Software Engineering Lab

國立臺灣大學

敦品勵學．愛國愛人

selab.csie.ntu.edu.tw

Chain of Responsibility Pattern

Prof. Jonathan Lee（李允中)

Department of CSIE

National Taiwan University

Design Aspect of Chain of Responsibility

object that can fulfill a request

Outline

- ❑ Email Handler for Enterprise Requirements Statements
- ❑ Initial Design and Its Problems
- ❑ Design Process
- ❑ Refactored Design after Design Process
- ❑ Recurrent Problems
- ❑ Intent
- ❑ Chain of Responsibility Pattern Structure
- ❑ Purchase Request Authorization: Another Example

Email Handler for Enterprise

Prof. Jonathan Lee (李允中)

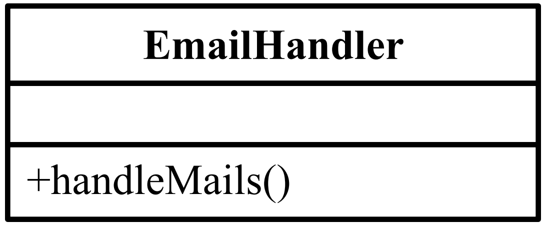
Department of Computer Science and Information Engineering
National Taiwan University

Requirements Statements

- An Email Handler for enterprise has ability to handle all received emails.
- The mail handling process of the Email Handler is as follows:
  - If an email is a spam, it will be put in a spam box.
  - If an email is a complaint mail rather than a spam, it will be forwarded to the legal department.
  - If an email is a fan email, it will be forwarded to the CEO.

Requirements Statements$_1$

- An Email Handler for enterprise has ability to handle all received emails.

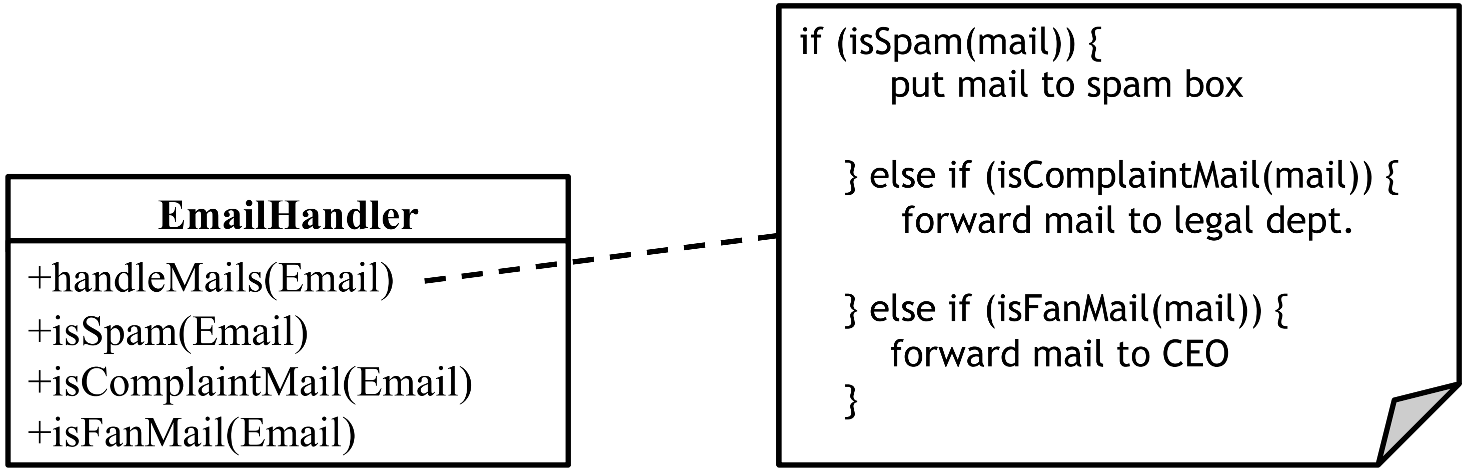


EmailHandler

+handleMails()

Requirements Statements$_2$

- The mail handling process of the Email Handler is as follows:
  - If an email is a spam, it will be put in a spam box.
  - If an email is a complaint mail rather than a spam, it will be forwarded to the legal department.
  - If an email is a fan email, it will be forwarded to the CEO.

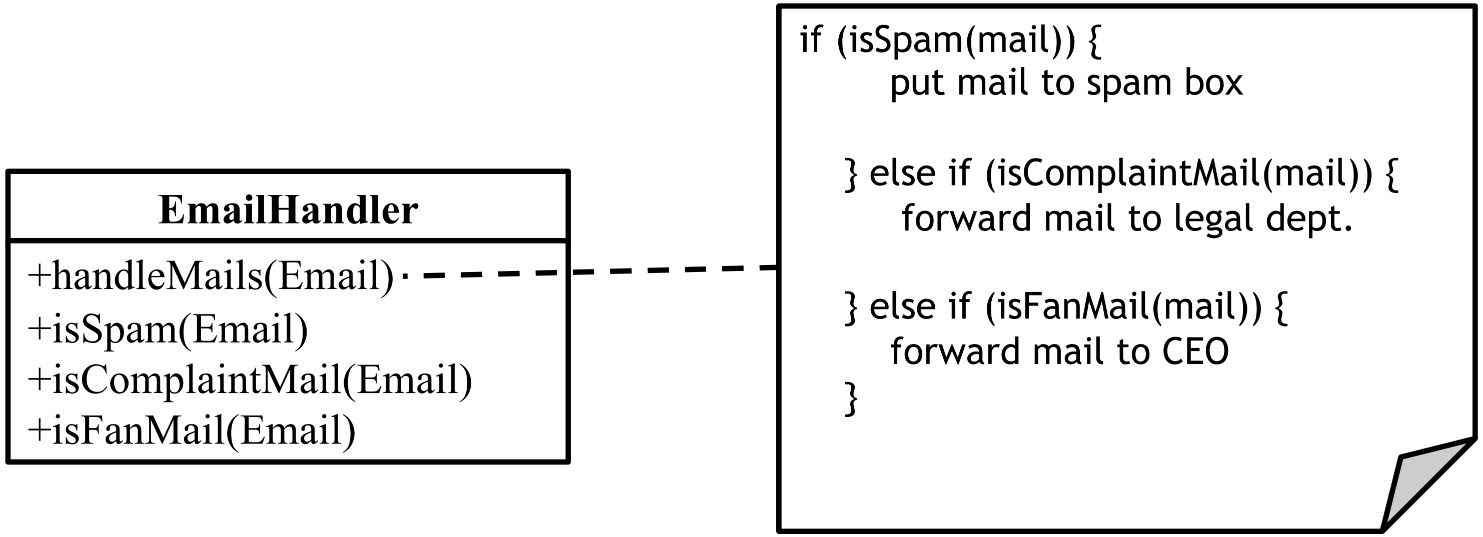


EmailHandler

+handleMails(Email)
+isSpam(Email)
+isComplaintMail(Email)
+isFanMail(Email)

```
if (isSpam(mail)) {
      put mail to spam box

   } else if (isComplaintMail(mail)) {
       forward mail to legal dept.

   } else if (isFanMail(mail)) {
      forward mail to CEO
   }
```

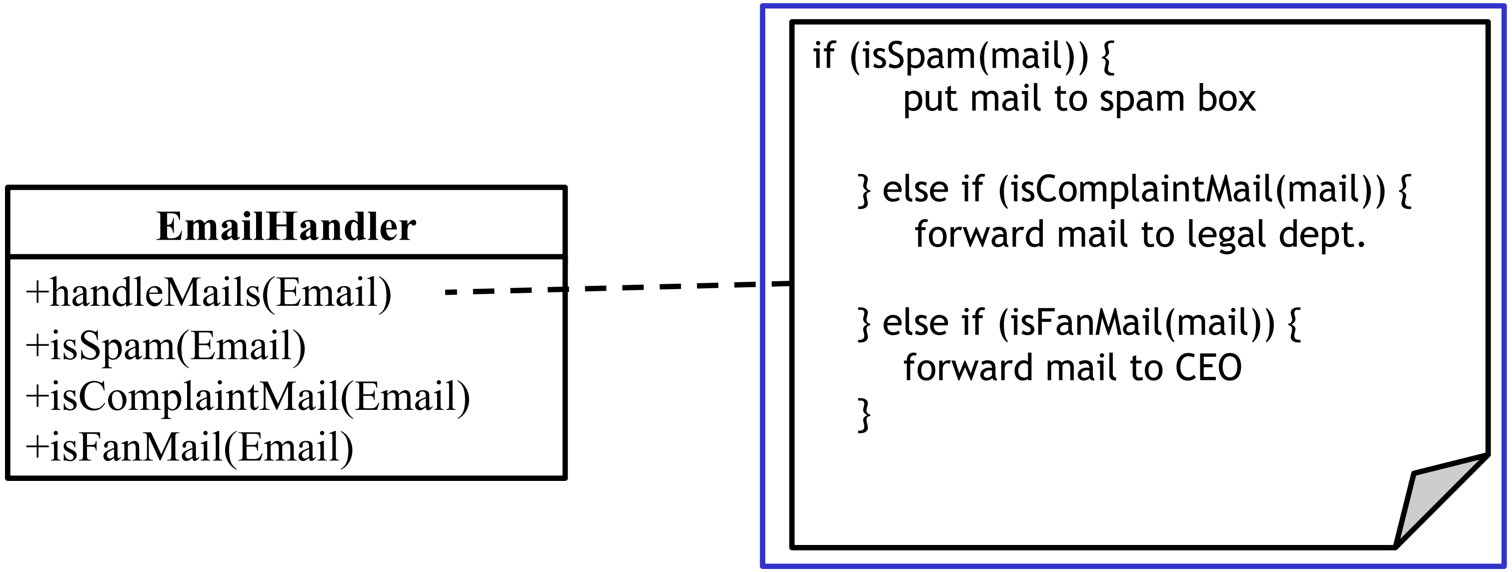
Initial Design - Class Diagram



```
if (isSpam(mail)) {
        put mail to spam box

    } else if (isComplaintMail(mail)) {
         forward mail to legal dept.

    } else if (isFanMail(mail)) {
        forward mail to CEO
    }
```

Problems with Initial Design

Problem: If we’d like to handle new kind of emails, the EmailHandler would be opened and modified.

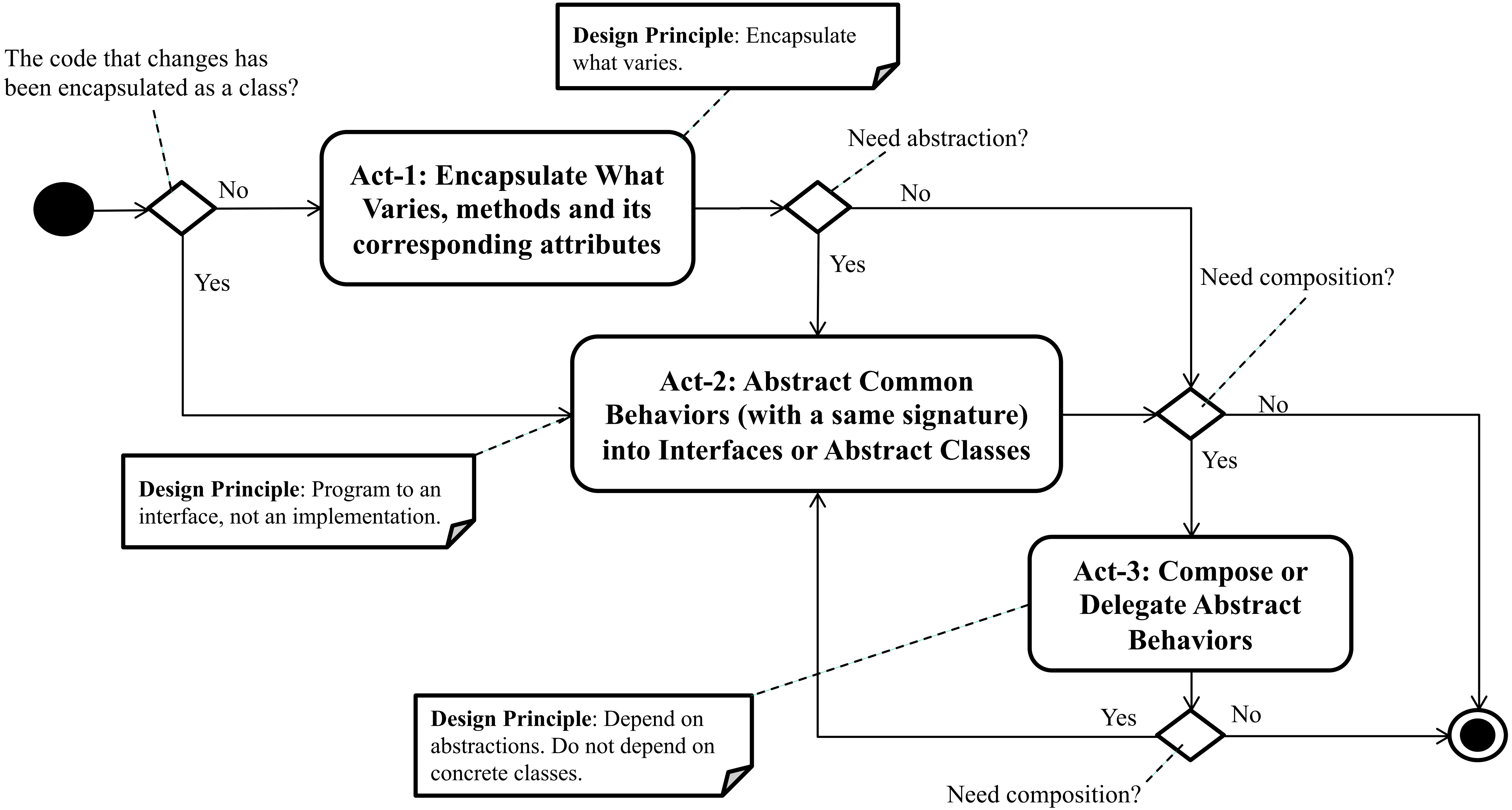


EmailHandler

+handleMails(Email)
+isSpam(Email)
+isComplaintMail(Email)
+isFanMail(Email)

```
if (isSpam(mail)) {
        put mail to spam box

    } else if (isComplaintMail(mail)) {
         forward mail to legal dept.

    } else if (isFanMail(mail)) {
        forward mail to CEO
    }
```

Design Process for Change



The code that changes has been encapsulated as a class?

No

Yes

Act-1: Encapsulate What Varies, methods and its corresponding attributes

Design Principle: Encapsulate what varies.

Need abstraction?

No

Yes

Act-2: Abstract Common Behaviors (with a same signature) into Interfaces or Abstract Classes

Design Principle: Program to an interface, not an implementation.

Need composition?

No

Yes

Act-3: Compose or Delegate Abstract Behaviors

Design Principle: Depend on abstractions. Do not depend on concrete classes.

