The background features a collection of various 3D geometric shapes in a light sage green color. These include spheres, cylinders, cones, a rectangular prism, a torus (donut shape), a wavy line, and several flat circular discs. The shapes are scattered across the frame, some overlapping, creating a modern and abstract aesthetic.

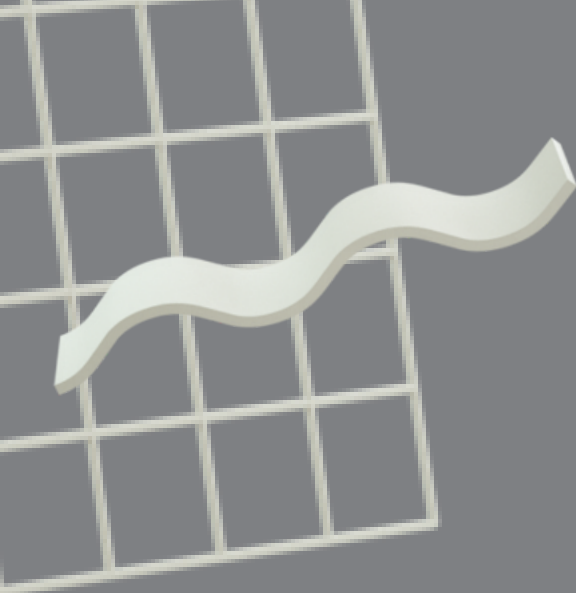
# Object Oriented JavaScript

Rohan Chhetry

# Class and Objects

- A class is a blueprint for creating objects with shared properties and methods.
- Objects are instances of classes.





# Question 1

Create a Person class with name and age properties, and a method greet that prints a greeting.




