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 winter activity?*

**Respond on sli.do!**

**Instructor:** Brett Wortzman

**TAs:**

| | | | | |
|---|---|---|---|---|
| Arohan | Jonah | Kavya | Eeshani | Trien |
| Ashar | Brice | Misha | Aidan | Evan |
| Sean | Chris | Kieran | Cora | Rena |
| Chloe | Elden | Sahana | Dixon | Katharine |
| Jenny | Ishita | Anirudh | Nhan | Anya |
| Nate | Kuhu | Crystal | | |

*Now playing:* CSE 123 26wi Lecture Tunes

# 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, January 12th
  - Schedule posted soon
- 🔎 Creative Project 0: Search Engine out now
  - Due Wednesday, January 14th, 11:59pm

# Lecture Outline

- `Comparable`
- Inheritance
- Assignments and Grading

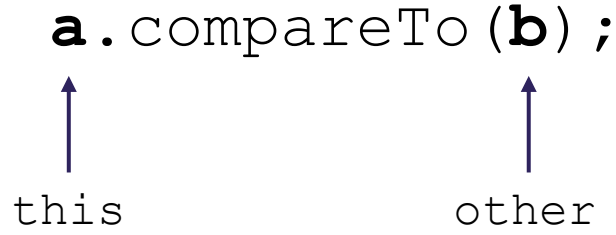
# Comparable

- `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`



# 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.

# Lecture Outline

- `Comparable`
- **Inheritance** 
- Assignments and Grading

# 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")

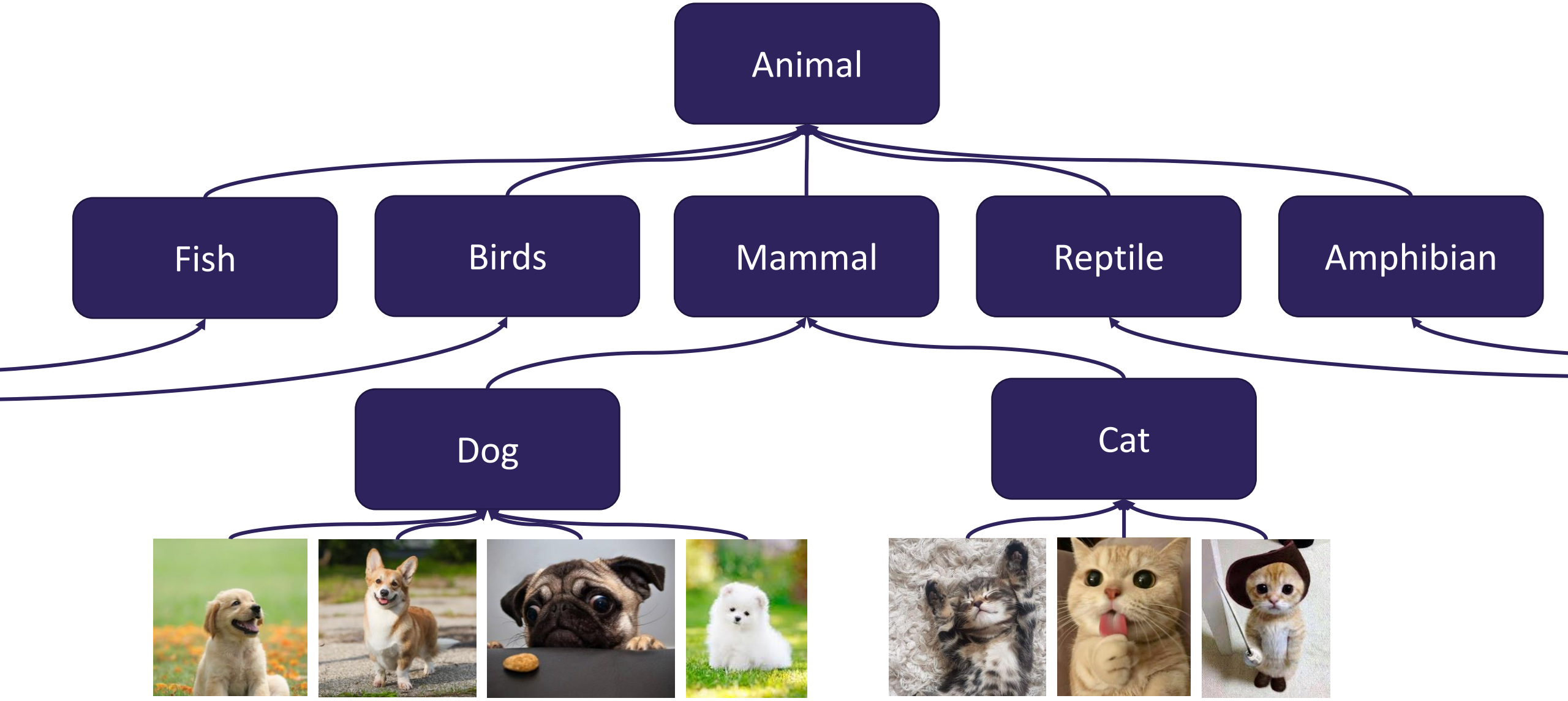
C. Java will prevent you from writing "ClassA extends ClassB" unless ClassA "is-a" ClassB

