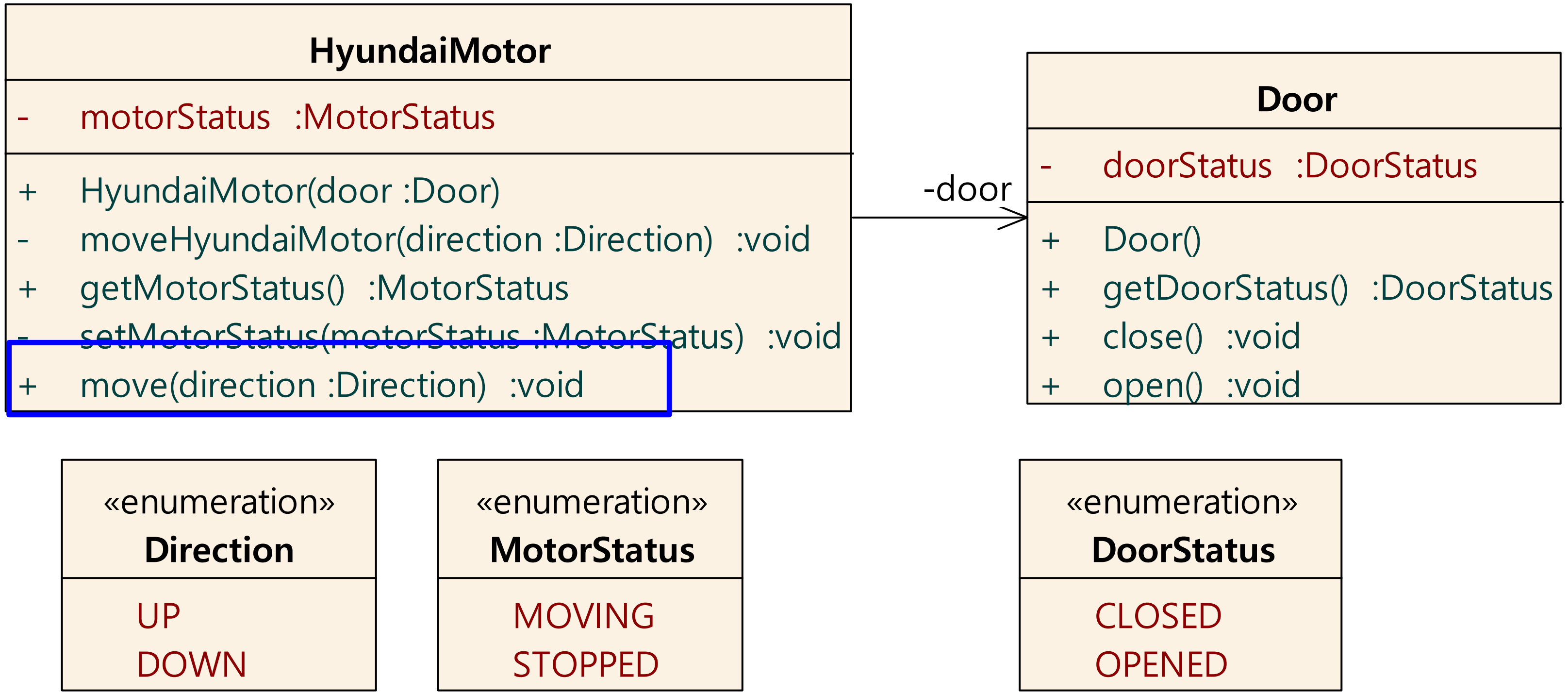
Behavioral Patterns

Template Method Pattern

Motivating Example – Hyundai Motor

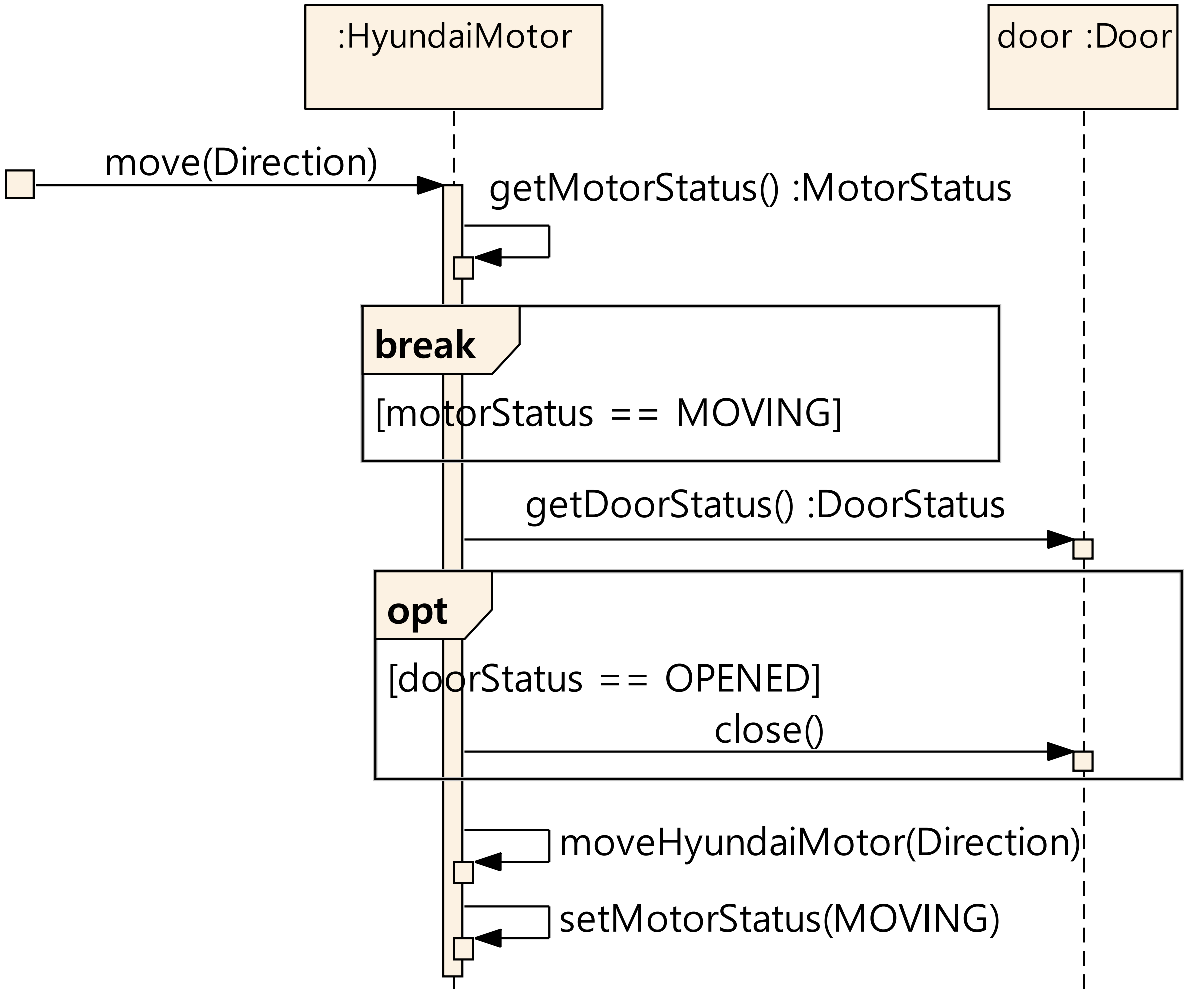
- HyundaiMotor controls the movement of an elevator



