Question Group #1

1. [2pts] If your project has unstable requirements (i.e., that are prone to change), you should use a waterfall process model.

   a. True

   b. False

2. [2pts] In iterative development, how long should an iteration generally be?

   a. 1 week

   b. 2–6 weeks

   c. 2–4 months

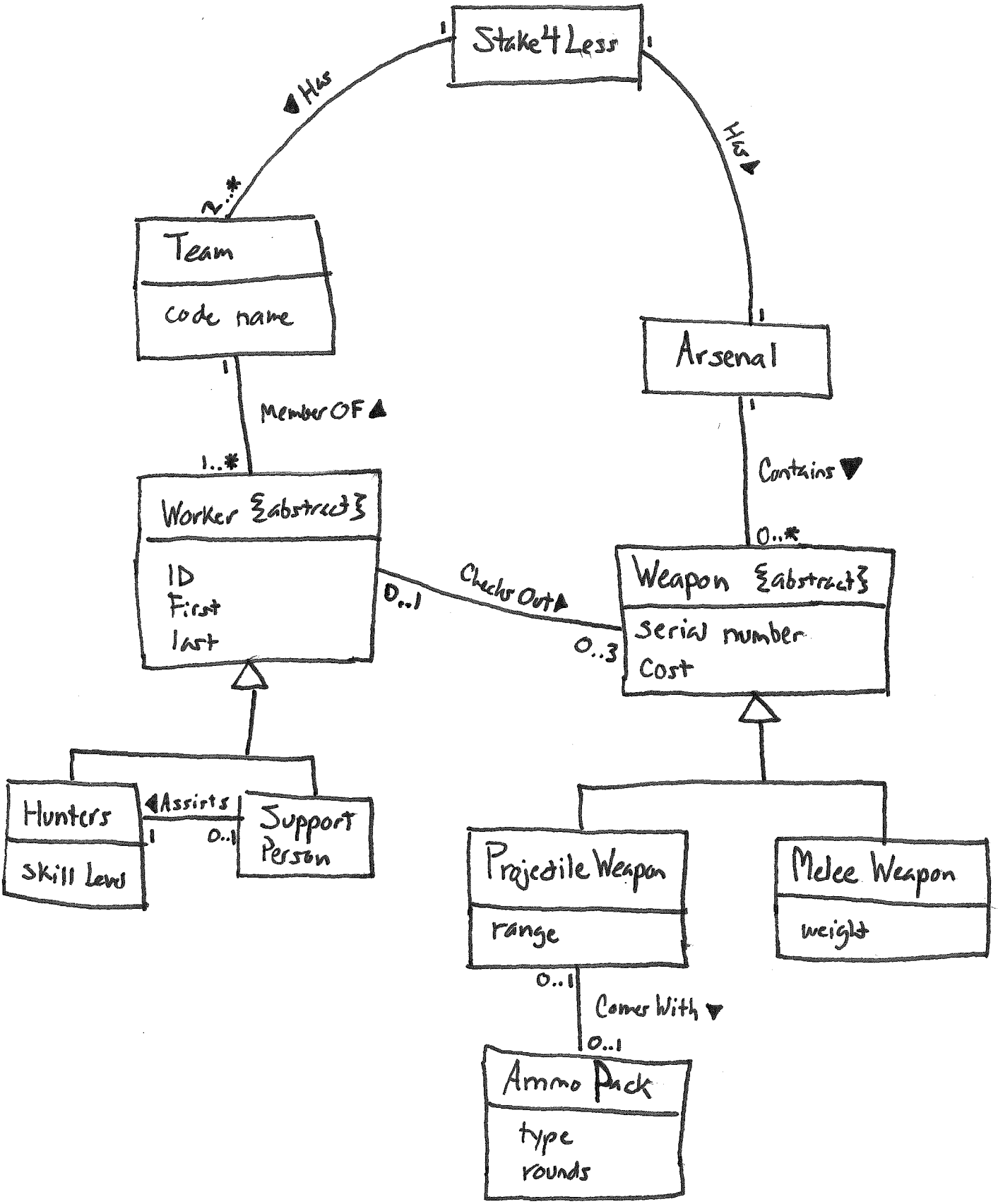
   d. 6 months to a year

   e. None of the above

3. [12pts] On the next page, create a domain model (using class diagram notation) based on the following description.

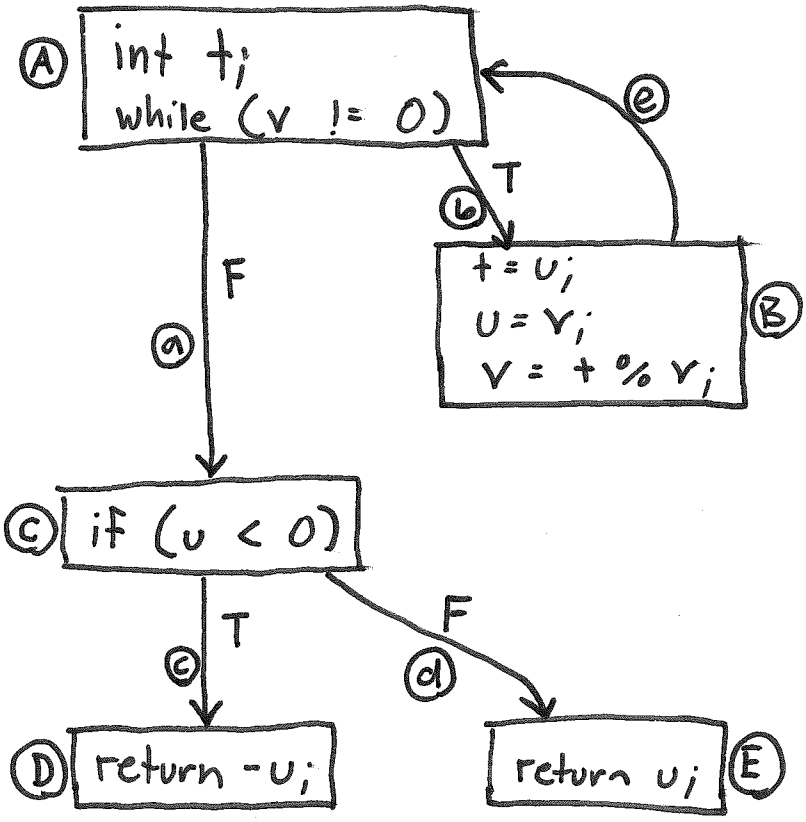
   You have been contracted to build an asset-management system for a vampire-hunting company, *Stake4Less*. The organization consists of multiple teams of workers. Each team has a code name. Each worker belongs to one of the teams, and has an ID number, a first name, and a last name. There are two types of workers: hunters and support personnel. Hunters have a skill level. Each support person assists exactly one hunter on the team. Some hunters on the team may not be assigned a support person. The company also manages an arsenal of weapons. Workers of any type may check out up to 3 weapons from the arsenal. Each weapon has a serial number and replacement cost. There are two types of weapons in the arsenal: projectile and melee. Projectile weapons have a range, and may come with an ammo pack. An ammo pack has an ammo type, and includes some number of rounds. Melee weapons have a weight.

