CS 1181
Week One

Reese Hatfield

Introduction

- Syllabus
- Zybooks
  - Need to buy this
- Lecture Attendance
  - Required
- Group Enrollment

Lab

- MW 1:30 - 2:35
- Teaching Assistants
  - Greg Rosen
- Projects
- Lab Problems

Course Work

- JDK
  - Need Java 17+
- IDE
  - VSCode or IntelliJ
- Github

Academic Integrity

- No Citation = Violation
  - Air on the side of caution
- Just as strict as 1180
- You will fail this course
  - 2 Violations = You fail

1180 Review

- Where to go from here?
- Polymorphism
  - Classes and Objects
  - What does this *actually* mean
- We'll see this in the Exam Review!

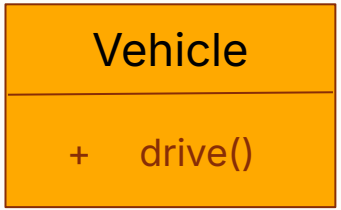
Exam Review

Inheritance

- What does “inherit” actually mean?
- Does this always make sense?
- Locks you into *always* adopting the parents behavior

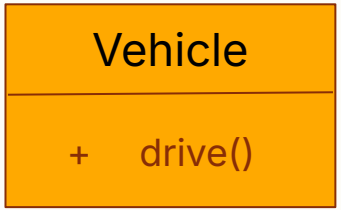
Inheritance

- Vehicle car = new Vehicle()
- Does every car drive the same?



Inheritance

- Vehicle car = new Vehicle()
- Does every car drive the same?

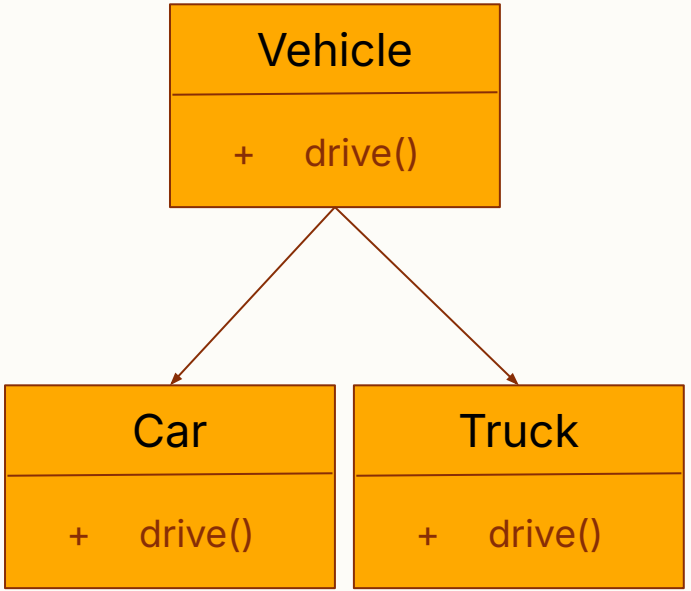


- Of course not!
- So what should we do?

Inheritance

- We could make child classes!

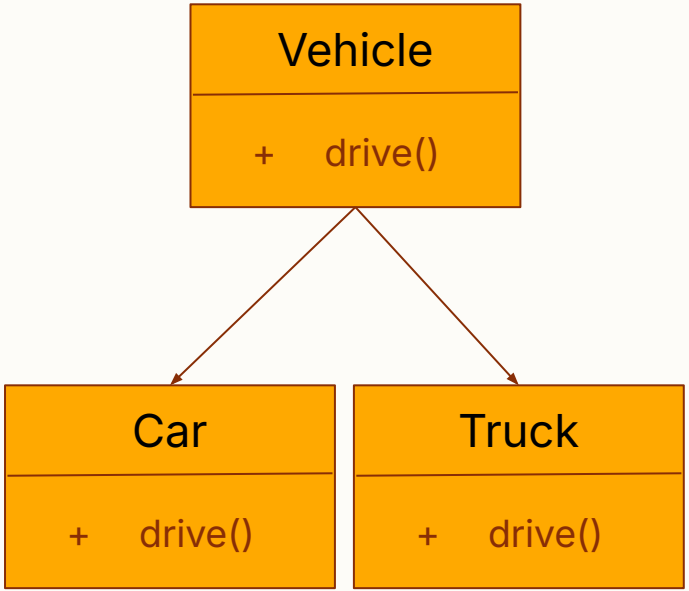
- Vehicle v1 = new Car()
- Vehicle v2 = new Truck()



