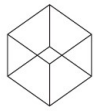
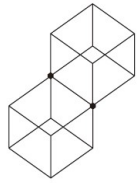
10. 데커레이터 패턴



JAVA 객체 지향 디자인 패턴

UML과 GoF 디자인 패턴 핵심 10가지로 배우는



학습목표

학습목표

- 독립적인 추가 기능의 조합 방법 이해하기
- 데커레이터 패턴을 통한 기능의 조합 방법 이해하기
- 사례 연구를 통한 데커레이터 패턴의 핵심 특징 이해하기

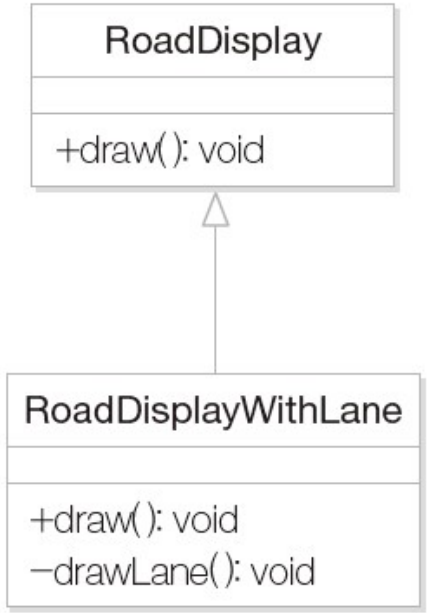
10.1 도로 표시 방법 조합하기

- 네비게이션 SW에 도로를 표시하는 기능
  - 기본기능 : 도로를 간단한 선으로 표시
  - 도로의 차선을 표시하는 기능 (추가기능)

❖ 구현

- RoadDisplay 클래스: 기본 도로 표시 기능을 제공하는 클래스
- RoadDisplayWithLane 클래스: 기본 도로 표시에 추가적으로 차선을 표시하는 클래스

그림 10-1 기본 도로 및 차선을 표시하는 RoadDisplay와 RoadDisplayWithLane 클래스의 설계



소스 코드

코드 10-1

```
public class RoadDisplay {  // 기본 도로 표시 클래스
   public void draw() {
      System.out.println("도로 기본 표시") ;
   }
}
```

```
public class RoadDisplayWithLane extends RoadDisplay {  // 기본 도로 표시 + 차선 표시 클래스
   public void draw() {
      super.draw(); // 상위 클래스 즉 RoadDisplay의 draw 메서드를 호출해서 기본 도로를 표시
      drawLane();
   }
   private void drawLane() {
      System.out.println("차선 표시") ;
   }
}
```

```
public class Client {
   public static void main(String[] args) {
      RoadDisplay road = new RoadDisplay() ;
      road.draw() ;  // 기본 도로만 표시

      RoadDisplay roadWithLane = new RoadDisplayWithLane() ;
      roadWithLane.draw() ; // 기본 도로 + 차선 표시
   }
}
```

10.2 추가변경사항 & 문제점

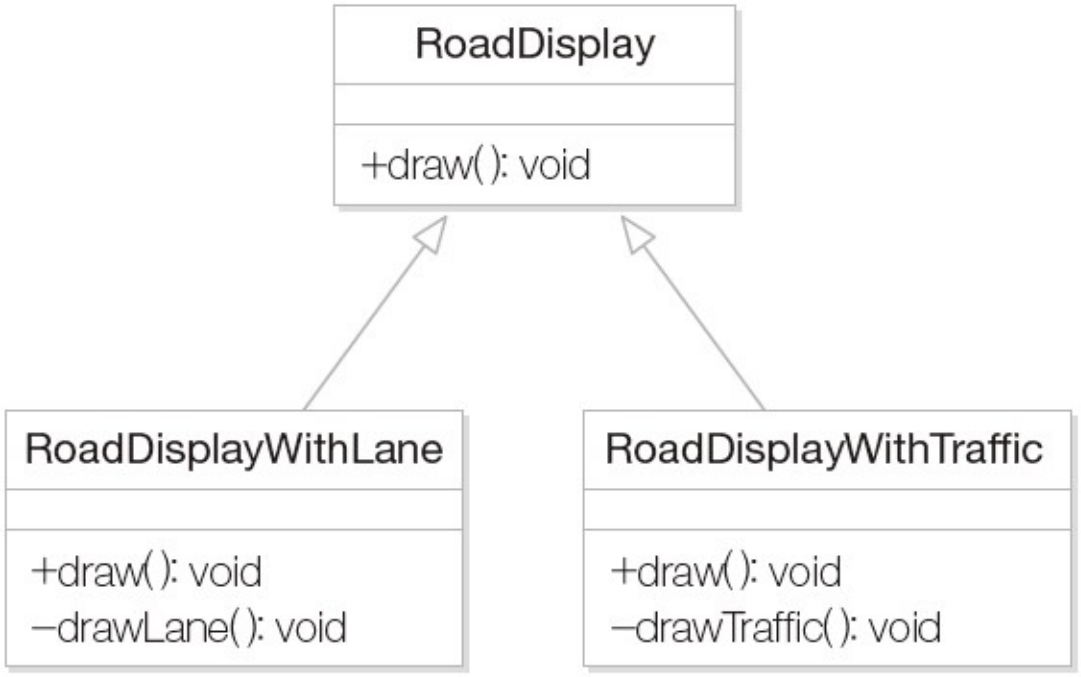
❖ 추가변경사항

- **또다른 추가적인 도로 표시 기능을 구현하고 싶다면 어떻게 해야 하는가?**
  - 예를 들어 기본 도로 표시에 교통량을 표시하고 싶다면?
- **뿐만 아니라 여러가지 다양한 추가 기능의 (다양하게) 조합하여 제공하고 싶다면 어떻게 해야 하는가?**
  - 예를 들어 기본 도료 표시에 차선 표시 기능과 교통량 표시 기능을 함께 제공하고 싶다면 ?

