

TemplateMethod 패턴

2015004239 정성운

원문 :

https://github.com/OdysseyJ/Software_Engineering/wiki/Template_Method_Pattern

Template_Method_Pattern

SeongWoon Jeong edited this page 1 minute ago · 1 revision

Software Engineering - CSE4006

Template Method Pattern

professor. 정소희

1. 개요

- Template Method Pattern에 대해서 이해한다.
 - Template Method Pattern : 전체적인 알고리즘을 구현하면서, 상이한 부분은 하위 클래스에서 구현할 수 있도록 해 주는 디자인 패턴. 전체적으로 동일하면서, 부분적으로 상이한 문장을 가지는 메소드의 코드 중복을 최소화할 때 유용하다.
 - Template 메서드와, Primitive 메서드 (또는 hook 메서드)로 구성된다.

2. 과제 설명

- Customer 클래스의 정보를 SimpleReportGenerator와 ComplexReportGenerator 클래스를 이용해 출력하려고 한다.
 - 단, ComplexReportGenerator 클래스는 주어진 고객 중 점수가 100점 이상인 고객만을 대상으로 보고서를 만든다.
- 다음 소스코드를 보고 설계 취약점을 OCP측면에서 설명하시오.
 - OCP(Open Closed Principle) : 소프트웨어는 확장에는 열려있고, 수정에는 닫혀있어야 한다는 특성.
- 설계의 취약점을 Template Method Pattern을 활용해 개선할 수 있도록 설명하고 소스코드를 작성하시오.

3. 사용언어 / 환경

- Java / Mac OS

[주어진 소스코드]

과제

```
1 package exercise02.templateMethod;
2
3 import java.util.List;
4
5 public class SimpleReportGenerator {
6     public String generate(List<Customer> customers) {
7
8         String report = String.format("고객의 수: %d 명\n", customers.size());
9
10        for ( int i = 0 ; i < customers.size() ; i ++ ) {
11            Customer customer = customers.get(i) ;
12            report += String.format("%s: %d\n", customer.getName(), customer.getPoint());
13        }
14        return report ;
15    }
16 }
```

```
1 package exercise02.templateMethod;
2
3 public class Customer {
4     private String name ;
5     private int point ;
6
7     public Customer(String name, int point) {
8         this.setName(name) ;
9         this.setPoint(point) ;
10    }
11    public int getPoint() {
12        return point;
13    }
14    public void setPoint(int point) {
15        this.point = point;
16    }
17    public String getName() {
18        return name;
19    }
20    public void setName(String name) {
21        this.name = name;
22    }
23 }
```

과제

```
1 package exercise02.templateMethod;
2
3 import java.util.ArrayList;
4
5
6 public class ComplexReportGenerator {
7     public String generate(List<Customer> customers) {
8
9         String report = String.format("고객의 수: %d 명입니다\n", customers.size());
10
11         List<Customer> selected = new ArrayList<Customer>();
12         for (Customer customer : customers)
13             if (customer.getPoint() >= 100)
14                 selected.add(customer);
15
16         for (int i = 0; i < selected.size(); i++) {
17             Customer customer = selected.get(i);
18             report += String.format("%d: %s\n", customer.getPoint(), customer.getName());
19         }
20
21         int totalPoint = 0;
22         for (Customer customer : customers)
23             totalPoint += customer.getPoint();
24
25         report += String.format("점수 합계: %d", totalPoint);
26
27         return report;
28     }
29 }
```