Inheritance

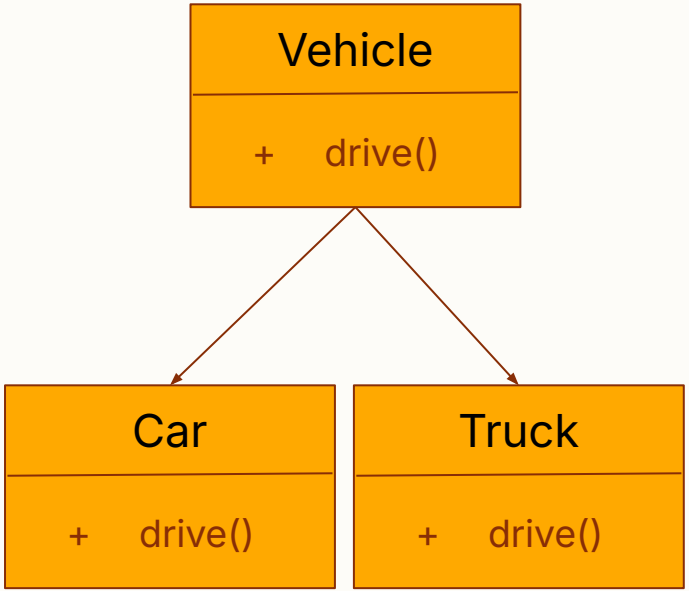
- Does our old code still make sense?

- Vehicle v = new Vehicle()
- How should it drive?



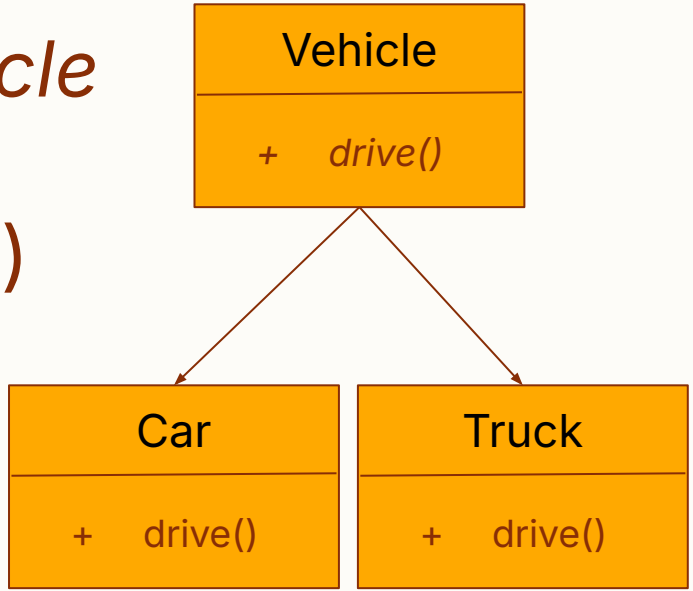
Abstract Classes

- Java “abstract” keyword
- Enforce that this class cannot be instantiated
- Mark methods+classes with “abstract” keyword



