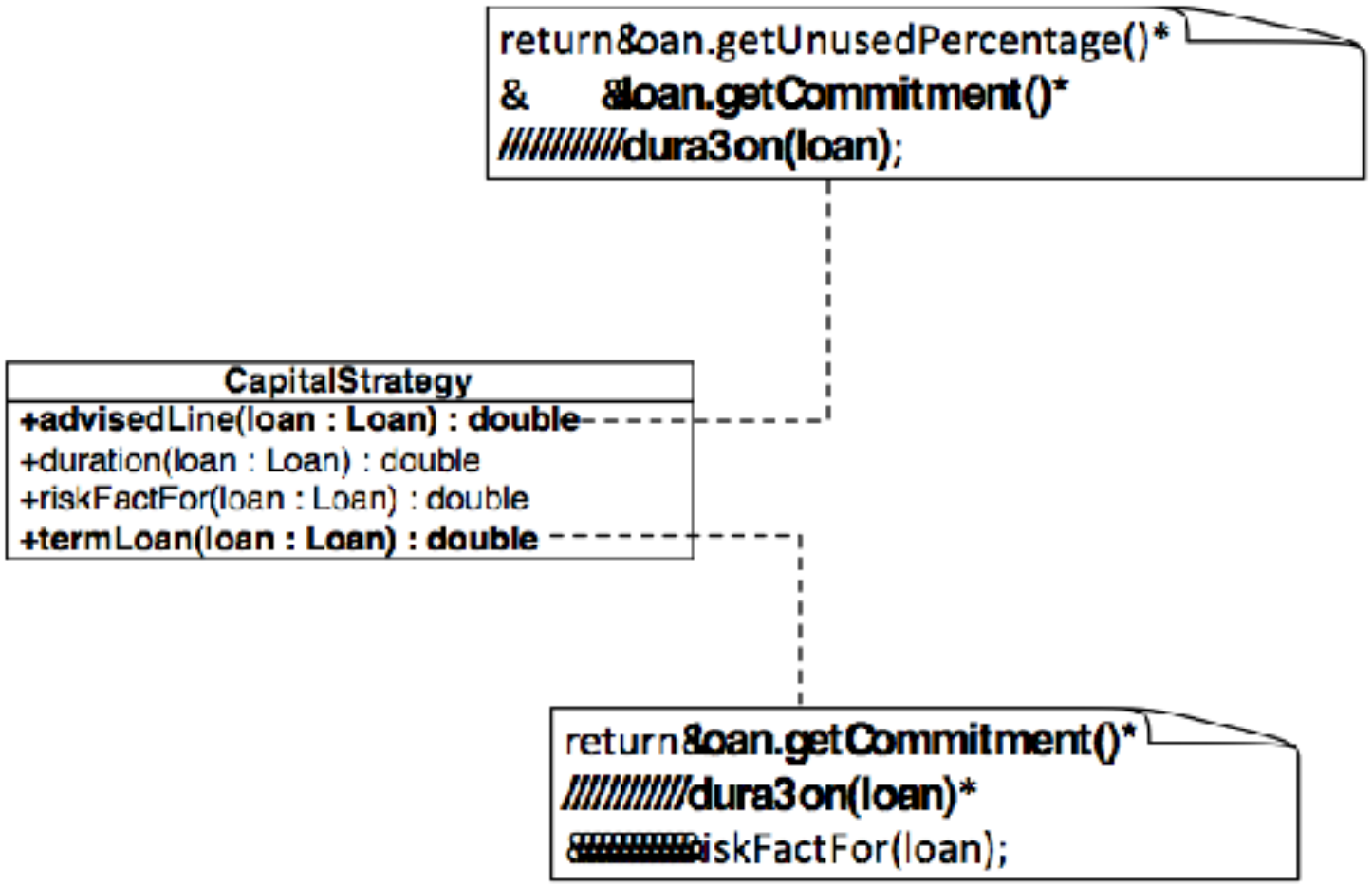
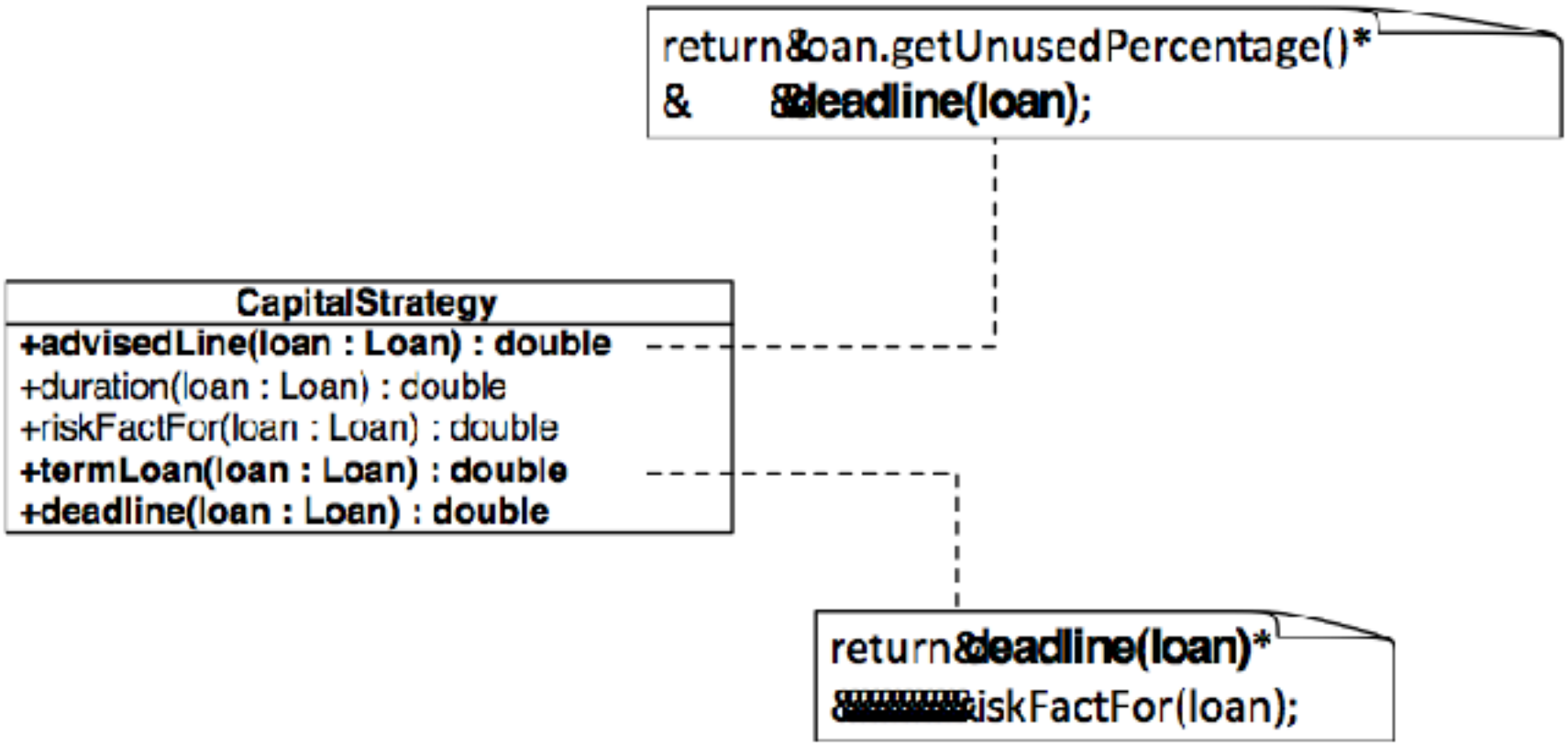
Bad Smells to Patterns

Baldoino Fonseca
baldoino@ic.ufal.br

Problem


return&loan.getUnusedPercentage()*
&&loan.getCommitment()*
////////////dura3on(loan);
CapitalStrategy
+advisedLine(loan : Loan) : double
+duration(loan : Loan) : double
+riskFactFor(loan : Loan) : double
+termLoan(loan : Loan) : double
return&loan.getCommitment()*
////////////dura3on(loan)*
&&&&&&&&&&iskFactFor(loan);

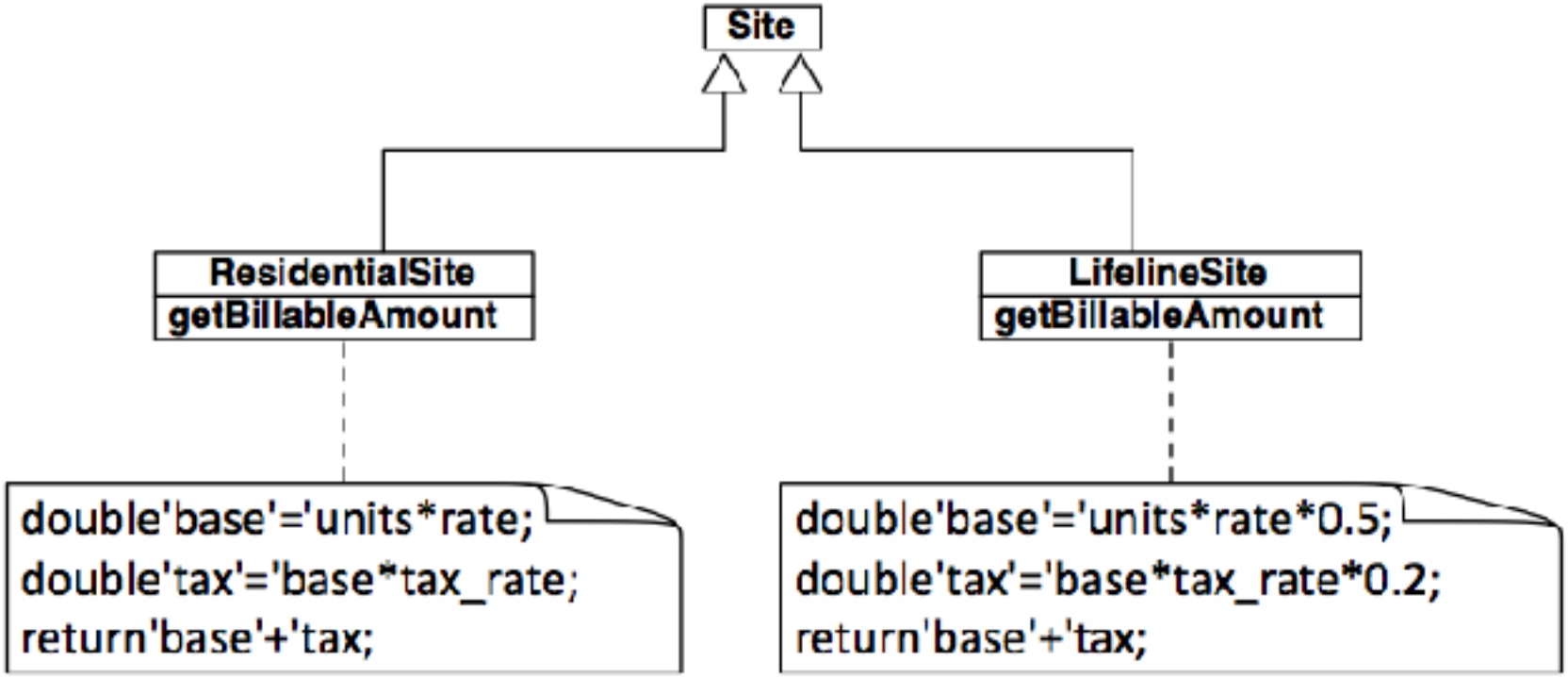
Solution


return&loan.getUnusedPercentage()*
&deadline(loan);
CapitalStrategy
+advisedLine(loan : Loan) : double
+duration(loan : Loan) : double
+riskFactFor(loan : Loan) : double
+termLoan(loan : Loan) : double
+deadline(loan : Loan) : double
return&deadline(loan)*
&riskFactFor(loan);

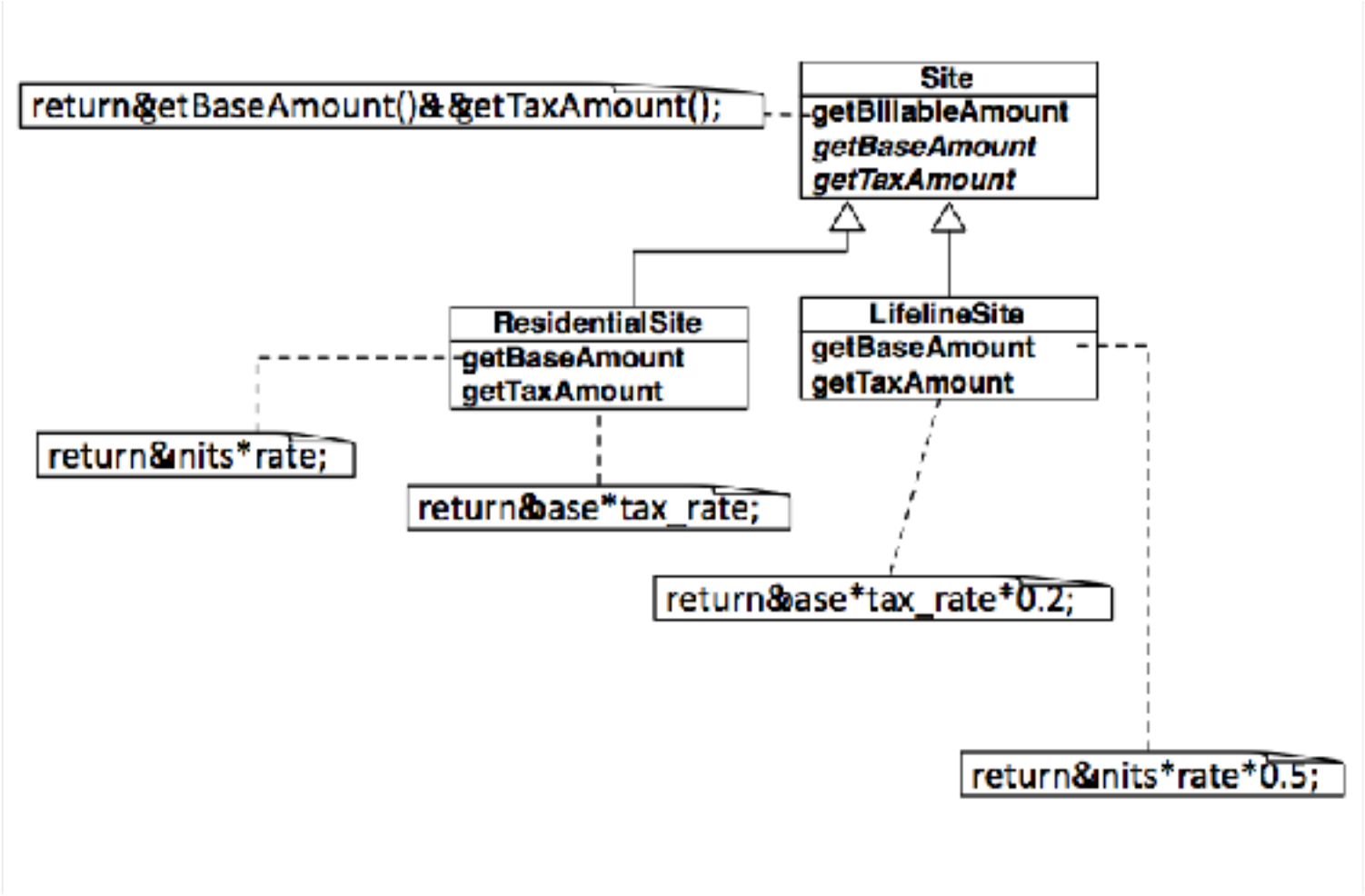
Extract Method

You have a code fragment that can be grouped together

Problem


Site
ResidentialSite
getBillableAmount
LifelineSite
getBillableAmount
double'base'='units*rate;
double'tax'='base*tax_rate;
return'base'+'tax;
double'base'='units*rate*0.5;
double'tax'='base*tax_rate*0.2;
return'base'+'tax;

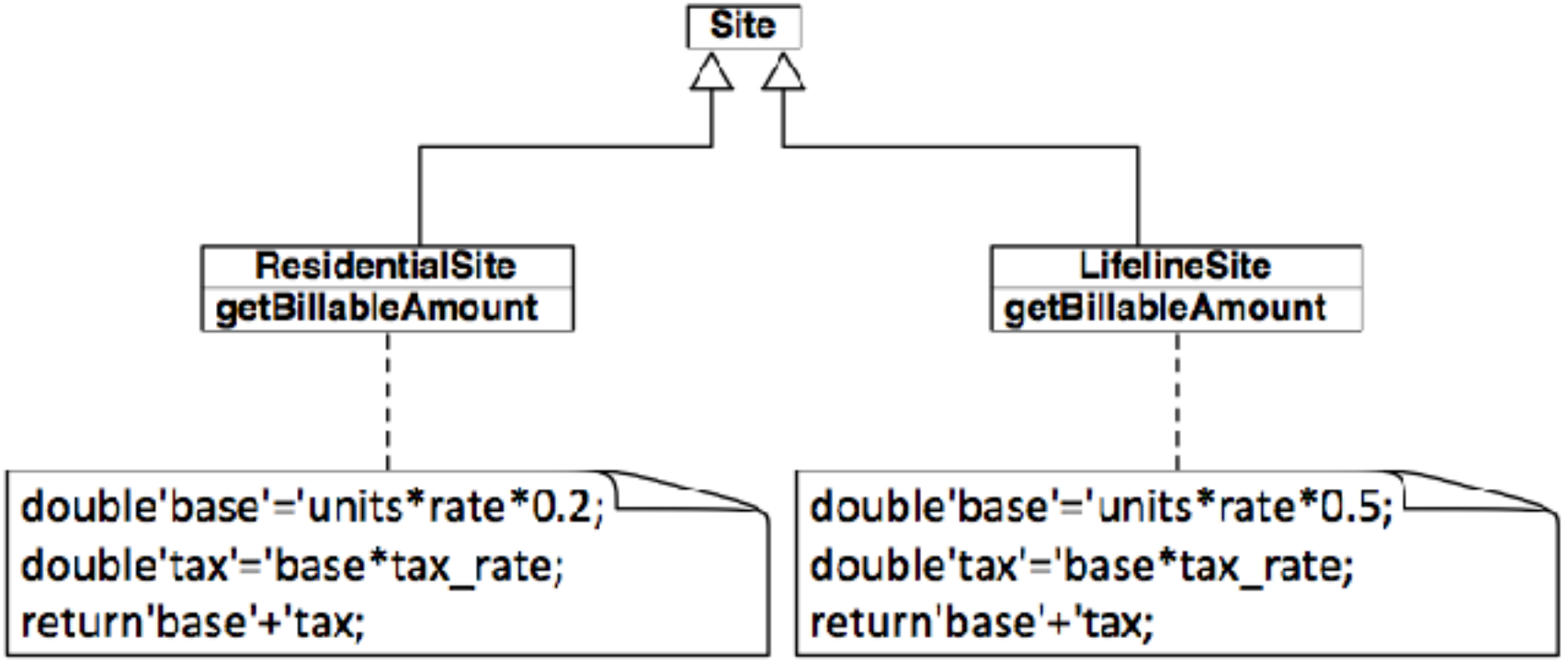
Solution


return&getBaseAmount()&&getTaxAmount();
Site
getBillableAmount
getBaseAmount
getTaxAmount
ResidentialSite
getBaseAmount
getTaxAmount
LifelineSite
getBaseAmount
getTaxAmount
return&nits*rate;
return&base*tax_rate;
return&base*tax_rate*0.2;
return&nits*rate*0.5;

