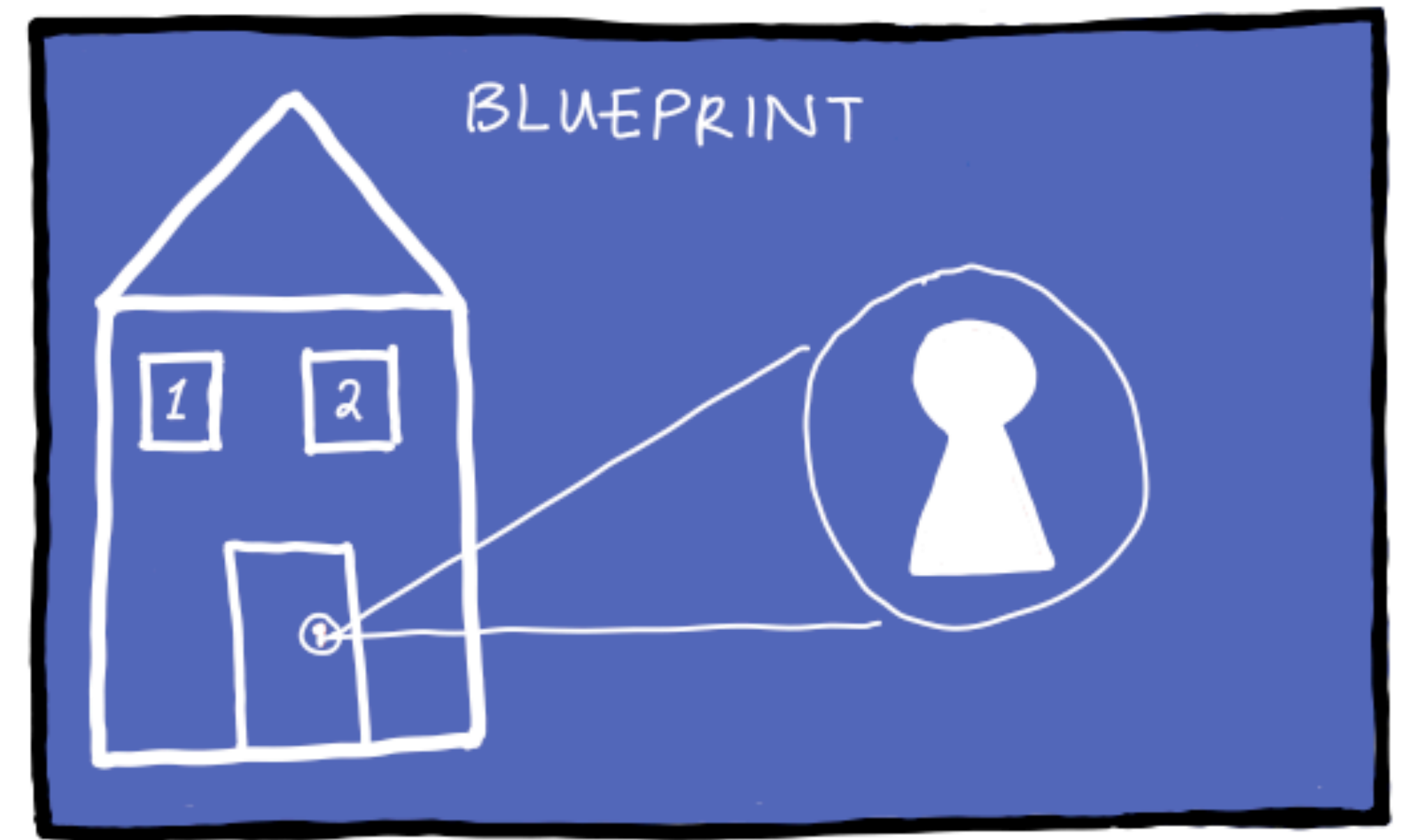