10.2.1 또다른 도로 표시 기능을 추가로 구현하는 경우

❖ 기본 도로 표시에 추가적으로 교통량을 표시하는 경우

그림 10-2 기본 도로 및 교통량을 표시하는 RoadDisplayWithTraffic 클래스의 설계



10.2.1 또다른 도로 표시 기능을 추가로 구현하는 경우

코드 10-2

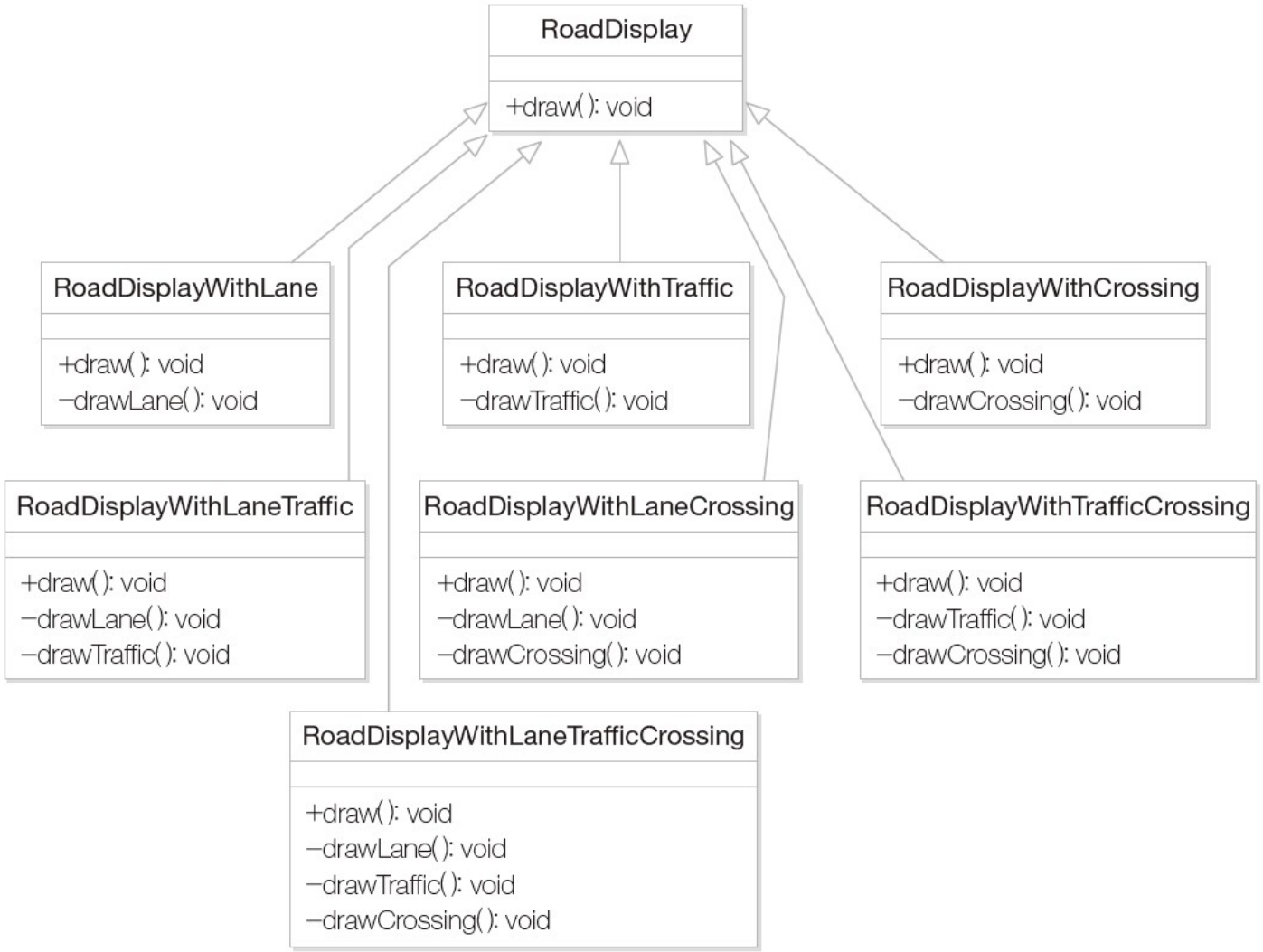
```
public class RoadDisplayWithTraffic extends RoadDisplay {
   public void draw() {
      super.draw();
      drawTraffic();
   }
   private void drawTraffic() {
      System.out.println("교통량 표시") ;
   }
}
```

10.2.2 여러가지(다양한) 추가 기능을 (다양하게) 조합해야 하는 경우

경우	기본 기능 도로 표시	추가 기능			클래스 이름
		차선 표시	교통량 표시	교차로 표시	
1	√				RoadDisplay
2	√	√			RoadDisplayWithLane
3	√		√		RoadDisplayWithTraffic
4	√			√	RoadDisplayWithCrossing
5	√	√	√		RoadDisplayWithLaneTraffic
6	√	√		√	RoadDisplayWithLaneCrossing
7	√		√	√	RoadDisplayWithTrafficCrossing
8	√	√	√	√	RoadDisplayWithLaneTrafficCrossing