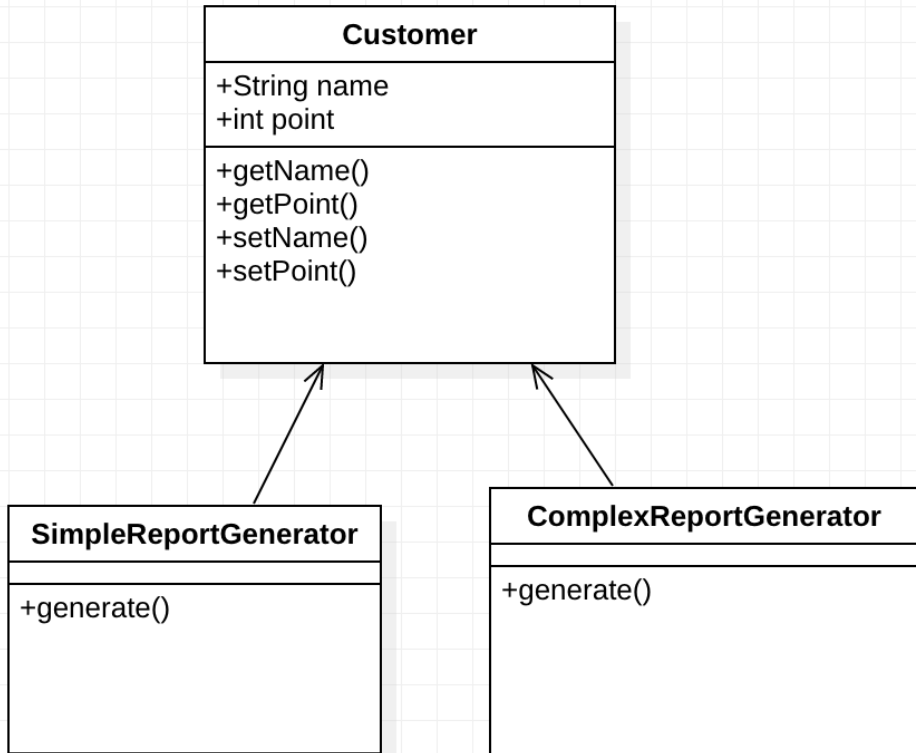
과제

```
1 package exercise02.templateMethod;
2
3 import java.util.ArrayList;
4
5
6 public class Client {
7
8     public static void main(String[] args) {
9         List<Customer> customers = new ArrayList<Customer>();
10
11         customers.add(new Customer("홍길동", 150));
12         customers.add(new Customer("우수한", 350));
13         customers.add(new Customer("부족한", 50));
14         customers.add(new Customer("훌륭한", 450));
15         customers.add(new Customer("최고의", 550));
16
17         SimpleReportGenerator simpleGenerator = new SimpleReportGenerator();
18         System.out.println(simpleGenerator.generate(customers));
19
20         ComplexReportGenerator complexGenerator = new ComplexReportGenerator();
21         System.out.println(complexGenerator.generate(customers));
22     }
23 }
```

규칙
고객의 수()
구체적 데이터()
마지막 규칙()
-> (simple x)
-> (complex sum)

고객의 수: 5 명
홍길동: 150
우수한: 350
부족한: 50
훌륭한: 450
최고의: 550

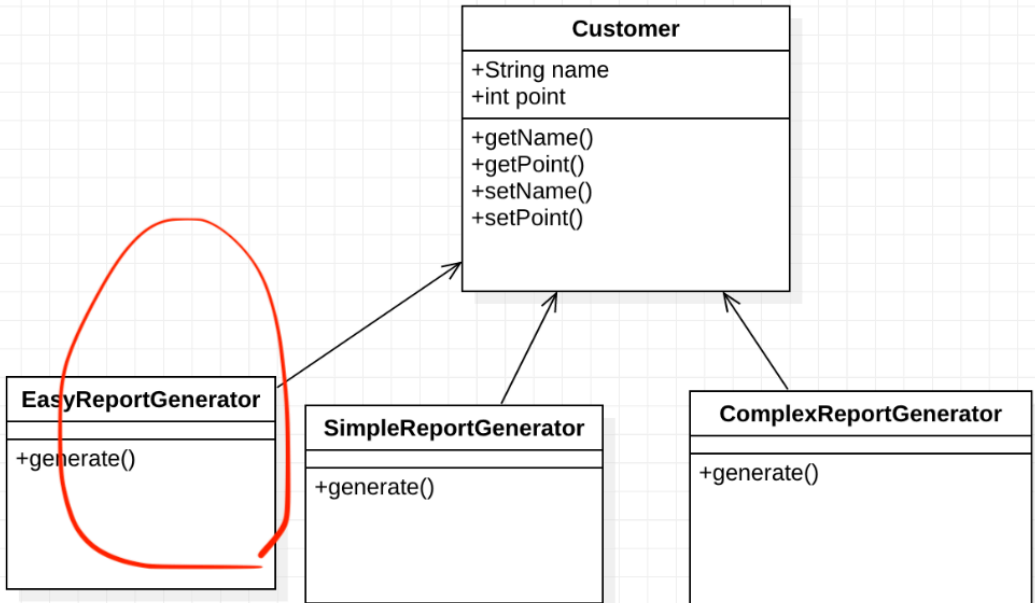
고객의 수: 5 명입니다
150: 홍길동
350: 우수한
450: 훌륭한
550: 최고의
점수 합계: 1550



비슷한 기능을하는 두개의 클래스가 Customer클래스를 출력하고 있다.

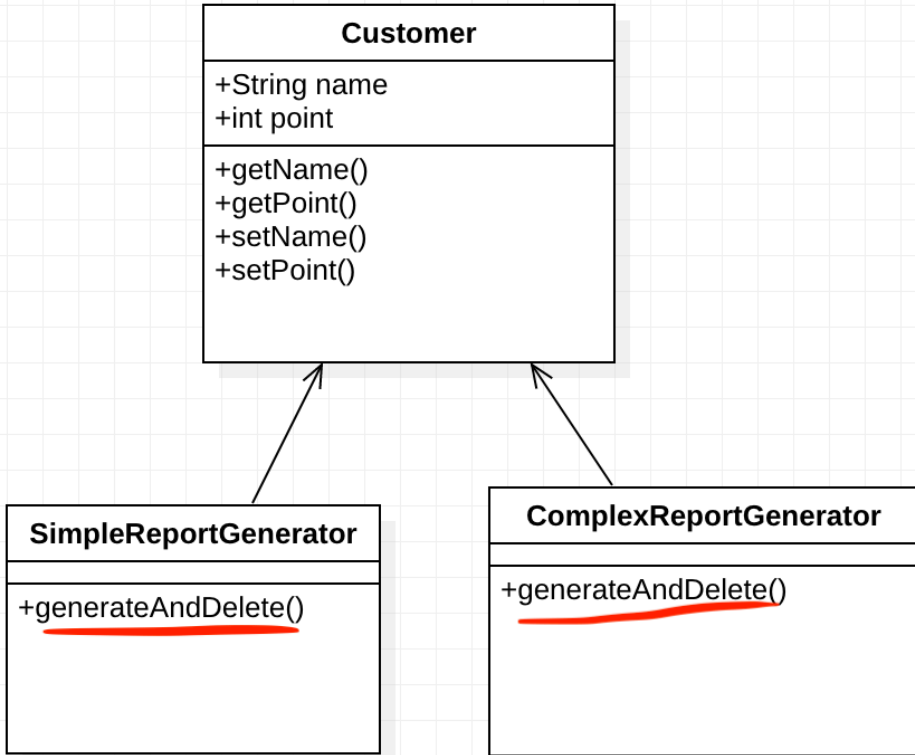
[주어진 소스코드의 취약점 분석]

- OCP의 측면에서 소프트웨어는 확장에 열려있고 변경에 닫혀있어야 하지만, 해당 설계는 확장에 닫혀있고 변경에 열려있다.
- <확장>
 - 만약, 새로운 방식의 Generator(비슷한 알고리즘, 조금 다른 출력방식)을 생성하고 싶다면 불필요한 코드의 반복을 다시해야만 한다.
 - 따라서 확장에 닫혀있다.



- <변경>

- 만약, 비슷하게 동작하는 녀석들의 동작 방식을 수정하고 싶다면? 모든 클래스의 동작 방식을 변경해 주어야 한다.
- 따라서 변경에 열려있다.

