

LEC 01

CSE 123

Comparable; Inheritance

Questions during Class?
Raise hand or send here

sli.do #cse123



BEFORE WE START

Talk to your neighbors:

What is your favorite fall food?

Respond on sli.do!

Instructor: Nathan Brunelle

TAs:

Cora	Chris	Crystal	Anya	Evan
Jonah	Nhan	Saachi	Ishita	Sahana
Chloe	Shiven	Nate	Renee	Yuntong
Amiya	Shashmit	Akshaya	Vihan	Shreedhaarini
Roshan	Adiyan	Nancy	Rachel	Ananya
Ryuta	Mimi	Isaac	Megan	

Coming up...

-  Complete the [Introductory Survey](#)
 - This helps us gather data about the students taking our classes and their backgrounds, to inform future offerings.
-  The IPL opens Monday, October 5th
 - Schedule posted soon

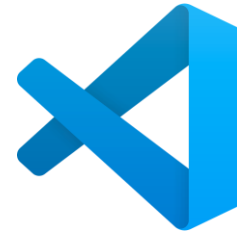
Course Tools



Ed

- Community & Information
 - Discussion Board
(please ask & answer!; anonymous option)
 - Announcements
- Pre-Class Materials / Section Activities
- Computer Quizzes
- Assignments
 - Online IDE
 - Submit assignments
 - View Feedback

My Digital Hand



My Digital Hand

- Queueing in office hours

VSCode

- Develop offline
- Visual debugger



Gradescope

- Paper Quiz grades



Sli.do

- In-class activities
(ungraded)

How Learning Works

- Learning requires **active participation** in the process. It's not as simple as sitting and listening to someone talk at you.
 - Requires **deliberate practice** in **learning by doing**
 - Benefits from **collaborative learning**
- Hybrid classroom model
 - Asks you to do some preparation before class in the form of readings and practice problems.
 - Should take ~30 minutes outside of class per lesson
 - Class will start with brief recap, then pick up where the reading and practice problems leave off.
 - Attendance isn't graded, but showing up and trying is the first step in succeeding in the class!
- For participation credit you only need to attempt the pre-class materials, but...
 - It's okay if you find them challenging! That means you are learning!



Learning Pattern

- Pre-class Work
 - Your first introduction to a topic
 - Interactive components
- In-class lessons
 - Recap of pre-class work
 - Further instruction from where it left off
 - Interactive components
- Section
 - Review, restate, rephrase content
 - Very interactive
- Homework
 - Learn by applying
 - Help when you need it!
- Quizzes
 - Show us what you've learned

Instruction

Practice

Instruction

Practice

Instruction

Practice

Practice/Assessment

Instruction

Assessment

Lecture Outline: Comparable

- Comparable ◀
- Inheritance

Comparable

- `Comparable<E>` is an interface that allows implementers to define an ordering between two objects
 - Used by `TreeSet`, `TreeMap`, `Collections.sort`, etc.
- One required method:

```
public int compareTo(E other);
```
- Returned integer falls into 1 of 3 categories
 - < 0: this is “less than” other
 - = 0: this is “equal to” other
 - > 0: this is “greater than” other

`a.compareTo(b);`

The diagram illustrates the parameters of the `compareTo` method. A blue arrow points from the variable `a` in the expression `a.compareTo(b)` down to the word `this`. Another blue arrow points from the variable `b` in the same expression down to the word `other`.

Subtraction Trick

- `compareTo` implementation when comparing two integers (a) ascending:

```
if (this.a < other.a)      -> negative number
else if (this.a > other.a) -> positive number
else                      -> 0
```

- This is just subtraction!

`this.a - other.a`

- What if we wanted to sort descending?

`other.a - this.a`

- **Warning**: this only works for integers! Doubles have issues with truncation.

`equals` should be consistent with `compareTo`!

The `equals()` method returns `true` if the given parameter is considered equal to this object, and `false` otherwise.

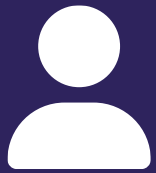
Used by lots of library methods! e.g. `contains`, `remove` for specific elements, etc.

Each class implicitly *extends* `Object`, so they inherit an `equals` method, but it checks for **reference equality**.
(Thanks?)

If you want `equals` to check for **value equality**, you need to write this method yourself.

Lecture Outline: Inheritance

- Comparable
- **Inheritance** ◀



Practice : Think

[sli.do](#)[#cse123](#)

Which of the following are true? (Select all that apply.)

