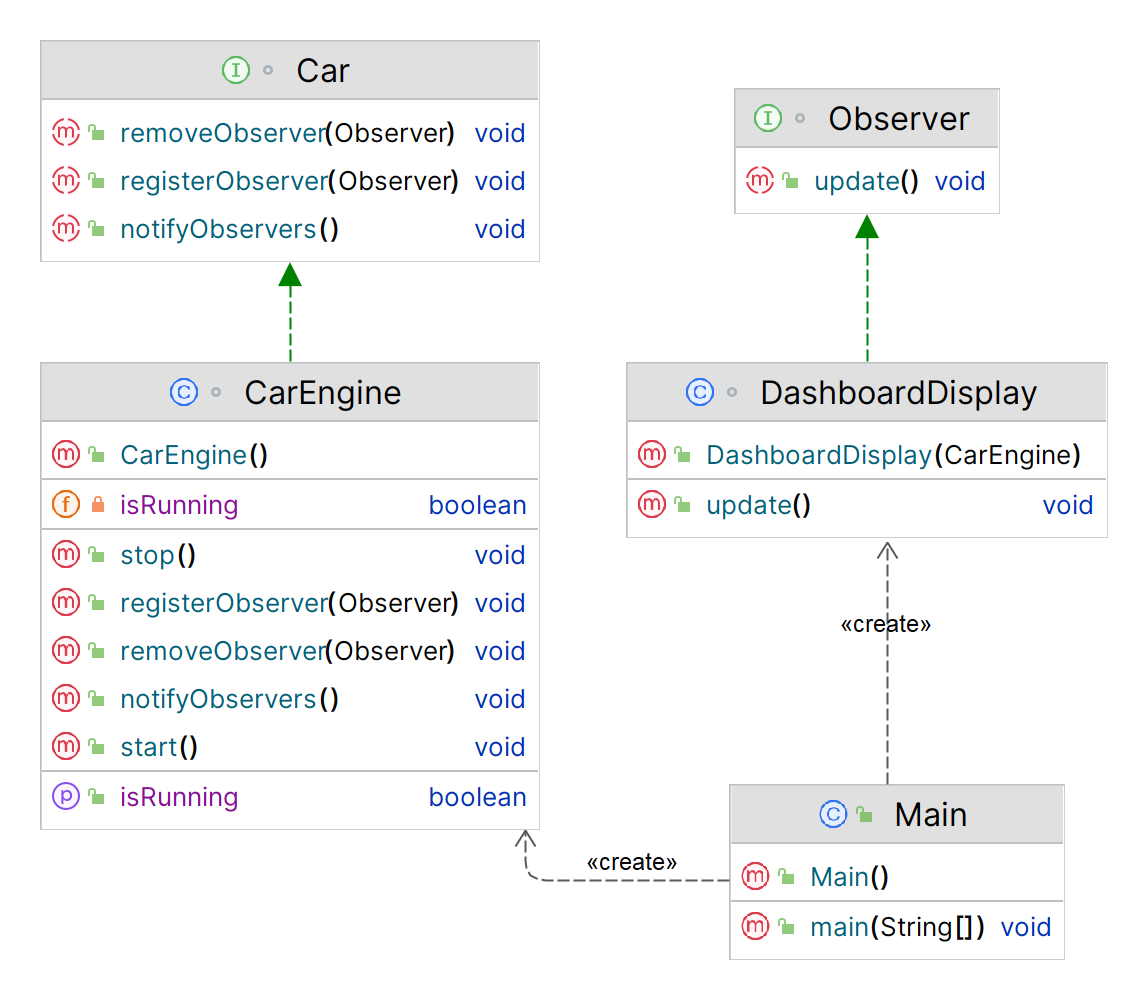
**Assignment 15: Observer Design Pattern**

**What is Observer Design Pattern?**

**Observer** is a **behavioural** design pattern that lets you define a **subscription mechanism** to **notify** multiple objects about any events that happen to the object they’re **observing**.

**Structure (Class Diagram)**



**Implementation (Code)**

|  |
| --- |
| import java.util.ArrayList; import java.util.List;  ***// Define the Subject interface***interface Car {  void registerObserver(Observer observer);  void removeObserver(Observer observer);  void notifyObservers(); }  ***// Define the Observer interface***interface Observer {  void update(); }  ***// Define the concrete Subject class***class CarEngine implements Car {  private List<Observer> observers = new ArrayList<>();  private boolean isRunning = false;   public void registerObserver(Observer observer) {  observers.add(observer);  }   public void removeObserver(Observer observer) {  observers.remove(observer);  }  public void notifyObservers() {  for (Observer observer : observers) {  observer.update();  }  }   ***// This method is called to start the car engine***public void start() {  System.*out*.println("Starting the car engine.");  isRunning = true;  notifyObservers();  }  ***// This method is called to stop the car engine***public void stop() {  System.*out*.println("Stopping the car engine.");  isRunning = false;  notifyObservers();  }  ***// This method is called to check if the car engine is running***public boolean isRunning() {  return isRunning;  } }  ***// Define the concrete Observer class***class DashboardDisplay implements Observer {  private CarEngine carEngine;   public DashboardDisplay(CarEngine carEngine) {  this.carEngine = carEngine;  }  public void update() {  if (carEngine.isRunning()) {  System.*out*.println("Displaying current speed.");  } else {  System.*out*.println("Displaying engine warning light.");  }  } }  ***// Usage example***public class Main {  public static void main(String[] args) {  *// Create the subject (car engine)* CarEngine carEngine = new CarEngine();   *// Create some observers (dashboard displays)* DashboardDisplay display1 = new DashboardDisplay(carEngine);  DashboardDisplay display2 = new DashboardDisplay(carEngine);   *// Register the observers with the subject* carEngine.registerObserver(display1);  carEngine.registerObserver(display2);   *// Start the car engine* carEngine.start();   *// Stop the car engine* carEngine.stop();} }  **Output** |
|  |

**Applicability**

1. Use the **Observer** pattern when changes to the state of one object may **require changing other objects**, and the actual set of objects is unknown beforehand or **changes dynamically**.
2. Use the pattern when some objects in your app must **observe others**, but only for a **limited time** or in specific cases.