10.2.2 여러가지 추가 기능을 조합해야 하는 경우

그림 10-3 상속을 이용한 추가 기능 조합의 설계



❖ 추가되는 모든 Class

- RoadDisplay 클래스의 하위 클래스로 설계

❖ 문제점

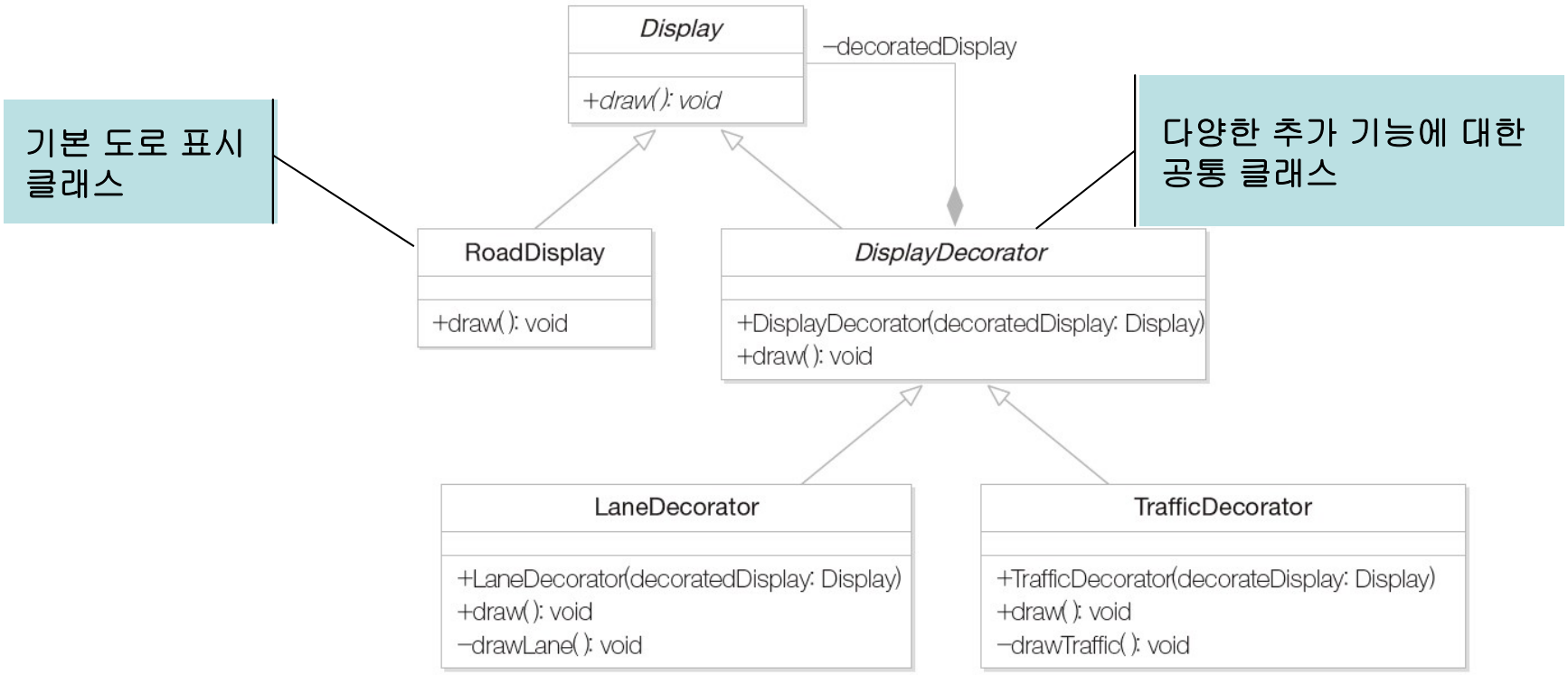
- 조합하는 방식에 따라서 많은 수의 Class 필요

10.3. 해결책

❖ **추가 기능 별로 개별적인 클래스를 설계하고 이를 조합**

- **기능을 조합할때 각 클래스의 객체 조합을 이용**

그림 10-4 개선된 추가 기능 조합의 설계



❖ 그림 10 – 4

- **기본 기능만 이용할 때 …. RoadDisplay 클래스의 객체를 생성**
- **차선을 표시하는 기능이 추가적으로 필요할때**
  - LaneDecorator 클래스의 객체 필요
  - 이때 LaneDecorator 클래스에서는 차선표시기능만 직접 제공
    - ➢ 도로표시기능은 RoadDisplay 클래스의 draw 메서드를 호출함
  - LaneDecorator 클래스에서 RoadDisplay 객체 참조 필요
    - ➢ LaneDecorator 클래스인 DisplayDecoratory에서 Display 클래스의 컴포지션 관계로 표현

10.3. 해결책: 소스 코드

코드 10-3

```
public abstract class Display {
    public abstract void draw() ;
}
```

```
public class RoadDisplay extends Display {  // 기본 도로 표시 클래스
   public void draw() {
      System.out.println("도로 기본 표시") ;
   }
}
```

```
// 다양한 추가 기능에 대한 공통 클래스
public class DisplayDecorator extends Display {
   private Display decoratedDisplay ;
   public DisplayDecorator(Display decoratedDisplay) {
      this.decoratedDisplay = decoratedDisplay ;
   }
   public void draw() {
      decoratedDisplay.draw() ;
   }
}
```

10.3. 해결책: 소스 코드

코드 10-3

```
public class LaneDecorator extends DisplayDecorator {  // 차선표시를 축하는 클래스
   public LaneDecorator(Display decoratedDisplay) { // 기존 표시 클래스의 설정
      super(decoratedDisplay);
   }
   public void draw() {
      super.draw(); // 설정된 기존 표시 기능을 수행
      drawLane() ; // 추가적으로 차선을 표시
   }
   private void drawLane() { System.out.println("\t차선 표시") ; }
}
```

```
public class TrafficDecorator extends DisplayDecorator { // 교통량 표시를 추가하는 클래스
   public TrafficDecorator(Display decoratedDisplay) { // 기존 표시 클래스의 설정
      super(decoratedDisplay);
   }
   public void draw() {
      super.draw();  // 설정된 기존 표시 기능을 수행
      drawTraffic() ; // 추가적으로 교통량을 표시
   }
   private void drawTraffic() { System.out.println("\t교통량 표시") ; }
}
```

10.3. 해결책: 소스 코드

코드 10-4

```
public class Client {
   public static void main(String[] args) {
      Display road = new RoadDisplay() ;
      road.draw() ;  // 기본 도로 표시

      Display roadWithLane = new LaneDecorator(new RoadDisplay()) ;
      roadWithLane.draw() ;  // 기본 도로 표시 + 차선 표시

      Display roadWithTraffic = new TrafficDecorator(new RoadDisplay()) ;
      roadWithTraffic.draw() ;  // 기본 도로 표시 + 교통량 표시
}
```