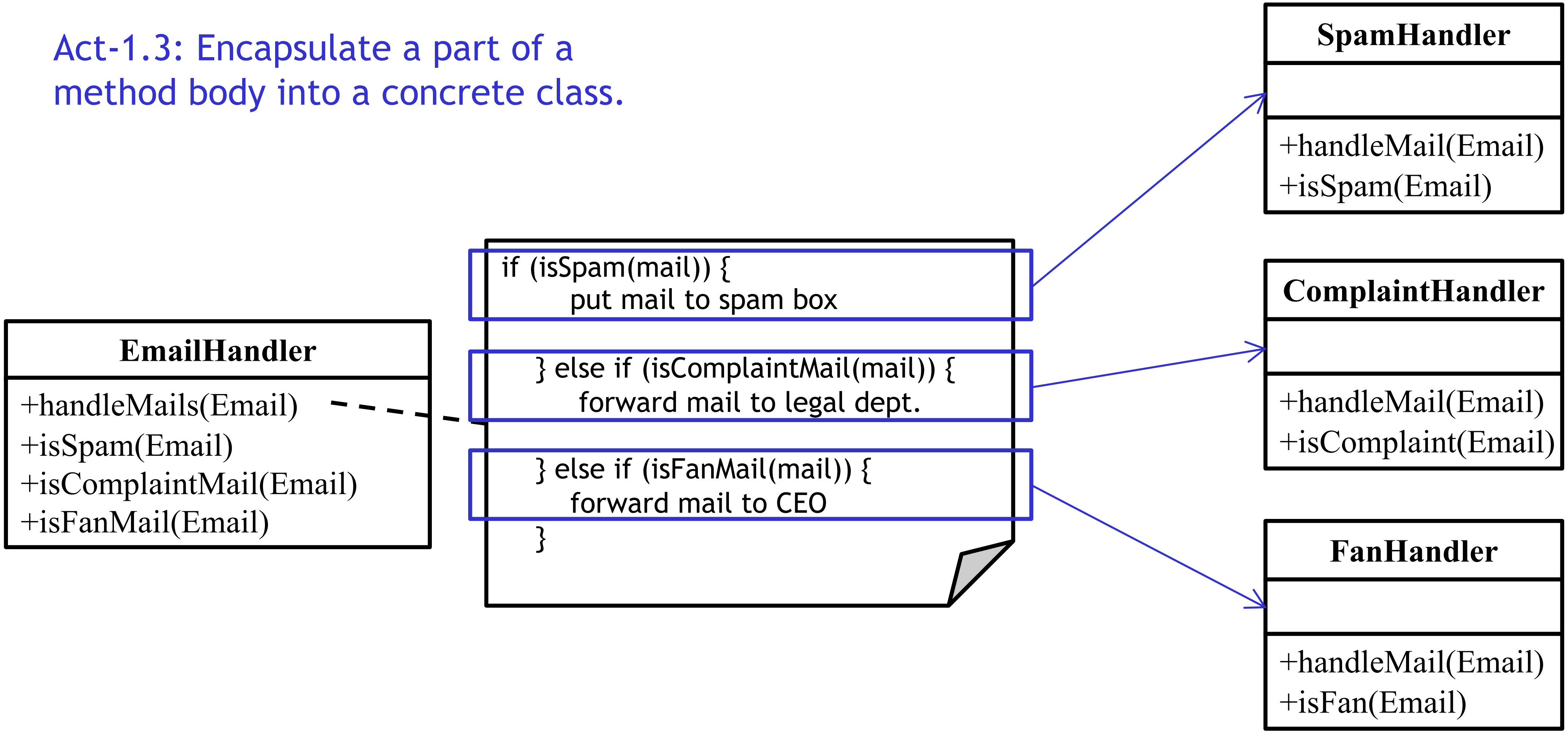
Yes

No

Need composition?

Act-1: Encapsulate What Varies

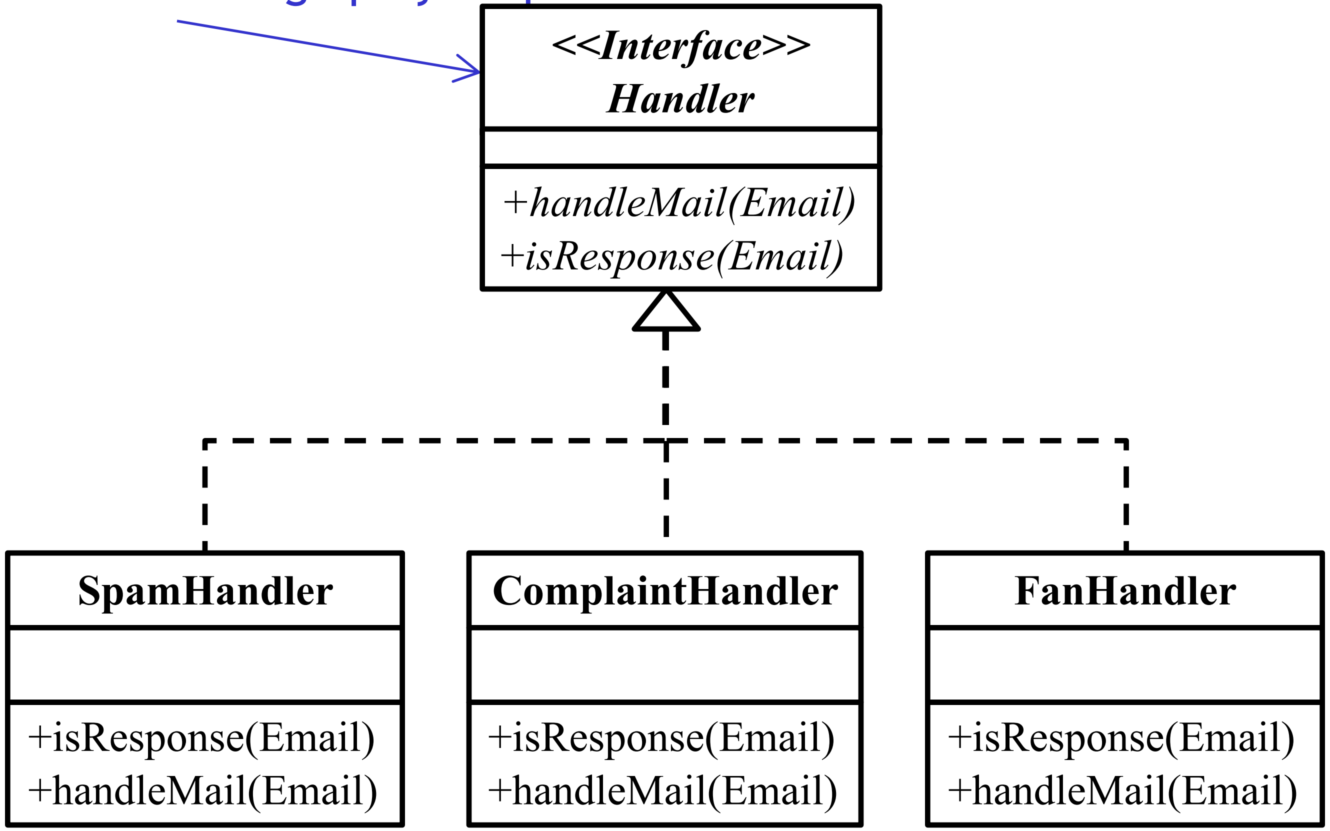
Act-1.3: Encapsulate a part of a method body into a concrete class.



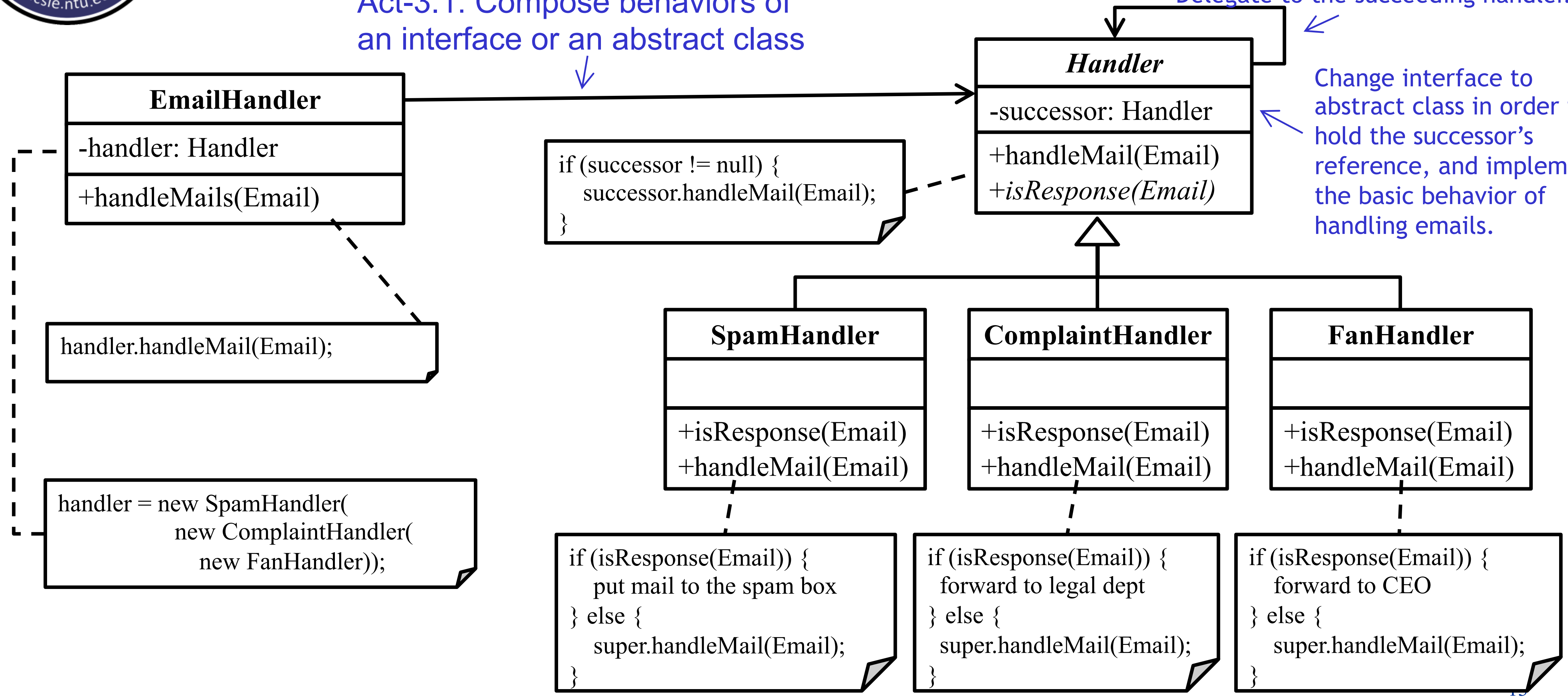
```
if (isSpam(mail)) {
        put mail to spam box
    } else if (isComplaintMail(mail)) {
         forward mail to legal dept.
    } else if (isFanMail(mail)) {
        forward mail to CEO
    }
```

Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



Act-3: Compose Abstract Behaviors



Act-3.1: Compose behaviors of an interface or an abstract class

Delegate to the succeeding handler.

Change interface to abstract class in order to hold the successor’s reference, and implement the basic behavior of handling emails.

```
handler.handleMail(Email);
```

```
handler = new SpamHandler(
              new ComplaintHandler(
                  new FanHandler));
```

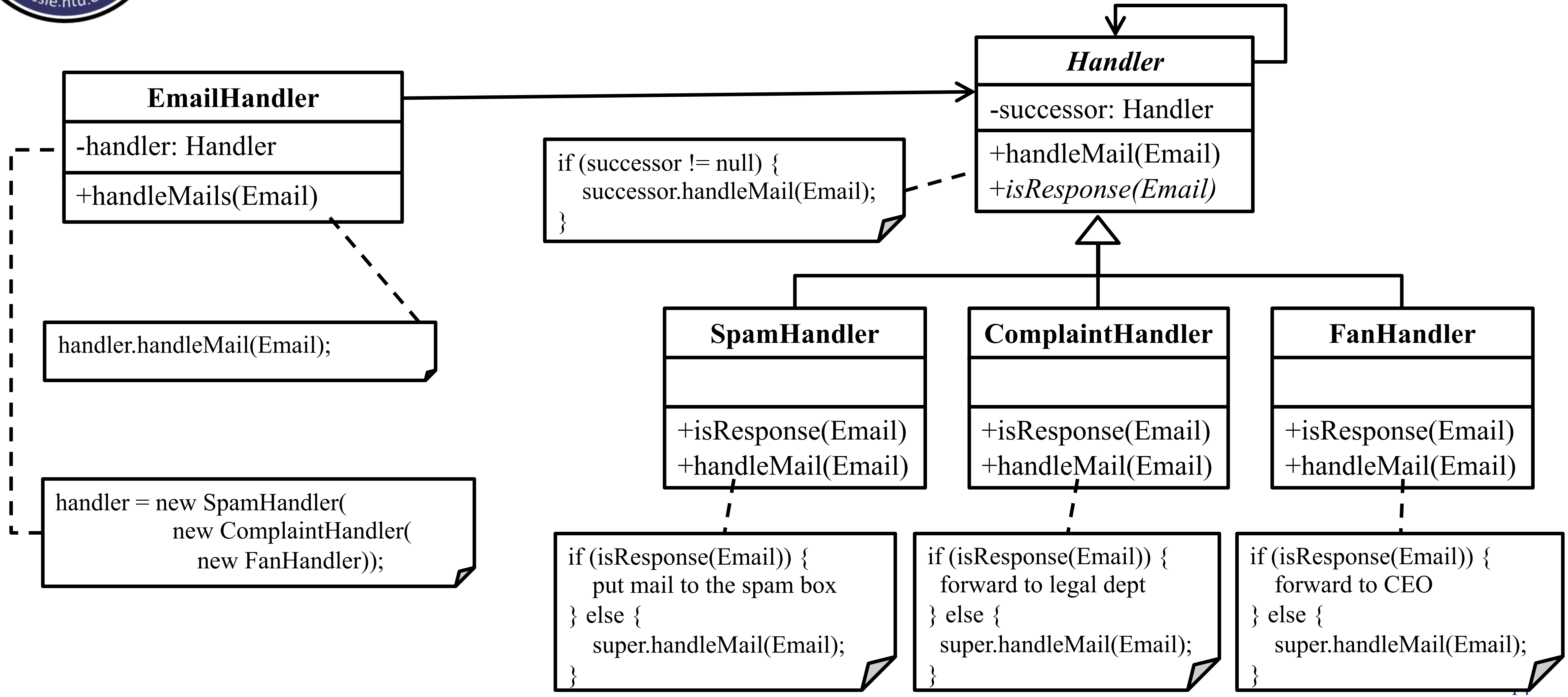
```
if (successor != null) {
   successor.handleMail(Email);
}
```

```
if (isResponse(Email)) {
   put mail to the spam box
} else {
   super.handleMail(Email);
}
```

```
if (isResponse(Email)) {
  forward to legal dept
} else {
  super.handleMail(Email);
}
```

```
if (isResponse(Email)) {
   forward to CEO
} else {
   super.handleMail(Email);
}
```

Refactored Design after Design Process



```
handler.handleMail(Email);
```

```
handler = new SpamHandler(
              new ComplaintHandler(
                new FanHandler));
```

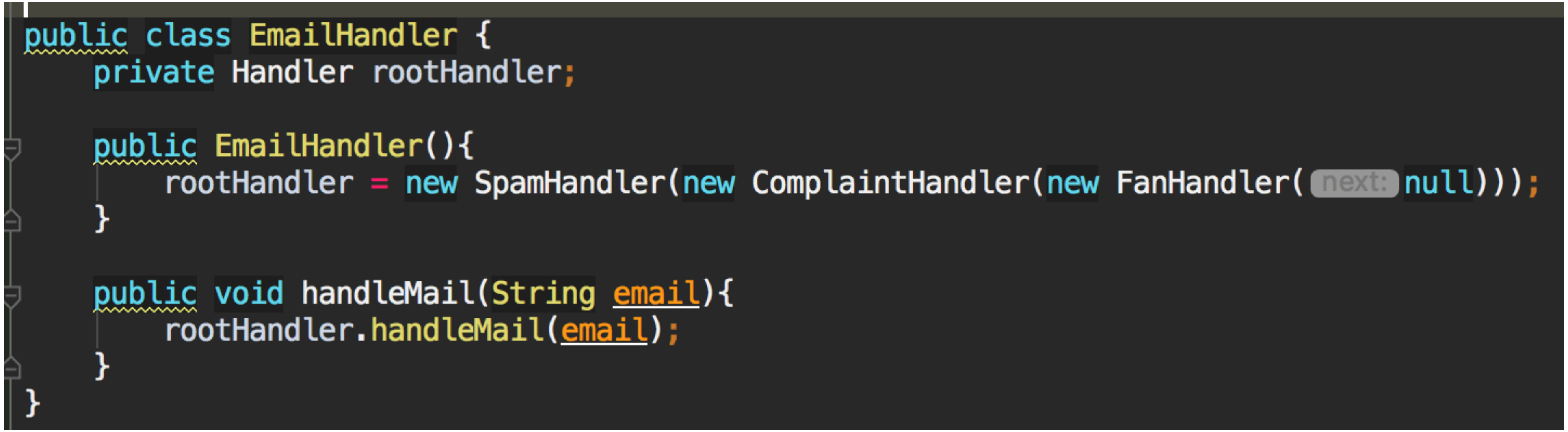
```
if (successor != null) {
    successor.handleMail(Email);
}
```

```
if (isResponse(Email)) {
    put mail to the spam box
} else {
    super.handleMail(Email);
}
```

```
if (isResponse(Email)) {
  forward to legal dept
} else {
  super.handleMail(Email);
}
```