(Write your answer here.)



4. [8pts] Draw a control flow diagram for this function. Label each node in the graph with a capital letter, and label each edge with a lowercase letter.

```
int blammo(int u, int v) {
  int t;
  while (v != 0) {
    t = u;
    u = v;
    v = t % v; // Recall that % computes remainder of t/v
  }
  if (u < 0) { return -u; }
  return u;
}
```



5. [5pts] Fill in the table below with a test suite that provides statement coverage of the code from question 4. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Covers
u	v	
2	2	A, B, C, E
-1	0	A, C, D

6. [5pts] Fill in the table below with a test suite that provides path coverage of the code from question 4. Cover no more than 1 iteration of the loop. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Covers
u	v	
-1	0	a, c
0	0	a, d
-2	-2	b, e, a, c
2	2	b, e, a, d

Paths:

a, c
a, d
b, e, a, c
b, e, a, d

7. [2pts] Which of the following best exemplifies the single responsibility principle?

   a. Writing test code before you write the code under test

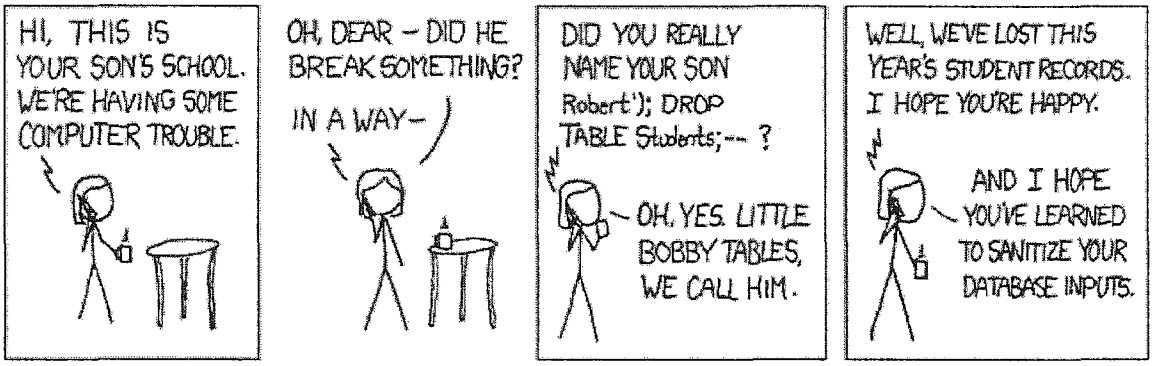
   b. Creating mock objects to stand in for other objects in the system

   c. Dividing each user story to be built next iteration into tasks and assigning each task to a developer on the team

   (d.) For objects of a class *C*, creating a DAO (which knows how to read/write *C* objects to a database) instead of putting all that database accessing logic in the *C* class

   e. Collecting feedback from the customer at the end of each iteration, instead of waiting until the system is finally delivered

8. [2pts] What type of attack did the parents in this XKCD comic perform?



