Module 13 Day 1

Lesson 1

What is Object Oriented Programming?

Source: [JavaScript and Object-Oriented Programming - Rainer Hahnekamp]

Here are some key concepts in OOP:

1. **Classes:** A class is a blueprint for creating objects with a particular set of attributes and methods. It is a template for creating objects of the same type.
2. **Objects:** An object is an instance of a class. It has its own set of attributes and methods that are defined by the class.
3. **Encapsulation:** Encapsulation is the practice of hiding the internal workings of an object from the outside world. It helps to prevent unintended changes to an object's state.
4. **Inheritance:** Inheritance is the process of creating new classes based on existing ones. The new class inherits the attributes and methods of the existing class and can add new ones or override existing ones.
5. **Polymorphism:** Polymorphism is the ability of objects of different classes to be used interchangeably. It allows for more flexible and extensible code.

OOP has several advantages, including:

- Encapsulation makes it easier to manage code and prevent bugs.
- Inheritance and polymorphism allow for code reuse and flexibility.
- Objects provide a natural way to model real-world entities and systems.

Overview & Syntax of Classes

https://codepen.io/shafferma08/pen/xxydvqJ

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  } // end constructor

  value() {
    return "(" + this.x + ", " + this.y + ")";
  } // end value
} // end class Point
```

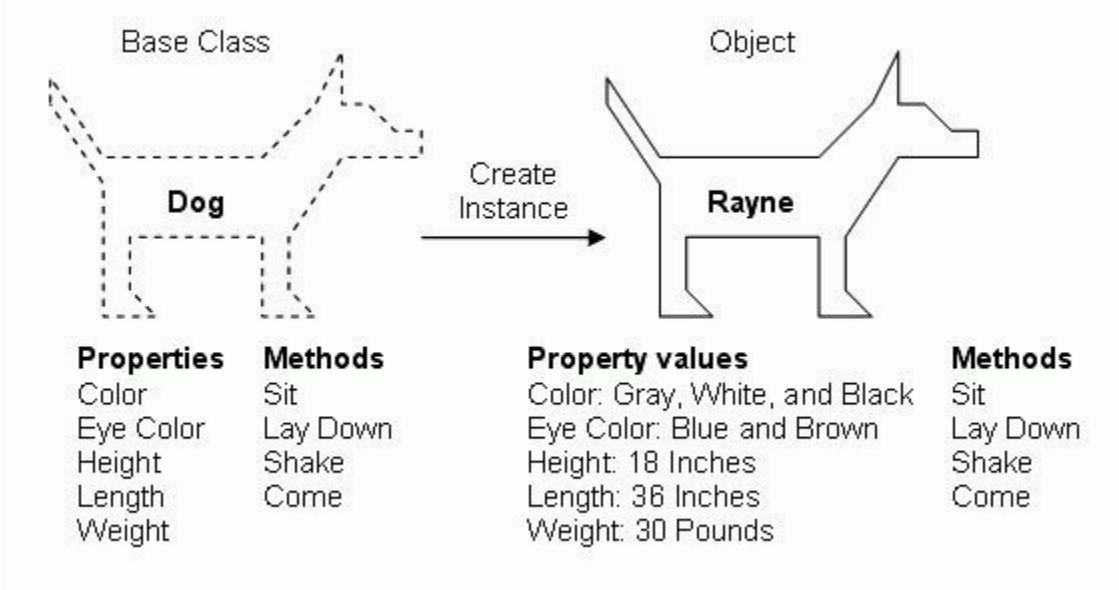
class heading

constructor heading

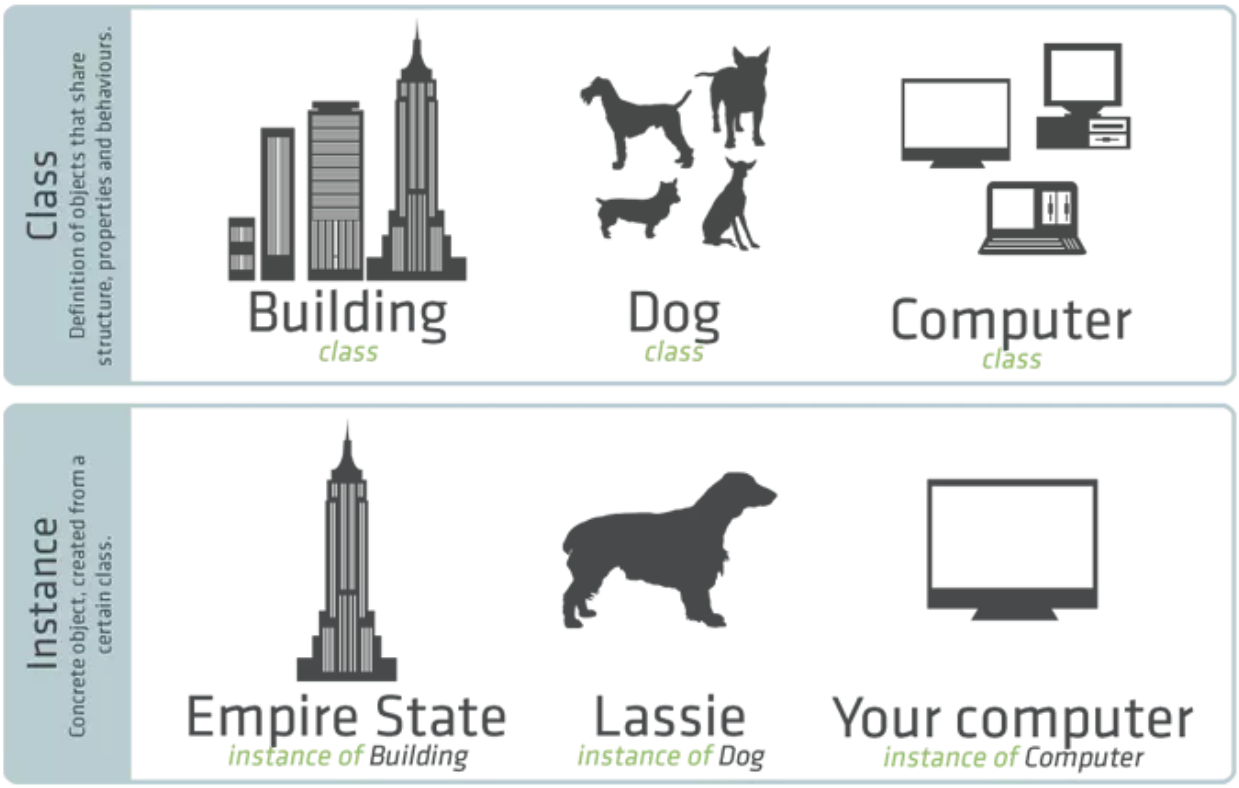
Use "`this.`" to define `x` and `y` properties for the class.

A method that returns a `Point` object's value.

Source:
https://learning.oreilly.com/library/view/web-programming-with/9781284091809/xhtml/29_Chapter11_03.xhtml



Source: [Classes and Objects in Javascript ES6 · Intermediate Programming @Woodbury]



Source: [Object-Oriented JavaScript]

Benefits of using classes

- Simpler and cleaner syntax for defining classes and creating instance objects
