LEC 11

CSE 122

Introduction to Objects

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Slido vote & chat with your neighbors:

What are your favorite places to study on/near campus?

Music: [122 26Wi Lecture Tunes ⛄](#)

Instructor: Adrian Salguero

TAs:

Ava	Dalton	Neal	Shreyank
Blake R	Dani	Neha	Sthiti
Blake P	David	Nicolae	Sushma
Cady	Diya	Nicole	Suyash
Caleb	Hanna	Rio	TJ
Cole	Ivy	Rohan	Wesley
Colin	Mahima	Saachi	Yang
Connor	Medha	Shreya	

Lecture Outline

- **Announcements**
- OOP Review
- Example
- Abstraction

Announcements

- Programming Assignment 2 (P2) out
  - Due Tuesday, Feb 17th by 11:59pm PT
- Quiz 1 on Tuesday, Feb 17th in your registered quiz section
  - Practice quiz out tonight—Solutions Sunday!
- Resubmission Cycle 3 (R3) out
  - Due Tuesday, Feb 17th by 11:59pm PT
  - Eligible assignments: **P0**, C1, P1

Lecture Outline

- Announcements
- **OOP Review**
- Example
- Abstraction

Object Oriented Programming (OOP)

- **Procedural programming**: Programs that perform their behavior as a series of steps to be carried out
  - Classes that do things

- **Object-oriented programming (OOP):** Programs that perform their behavior as interactions between objects
  - Classes that represent things
  - We're going to start writing our own objects!

Classes & Objects

- **Classes** can define the template for an object
  - 📰 Like the blueprint for a house!
    "What does it mean to be this thing?"

- **Objects** are the actual instances of the class
  - 🏠 Like the actual house built from the blueprint!
    "It is an example of this thing!"

We create a new instance of a class with the `new` keyword

e.g., `Scanner console = new Scanner(System.in);`

State & Behavior

- **Objects** can tie related *state* and *behavior* together

- **State** is defined by the object's *fields* or *instance variables*
  - *Scanner's state may include what it's scanning, where it is in the input, etc.*

- **Behavior** is defined by the object's *instance methods*
  - *Scanner's behavior includes "getting the next token and returning it as an `int`", "returning whether there is a next token or not", etc.*

Syntax

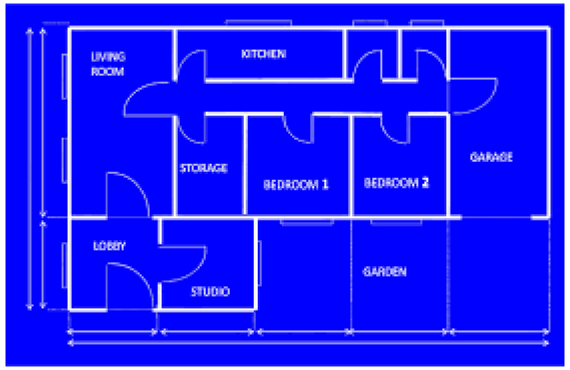
```
public class MyObject {
    // fields (or instance variables)
    type1 fieldName1;
    type2 fieldName2;
    ...

    // instance methods
    public returnType methodName(...) {
        ...
    }
}
```

Instance Variables

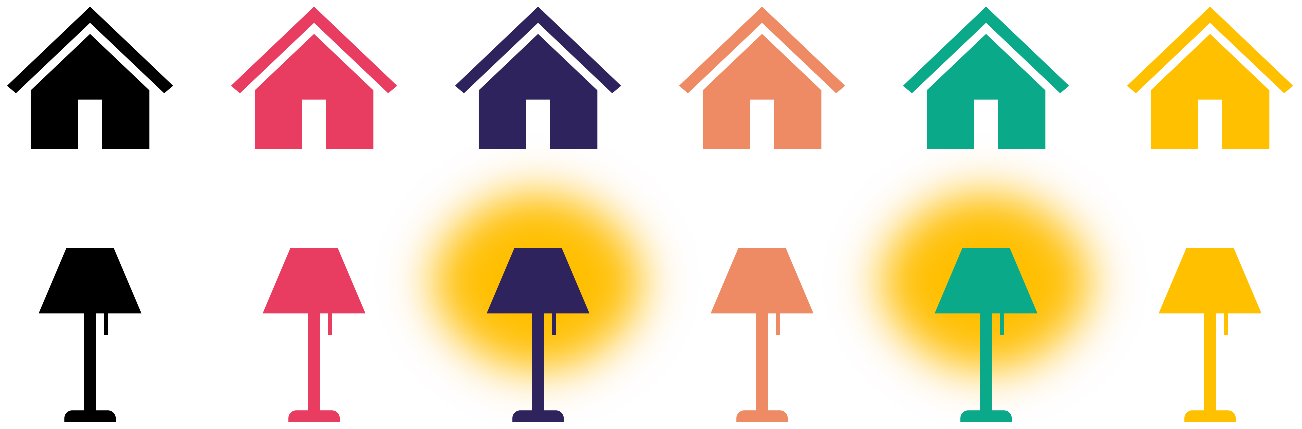
- Fields are also referred to as **instance variables**