Abstract Classes

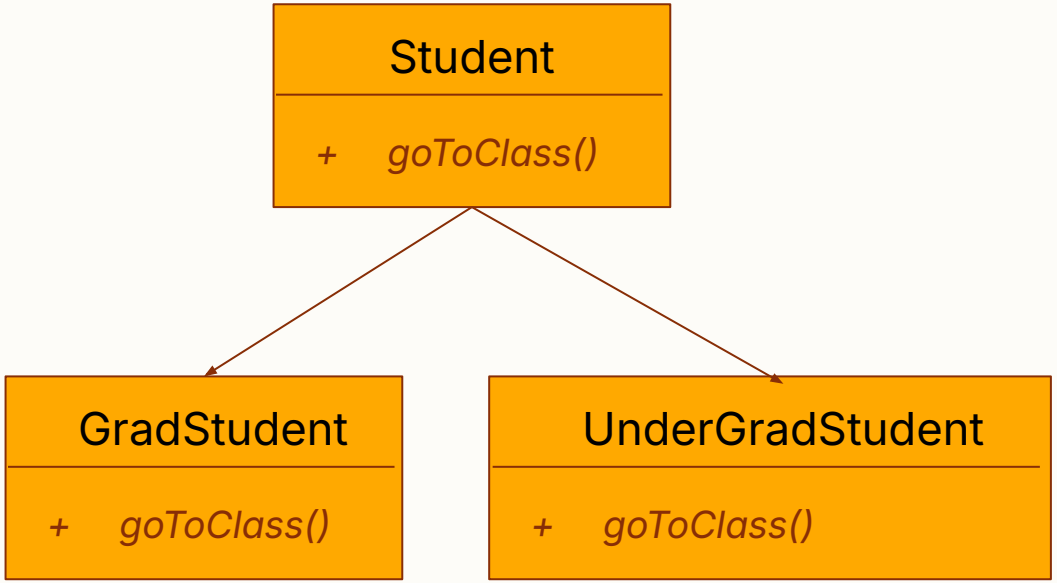
- *public abstract class Vehicle*
- public abstract void drive()
- Each class can implement drive() differently



Abstract Classes

- Let’s do it!

- Student class
- Multiple types



What did that tell us?

- Student Class
  - Described General Behavior
- Child Classes
    - Described Implementation
- Can we mix these?

Abstract Classes Continued

- A class is abstract if
  - It has at least one abstract method
- Not all methods need to be abstract

- Lets add a getGPA() method

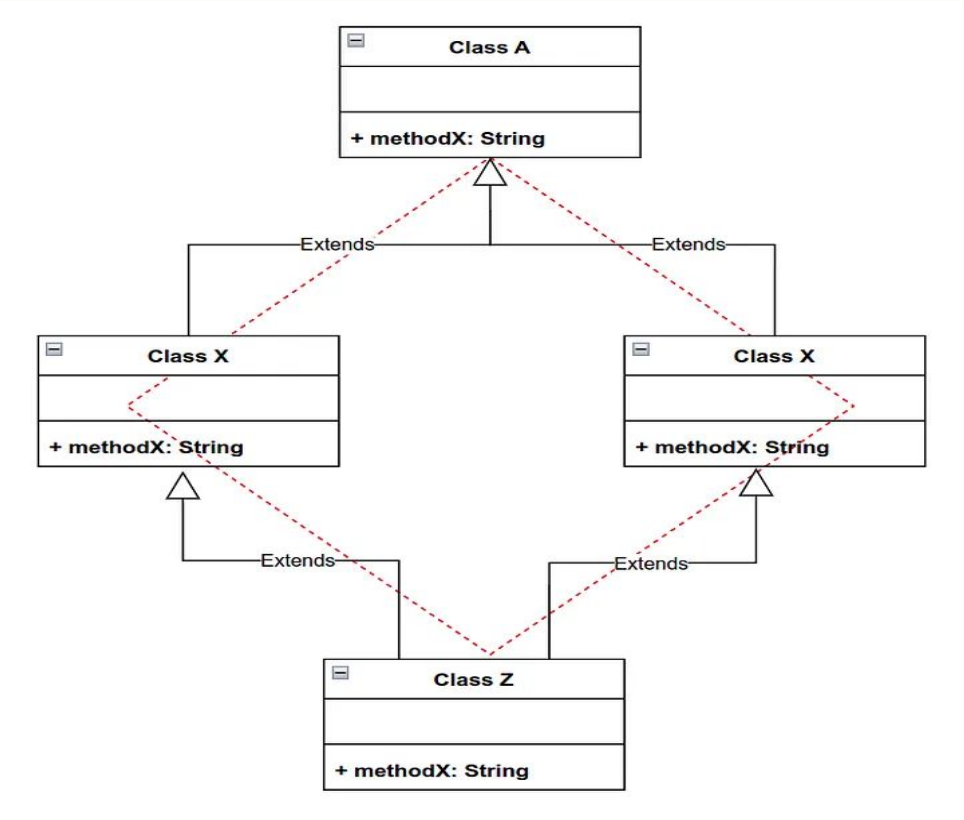
What if we want more?