- Clearer and more intuitive inheritance and subclassing
- Easier to understand and maintain code
- Better support for encapsulation and data privacy

Differences between Classes and ES6 Prototype-based OOP

While the class syntax introduced in ES6 is a more modern and convenient way to define objects and classes, many existing codebases and libraries still use the older prototype-based approach.

https://codepen.io/shafferma08/pen/poxPMZd

Aspect	ES5 Prototype-based OOP	ES6 Classes
Syntax	Complex and error-prone	Simple and intuitive
Inheritance	Prototype chaining	Inheritance using the `extends` keyword
Subclassing	Complex and error-prone	Subclassing using the `extends` keyword
Code organization	Can be messy and hard to read	Organized and easy to read
Support for static methods	Supported	Simple and intuitive syntax for defining static methods
Support for getters and setters	Not directly supported, but can be implemented using the prototype pattern	Supported through the `get` and `set` keywords

Browser support	Widely supported in older browsers	Limited support in older browsers

Resources:

[JavaScript Classes]

[Classes - JavaScript | MDN]

Lesson 2

Defining classes and Creating instances

1. *Defining classes*: Classes are a way to define objects with shared properties and methods. In JavaScript, classes can be defined using the class keyword. A class can include properties and methods, which are defined inside the class's body. Properties can be assigned values in the class's constructor or directly inside the class body. Methods can be defined using regular function syntax, and they can access properties and other methods defined within the class.

2. *Creating instances*: Once a class is defined, you can create instances of it using the new keyword. This creates a new object that is an instance of the class. Instances have access to the properties and methods defined in the class, which they inherit.