- Fields are defined in a class
- Each instance of the class has their own copy of the fields
  - Hence *instance* variable! It's a variable tied to a **specific** instance of the class!



Instance Methods

- **Instance methods** are defined in a class
- Calling an instance method on a particular *instance* of the class will have effects only on that instance



Lecture Outline

- Announcements
- OOP Review
- **Example**
- Abstraction

Representing a Coordinate Point

How would we do this given what we knew last week?

Maybe `int x`, `int y`?

Maybe `int[]`?

Representing a point

`int x, int y`

- Easy to mix up x, y
- Just two random `ints` floating around – easy to make
  mis

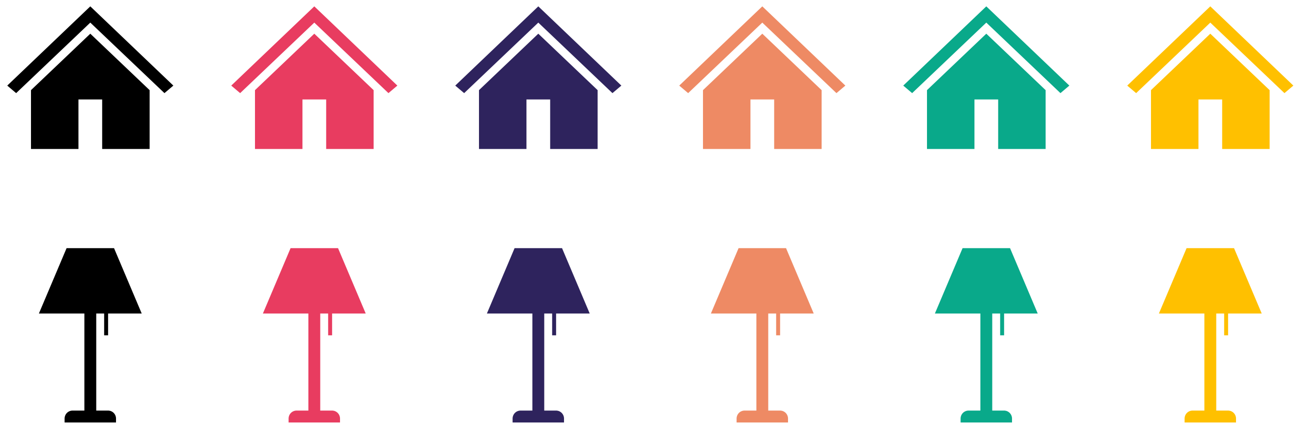
`int`

Let's make a class instead!

- Not really what an array is for
- Again, just two `ints` – just have to "trust" that we'll remember to treat it like a point

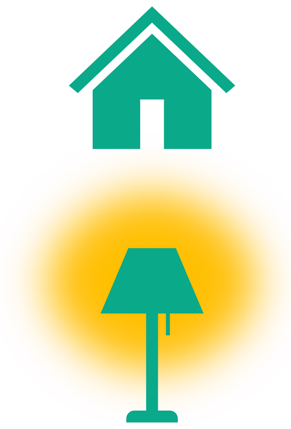
Instance Methods

- **Instance methods** are defined in a class
- Calling an instance method on a particular *instance* of the class will have effects on <u>that</u> instance



Instance Methods

- **Instance methods** are defined in a class
- Calling an instance method on a particular *instance* of the class will have effects only on that instance



Lecture Outline

- Announcements
- OOP Review
- Example
- **Abstraction**

Abstraction

The separation of ideas from details, meaning that we can <u>use</u> something without knowing exactly <u>how</u> it works.

You were able use the `Scanner` class without understanding how it works internally!

Client v. Implementor

We have been the clients of many objects this quarter!

Now we will become the implementors of our own objects!