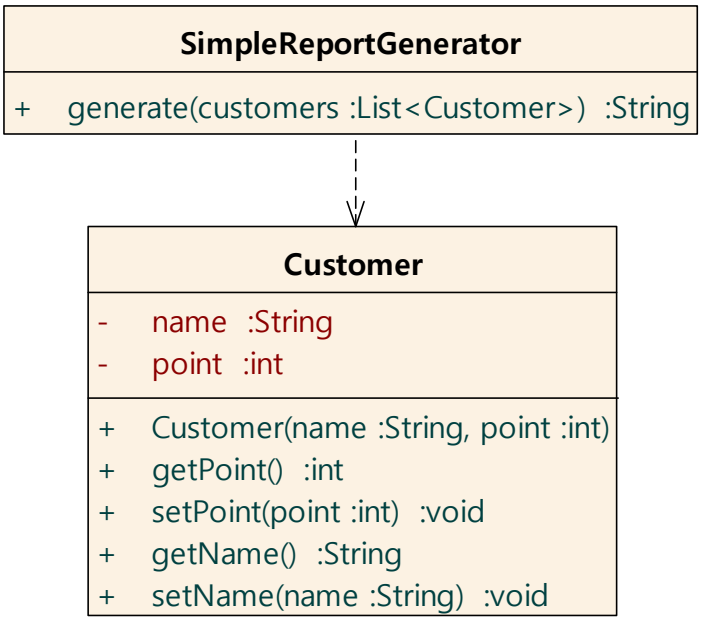
Template Method Pattern

PRACTICE – REPORT GENERATOR

Report Generator

- ReportGenerator produces a simple report based on a given Customers



Source Code - SimpleReportGenerator

```
public class SimpleReportGenerator {
  public String generate(List<Customer> customers) {

    String report = String.format("고객의 수: %d 명₩n", customers.size()) ;

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

Source Code - Client

```
public class Client {
  public static void main(String[] args) {
    List<Customer> customers = new ArrayList<Customer>() ;
    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 =
      new SimpleReportGenerator() ;
    System.out.println(simpleGenerator.generate(customers)) ;
  }
}
```

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


Violation of OOD Principle

Principle	Codes(class/method) that violate the principle
SRP	
OCP	
LSP	
ISP	
DIP	

Problem – Source Code

ComplexReportGenerator

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

    ComplexReportGenerator complexGenerator =
      new ComplexReportGenerator() ;
    System.out.println(complexGenerator.generate(customers)) ;
  }
}
```

```
고객의 수: 4 명입니다
150: 홍길동
350: 우수한
450: 훌륭한
550: 최고의
점수 합계: 1500
```

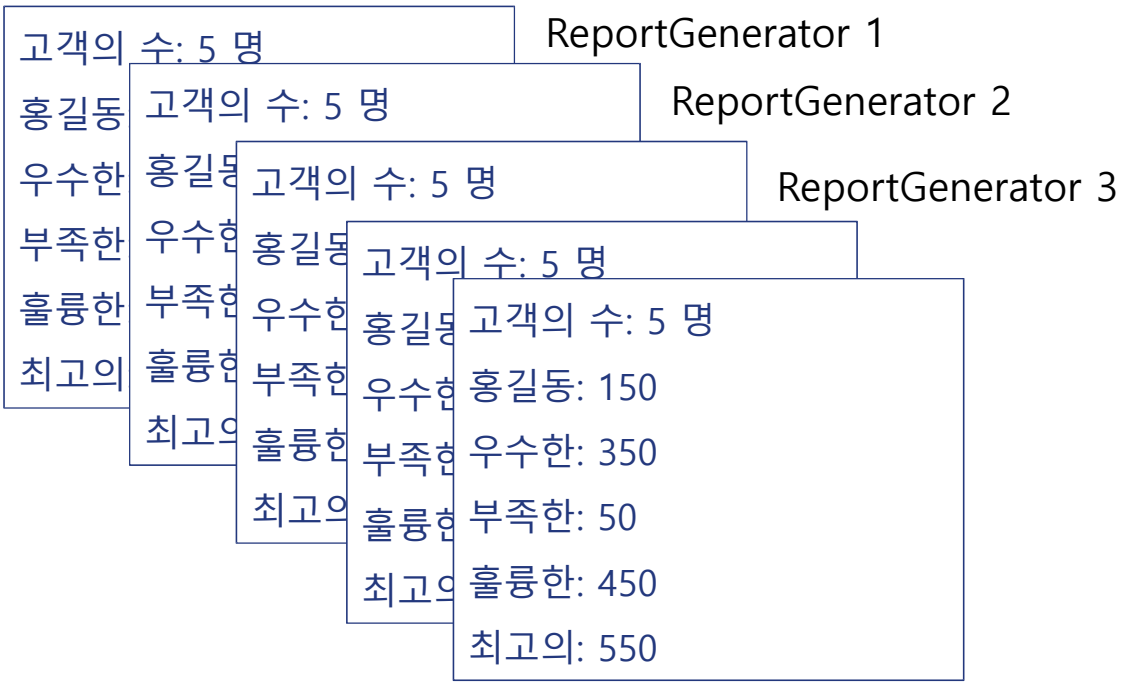
ComplexReportGenerator

```
public class ComplexReportGenerator  {
  public String generate(List<Customer> customers) {
    List<Customer> selectedCustomers = new ArrayList<Customer>() ;
    for ( Customer customer: customers )
      if ( customer.getPoint() >= 100 ) selectedCustomers.add(customer) ;
    String report = String.format("고객의 수: %d 명입니다₩n",
      selectedCustomers.size()) ;

    for ( int i = 0 ; i < selectedCustomers.size() ; i ++ ) {
      Customer customer = selectedCustomers.get(i) ;
      report += String.format("%d: %s₩n", customer.getPoint(),
        customer.getName()) ;
    }
    int totalPoint = 0 ;
    for ( Customer customer: customers)
      totalPoint += customer.getPoint() ;
    report += String.format("점수 합계: %d",  totalPoint) ;
    return report ;
  }
}
```


Problems

- New report format requires new class



Solution – Template Method Pattern

- The general format of the report is the same.
- But, they have difference in terms of
  - Header message
  - Footer message
  - Selection, Sorting

SimpleReportGenerator

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

ComplexReportGenerator

```
고객의 수: 4 명입니다
150: 홍길동
350: 우수한
450: 훌륭한
550: 최고의
점수 합계: 1500
```


Solution – Generalized Report Generator

SourceCode - ReportGenerator

Source Code - SimpleReportGenerator

Source Code - ComplexReportGenerator

Template Method vs. Strategy

	Template Method	Strategy
motivation	Reuse of general common code	Support of different algorithms
Variation Scope	Part of algorithm	Entire algorithm
Variation mechanism	Inheritance	delegation
Variation time	Compile time	Run time
Operation in superclass	Concrete	Abstract

Template Method Pattern - Summary

- Define the skeleton of an algorithm in an operation, deferring some steps to client subclasses.
- Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure