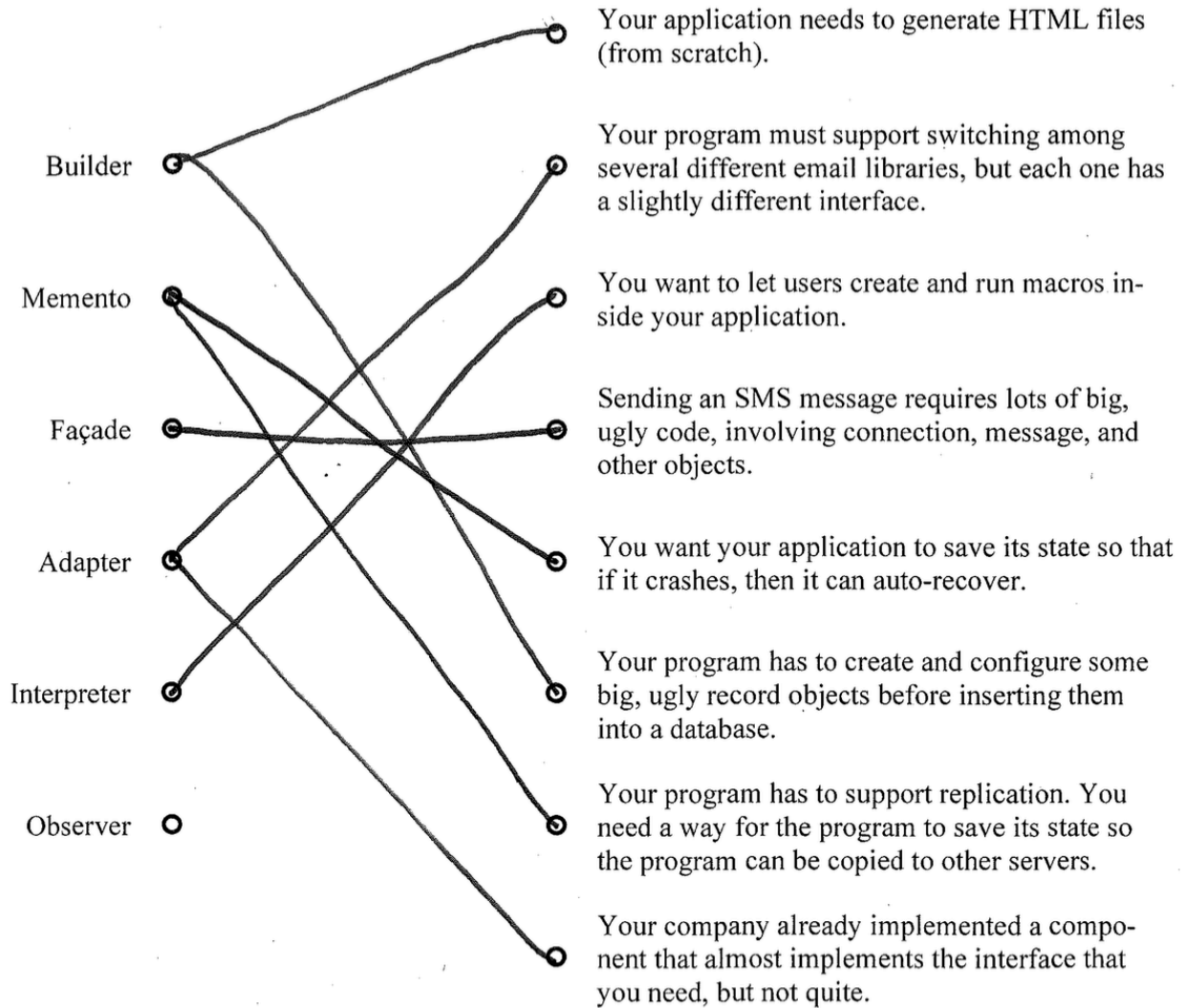


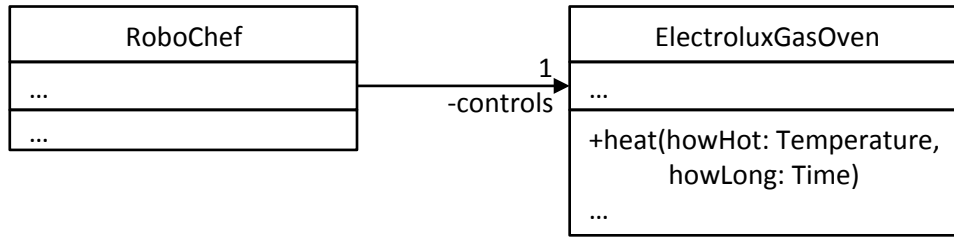
Problem: Match the design pattern to the situation to which you should apply it.