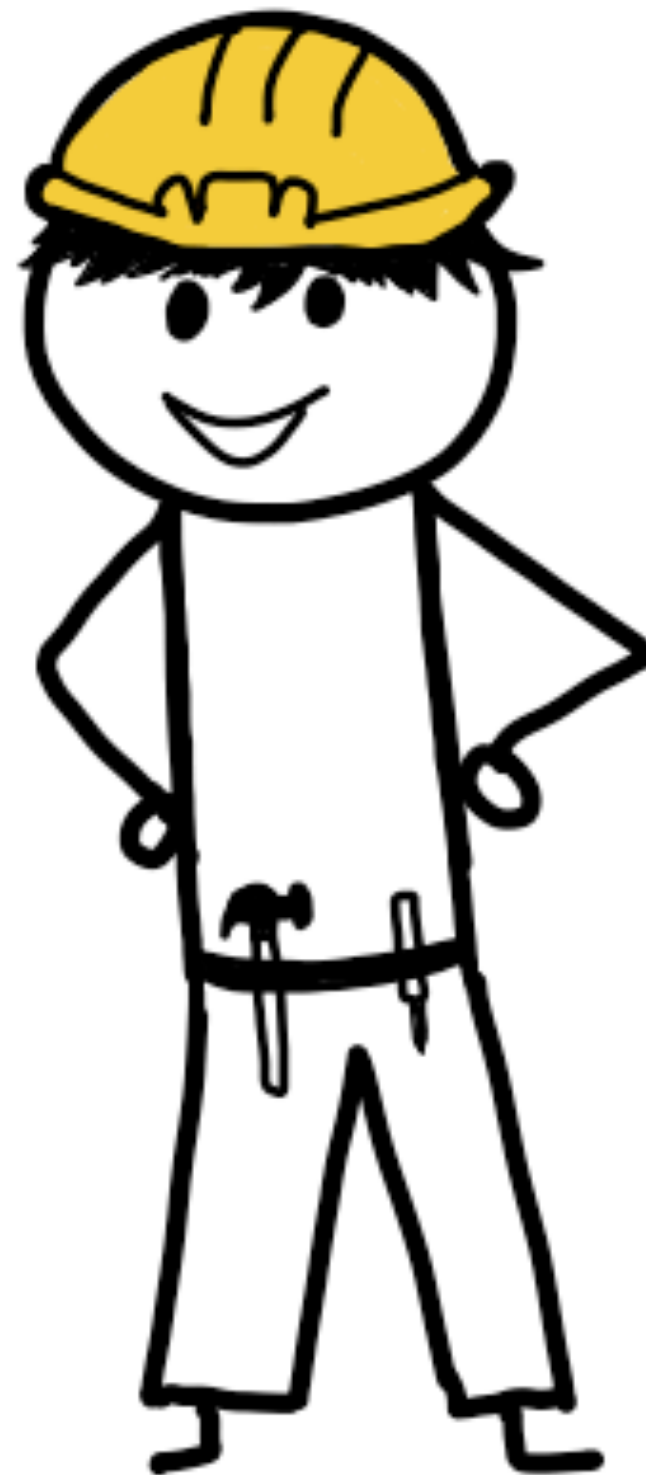# Object-Oriented Programming