Client 클래스는 동일한 Display 클래스만을 통해서 일관성 있는 방식으로 도로 정보를 표시

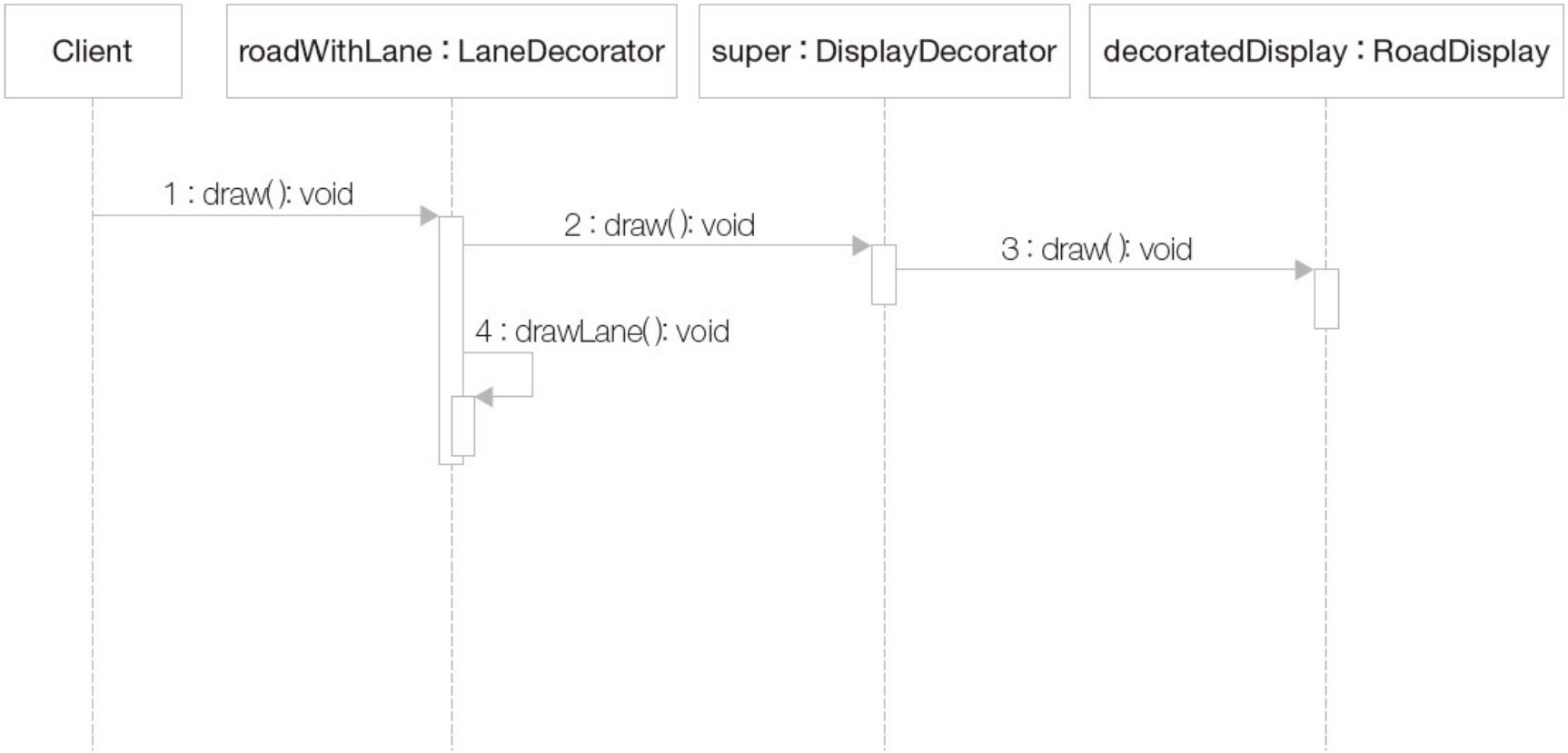
```
도로 기본 표시
도로 기본 표시
            차선 표시
도로 기본 표시
            교통량 표시
```

❖ 코드 10-4

- road, roadWithLane, roadWithTraffic 객체의 접근이 모두 Display 클래스를 통해 이루어짐
- Client 클래스는 동일한 Display 클래스만을 통해 일관성있는 방식으로 도로 정보를 표시할수 있음

10.3. 해결책

그림 10-5 roadWithLane 객체의 draw 메서드 동작



❖ 추가 요구사항

- 기본도로표시기능에 추가적으로 차선도 표시하고, 교통량도 표시하고 싶다.