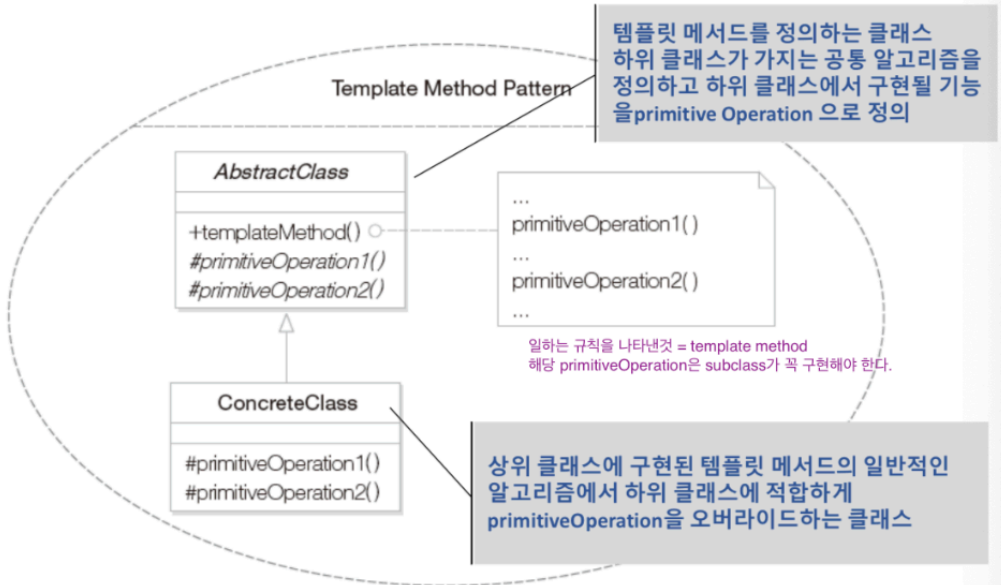


- 따라서 해당 코드는 OCP를 지키지 않고 확장에 닫혀있고 변경에 열려있는 취약한 코드이다.

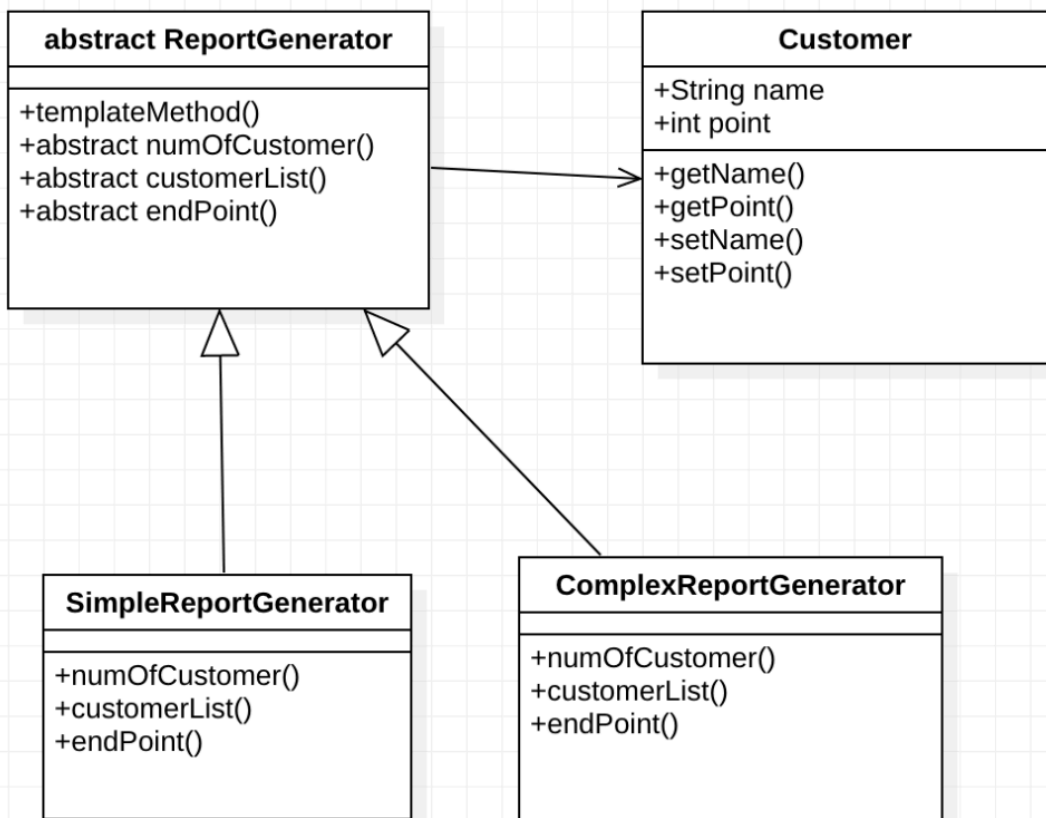
[취약점 개선 - Observer Pattern]