- | | | |
|-------------|-----------------------|--|
| | <input type="radio"/> | Your application needs to generate HTML files (from scratch). |
| Builder | <input type="radio"/> | Your program must support switching among several different email libraries, but each one has a slightly different interface. |
| Memento | <input type="radio"/> | You want to let users create and run macros inside your application. |
| Façade | <input type="radio"/> | Sending an SMS message requires lots of big, ugly code, involving connection, message, and other objects. |
| Adapter | <input type="radio"/> | You want your application to save its state so that if it crashes, then it can auto-recover. |
| Interpreter | <input type="radio"/> | Your program has to create and configure some big, ugly record objects before inserting them into a database. |
| Observer | <input type="radio"/> | Your program has to support replication. You need a way for the program to save its state so the program can be copied to other servers. |
| | <input type="radio"/> | Your company already implemented a component that almost implements the interface that you need, but not quite. |

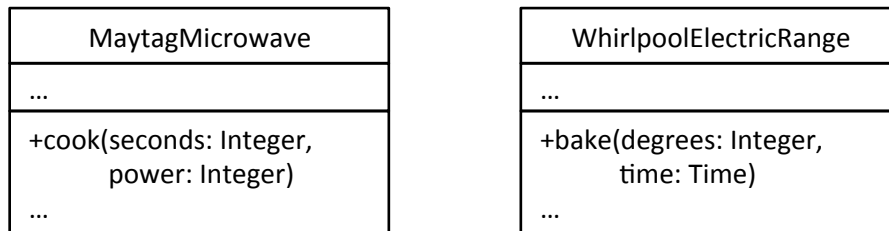
Solution:



Problem: Imagine that you are the creator of an “intelligent” kitchen system, RoboChef, that can actually control different kitchen appliances (e.g., ovens, choppers) to prepare food. Initially, you implemented RoboChef to use only Electrolux gas ovens. Here is an excerpt of your current software design:



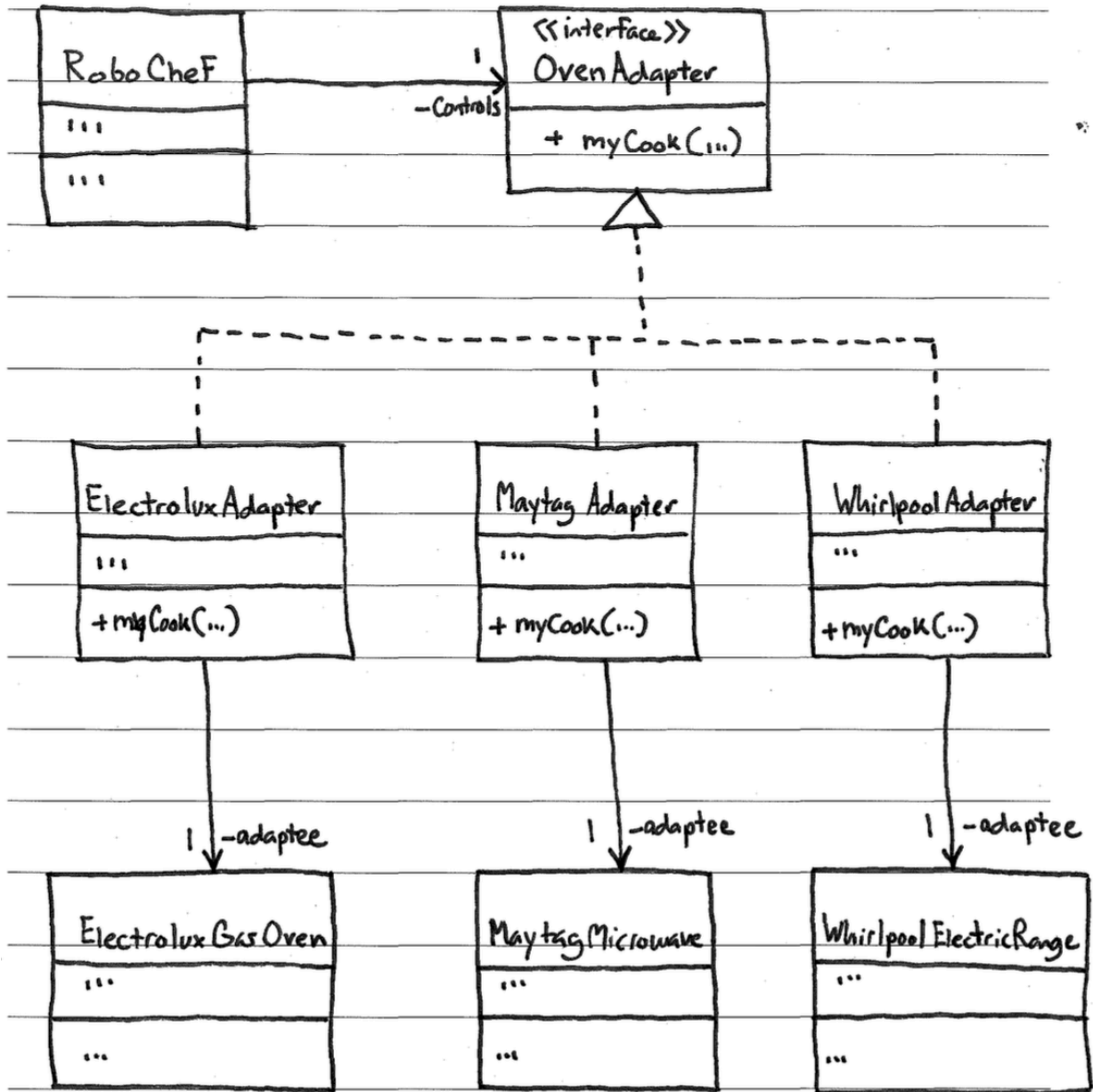
Note that the Electrolux Company provided the software interface for controlling the gas oven (ElectroluxGasOven), and you created the intelligent decision-making part (RoboChef). As your next step, you would like your system to support different types of ovens other than Electrolux gas ones. For example, Maytag and Whirlpool each provide their own software interfaces for their ovens:



Update your current software design to allow easy switching between oven-control systems. Your design must apply the **adapter pattern**.

Draw a class diagram for your design.