- What if our GradStudent was also a Commuter?
- public class Commuter

- Now we are forced to pick between extending Commuter or Student

Diamond Problem

- Java does not support multi-inheritance
- Diamond Problem



Interfaces

- Java solves this with interfaces
- Similar to abstract classes
  - Describe method behavior without implementation
- Can implement as many interfaces as the class developer wants

Interfaces

```
public interface Commuter {

    public void driveToCampus()

}
```

Interfaces

- Methods *cannot* specify body
- Use “implements” instead of “extends”
- Interfaces are like contracts

- Lets see that contract in action!
- “must implement the inherited abstract method”

Interfaces

- Implement multiple interfaces with commas
- Java has a number of interfaces built in
- “Comparable” will be very useful to you

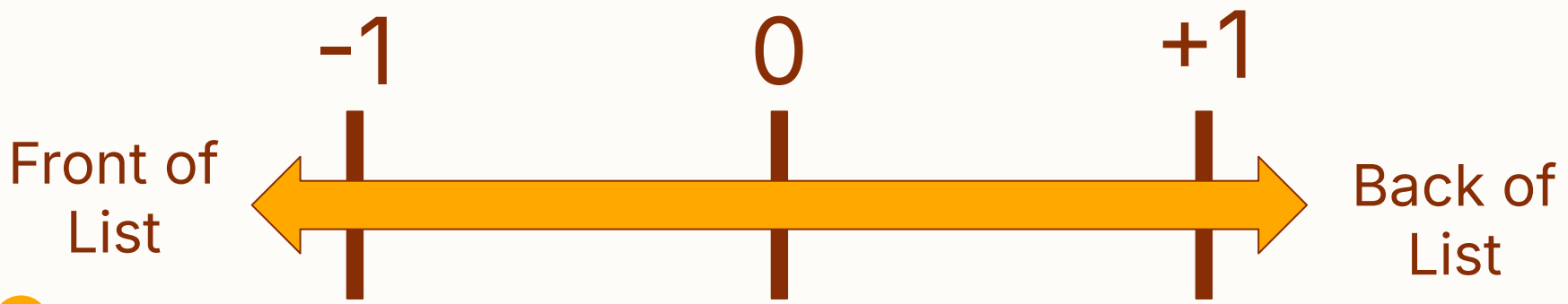
Comparable

- Interface with single method
- “compareTo”
- Used to specify how Collections.sort() works under the hood

- Let’s do it! (with GPA)

Comparable

- How sort two elements?
- “this”
- “other”



Summary

Abstract Classes	Interfaces
No-multi inheritance	Can implement infinite interfaces
Can have default behavior	Cannot* have default behavior
“extends” keyword	“implements” keyword
Not many built in	Many built in interfaces

How do I pick?

- What is the core difference?

How do I pick?

- What is the core difference?

- Default behavior!
- But interfaces are harder to think about
- So why use them?
    - Can’t I just program around hierarchies?