- Template Method Pattern은 아래의 4가지 요소로 구성된다.
 - Abstract Class
 - Concrete Class
 - Template
 - Primitive 메서드 또는 hook 메서드

Template Method Pattern 클래스 다이어그램

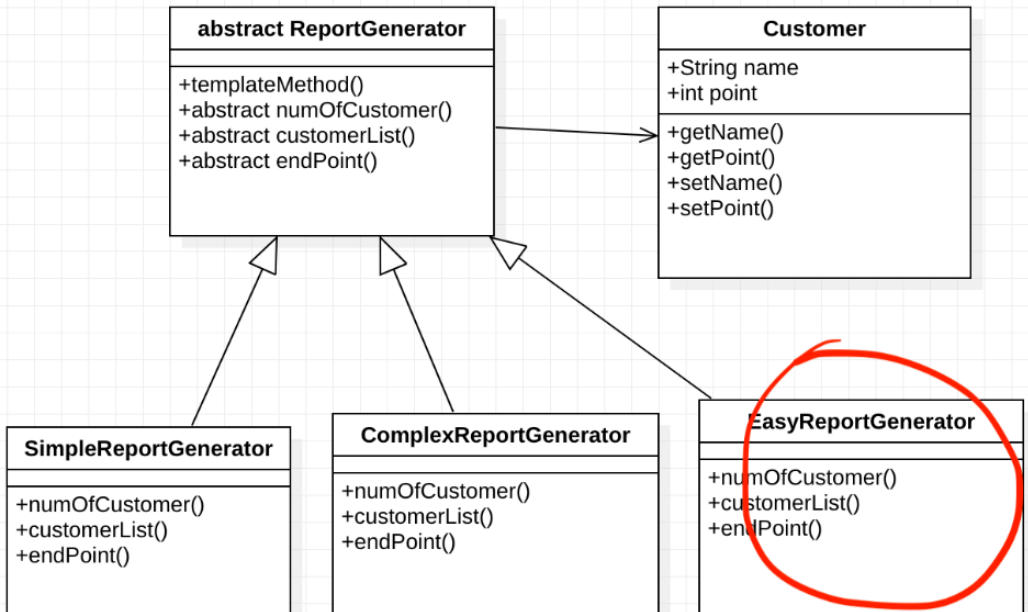


- 과제에서 주어진 소스코드를 Template Method Pattern을 적용한 결과이다.
 - Abstract Class : ReportGenerator
 - Concrete Class : SimpleReportGenerator, ComplexReportGenerator
 - Template : templateMethod()
 - Primitive 메서드 또는 hook 메서드 : numOfCustomer(), customerList(), endPoint()