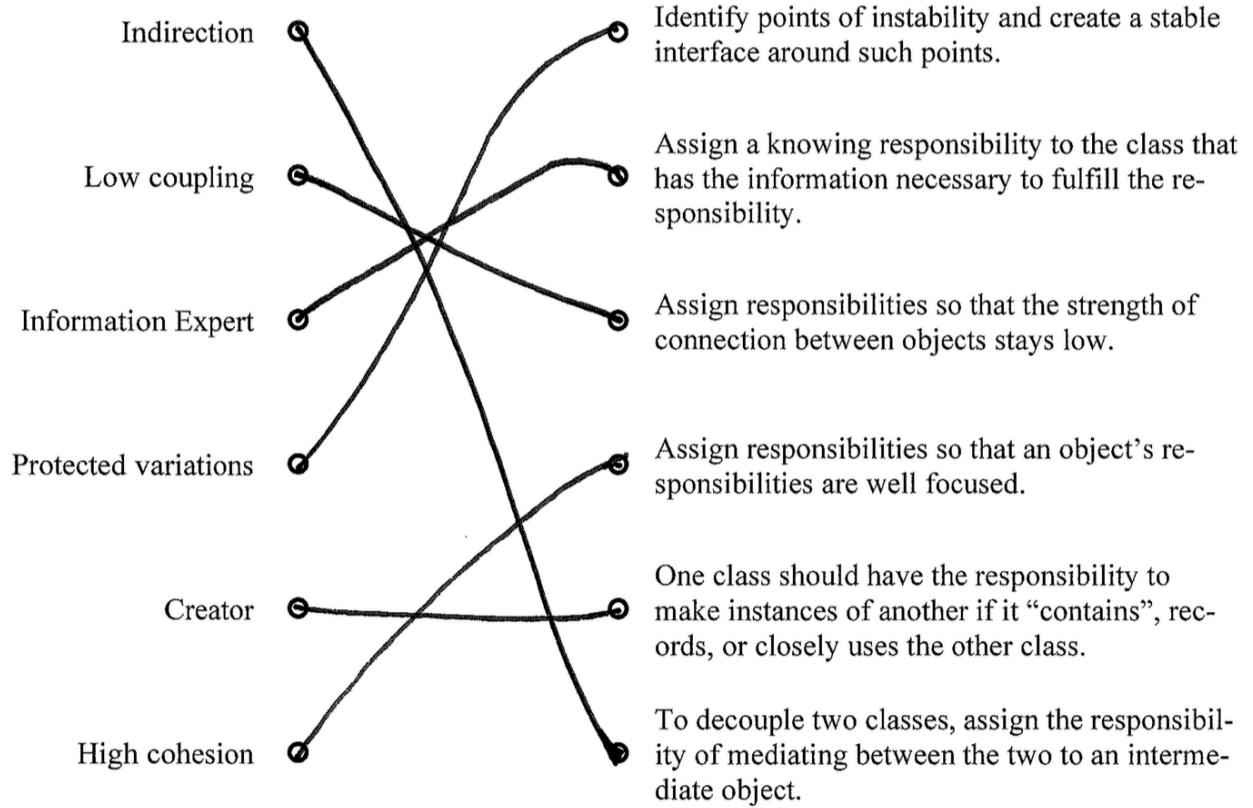
Solution:



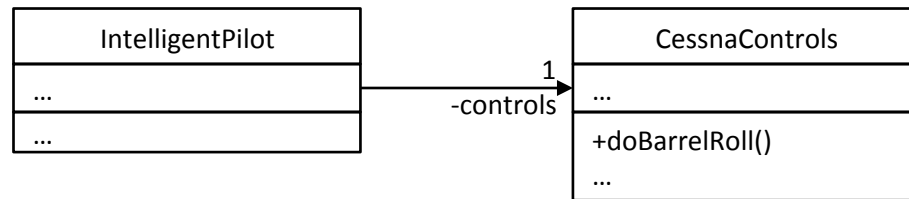
Problem: For each pattern below, draw a line from the pattern to its definition.

- | | | | |
|----------------------|-----------------------|-----------------------|---|
| Indirection | <input type="radio"/> | <input type="radio"/> | Identify points of instability and create a stable interface around such points. |
| Low coupling | <input type="radio"/> | <input type="radio"/> | Assign a knowing responsibility to the class that has the information necessary to fulfill the responsibility. |
| Information Expert | <input type="radio"/> | <input type="radio"/> | Assign responsibilities so that the strength of connection between objects stays low. |
| Protected variations | <input type="radio"/> | <input type="radio"/> | Assign responsibilities so that an object's responsibilities are well focused. |
| Creator | <input type="radio"/> | <input type="radio"/> | One class should have the responsibility to make instances of another if it "contains", records, or closely uses the other class. |
| High cohesion | <input type="radio"/> | <input type="radio"/> | To decouple two classes, assign the responsibility of mediating between the two to an intermediate object. |

Solution:

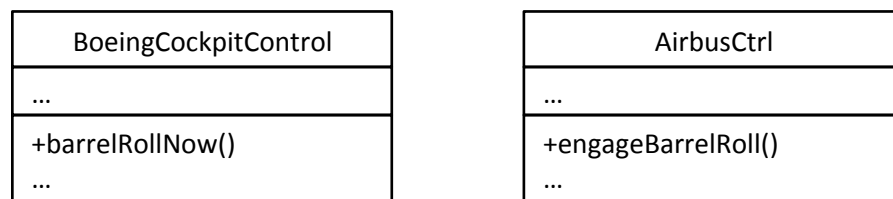


Problem: Imagine that you are the creator of an “intelligent” autopilot system that can actually fly and land real airplanes (wow!). Initially, you implemented your system to fly small Cessna airplanes. Here is an excerpt of your current software design:



Note that the Cessna Aircraft Company provided the software interface for controlling the plane (CessnaControls), and you created the intelligent decision-making part (IntelligentPilot).