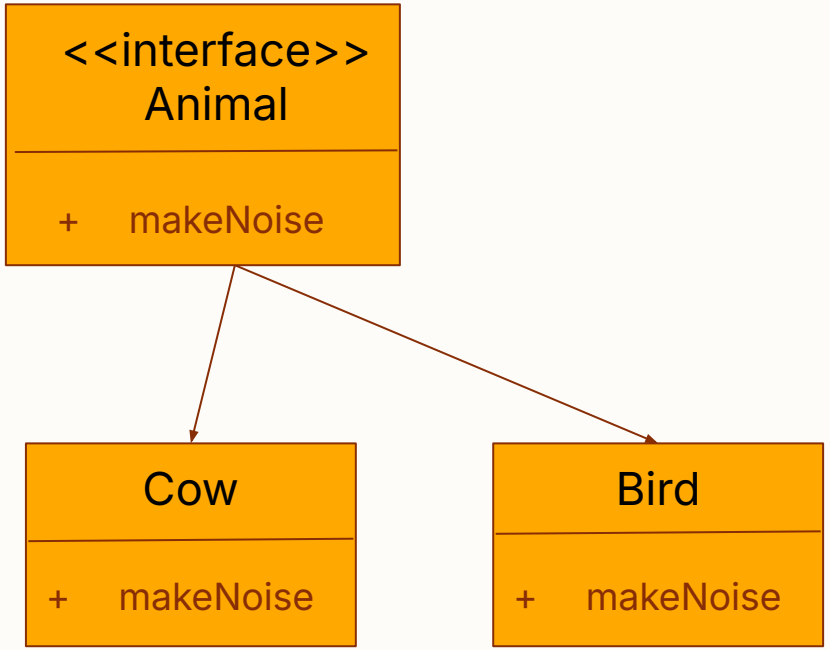
Dynamic Dispatch

- Polymorphism in action
- Interfaces work just like classes
    - Including usage in declarations

- Interfaces can be used on the left on the equals sign
- Interface i = new ConcreteClass()

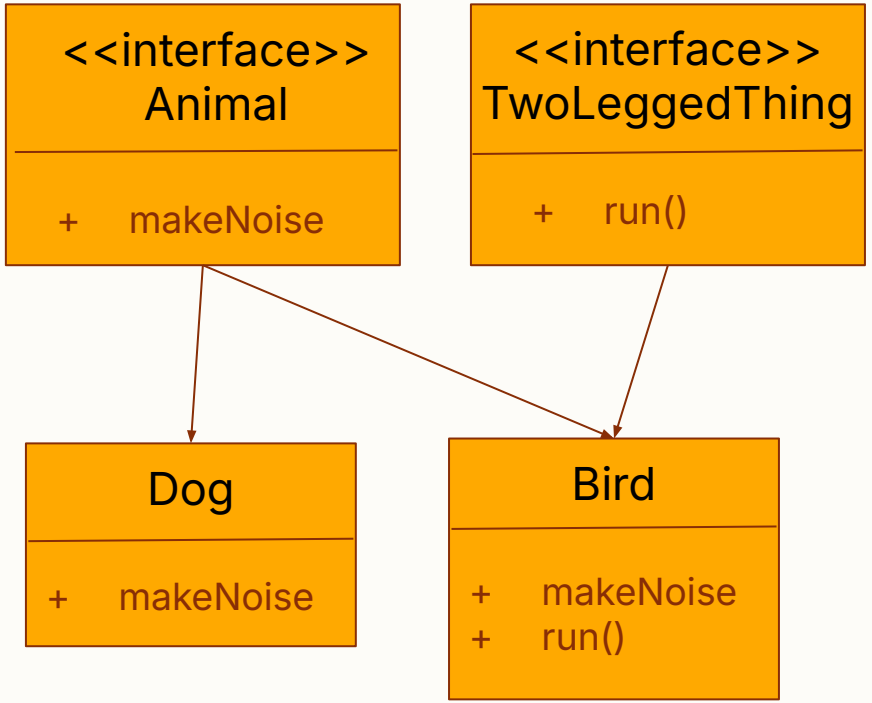
Dynamic Dispatch

- Declare variables as interfaces
- Let’s do it!



