JavaScript OOP Cheat Sheet

Object-Oriented Programming (OOP) - is a paradigm that organizes software design around data, or objects, rather than functions and logic.

Class - A blueprint for creating objects. It defines all the properties (attributes) and methods (behavior) of a class. It doesn't hold data, it's just a holder of how the object going to look like.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  getFriends() {

    // Instance Methods
  }
}
```

Properties (attributes) *is a variable that holds data for an object. It stores the state of an object.*

Instance Variable - variables that belong to an instance of a class. Each instance of the class has its own copy of these variables. They represent the state or data of the object.

Class Variables - variables that belong to the class itself rather than to any particular instance. They are shared among all instances of the class.

```
static numberOfChildren = 5; // Class variable (static variable)
  constructor(name, age) {
    this.name = name; // Instance variable
    this.age = age; // Instance variable
  }
```

Methods *is a function defined within a class that operates on instances of that class. A method is an action that an object is able to perform. (operations/behaviors)*

```
eat(){}
walk(){}
introduce() {
    return `Hi! my name is ${this.name} I'm ${this.age} years old`;
}
```

Instance Methods *are methods that operate on an instance of the class. They can access and modify the instance's properties.*

```
eat() {
   // Instance method
}
walk() {
   // Instance method
}
```

Static Methods *do not operate on an instance of the class and do not have access to instance properties. They are bound to the class itself.*

```
class MathUtils {
  static add(a, b) {
    return a + b; // Static method
  }
}
```

Class Methods *operate on the class itself rather than on instances of the class. They have access to the class properties and can modify the class state.*

```
class Car {
  static numberOfWheels = 4;

  static getNumberOfWheels() {
    return Car.numberOfWheels; // Class method
  }
}
```

Instantiate *is the process of creating an object from a class.*

```
let person1 = new Person();
let person2 = new Person();
```

New keyword is how we take a class and turn it into objects. Basically, is used to create a new instance of a class.

Object - An instance of a class. It contains real values instead of variables.

- A thing from the real world
- A thing that you want to store and process
- Another name is *Entity*
- Each object is an instance of a class

```
// Creating an instance (object) of the Car class
let person1 = new Person("Vlahd", 21);

// Accessing properties and methods of the object
console.log(person1.introduce());
```

Constructor - Special method invoked at the time of object creation (it is called whenever a class is instantiated). It usually initializes the object's properties.

```
constructor(name, age) {
    this.name = name;
    this.age = age;
}
```

ABSTRACTION means to simplify reality

- Being concerned only to the data relevant to class example Person and the task we want to perform with the data.
- It involves hiding complex implementation details and showing only the necessary features of an object.

```
class Rectangle extends Shape {
  constructor(width, height) {
    super();
    this.width = width;
    this.height = height;
  }

  calculateArea() {
    return this.width * this.height;
  }
}

const circle = new Circle(7);
console.log(`Circle Area: ${circle.calculateArea()}`);
```

ENCAPSULATION

- Hiding data and complexity
- information hiding
- It restricts direct access to some of an object's components, which can prevent the accidental modification of data.

```
class Person {
  constructor(name, age) {
    this._name = name; // Private variable convention
    this._age = age; // Private variable convention
  }

  // Getter for name
  getName() {
    return this._name;
  }

  // Setter for name
  setName(name) {
    this._name = name;
  }

  // Getter for age
  getAge() {
    return this._age;
  }

  // Setter for age
  setAge(age) {
    if (age > 0) {
      this._age = age;
    } else {
      console.log("Age must be a positive number");
    }
  }

  introduce() {
    return `Hi! My name is ${this._name} and I'm ${this._age} years old`;
  }
}

const person = new Person('Alice', 30);
console.log(person.introduce()); // Hi! My name is Alice and I'm 30 years old
```

Naming Convention: Using an underscore (_) before the variable name to indicate that it is intended to be private. This relies on developers respecting the convention.

Private Fields with #: Introduced in ECMAScript 2019, this syntax allows true private fields within classes, providing encapsulation at the language level.

Access Modifiers

1. **Public** properties and methods are accessible from anywhere, both inside and outside the class. By default, all properties and methods in JavaScript classes are public.
2. **Private** properties and methods are accessible only within the class where they are defined. Private fields are declared using the # syntax introduced in ECMAScript 2019 (ES10).
3. JavaScript does not have a native protected keyword like some other programming languages. However, developers often simulate **protected** members by using naming conventions, indicating that such properties or methods should be treated as protected (***i.e., accessible within the class and subclasses***).

INHERITANCE

- a class can derive its methods and properties from another class
- can result to hierarchy of classes
- inheritance defines ***type of*** relationships

Base class *is the start or root of the inheritance hierarchy*

Sub class *– derives from another class*

Super class – class that derives from

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a noise.`);
  }
}

class Dog extends Animal {
  constructor(name) {
    super(name);
  }
```

```
  speak() {
    console.log(`${this.name} barks.`);
  }
}

class Cat extends Animal {
  constructor(name) {
    super(name);
  }

  speak() {
    console.log(`${this.name} meows.`);
  }
}

const dog = new Dog('Rex');
dog.speak(); // Rex barks.

const cat = new Cat('Whiskers');
cat.speak(); // Whiskers meows.
```

POLYMORPHISM

- a class can implement an inherited method in its own way
- subclass can override methods from base class or super class with its new version of its own
- the same interface but behave in different ways

```
class Animal {
  speak() {
    console.log("Animal speaking");
  }
}

class Dog extends Animal {
  speak() {
    console.log("Woof!");
  }
}

class Cat extends Animal {
  speak() {
```

```
    console.log("Meow!");
  }
}

let animals = [new Dog(), new Cat()];
animals.forEach((animal) => animal.speak());
// Output:
// Woof!
// Meow!
```

`**static**` keyword is used to define methods and properties that belong to the class itself rather than to instances of the class. This means that static methods and properties are shared among all instances of the class and can be accessed without creating an instance.