
    public abstract void abstractMethod();
}

public class SubClass extends AbstractClass {

}
```

"The type SubClass must implement the inherited abstract method AbstractClass.abstractMethod() "

Example 1

```
public abstract class AbstractClass {
    public void method() { }
    public abstract void abstractMethod();
}

public class SubClass extends AbstractClass {
    public void abstractMethod() { }
}
```

Example 1

```
public abstract class AbstractClass {
    public void method() { }
    public abstract void abstractMethod();
}

public abstract class SubClass extends AbstractClass {

}
```

Example 2

```
public abstract class AbstractClass {
    public void method() { }
}


public class SubClass extends AbstractClass {

}
```

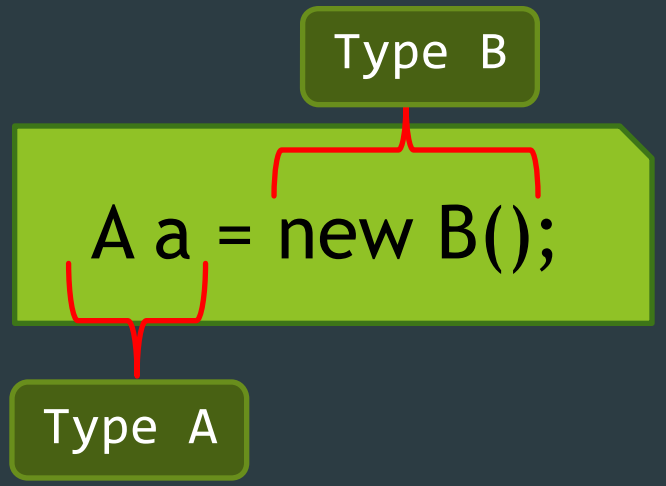
Example 3

```
public class AbstractClass {
    public void method() { }
    public abstract void abstractMethod();
}
```

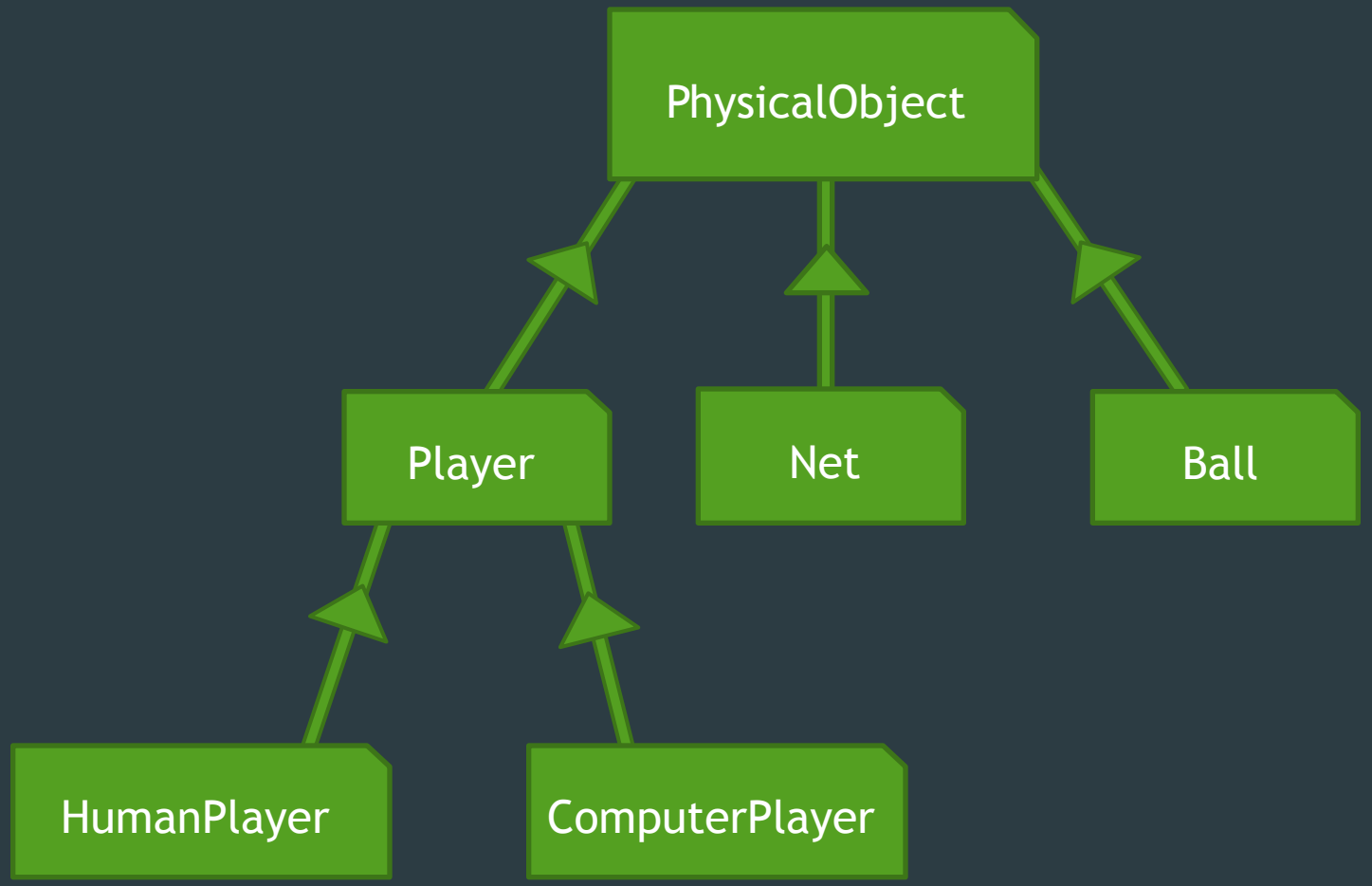
“ The abstract method abstractMethod in type AbstractClass can only be defined by an abstract class “

Polymorphism: Is it awesome?

- It is awesome.
- It’s half of what “object-oriented” is about.
- More in the lecture. Stay tuned.
- In a nutshell:



Example: Simplified Volleyball Game



Example: Simplified Game Loop