Dynamic Dispatch

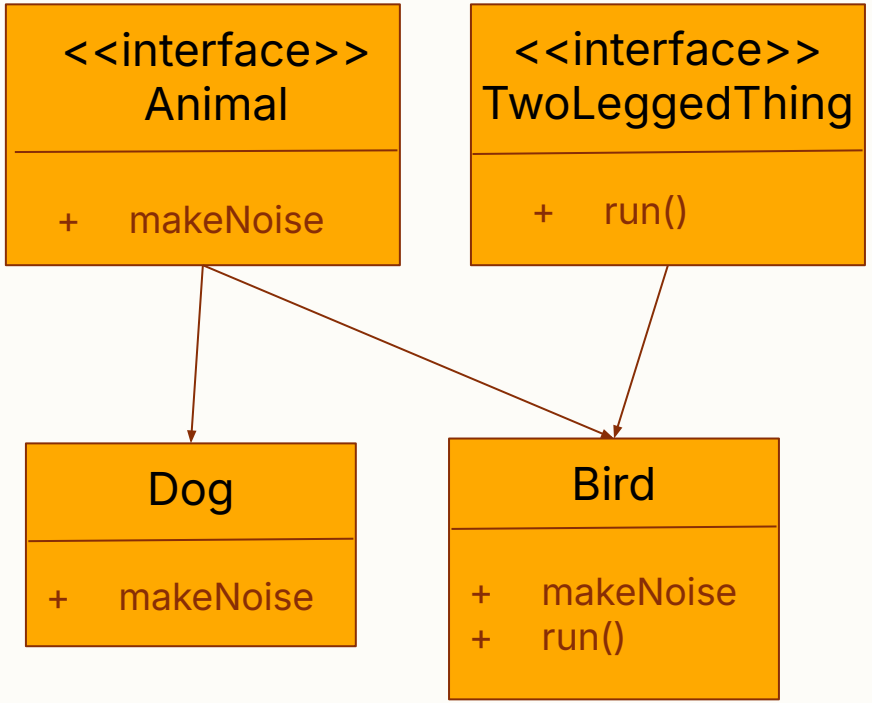
- This works both way
- Dog is not a TwoLeggedThing



Dynamic Dispatch

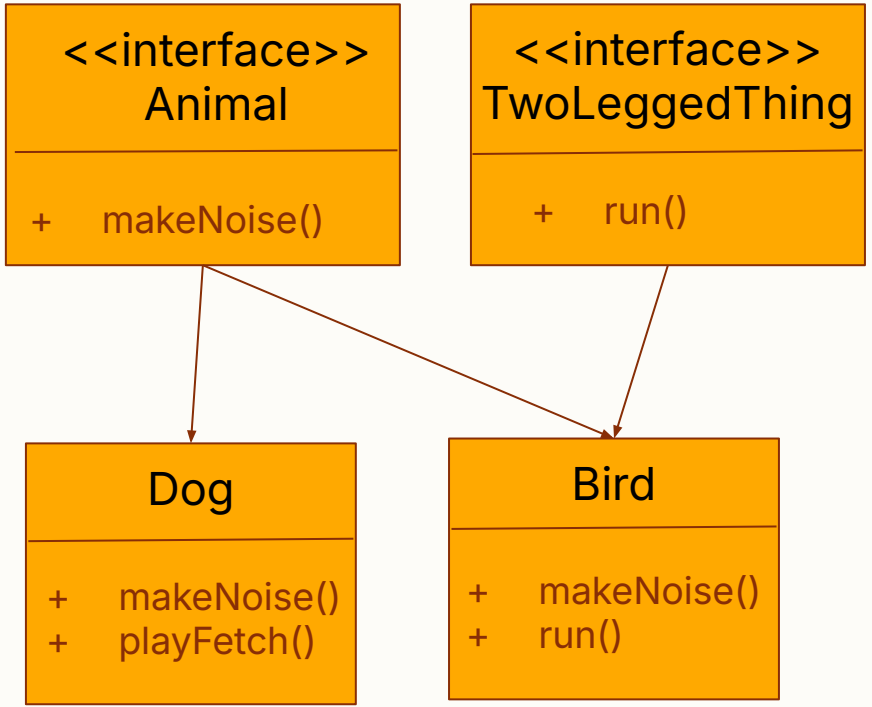
- We can also use this in declarations

public static void wakeBeast(Animal a)