10.3. 해결책: 소스 코드

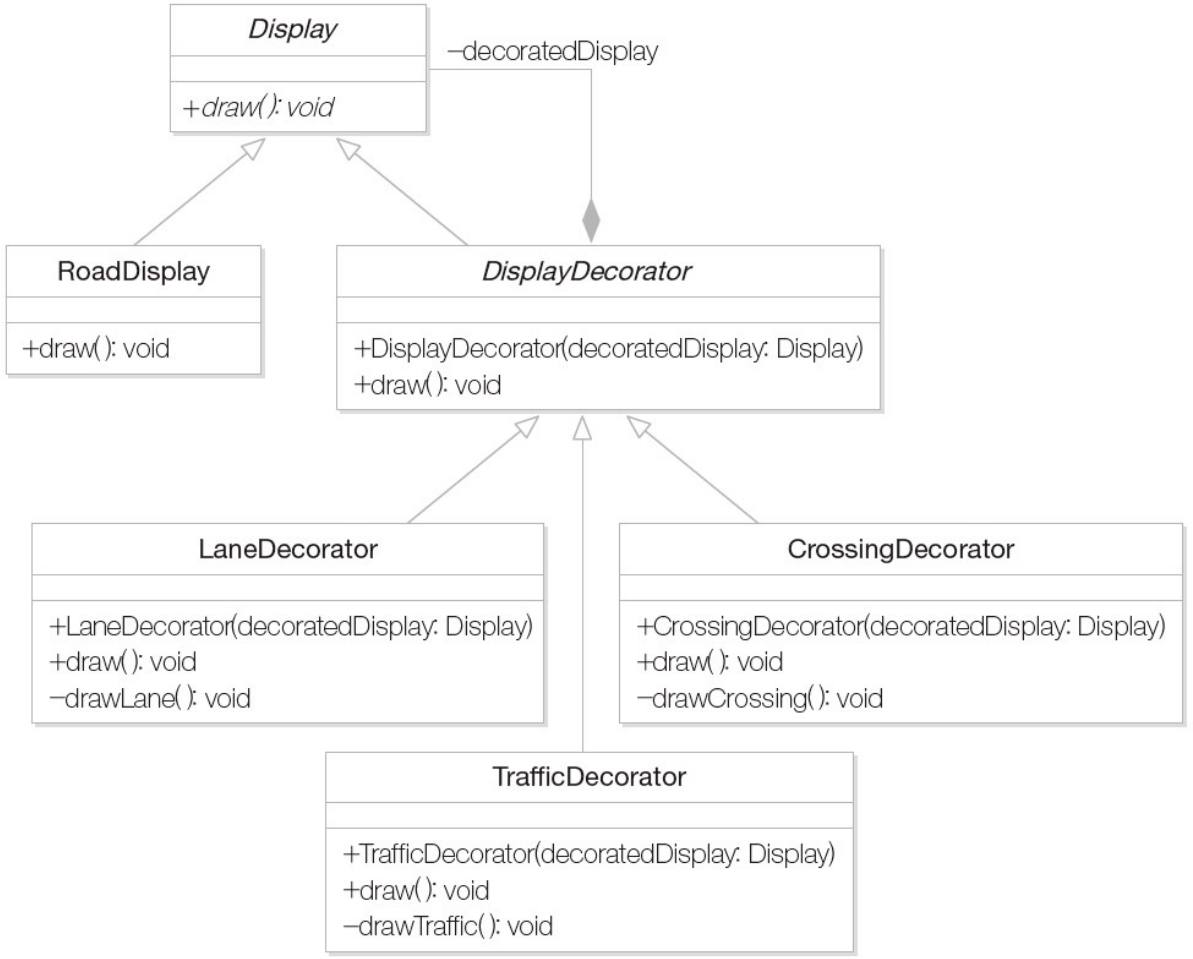
코드 10-5

```
public class Client {
   public static void main(String[] args) {
   Display roadWithLaneAndTraffic =
      new TrafficDecorator(
         new LaneDecorator(
            new RoadDisplay()));
      roadWithLaneAndTraffic.draw() ;
   }
}
```

```
도로 기본 표시
        차선 표시
        교통량 표시
```

추가 기능 : 교차로 기능 추가

그림 10-6 LaneDecorator, TrafficDecorator, CrossingDecorator의 관계



CrossingDecoroator

코드 10-6

```
public class CrossingDecorator extends DisplayDecorator {
   public CrossingDecorator(Display decoratedDisplay) {
      super(decoratedDisplay);
   }
   public void draw() {
      super.draw();
      drawCrossing() ;
   }
   private void drawCrossing() {
      System.out.println("\t횡단보도 표시") ;
   }
}
```

교차로 기능 추가

코드 10-7

```
public class Client {
   public static void main(String[] args) {
   Display roadWithLaneAndTraffic =
      new LaneDecorator(
         new TrafficDecorator(
            new CrossingDecorator(
               new RoadDisplay())));
      roadWithCrossingAndTrafficAndLane.draw() ;
   }
}
```

```
도로 기본 표시
        횡단보도 표시
        교통량 표시
        차선 표시
```

10.4 데커레이터(Decorator) 패턴

❖ 기본 기능에 추가할 수 있는 기능의 종류가 많은 경우

- 각 추가가능을 Decorator 클래스로 정의한후
- Decoroator 객체를 조합함

- 예) 기본도로표시, 차선표시, 교통량표시, 교차로표시, 단속카메라표시
  - 총 조합의 수 15가지

데코레이터 패턴은 기본 기능에 추가될 수 있는 많은 수의 부가 기능에 대해서 다양한 조합을 동적으로 구현할 수 있는 패턴이다.