- A. If ClassA extends ClassB, then ClassA is the superclass
- B. Classes can extend multiple classes (e.g. "ClassA extends ClassB, ClassC")
- C. Java will prevent you from writing "ClassA extends ClassB" unless ClassA "is-a" ClassB
- D. You can call a method from the superclass on an instance of the subclass



Practice : Pair

[sli.do](#)

#cse123

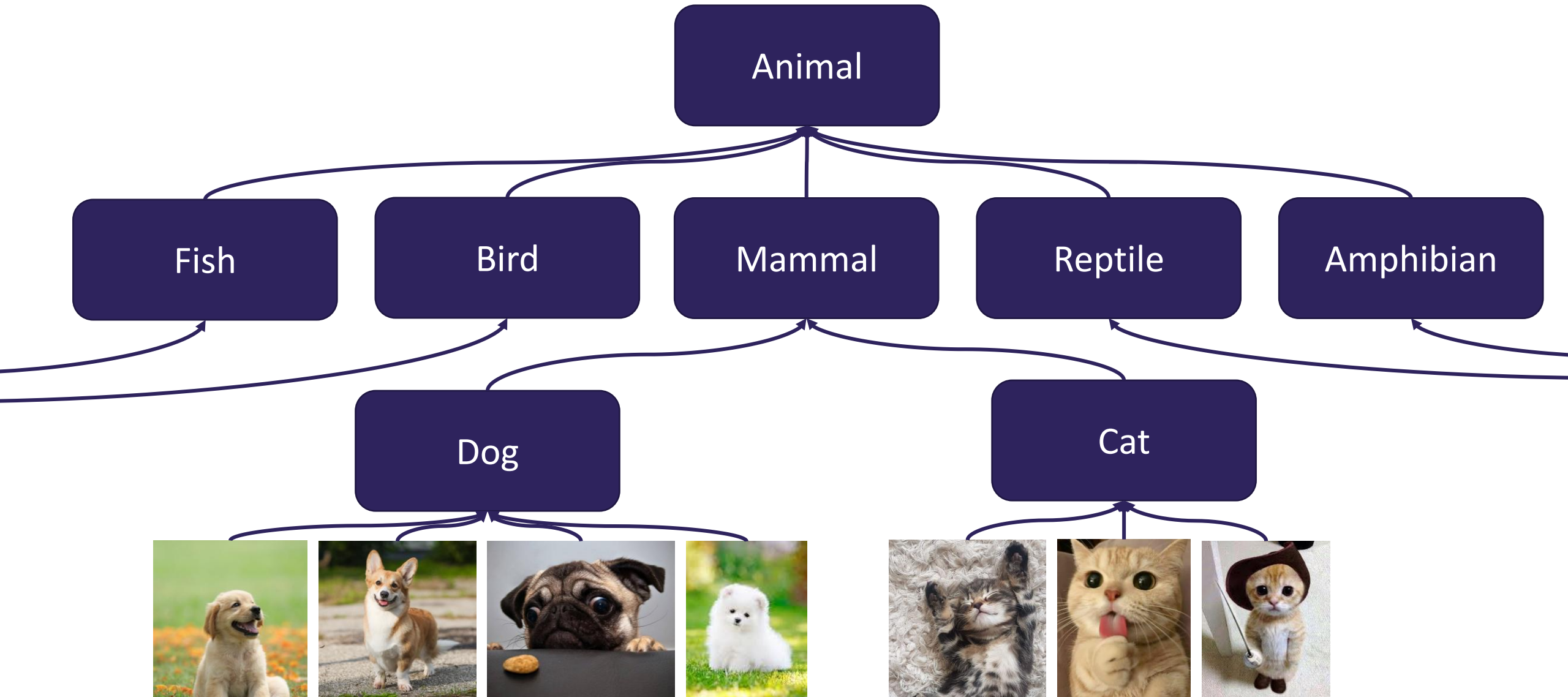
Which of the following are true? (Select all that apply.)

- A. If ClassA extends ClassB, then ClassA is the superclass
- B. Classes can extend multiple classes (e.g. "ClassA extends ClassB, ClassC")
- C. Java will prevent you from writing "ClassA extends ClassB" unless ClassA "is-a" ClassB
- D. You can call a method from the superclass on an instance of the subclass

Inheritance

- Connect together a “subclass” and “superclass”
 - Borrow / “inherit” code to reduce redundancy
 - `super()` keyword can be used just like `this()` (in a constructor)
- Syntax: `public class Subclass extends Superclass`
- *Should* Represent “is-a” relationships
 - `public class Chef extends Employee`
 - `public class Server extends Employee`
- In Java, all objects implicitly inherit from the `Object` class
 - `toString()`, `equals(Object)`, etc.

Is-a Relationships



The “is-a” Rule of Thumb

- class A should only extend class B if an A “is-a” B in the real world
- Java will *not* force you to follow this rule! That’s your job!

Inheritance Design: Typical Problem

- We give you:

Description of several classes, methods, and their behavior

- Your job:

Design an inheritance hierarchy that

- minimizes redundancy
- follows best practices (“is-a” test)

Inheritance Design: **Solution Technique**

- Use the “is-a” test to decide which classes should extend each other
- Keep an eye out for redundant code and consider whether you can reduce it even further using inheritance
 - Sometimes you can't, and that's ok!
 - The “is-a” test is more important than getting rid of all redundancy
- Often more than one reasonable answer!

What inheritance hierarchy did you come up with?