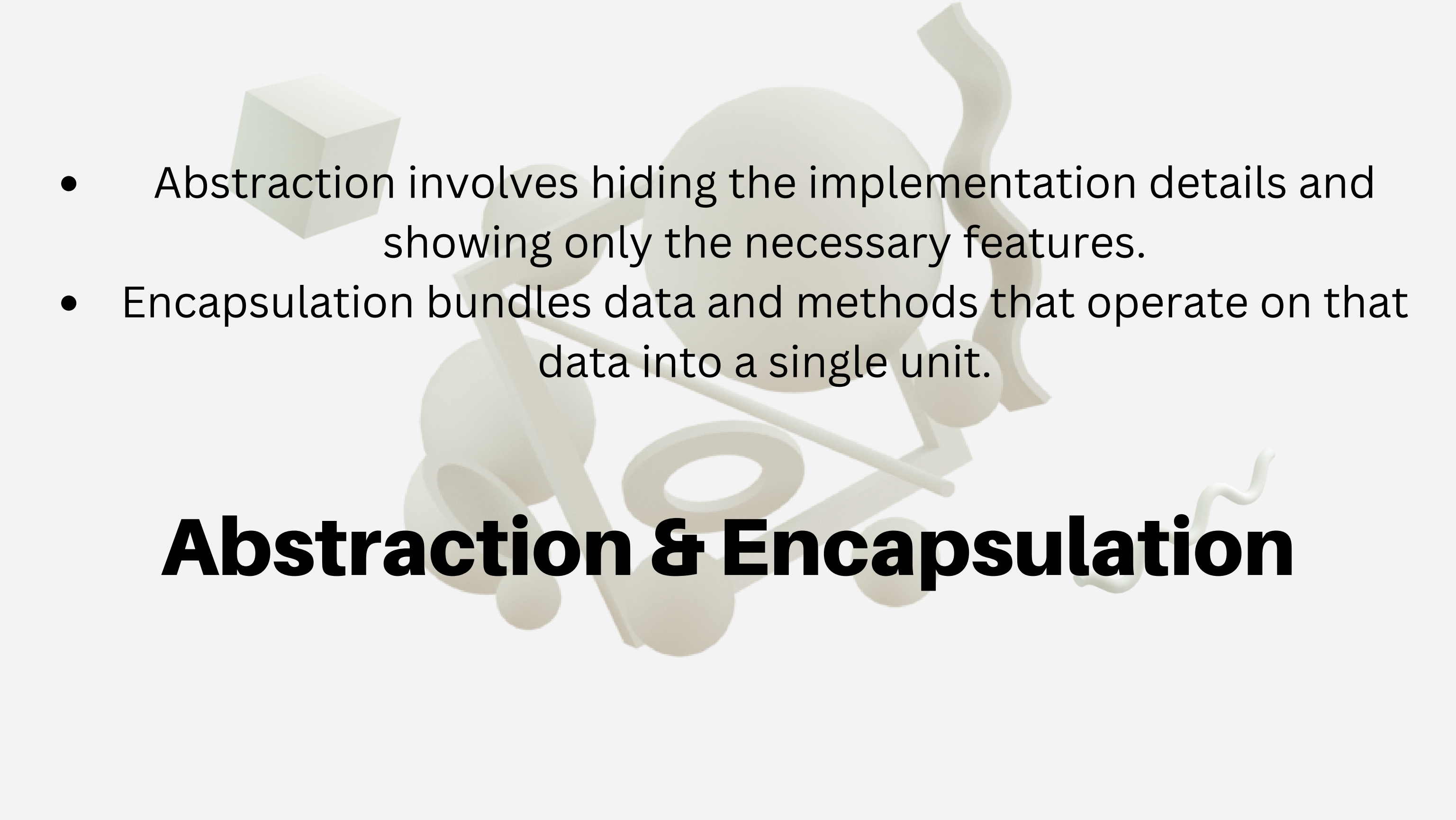


# Solution 1


```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  greet() {  
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);  
  }  
}
```

```
const person1 = new Person('John', 25);  
person1.greet(); // Output: Hello, my name is John and I am 25 years old.
```



- 
- Abstraction involves hiding the implementation details and showing only the necessary features.
  - Encapsulation bundles data and methods that operate on that data into a single unit.

# **Abstraction & Encapsulation**



# Problem 2

**Create a Car class with private properties (make and model) and a method to get the car details**




# Solution 2

```
class Car {  
  #make;  
  #model;
```

```
  constructor(make, model) {  
    this.#make = make;  
    this.#model = model;  
  }
```