As your next step, you would like your system to support different types of airplanes other than Cessnas. For example, Boeing and Airbus each provide their own software control interfaces:



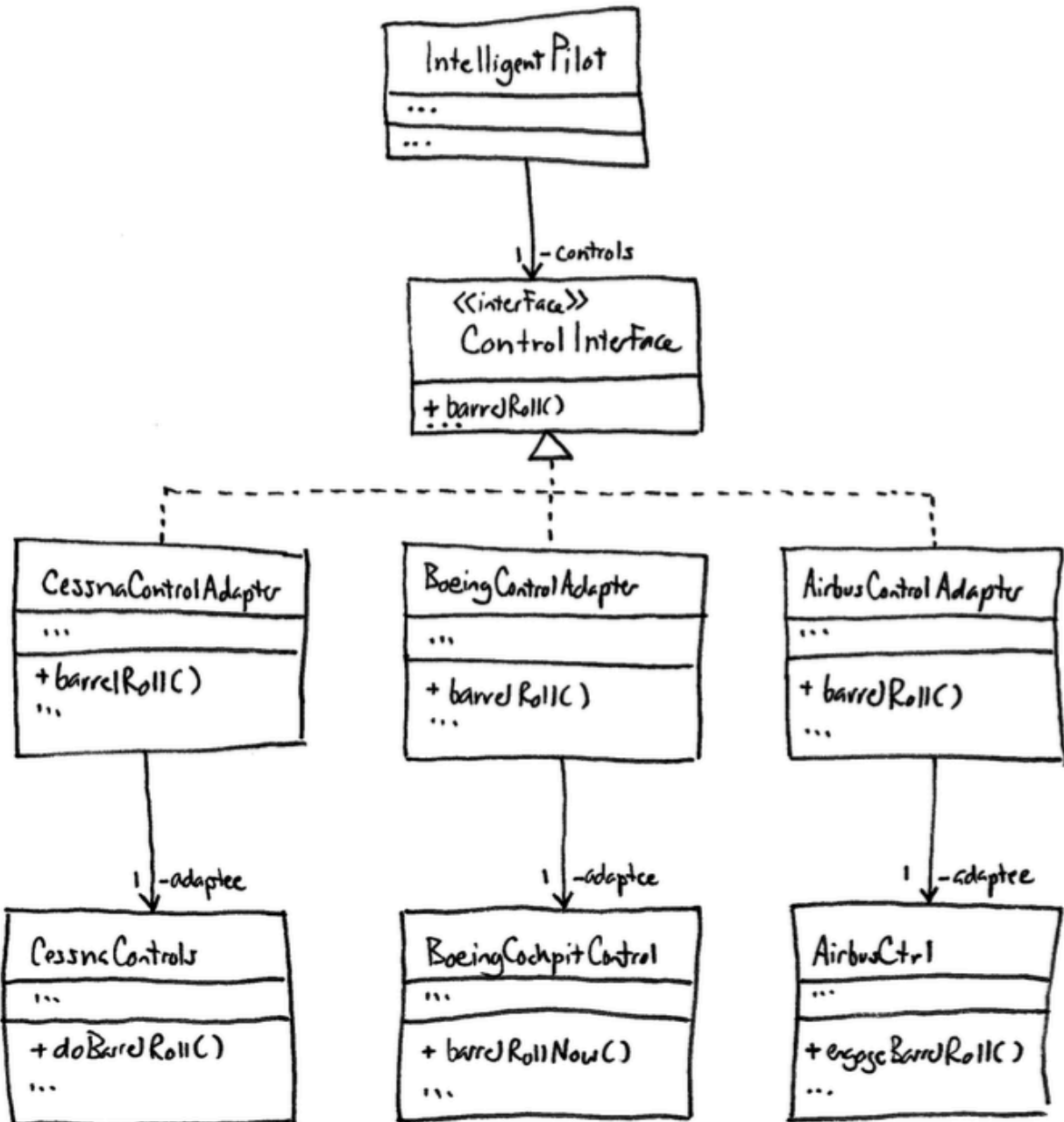
Update your current software design to allow easy switching between control systems. Your design must apply the **adapter pattern**.

Draw a class diagram for your design.

What effect did your new design have on the coupling between class IntelligentPilot and class CessnaControls.

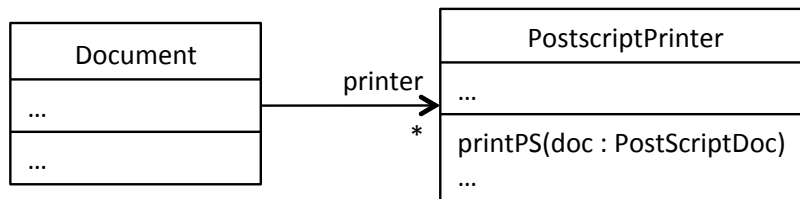
- Reduced their coupling
- Increased their coupling
- Had no effect on their coupling

Solution:

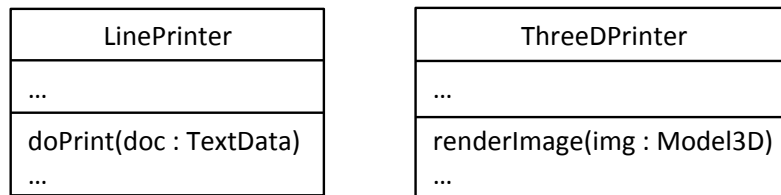


- a. Reduced their coupling
- b. Increased their coupling
- c. Had no effect on their coupling

Problem: Consider the following design for a document-editing system. The Document class represents a document, and Document objects know how to print themselves using a PostscriptPrinter object.