HyundaiMotor
- motorStatus :MotorStatus
+ HyundaiMotor(door :Door)
- moveHyundaiMotor(direction :Direction) :void
+ getMotorStatus() :MotorStatus
- setMotorStatus(motorStatus :MotorStatus) :void
+ move(direction :Direction) :void
-door
Door
- doorStatus :DoorStatus
+ Door()
+ getDoorStatus() :DoorStatus
+ close() :void
+ open() :void
«enumeration» Direction
UP
DOWN
«enumeration» MotorStatus
MOVING
STOPPED
«enumeration» DoorStatus
CLOSED
OPENED

Hyundai Motor

- move() depends on the status of the door as well as itself



Source Code – Door

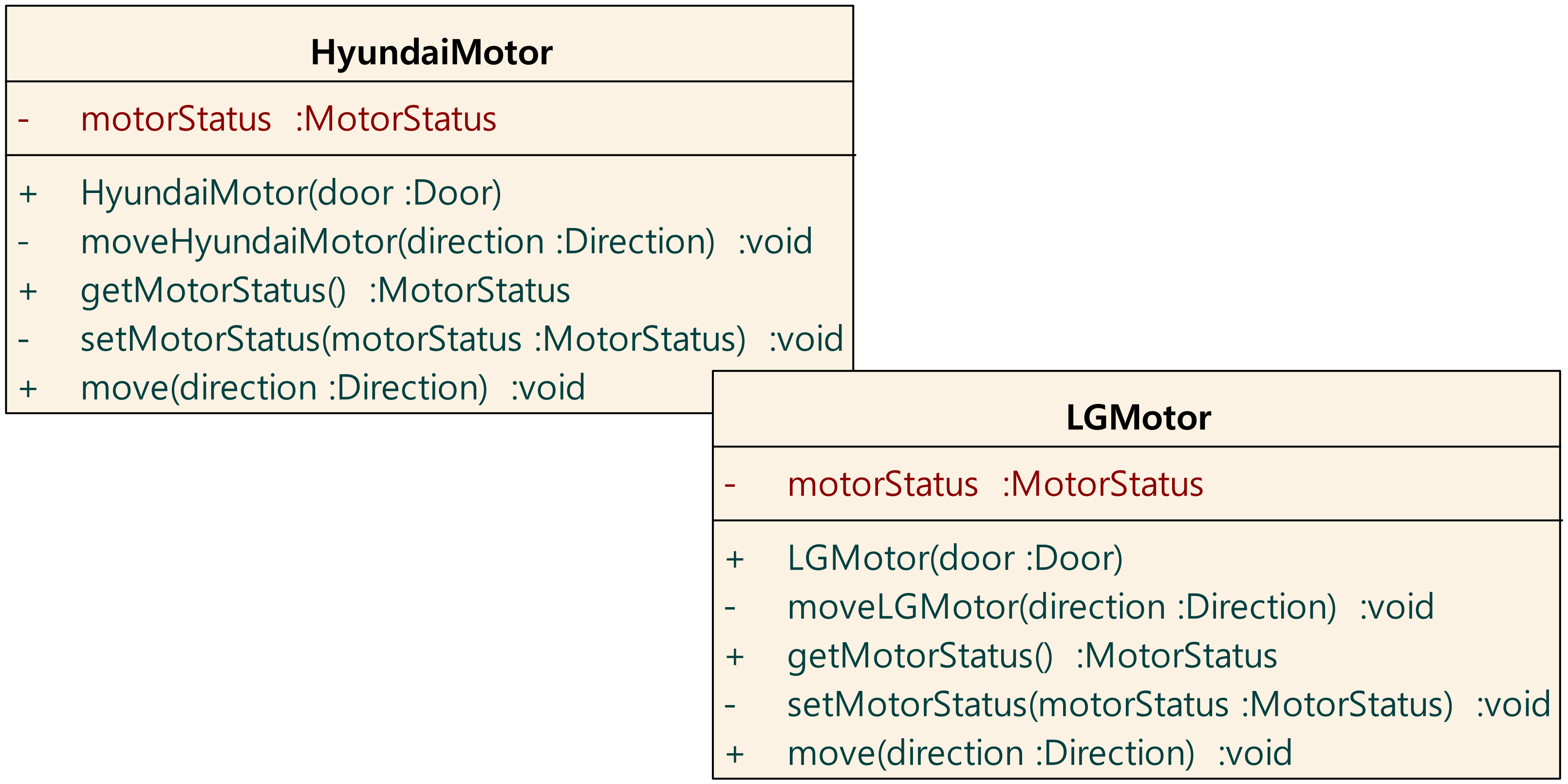
```
public class Door {
  private DoorStatus doorStatus ;
  public Door() {
    doorStatus = DoorStatus.CLOSED ;
  }
  public DoorStatus getDoorStatus() {
    return doorStatus ;
  }
  public void close() {
    doorStatus = DoorStatus.CLOSED ;
  }
  public void open() {
    doorStatus = DoorStatus.OPENED ;
  }
}
```

Source Code – HyundaiMotor

```
public class HyundaiMotor {
  private Door door ;
  private MotorStatus motorStatus ;
  public HyundaiMotor(Door door) {
    this.door = door ; motorStatus = MotorStatus.STOPPED ;
  }
  private void moveHyundaiMotor(Direction direction) {
    System.out.println("Hyundai Motor: Move " + direction) ;
  }
  public MotorStatus getMotorStatus() { return motorStatus; }
  private void setMotorStatus(MotorStatus motorStatus) {
    this.motorStatus = motorStatus;
  }
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED ) door.close() ;
    moveHyundaiMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

LGMotor

- What if LGMotor is supported?
- Just copy HyundaiMotor as LGMotor and replace Hyundai with LG


HyundaiMotor
- motorStatus :MotorStatus
+ HyundaiMotor(door :Door)
- moveHyundaiMotor(direction :Direction) :void
+ getMotorStatus() :MotorStatus
- setMotorStatus(motorStatus :MotorStatus) :void
+ move(direction :Direction) :void
LGMotor
- motorStatus :MotorStatus
+ LGMotor(door :Door)
- moveLGMotor(direction :Direction) :void
+ getMotorStatus() :MotorStatus
- setMotorStatus(motorStatus :MotorStatus) :void
+ move(direction :Direction) :void

Source Code – LGMotor

```
public class LGMotor {
  private Door door ;
  private MotorStatus motorStatus ;
  public LGMotor(Door door) {
    this.door = door ; motorStatus = MotorStatus.STOPPED ;
  }
  private void moveLGMotor(Direction direction) {
    System.out.println("LG Motor: Move " + direction) ;
  }
  public MotorStatus getMotorStatus() { return motorStatus; }
  private void setMotorStatus(MotorStatus motorStatus) {
    this.motorStatus = motorStatus;
  }
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED ) door.close() ;
    moveLGMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