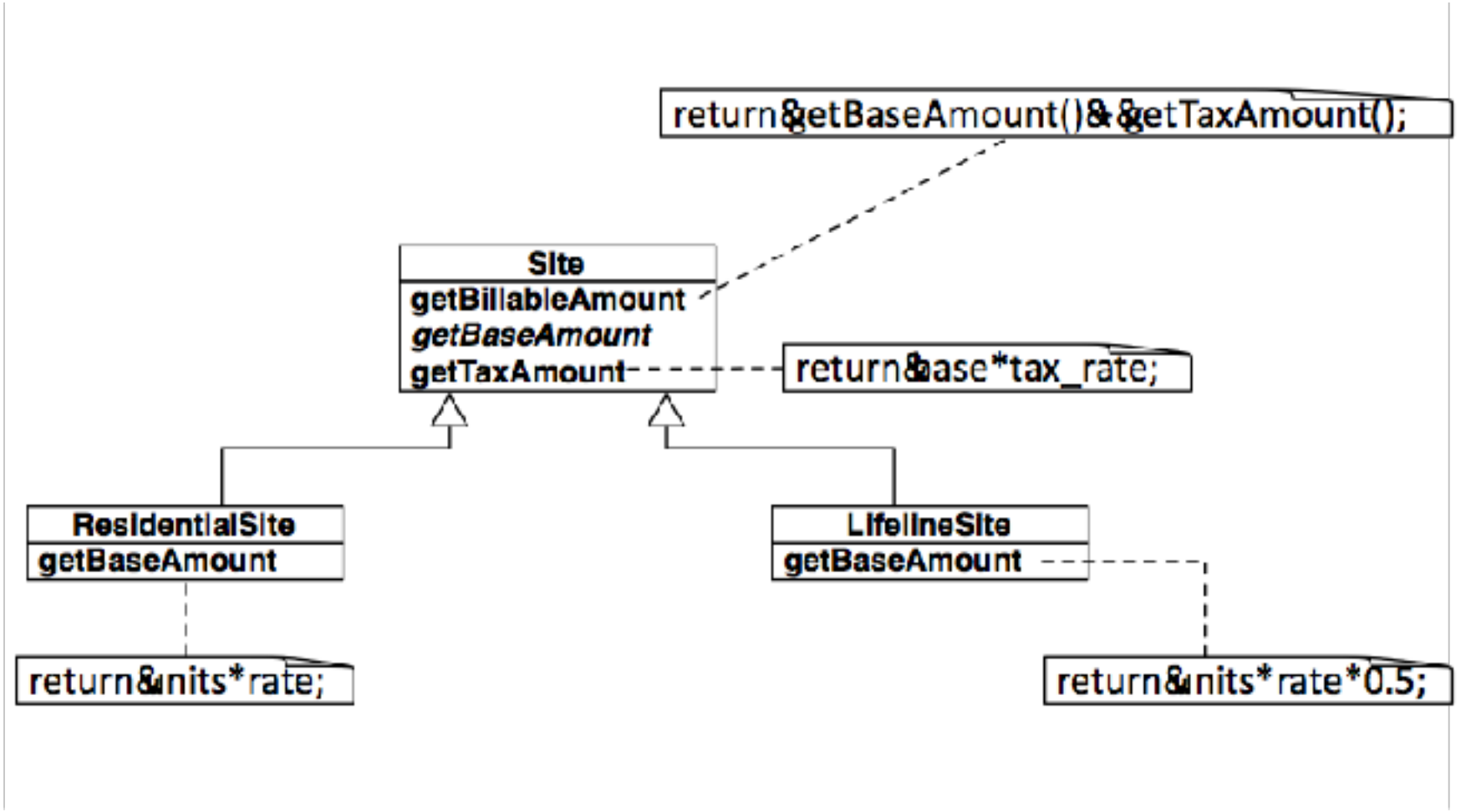
Template Method

You have two methods in subclasses that perform steps in the same order, yet the steps are different.

Problem

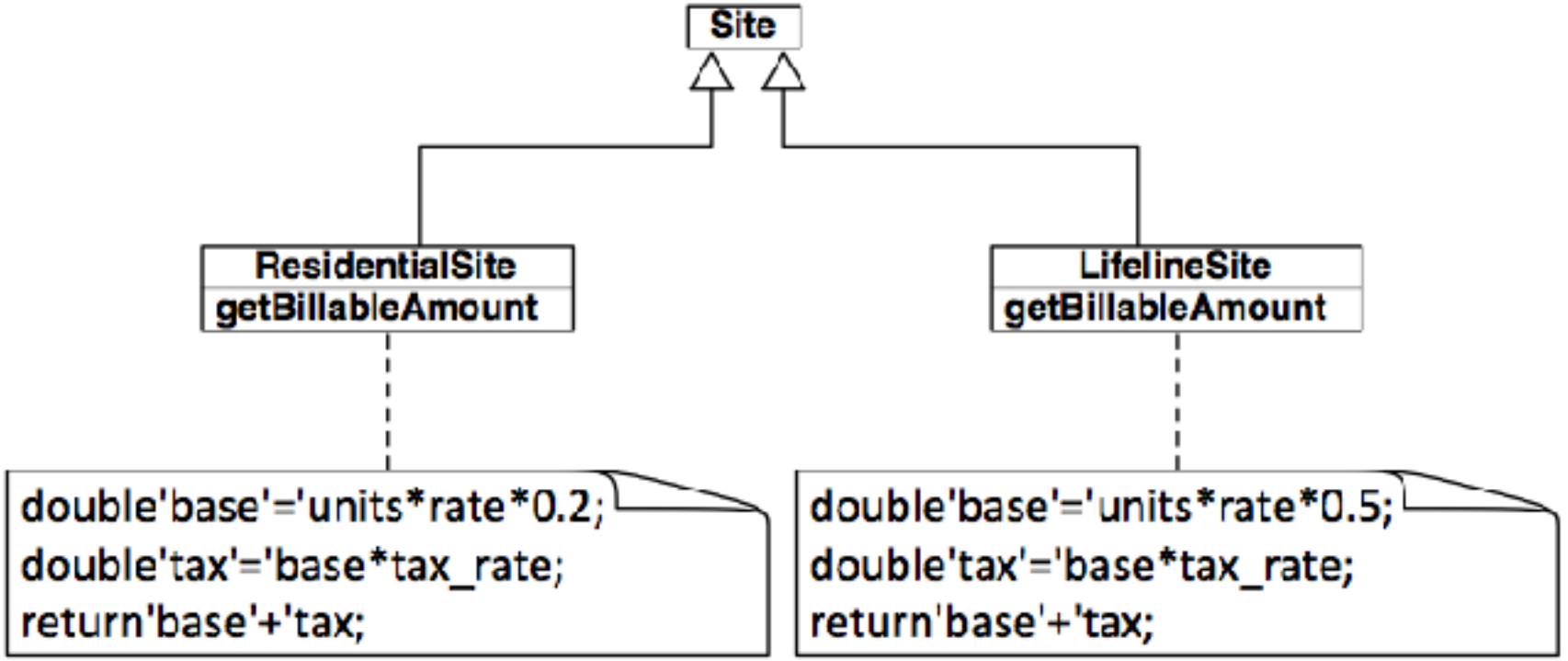

Site
ResidentialSite
getBillableAmount
LifelineSite
getBillableAmount
double'base'='units*rate*0.2;
double'tax'='base*tax_rate;
return'base'+'tax;
double'base'='units*rate*0.5;
double'tax'='base*tax_rate;
return'base'+'tax;

Solution

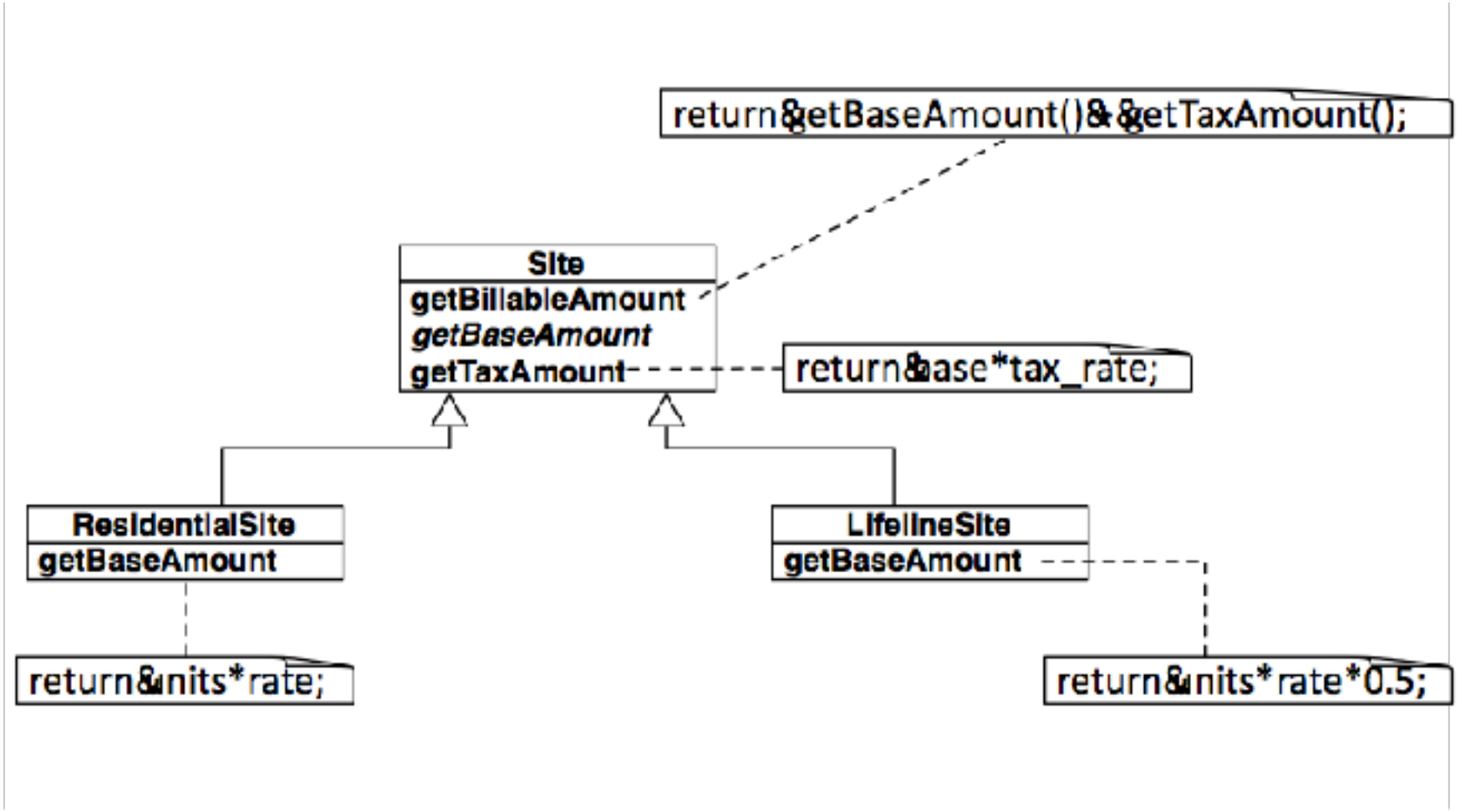

return&getBaseAmount()&&getTaxAmount();
Site
getBillableAmount
getBaseAmount
getTaxAmount
return&base*tax_rate;
ResidentialSite
getBaseAmount
LifelineSite
getBaseAmount
return&units*rate;
return&units*rate*0.5;

Template e Extract Method

Problem

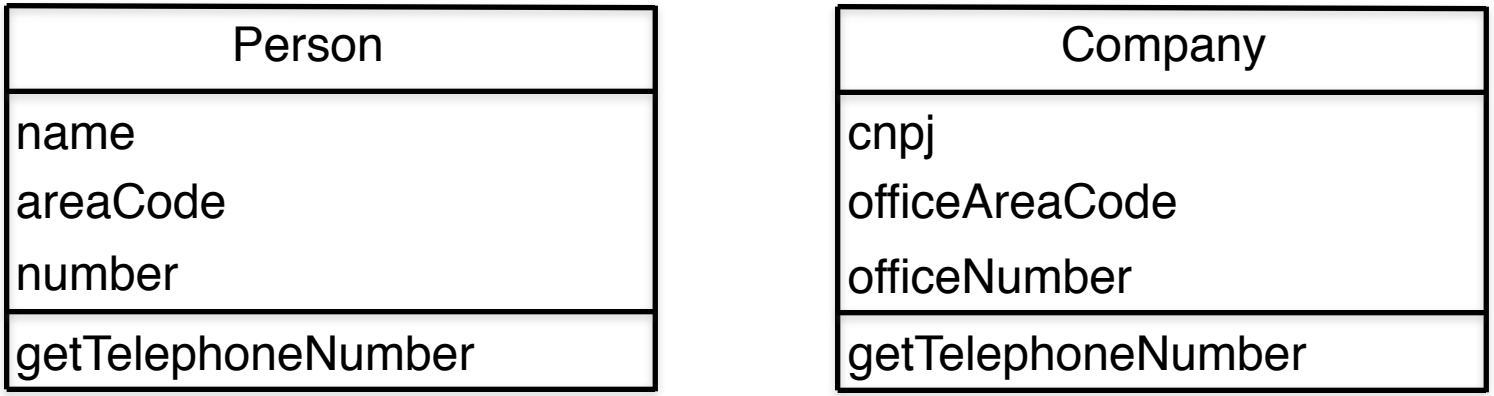

Site
ResidentialSite
getBillableAmount
LifelineSite
getBillableAmount
double'base'='units*rate*0.2;
double'tax'='base*tax_rate;
return'base'+'tax;
double'base'='units*rate*0.5;
double'tax'='base*tax_rate;
return'base'+'tax;

Solution

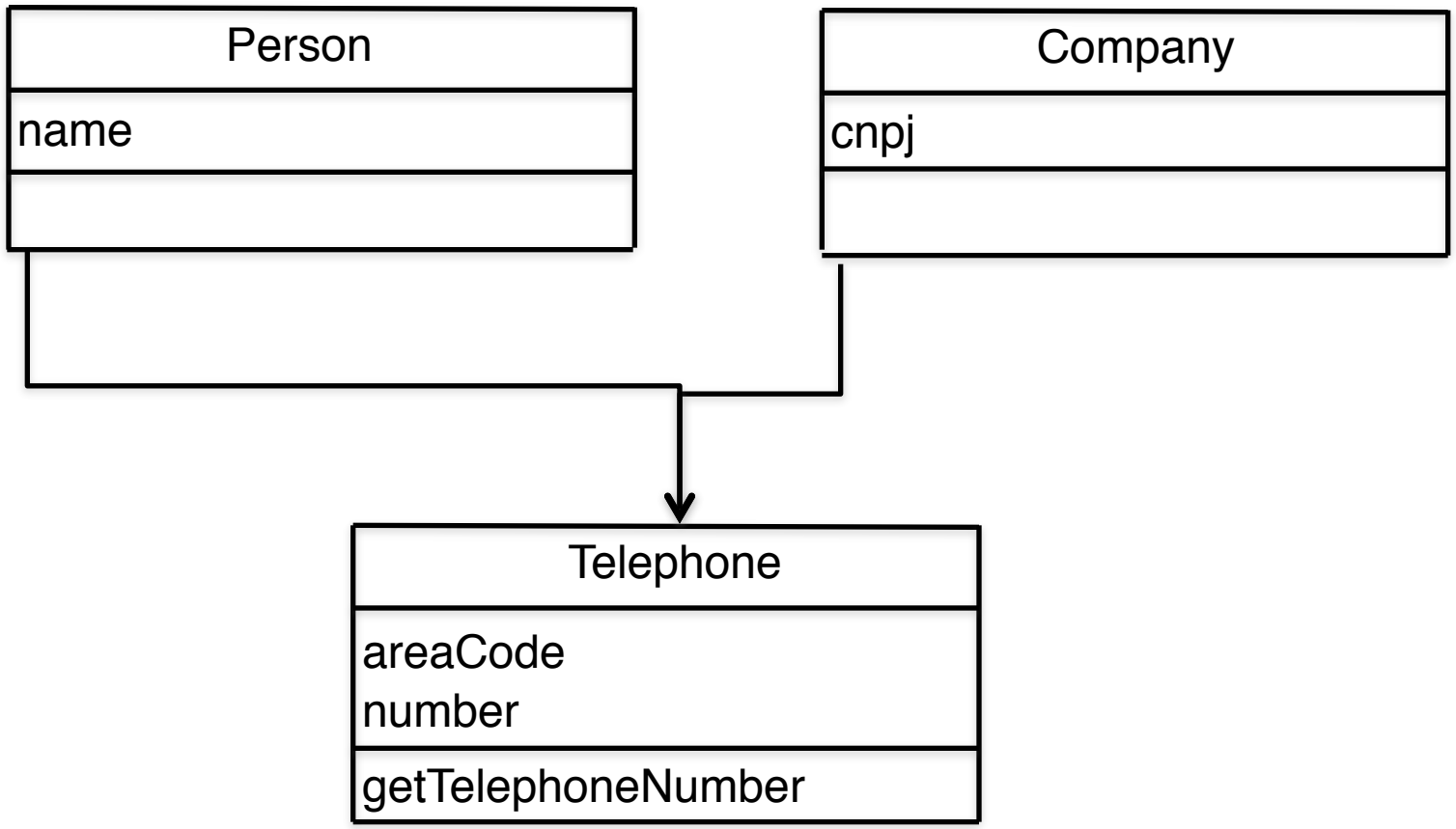

return&getBaseAmount()&&getTaxAmount();
Site
getBillableAmount
getBaseAmount
getTaxAmount
return&base*tax_rate;
ResidentialSite
getBaseAmount
LifelineSite
getBaseAmount
return&units*rate;
return&units*rate*0.5;

Template e Extract Method

Problem


Person
name
areaCode
number
getTelephoneNumber
Company
cnpj
officeAreaCode
officeNumber
getTelephoneNumber

Solution


Person
name
Company
cnpj
Telephone
areaCode
number
getTelephoneNumber

Extract Class

Problem

```
public class Loan {

    private CapitalStrategy strategy;
    private float notional;
    private float outstanding;
    private int rating;
    private Date expiry;
    private Date maturity;

    public Loan(float notional, float outstanding, int rating,
Date expiry){
        this.strategy = new TermROC();
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
    }

    public Loan(float notional, float outstanding, int rating,
Date expiry,                Date maturity){
        this.strategy = new RevolvingTermROC();
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }

    public Loan(CapitalStrategy strategy, float notional,
float outstanding, int rating, Date expiry, Date maturity){
        this.strategy = strategy;
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }
}
```

Solution

```
public class Loan {

    private CapitalStrategy strategy;
    private float notional;
    private float outstanding;
    private int rating;
    private Date expiry;
    private Date maturity;

    public Loan(float notional, float outstanding, int rating, Date expiry){
        this(new TermROC(), notional, outstanding, rating, expiry, null);
    }

    public Loan(float notional, float outstanding, int rating, Date expiry, Date maturity){
        this(new RevolvingTermROC(), notional, outstanding, rating, expiry, maturity);
    }

    public Loan(CapitalStrategy strategy, float notional, float outstanding, int rating,
Date expiry, Date maturity){
        this.strategy = strategy;
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }
}
```

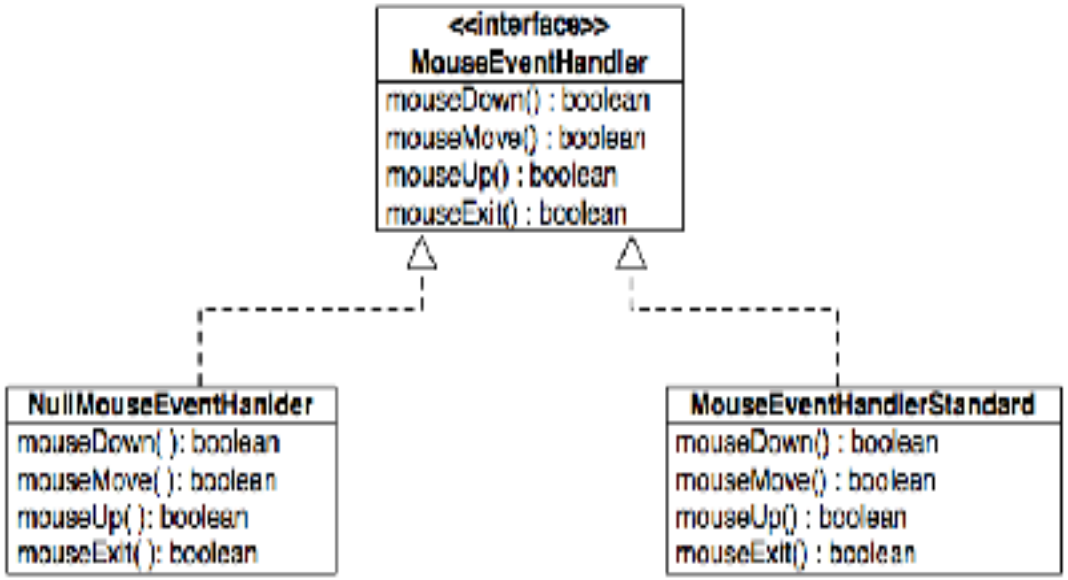
Chain Constructors

```
public class NavigationApplet extends Applet {

    private MouseEventHandler mouseEventHandler = null;

    public NavigationApplet(MouseEventHandler mouseEventHandler) {
        // TODO Auto-generated constructor stub
        this.mouseEventHandler = mouseEventHandler;
    }
    @Override
    public boolean mouseMove() {
        // TODO Auto-generated method stub
        if (mouseEventHandler != null) {
            mouseEventHandler.mouseMove();
        }
        return true;
    }
    @Override
    public boolean mouseDown() {
        // TODO Auto-generated method stub
        if (mouseEventHandler != null) {
            mouseEventHandler.mouseDown();
        }
        return true;
    }
    @Override
    public boolean mouseUp() {
        // TODO Auto-generated method stub
        if (mouseEventHandler != null) {
            mouseEventHandler.mouseUp();
        }
        return true;
    }
    @Override
    public boolean mouseExit() {
        // TODO Auto-generated method stub
        if (mouseEventHandler != null) {
            mouseEventHandler.mouseExit();
        }
        return true;
    }
}
```

Problem

```
public class NavigationApplet extends Applet {

   private MouseEventHandler mouseEventHandler = new NullMouseEventHandler();

   public NavigationApplet(MouseEventHandler mouseEventHandler) {
      // TODO Auto-generated constructor stub
      this.mouseEventHandler = mouseEventHandler;
   }

   @Override
   public boolean mouseMove() {
      // TODO Auto-generated method stub
      return mouseEventHandler.mouseMove();
   }

   @Override
   public boolean mouseDown() {
      // TODO Auto-generated method stub

      return mouseEventHandler.mouseDown();
   }

   @Override
   public boolean mouseUp() {
      // TODO Auto-generated method stub
      return mouseEventHandler.mouseUp();
   }

   @Override
   public boolean mouseExit() {
      // TODO Auto-generated method stub
      return mouseEventHandler.mouseExit();
   }
}
```

Solution



Introduce Null Object

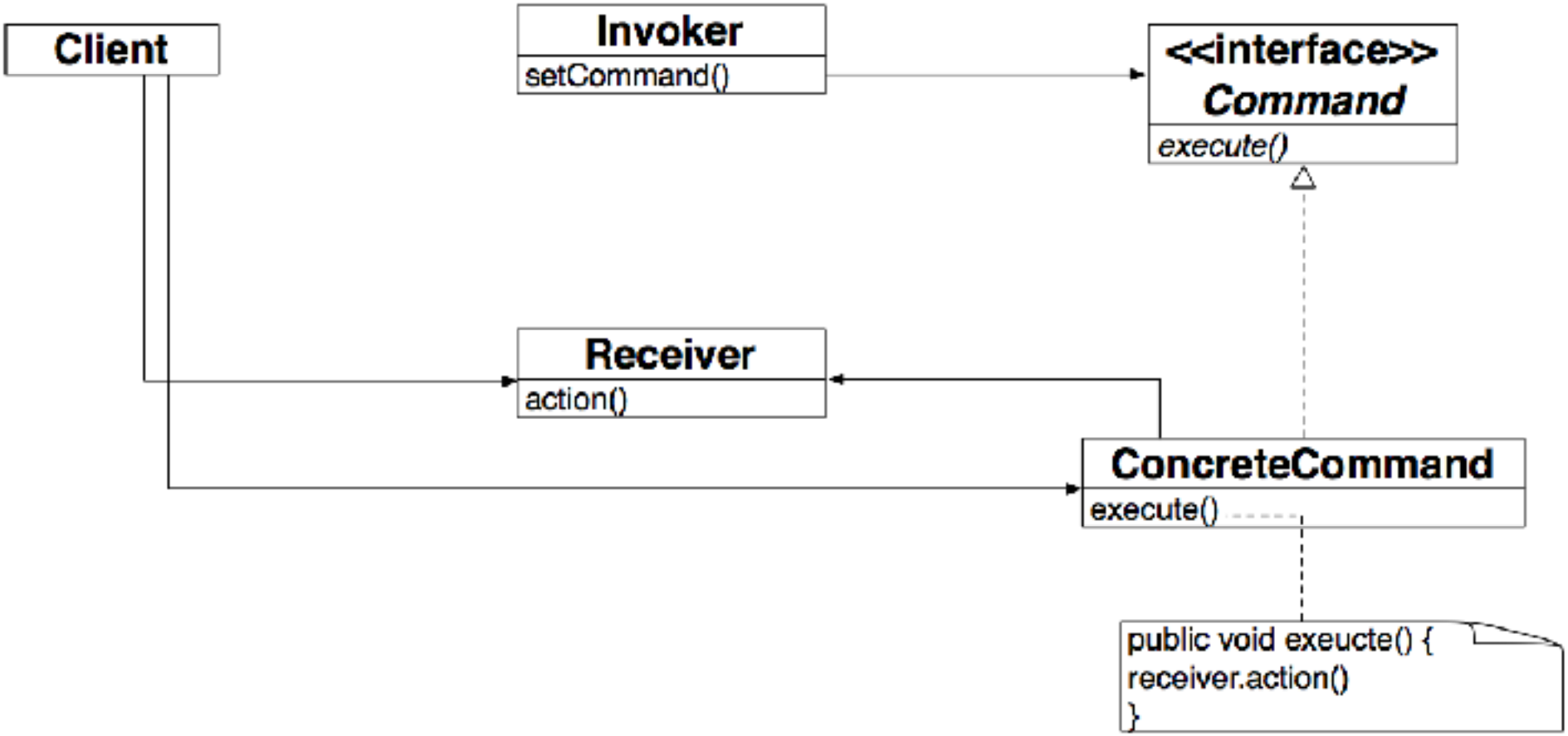
```
public class RemoteControl {

    private GarageDoor garageDoor = new GarageDoor();
    private Light light = new Light();

    public RemoteControl() {

    }

    public void execute(Command command) {

        if (command.equals(Command.UP)) {
            garageDoor.up();
        } if (command.equals(Command.DOWN)) {
            garageDoor.down();
        } if (command.equals(Command.GARAGE_LIGHT_ON)) {
            garageDoor.lightOn();
        } if (command.equals(Command.GARAGE_LIGHT_OFF)) {
            garageDoor.lightOff();
        } if (command.equals(Command.ON)) {
            light.on();
        } if (command.equals(Command.OFF)) {
            light.off();
        }
    }

}
```

GarageDoor
up() : void
down() : void
stop() : void
lightOn() : void
lightOff() : void

Light
on() : void
off() : void

Problem

Abstract Solution

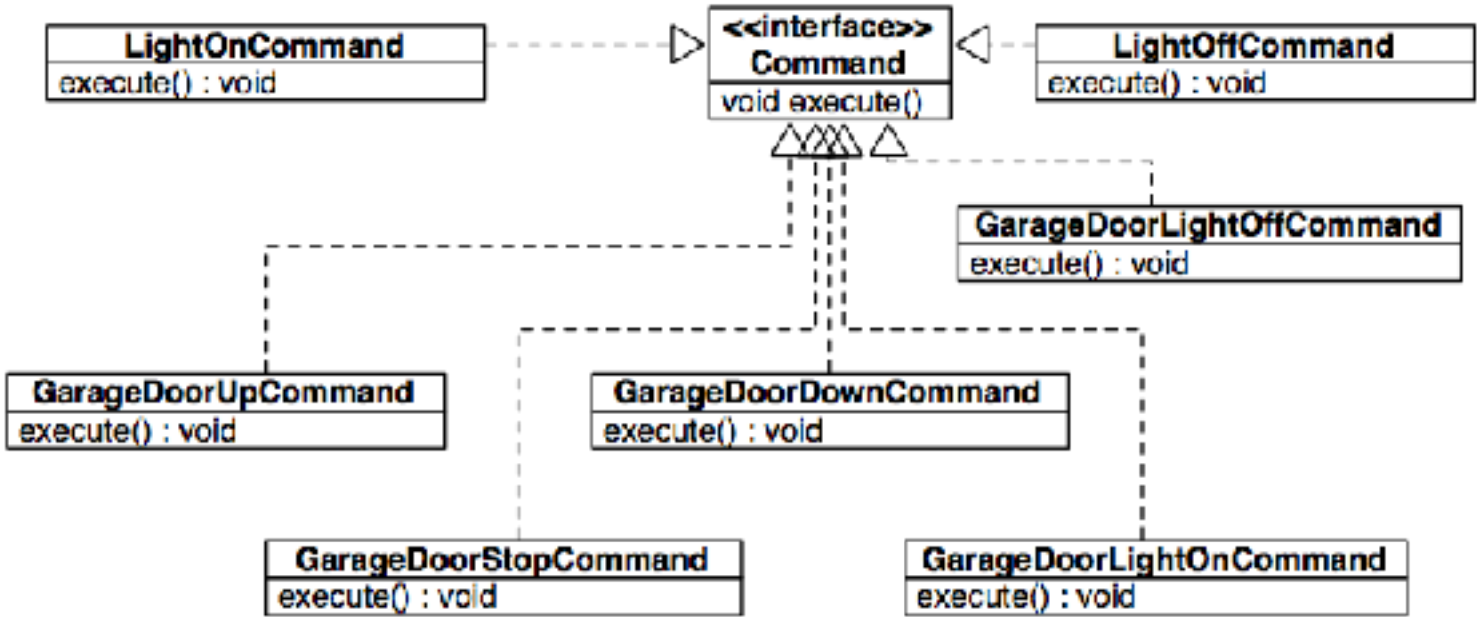


```
public class RemoteControl {
    Command slot;

    public RemoteControl() {}

    public void setCommand(Command command) {
        slot = command;
    }

    public void buttonWasPressed() {
        slot.execute();
    }
}
```

Light
on() : void
off() : void

Solution

GarageDoor
up() : void
down() : void
stop() : void
lightOn() : void
lightOff() : void

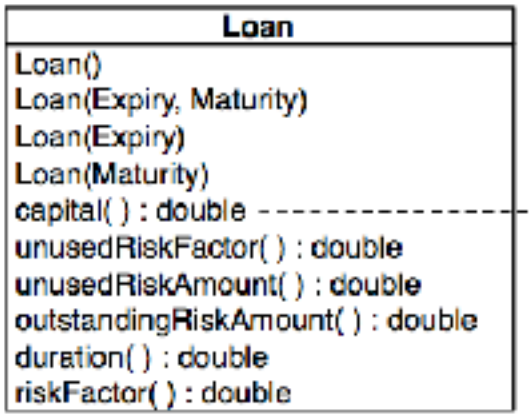


Command

Benefits and Liabilities

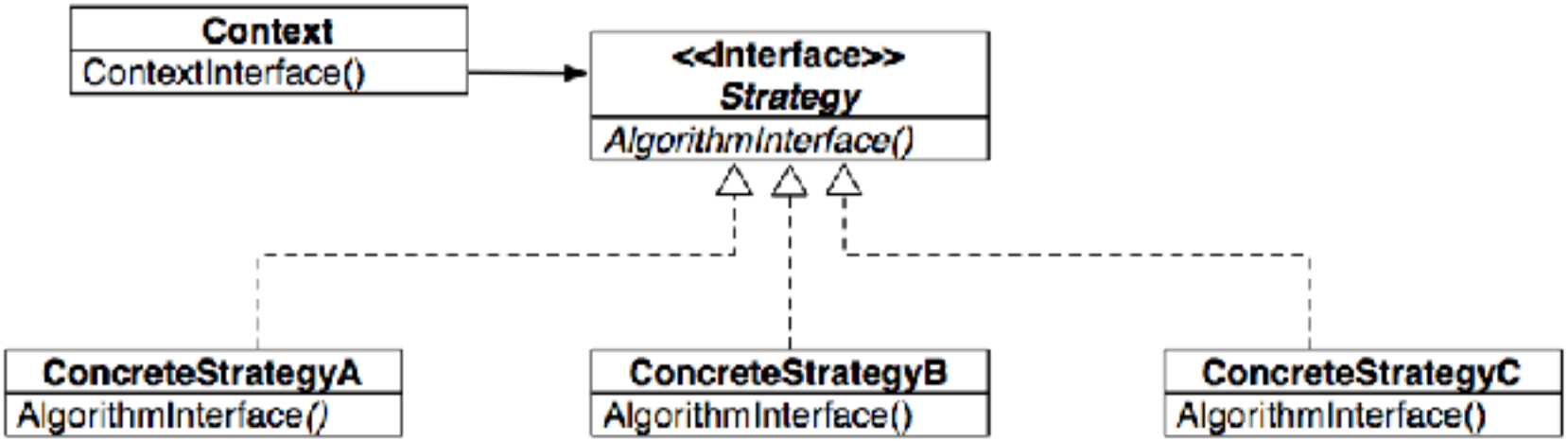
- (+) Provides a simple mechanism for executing diverse behavior in a uniform way
- (+) Enables runtime changes regarding which requests are handled and low
- (+) Requires trivial code to implement
- (-) Complicates a design when a conditional dispatcher is sufficient

