Lesson 13-1: Inheritance

Computer Science 46A: Introduction to Programming

San José State University

Announcements

- Lab tomorrow
- Next week, Prof Narayan Balasubramanian might be back
- When is the second mid term?
  - Format very similar to the first mid term

Learning Objectives

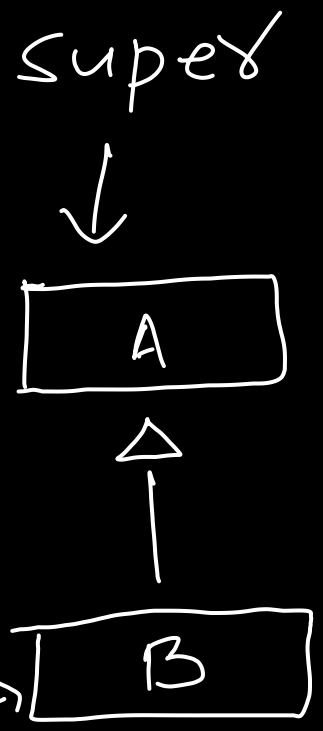
By the end of this lesson, you should be able to:

1. Explain the relationship between superclasses and subclasses
2. Reorganize a set of similar classes as subclasses that extend a superclass
3. Explain the use of the `super` reference and implement it in a subclass

Inheritance

Inheritance Defined

- Often, we want to create objects with similar behavior
  - These objects share some of the same instance variables and methods
- In Java, we can create a superclass that contains methods to be shared across other subclasses
  - Subclasses *extend* the functionality of a superclass with additional instance variables, constructors, and methods
  - Subclasses **inherit** the constructors and methods of the super class
- Inheritance is our second main pillar of Object Oriented Programming