April 13, 2021

Classes

High-Level

Imagine I'm opening a residential construction company which is going to build several houses...

First, I need a blueprint for a house. This is the **class object**.
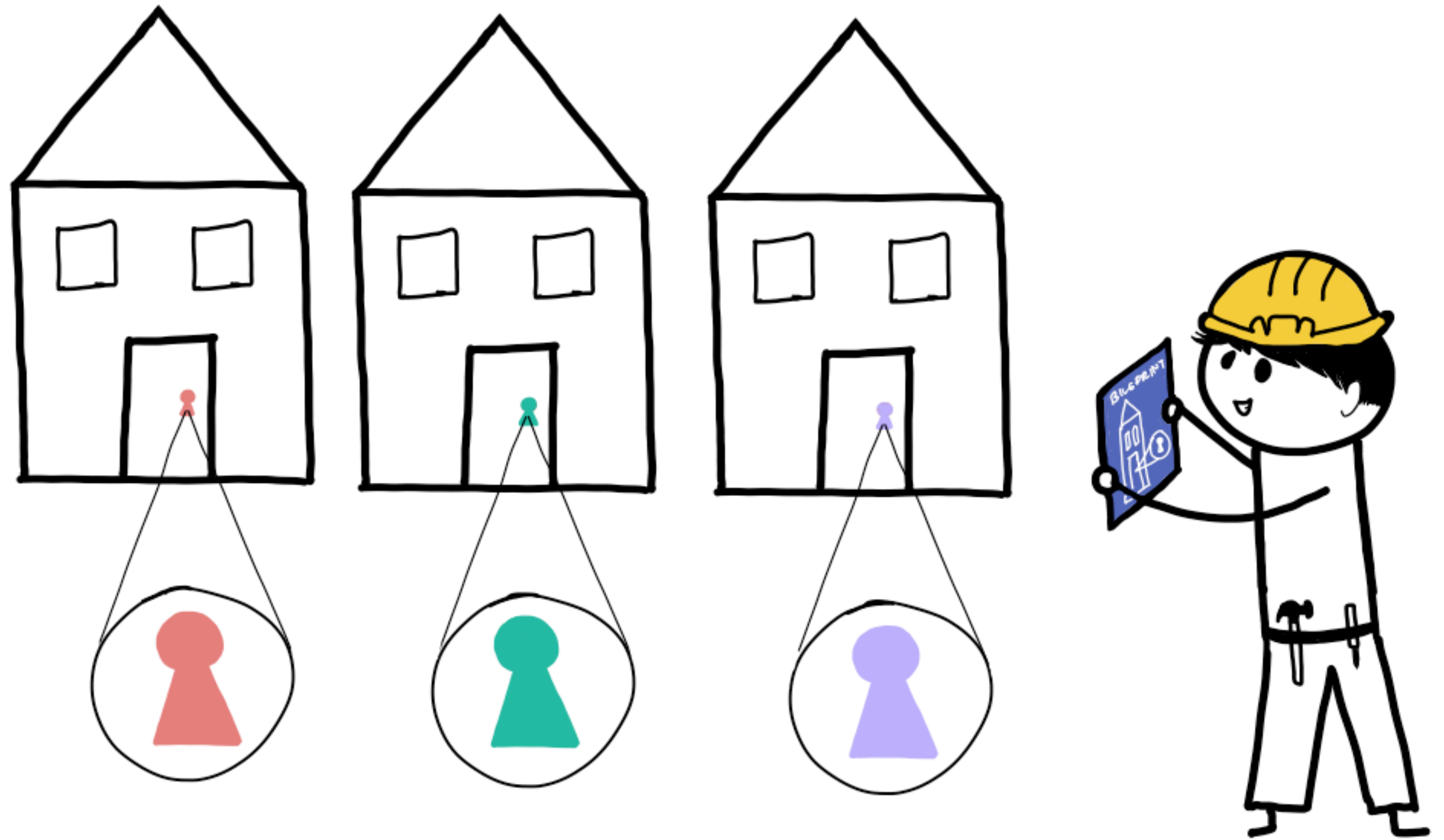
✨ btw y'all, my sister made these! 💜

High-Level

Then, I can use that blueprint to build several houses. Some properties of the houses will be the same and others will be different.

Each house is **an instance (object)** of the class.

High-Level

The blueprint for a house

```python
class House:
    utilities = {
        'electricity': 'A&E #8675309',
        'water': 'Palo Alto Mutual #6054756961'
    }

    def __init__(self):
        self.locked = True
```

These attributes are shared among the instances (houses)

This is run every time an instance is declared and sets up instance-specific properties (it's the "constructor")

High-Level

The actual houses

```python
red = House()
blue = House()
green = House()

House.utilities['electricity']  # => 'A&E #8675309'
red.utilities['electricity']    # => 'A&E #8675309'
green.utilities['electricity']  # => 'A&E #8675309'


red.locked   # => True
blue.locked  # => True

red.locked = False
blue.locked  # => True
```

Note: In Python, all attributes are public

```python
class House:
    utilities = {
        'electricity': 'A&E #8675309',
        'water': 'Palo Alto Mutual #6054756961'
    }

    def __init__(self):
        self.locked = True
```

But wait... what's `self`?

```python
class House:
    def __init__(self):
        self.locked = True
```

When the function is run on a class instance, the first parameter to every method is a reference to the object itself. It could be named anything, but `self` is the traditional name.

```python
House.__init__  # => <function __init__(self)>

red = House()
red.__init__  # => <bound method House.__init__>
```

This applies to other methods as well, not just `__init__`.

```python
instance.method(some args) ~ function(instance, some args)
```

Custom Instantiation

```python
class Student:
    def __init__(self, name, sunet):
        self.name = name.title()

        # validate the SUNet
        if not set(sunet) <= set('0123456789'):
            raise ValueError(f"Invalid SUNet: {sunet}.")
        self.sunet = sunet
```

> Just like a normal function, __init__ can have parameters!

```python
parth = Student('parth sarin', 'noneya') # ValueError

pop_tart = Student('pop tart', '5625165')
pop_tart.name # => 'Pop Tart'
```

Magic Methods

Python Uses Magic Methods!

```python
str(x)   # => x.__str__()


x == y   # => x.__eq__(y)
x < y    # => x.__lt__(y)
x + y    # => x.__add__(y)


next(x) # => x.__next__()
len(x)   # => x.__len__()
hash(x) # => x.__hash__()


el in x # => x.__contains__(el)
```

Full list [here](.)!