Problem

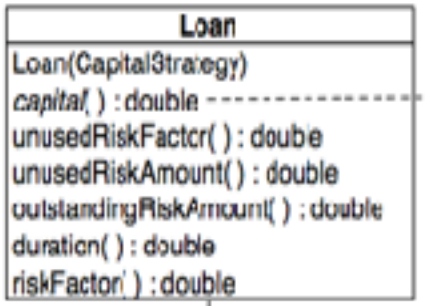



```
if (expiry == null && maturity != null) {
        return commitment * duration() *
riskFactor();
    }else if (expiry == null &&
              maturity == null &&
              getUnusedPercentage() != 1) {
        return commitment *
               getUnusedPercentage() *
               duration() *
               riskFactor();
    }else {
        return outstandingRiskAmount() *
               duration() *
               riskFactor() +
               unusedRiskAmount() *
               duration() *
               unusedRiskFactor();
    }
```

Abstract Solution

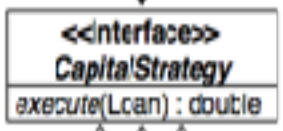


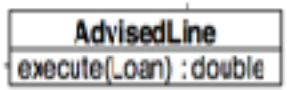
Solution

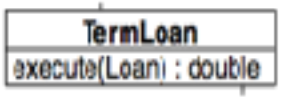


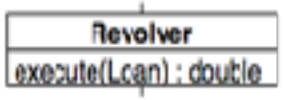

```
return capitalStrategy.execute(this);
```

```
public class Execute {
    public static void main(String[] args) {
        Loan loan = new Loan(new AdvisedLine());
        System.out.println(loan.capital());
    }
}
```










```
return loan.getCommitment() *
       loan.duration() *
       loan.riskFactor();
```

```
return loan.getCommitment() *
       loan.getUnusedPercentage() *
       loan.duration() *
       loan.riskFactor();
```

```
return loan.outstandingRiskAmount() *
       loan.duration() *
       loan.riskFactor() +
       loan.unusedRiskAmount() *
       loan.duration() *
       loan.unusedRiskFactor();
```

Strategy

Benefits and Liabilities

- (+) Clarifies algorithms by decreasing or removing conditional logic
- (+) Simplifies a class by moving variations on an algorithm to a hierarchy
- (+) Enables one algorithm to be swapped for another runtime
- (-) Complicates a design when an inheritance-based solution
- (-) Complicates how algorithms obtain or receive data from their context class

```
public class TagNode {
    @Override
    public String toString() {
        String tagName = null;
        String attributes = null;
        String tagValue = null;

        String result = new String();
        result += "<" + tagName + " " + attributes + ">";
        Iterator it = null;
        while(it.hasNext()){
            TagNode node = (TagNode)it.next();
            result += node.toString();
        }
        if(! tagValue.equals(" "))
            result+= tagValue;
        result+= "<" + tagName + ">";
        return result;
    }
}
```

Problem

Solution

```
public class TagNode {

    private String tagName = null;
    private String attributes = null;
    private String tagValue = null;
    private String result = null;

    private String appendContentsTo( ){
        return writeOpenTagTo( ) + writeChildrenTo( ) +
               writeValueTo( ) + writeEndTagTo( ); }
    private String writeOpenTagTo( ) {
        return  result+="<" + tagName + " " + attributes + ">"; }
    private String writeChildrenTo( ){
        Iterator it = null;
        while(it.hasNext()){
           TagNode node = (TagNode)it.next();
           result += node.toString();
        } return result; }
    private String writeValueTo( ){
         result += "<" + tagName + " " + attributes + ">";
         if(! tagValue.equals(" "))
               result+= tagValue;
         return result; }
    private String writeEndTagTo( ){
        return result+= "<" + tagName + ">"; }
    public String toString() {
        return appendContentsTo( ); }
}
```

Move Accumulation to Collecting Parameter

Benefits and Liabilities

- (+) Helps transform bulky methods into smaller and simples methods.
- (+) Can make resulting code run faster.

Problem

GumballMachine

final static int SOLD_OUT = 0;
final static int NO_QUARTER = 1;
final static int HAS_QUARTER = 2;
final static int SOLD = 3;
int state = SOLD_OUT;

insertQuarter() ejectQuarter() turnCrank() dispense()

```
public void insertQuarter() {
        if (state == HAS_QUARTER) {
            System.out.println("You can't insert another quarter");
        } else if (state == NO_QUARTER) {
            state = HAS_QUARTER;
            System.out.println("You inserted a quarter");
        } else if (state == SOLD_OUT) {
            System.out.println("You can't insert a quarter, the machine is sold out");
        } else if (state == SOLD) {
            System.out.println("Please wait, we're already giving you a gumball");
        }
    }
public void ejectQuarter() {
        if (state == HAS_QUARTER) {
            System.out.println("Quarter returned");
            state = NO_QUARTER;
        } else if (state == NO_QUARTER) {
            System.out.println("You haven't inserted a quarter");
        } else if (state == SOLD) {
            System.out.println("Sorry, you already turned the crank");
        } else if (state == SOLD_OUT) {
            System.out.println("You can't eject, you haven't inserted a quarter yet");
        }
    }
```

```
public void turnCrank() {
        if (state == SOLD) {
            System.out.println("Turning twice doesn't get you another gumball!");
        } else if (state == NO_QUARTER) {
            System.out.println("You turned but there's no quarter");
        } else if (state == SOLD_OUT) {
            System.out.println("You turned, but there are no gumballs");
        } else if (state == HAS_QUARTER) {
            System.out.println("You turned...");
            state = SOLD;
            dispense();
        }
    }
public void dispense() {
        if (state == SOLD) {
            System.out.println("A gumball comes rolling out the slot");
            count = count - 1;
            if (count == 0) {
                System.out.println("Oops, out of gumballs!");
                state = SOLD_OUT;
            } else {
                state = NO_QUARTER;
            }
        } else if (state == NO_QUARTER) {
            System.out.println("You need to pay first");
        } else if (state == SOLD_OUT) {
            System.out.println("No gumball dispensed");
        } else if (state == HAS_QUARTER) {
            System.out.println("No gumball dispensed");
        }
    }
```

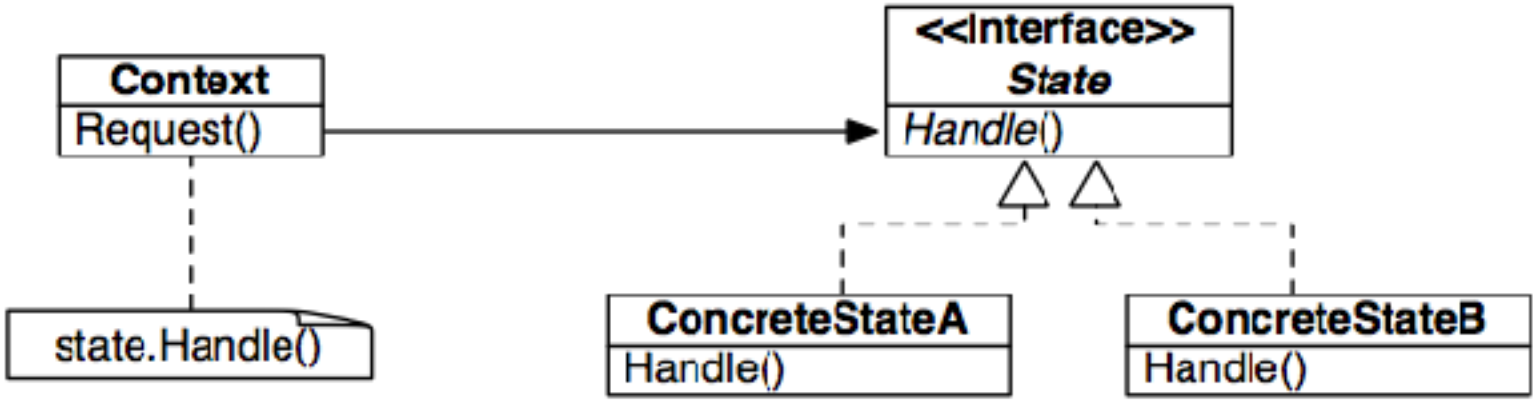
Problem

GumballMachine

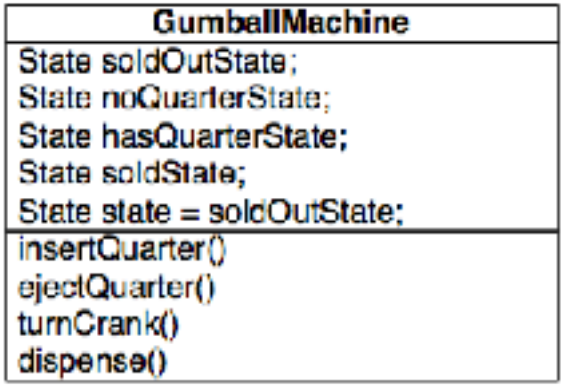
final static int SOLD_OUT = 0;
final static int NO_QUARTER = 1;
final static int HAS_QUARTER = 2;
final static int SOLD = 3;
int state = SOLD_OUT;

insertQuarter() ejectQuarter() turnCrank() dispense()