```
public class Volleyball {
    public static void playGame() {
        PhysicalObject[] objects =
            { new HumanPlayer(), new ComputerPlayer(),
                new Net(), new Ball() };

        while(true) {
            //update logic - movement etc.
            for(PhysicalObject obj : objects)
                obj.update();
            //draw on screen
            for(PhysicalObject obj : objects)
                obj.render();
        }
    }
}
```

How To Implement PhysicalObject ?

- Each object has a different “update”
- What to render ??

```
public abstract class PhysicalObject {
    private Vector3d pos, velocity, accelaration;

    public PhysicalObject(Vector3d pos) {
        this.pos        = new Vector3d(pos);
        velocity        = new Vector3d(0, 0, 0);
        accelaration    = new Vector3d(0, 0, 0);
    }

    public void update() {
        //update pos, velocity
    }

    public abstract void render();
}
```

Example: Simplified Game Loop

```
public class Volleyball {
    public static void playGame() {
        PhysicalObject[] objects =
            { new HumanPlayer(), new ComputerPlayer(),
              new Net(), new Ball() };

        while(true) {
            //update logic - movement etc.
            for(PhysicalObject obj : objects)
                obj.update();
            //draw on screen
            for(PhysicalObject obj : objects)
                obj.render();
        }
    }
}
```

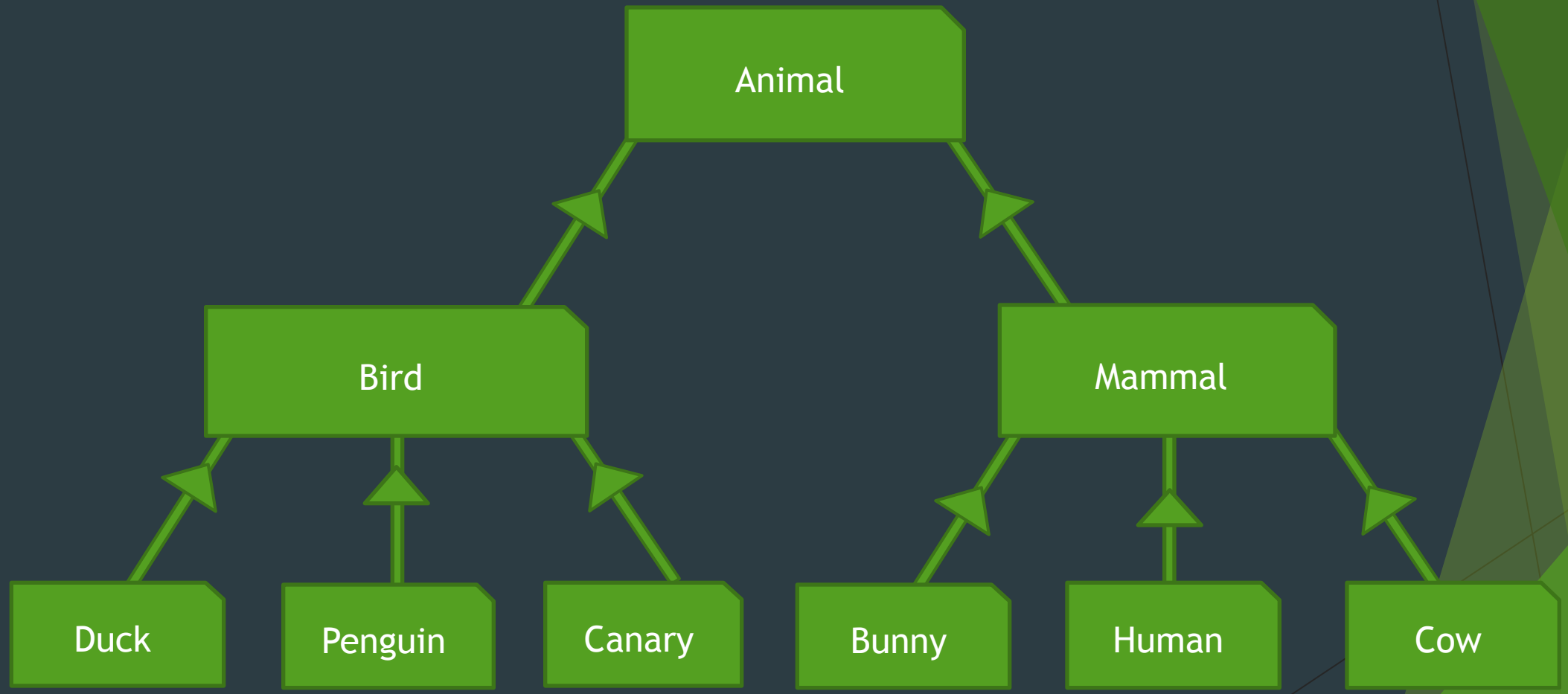
Yo, no, wait buddy. There are no PhysicalObjects.

Yes, there are.

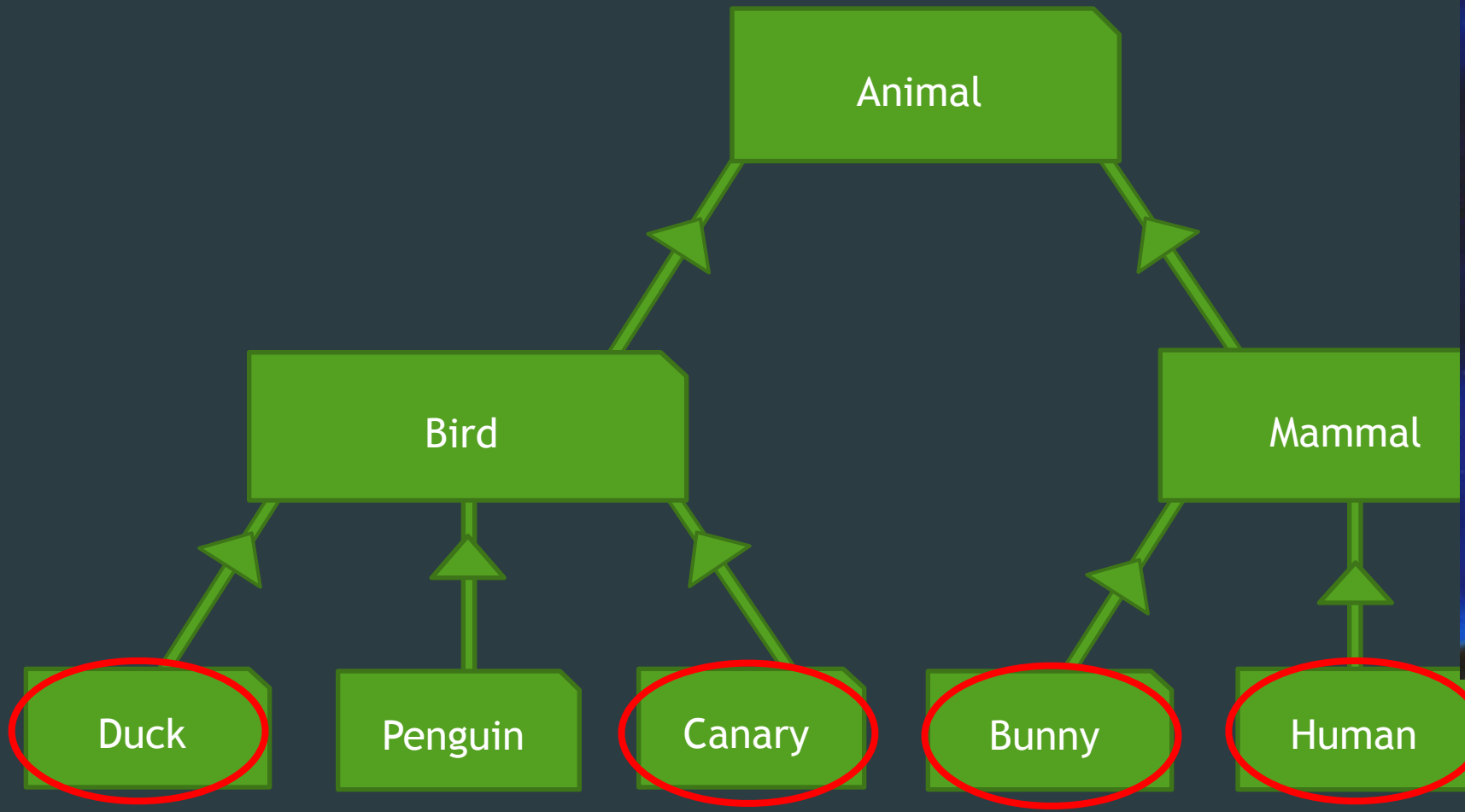
```
public abstract class Player extends PhysicalObject {