Problems by Copy&Paste

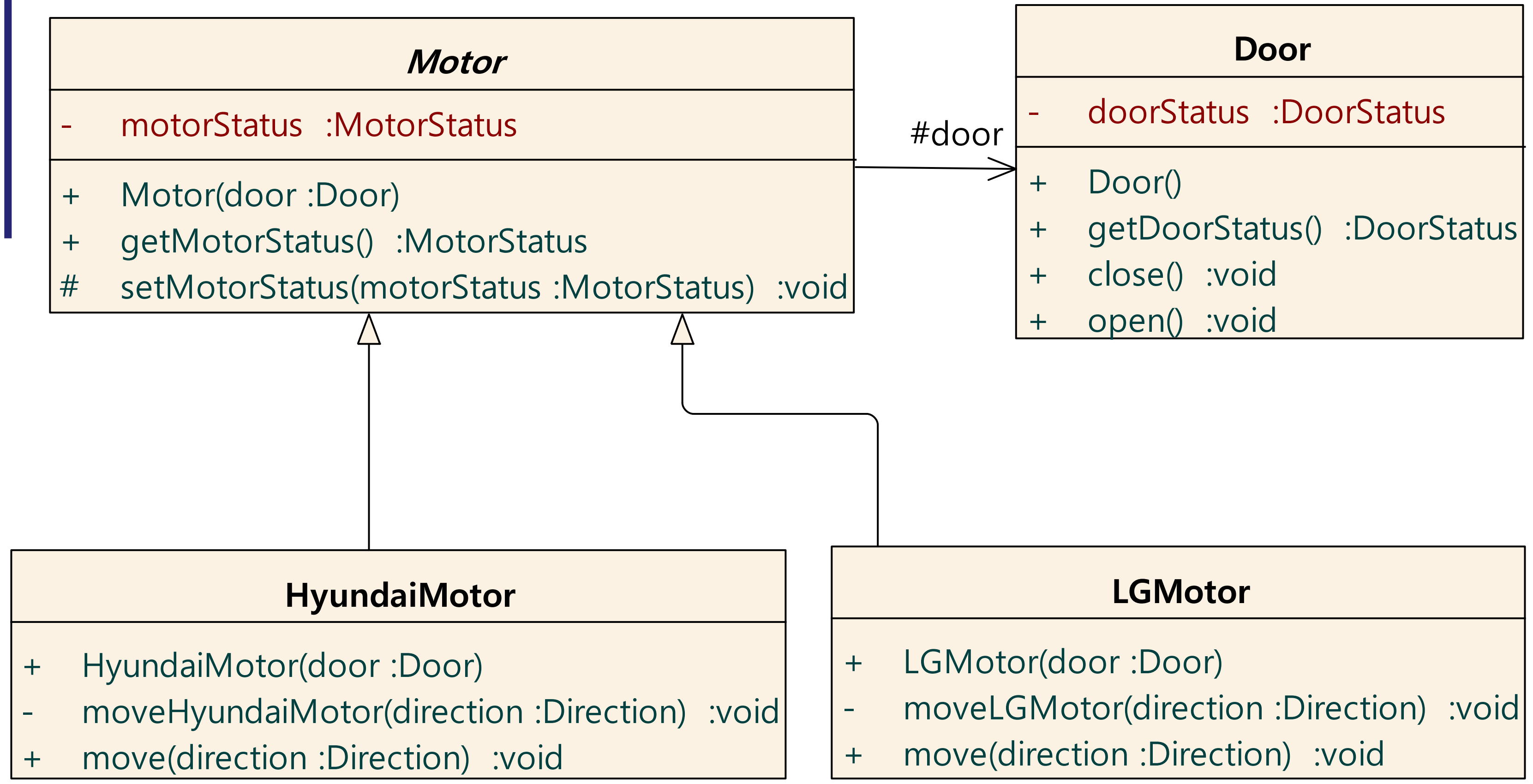
- Too many codes are duplicated!

```
public class HyundaiMotor {
  private Door door ;
  private MotorStatus motorStatus ;
  public HyundaiMotor(Door door) {
     this.door = door ; motorStatus = MotorStatus.STOPPED ;
  }
  private void moveHyundaiMotor(Direction direction) {
     System.out.println("Hyundai Motor: Move " + direction) ;
  }
  public MotorStatus getMotorStatus() { return motorStatus; }
  private void setMotorStatus(MotorStatus motorStatus) {
     this.motorStatus = motorStatus;
  }
  public void move(Direction direction) {
     MotorStatus motorStatus = getMotorStatus() ;
     if (  motorStatus == MotorStatus.MOVING ) return ;
     DoorStatus doorStatus = door.getDoorStatus() ;
     if ( doorStatus == DoorStatus.OPENED ) door.close() ;
     moveHyundaiMotor(direction) ;
     setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

```
public class LGMotor {
  private Door door ;
  private MotorStatus motorStatus ;
  public LGMotor(Door door) {
     this.door = door ; motorStatus = MotorStatus.STOPPED ;
  }
  private void moveLGMotor(Direction direction) {
     System.out.println("LG Motor: Move " + direction) ;
  }
  public MotorStatus getMotorStatus() { return motorStatus; }
  private void setMotorStatus(MotorStatus motorStatus) {
     this.motorStatus = motorStatus;
  }
  public void move(Direction direction) {
     MotorStatus motorStatus = getMotorStatus() ;
     if (  motorStatus == MotorStatus.MOVING ) return ;
     DoorStatus doorStatus = door.getDoorStatus() ;
     if ( doorStatus == DoorStatus.OPENED ) door.close() ;
     moveLGMotor(direction) ;
     setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

Solution – Make a superclass

- The superclass Motor contains the common code


Motor
- motorStatus :MotorStatus
+ Motor(door :Door)
+ getMotorStatus() :MotorStatus
# setMotorStatus(motorStatus :MotorStatus) :void
#door
Door
- doorStatus :DoorStatus
+ Door()
+ getDoorStatus() :DoorStatus
+ close() :void
+ open() :void
HyundaiMotor
+ HyundaiMotor(door :Door)
- moveHyundaiMotor(direction :Direction) :void
+ move(direction :Direction) :void
LGMotor
+ LGMotor(door :Door)
- moveLGMotor(direction :Direction) :void
+ move(direction :Direction) :void

Source Code - Motor

```
public abstract class Motor {
  protected Door door ;
  private MotorStatus motorStatus ;

  public Motor(Door door) {
    this.door = door ;
    motorStatus = MotorStatus.STOPPED ;
  }
  public MotorStatus getMotorStatus() {
    return motorStatus;
  }
  protected void setMotorStatus(MotorStatus motorStatus) {
    this.motorStatus = motorStatus;
  }
}
```

Source Code - HyundaiMotor

```
public class HyundaiMotor extends Motor {
  public HyundaiMotor(Door door) {
    super(door) ;
  }
  private void moveHyundaiMotor(Direction direction) {
    System.out.println("Hyundai Motor: Move " + direction) ;
  }
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED )
      door.close() ;
    moveHyundaiMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

Source Code - LGMotor

```
public class LGMotor extends Motor {
  public LGMotor(Door door) {
    super(door) ;
  }
  private void moveLGMotor(Direction direction) {
    System.out.println("LG Motor: Move " + direction) ;
  }
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED )
      door.close() ;
    moveLGMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

We still have problem!