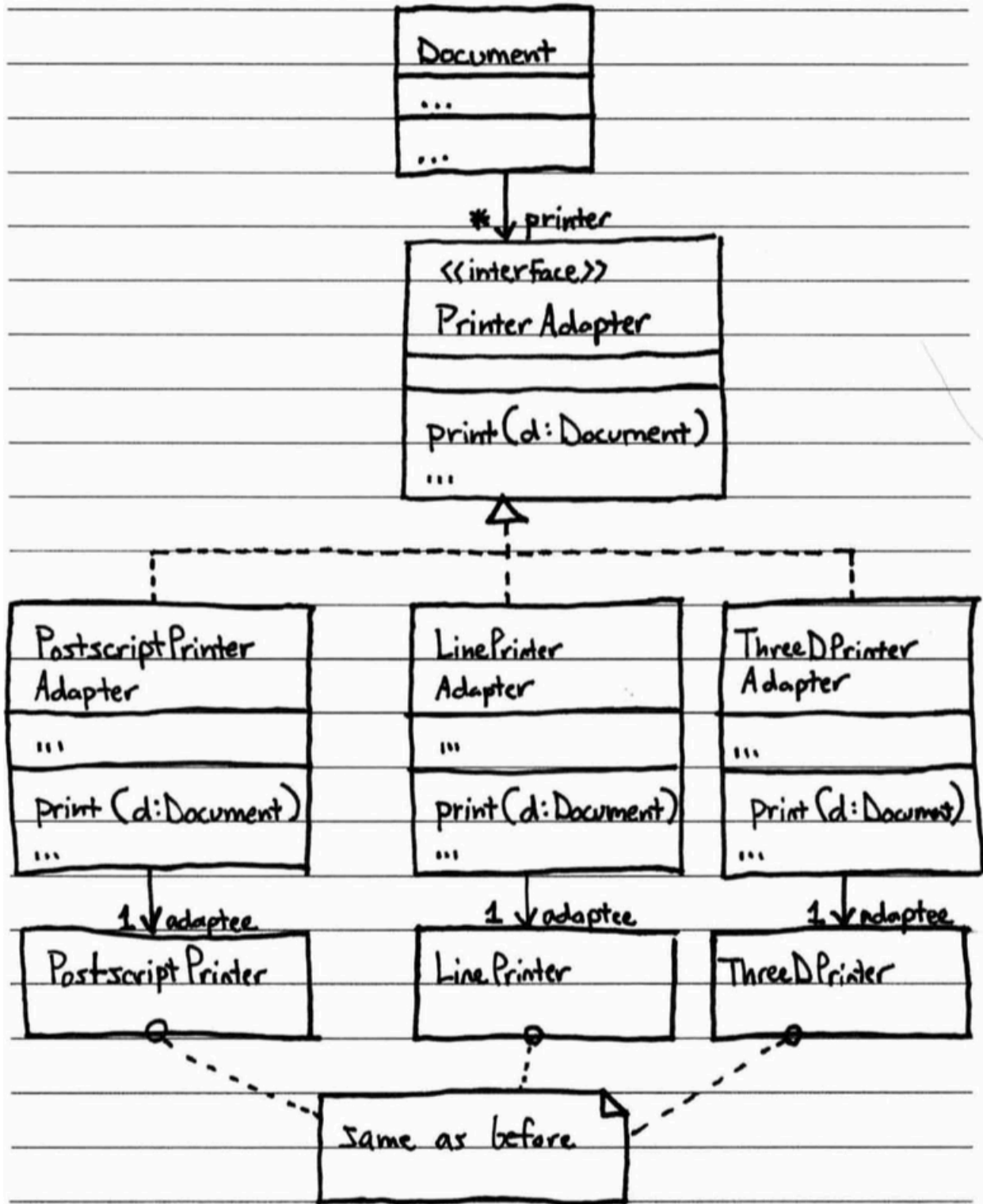
However, there are other types of printers that a document might want to print itself on, but these printers have slightly different interfaces than the Postscript printer, for example:



Using the **adapter pattern**, refactor the design, so that the different types of printers can be easily swapped in and out.

Draw a design class diagram for your design.

Solution:










Problem:

Match the design pattern to the situation to which you should apply it.

- Observer ○
 - Your Pac-Man program needs to listen for presses of the arrow keys and to update Pac-Man's position in the maze accordingly.
- Builder ○
 - Your program has to create and configure some big, ugly record objects before inserting them into a database.
- Adapter ○
 - Your GUI interface has many interrelated buttons and other widgets (e.g., such that when each button is pressed many other widgets must be updated).
- Mediator ○
 - You want your application to save its state so that if it crashes, then it can auto-recover.
- Memento ○
 - Your program must support switching among several different email libraries, but each one has a slightly different interface.
- Interpreter ○
 - You want to let users create and run macros inside your application.
- Facade ○
 - Sending an SMS message requires lots of big, ugly code, involving connection, message, and other objects.