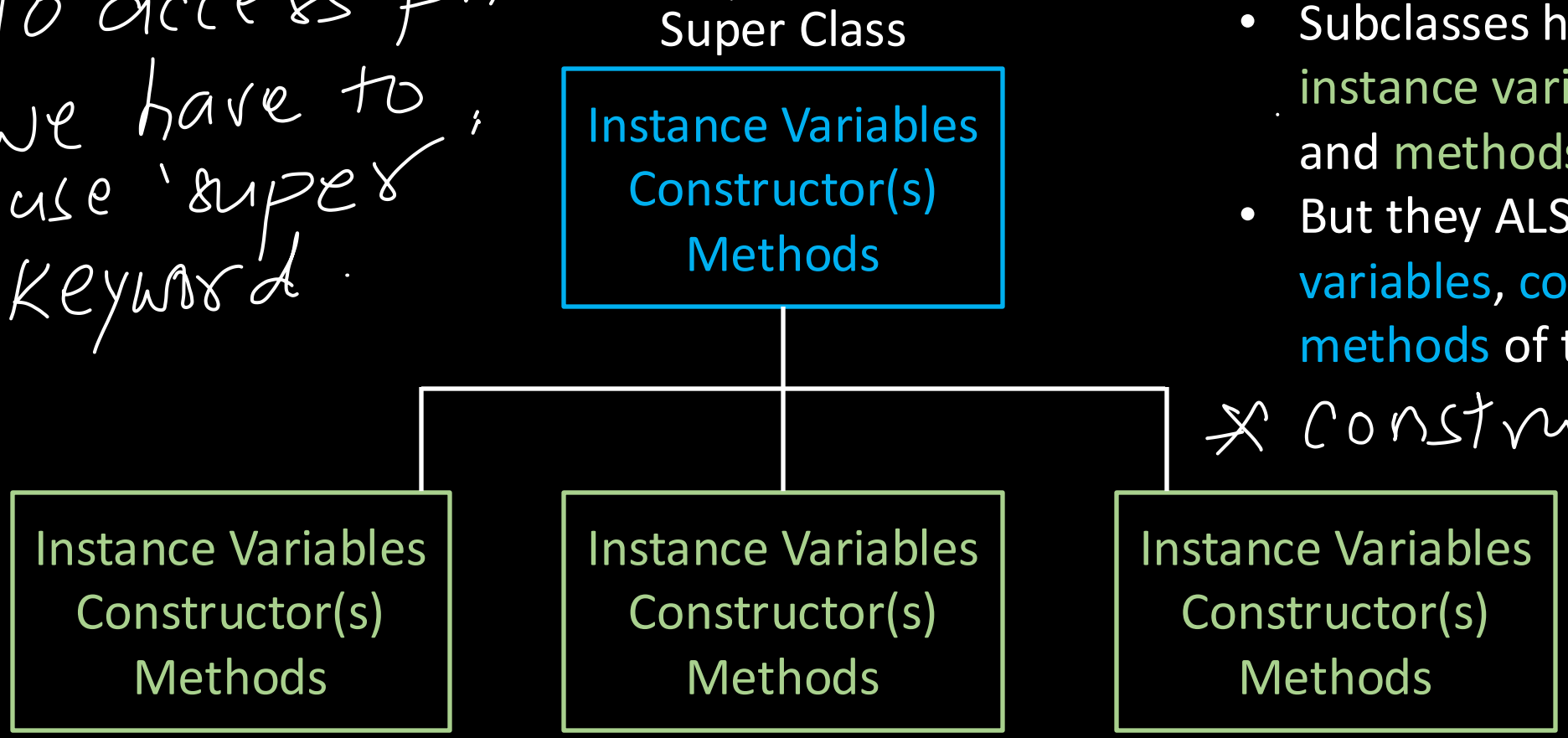


class B extends A {
↑ inherits all public & protected

properties of class A. Not the private properties

Inheritance: A visualization

To access private properties. we have to use 'super' keyword.

