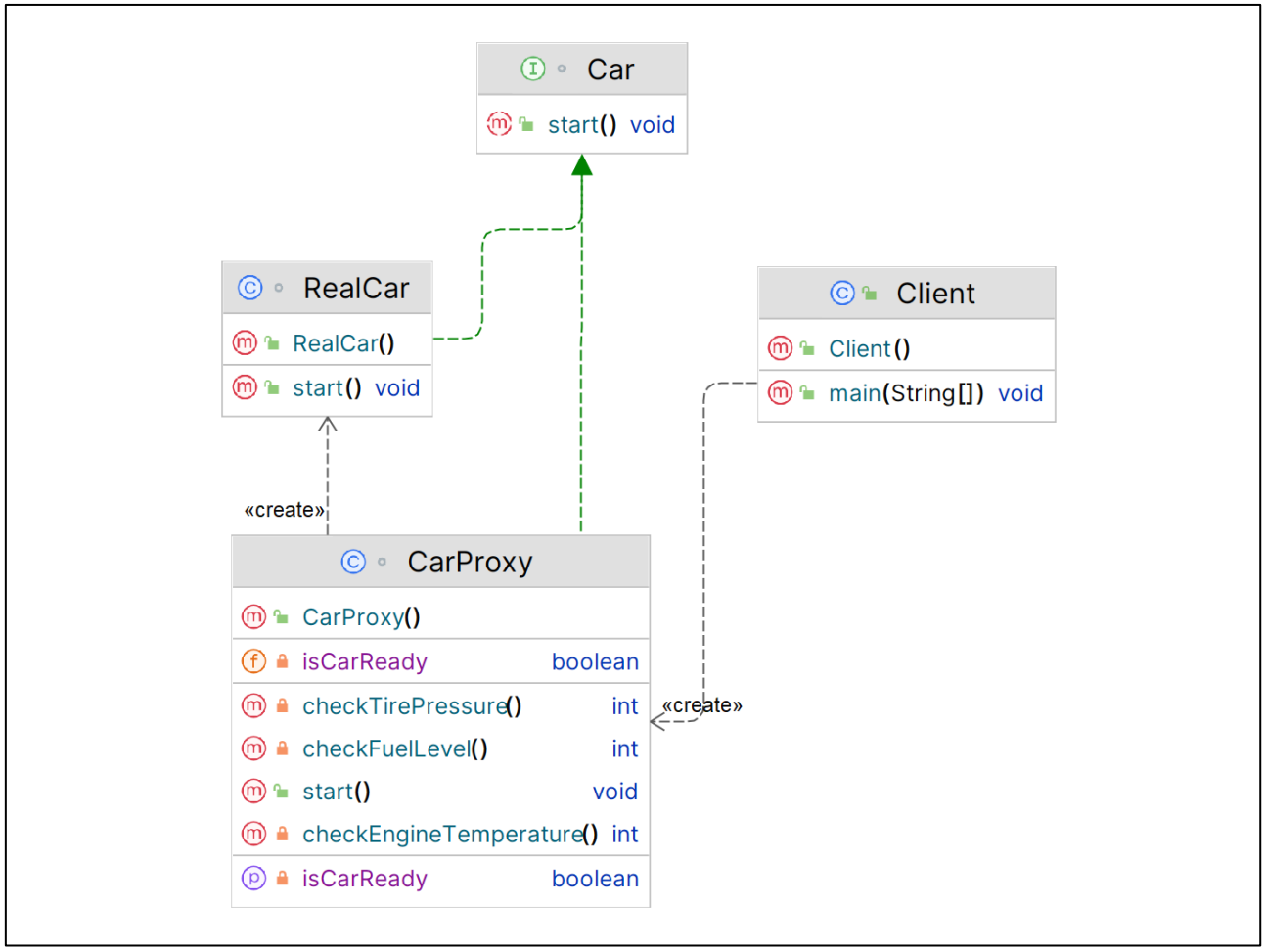
Assignment 11: Proxy Design Pattern

What is Proxy Design Pattern?

Proxy is a structural design pattern that lets you provide a **substitute** or **placeholder** for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

Structure (Class Diagram)



Implementation (Code)

```
// Subject interface
interface Car {
    void start();
}

// Real subject
class RealCar implements Car {
    public void start() {
        System.out.println("Starting the car");
    }
}
```

```
// Proxy
class CarProxy implements Car {
    private RealCar realCar;
    private boolean isCarReady;
    private int fuelLevel;
    private int tirePressure;
    private int engineTemperature;

    public void start() {
        if (realCar == null) {
            realCar = new RealCar();
        }
        if (isCarReady()) {
            realCar.start();
        }
    }

    private boolean isCarReady() {
        if (!isCarReady) {
            System.out.println("Checking the car's status...");
            fuelLevel = checkFuelLevel();
            tirePressure = checkTirePressure();
            engineTemperature = checkEngineTemperature();
            isCarReady = fuelLevel > 0 && tirePressure > 0 && engineTemperature < 100;
        }
        return isCarReady;
    }

    private int checkFuelLevel() {
        // Perform checks to determine the fuel level
        int fuelLevel = 50; // Set to 50 for demonstration purposes only
        System.out.println("Fuel level: " + fuelLevel);
        return fuelLevel;
    }

    private int checkTirePressure() {
        // Perform checks to determine the tire pressure
        int tirePressure = 30; // Set to 30 for demonstration purposes only
        System.out.println("Tire pressure: " + tirePressure);
        return tirePressure;
    }

    private int checkEngineTemperature() {
        // Perform checks to determine the engine temperature
        int engineTemperature = 90; // Set to 90 for demonstration purposes only
        System.out.println("Engine temperature: " + engineTemperature);
        return engineTemperature;
    }
}
```

```
// Client code
public class Client {
    public static void main(String[] args) {
        Car car = new CarProxy();
        car.start();
    }
}
```

Output:

```
Checking the car's status...
Fuel level: 50
Tire pressure: 30
Engine temperature: 90
Starting the car
```

Applicability

1. **Lazy initialization (virtual proxy):** This is when you have a heavyweight service object that wastes system resources by being always up, even though you only need it from time to time.
2. **Access control (protection proxy):** This is when you want only specific clients to be able to use the service object; for instance, when your objects are crucial parts of an operating system and clients are various launched applications (including malicious ones).
3. **Local execution of a remote service (remote proxy):** This is when the service object is located on a remote server.
4. **Logging requests (logging proxy):** This is when you want to keep a history of requests to the service object.
5. **Caching request results (caching proxy):** This is when you need to cache results of client requests and manage the life cycle of this cache, especially if results are quite large.
6. **Smart reference:** This is when you need to be able to dismiss a heavyweight object once there are no clients that use it.