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

-Notably, this definition **doesn't** include words like method, class, encapsulation, polymorphism

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

-More specific. Note that calling a method isn't necessarily straightforward - we might not have the method, we might have a backup plan if we don't have the method, and determining the correct method may be complex

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 good ideas* 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 good ideas* are object-oriented

Not specific to OOP

-Encapsulation is possible in C
-Anything with higher-order functions allows polymorphism
-Typeclasses (which are unrelated to OOP classes) allow overriding and inheritance
-Abstraction existed before computers did

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 good ideas* are object-oriented

Specific to class-based OOP

-Prototype-based OOP doesn't have classes

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 good ideas* are object-oriented

Many OOP languages do not support this

-Commonly dynamic languages don't support proper encapsulation (Python, Ruby)

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 good ideas* are object-oriented

Often considered a bad idea

-Commonly dynamic languages don't support proper encapsulation (Python, Ruby)

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 good ideas* 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 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) { ... }
```

-For example, I can define a method that takes a sorting routine...

Virtual Dispatch Use

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

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

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

-...and then call it with different sorting routines
-InsertionSort makes sense on data that you know to be nearly sorted, and MergeSort works best when the data is not nearly sorted

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

Virtual Dispatch Example in Java