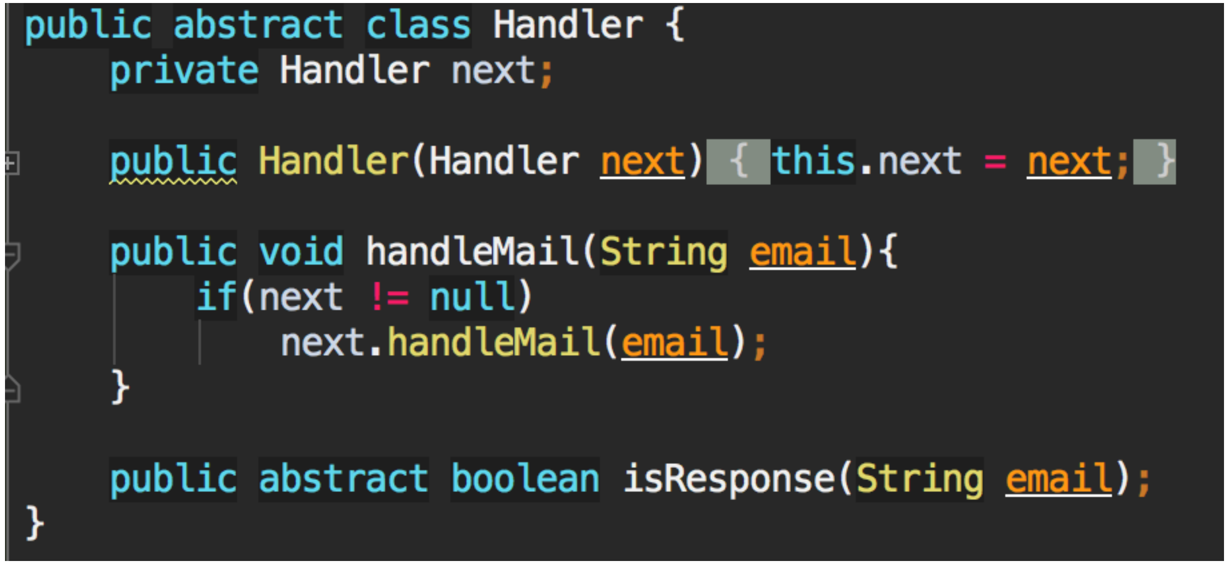
```
if (isResponse(Email)) {
    forward to CEO
} else {
    super.handleMail(Email);
}
```

EmailHandler



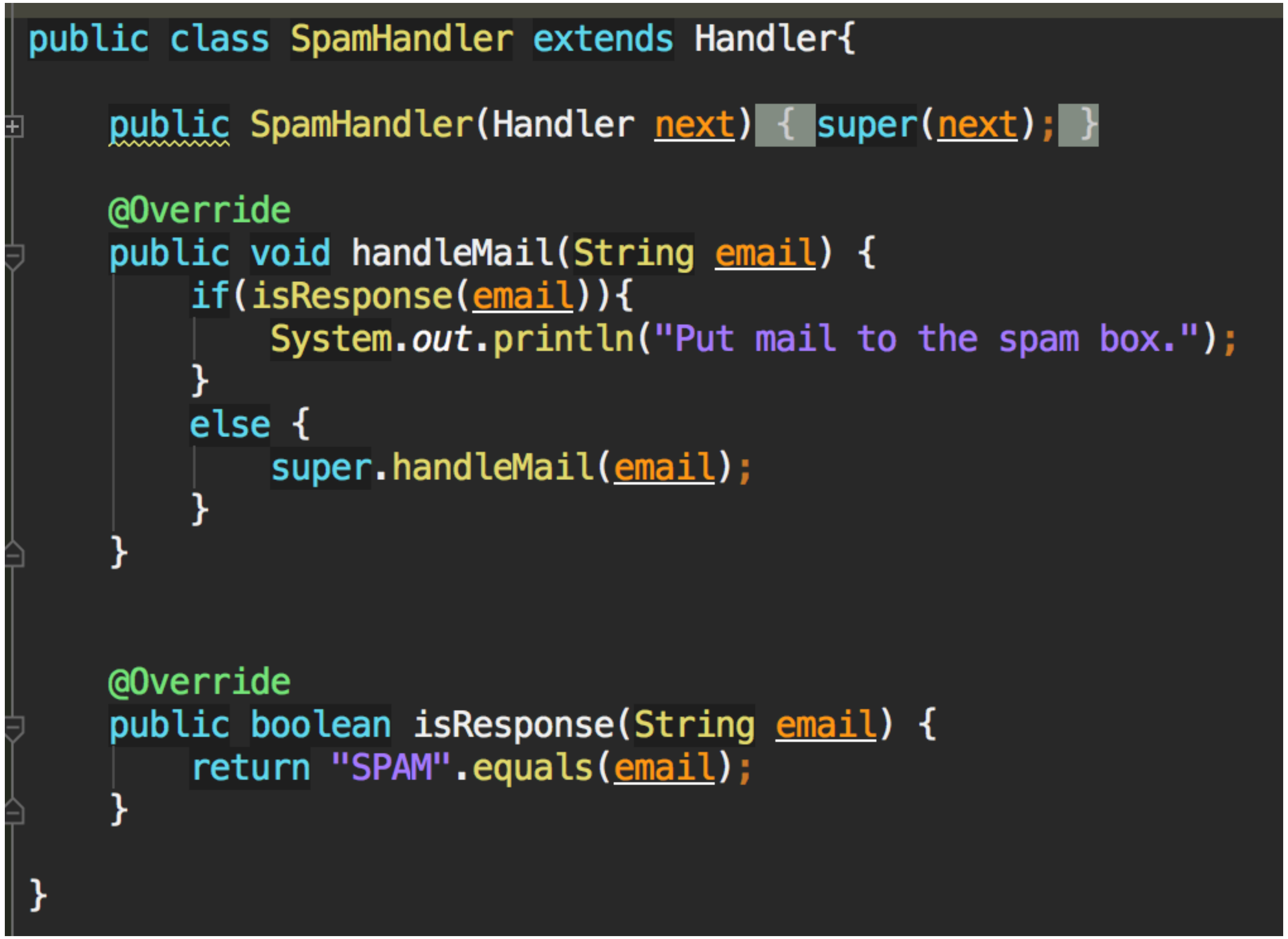
```
public class EmailHandler {
    private Handler rootHandler;

    public EmailHandler(){
        rootHandler = new SpamHandler(new ComplaintHandler(new FanHandler( next: null)));
    }

    public void handleMail(String email){
        rootHandler.handleMail(email);
    }
}
```

Handler



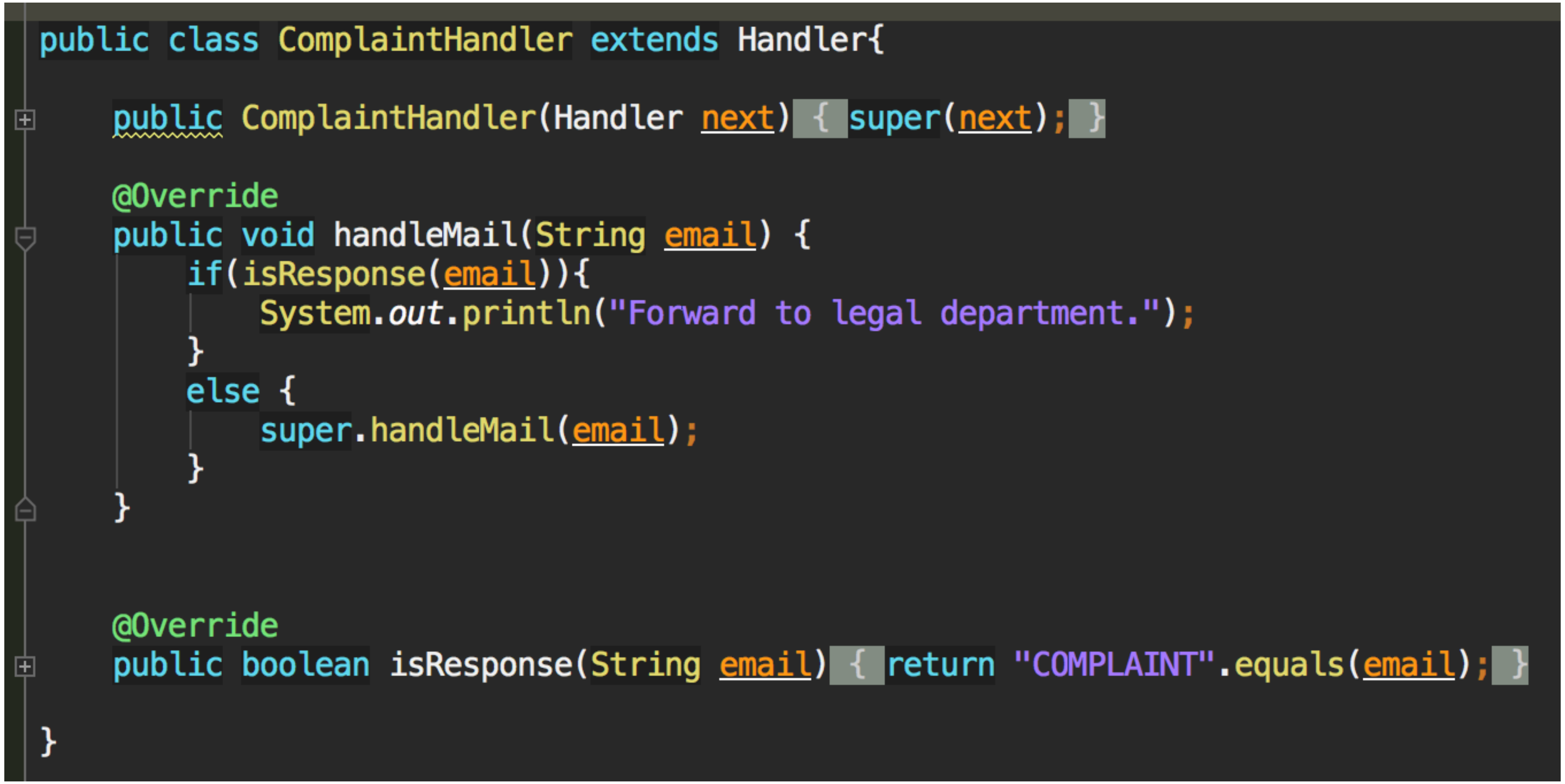
```java
public abstract class Handler {
    private Handler next;

    public Handler(Handler next) { this.next = next; }

    public void handleMail(String email){
        if(next != null)
            next.handleMail(email);
    }

    public abstract boolean isResponse(String email);
}
```

SpamHandler



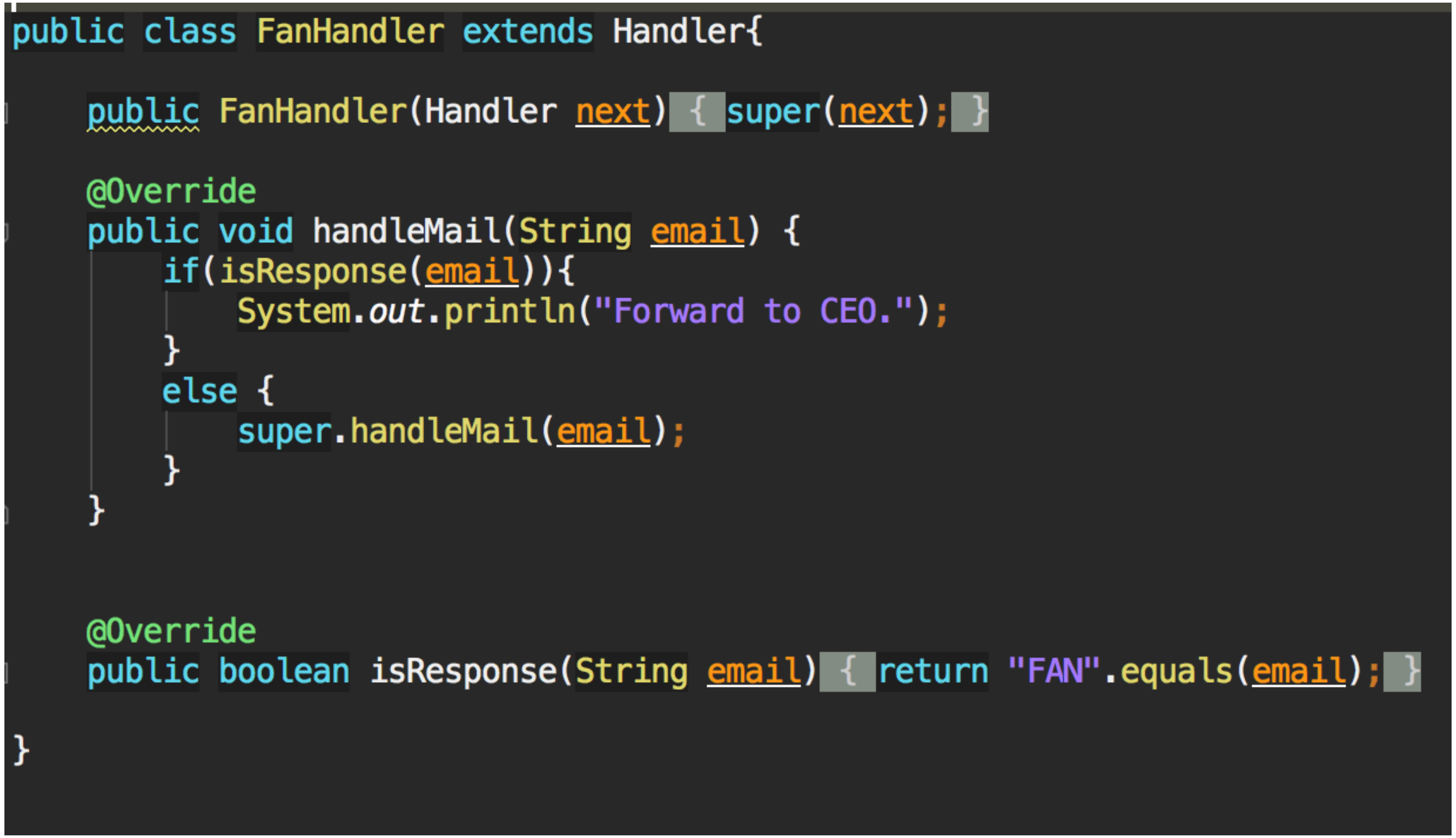
```
public class SpamHandler extends Handler{

    public SpamHandler(Handler next) { super(next); }

    @Override
    public void handleMail(String email) {
        if(isResponse(email)){
            System.out.println("Put mail to the spam box.");
        }
        else {
            super.handleMail(email);
        }
    }


    @Override
    public boolean isResponse(String email) {
        return "SPAM".equals(email);
    }

}
```