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!

# Lecture Outline

- `Comparable`
- Inheritance
- **Assignments and Grading**

# Assignments and Grading

- Our goal in the course is for you to **gain proficiency the concepts and skills** we teach
- We assess your proficiency by asking you to apply the concepts and skills on tasks or problems
- By necessity, we are assessing your *work* as a proxy for your proficiency

# Assignments

- Your learning in this course will be assessed in four ways:
  - Programming Assignments (~biweekly, 4 total)
    - Structured programming assignments to assess your proficiency of programming concepts
  - Creative Projects (~biweekly, 4 total)
    - Smaller, more open-ended assignments to give you space to explore
  - Quizzes (3 total, in section)
    - Series of problems covering all material up to that point
  - Final Exam (Tuesday, March 17, 12:30-2:20)
    - Final, culminating assessment of all your skills and knowledge

# Grading

*Grades should reflect your proficiency in the course objectives*

- All assignments will be graded **E (Excellent), S (Satisfactory), or N (Not yet)**
  - Under certain circumstances, a grade of U (Unassessable) may be assigned
- Final grades will be assigned based on the **amount of work at each level**

- See the syllabus for more details

# Resubmission and Ignored Quiz Problems

*Learning takes time, and doesn't always happen on the first try*

- One previous Programming Assignment or Creative Project can be **resubmitted** each week
  - Must be accompanied by a write-up describing changes (via Google Form)
  - Grade on resubmission will replace original grade
  - An assignment can be resubmitted in the 3 cycles after feedback has been published
  - *Tip: Resubmit as early as possible!*
- We will ignore your **two lowest quiz/exam problem grades**
  - No special action required– we'll do this automatically

- See the syllabus for more details

# Academic Honesty Policy

- When we assess your work in this class, we need to know that it's *yours*.
- Unless otherwise specified, **all graded work must be completed individually and without touching AI tools.**

Some specific rules to highlight:

- do not share your own solution code or view solution code from any source – including but not limited to other students, tutors, or the internet
- do not use AI tools (e.g. ChatGPT, Claude) to create or modify graded work

See the syllabus for more details (this is *very* important to understand).

# AI and CSE 123: Our Philosophy

Computing applications enabled by **artificial intelligence (AI)** are increasingly common and more widely used for a variety of tasks.

It is becoming more difficult to teach an introductory computing course without acknowledging the **existence of AI tools**.

But as relatively new programmers, **you still need to learn and practice effectively using core programming 'building blocks'**.

# CSE 123 AI Policy

***No part of any graded work may touch an AI tool.***

*You may not copy and paste any work generated by AI into any graded submission, nor may you copy and paste any work from or for a graded assignment into an AI tool. All other uses of AI on graded work must be cited.*

cs.uw.edu/123/syllabus/#using-ai-in-cse-123

# CSE 123 AI Policy

***No part of any graded work may touch an AI tool.***

*You may not copy and paste any work generated by AI into any graded submission, nor may you copy and paste any work from or for a graded assignment into an AI tool. All other uses of AI on graded work must be cited.*

**ALLOWED**

- Asking AI to **explain an error message**
- Asking AI to **explain the functionality of non-graded code** snippets
- Asking AI to **suggest additional information** or resources

**PROHIBITED**

- **Generating code, comments, reflections**
- Using AI to **'solve' an assignment**
- Using AI to **write, modify, or extend** reflections, code, comments, etc.

cs.uw.edu/123/syllabus/#using-ai-in-cse-123