10.4 데커레이터 패턴

❖ 프로그램을 실행하는 도중에 Decorator 객체의 조합이 가능

- 필요한 추가의 조합을 동적으로 생성하는 것 가능
- 코드 10-8. 프로그램 인자를 통해 명시된 추가 기능을 동적으로 생성하는 예

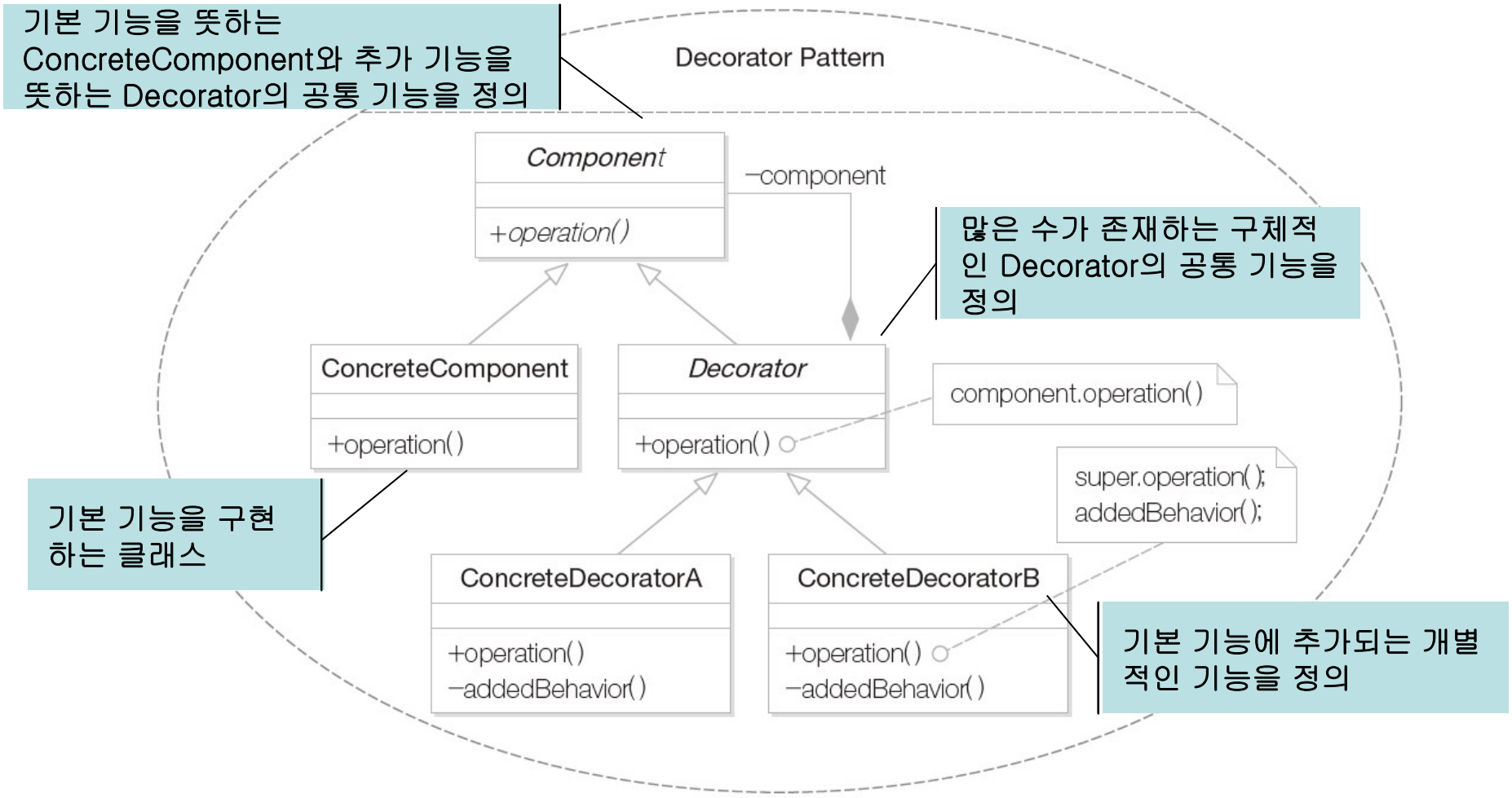
코드 10-8

```
public class Client {
   public static void main(String[] args) {
      Display road = new RoadDisplay() ;
      for ( String decoratorName : args ) {
         if ( decoratorName.equalsIgnoreCase("Lane"))
            road = new LaneDecorator(road) ;
         if ( decoratorName.equalsIgnoreCase("Traffic"))
            road = new TrafficDecorator(road) ;
         if ( decoratorName.equalsIgnoreCase("Crossing"))
            road = new CrossingDecorator(road) ;
      }
      road.draw() ;
   }
}
```

인자 예	Traffic Lane	Crossing Lane Traffic
실행 결과	도로 기본 표시 교통량 표시 차선 표시	도로 기본 표시 횡단보도 표시 차선 표시 교통량 표시