ComplaintHandler



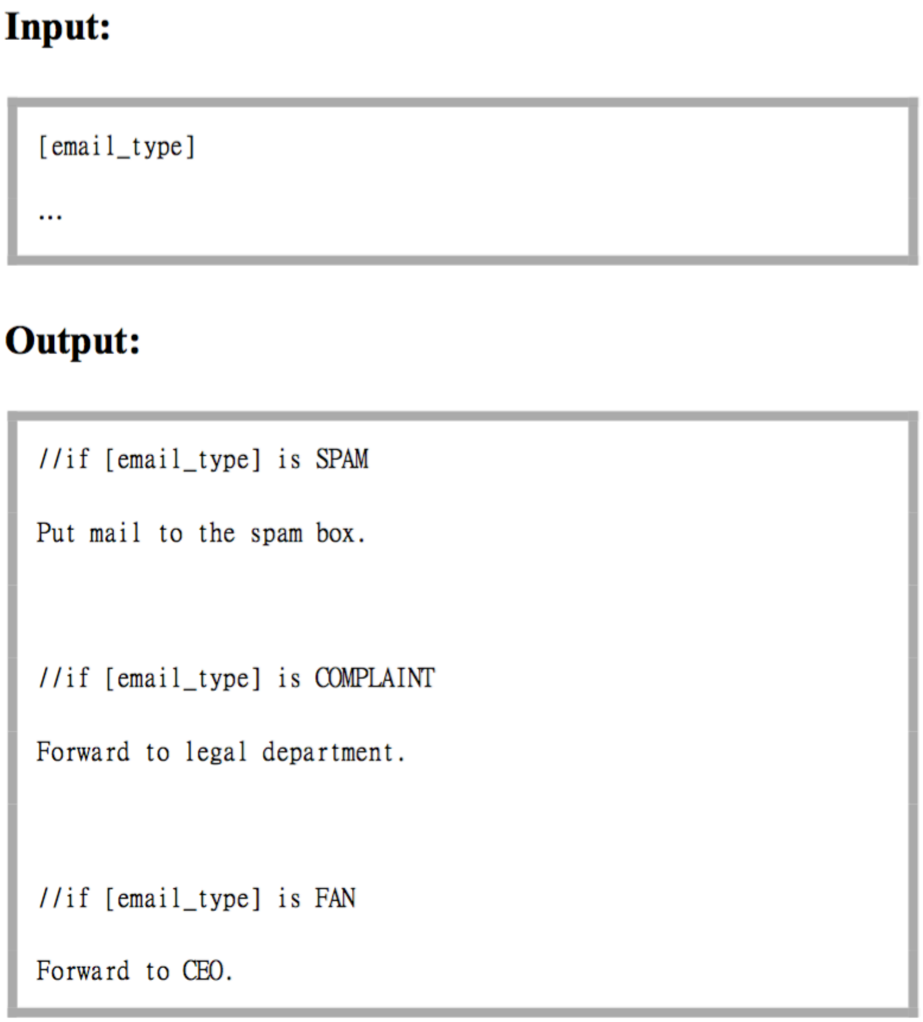
```
public class ComplaintHandler extends Handler{

    public ComplaintHandler(Handler next) { super(next); }

    @Override
    public void handleMail(String email) {
        if(isResponse(email)){
            System.out.println("Forward to legal department.");
        }
        else {
            super.handleMail(email);
        }
    }


    @Override
    public boolean isResponse(String email) { return "COMPLAINT".equals(email); }

}
```

FanHandler



```java
public class FanHandler extends Handler{

    public FanHandler(Handler next) { super(next); }

    @Override
    public void handleMail(String email) {
        if(isResponse(email)){
            System.out.println("Forward to CEO.");
        }
        else {
            super.handleMail(email);
        }
    }


    @Override
    public boolean isResponse(String email) { return "FAN".equals(email); }

}
```

Input / Output format



Input:

```
[email_type]

...
```

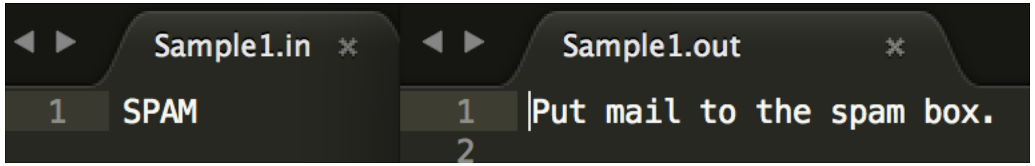
Output:

```
//if [email_type] is SPAM

Put mail to the spam box.


//if [email_type] is COMPLAINT

Forward to legal department.


//if [email_type] is FAN

Forward to CEO.
```

Test cases

- TestCase1: SPAM
- TestCase2: COMPALINT
- TestCase3: FAN
- TestCase 4: Complex

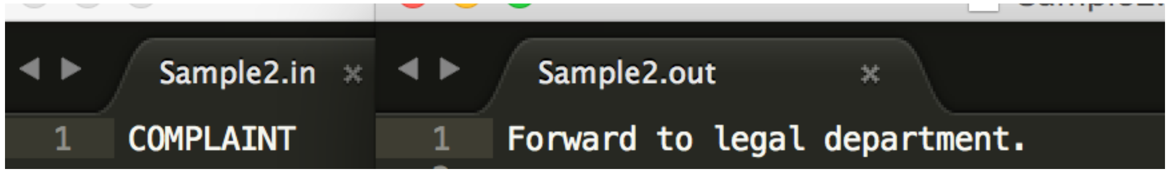
Test case1



```
SPAM
```

```
Put mail to the spam box.
```

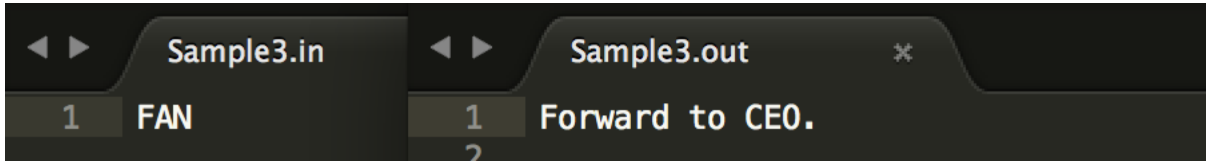
Test case2



```
COMPLAINT
```

```
Forward to legal department.
```

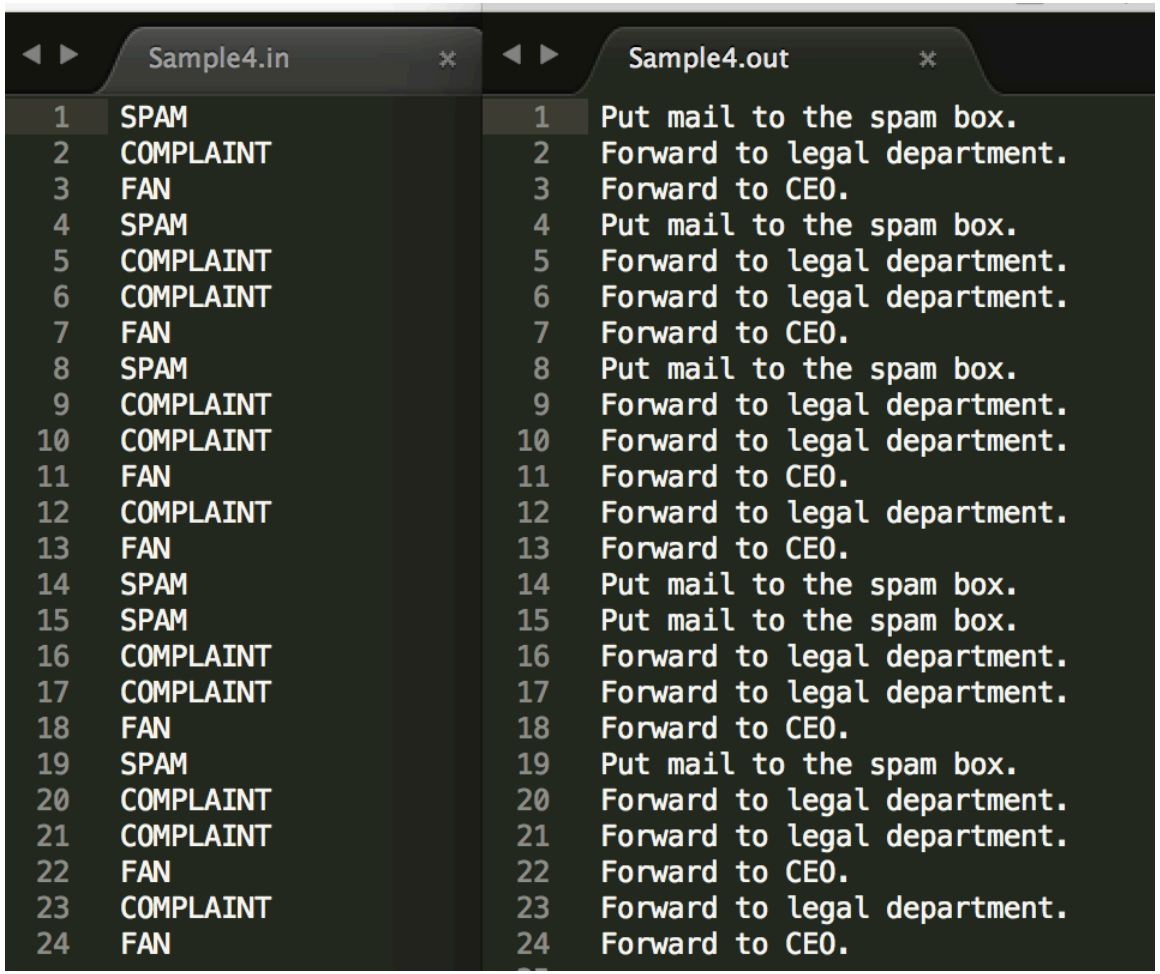
Test case3



```
FAN
```

```
Forward to CEO.
```

Test case4



Sample4.in

```
SPAM
COMPLAINT
FAN
SPAM
COMPLAINT
COMPLAINT
FAN
SPAM
COMPLAINT
COMPLAINT
FAN
COMPLAINT
FAN
SPAM
SPAM
COMPLAINT
COMPLAINT
FAN
SPAM
COMPLAINT
COMPLAINT
FAN
COMPLAINT
FAN
```

Sample4.out

```
Put mail to the spam box.
Forward to legal department.
Forward to CEO.
Put mail to the spam box.
Forward to legal department.
Forward to legal department.
Forward to CEO.
Put mail to the spam box.
Forward to legal department.
Forward to legal department.
Forward to CEO.
Forward to legal department.
Forward to CEO.
Put mail to the spam box.
Put mail to the spam box.
Forward to legal department.
Forward to legal department.
Forward to CEO.
Put mail to the spam box.
Forward to legal department.
Forward to legal department.
Forward to CEO.
Forward to legal department.
Forward to CEO.
```

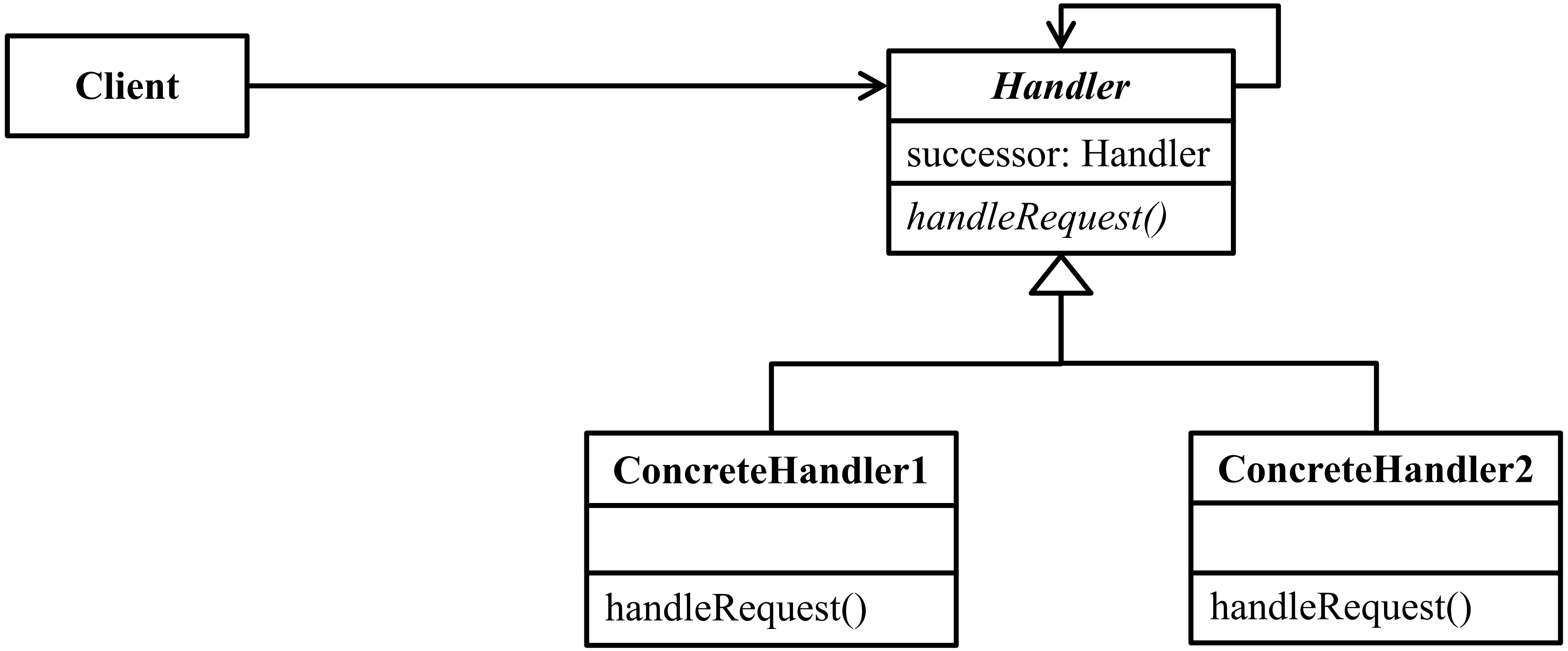
Recurrent Problems

- ❑ More than one object may handle a request, and the handler isn't known a priori. The handler should be ascertained automatically.
- ❑ You want to issue a request to one of several objects without specifying the receiver explicitly.
- ❑ The set of objects that can handle a request should be specified dynamically.

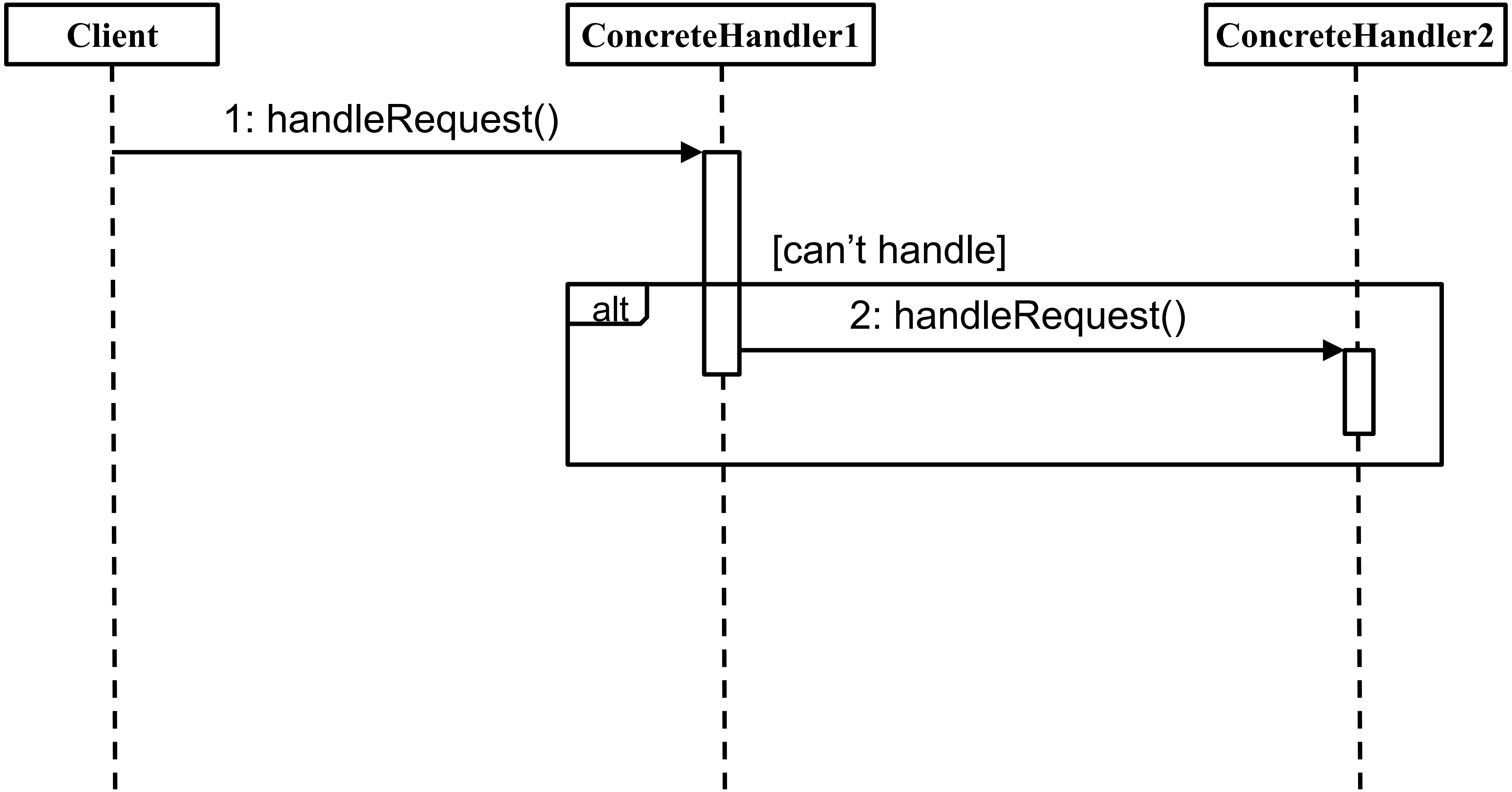
Intent

- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Chain of Responsibility Pattern Structure$_1$