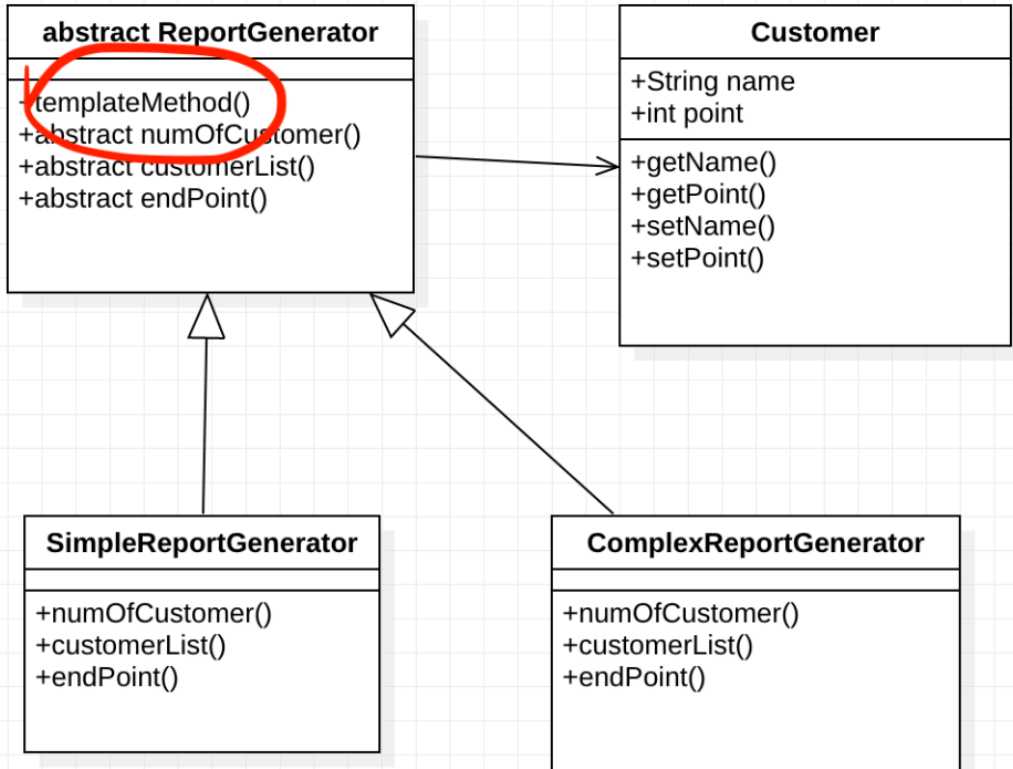
[개선된 패턴에 대한 분석]

- 해당 패턴을 적용한뒤에 얻을 수 있는 효과를 OCP의 측면에서 분석해보면,
- <확장>
 - 만약, 새로운 ReportGenerator가 추가될 경우, 코드에 대한 불필요한 반복 없이 해당 concrete class의 세부 동작 방식에 대한 구현만으로 확장할 수 있음
 - 따라서 확장에 열려있다.



- <변경>

- 만약, 기존의 알고리즘의 동작방식을 변경하려면, 원래는 모든 클래스의 동작 방식을 변경해야 했으나, 이제는 해당 Concrete Class의 templateMethod()만 변경해주면 된다.
- 따라서 변경에 닫혀있다.



- 따라서 해당 설계는 OCP를 만족한다고 할 수 있다.

[전체 소스코드]

[Customer.java] :

```
public class Customer {  
    private String name;  
    private int point;  
  
    public Customer(String name, int point) {  
        this.setName(name);  
        this.setPoint(point);  
    }  
  
    public int getPoint() {  
        return point;  
    }  
  
    public void setPoint(int point) {  
        this.point = point;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

고객 정보를 관리하기 위한 기본 customer 클래스이다.

[ReportGenerator.java] :

```
import java.util.ArrayList;
import java.util.List;

// Abstract Class
public abstract class ReportGenerator {
    // template Method
    public final String templateMethod(List<Customer>
        String head = numOfCustomer(customers);
        String body = customerList(customers);
        String tail = endPoint(customers);
        String report = head + body + tail;
        return report;
    }
    // hooker methods
    public abstract String numOfCustomer(List<Customer>
    public abstract String customerList(List<Customer>
    public abstract String endPoint(List<Customer> cus
}
```

추상메서드, 부모메서드이자 template Method, hooker method를 가지고 있는 클래스이다.

[ComplexReportGenerator.java] :