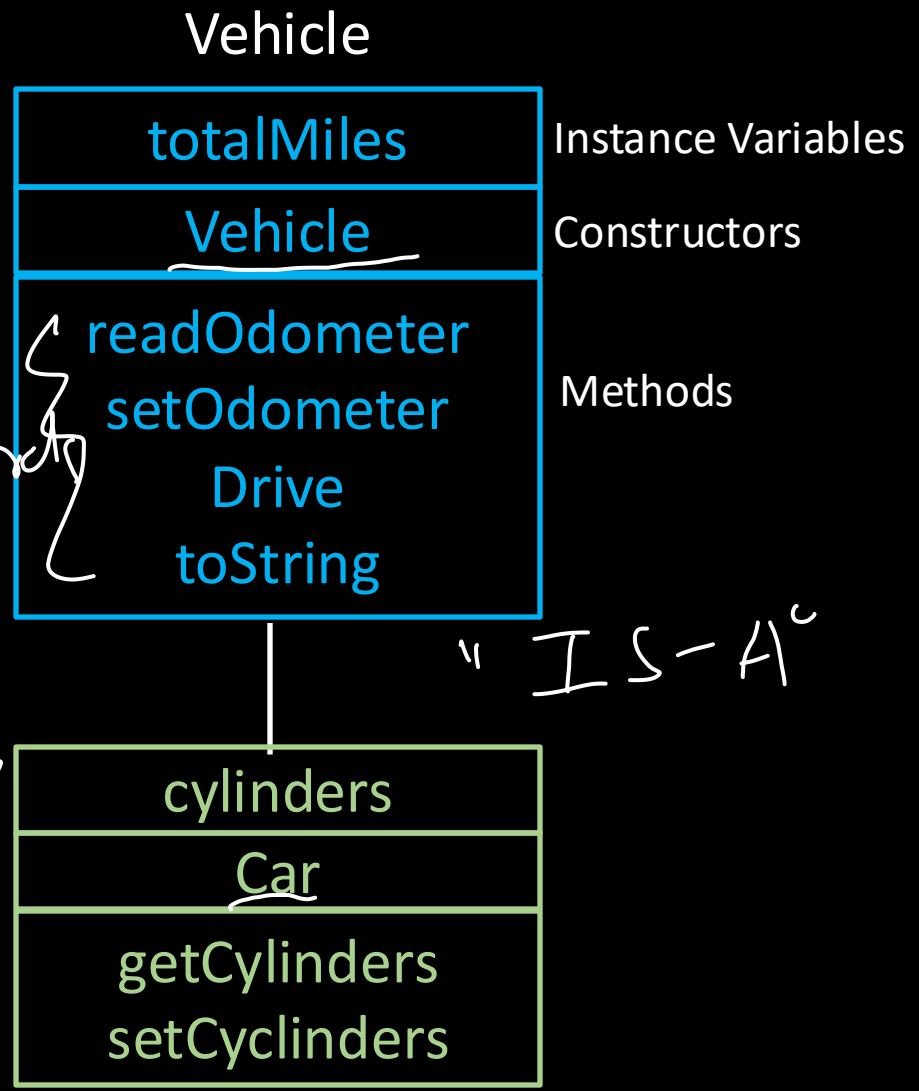


- Subclasses have their own instance variables, constructors and methods.
- But they ALSO have the instance variables, constructors, and methods of the super class

* Constructor of parent class cannot be inherited. But we can call parent class constructor

An Example of Inheritance

- Car objects are an extension of Vehicle objects
- Car objects *inherit* the following:
  - An instance variable for totalMiles
  - Methods for readOdometer, setOdometer, drive, and toString
- Car objects have their own:
  - Instance variable for cylinders
  - Methods for getCylinders and setCylinders



See example code `Vehicle` and `Car` in lec13-1a

Poll Everywhere: Question 1

In VehicleTester, we called

```
vehicle.readOdometer()
car1.readOdometer()
car2.readOdometer()
```

Fill in the blanks:

readOdometer is declared in the _____ class and accesses the instance variable `totalMiles` declared in the _____ class.

super() → it is calling parent class constructor

A) Vehicle, Vehicle

B) Vehicle, Car

C) Car, Vehicle

D) Car, Car

super.method() → then it is calling parent class method.

PollEv.com/narayanbalasubramanian644

Substitution Principle

super.var → You are accessing parent class variable.

- Subclasses can be used in methods and other structures that are defined in terms of a super class
  - Ex: a Car object can be added to an ArrayList<Vehicle>
- This property is called the substitution principle
- But, this only goes one way – superclass objects cannot be used in methods or structures defined for objects of their subclasss
  - Ex: a Vehicle object cannot be added to an ArrayList<Car>

↑ Parent reference variable can hold child objects.

ArrayList<Vehicle> al = new ArrayList<Vehicle>();

↑ can hold Car type objects

See example code `Vehicle` and `Car` in lec13-1a

ArrayList<Car> alc = new ArrayList<Car>();

↑ This cannot hold Vehicle objects.

Organizing Similar Classes With a Superclass

al.add(new Car());