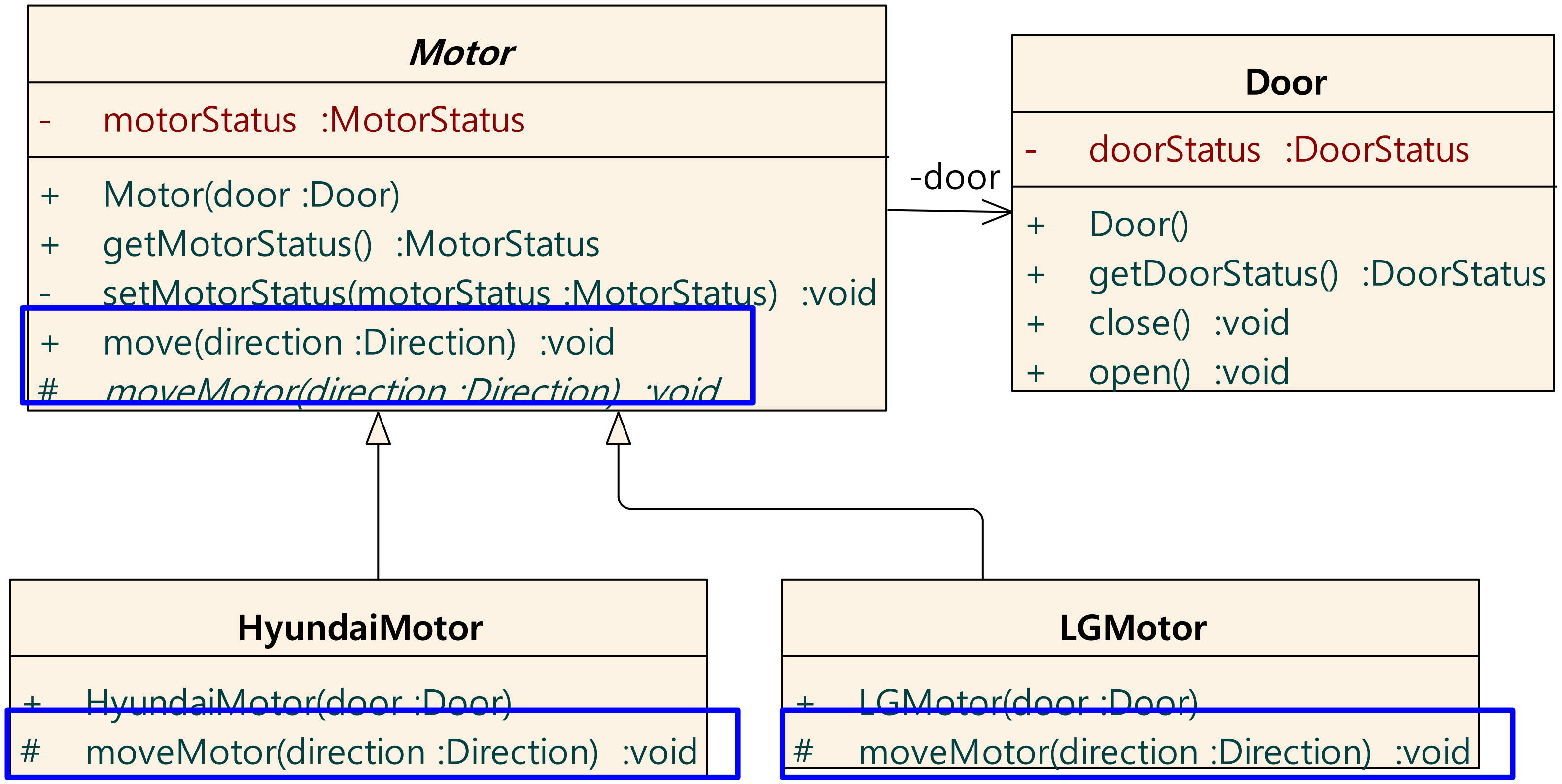
- Many codes in move() of two motors are duplicated

```
public class LGMotor extends Motor {
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED ) door.close() ;
    moveLGMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

```
public class HyundaiMotor extends Motor {
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED ) door.close() ;
    moveHyundaiMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
}
```

Solution – Template Method Pattern

- Define the skeleton of an algorithm, deferring some steps to subclasses


Motor
- motorStatus :MotorStatus
+ Motor(door :Door)
+ getMotorStatus() :MotorStatus
- setMotorStatus(motorStatus :MotorStatus) :void
+ move(direction :Direction) :void
# moveMotor(direction :Direction) :void
-door
Door
- doorStatus :DoorStatus
+ Door()
+ getDoorStatus() :DoorStatus
+ close() :void
+ open() :void
HyundaiMotor
+ HyundaiMotor(door :Door)
# moveMotor(direction :Direction) :void
LGMotor
+ LGMotor(door :Door)
# moveMotor(direction :Direction) :void

Source Code – Motor

```
public abstract class Motor {
  private Door door ;
  private MotorStatus motorStatus ;
  public Motor(Door door) {
    this.door = door ; motorStatus = MotorStatus.STOPPED ;
  }
  public MotorStatus getMotorStatus() { return motorStatus; }
  private void setMotorStatus(MotorStatus motorStatus) {
    this.motorStatus = motorStatus;
  }
  public final void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED ) door.close() ;
    moveMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
  protected abstract void moveMotor(Direction direction) ;
}
```

Implements skeleton of an algorithm

This step is deferred to subclasses

Source Code – Concrete Motors

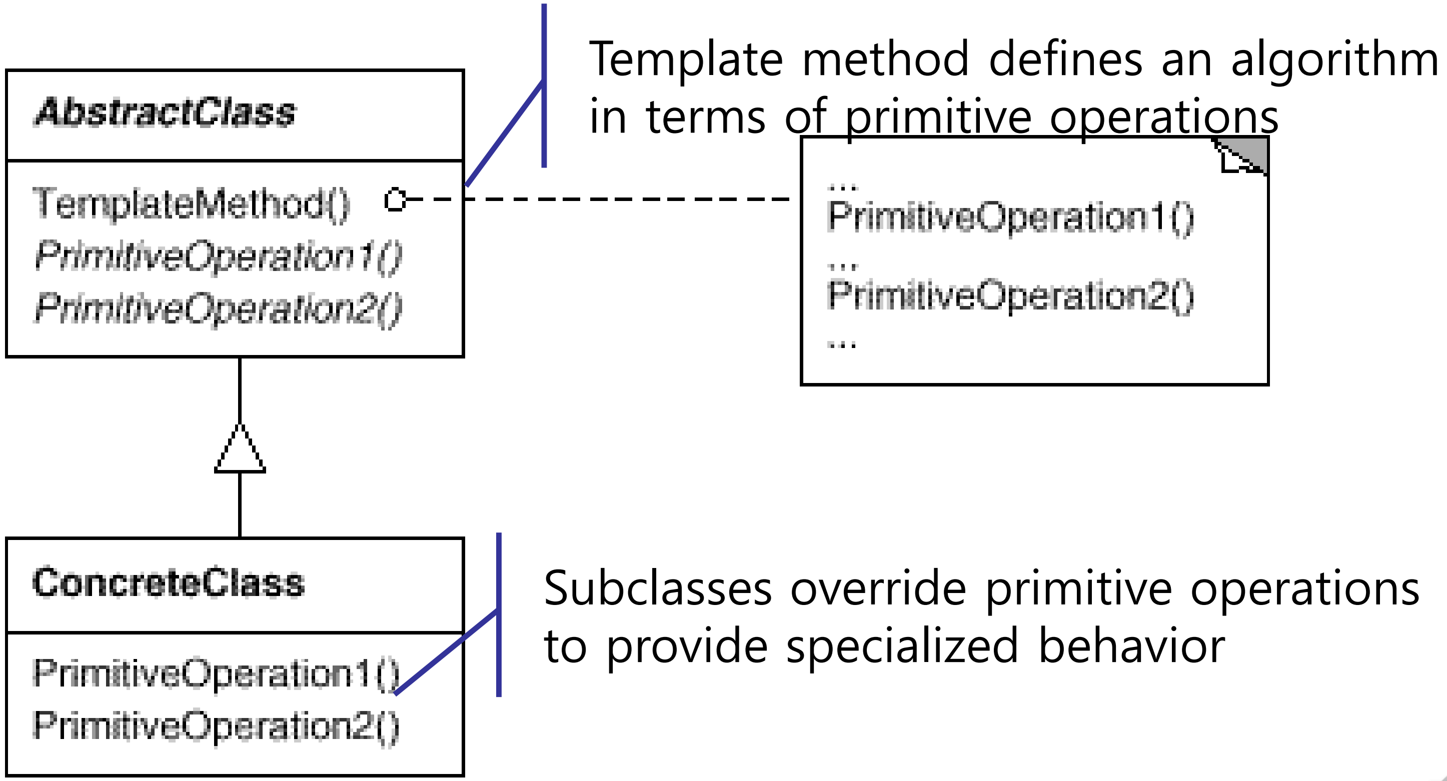
- Concrete motor implements the deferred step to perform subclass-specific steps of the algorithm