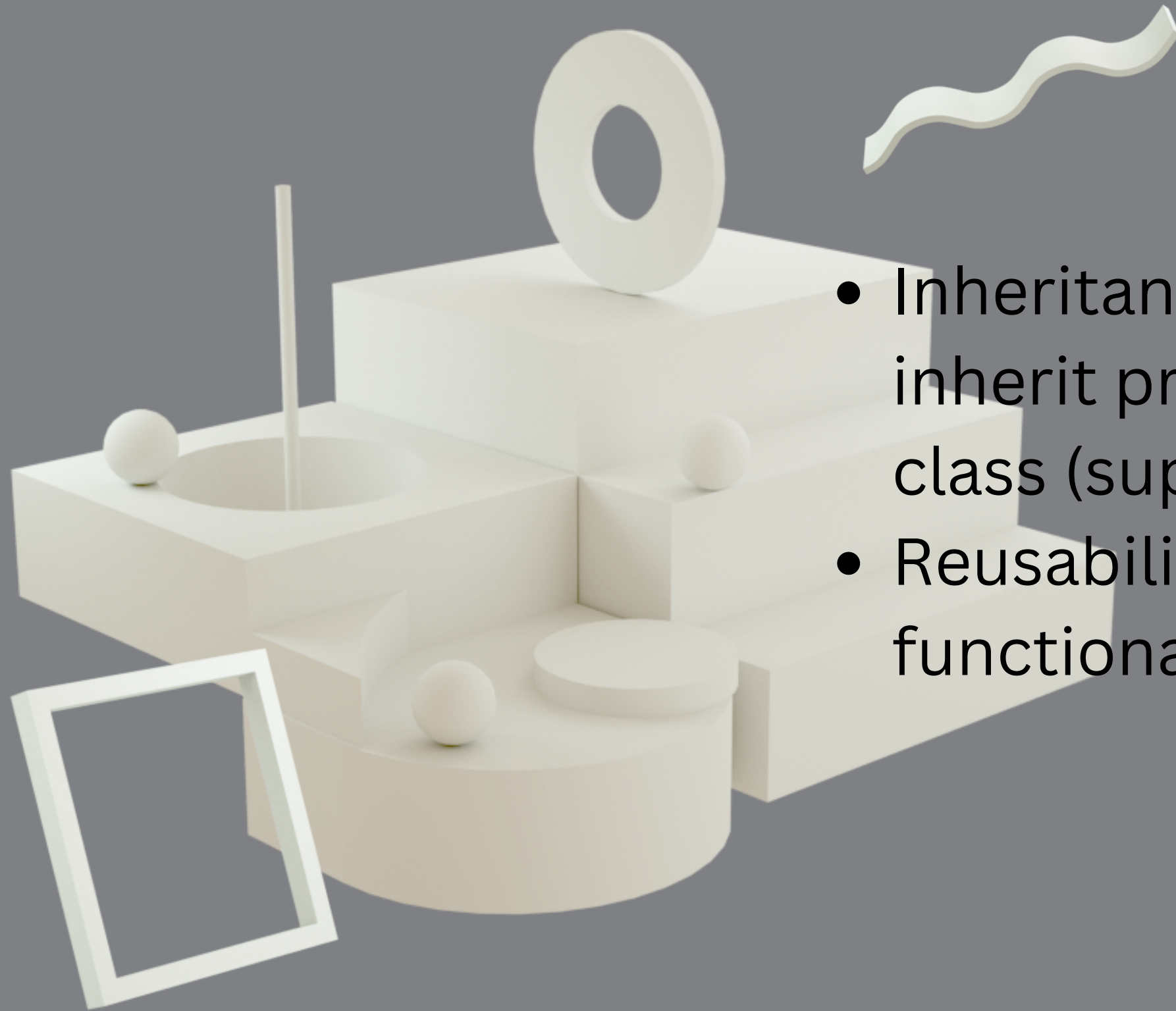
```
  getDetails() {  
    return `${this.#make} ${this.#model}`;  
  }  
}
```

```
const myCar = new Car('Toyota', 'Camry');  
console.log(myCar.getDetails()); // Output: Toyota Camry
```





# Reusability & Inheritance



- Inheritance allows a class (subclass/child) to inherit properties and methods from another class (superclass/parent).
- Reusability is achieved by using common functionalities from a parent class.



# Problem 3

Create a Student class that inherits from the Person class and adds a grade property.



# Solution 3

```
class Student extends Person {  
  constructor(name, age, grade) {  
    super(name, age);  
    this.grade = grade;  
  }  
}
```

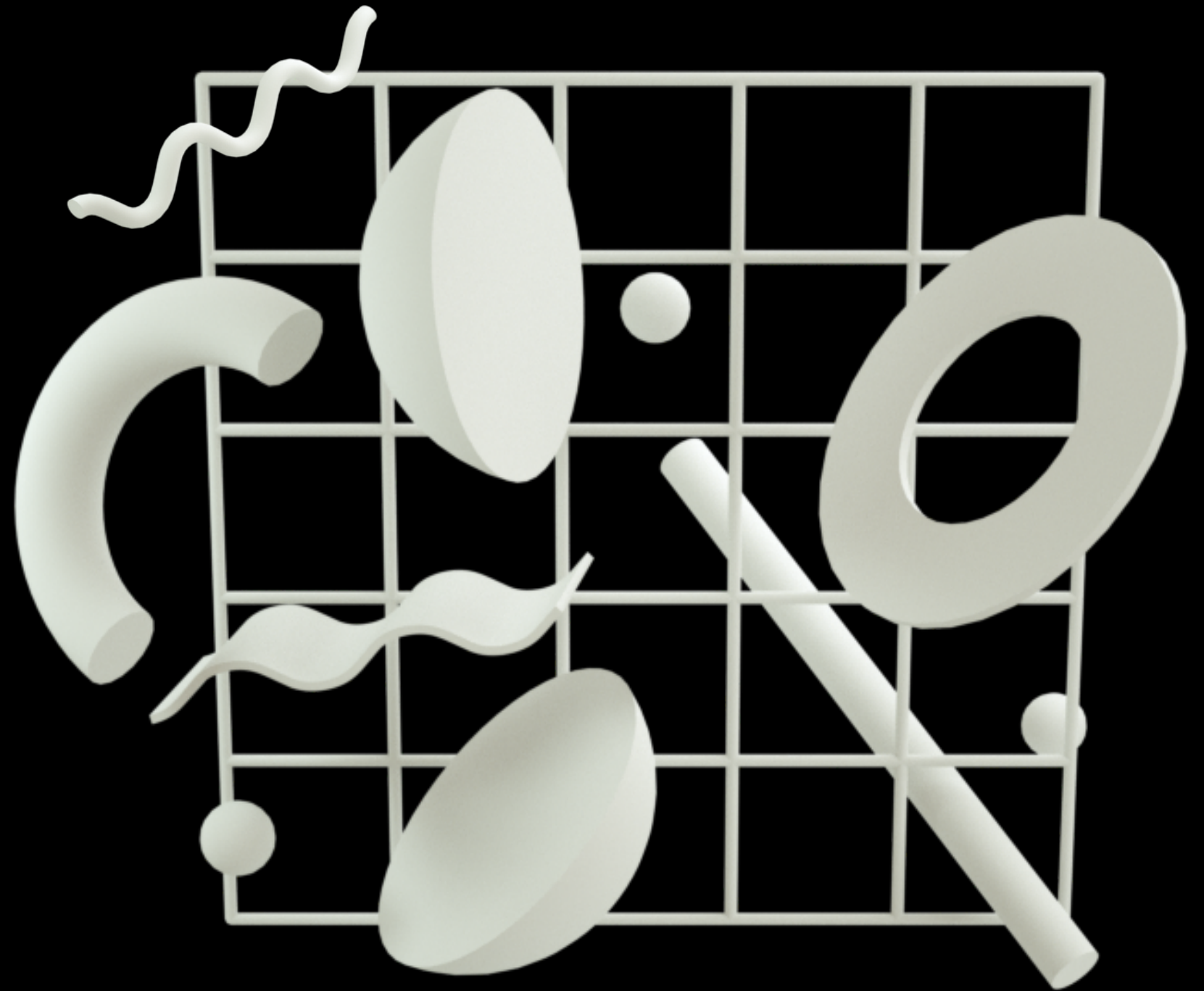
```
  displayInfo() {  
    console.log(` ${this.name} is a student with grade ${this.grade}.`);  
  }  
}
```

```
const student1 = new Student('Alice', 20, 'A');  
student1.displayInfo(); // Output: Alice is a student with grade A.
```



# Polymorphism

- Polymorphism allows objects to be treated as instances of their parent class, providing flexibility in code.
- Method overriding is a form of polymorphism where a subclass provides a specific implementation of a method defined in its superclass.






# Problem 4

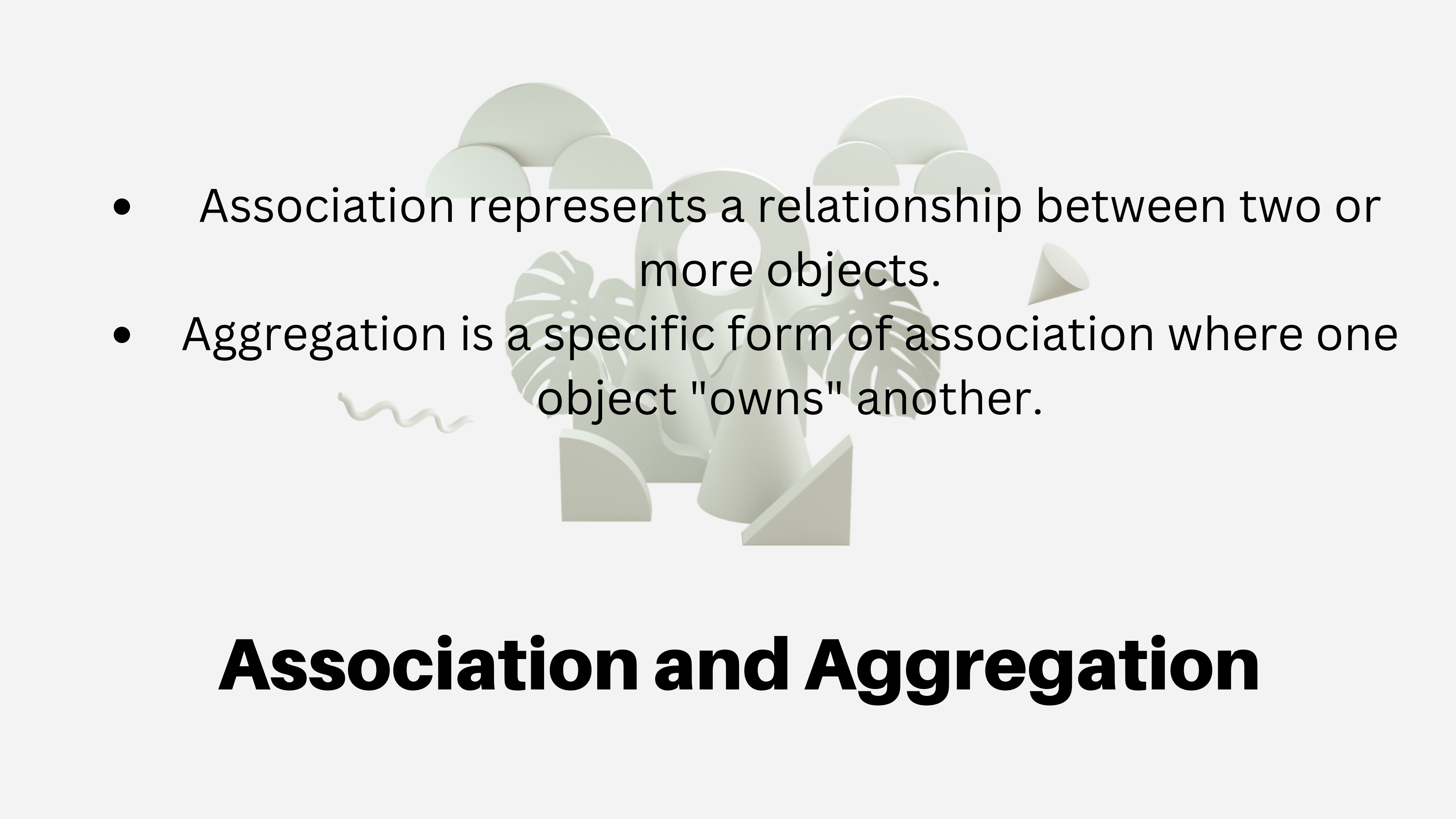
Override the greet method in the Student class to include the grade.

# Solution 4

An abstract geometric composition on the left side of the slide. It features a thin white line with three small white spheres attached to it. Various 3D shapes are scattered around: a wavy line, a small sphere, a large oval, a ring, a cone, a smaller cone, and a wavy line. The shapes are rendered in a light beige color with soft shadows.

```
class Student extends Person {  
  constructor(name, age, grade) {  
    super(name, age);  
    this.grade = grade;  
  }  
  
  greet() {  
    console.log(`Hello, my name is ${this.name}, I am ${this.age} years  
old, and my grade is ${this.grade}.`);  
  }  
}  
  
const student1 = new Student('Bob', 22, 'B');  
student1.greet(); // Output: Hello, my name is Bob, I am 22 years old,  
and my grade is B.
```



- 
- The background features a collection of light green, semi-transparent geometric shapes including circles, triangles, and a wavy line, along with stylized foliage like a monstera leaf and a palm frond.
- Association represents a relationship between two or more objects.
  - Aggregation is a specific form of association where one object "owns" another.

# **Association and Aggregation**



# Problem 5

Create a Library class that has an association with multiple Book objects.