   a. Cross-site scripting

   (b.) SQL injection

   c. Child endangerment

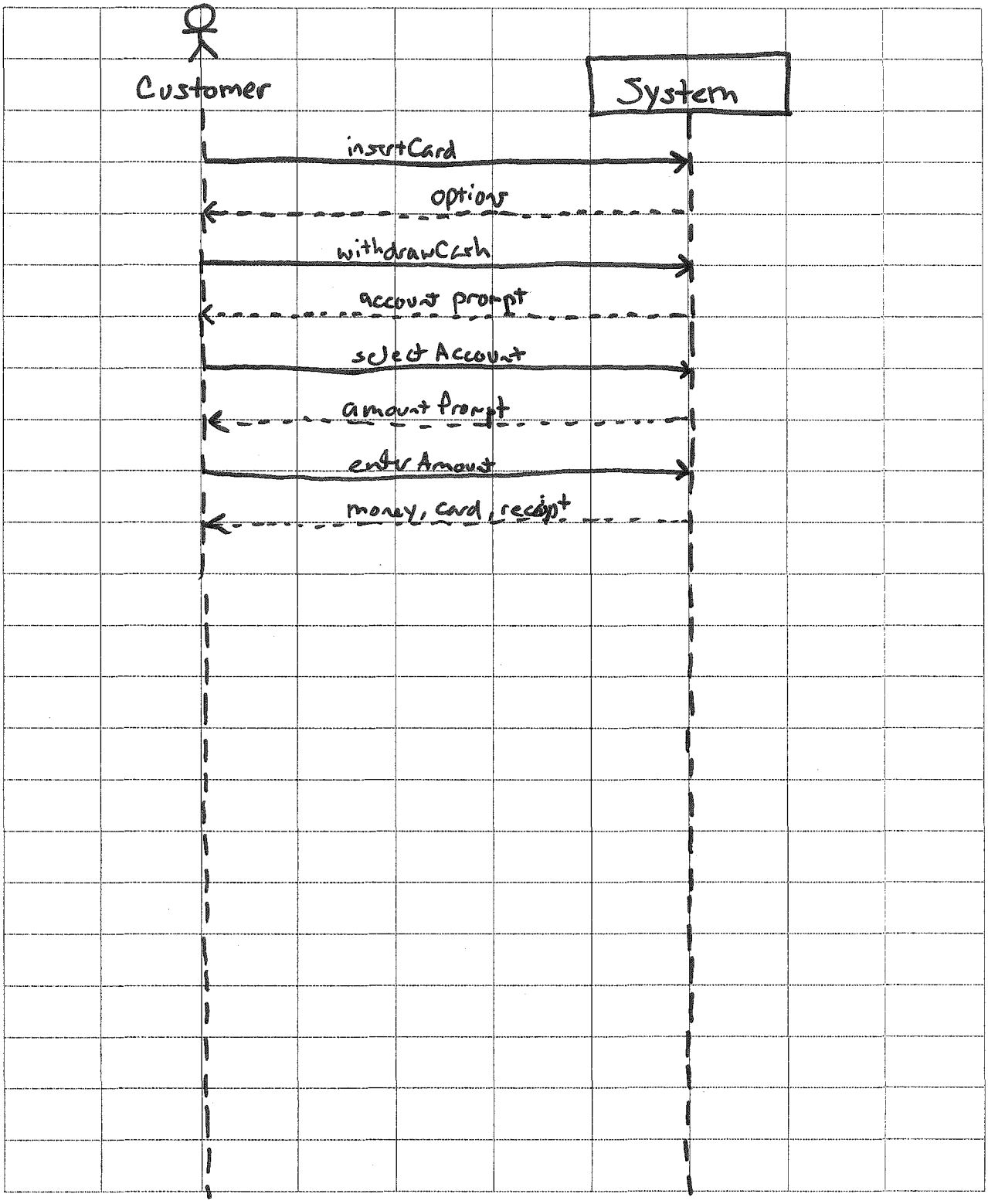
   d. Reverse lookup

   e. Mask and shift

9. [12pts] Given the following scenario, draw a system sequence diagram on the next page.

   1. Bank Customer inserts their Bank Card.
   2. The ATM displays the different alternatives that are available on this unit.
   3. The Bank Customer selects "Withdraw Cash".
   4. The ATM prompts for an account.
   5. The Bank Customer selects an account.
   6. The ATM prompts for an amount.
   7. The Bank Customer enters an amount.
   8. The ATM validates the withdrawal.
   9. Then money is dispensed, the Bank Card is returned, and the receipt is printed.

(Write your answer here.)



10. [2pts] Which question does non-functional requirements answer?

    a. What does the system do?

    b. When does the system do it?

    c. Where does the system do it?

    d. Why does the system do it?

    (e.) How well does the system do it?

11. [2pts] Which of the following should a user story not do?

    a. Describe one thing the software needs to do for the customer

    b. Be short

    (c.) Discuss specific technologies

    d. Be written using language the customer understands

    e. None of the above

12. [2pts] In the agile development process taught in class, the development team estimates each user story and decides the priority for each story.

    a. True

    (b.) False

13. [2pts] What type of object is used to store "conversational state" with a user?

    a. Servlet

    b. Request

    c. Request Dispatcher

    d. Response

    (e.) Session

14. [2pts] Which of the following is not a desirable quality of a unit test?

    a. No I/O

    b. Fast

    (c.) Non-deterministic

    d. Tests one property

    e. None of the above

15. [10pts] Given the class `Foo`, complete the JUnit test below. The `compute` method of a `Foo` object with `z = 10` should return 69 if passed the argument 7 (as `x`). Your test should verify this behavior. Don't forget the JUnit method `fail`, which takes a `String` comment as its argument.

```
public class Foo {
    private int z;
    public Foo(int zarg) { z = zarg; }
    public int compute(int x) { … }
    …
}
```

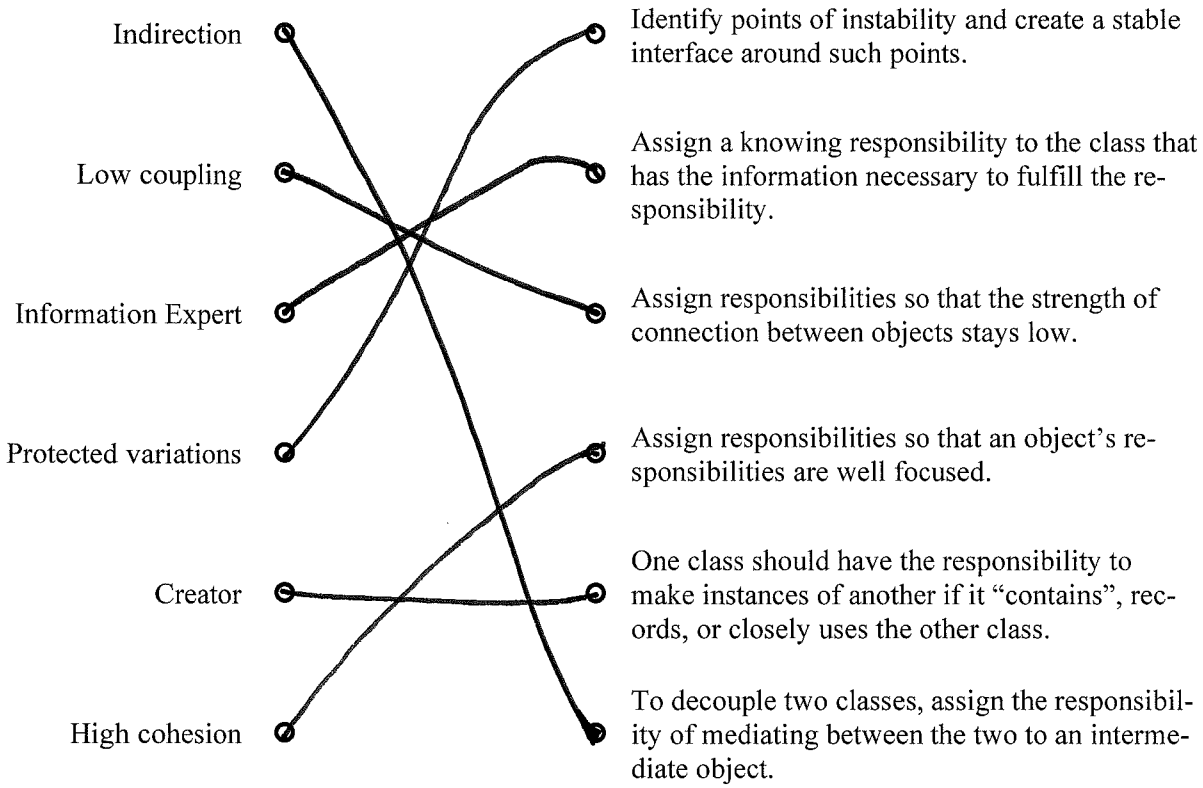
```
public class FooTest {
    @Test
    public int testCompute(int x) {
```

```
    int expected = 69;
    Foo myFoo = new Foo(10);
    int actual = myFoo.compute(7);
    if ( actual != expected ) {
        Fail ("Test Failed");
    }
```

```
    }
}
```