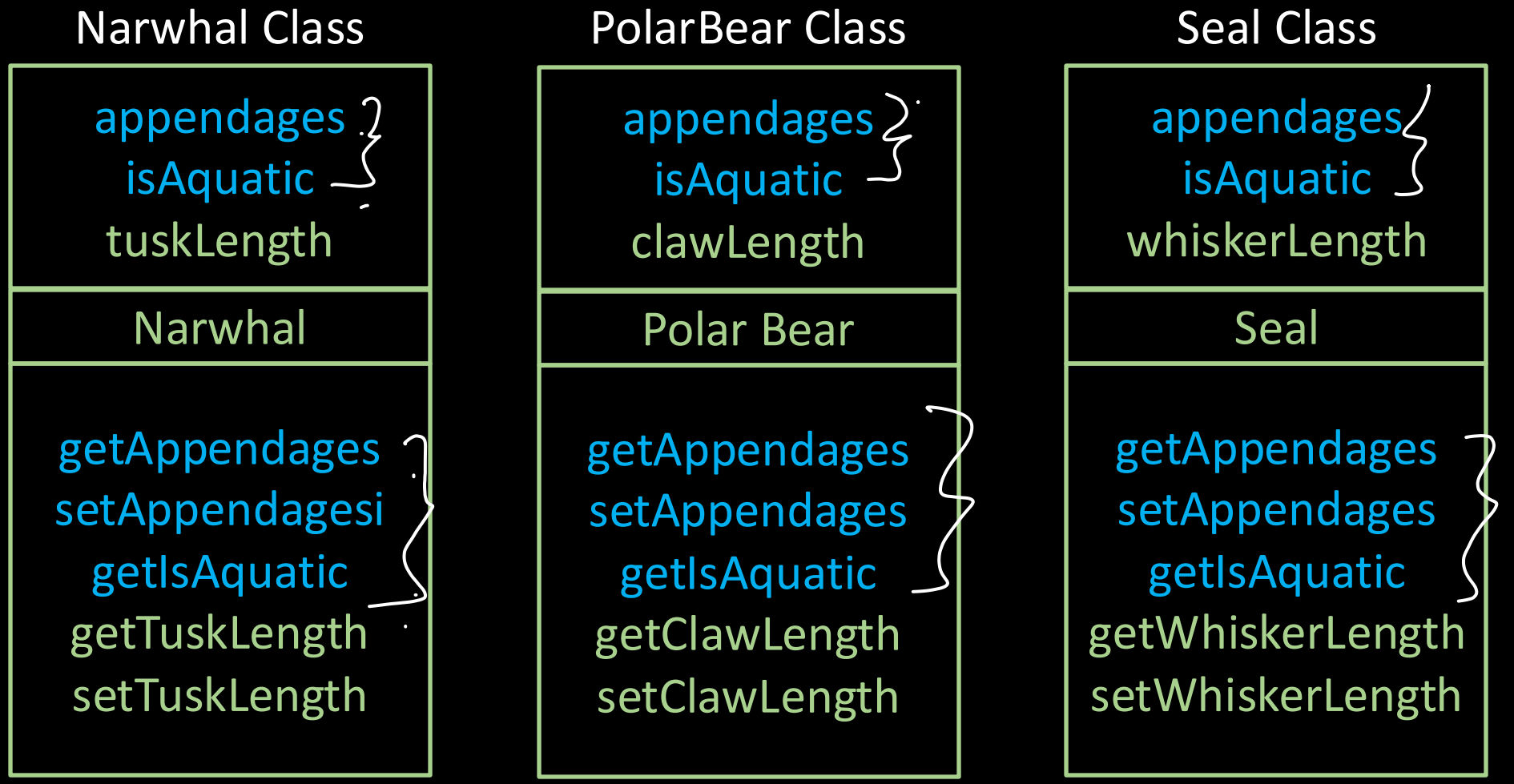
Identifying When We Need a Superclass

- In Lecture 12-1, we created a package called GreenlandMammals
- This package had 3 classes for Narwhal, PolarBear, and Seal
- These classes share some of the same instance variables and methods
  - If we want to edit one of these methods, we need to do it in all 3 classes
  - In these situations, it is best to use a superclass
- Duplicate code across similar classes is one of the easiest ways to accidentally introduce bugs into a program
  - It is also one of the easiest issues to avoid