```
public class HyundaiMotor extends Motor {
  public HyundaiMotor(Door door) {
    super(door) ;
  }
  protected void moveMotor(Direction direction) {
    System.out.println("Hyundai Motor: Move " + direction) ;
  }
}
```

```
public class LGMotor extends Motor {
  public LGMotor(Door door) {
    super(door) ;
  }
  protected void moveMotor(Direction direction) {
    System.out.println("LG Motor: Move " + direction) ;
  }
}
```

Template Method Pattern: Structure

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



Template Method Pattern

```
public abstract class Motor {
  …
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED ) door.close() ;
    moveMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
  protected abstract void moveMotor(Direction direction) ;
}
```

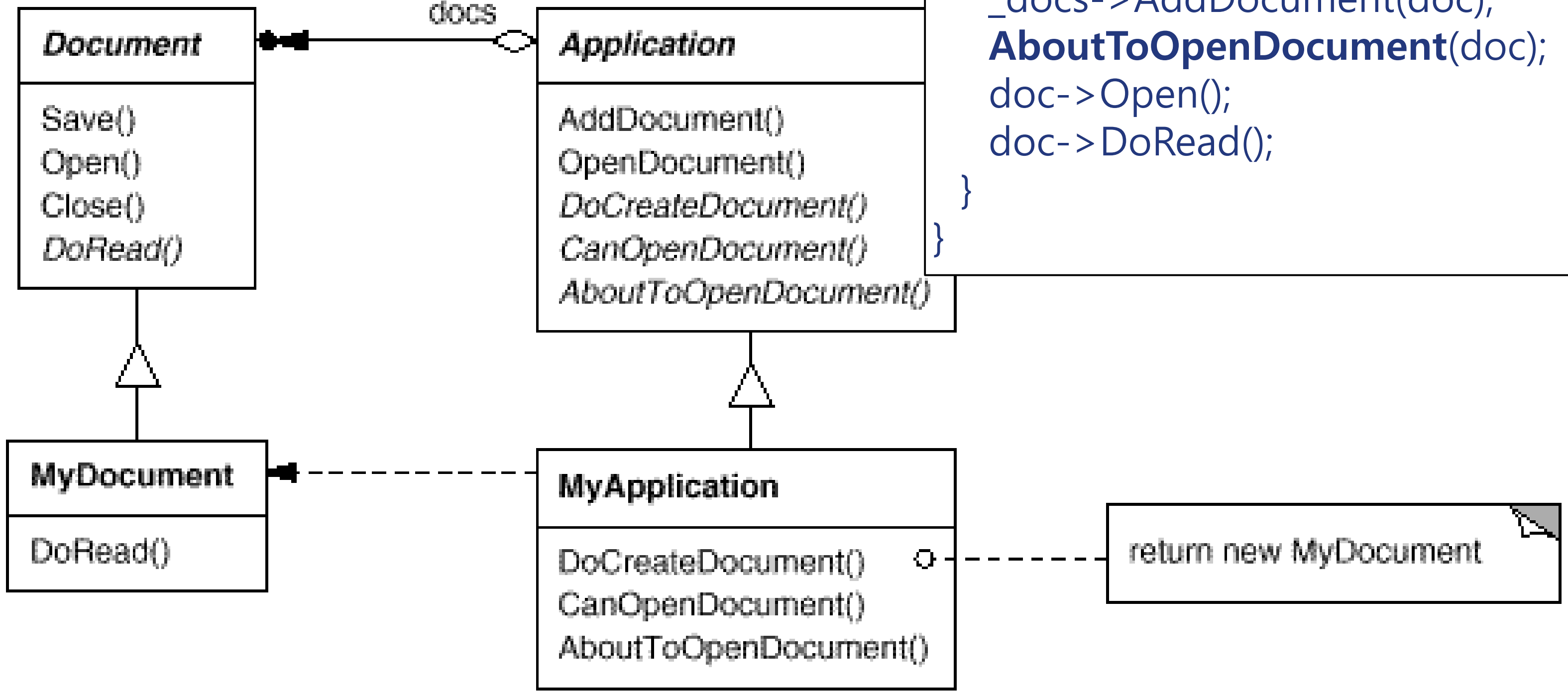
Template method

Primitive operation

```
public class HyundaiMotor extends Motor {
  public HyundaiMotor(Door door) {
    super(door) ;
  }
  protected void moveMotor(Direction direction) {
    System.out.println("Hyundai Motor: Move " + direction) ;
  }
}
```

Can be defined as primitive operation (preprocess()) for more reusability

Template Method Pattern

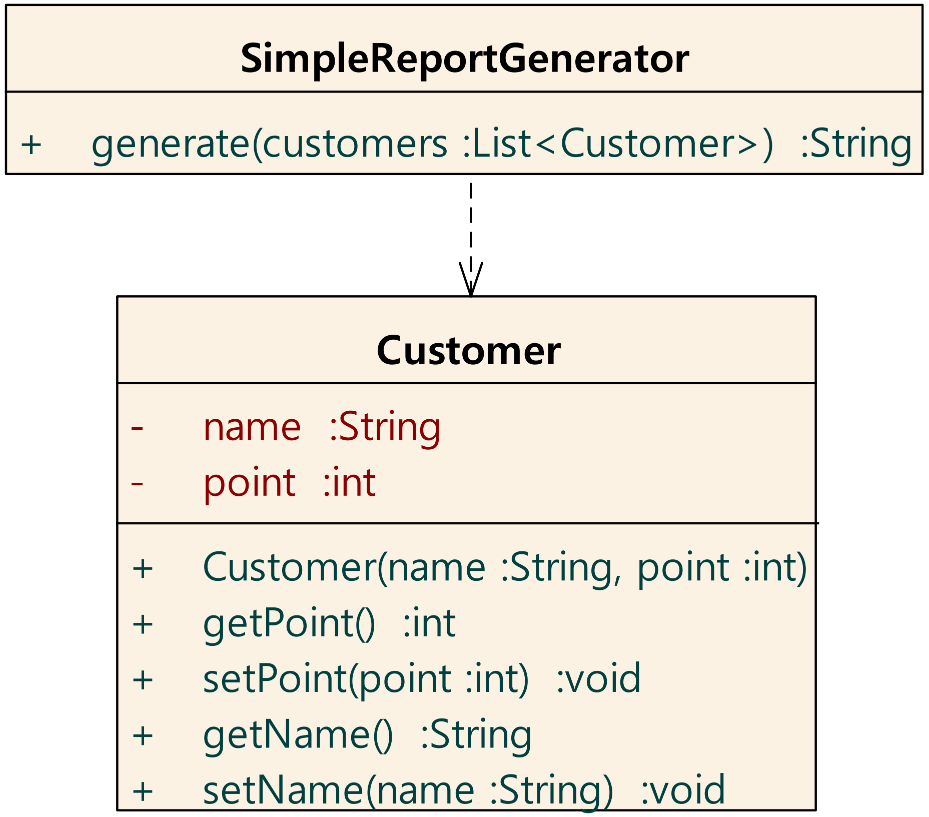