Chain of Responsibility Pattern Structure_2



Chain of Responsibility Pattern Structure$_3$

	Instantiation	Use	Termination
Handler	X	Client sends request to Handler, and a ConcreteHandler handles the request through polymorphism.	X
ConcreteHandler	Don’t Care	If a ConcreteHandler isn’t able to handle a request, it passes the request to its successor, another Handler, if it has one.	Don’t Care

Purchase Request Authorization

Prof. Jonathan Lee (李允中)

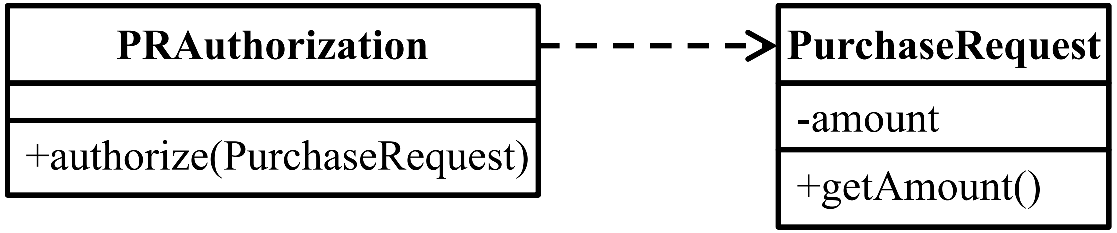
Department of Computer Science and Information Engineering
National Taiwan University

Requirements Statements

- In a typical organization, Purchase Request Authorization is responsible to authorize purchase requests by dispatching requests to appropriate management representatives according to the amount.
- Different management representative has different authorization limit to authorize purchase requests as follows:
  - Mgmt. Level: Branch Manager / Authorization Limit: $25,000
  - Mgmt. Level: Regional Director / Authorization Limit: $100,000
  - Mgmt. Level: Vice President / Authorization Limit: $200,000
  - Mgmt. Level: President / Authorization Limit: $400,000

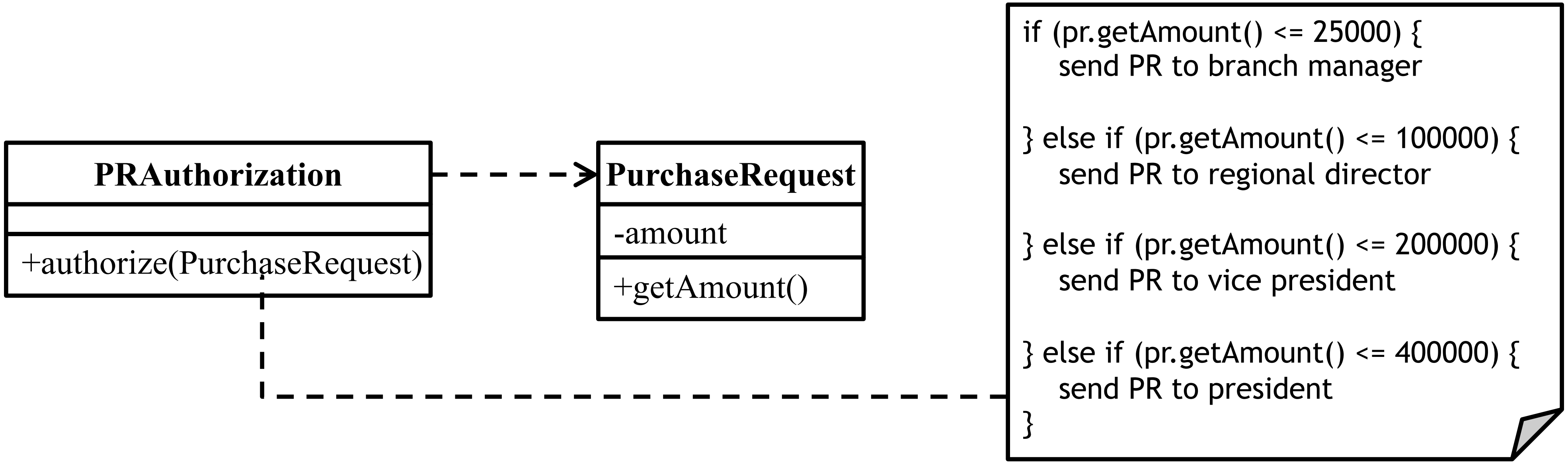
Requirements Statements$_1$

- In a typical organization, Purchase Request Authorization is responsible to authorize purchase requests by dispatching requests to appropriate management representatives according to the amount.



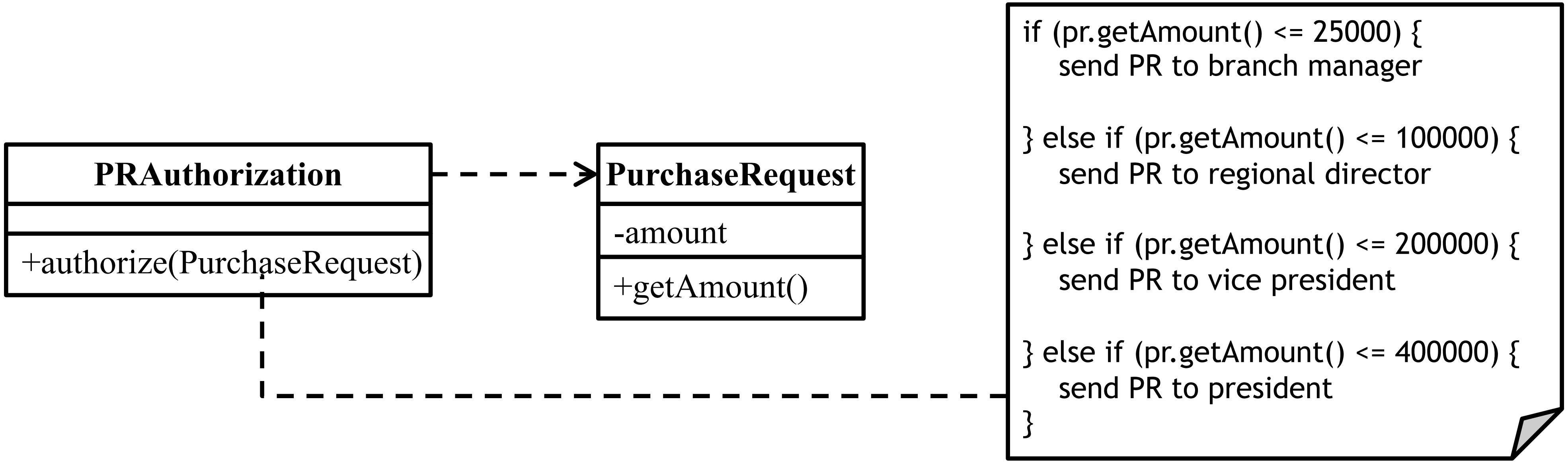
Requirements Statements$_2$

- Different management representative has different authorization limit to authorize purchase requests as follows:
  - Mgmt. Level: Branch Manager / Authorization Limit: $25,000
  - Mgmt. Level: Regional Director / Authorization Limit: $100,000
  - Mgmt. Level: Vice President / Authorization Limit: $200,000
  - Mgmt. Level: President / Authorization Limit: $400,000



```
if (pr.getAmount() <= 25000) {
    send PR to branch manager

} else if (pr.getAmount() <= 100000) {
    send PR to regional director

} else if (pr.getAmount() <= 200000) {
    send PR to vice president

} else if (pr.getAmount() <= 400000) {
    send PR to president
}
```

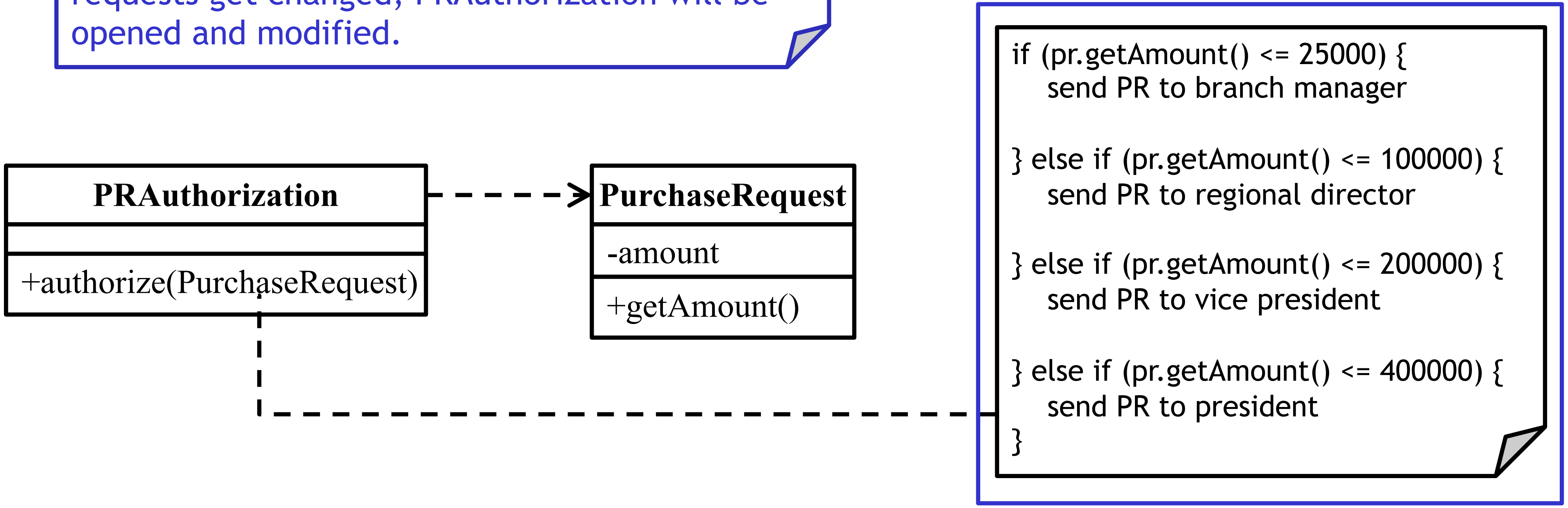
Initial Design - Class Diagram



```
if (pr.getAmount() <= 25000) {
    send PR to branch manager

} else if (pr.getAmount() <= 100000) {
    send PR to regional director

} else if (pr.getAmount() <= 200000) {
    send PR to vice president

} else if (pr.getAmount() <= 400000) {
    send PR to president
}
```

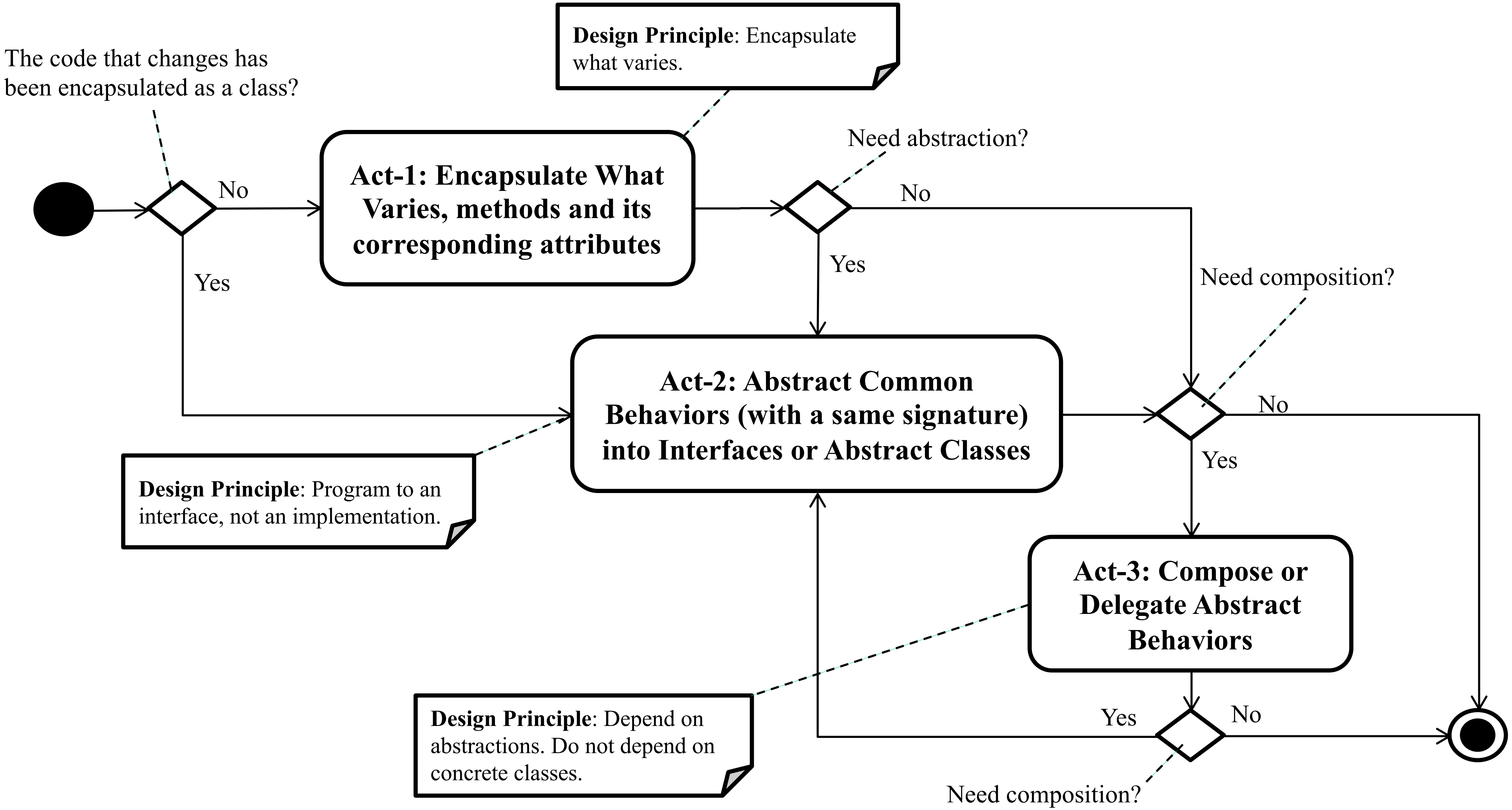
Problems with Initial Design

Problem: Once the rules of dispatching purchase requests get changed, PRAuthorization will be opened and modified.



```
if (pr.getAmount() <= 25000) {
   send PR to branch manager

} else if (pr.getAmount() <= 100000) {
   send PR to regional director

} else if (pr.getAmount() <= 200000) {
   send PR to vice president

} else if (pr.getAmount() <= 400000) {
   send PR to president
}
```