See example code `GreenlandMammals` in lec13-1b

GreenlandMammals without a Superclass

Instance variables and methods are duplicated across the same classes



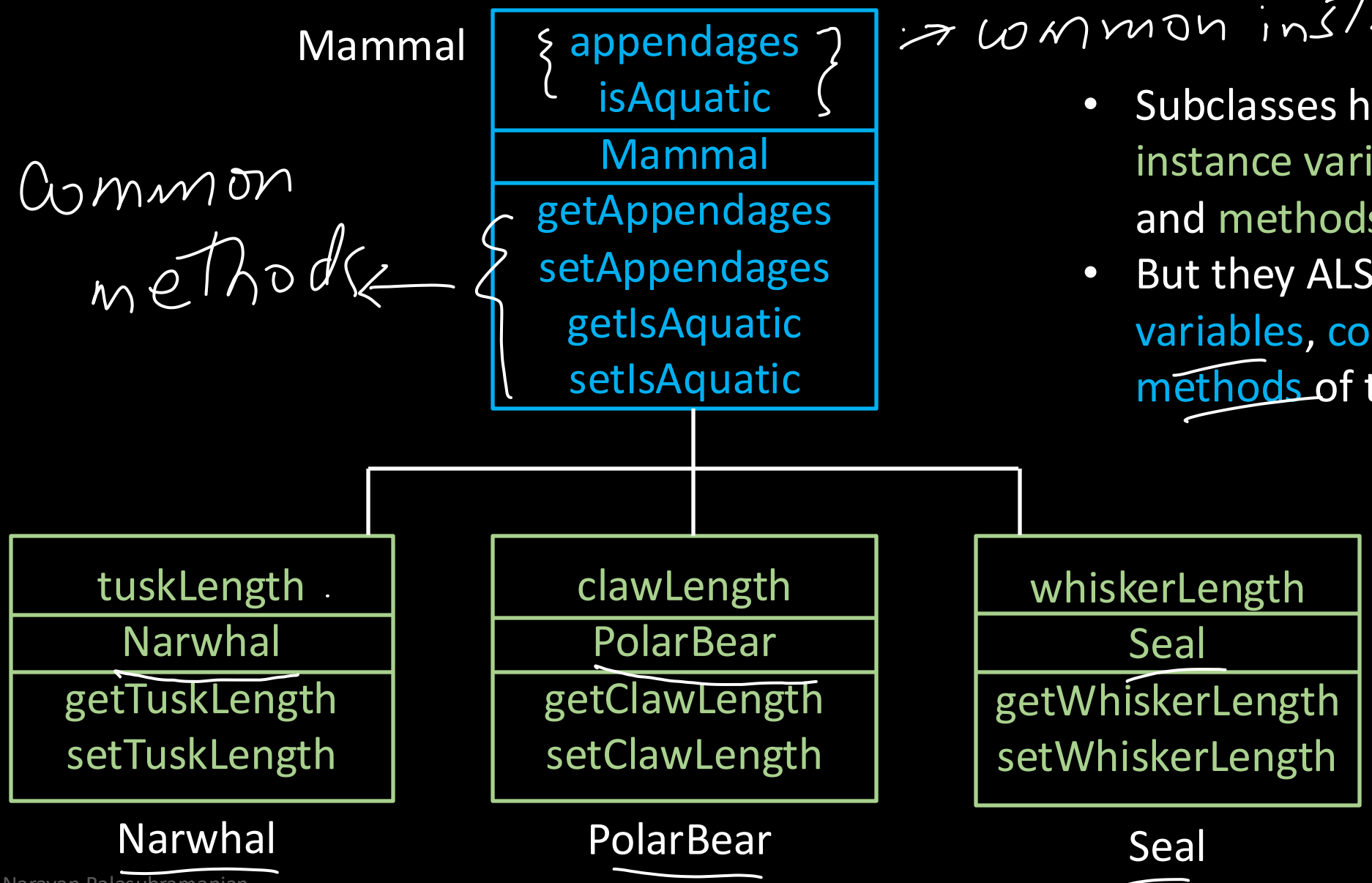
See example code `GreenlandMammals` in lec13-1b

Reorganizing to Leverage Inheritance

- We can reorganize these classes by putting common instance variables and methods into a superclass (e.g. Mammal)
- Then, each subclass keeps only the instance variables and methods that are specific to that class
  - E.g. Narwhals only have references to tusk