Solution:

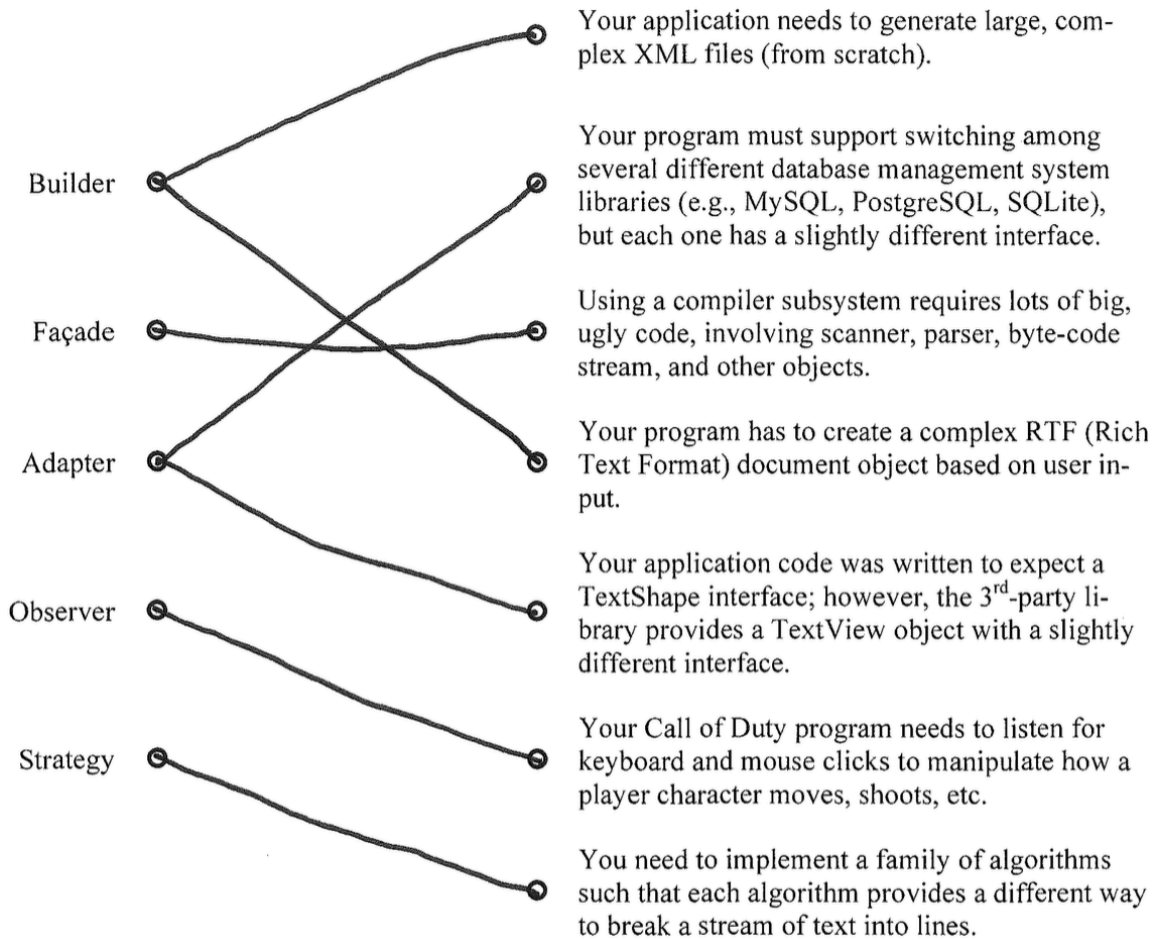
- Observer  Your Pac-Man program needs to listen for presses of the arrow keys and to update Pac-Man's position in the maze accordingly.
- Builder  Your program has to create and configure some big, ugly record objects before inserting them into a database.
- Adapter  Your GUI interface has many interrelated buttons and other widgets (e.g., such that when each button is pressed many other widgets must be updated).
- Mediator  You want your application to save its state so that if it crashes, then it can auto-recover.
- Memento  Your program must support switching among several different email libraries, but each one has a slightly different interface.
- Interpreter  You want to let users create and run macros inside your application.
- Facade  Sending an SMS message requires lots of big, ugly code, involving connection, message, and other objects.

Problem:

Match the design pattern to the situation to which you should apply it.

- | | |
|--------------------------------|---|
| Builder <input type="radio"/> | <input type="radio"/> Your application needs to generate large, complex XML files (from scratch). |
| Façade <input type="radio"/> | <input type="radio"/> Your program must support switching among several different database management system libraries (e.g., MySQL, PostgreSQL, SQLite), but each one has a slightly different interface.
<input type="radio"/> Using a compiler subsystem requires lots of big, ugly code, involving scanner, parser, byte-code stream, and other objects. |
| Adapter <input type="radio"/> | <input type="radio"/> Your program has to create a complex RTF (Rich Text Format) document object based on user input. |
| Observer <input type="radio"/> | <input type="radio"/> Your application code was written to expect a TextShape interface; however, the 3 rd -party library provides a TextView object with a slightly different interface. |
| Strategy <input type="radio"/> | <input type="radio"/> Your Call of Duty program needs to listen for keyboard and mouse clicks to manipulate how a player character moves, shoots, etc.
<input type="radio"/> You need to implement a family of algorithms such that each algorithm provides a different way to break a stream of text into lines. |

Solution:



Multiple-Choice Questions:

1. Which of the following describes the Adapter Pattern?
 - a. Builds a complex object using simple objects and using a step by step approach
 - b. Creates a duplicate object while keeping performance in mind
 - c. Works as a bridge between two incompatible interfaces
 - d. Used to decouple an abstraction from its implementation so that the two can vary independently
 - e. None of the above