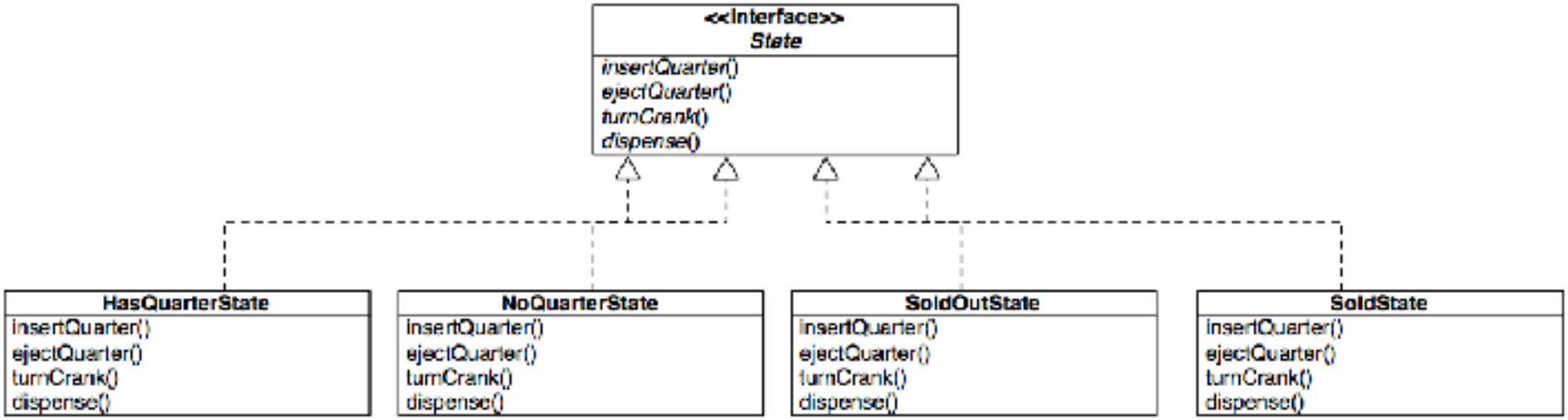
Abstract Solution



Solution




```
public void insertQuarter() {
        state.insertQuarter();
    }

    public void ejectQuarter() {
        state.ejectQuarter();
    }

    public void turnCrank() {
        state.turnCrank();
        state.dispense();
    }
```



State

Benefits and Liabilities

- (+) Reduces or removes state-changing conditional logic
- (+) Simplifies complex state-changing logic.
- (+) Provides a good bird's-eye view of state-changing logic.
- (-) Complicates a design when state transition logic is already easy to follow.

```
public List byColor(Color colorOfProductToFind) {
        List foundProducts = new ArrayList();
        Iterator<Product> products = null;
        while(products.hasNext()){
            Product product = (Product)products.next();
            if (product.getColor().equals(colorOfProductToFind)) {
                foundProducts.add(product);
            }
        }
        return foundProducts;
    }
public List byPrice(double priceLimit) {
        List foundProducts = new ArrayList();
        Iterator<Product> products = null;
        while(products.hasNext()){
            Product product = (Product)products.next();
            if (product.getPrice() == priceLimit) {
                foundProducts.add(product);
            }
        }
        return foundProducts;
    }
```

Problem

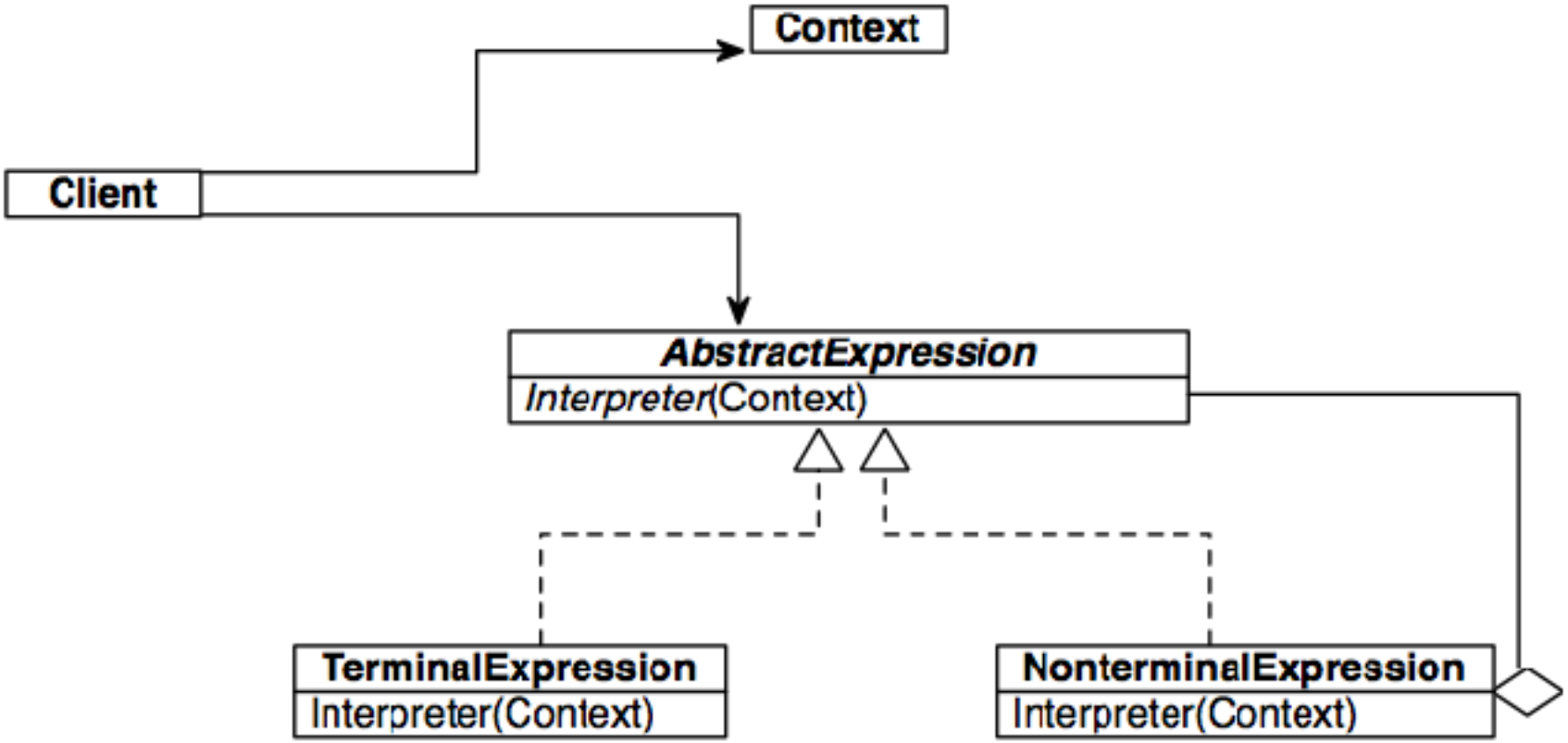
ProductFinder
byColor(Color) : List byPrice(double) : List byColorSizeAndBelowPrice(Color, ProductSize, double) : List belowPriceAvoidingColor(double, Color) : List

```
public List byColorSizeAndBelowPrice(Color color, ProductSize size, float price) {
        List foundProducts = new ArrayList();
        Iterator<Product> products = null;
        while(products.hasNext()){
            Product product = (Product)products.next();
            if (product.getColor() == color &&
                product.getSize() == size &&
                product.getPrice() < price) {
                foundProducts.add(product);
            }
        }
        return foundProducts;
    }
public List belowPriceAvoidingAColor(float price, Color color) {
        List foundProducts = new ArrayList();
        Iterator<Product> products = null;
        while(products.hasNext()){
            Product product = (Product)products.next();
            if (product.getColor() != color &&
                product.getPrice() < price) {
                foundProducts.add(product);
            }
        }
        return foundProducts;
    }
```

Problem

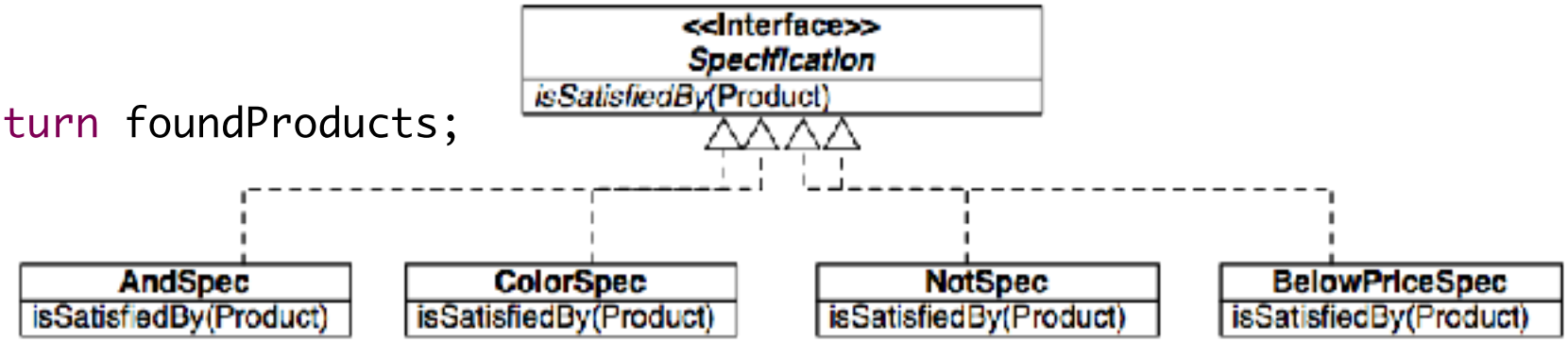
ProductFinder
byColor(Color) : List
byPrice(double) : List
byColorSizeAndBelowPrice(Color, ProductSize, double) : List
belowPriceAvoidingColor(double, Color) : List

Abstract Solution



Solution

```
public class ProductFinder {

    private ProductRepository repository;

    public ProductFinder(ProductRepository repository) {
        this.repository = repository;
    }

    public List selectBy(Specification spec) {
        List foundProducts = new ArrayList();
        Iterator products = null;

        while(products.hasNext()){
            Product product = (Product)products.next();
            if (spec.isSatisfiedBy(product)) {
                foundProducts.add(product);
            }
        }

        return foundProducts;
    }
}
```

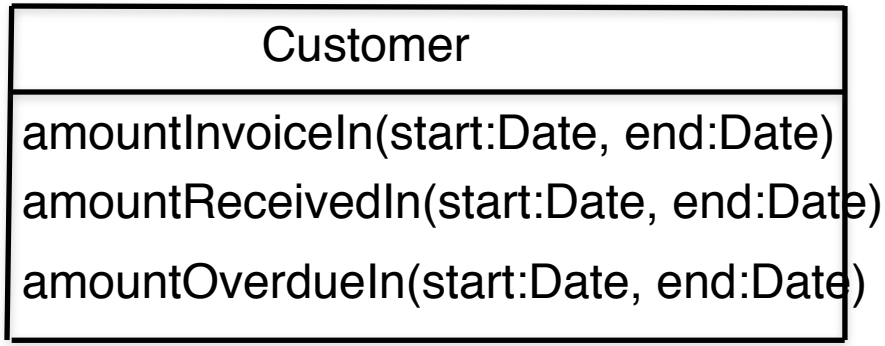


Interpreter

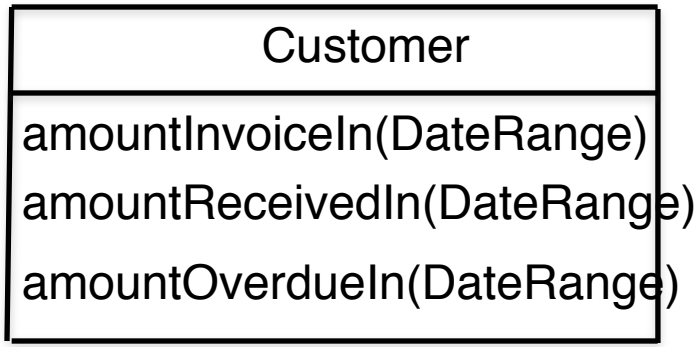
Benefits and Liabilities

- (+) Supports combination of language elements better than an implicit language does.
- (+) Requires no new code to support new combinations of language elements
- (+) Allows for runtime configuration of behavior
- (-) Has a start-up cost for defining a grammar and changing client code to use it
- (-) Requires too much programming when your language is complex
- (-) Complicates a design when a language is simple