See example code `GreenlandMammals` in lec13-1c

GreenlandMammals with a Superclass



- Subclasses have their own instance variables, constructors and methods.
- But they ALSO have the instance variables, constructors, and methods of the super class

See example code `GreenlandMammals` in lec13-1c

The `super` keyword

The `super` reference

- The `super` keyword allows us to reference the constructor and methods from the superclass
  - We cannot reference the instance variables of the superclass using the super reference – instead, use the get methods!
- The `super` keyword is similar to the `this` keyword – it provides an explicit reference to which class we are talking about

The `super` keyword in subclass constructors

- Inside of a constructor, the `super()` keyword refers to the constructor of the superclass
- If the constructor of the superclass does not have any parameters, then the `super()` reference is not necessary
  - The `super()` method is called by default in subclass constructors
  - See example in the Car subclass
- If the constructor of the the superclass does have argument(s), then the `super(args)` reference *must* be in the constructor of the subclass

See example code `Planet` in lec13-1e

The `super` keyword in subclass methods

- Inside methods, the `super` keyword can be used to reference the methods of the superclass
- The default call for methods is to methods in the subclass
  - Superclass methods will only be called if a method is not found in the subclass
- The `super` keyword is especially important for methods with the same name but different declarations in the superclass and subclass

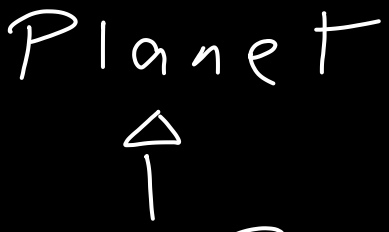
See example code `Planet` in lec13-1e

Poll Everywhere: Question 2

In the toString() method of the Earth subclass, what class does the getRadius() method come from?

A) The getRadius() method in Planet

B) The getRadius() method in the Earth




```
@Override
public String toString()
{
    double radius = getRadius();
    return "Earth[Radius:"+radius+",Atmoshphere Height:"+atmosphereHeight+"]";
}
```

is present Earth class otherwise super.getRadius() will be used.

PollEv.com/narayanbalasubramanian644

Participation Exercise 13-1a: Guitar

Goal: Complete 2 subclasses called BassGuitar and AcousticGuitar that will extend the Guitar class with new attributes.

```
Testing the string count:
6
Expected: 6
4
Expected: 4
6
Expected: 6

Testing the pickup count:
3
Expected: 3

Testing the wood type:
Maple
Expected: Maple

Testing overridden toString methods:
Guitar[NumberOfStrings:6]
Expected: Guitar[NumberOfStrings:6]
BassGuitar[NumberOfStrings:4,NumberOfPickups:3]
Expected: BassGuitar[NumberOfStrings:4,NumberOfPickups:3]
AcousticGuitar[NumberOfStrings:6,WoodType:Maple]
Expected: AcousticGuitar[NumberOfStrings:6,WoodType:Maple]
```

Codecheck Link: HERE and on Canvas

Expected output of the GuitarTester class

Participation Exercise 13-1b: Insects

Goal: Write a superclass called Insect for the provided classes Bee and Ant.

To complete this task, you will need identify which instance variables and methods are shared in the Bee and Ant classes, implement them in the Insect class, and modify the Bee and Ant code to operate as extensions of the Insect class.

Codecheck Link: HERE and on Canvas

```
Testing the leg count:
6
Expected: 6
6
Expected: 6
6
Expected: 6

Testing the wing count:
4
Expected: 4

Testing the body parts:
3
Expected: 3

Testing overridden toString methods:
Insect[NumberOfLegs:6]
Expected: Insect[NumberOfLegs:6]
Bee[NumberOfLegs:6,NumberOfWings:4]
Expected: Bee[NumberOfLegs:6,NumberOfWings:4]
Ant[NumberOfLegs:6,NumberOfBodyParts:3]
Expected: Ant[NumberOfLegs:6,NumberOfBodyParts:3]
```

Expected output of the InsectTester class