    public Player(Vector3d pos){
        super(pos);
    }
    public void render() {
        //draw Pikachu
    }
}
```

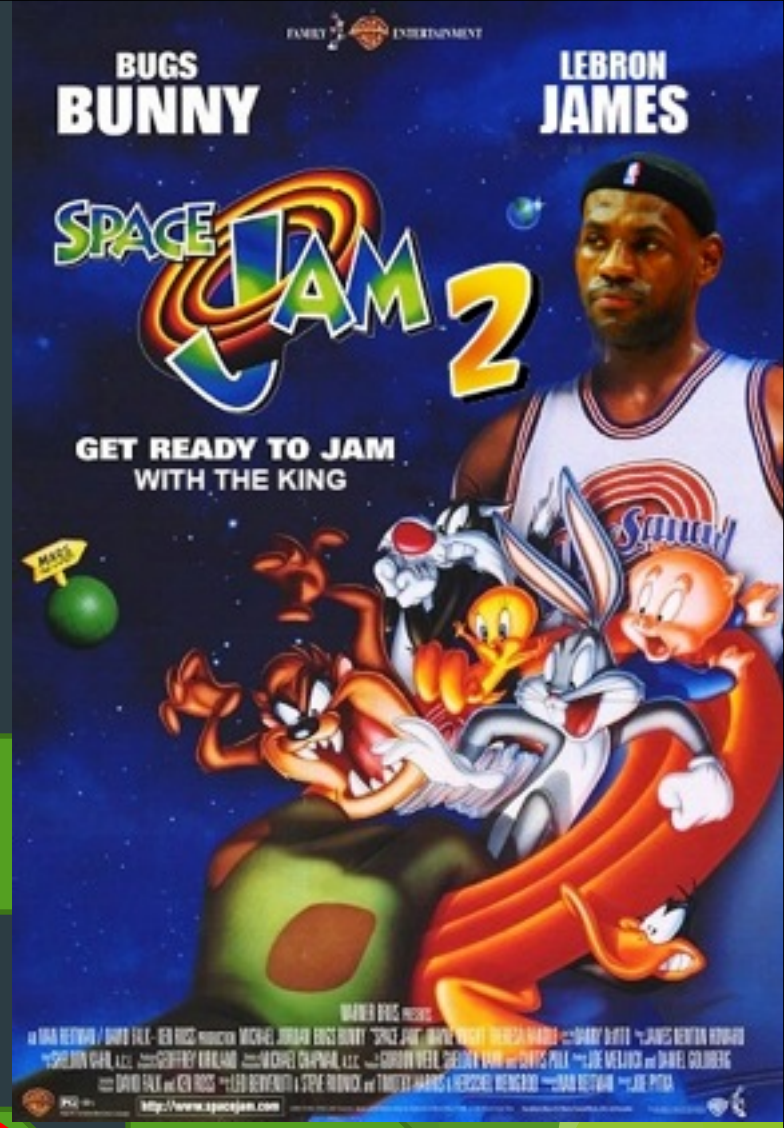
```
public class HumanPlayer extends Player{