3. *With a constructor*: A constructor is a special method that is called when a new instance of a class is created using the new keyword. Constructors are used to initialize the object's properties and set their initial values. In JavaScript, the constructor method is defined using the constructor keyword inside the class body.

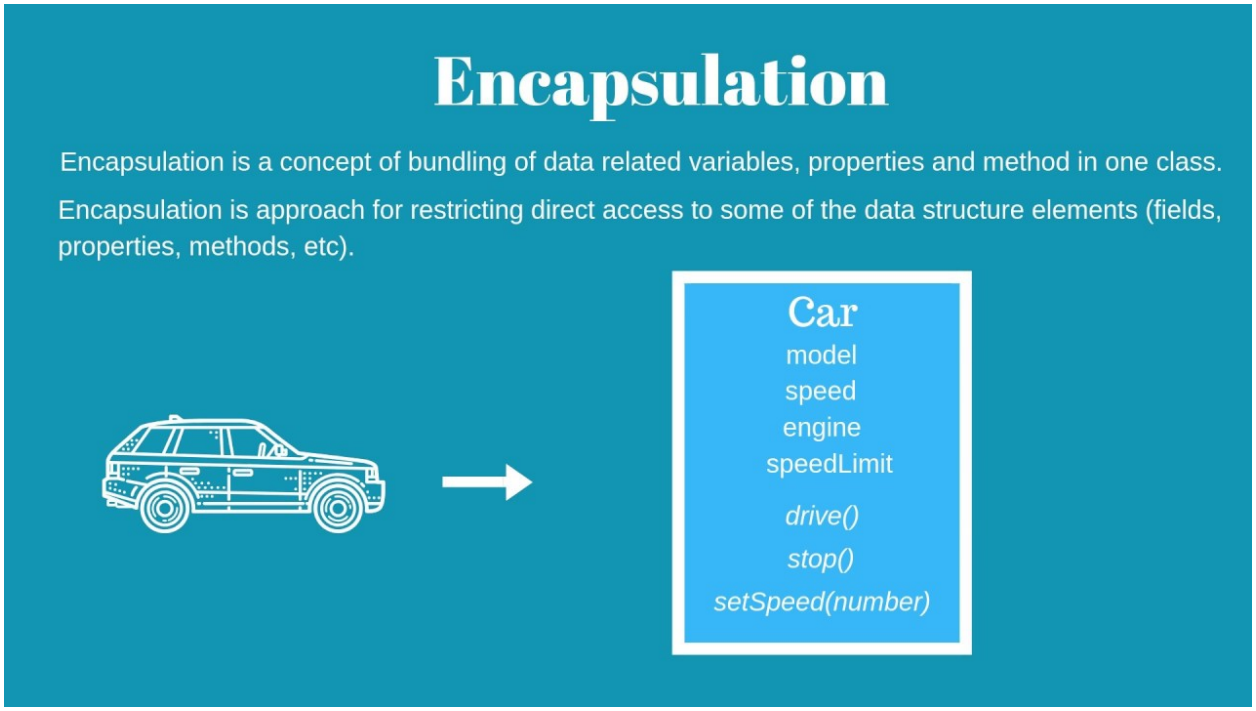
4. *Without a constructor*: In JavaScript, it's possible to define a class without a constructor. In this case, the class's properties are defined directly inside the class body, and their

values are set using literals or other expressions. When a new instance of the class is created using the new keyword, the properties are initialized to their default values.

Approach	Scenario	Description
Classes **with** Constructors	Initialization Requirements	Instances require unique initialization based on parameters (e.g., `BankAccount` with unique account number).
	Parameter Validation	Constructors can validate input parameters before object creation (e.g., validating day, month, and year for a `Date` class).
	Resource Allocation	Handle setup like allocating resources (e.g., network connections or file handles) during object creation.
Classes **without** Constructors	Optional Properties	Many properties are optional with sensible default values (e.g., `UserProfile` with optional `profilePicture` or `bio`).
	Simplified Object Creation	Quickly create instances without providing many details immediately (e.g., default position/velocity for a `Particle` class).
	Flexible Object Configuration Over Time	Properties filled out over time, not immediately (e.g., `Form` class populated as user interacts).
	Reducing Complexity for Rare Customization	Simplify code when objects rarely deviate from defaults (e.g., `DefaultSettings` class for software application).

Encapsulation

- Encapsulation is a fundamental concept in object-oriented programming (OOP) that refers to the practice of bundling data and behavior within an object, and controlling access to that data and behavior from outside the object.
- Encapsulation provides a way to enforce data privacy and security, and makes it easier to manage complex systems by organizing code into smaller, self-contained units.