2. Which of the following is true about design patterns?
 - a. Represent the best practices used by experienced object-oriented software developers
 - b. Solutions to general problems that developers commonly face during software development
 - c. Obtained by trial and error of numerous software developers over a substantial period of time
 - d. All of the above
 - e. None of the above

3. Which of the following is true about the Singleton Pattern?
 - a. A “creational” pattern
 - b. Responsible for ensuring that no more than one instance of a particular class is created
 - c. Provides a way to create an instance of a class without directly calling the class’ constructor
 - d. All of the above
 - e. None of the above

4. Which pattern automatically notifies dependent objects when a subject object is modified?
 - a. Adapter
 - b. Observer
 - c. Singleton
 - d. Memento
 - e. None of the above

5. Which pattern allows incompatible classes to work together by converting the interface of one class into an interface expected by client?
 - a. Observer
 - b. Builder
 - c. Adapter
 - d. Memento
 - e. None of the above

Solutions:

1. c

2. d

3. d

4. b

5. c

Problem:

```
1 require "observer"
2 class SMSNotifier
3   def _____ 1 _____
4
5     if bank_account.balance <= 10
6       #Send SMS notifying owner
7     end
8
9   end
10 end
11
12 class BankAccount
13   _____ 2 _____
14
15   attr_reader :owner, :balance
16
17   def _____ 3 _____
18
19     @owner,@balance = owner,amount
20
21     _____ 4 _____
22
23
24   end
25   # withdraw operation
26   def _____ 5 _____
27     #perform action
28     @balance -=amount if(@balance - amount) > 0
29
30     _____ 6 _____
31
32     _____ 7 _____
33
34   end
35 end
36
37 account = BankAccount.new "Liza", 100
38 account.withdraw 95
```

In the above code, if a customer's balance drops to 10 or less, an SMSNotifier notifies the customer about the low bank balance. For each code fragment below, tell where it belongs above.

- a. _____ add_observer SMSNotifier.new
- b. _____ include Observable
- c. _____ withdraw(amount)
- d. _____ notify_observers self
- e. _____ initialize(owner,amount)
- f. _____ update(bank_account)
- g. _____ changed

Solution:

a. 4

b. 2

c. 5

d. 7

e. 3

f. 1

g. 6

Problem:

```
require "observer"

class 
  

  def initialize
    
  end

  def create
    if @quizSubmit.save
      
      

      format.html { redirect_to @quizSubmit, notice: 'Reponse recorded successfully.' }
      format.json { render :show, status: :created, location: @quizSubmit }
    else
      format.html { render :new }
      format.json { render json: @quizSubmit.errors, status: :unprocessable_entity }
    end
  end
end

class 
  def update()
    # Calculates result of the student and display on the results page
  end
end
```

In the above auto-tutor app, when a student submits a quiz, a ResultCalculator calculates the outcome and displays it. For each code fragment below, tell where it belongs above.

- a. changed
- b. QuizController
- c. add_observer ResultCalculator.new
- d. include Observable
- e. notify_observers self
- f. ResultCalculator

Solution:

a. 4

b. 1

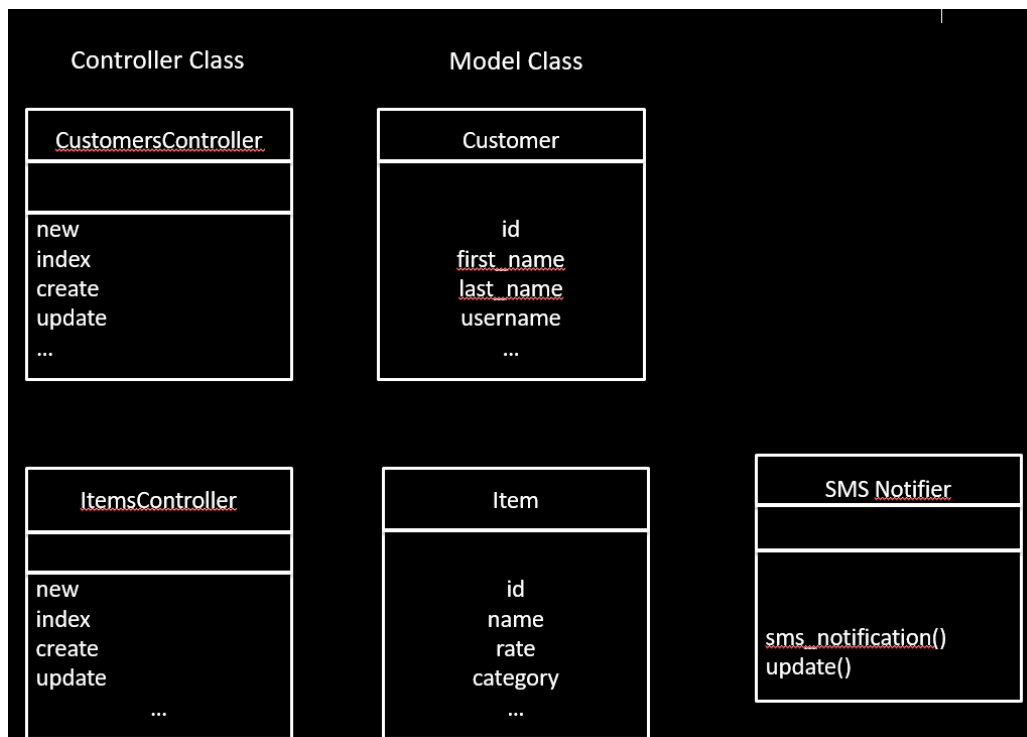
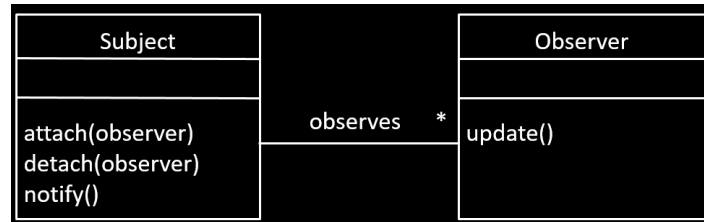
c. 3