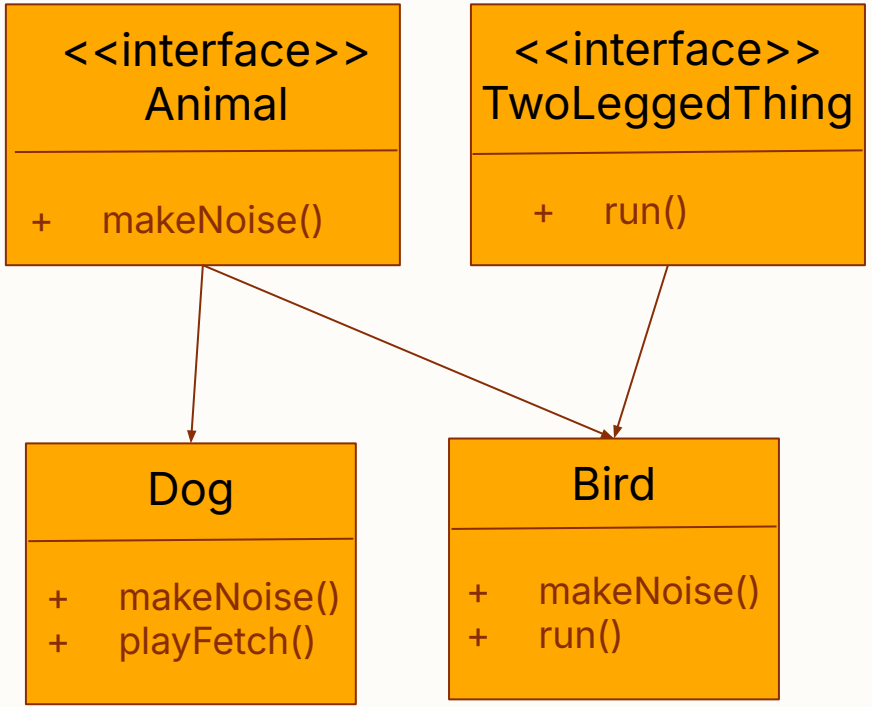
Dynamic Dispatch

- wakeBeast *cannot* use methods not defined in animal
- Even if Dog is passed



Dynamic Dispatch

- Extremely powerful construct
- Buzzword for this is
  - Strategy Pattern
- Interchangeable behavior by design



Compile vs Runtime Assignment

- Left side = Checked at compile time
- Right side = Checked at runtime

- Can we do this?

```
TwoLeggedThing bird = new Bird()
bird.makeNoise()
```

A secret, third option?

- We want a class the keeps Track of logs
- These logs are ordered
- Each log is a string
- Want method to print every other log

A secret, third option?

```
LogList ll = new LogList();
ll.addLog("Event 1 occurred");
ll.addLog("Event 2 occurred");
ll.addLog("Event 3 occurred");
ll.addLog("Event 4 occurred");
ll.addLog("Event 5 occurred");
ll.printEveryOther()
```

A secret, third option?

- What if I wanted my class to have List-like behavior?

A secret, third option?

- What if I wanted my class to have List-like behavior?

- List is an interface!
- ArrayList is a class!
- Lets go check some documentation

A secret, third option?

- How you would have handled this before 1181?
- Is polymorphism really the best option?

A secret, third option?

- How you would have handled this before 1181?
- Is polymorphism really the best option?

- Do we really want our list to have a remove()?

Type Composition

- Let’s just wrap the type instead!

```
public class LogList{
    private ArrayList<String>list = new ArrayList()

    ... Provide only methods we want to

}
```

When do I use this??

- Depends who you work for!
- “Favor composition over inheritance”
- As you code scales, they will both get messy

Brief Aside: Javadoc

- Why do we make you do this?
- Nobody handmade that documentation from earlier
- Generated from source code
- Let’s see it!