OO Programming

Learning Objectives

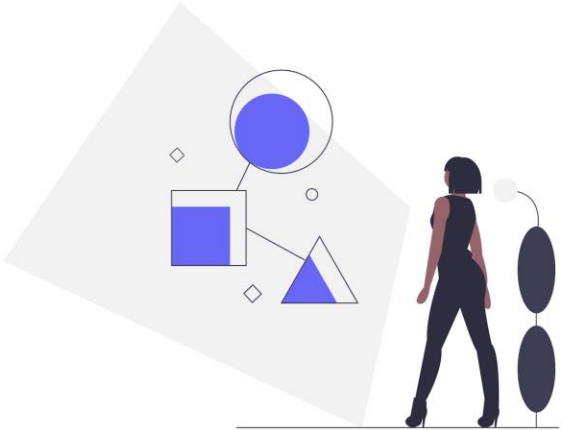
- What is OO programming
  - When to use OO
  - OO vs Functional
- Class vs Object Recap

Why - OOP

Abstract real world into code

Programming Paradigms

SmallTalk



Classes

Objects

4 Pillars

What is OO Programming?

- OO is a programming paradigm (way of programming) that is based on the concepts of "objects", which may contain data, often referred to as properties; and code, in the form of procedures, often known as methods.
    - We combine a group of related variables (properties) and functions (methods) into a unit, called an Object.
- It is a style of programming, NOT a programming language
    - It is supported by lot's of languages like:
        - JavaScript
        - Python
        - Java
        - C#

What are Classes?

- Classes are a template for creating objects.
- They encapsulate data with code to work on that data.

- As we mentioned, classes are templates
  - We create classes, like a blueprint for a house, that includes all the “dimensions” and “features” and then when we actually build the house we create something called an Object…
  - A template for a Person would include their features, and would have many “people” who are Objects

What are Objects?

- Objects are instances of a Class.
- We would have one Class, called say “Person”, and many instances of that Person
  - Sam, Andy, Bex, Ollie… etc would all be instances of the Person class

- Being an instance of the Person class means that we would all have the features that were defined on that class
  - In this example that could be
    - Name
    - Age
    - Eye Colour
    - Height
    - Gender
    - Etc, etc

Benefits of OO Programming Summary

- Encapsulation
  - Reduce complexity and increase reusability
- Abstraction
  - Reduce complexity and reduce impact of changes
- Inheritance
  - Eliminate redundant code
- Polymorphism
  - Refactor/avoid lengthy switch cases

Inheritance

```
class User {
  constructor(name, email) {
    this.name = name;
    this.email = email;
    this.score = 0;
  }

  login() {…
  }

  logout() {…
  }
}
```

```
const user1 = new User('Sam', 'sam@nology.io');
const user2 = new User('Ollie', "ollie@nology.io");
```

We have a class of User that represnts user of our app... each one has

- Name
- Email
- A default starting score of 0

```
class Admin extends User {
  deleteUser(user) {…
  }
}
```

```
const users = [user1, user2];
```

We want to make an admin that will still be a User, but also have certain privileges (like deleting users)

We can extend the User class to inherit the properties such as name and email etc, but then also define new methods on the Admin class that aren't applied to User, only Admin.

DeleteUser filters through an array of users and deletes the relevant one.

Encapsulation

- Encapsulation – Hide/encapuslate properties within the class from outside access – which makes it great for security reasons, along with leaky, unwanted or accidental usage.
  - Think big code base, lot's of developer working together… we don't want people unnecessarily interacting with our code

```
console.log(user1.login()); // will return sam@nology.io has logged in.
console.log(login()); // will give a ReferenceError  of login not defined as we are outside the Class

console.log(user2.name); // will return Ollie
console.log(name); // will give ReferenceError as name is not defined outside of the Class
```

Abstraction

- We can hide the details (implemntation), and only show the essentials to the user

```
class User {
  constructor(firstName, lastName, email) {
    this._firstName = firstName;
    this._lastName = lastName;
    this.email = email;
    this.score = 0;
  }

  get name() {
    return (this._firstName + " " + this._lastName).toUpperCase();
  }

  set name(newName) {
    const namesArr = newName.split(' ');
    this._firstName = namesArr[0];
    this._lastName = namesArr[1];
  }

  login() {…
  }

  logout() {…
  }
}
```

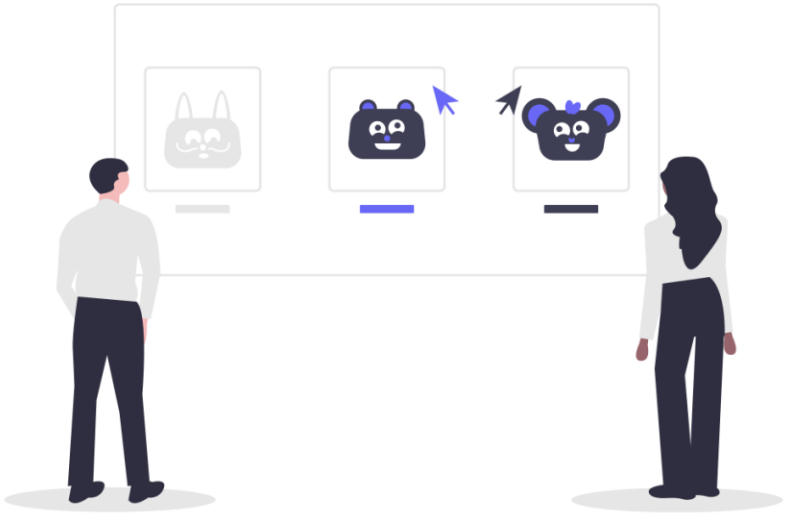
```
const userUno = new User("Sam","Joyce", "sam@nology.io");

console.log(userUno); // will show first name as Sam
userUno.name = "Samuel Joyce"; // using the setter to change the
first name
console.log(userUno.name); // will show name as SAMUEL JOYCE
console.log(userUno); // change will be reflected in a log of the
whole Object
```

Polymorphism

Polymorphism is the ability for an Object to take many forms.

For example, a Translator Class can take on the form of..

- A French translator
- A Spanish translator
- A Morse Code translator
- Etc...



Challenges

- make a Class of Employee with the following
- has properties of
    - name
    - rating (0-10)
    - grade
    - nextGrade
- method of
    - how likely they are to be promoted that resolves to
        - if rating is 7 or higher, they are likely to be promoted to next grade
        - if not, they are staying put

Challenges

- Start with a base class of Vehicle, then create a Car class that inherits from this base class.
- Finally create another class, a specific type of Car (VW, Mercedes, Audi etc) that inherits from the Car class.
- You want to be able to handle steering, changing gears, and moving (speed).
- You need to decide where to put the appropriate state/behavious, at what level does it make sense to place them.
- Changing gears and increasing/decreasing speed should be included.
- Finally, for the specific car you choose, you need to add something specific to for that type of car.

Challenges

- Build a Class of Translator – this will be extended multiple times for lots of different types of translators.

- Build 2 further Classes (Morse & French/Spanish/German etc) that extend from the translator class and inherit any properties/methods you think it should have.

- Build these translators out to be fully functioning.
- The definition of done is when I (the user) can visit you hosted site, input something in English and receive back the equivalent in, for example, Morse Code.

Further Reading

- https://medium.com/javascript-scene/the-forgotten-history-of-oop-88d71b9b2d9f
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS