COMP 333 Lecture 2

Kyle Dewey

Object-Oriented Programming (OOP)

OOP (Minimal Definition)

- *Objects* contain *fields* holding data
- Objects can pass *messages* to each other

OOP (Explicit Methods)

- *Objects* contain *fields* holding data and *methods* holding executable procedures
- ~~Objects can pass *messages* to each other~~
- Objects can call methods on other objects/ have their methods called on

OOP (As Commonly Understood)

- *Objects* contain *fields* holding data and *methods* holding executable procedures
- Objects can call methods on other objects/have their methods called on
- Objects *encapsulate* their state using *access modifiers*
- On a call, the correct method may be chosen at runtime, which is a form of *polymorphism*
- Methods can be *overridden*, allowing for more specific behavior
- *Abstraction* allows for *interfaces* to contain only immediately relevant information
- *Classes* define a template to make objects from
- Classes may *inherit* fields and methods from other classes
- *All ideas that were ever good* are object-oriented

OOP (As Commonly Understood)

- *Objects* contain *fields* holding data and *methods* holding executable procedures
- Objects can call methods on other objects/have their methods called on
- Objects *encapsulate* their state using *access modifiers*
- On a call, the correct method may be chosen at runtime, which is a form of *polymorphism*
- Methods can be *overridden*, allowing for more specific behavior
- *Abstraction* allows for *interfaces* to contain only immediately relevant information
- *Classes* define a template to make objects from
- Classes may *inherit* fields and methods from other classes
- *All ideas that were ever good* are object-oriented

Not specific to OOP

OOP (As Commonly Understood)

- *Objects* contain *fields* holding data and *methods* holding executable procedures
- Objects can call methods on other objects/have their methods called on
- Objects *encapsulate* their state using *access modifiers*
- On a call, the correct method may be chosen at runtime, which is a form of *polymorphism*
- Methods can be *overridden*, allowing for more specific behavior
- *Abstraction* allows for *interfaces* to contain only immediately relevant information
- *Classes* define a template to make objects from
- Classes may *inherit* fields and methods from other classes
- *All ideas that were ever good* are object-oriented

Specific to class-based OOP

OOP (As Commonly Understood)

- *Objects* contain *fields* holding data and *methods* holding executable procedures
- Objects can call methods on other objects/have their methods called on
- Objects *encapsulate* their state using *access modifiers*
- On a call, the correct method may be chosen at runtime, which is a form of *polymorphism*
- Methods can be *overridden*, allowing for more specific behavior
- *Abstraction* allows for *interfaces* to contain only immediately relevant information
- *Classes* define a template to make objects from
- Classes may *inherit* fields and methods from other classes
- *All ideas that were ever good* are object-oriented

Many OOP languages do not support this

OOP (As Commonly Understood)

- *Objects* contain *fields* holding data and *methods* holding executable procedures
- Objects can call methods on other objects/have their methods called on
- Objects *encapsulate* their state using *access modifiers*
- On a call, the correct method may be chosen at runtime, which is a form of *polymorphism*
- Methods can be *overridden*, allowing for more specific behavior
- *Abstraction* allows for *interfaces* to contain only immediately relevant information
- *Classes* define a template to make objects from
- Classes may *inherit* fields and methods from other classes
- *All ideas that were ever good* are object-oriented

Often considered a bad idea

OOP (As Commonly Understood)

- *Objects* contain *fields* holding data and *methods* holding executable procedures
- Objects can call methods on other objects/have their methods called on
- Objects *encapsulate* their state using *access modifiers*
- On a call, the correct method may be chosen at runtime, which is a form of *polymorphism*
- Methods can be *overridden*, allowing for more specific behavior
- *Abstraction* allows for *interfaces* to contain only immediately relevant information
- *Classes* define a template to make objects from
- Classes may *inherit* fields and methods from other classes
- *All ideas that were ever good* are object-oriented

OOP was originally heralded as a silver bullet

Core Concept: Virtual Dispatch

Virtual Dispatch

- AKA dynamic dispatch, polymorphism
- The method/code actually called is determined at runtime

Virtual Dispatch

Example in Java

Virtual Dispatch Use

- Allows for abstracting over computation
- The computation itself becomes a parameter

Virtual Dispatch Use

- Allows for abstracting over computation
- The computation itself becomes a parameter

```
void foo(SortRoutine s) { ... }
```

Virtual Dispatch Use

- Allows for abstracting over computation
- The computation itself becomes a parameter

```
void foo(SortRoutine s) { ... }
```

```
foo(new InsertionSort());
foo(new MergeSort());
```

Virtual Dispatch vs. `if`

- Both conditionally execute code
    - `if`: based on if condition is true/false
    - Virtual dispatch: based on the specific runtime method passed
- `if`'s that are used to select between different code behaviors are undesirable
- Smalltalk has `ifTrue:ifFalse:` method on its boolean type

Exercise: Virtual Dispatch

Virtual Dispatch in...C?

```
void qsort(
   void* base,
   size_t num,
   size_t size,
   int (*comparator)(const void*,
                     const void*));
```

Prototype-Based Inheritance

Classes vs. Prototypes

- Classes: classes inherit from other classes
- Prototypes: objects inherit from other objects
- Since objects can be mutated, prototypes allow:
    - Dynamically adding or removing inherited methods
    - Dynamically changing hierarchies
- Much more flexible than classes

Demo: Prototype-Based Inheritance in JavaScript

Exercise: Prototype-Based Inheritance in JavaScript