Question Group #2

1. [6pts] For each pattern below, draw a line from the pattern to its definition.



2. [2pts] T or F? Coupling and cohesion are closely linked in that as one increases, so does the other.

   a. True

   (b.) False

3. [2pts] Given classes *A* and *B*, which of the following is <u>not</u> a common type of coupling in Java?

   a. *A* is a direct or an indirect subclass of *B*

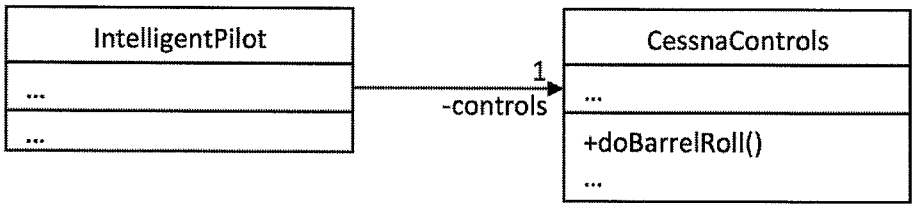
   (b.) *A* belongs to the same package as *B*

   c. A method parameter or local variable in *A* references *B*

   d. *A* has an instance variable that refers to *B*

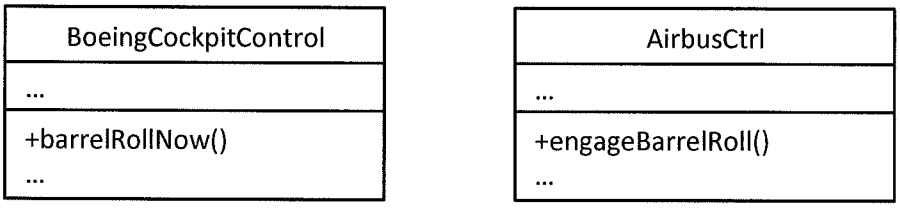
   e. *A* invokes methods of *B*

4. [10pts] 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 **Indirection Pattern**, the **Protected Variations Pattern**, and the **Polymorphism Pattern**. (Hint: Recall that we discussed a similar design in class for switching between tax calculators.)

Draw a class diagram for your design on the next page.

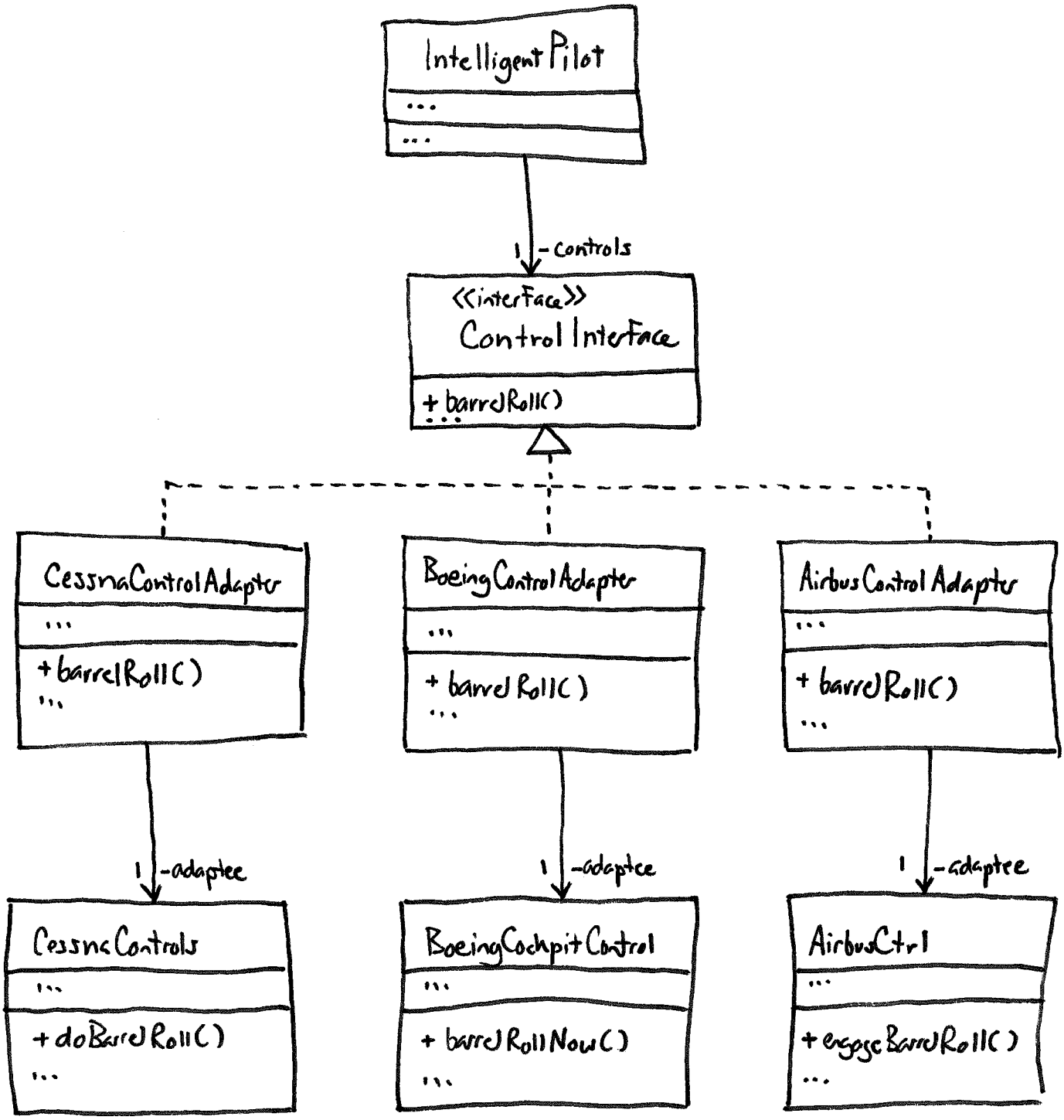
5. [2pts] What effect did your new design have on the coupling between class IntelligentPilot and class CessnaControls.