Source: [Object-oriented programming in JavaScript #4. Encapsulation. | by Viktor Kukurba]

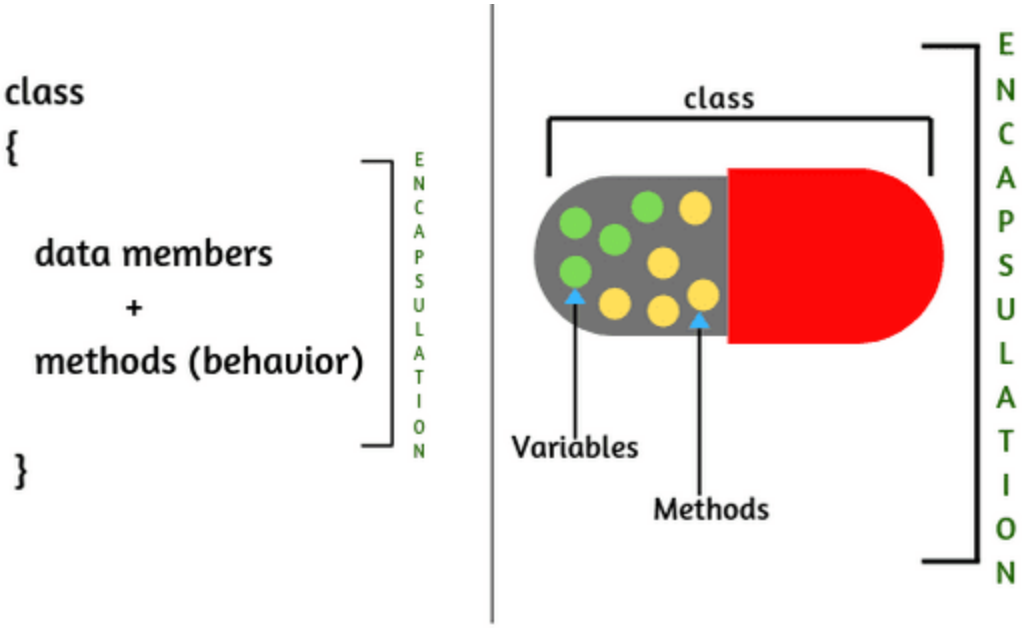
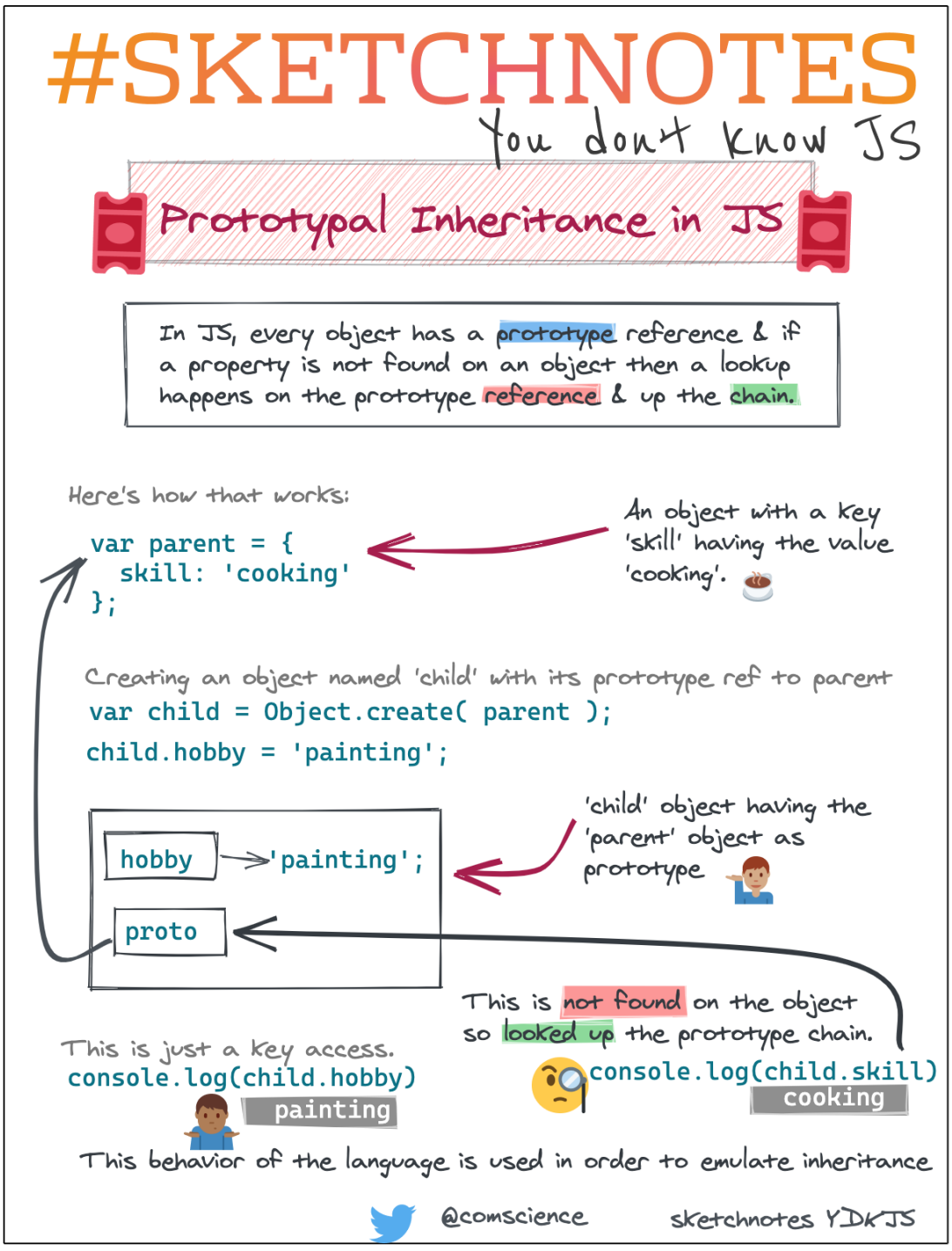


Fig: Encapsulation

Prototypal Inheritence



Source: [Kapeel Kokane - CodeSketched (@kokaneka) / X]

Resource: [Inheritance and the prototype chain - JavaScript | MDN]

Private properties and methods

https://codepen.io/shafferma08/pen/YzJQGao

Private properties and methods are those that are only accessible within the object itself, and not from outside.

Accessing instance properties and methods

1. *Accessing Instance Properties and Methods*: In JavaScript, an instance is created when you use a constructor function to create a new object. Instance properties and methods are unique to each instance of an object, and can be accessed using the dot notation. For example, if you have a constructor function called **Person**, and you create a new instance of **Person** called **john**, you can access the instance properties and methods using **john.propertyName** or **john.methodName()**.

2. *Inheritance*: Inheritance is a concept in object-oriented programming where a new class is created based on an existing class, inheriting all its properties and methods. In JavaScript, inheritance is achieved through the **prototype** property of a constructor function. When you create a new instance of the child class, it inherits all the properties and methods of the parent class, and can also have its own unique properties and methods

3. *Composition*: Composition is a way to combine multiple objects or classes to create a new object or class with the desired properties and behavior. It involves creating a new object or class that contains one or more instances of other objects or classes. Composition is often used as an alternative to inheritance when you need more flexibility in how you combine functionality.

