Interface Segregation Principle

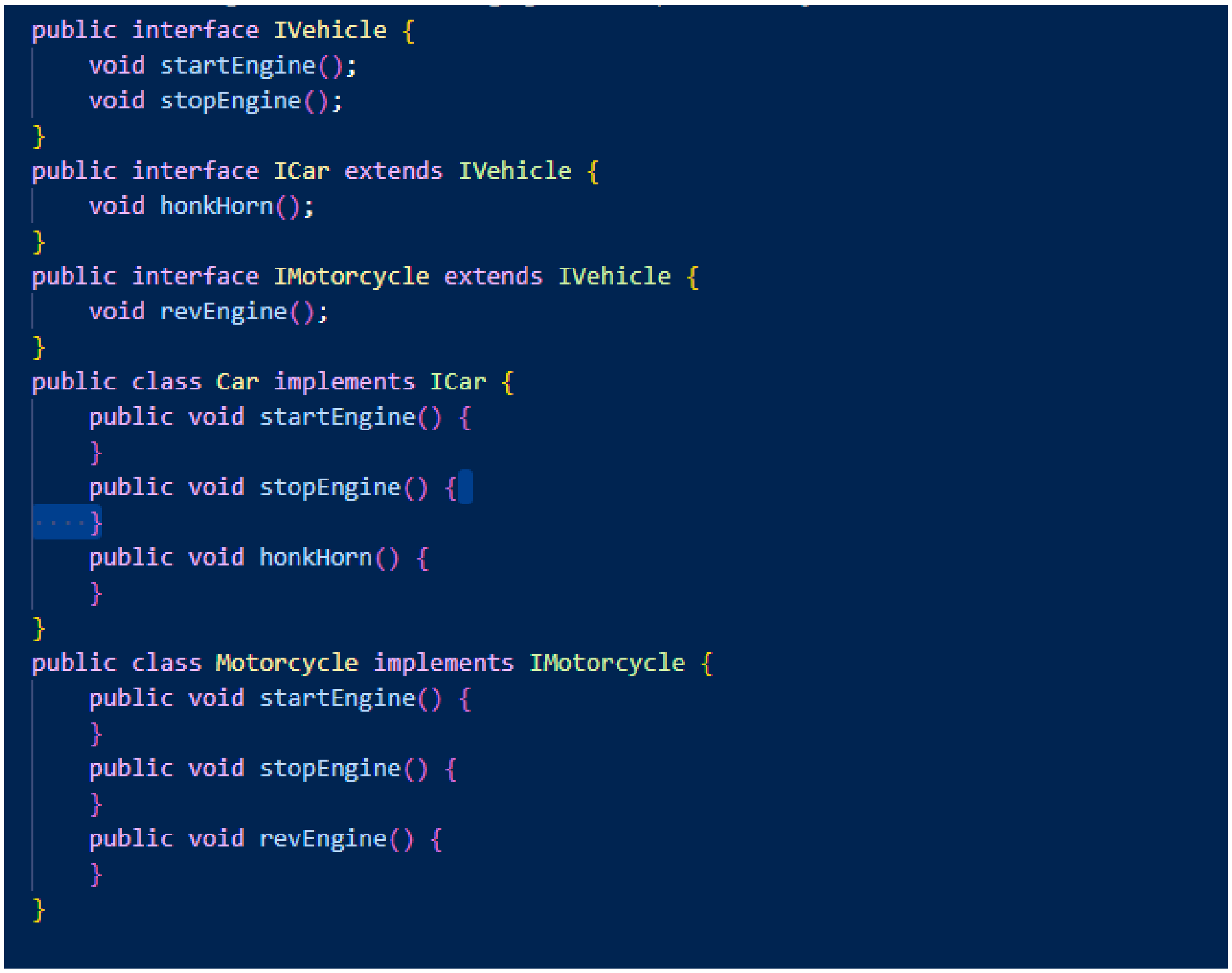
The interface segregation principle is a principle of software engineering that states that clients should not be forced to depend on methods they do not use. This principle is intended to promote the design of modular and reusable software systems by ensuring that the interfaces of a system are fine-grained and client-specific. In other words, the interface segregation principle suggests that it is better to have many small and specific interfaces, rather than a few large and general ones. This allows clients to implement only the features they need, without being forced to implement unnecessary ones. This can help to reduce the complexity of a system and make it more maintainable and extensible over time.

Example:

For example, consider a class that represents a vehicle. This class might have methods for starting the engine, stopping the engine, and honking the horn. However, not all vehicles have horns, so it would be better to create separate interfaces for different types of vehicles (e.g. a car interface and a motorcycle interface), each with their own set of methods. This way, clients that only need to start and stop the engine (e.g. a car rental company) can implement the car interface without having to worry about the unnecessary horn method.

Here is an example implementation of the vehicle class using the Interface Segregation Principle in Java:

In this example, the IVehicle interface defines the basic functionality of a vehicle, while the ICar and IMotorcycle interfaces extend this functionality with methods specific to cars and motorcycles respectively. This allows clients to implement only the functionality they need, without being forced to implement unnecessary methods.



```
public interface IVehicle {
    void startEngine();
    void stopEngine();
}
public interface ICar extends IVehicle {
    void honkHorn();
}
public interface IMotorcycle extends IVehicle {
    void revEngine();
}
public class Car implements ICar {
    public void startEngine() {
    }
    public void stopEngine() {
    }
    public void honkHorn() {
    }
}
public class Motorcycle implements IMotorcycle {
    public void startEngine() {
    }
    public void stopEngine() {
    }
    public void revEngine() {
    }
}
```