```
import java.util.ArrayList;
import java.util.List;

// Concrete Class
public class ComplexReportGenerator extends ReportGenerator {

    @Override
    public String numOfCustomer(List<Customer> customers) {
        String report = String.format("고객의 수: %d", customers.size());

        return report;
    }

    @Override
    public String customerList(List<Customer> customers) {
        String report = "";

        List<Customer> selected = new ArrayList<Customer>();

        for (Customer customer : customers)
            if (customer.getPoint() >= 100)
                selected.add(customer);

        for (int i = 0; i < selected.size(); i++)
            Customer customer = selected.get(i);
            report += String.format("%d: %s\n", i, customer.getName());
        }

        return report;
    }

    @Override
    public String endPoint(List<Customer> customers) {
        int totalPoint = 0;
        for (Customer customer : customers)
            totalPoint += customer.getPoint();

        String report = String.format("점수 합계: %d", totalPoint);

        return report;
    }
}
```

[SimpleReportGenerator.java] :

```
import java.util.ArrayList;
import java.util.List;

//Concrete Class
public class SimpleReportGenerator extends ReportGenerator

    @Override
    public String numOfCustomer(List<Customer> customer)
        String report = String.format("고객의 수: %d\n");

        return report;
    }

    @Override
    public String customerList(List<Customer> customer)
        String report = "";

        for (int i = 0; i < customers.size(); i++)
            Customer customer = customers.get(i);
            report += String.format("%s: %d\n", customer.getName(), customer.getAge());
        }

        return report;
    }

    @Override
    public String endPoint(List<Customer> customers) {
        String report = "";

        return report;
    }
}
```

Concrete Class로 ReportGenerator의 자식클래스이며, 세부 기능들에 대한 구현을 했다.

[Client.java] :

```
import java.util.ArrayList;
import java.util.List;

public class Client {
    public static void main(String[] args) {
        List<Customer> customers = new ArrayList<C

        customers.add(new Customer("홍길동", 150));
        customers.add(new Customer("우수한", 350));
        customers.add(new Customer("부족한", 50));
        customers.add(new Customer("훌륭한", 450));
        customers.add(new Customer("최고의", 550));

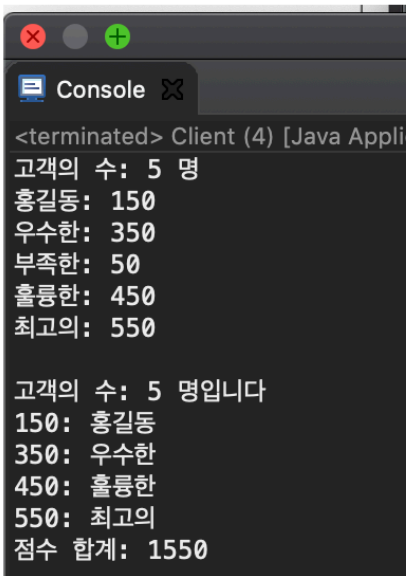
        SimpleReportGenerator simpleGenerator = ne
        System.out.println(simpleGenerator.templat

        ComplexReportGenerator complexGenerator =
        System.out.println(complexGenerator.templa

    }
}
```

클라이언트 코드이다. 주어진 클라이언트 코드와 달라지지 않았다.

[클라이언트 실행 결과 예시]



The screenshot shows a Java application window titled "Console" with a dark background. The text inside the console is as follows:

```
<terminated> Client (4) [Java Appli  
고객의 수: 5 명  
홍길동: 150  
우수한: 350  
부족한: 50  
훌륭한: 450  
최고의: 550  
  
고객의 수: 5 명입니다  
150: 홍길동  
350: 우수한  
450: 훌륭한  
550: 최고의  
점수 합계: 1550
```

- 원래 클라이언트와 같은 결과를 출력하면서, OCP를 만족하는 코드를 만들었다.