LEC 01

CSE 123
Inheritance; Polymorphism; Comparable

Questions during Class?
Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Plans for the weekend?

Instructors: Brett Wortzman
Miya Natsuhara

TAs:

Arohan	Neha	Rushil	Johnathan	Nicholas
Sean	Hayden	Srihari	Benoit	Isayiah
Audrey	Chris	Andras	Jessica	Kavya
Cynthia	Shreya	Kieran	Rohan	Eeshani
Amy	Packard	Cora	Dixon	Nichole
Trien	Lawrence	Liza	Helena	

Music: CSE 123 25wi 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.
- 🙋 Consider attending the Review Session on Monday, Jan 13
  - 12:30pm – 1:20pm in ARC 147 ; 2:30pm – 3:20pm in GUG 220
  - Optional, hopefully recorded (waiting for confirmation)
- 💻 The IPL opens Monday, January 13
  - Schedule posted soon
- 🔎 Creative Project 0: Search Engine out now
  - Due Wednesday, January 15, 11:59pm

Collaboration 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.**

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) on graded work in any capacity

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

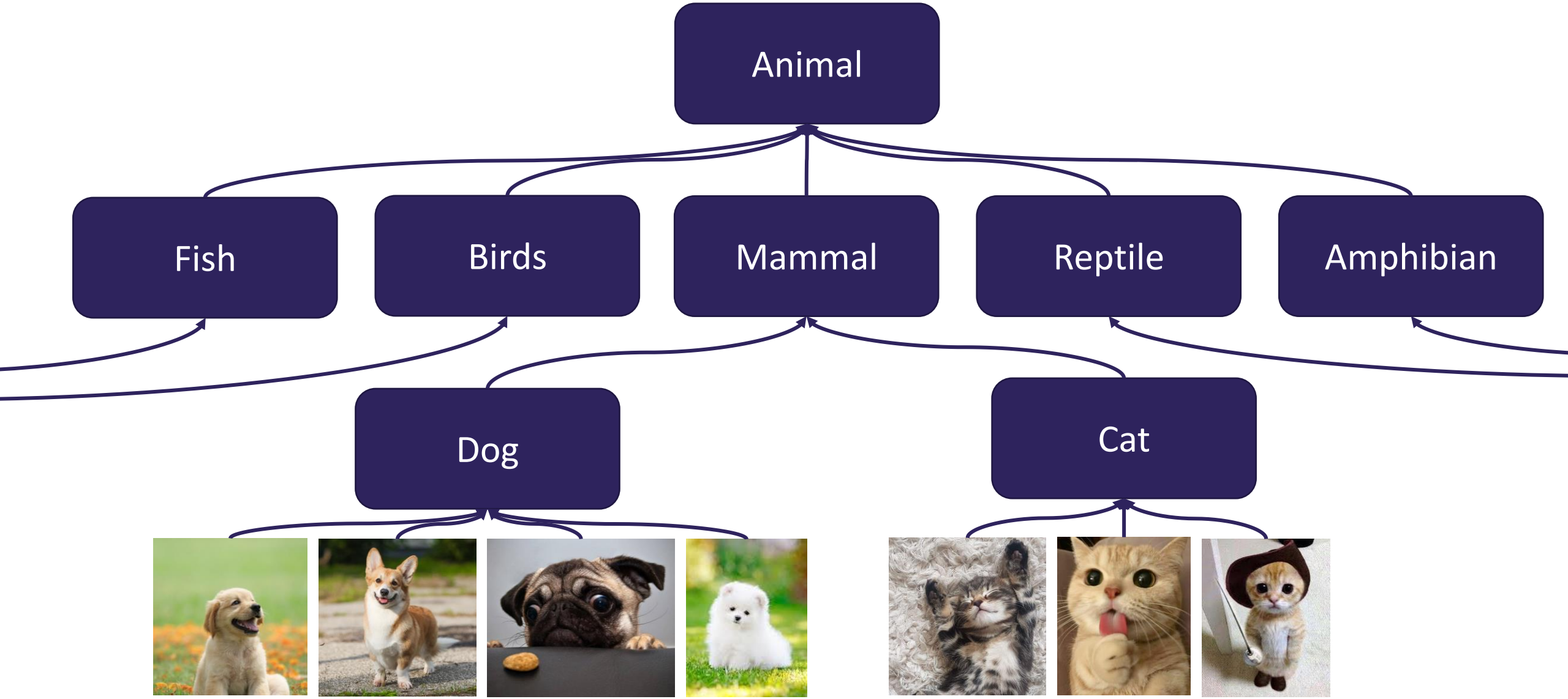
Lecture Outline

- Inheritance
- `Comparable`
- Polymorphism
  - Declared vs. Actual Type
  - Compiler vs. Runtime Errors

Inheritance

- Connect together a “subclass” and “superclass”
  - Borrow / “inherit” code to reduce redundancy
  - `super()` keyword can be used just like `this()`
- 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



PCM Review

sli.do #cse123

Lecture Outline

- Inheritance
- `Comparable`
- Polymorphism
  - Declared vs. Actual Type
  - Compiler vs. Runtime Errors

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);
```

this 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
```

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

Lecture Outline

- Inheritance
- `Comparable`
- **Polymorphism**
  - Declared vs. Actual Type
  - Compiler vs. Runtime Errors

Polymorphism

- `DeclaredType x = new ActualType()`
  - All methods in `DeclaredType` can be called on `x`
  - We've seen this with interfaces (`List<String>` vs. `ArrayList<String>`)
  - Can also be to inheritance relationships

```
Animal[] arr = {new Dog(), new Cat(), new Bear()};
for (Animal a : arr) {
    a.feed();
}
```

Compiler vs. Runtime Errors

- `DeclaredType x = new ActualType()`
  - At compile time, Java only knows `DeclaredType`
  - Compiler error: trying to call a method that isn't present

```
Animal a = new Dog();
a.bark();                    // No bark() -> CE
```

  - Can cast to change the `DeclaredType` of an object

```
((Dog) a).bark();            // No more CE
```

  - Runtime error: attempting to cast to an invalid `DeclaredType*`

```
Animal a = new Fish();
((Dog) a).bark();            // Can't cast -> RE
```

  - Order matters! Compilation before runtime

Compiler vs. Runtime Errors

With the following declaration and initialization:

```
DeclaredType name = new ActualType();
```

If we call:

```
name.method();
```

Does **DeclaredType** contain **method**?

- Yes → Execute **method** of **ActualType**
- No → Compiler Error

```
((CastToType) name).method();
```

Is it conceivable that an instance of **DeclaredType** could be a **CastToType**? *

*This is true if either **DeclaredType** or at least one of its subtypes can substitute for a **CastToType**

- No → Compiler Error
- Yes → Does **CastToType** contain **method**?
  - No → Compiler Error
  - Yes → Does **ActualType** cast to **CastToType**?
    - Yes → Execute **method** of **ActualType**
    - No → Runtime Error