d. 2

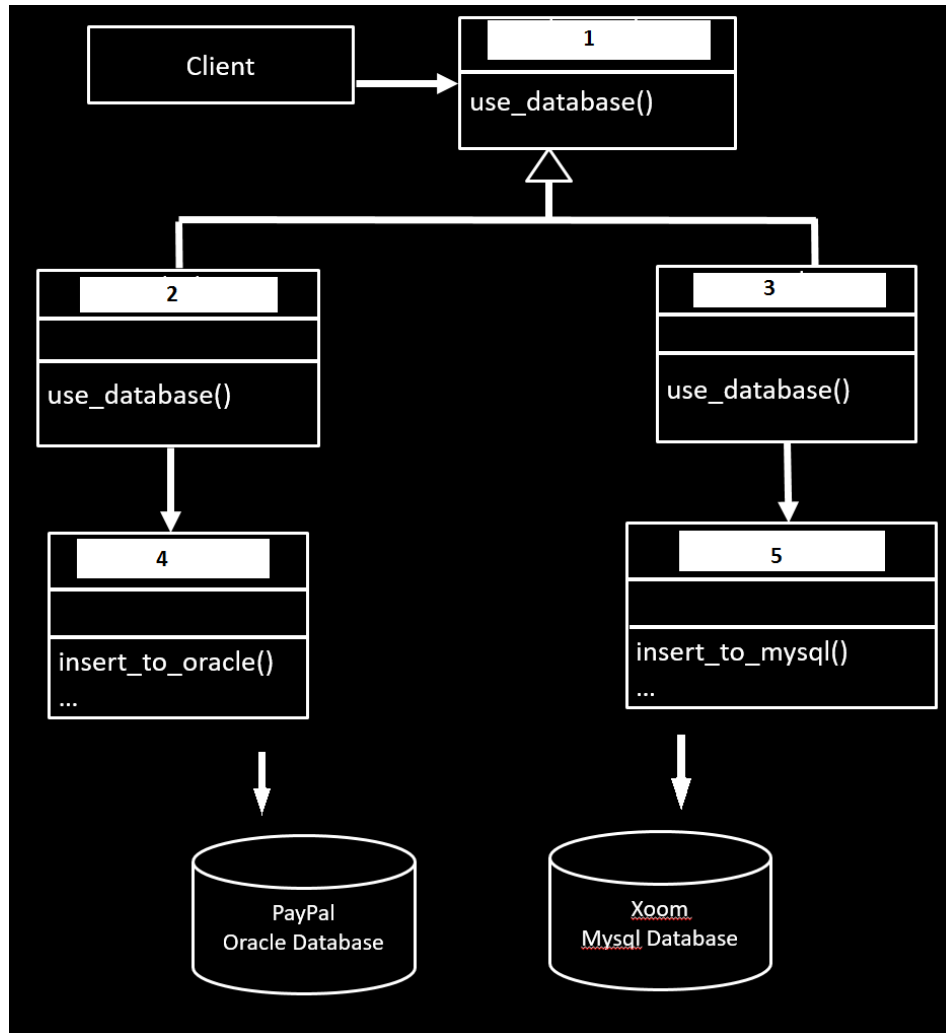
e. 5

f. 6

Multiple-Choice Questions:



1. When a customer visits a product on eBay and searches the product multiple times, eBay tracks the search information. Based on this information, eBay sends an SMS notification whenever the product's cost goes down, or there is a discount available for it. Given the Observer design pattern at top, which classes in the diagram below would play the roles of Subject and Observer. (Hint: the system notifies the customer if there is a change in the cost to the item).
 - a. Subject: CustomersController and Observer: Item
 - b. Subject: ItemsController and Observer: SMSNotifier
 - c. Subject: ItemsController and Observer: Item
 - d. Subject: CustomersController and Observer: SMSNotifier
 - e. None of the above



2. A merger happened between two companies, PayPal and Xoom. PayPal used a PostgreSQL database, whereas Xoom used a MySQL one. Although both types of database have similar interfaces, they are not quite the same. Given the partially elided class diagram above that applies the Adapter Pattern to solve this problem, which of the below would be the correct assignment of classes?
- 1: DatabaseSelector, 2: PayPalAdapter, 3: XoomAdapter, 4: PayPalAdaptee, 5: XoomAdaptee
 - 1: DatabaseSelector, 2: PayPalAdaptee, 3: XoomAdaptee, 4: PayPalAdapter, 5: XoomAdapter
 - 1: DatabaseSelector, 2: PayPalAdapter, 3: XoomAdapter, 4: XoomAdaptee, 5: PayPalAdaptee
 - 1: DatabaseSelector, 2: XoomAdaptee, 3: PayPalAdaptee, 4: PayPalAdapter, 5: XoomAdapter
 - None of the above

Solutions:

1. b

2. a