```
void Application::OpenDocument(char* name){
  if (!CanOpenDocument(name)) {
    // cannot handle this document
    return;
  }
  Document* doc = DoCreateDocument();
  if (doc) {
    _docs->AddDocument(doc);
    AboutToOpenDocument(doc);
    doc->Open();
    doc->DoRead();
  }
}
```

PRACTICE – REPORT GENERATOR

Report Generator

- ReportGenerator produces a simple report based on a given Customers


SimpleReportGenerator
+ generate(customers :List<Customer>) :String
Customer
- name :String
- point :int
+ Customer(name :String, point :int)
+ getPoint() :int
+ setPoint(point :int) :void
+ getName() :String
+ setName(name :String) :void

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
```

Problem – Source Code

```
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 ;
  }
}
```

OCP
We have to modify the code to support different report

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
```

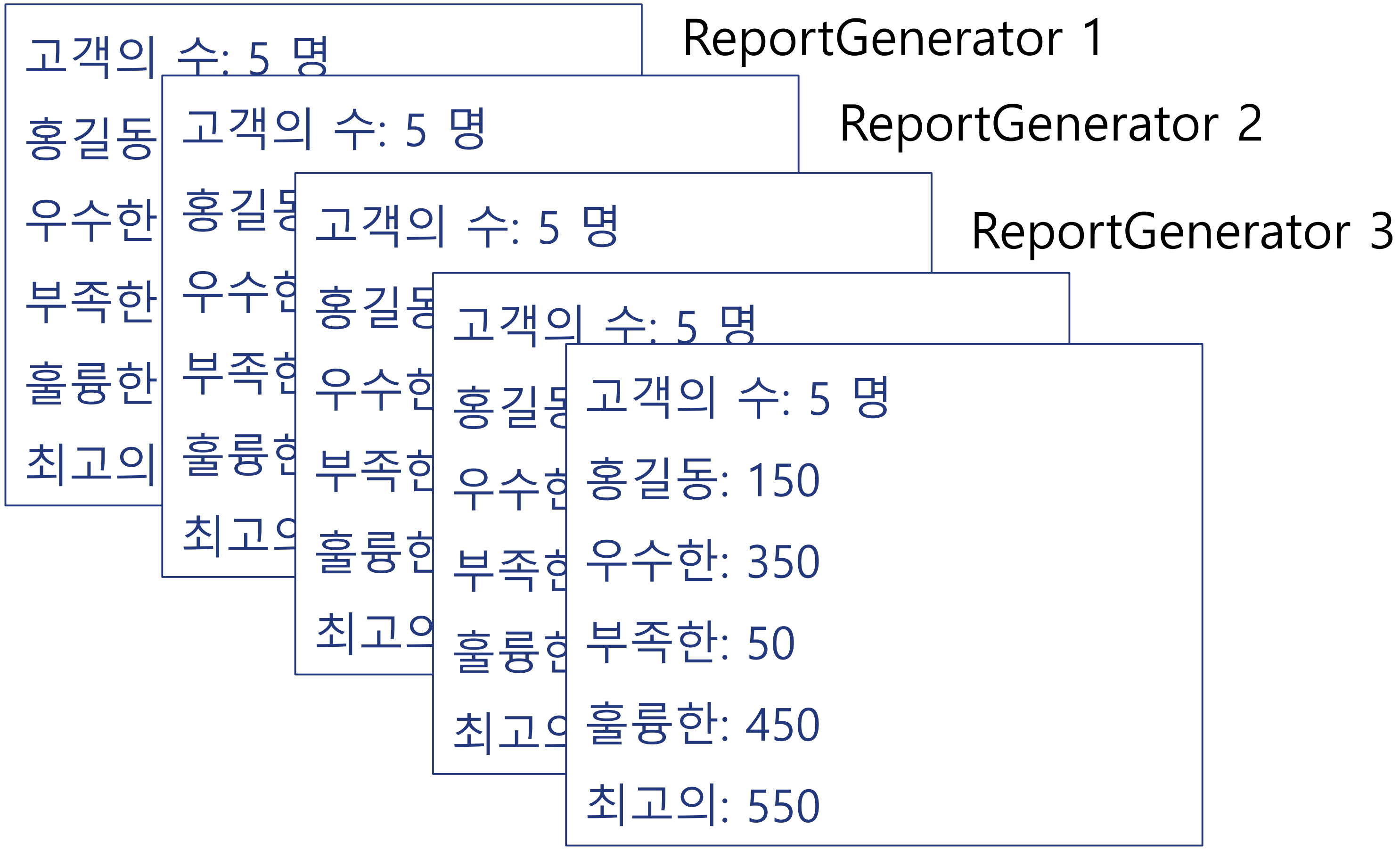
ComplexReportGenerator

```
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

- Classes for each report format have common codes



고객의 수: 5 명

홍길동: 150

우수한: 350

부족한: 50

훌륭한: 450

최고의: 550

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

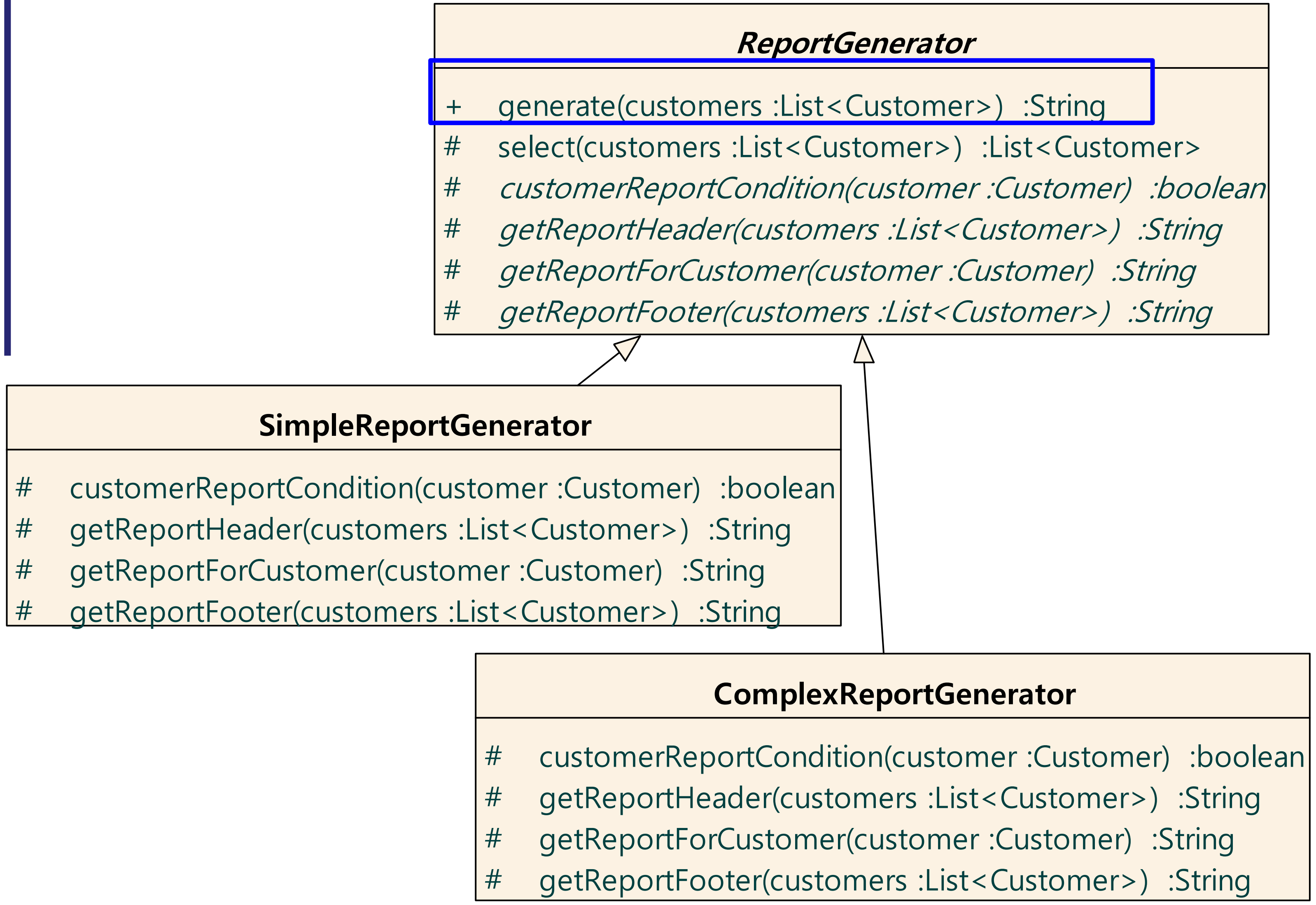
SimpleReportGenerator

```
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


ReportGenerator
+ generate(customers :List<Customer>) :String
# select(customers :List<Customer>) :List<Customer>
# *customerReportCondition(customer :Customer) :boolean*
# *getReportHeader(customers :List<Customer>) :String*
# *getReportForCustomer(customer :Customer) :String*
# *getReportFooter(customers :List<Customer>) :String*
SimpleReportGenerator
# customerReportCondition(customer :Customer) :boolean
# getReportHeader(customers :List<Customer>) :String
# getReportForCustomer(customer :Customer) :String
# getReportFooter(customers :List<Customer>) :String
ComplexReportGenerator
# customerReportCondition(customer :Customer) :boolean
# getReportHeader(customers :List<Customer>) :String
# getReportForCustomer(customer :Customer) :String
# getReportFooter(customers :List<Customer>) :String

Source Code - ReportGenerator

```
public abstract class ReportGenerator {
  public final String generate(List<Customer> customers) {
    List<Customer> selectedCustomers = select(customers) ;
    String report = getReportHeader(selectedCustomers) ;
    for ( int i = 0 ; i < selectedCustomers.size() ; i ++ ) {
      Customer customer = selectedCustomers.get(i) ;
      report += getReportForCustomer(customer) ;
    }
    report += getReportFooter(selectedCustomers) ;
    return report ;
  }
  protected List<Customer> select(List<Customer> customers) {
    List<Customer> selected = new ArrayList<Customer>() ;
    for ( Customer customer: customers )
      if ( customerReportCondition(customer) ) selected.add(customer) ;
    return selected;
  }
  protected abstract boolean customerReportCondition(Customer customer) ;
  protected abstract String getReportHeader(List<Customer> customers) ;
  protected abstract String getReportForCustomer(Customer customer) ;
  protected abstract String getReportFooter(List<Customer> customers) ;
}
```

Source Code - SimpleReportGenerator

```
public class SimpleReportGenerator extends ReportGenerator {
  protected boolean customerReportCondition(Customer customer) {
    return true ;
  }
  protected String getReportHeader(List<Customer> customers) {
    return String.format("고객의 수: %d 명₩n", customers.size()) ;
  }
  protected String getReportForCustomer(Customer customer) {
    return String.format("%s: %d₩n", customer.getName(),
      customer.getPoint()) ;
  }
  protected  String getReportFooter(List<Customer> customers) {
    return "" ;
  }
}
```

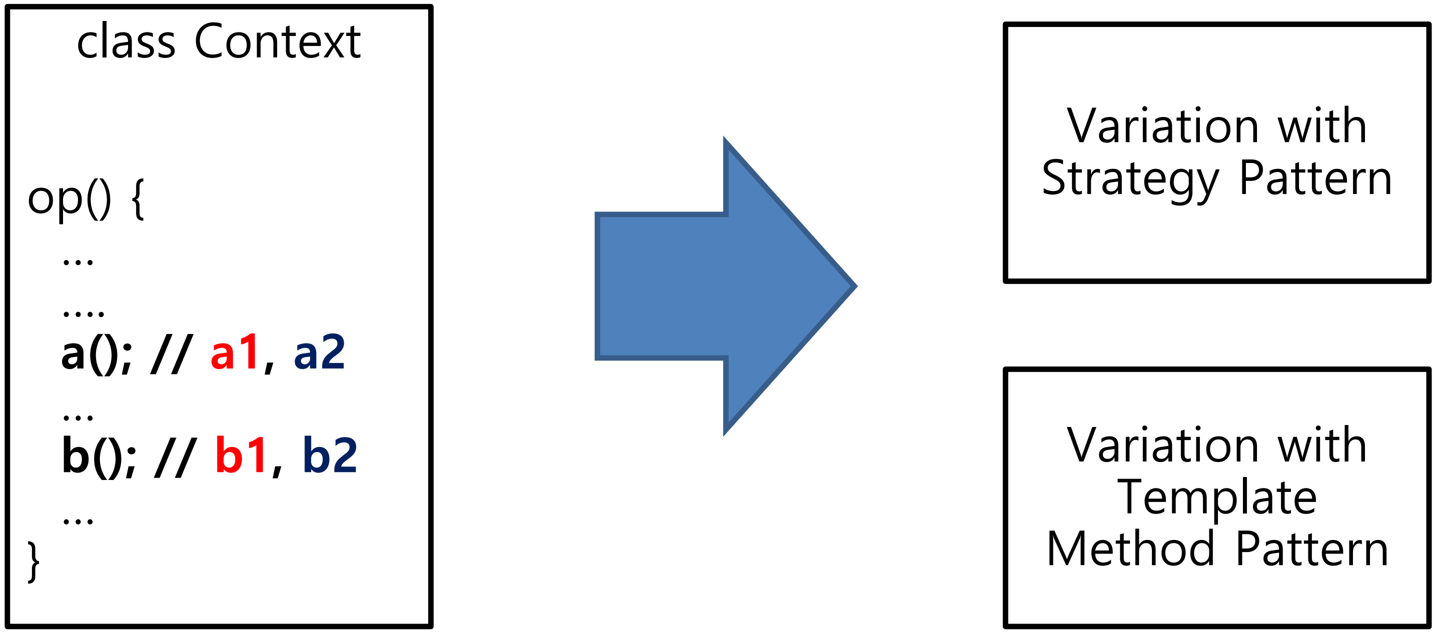
```
고객의 수: 5 명
홍길동: 150
우수한: 350
부족한: 50
훌륭한: 450
최고의: 550
```

Source Code - ComplexReportGenerator