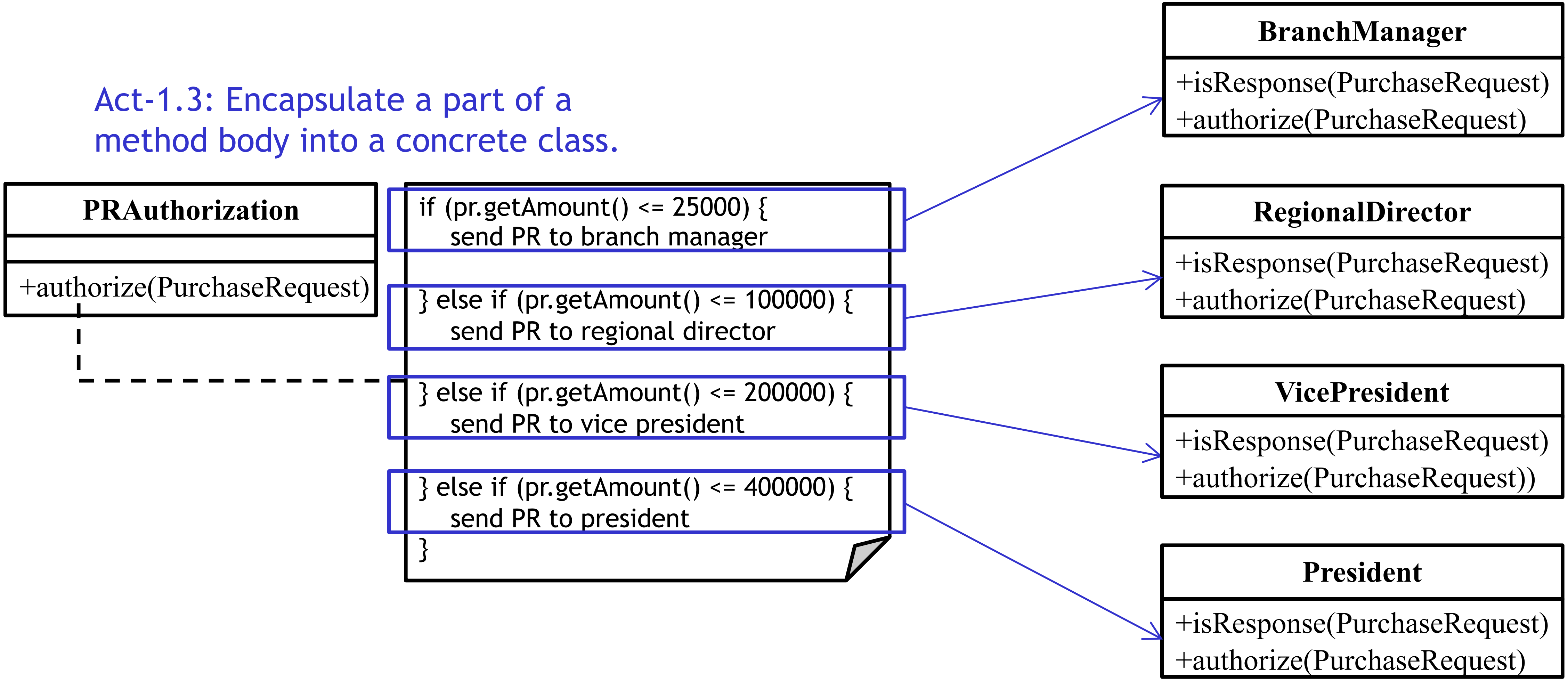
Design Process for Change



The code that changes has been encapsulated as a class?
Design Principle: Encapsulate what varies.
No
Act-1: Encapsulate What Varies, methods and its corresponding attributes
Yes
Need abstraction?
No
Yes
Need composition?
Act-2: Abstract Common Behaviors (with a same signature) into Interfaces or Abstract Classes
Design Principle: Program to an interface, not an implementation.
No
Yes
Act-3: Compose or Delegate Abstract Behaviors
Design Principle: Depend on abstractions. Do not depend on concrete classes.
Yes
No
Need composition?

Act-1: Encapsulate What Varies

Act-1.3: Encapsulate a part of a method body into a concrete class.



PRAuthorization

+authorize(PurchaseRequest)

```
if (pr.getAmount() <= 25000) {
    send PR to branch manager
} else if (pr.getAmount() <= 100000) {
    send PR to regional director
} else if (pr.getAmount() <= 200000) {
    send PR to vice president
} else if (pr.getAmount() <= 400000) {
    send PR to president
}
```

BranchManager

+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)

RegionalDirector

+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)

VicePresident

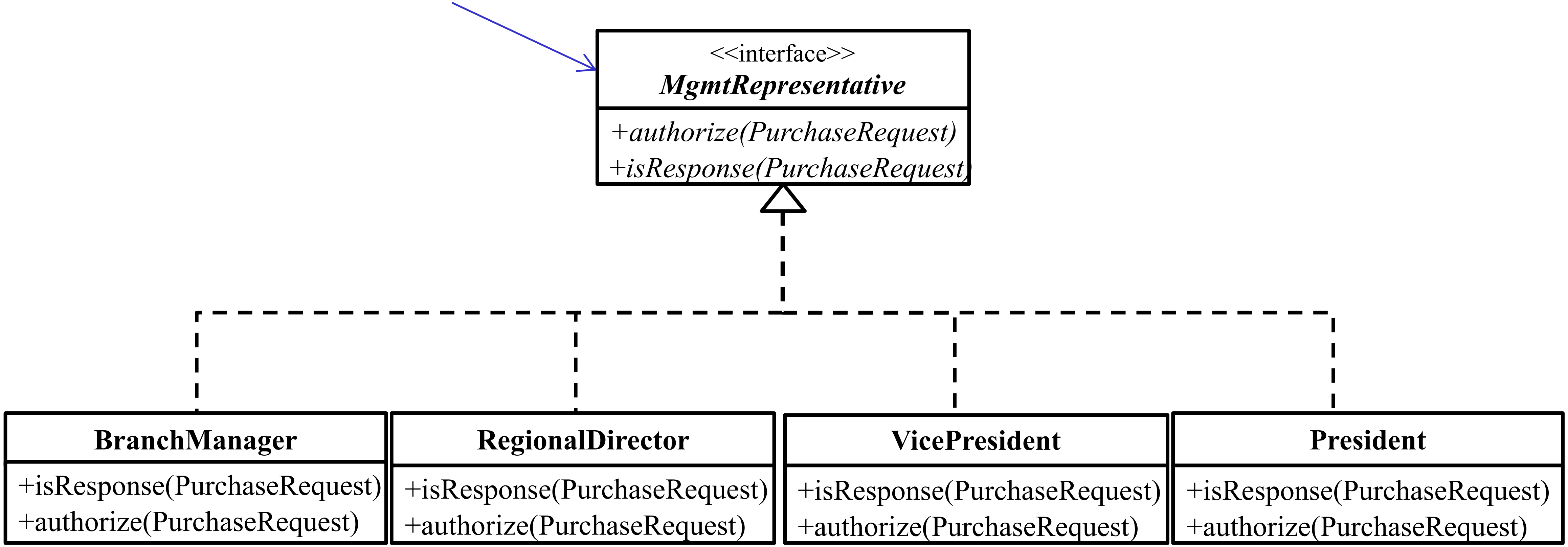
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest))

President

+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)

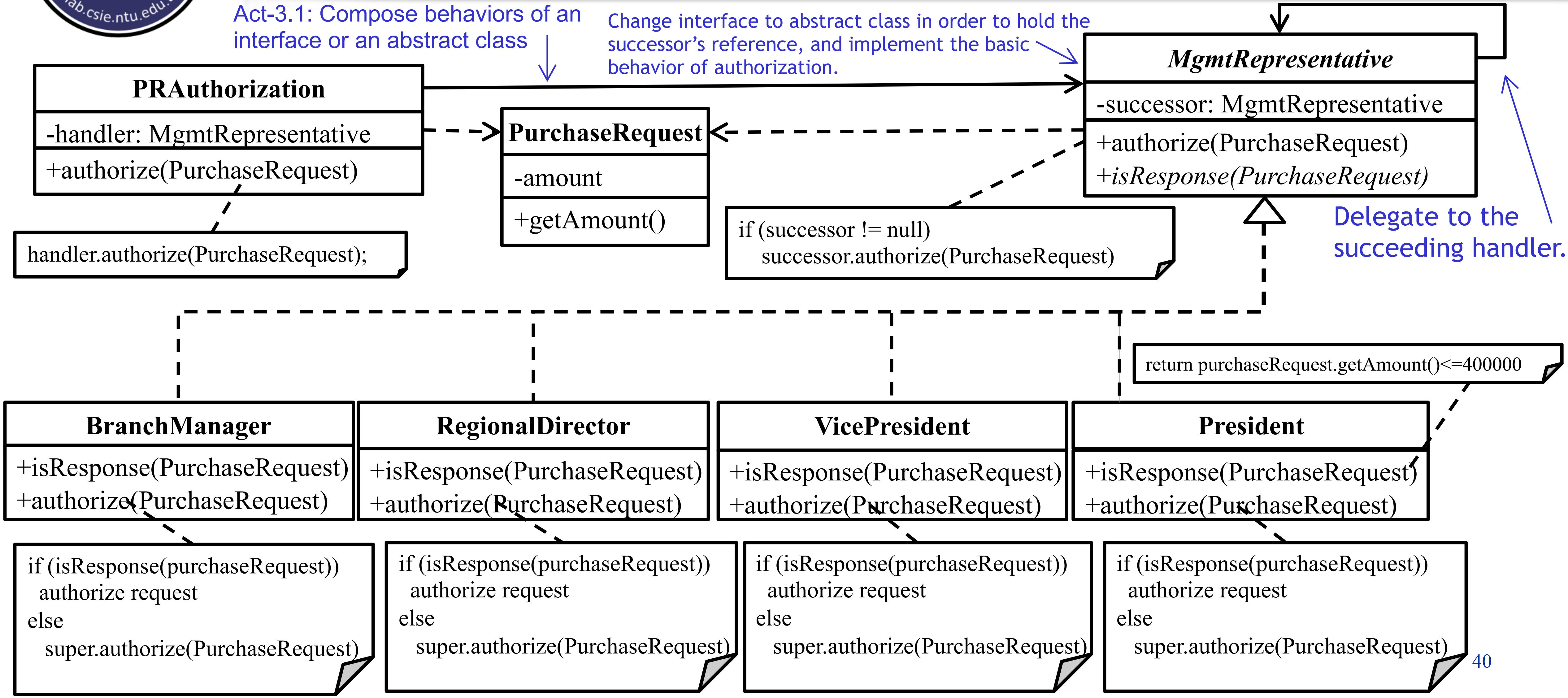
Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



<<interface>>
MgmtRepresentative
+authorize(PurchaseRequest)
+isResponse(PurchaseRequest)
BranchManager
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
RegionalDirector
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
VicePresident
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
President
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)

Act-3: Compose Abstract Behaviors



Act-3.1: Compose behaviors of an interface or an abstract class

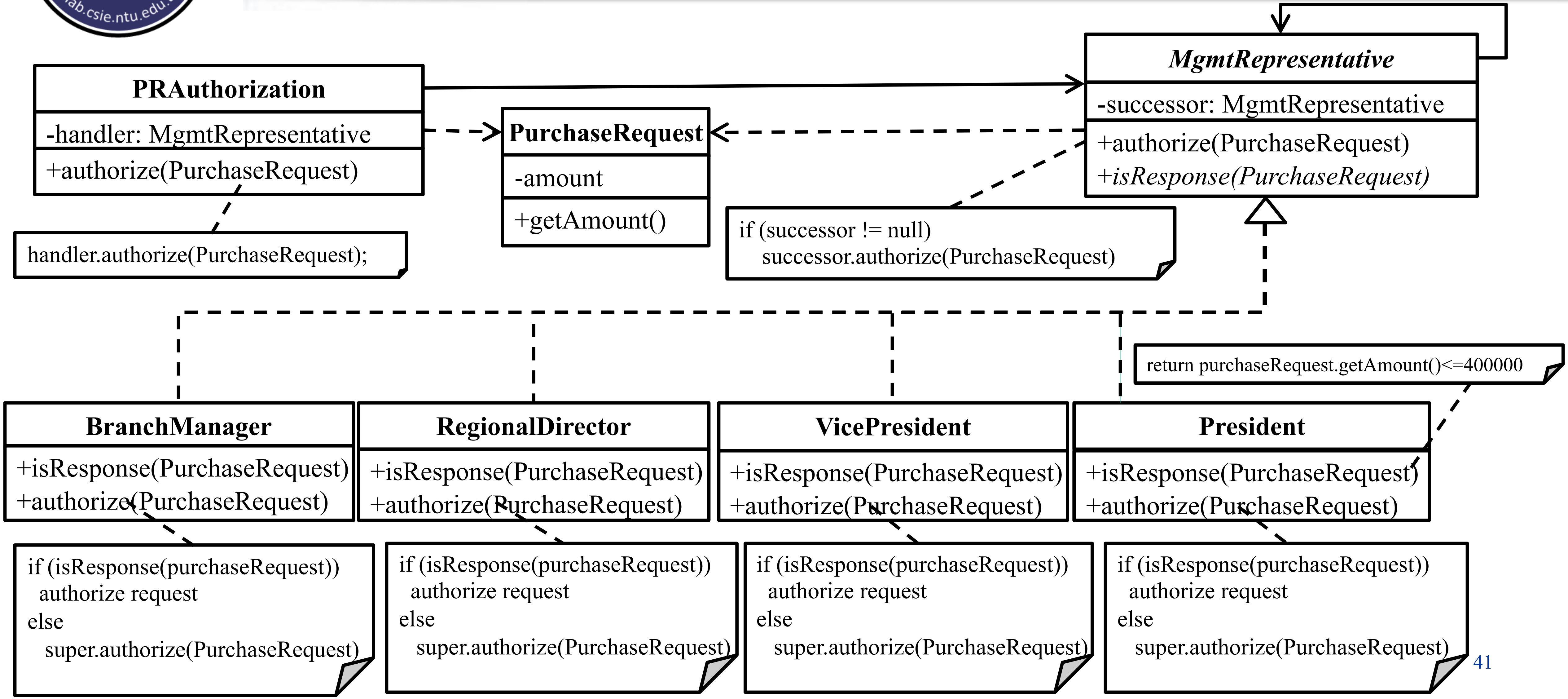
Change interface to abstract class in order to hold the successor's reference, and implement the basic behavior of authorization.

Delegate to the succeeding handler.

PRAuthorization
-handler: MgmtRepresentative
+authorize(PurchaseRequest)
handler.authorize(PurchaseRequest);
PurchaseRequest
-amount
+getAmount()
MgmtRepresentative
-successor: MgmtRepresentative
+authorize(PurchaseRequest)
+*isResponse(PurchaseRequest)*
if (successor != null)
successor.authorize(PurchaseRequest)
return purchaseRequest.getAmount()<=400000
BranchManager
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
RegionalDirector
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
VicePresident
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
President
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)

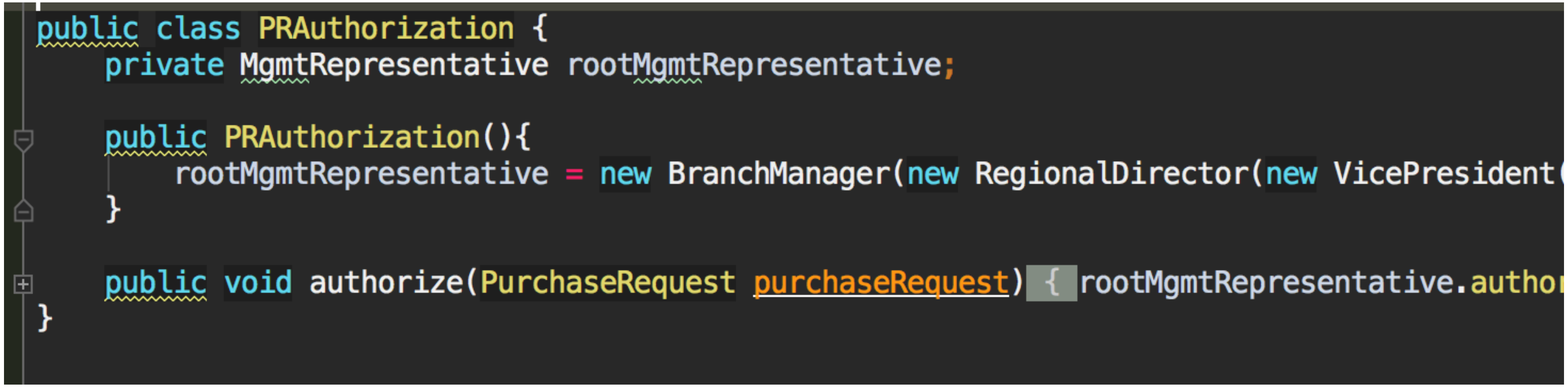
```
if (isResponse(purchaseRequest))
  authorize request
else
  super.authorize(PurchaseRequest)
```