(a.) Reduced their coupling

b. Increased their coupling

c. Had no effect on their coupling

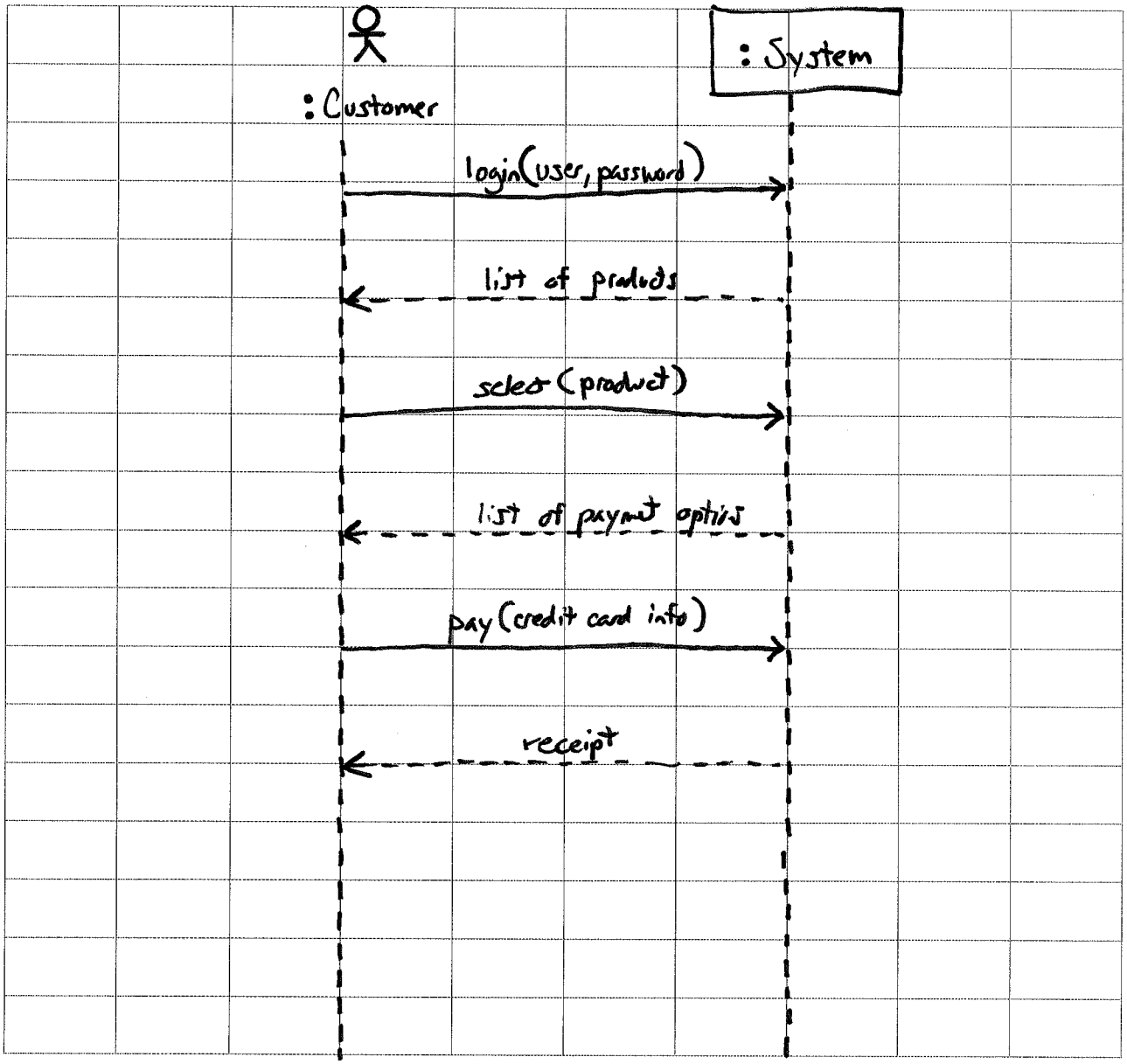
Write your answer to question 4 here.



9. [10pts] Given the following scenario, draw a system sequence diagram.

1. The customer enters his/her login and password.
2. The system validates the login information.
3. The system displays a list of products.
4. The customer selects a product.
5. The system displays a list of payment options.
6. The customer enters his/her credit card info.
7. The system charges the purchase to the credit card.
8. The system displays a purchase receipt.

(Write your answer here.)