    public HumanPlayer() {
        super( new Vector3d(1,0,0) );
    }
    public void update() {
        super.update(); //update pos, velocity
        /*
         * respond to keyboard
         */
    }
}
```

Example: Animal Kingdom



Example: Animals Playing Basketball





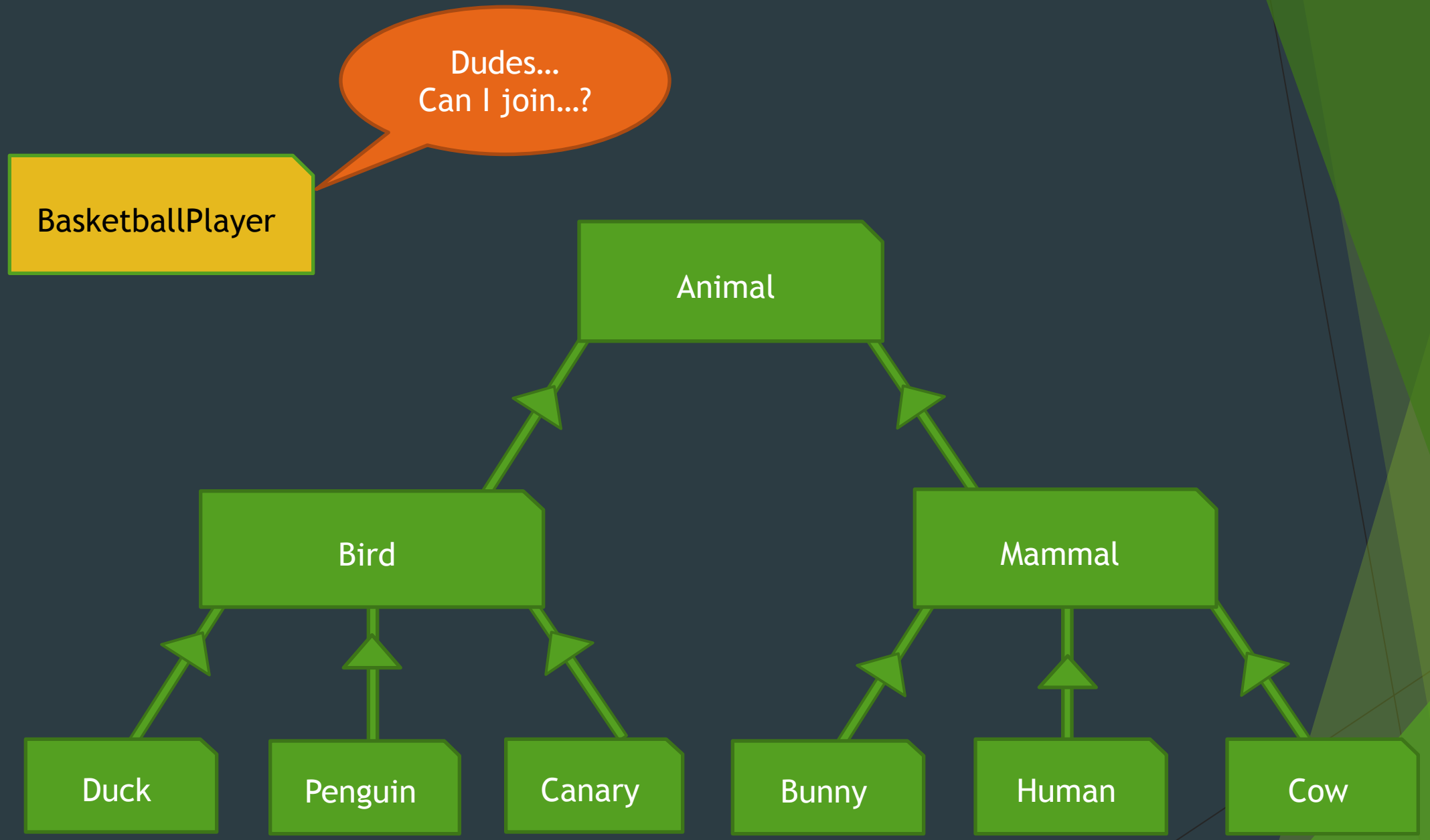
Buggs wants a team:

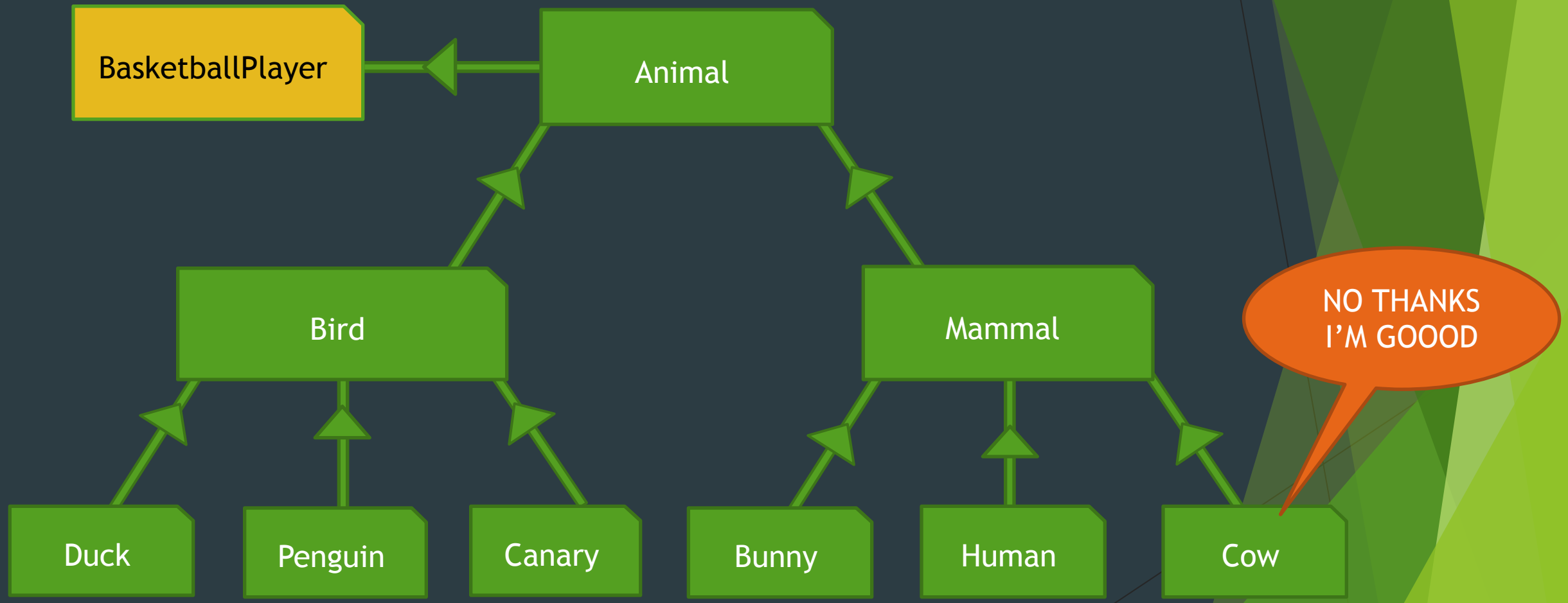
```
BasketBallPlayer[] team =
     { buggs, tweety, michael, duffy };