Refactored Design after Design Process



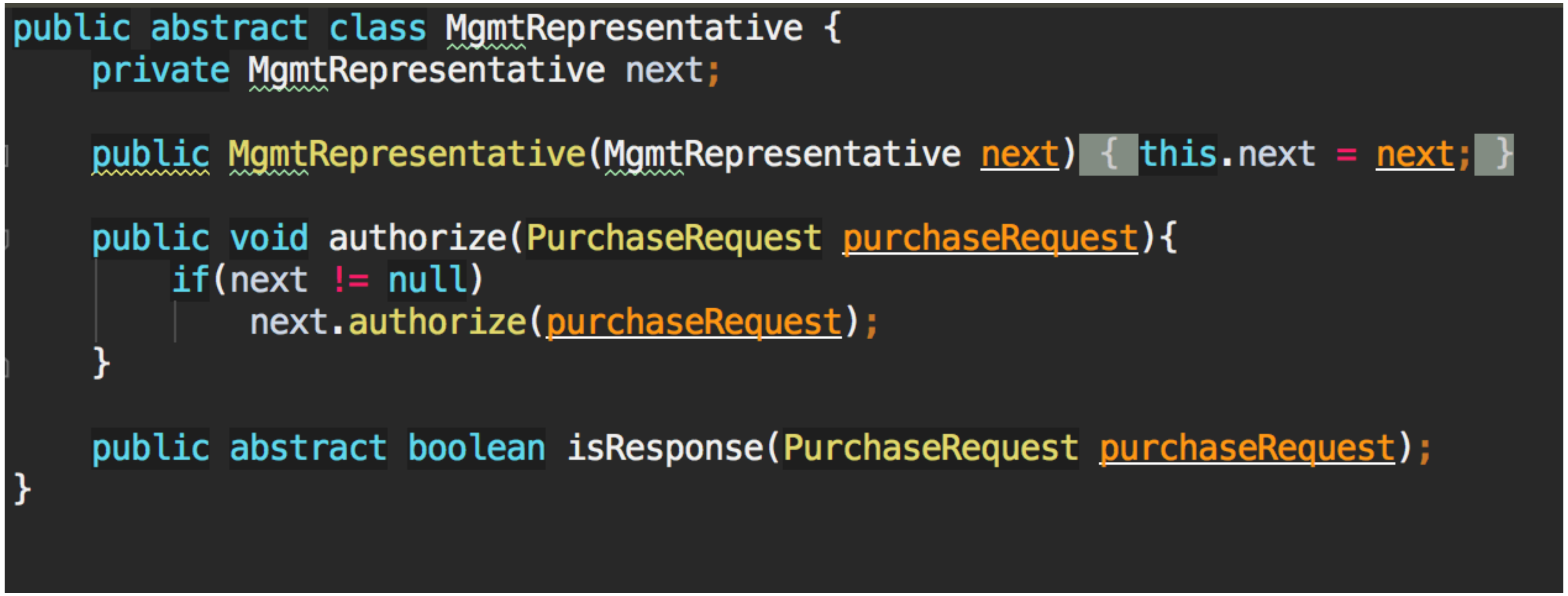
PRAuthorization
-handler: MgmtRepresentative
+authorize(PurchaseRequest)
handler.authorize(PurchaseRequest);
PurchaseRequest
-amount
+getAmount()
MgmtRepresentative
-successor: MgmtRepresentative
+authorize(PurchaseRequest)
+isResponse(PurchaseRequest)
if (successor != null)
successor.authorize(PurchaseRequest)
return purchaseRequest.getAmount()<=400000
BranchManager
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
RegionalDirector
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
VicePresident
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
President
+isResponse(PurchaseRequest)
+authorize(PurchaseRequest)
if (isResponse(purchaseRequest))
authorize request
else
super.authorize(PurchaseRequest)
if (isResponse(purchaseRequest))
authorize request
else
super.authorize(PurchaseRequest)
if (isResponse(purchaseRequest))
authorize request
else
super.authorize(PurchaseRequest)
if (isResponse(purchaseRequest))
authorize request
else
super.authorize(PurchaseRequest)

PRAuthorization



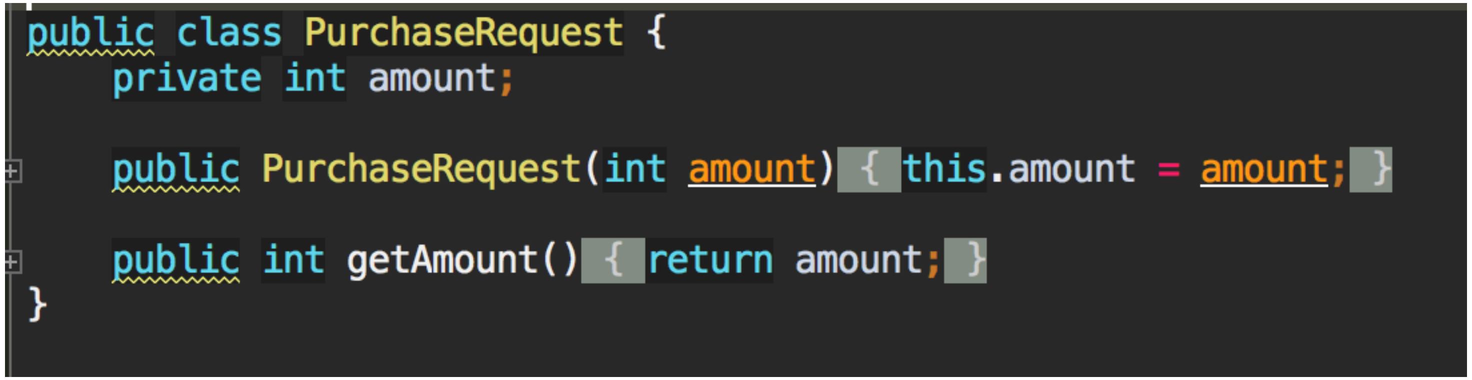
```java
public class PRAuthorization {
    private MgmtRepresentative rootMgmtRepresentative;

    public PRAuthorization(){
        rootMgmtRepresentative = new BranchManager(new RegionalDirector(new VicePresident(
    }

    public void authorize(PurchaseRequest purchaseRequest) { rootMgmtRepresentative.author
}
```

MgmtRepresentative



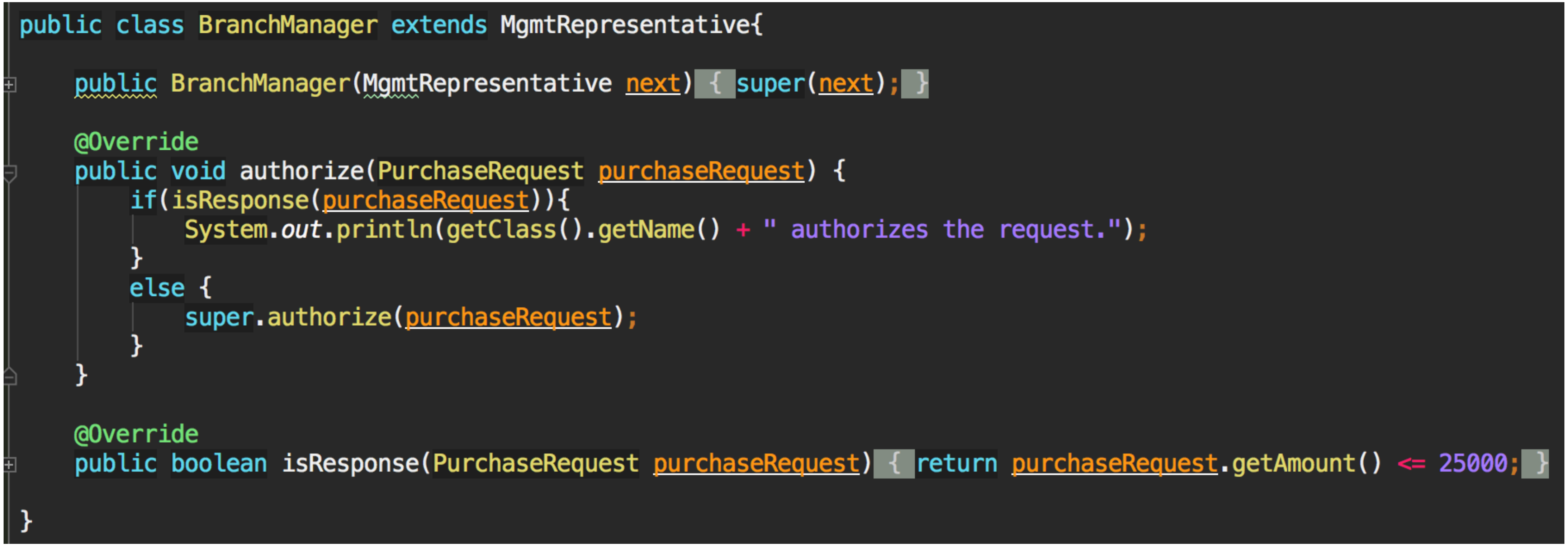
```
public abstract class MgmtRepresentative {
    private MgmtRepresentative next;

    public MgmtRepresentative(MgmtRepresentative next) { this.next = next; }

    public void authorize(PurchaseRequest purchaseRequest){
        if(next != null)
            next.authorize(purchaseRequest);
    }

    public abstract boolean isResponse(PurchaseRequest purchaseRequest);
}
```

PruchaseRequest



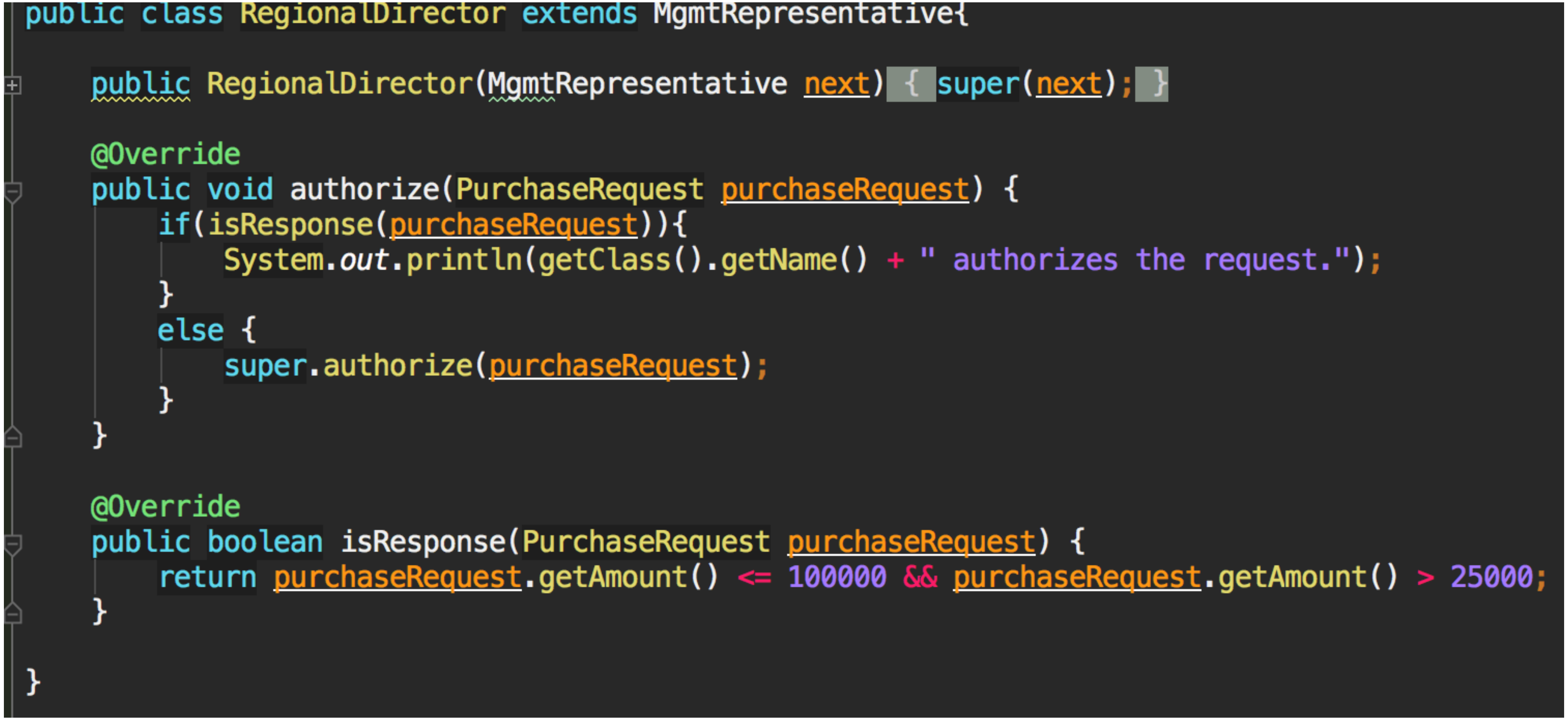
```java
public class PurchaseRequest {
    private int amount;

    public PurchaseRequest(int amount) { this.amount = amount; }

    public int getAmount() { return amount; }
}
```

BranchManager



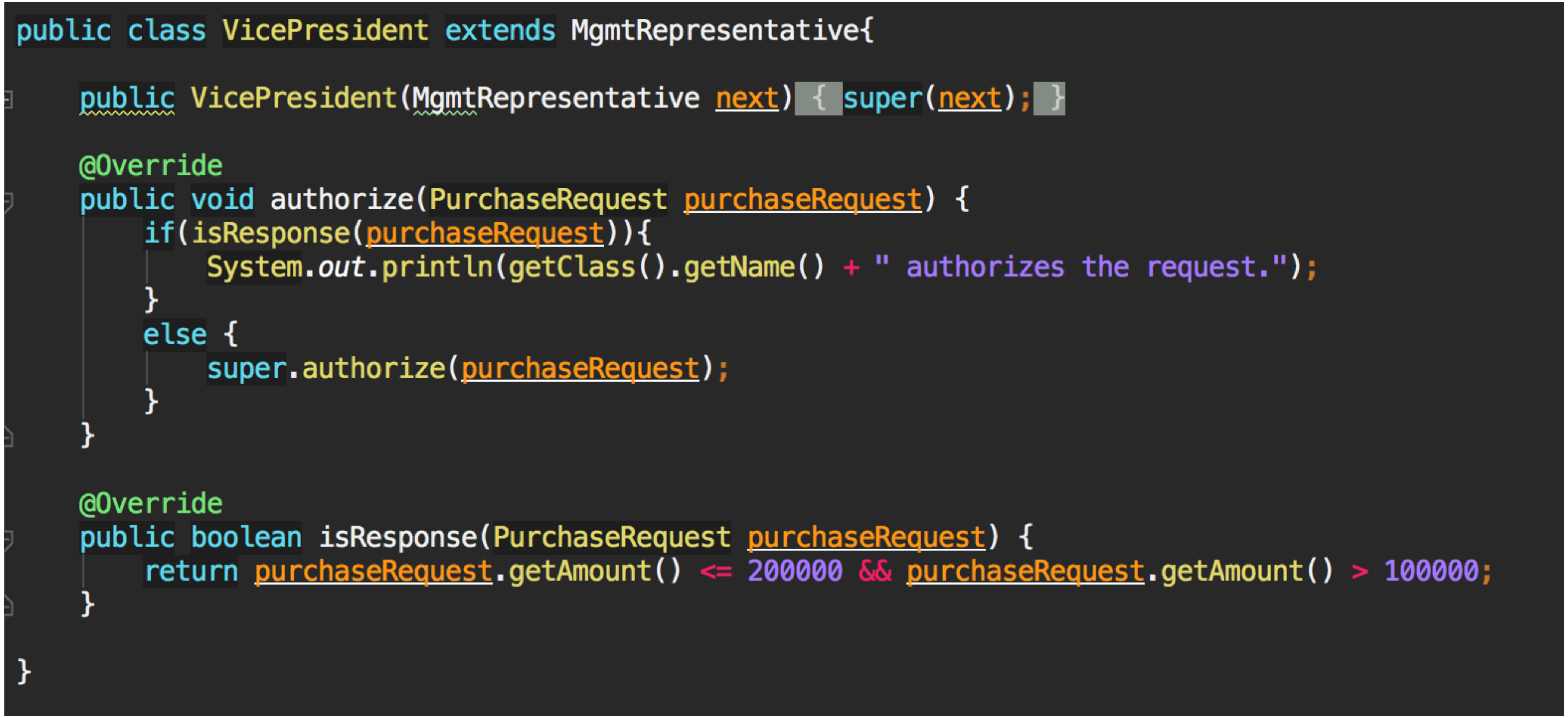
```java
public class BranchManager extends MgmtRepresentative{

    public BranchManager(MgmtRepresentative next) { super(next); }

    @Override
    public void authorize(PurchaseRequest purchaseRequest) {
        if(isResponse(purchaseRequest)){
            System.out.println(getClass().getName() + " authorizes the request.");
        }
        else {
            super.authorize(purchaseRequest);
        }
    }

    @Override
    public boolean isResponse(PurchaseRequest purchaseRequest) { return purchaseRequest.getAmount() <= 25000; }

}
```