Problem

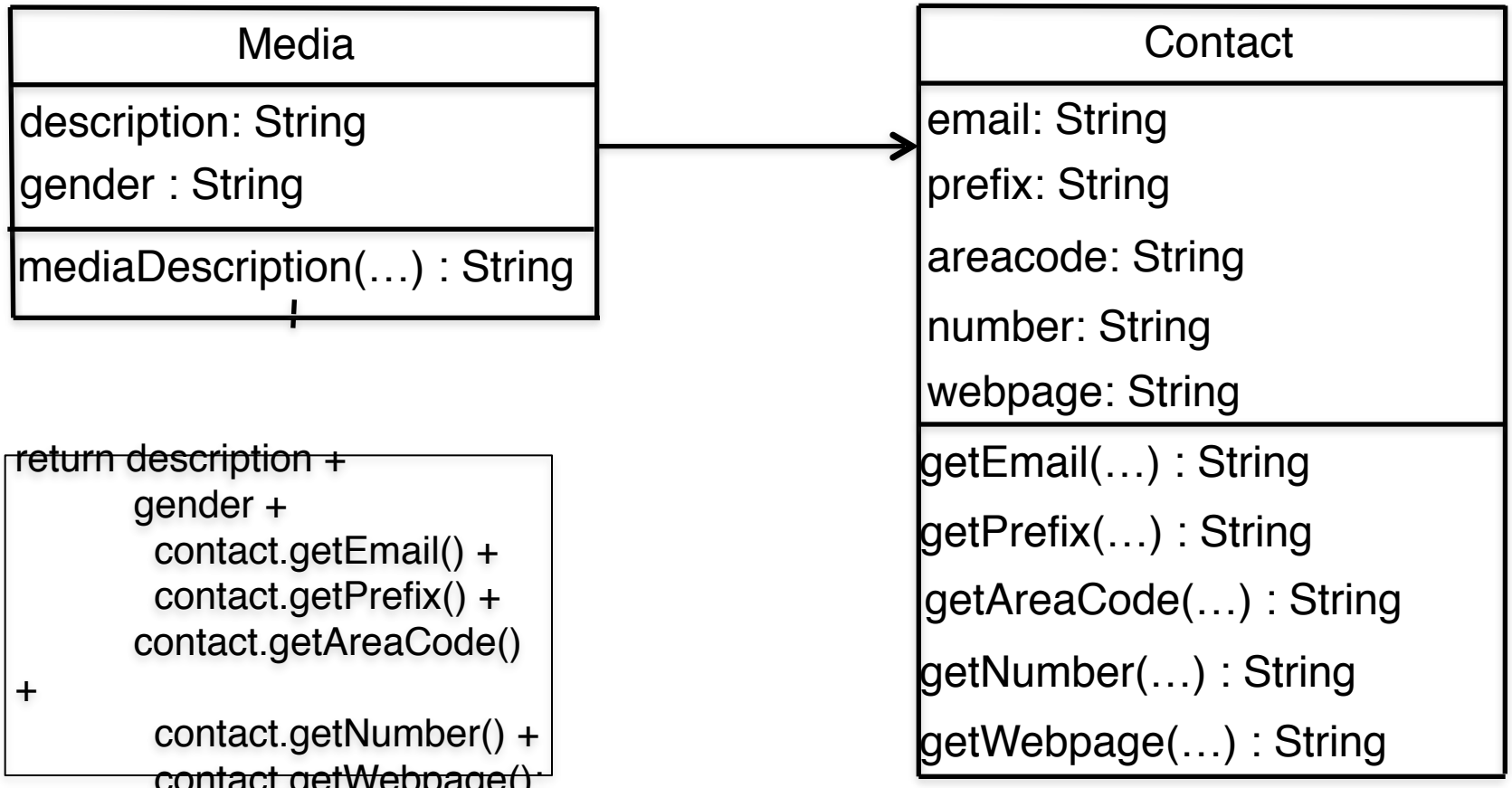

Customer
amountInvoiceIn(start:Date, end:Date)
amountReceivedIn(start:Date, end:Date)
amountOverdueIn(start:Date, end:Date)

Solution

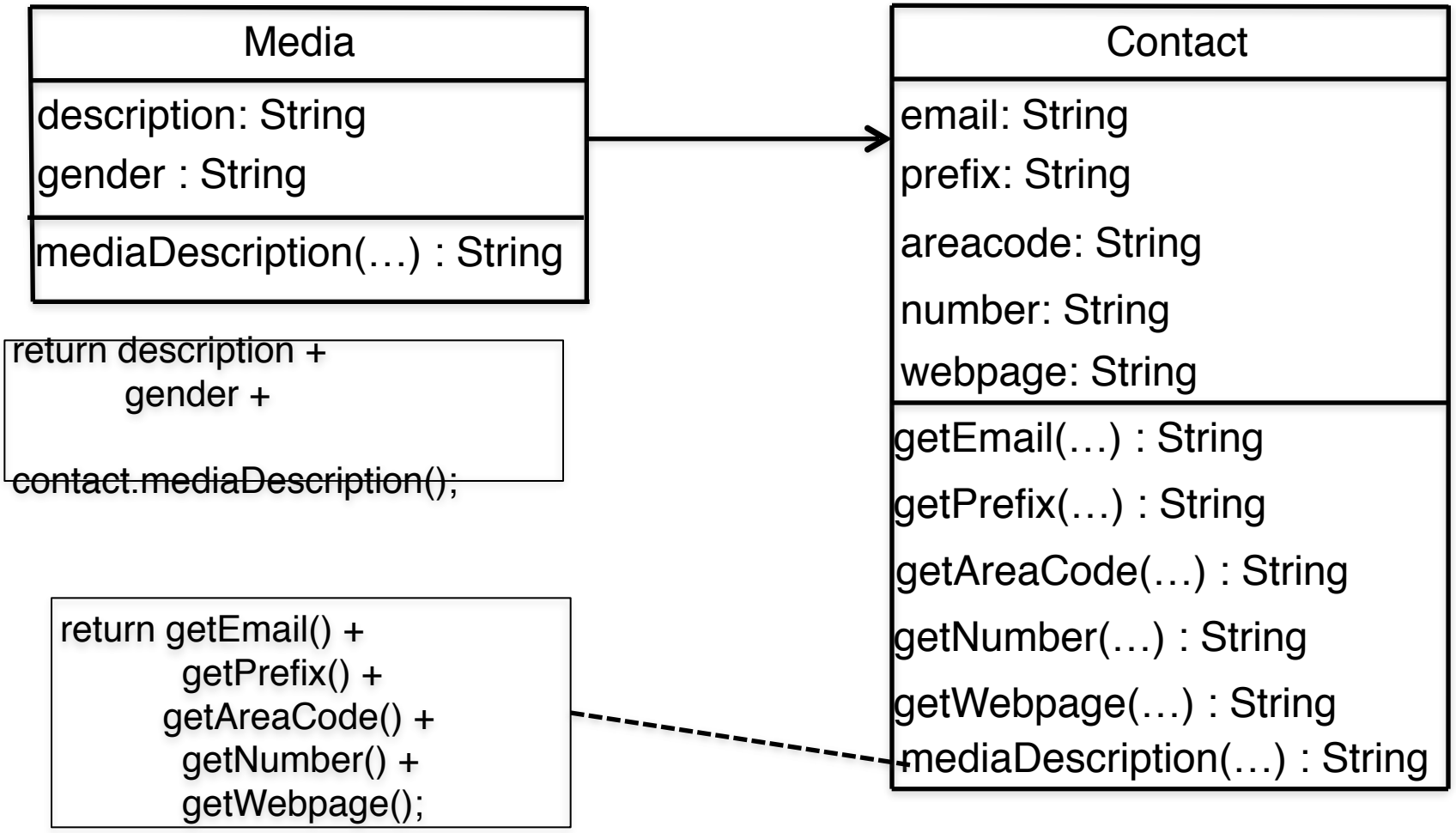

Customer
amountInvoiceIn(DateRange)
amountReceivedIn(DateRange)
amountOverdueIn(DateRange)

Introduce Parameter Object

Problem


Media
description: String
gender : String
mediaDescription(…) : String
Contact
email: String
prefix: String
areacode: String
number: String
webpage: String
getEmail(…) : String
getPrefix(…) : String
getAreaCode(…) : String
getNumber(…) : String
getWebpage(…) : String
return description +
gender +
contact.getEmail() +
contact.getPrefix() +
contact.getAreaCode()
+
contact.getNumber() +
contact.getWebpage();

Solution


Media
description: String
gender : String
mediaDescription(…) : String
Contact
email: String
prefix: String
areacode: String
number: String
webpage: String
getEmail(…) : String
getPrefix(…) : String
getAreaCode(…) : String
getNumber(…) : String
getWebpage(…) : String
mediaDescription(…) : String
return description +
gender +
contact.mediaDescription();
return getEmail() +
getPrefix() +
getAreaCode() +
getNumber() +
getWebpage();

Move Method

Problem

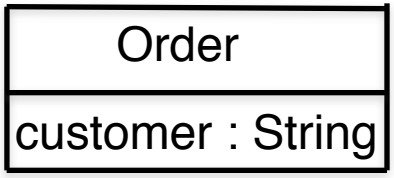
```
String[] row = new
String[3];
row[0] = “Liverpool”;
row[1] = “15”;
```

Solution

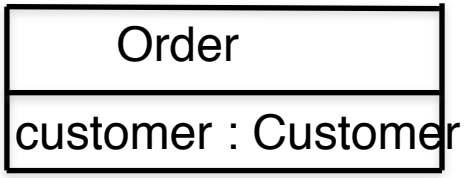
```
Performance row = new Performance();
row.setName(“Liverpool”);
row.setWins(“15”);
```

Replace Array with Object

Problem



Solution



Replace Data Value with Object

Problem

```
int getRating(){
   return (moreThanFiveLateDeliveries()) ? 2
: 1;
}
Boolean moreThanFiveLateDeliveries(){
   return _numberOfLateDeliveries > 5;
}
```

Solution

```
int getRating(){
   return (_numberOfLateDeliveries > 5) ?
2 : 1;
}
```

Inline Method