10.4 데커레이터 패턴

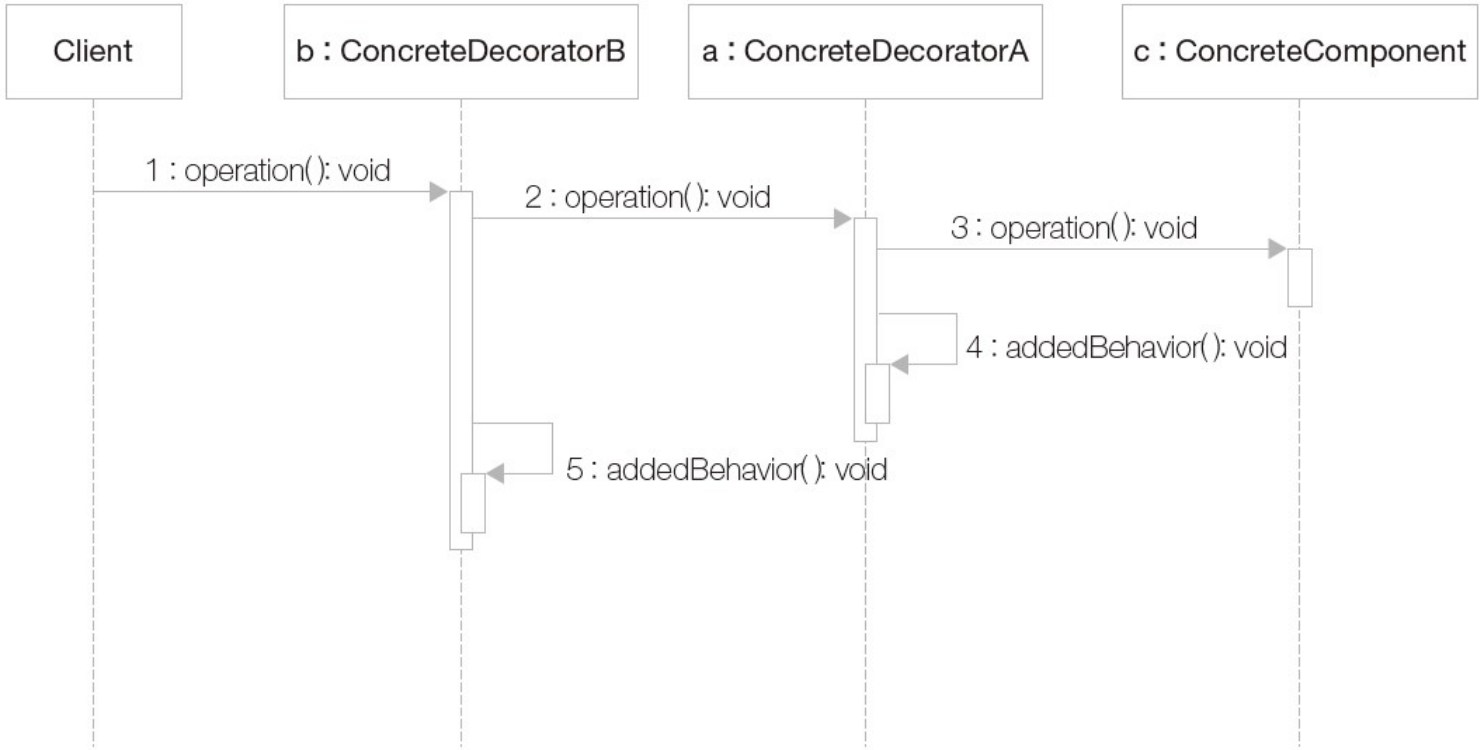
그림 10-7 데커레이터 패턴의 컬레보레이션



- **Component**: 기본 기능을 뜻하는 ConcreteComponent와 추가 기능을 뜻하는 Decorator의 공통 기능을 정의한다. 즉, 클라이언트는 Component를 통해 실제 객체를 사용한다.
- **ConcreteComponent**: 기본 기능을 구현하는 클래스다.
- **Decorator**: 많은 수가 존재하는 구체적인 Decorator의 공통 기능을 제공한다.
- **ConcreteDecoratorA, ConcreteDecoratorB**: Decorator의 하위 클래스로 기본 기능에 추가되는 개별적인 기능을 뜻한다.

9.4 데커레이터 패턴

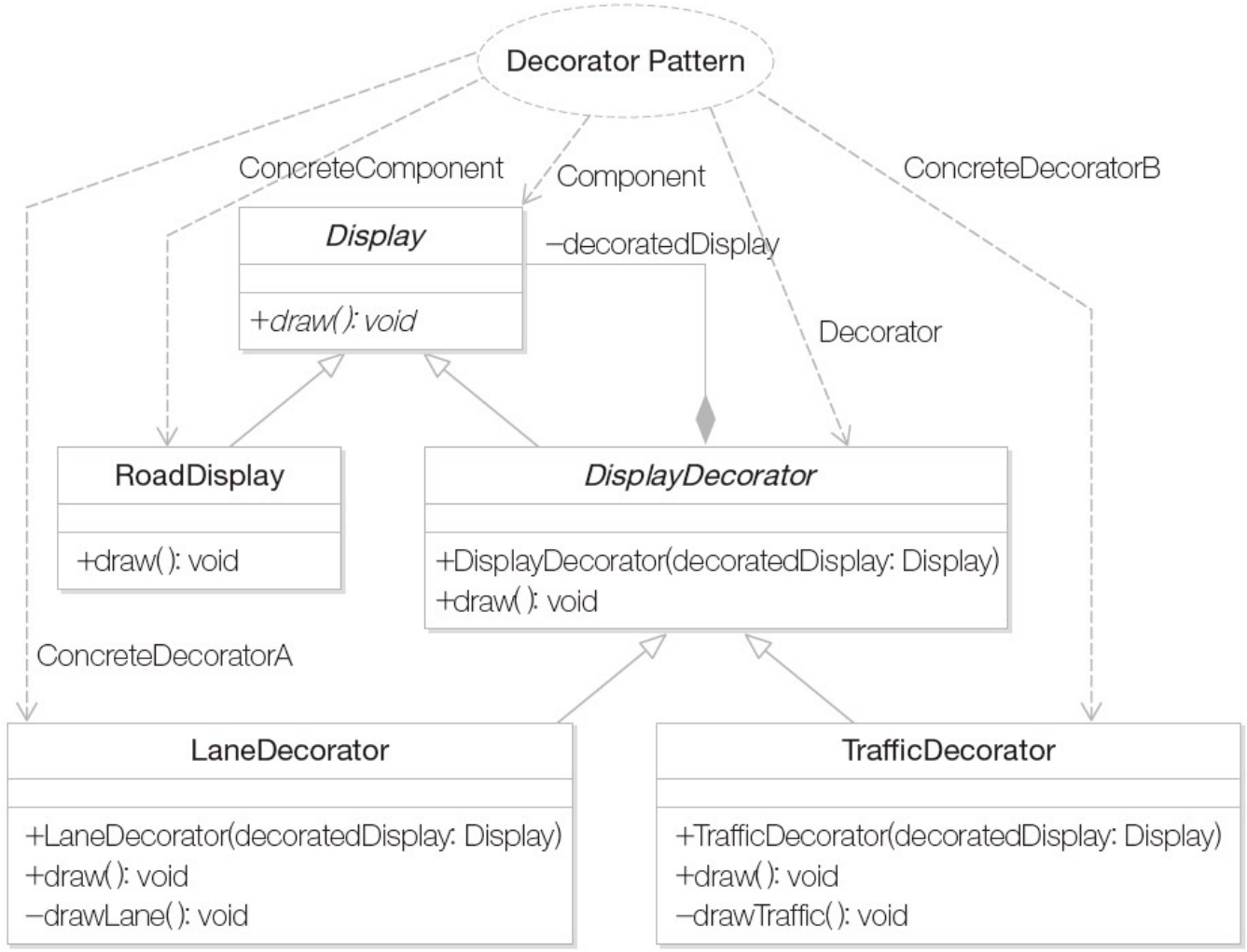
그림 10-8 데커레이터 패턴의 순차 다이어그램

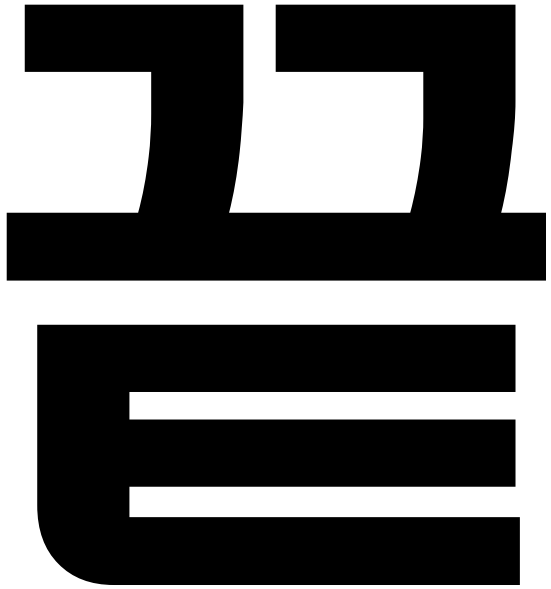



```
Component c = new ConcreteComponent() ;
Decorator a = new ConcreteDecoratorA(c) ;
Decorator b = new ConcreteDecoratorB(a) ;
```