3. *Polymorphism*: Polymorphism is the ability of an object to take on many forms. In JavaScript, polymorphism can be achieved through inheritance and method overriding. When a child class inherits a method from its parent class, it can override the method to provide a different implementation. This allows the child class to behave differently than the parent class, while still being able to use the same method name.

4. *Abstraction*: Abstraction is the process of hiding the implementation details of a function or object, while still providing an interface for the user to interact with. In JavaScript, abstraction can be achieved through the use of classes and objects. By defining a class with public and private methods and properties, you can provide a simplified interface for the user to interact with, while hiding the underlying implementation details. This helps to make your code more modular and easier to maintain.

Lesson 4

Calculator App

https://codepen.io/shafferma08/pen/xxyrqbY

AEL Practice Guide

1. Person Class

Steps:

1. Use the class keyword to define a new class called Person.
2. Inside the class, define a constructor that takes two parameters: name and age.
3. Define a method called greet() inside the class.
4. Inside the greet() method, use template literals to return a greeting.
5. Outside the class, create an instance of Person.
6. Call the greet() method on the instance.

2. Student Subclass

Steps:

1. Create a new class called Student that extends the Person class.
2. Inside the Student class, define a constructor that takes three parameters: name, age, and major.
3. Call the super() function in the constructor to pass the name and age parameters to the parent class.
4. Define a method called study() inside the Student class.
5. Outside the class, create an instance of Student.
6. Call the greet() and study() methods on the instance.

3. Animal Abstract Class

Steps:

1. Create an abstract class called Animal.
2. Inside the class, define a constructor that takes a name parameter.
3. Define an abstract method called speak().
4. Create two subclasses: Dog and Cat.
5. Implement the speak() method in both subclasses.
6. Outside the classes, create instances of Dog and Cat.
7. Call the speak() method on both instances.