```
public class ComplexReportGenerator extends ReportGenerator {
  protected boolean customerReportCondition(Customer customer) {
    return customer.getPoint() >= 100 ;
  }
  protected String getReportHeader(List<Customer> customers) {
    return String.format("고객의 수: %d 명입니다₩n", customers.size()) ;
  }
  protected String getReportForCustomer(Customer customer) {
    return String.format("%d: %s₩n", customer.getPoint(),
      customer.getName()) ;
  }
  protected String getReportFooter(List<Customer> customers) {
    int totalPoint = 0 ;
    for ( Customer customer: customers)
      totalPoint += customer.getPoint() ;
    return String.format("점수 합계: %d",  totalPoin
  }
}
```

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

Strategy Pattern vs Template Method Pattern



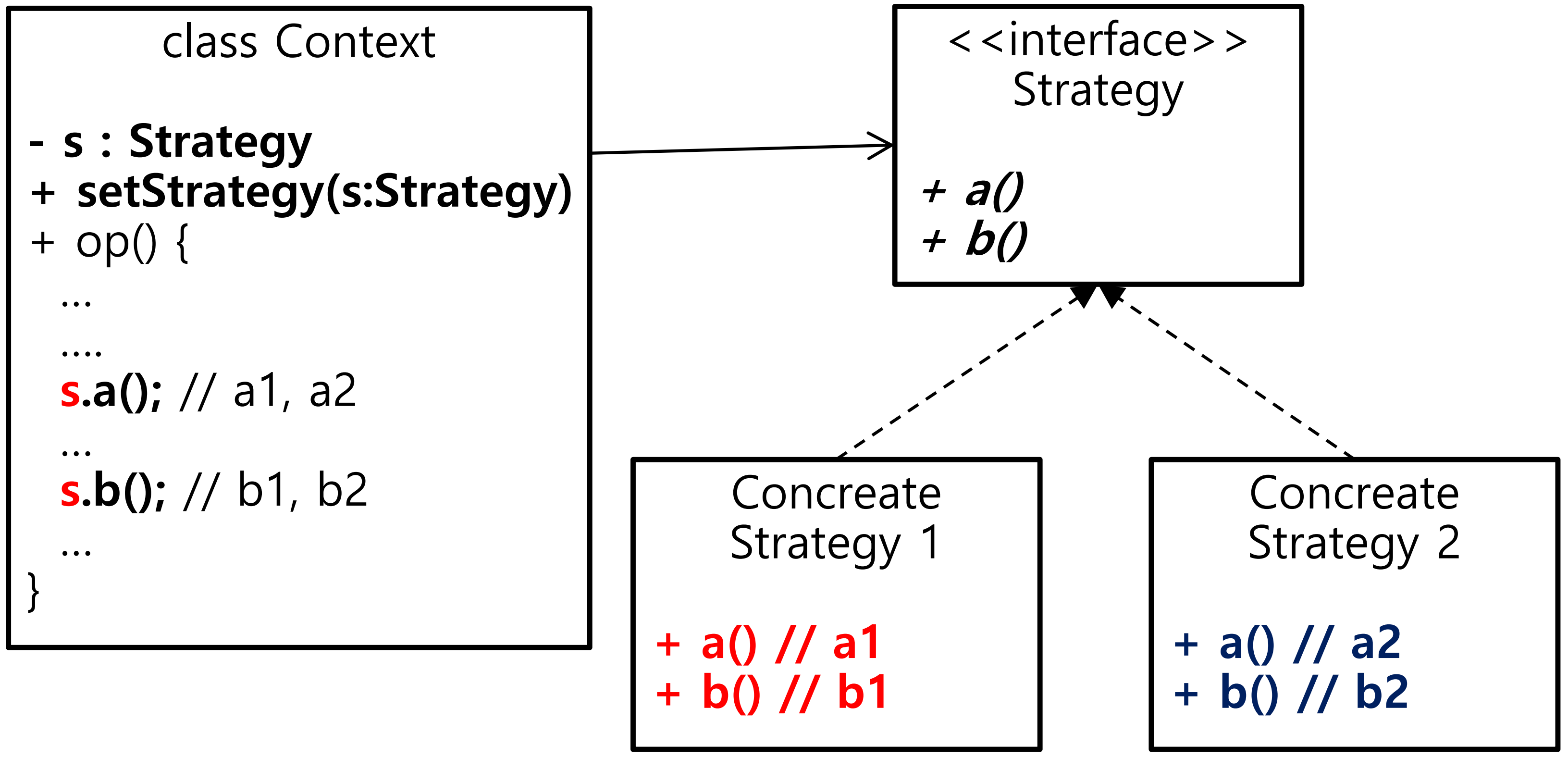
```
class Context

op() {
   …
   ….
   a(); // a1, a2
   …
   b(); // b1, b2
   …
}
```

Variation with Strategy Pattern
Variation with Template Method Pattern

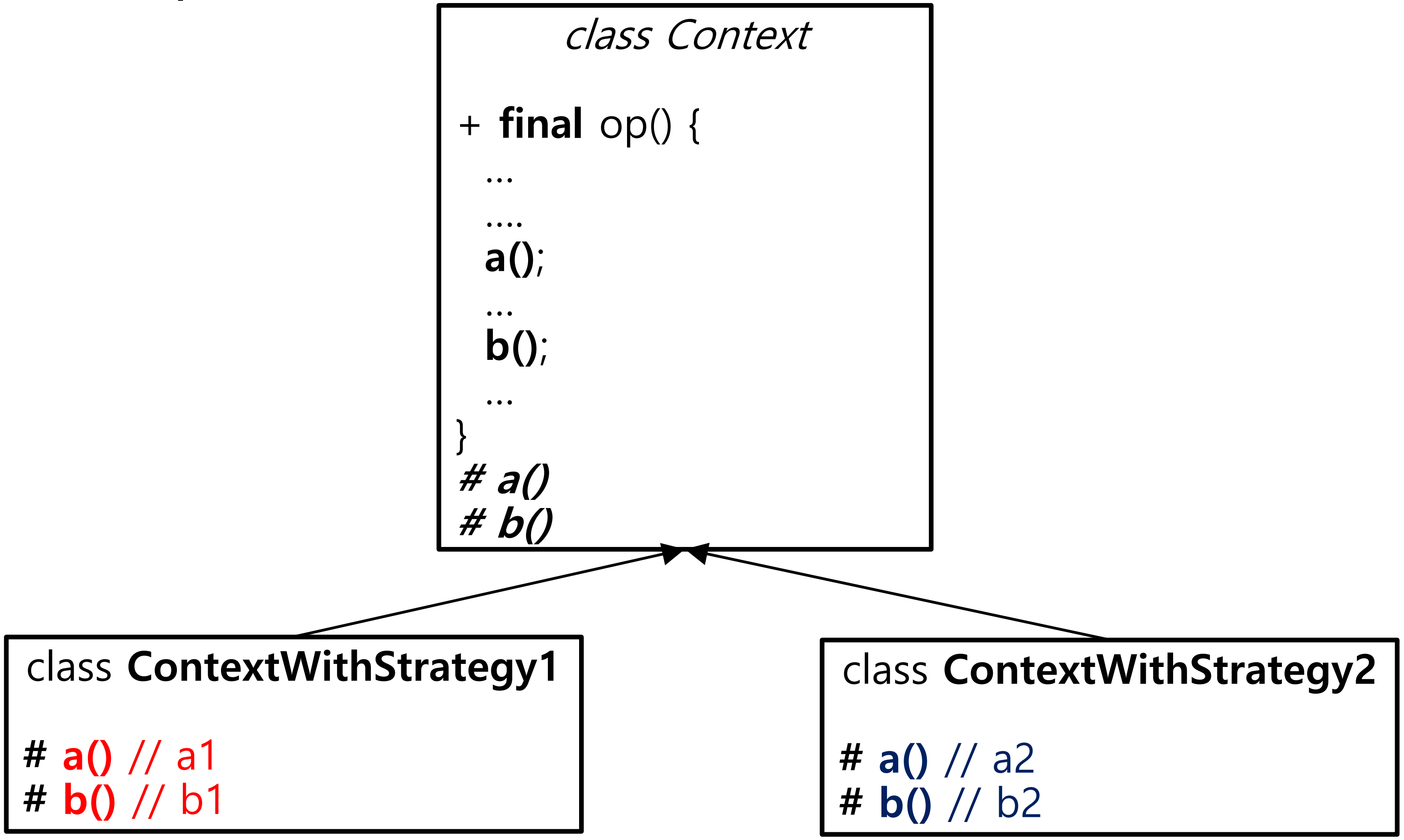
Variation with Strategy Pattern

- Implement the variation with strategies



Variation with Template Method Pattern

- Implement the variation with subclasses



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 Binding Time	Compile time	Run time(context creation time / pure 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