team[0].passBall(team[1]);
team[1].shoot();
// ..
```

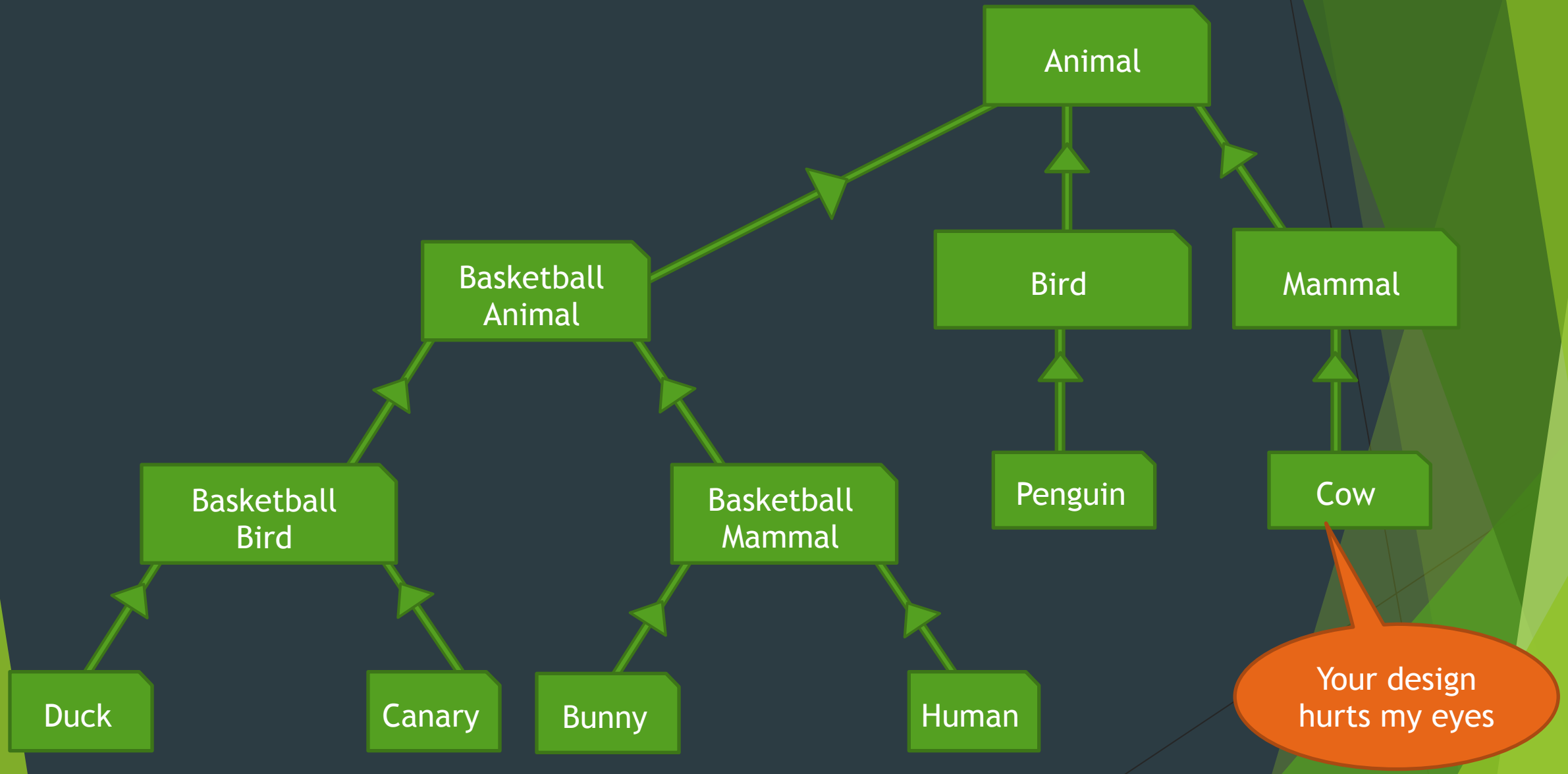
Solution!

```
public abstract class BasketballPlayer {
    public abstract void shoot();
    public abstract void dribble();
    public abstract void catchBall();
    public abstract void passBall(BasketballPlayer player);
}
```

Imlementation is physiology-dependant


Dudes…
Can I join…?
BasketballPlayer
Animal
Bird
Mammal
Duck
Penguin
Canary
Bunny
Human
Cow


BasketballPlayer
Animal
Bird
Mammal
NO THANKS
I'M GOOOD
Duck
Penguin
Canary
Bunny
Human
Cow

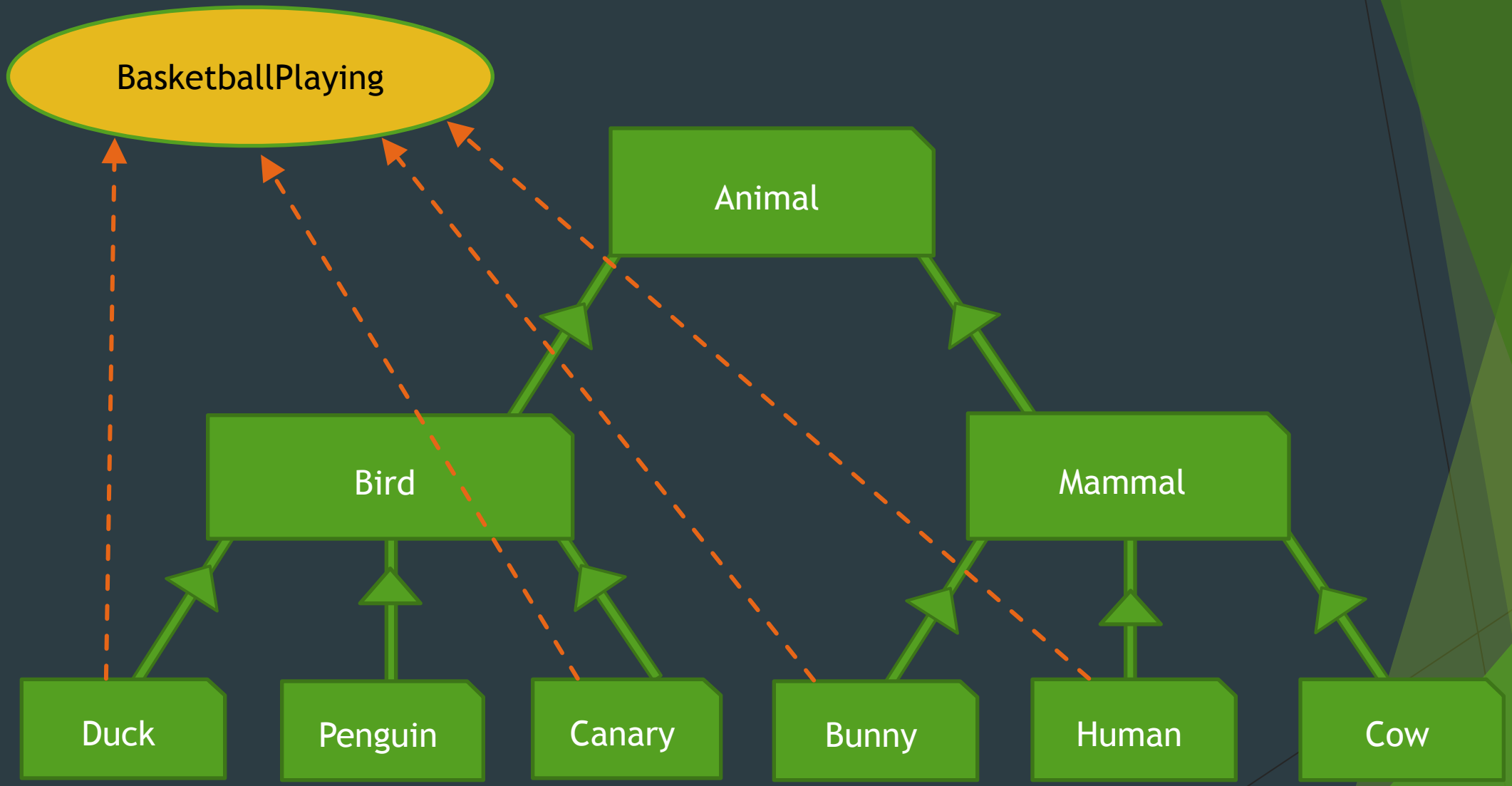

Animal
Basketball Animal
Bird
Mammal
Basketball Bird
Basketball Mammal
Penguin
Cow
Duck
Canary
Bunny
Human
Your design hurts my eyes

Solution: interface

```
public interface BasketballPlaying {
    void shoot();
    void dribble();
    void catchBall();
    void passBall(BasketballPlaying player);
}
```

Cues for using interfaces

- BasketballPlayer doesn’t provide any implementation
- A Bunny doesn’t meet “is-a BasketballPlayer”
- A Bunny CAN play basketball
- So can unrelated classes


BasketballPlaying
Animal
Bird
Mammal
Duck
Penguin
Canary
Bunny
Human
Cow

```
BasketBallPlaying[] team =
     { buggs, tweety, michael, duffy };

team[0].passBall(team[1]);
team[1].shoot();
// ..
```

Polymorphism is awesome

EX3: Space Wars