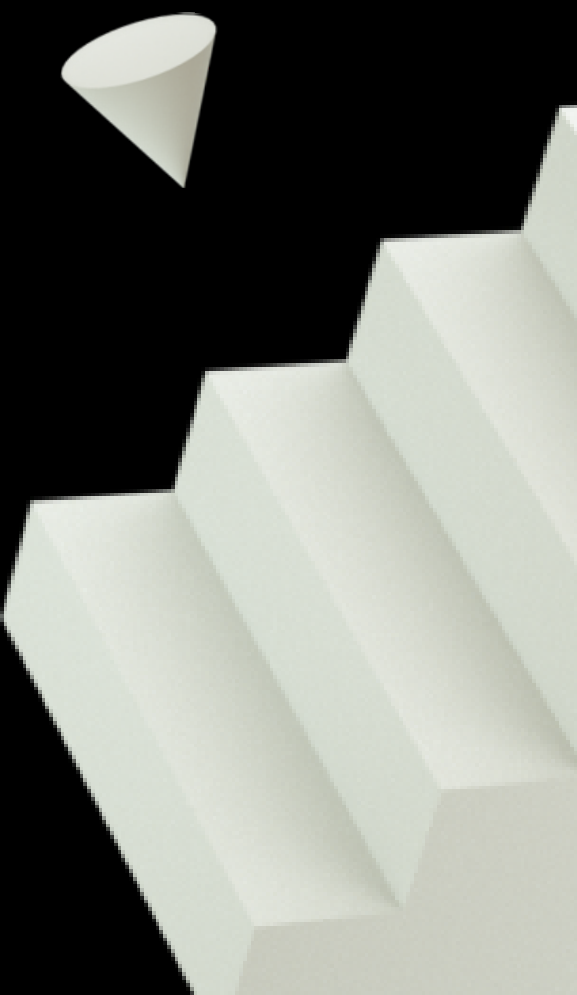


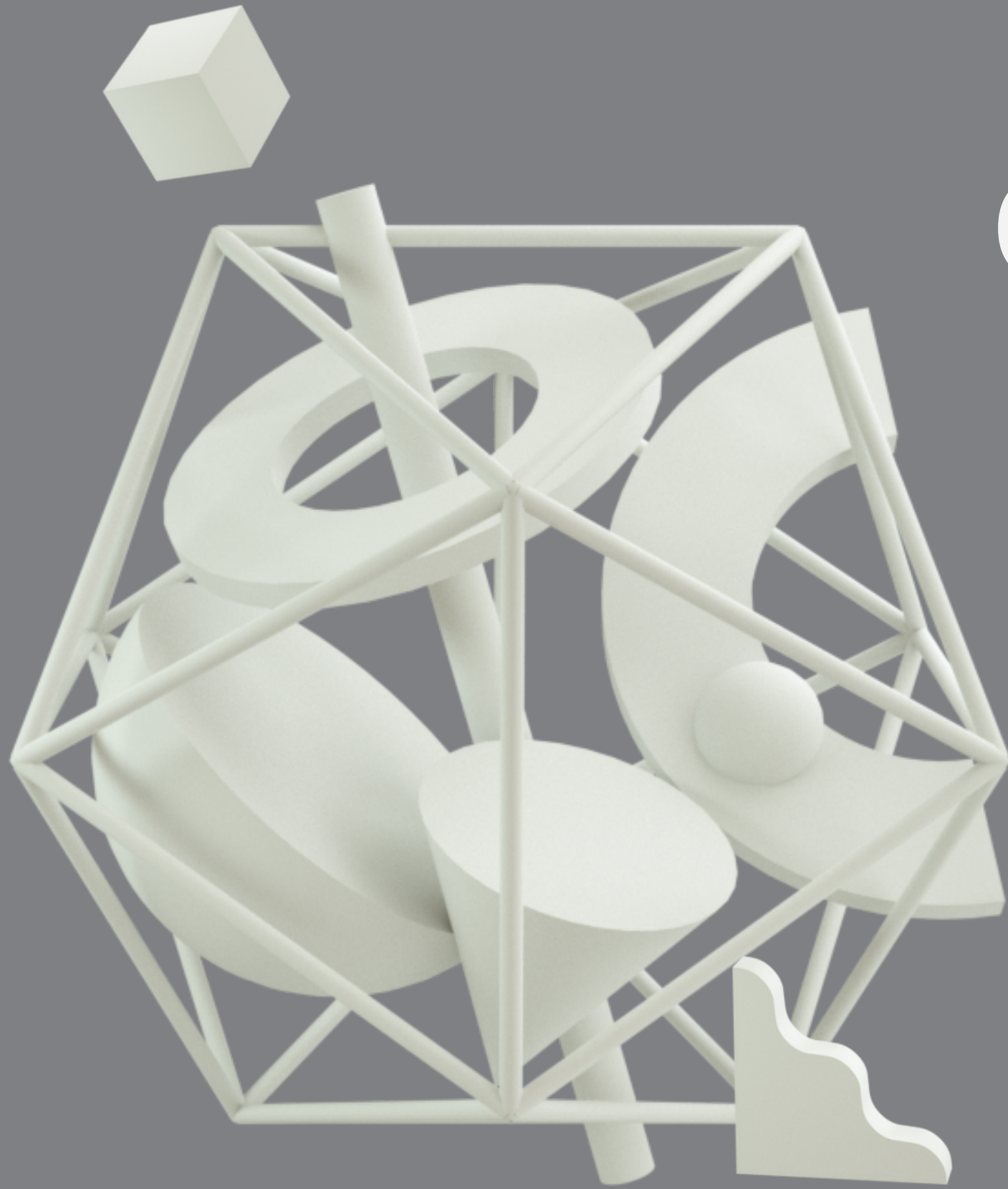


# Solution 5

```
class Book {  
  constructor(title) {  
    this.title = title;  
  }  
}  
  
class Library {  
  constructor() {  
    this.books = [];  
  }  
  
  addBook(book) {  
    this.books.push(book);  
  }  
  
  displayBooks() {  
    console.log('Library Books:');
```

```
    this.books.forEach(book =>  
      console.log(book.title));  
  }  
}  
  
const library = new Library();  
const book1 = new Book('JavaScript:  
The Good Parts');  
const book2 = new Book('Clean Code');  
library.addBook(book1);  
library.addBook(book2);  
library.displayBooks();  
// Output:  
// Library Books:  
// JavaScript: The Good Parts  
// Clean Code
```





# Composition

- Composition is a design concept where a class contains objects of other classes.