10. [2pts] SRP is short for:

    a. Software Requirements Process

    b. Sequential Response Protocol

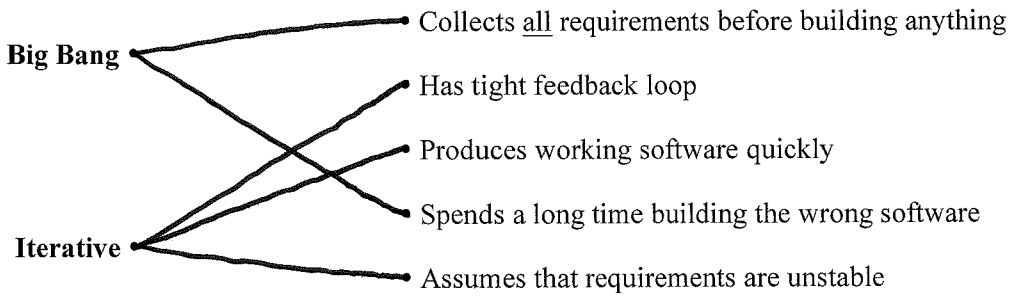
    c. Server Receive Packet

    d. Single Responsibility Principle

    e. None of the above

Problem Group #3

1. [5pts] Match each item on the right to the development approach that the item best characterizes.



2. [3pts] What three things does great software deliver?

a. What is needed

b. On time

c. On budget

3. [4pts] Of the two user stories in Figure 1, which was better written? Explain your answer.

US Review Flight is better because it is customer oriented. US Animate Buttons talks about implementation techologies (jQuery) that the may know nothing about.

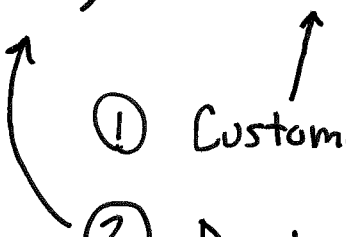
4. [2pts] All else being equal, which of the USs in Figure 1 most likely has the more accurate estimate?

US Animate Buttons

(Because estimates of less than 15 days are generally more accurate than ones over 15 days.)

5. [4pts] What two things are wrong with the following series of steps?

(1) First, the developers solicit user stories from the customer.
(2) Next, the developers assign a priority level to each user story.
(3) Next, the developers estimate the effort required to implement each user story.



① Customers assign priorities

② Developers must estimate effort before customers assign priorities (otherwise how can the customer assess the cost/benefit?)

6. [2pts] Which of the following techniques is used for estimating effort?

a. Role playing
b. Blueskying
c. Planning poker
d. Observation
e. None of the above

8. [6pts] After your team chooses the USs to implement in an iteration, but before the team begins implementing, what three things must the team do?

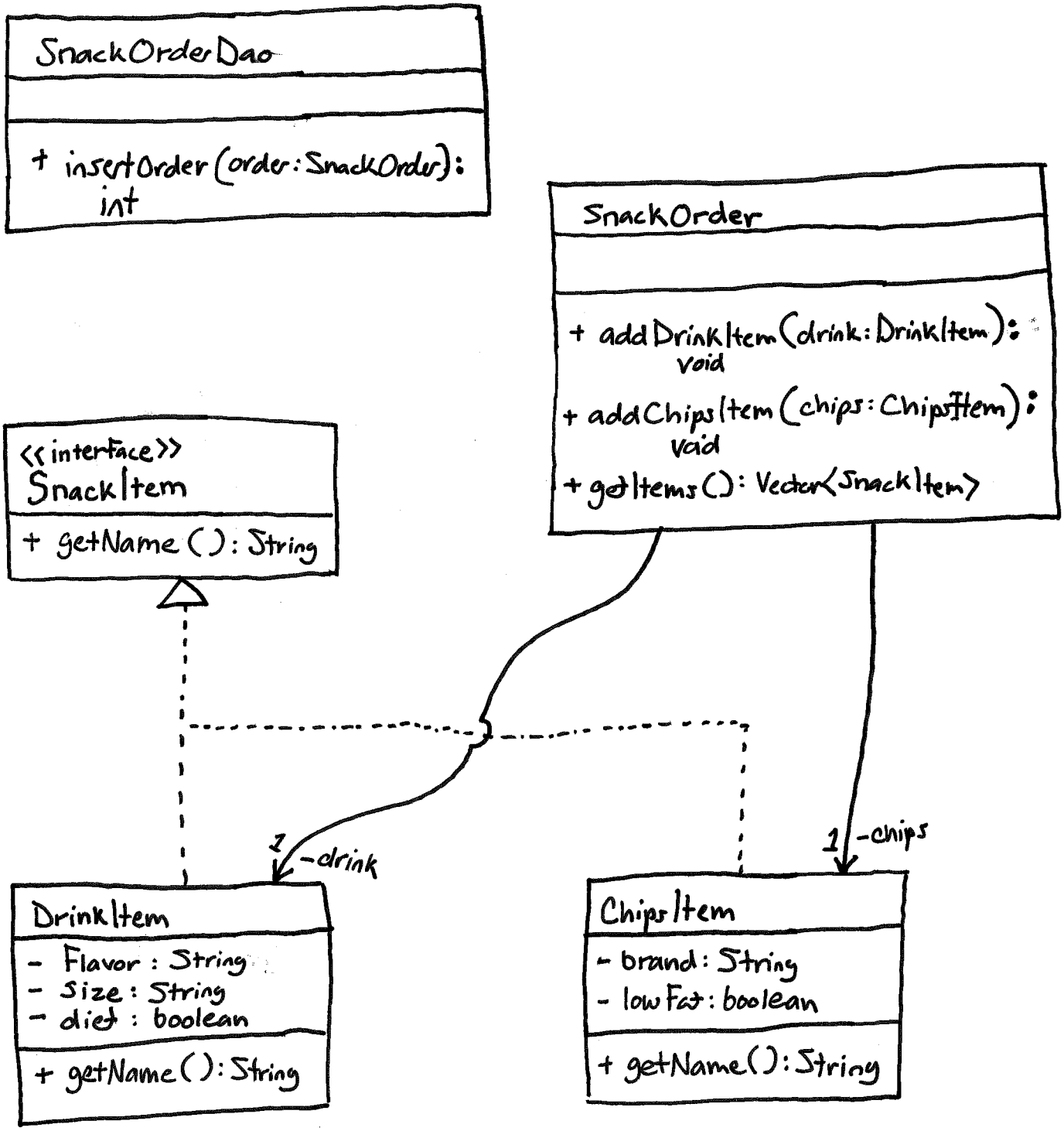
① Break the USs into tasks

② Estimate the time to complete each task

③ Assign each task to a developer

Remember the snack-ordering system from Exam 1? It's back! The code has changed, though. Updated screenshots and code can be found in Figure 3–Figure 15. Use these figures to answer the remaining questions.