데커레이터 패턴의 적용

그림 10-9 데커레이터 패턴을 도로 표시 예제에 적용한 경우





Digital Clock (초기버전)

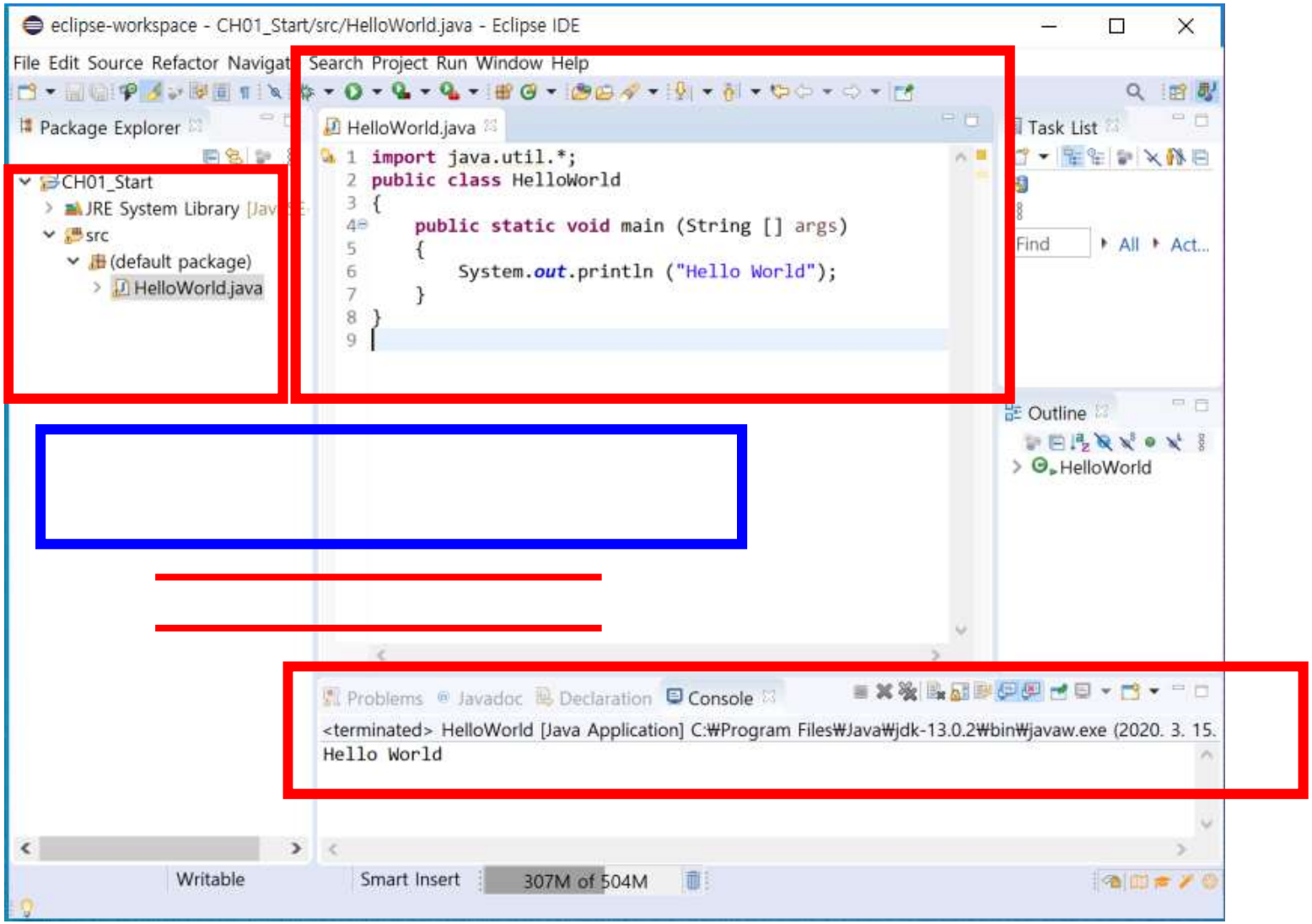
```
// Example 4.1. A Digital Clock Applet - The Initial Version


import java.awt.*;
import java.util.Calendar;

/**
 *  This is an applet that displays the time in the following format:
 *        HH:MM:SS
 */

 // must be a subclass of java.applet.Applet
public class DigitalClock
  extends java.applet.Applet implements Runnable {

  protected Thread clockThread = null;
  protected Font font = new Font("Monospaced", Font.BOLD, 48);
  protected Color color = Color.green;
```


eclipse-workspace - CH01_Start/src/HelloWorld.java - Eclipse IDE
File Edit Source Refactor Navigat Search Project Run Window Help
Package Explorer
CH01_Start
JRE System Library [Jav
src
(default package)
HelloWorld.java
HelloWorld.java
1 import java.util.*;
2 public class HelloWorld
3 {
4 public static void main (String [] args)
5 {
6 System.out.println ("Hello World");
7 }
8 }
9
Task List
Find
All
Act...
Outline
HelloWorld
Problems
Javadoc
Declaration
Console
<terminated> HelloWorld [Java Application] C:₩Program Files₩Java₩jdk-13.0.2₩bin₩javaw.exe (2020. 3. 15.
Hello World
Writable
Smart Insert
307M of 504M