- It enables creating complex objects by combining simpler ones.

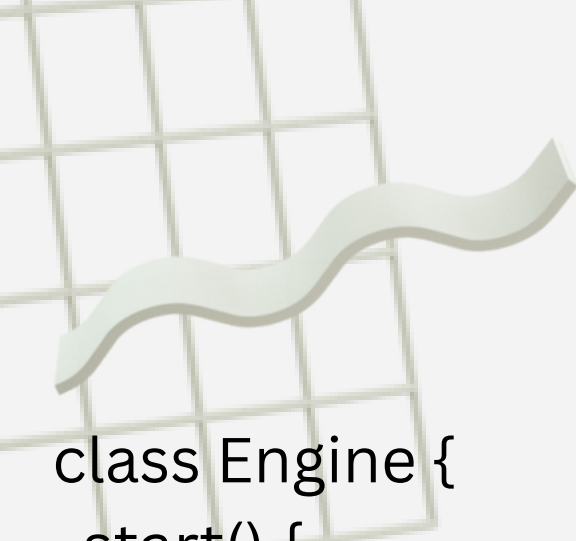


# Problem 6

**Create a Car class composed of an Engine class and a Wheel class.**



# Solution 6



```
class Engine {  
  start() {  
    console.log('Engine started.');  }  
}
```

```
class Wheel {  
  rotate() {  
    console.log('Wheel rotating.');  }  
}
```

```
class Car {  
  constructor() {  
    this.engine = new Engine();  
    this.wheels = [new Wheel(), new Wheel(), new Wheel(), new Wheel()];  
  }  
}
```

```
  drive() {  
    this.engine.start();  
    this.wheels.forEach(wheel => wheel.rotate());  
    console.log('Car is moving.');  }  
}
```

```
const myCar = new Car();  
myCar.drive();  
// Output:  
// Engine started.  
// Wheel rotating.  
// Wheel rotating.  
// Wheel rotating.  
// Wheel rotating.  
// Car is moving.
```