11. [8pts] Given the Java code in the figures, draw a class diagram that models the code. Include only the following classes in your diagram: **SnackOrder**, **SnackOrderDao**, **SnackItem**, **DrinkItem**, and **ChipsItem**. You need not include getter/setter methods that only assign/return instance variables.



12. [10pts] Refactor the JSPs (Figure 6–Figure 9) so that they better obey the DRY principle (i.e., write code). Of the three JSPs, you may redefine only one (e.g., **orderError.jsp**) as long as it's clear from that one what changes you would make to the others. Hint: you may also need to create new files. Recall that the JSP directive tag **<%@ include file="foo.jsp" %>** inserts the contents of **foo.jsp**.

std-advert.jsp

```
<!-- STANDARD ...
<div ...
<a href=...
<a href=...
<a href=...
<a href=...
<a href=...
<a href=...
</div>
<!-- STANDARD ...
```

orderError.jsp

```
<% @ page ...
<! DOCTYPE ...
<html>
<head>
<meta ...
<title ...
</head>
<body>
<%@ include file="std-advert.jsp" %>
<h1 ...
<%@ include file="std-advert.jsp" %>
</body>
</html>
```

13. [10pts] Refactor the **SnackOrder** class (i.e., write code) so that it better obeys the SRP principle. Hint #1: there should be no need to modify your code if new types of snacks are introduced (e.g., candy). Hint #2: the problem is that the class knows too much.

```
public class SnackOrder {
    private Vector<SnackItem> items = new Vector<SnackItem>();

    public void addSnackItem(SnackItem item) {
        items.addElement(item);
    }
    public Vector<SnackItem> getItems() {
        return items;
    }
}
```

14. [8pts] Define a white-box test suite that covers all possible paths through the **OrderSnackServlet**'s **doPost** method (Figure 10). You may use plain English, but it must be unambiguous what the inputs of each test case would be like.

	diet?	lowFat?	insertOrder returns -1
Test #1	Y	Y	Y
#2	N	Y	Y
#3	Y	N	Y
#4	N	N	Y
#5	Y	Y	N
#6	N	Y	N
#7	Y	N	N
#8	N	N	N

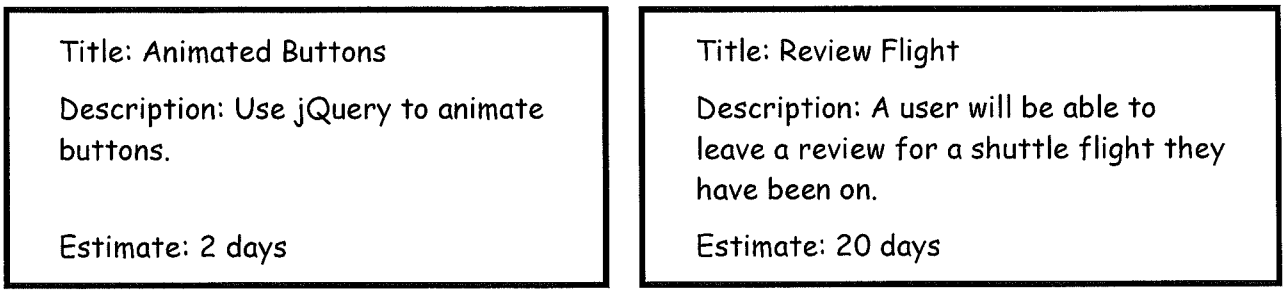


Figure 1. Two example user stories.