RegionalDirector



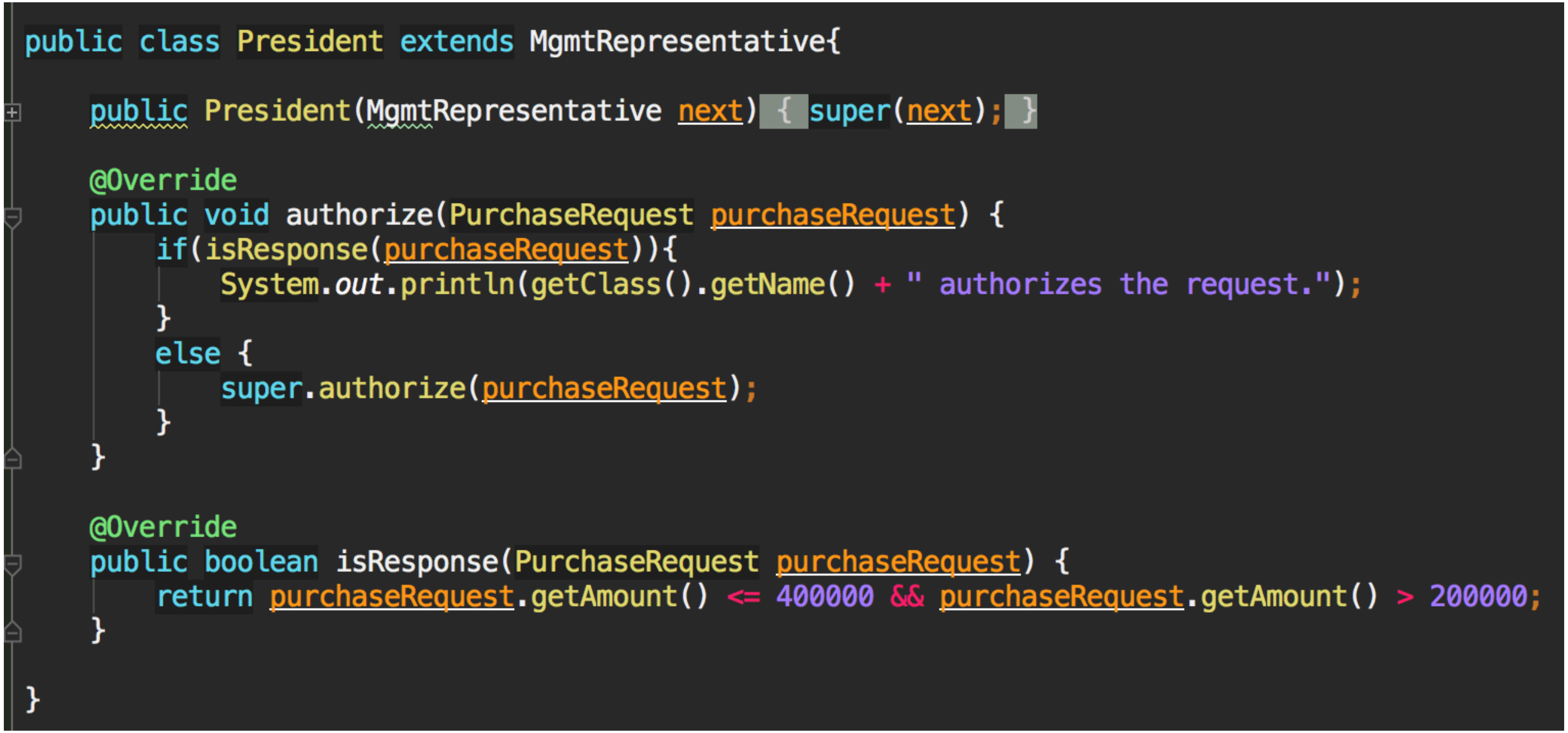
```
public class RegionalDirector extends MgmtRepresentative{

    public RegionalDirector(MgmtRepresentative next) { super(next); }

    @Override
    public void authorize(PurchaseRequest purchaseRequest) {
        if(isResponse(purchaseRequest)){
            System.out.println(getClass().getName() + " authorizes the request.");
        }
        else {
            super.authorize(purchaseRequest);
        }
    }

    @Override
    public boolean isResponse(PurchaseRequest purchaseRequest) {
        return purchaseRequest.getAmount() <= 100000 && purchaseRequest.getAmount() > 25000;
    }

}
```

VicePresident



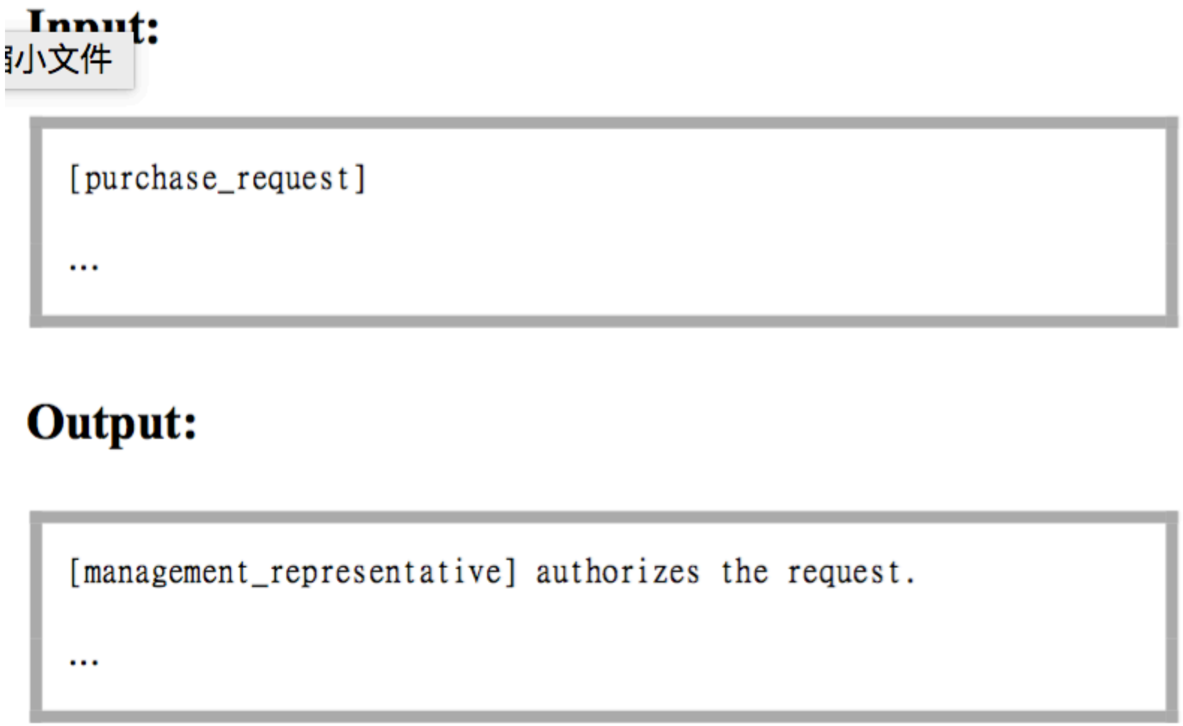
```
public class VicePresident extends MgmtRepresentative{

    public VicePresident(MgmtRepresentative next) { super(next); }

    @Override
    public void authorize(PurchaseRequest purchaseRequest) {
        if(isResponse(purchaseRequest)){
            System.out.println(getClass().getName() + " authorizes the request.");
        }
        else {
            super.authorize(purchaseRequest);
        }
    }

    @Override
    public boolean isResponse(PurchaseRequest purchaseRequest) {
        return purchaseRequest.getAmount() <= 200000 && purchaseRequest.getAmount() > 100000;
    }

}
```

President



```java
public class President extends MgmtRepresentative{

    public President(MgmtRepresentative next) { super(next); }

    @Override
    public void authorize(PurchaseRequest purchaseRequest) {
        if(isResponse(purchaseRequest)){
            System.out.println(getClass().getName() + " authorizes the request.");
        }
        else {
            super.authorize(purchaseRequest);
        }
    }

    @Override
    public boolean isResponse(PurchaseRequest purchaseRequest) {
        return purchaseRequest.getAmount() <= 400000 && purchaseRequest.getAmount() > 200000;
    }

}
```

Input / Output format



Input:

小文件

```
[purchase_request]

…
```

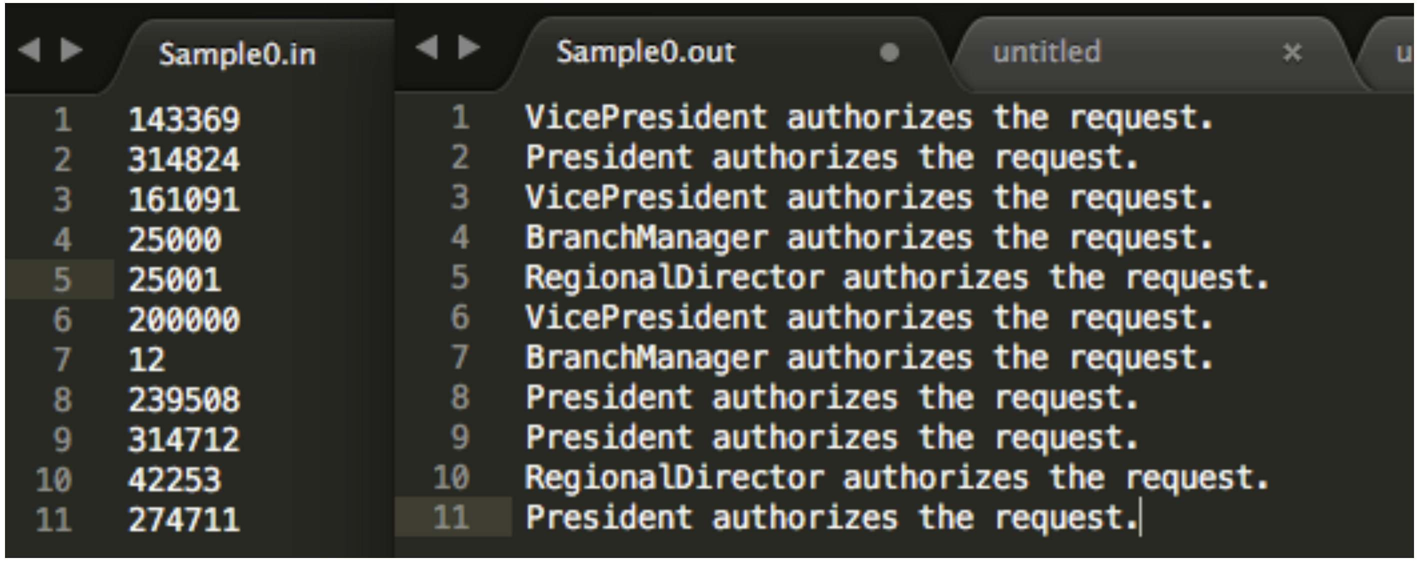
Output:

```
[management_representative] authorizes the request.

…
```

Test cases

- Test case1: simple
- Test case2: 100 random number

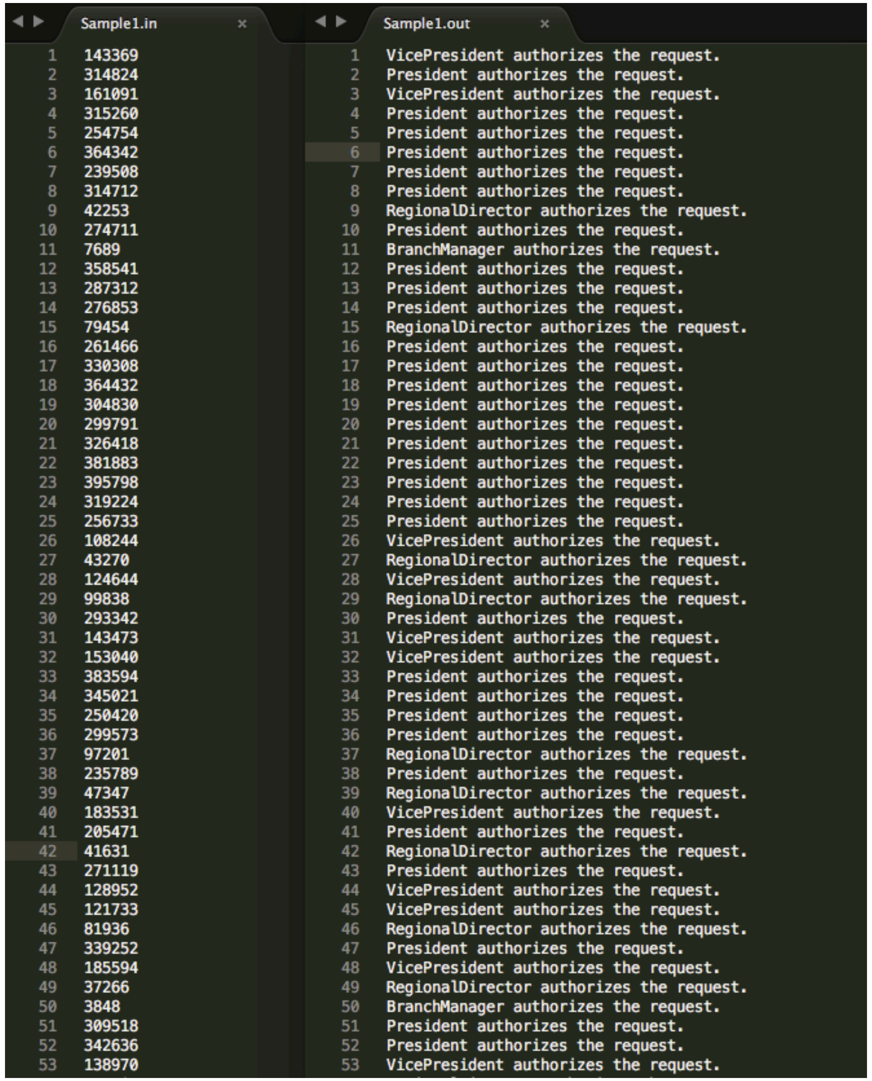
Test case1



```
143369
314824
161091
25000
25001
200000
12
239508
314712
42253
274711
```

```
VicePresident authorizes the request.
President authorizes the request.
VicePresident authorizes the request.
BranchManager authorizes the request.
RegionalDirector authorizes the request.
VicePresident authorizes the request.
BranchManager authorizes the request.
President authorizes the request.
President authorizes the request.
RegionalDirector authorizes the request.
President authorizes the request.
```

Test case2



```
143369
314824
161091
315260
254754
364342
239508
314712
42253
274711
7689
358541
287312
276853
79454
261466
330308
364432
304830
299791
326418
381883
395798
319224
256733
108244
43270
124644
99838
293342
143473
153040
383594
345021
250420
299573
97201
235789
47347
183531
205471
41631
271119
128952
121733
81936
339252
185594
37266
3848
309518
342636
138970
```

```
VicePresident authorizes the request.
President authorizes the request.
VicePresident authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
RegionalDirector authorizes the request.
President authorizes the request.
BranchManager authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
RegionalDirector authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
VicePresident authorizes the request.
RegionalDirector authorizes the request.
VicePresident authorizes the request.
RegionalDirector authorizes the request.
President authorizes the request.
VicePresident authorizes the request.
VicePresident authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
President authorizes the request.
RegionalDirector authorizes the request.
President authorizes the request.
RegionalDirector authorizes the request.
VicePresident authorizes the request.
President authorizes the request.
RegionalDirector authorizes the request.
President authorizes the request.
VicePresident authorizes the request.
VicePresident authorizes the request.
RegionalDirector authorizes the request.
President authorizes the request.
VicePresident authorizes the request.
RegionalDirector authorizes the request.
BranchManager authorizes the request.
President authorizes the request.
President authorizes the request.
VicePresident authorizes the request.
```