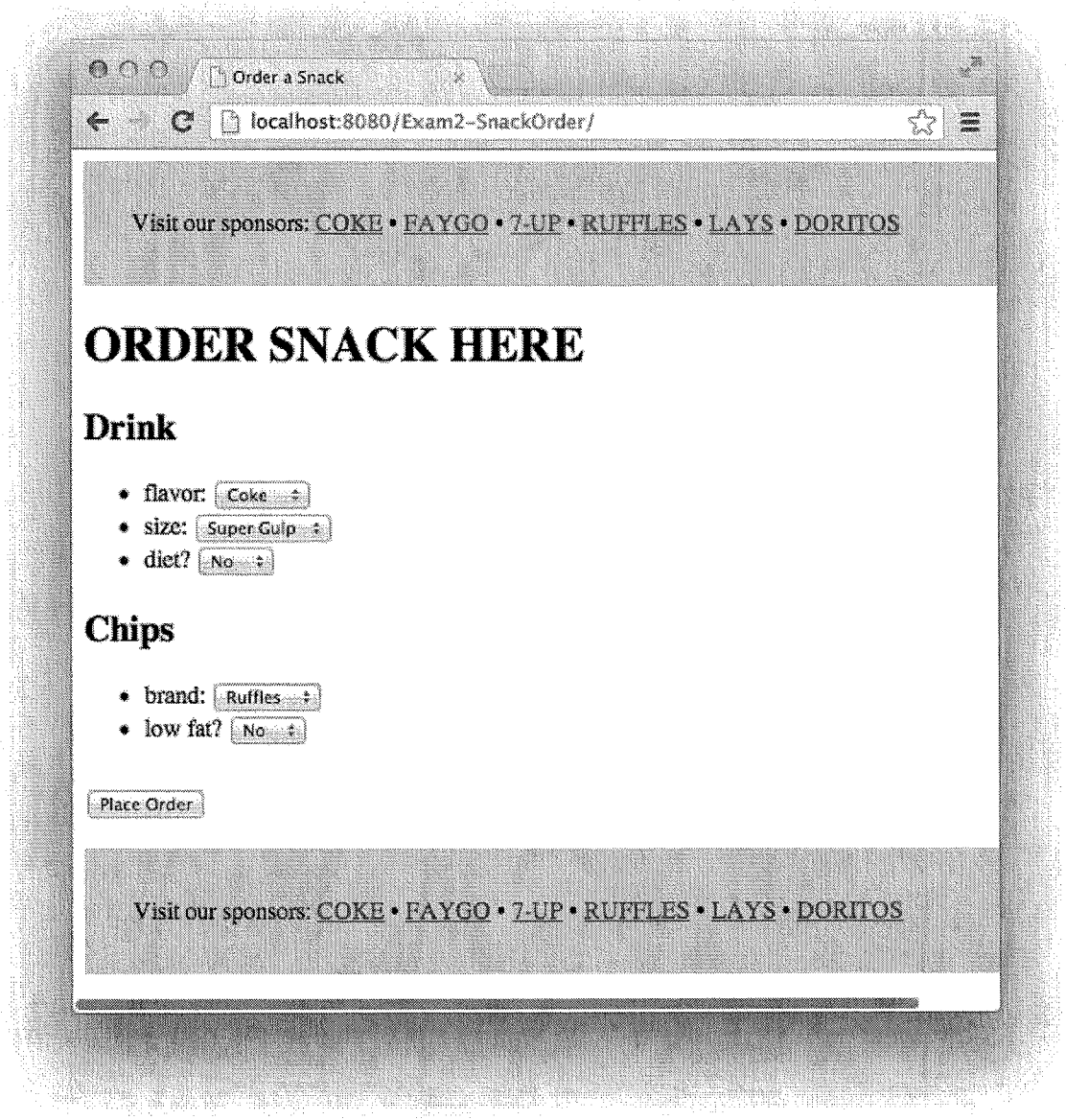


Figure 3. The snack-order system's index.jsp page.

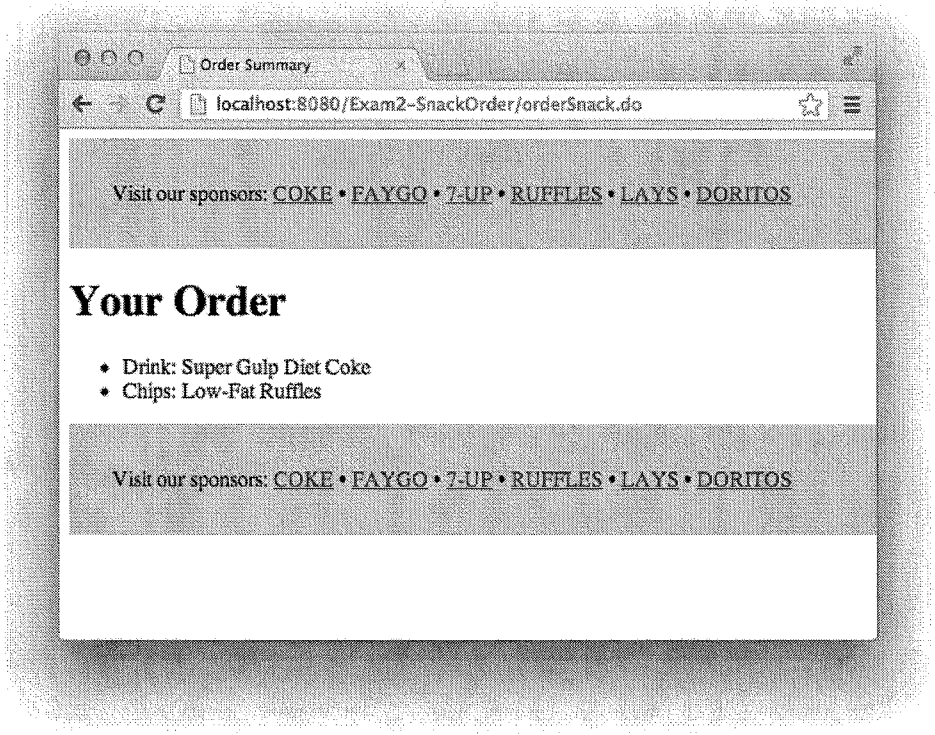


Figure 4. The snack-order system's orderSummary.jsp page.

Figure 5. The snack-order system's orderError.jsp page.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Order a Snack</title>
</head>
<body>

<!-- STANDARD ADVERTISEMENT BEGIN -->
<div style="width: 100%; background-color: #CCC; margin: 0; padding: 2em;">
Visit our sponsors:
<a href="http://www.coke.com">COKE</a> &bull;
<a href="http://www.faygo.com">FAYGO</a> &bull;
<a href="http://www.7up.com">7-UP</a> &bull;
<a href="http://www.fritolay.com/our-snacks/ruffles-original.html">RUFFLES</a> &bull;
<a href="http://www.lays.com">LAYS</a> &bull;
<a href="http://www.doritos.com">DORITOS</a>
</div>
<!-- STANDARD ADVERTISEMENT END -->

<form method="post" action="orderSnack.do">
<h1>ORDER SNACK HERE</h1>
<h2>Drink</h2>
<ul>
<li>flavor:
<select name="drinkFlavor">
<option value="Coke" selected>Coke</option>
<option value="Faygo">Faygo</option>
<option value="7-Up">7-Up</option>
</select></li>
<li>size:
<select name="drinkSize">
<option value="Small">Small</option>
<option value="Medium">Medium</option>
<option value="Large">Large</option>
<option value="Super Gulp" selected>Super Gulp</option>
</select></li>
<li>diet?
<select name="drinkDiet">
<option value="Yes">Yes</option>
<option value="No" selected>No</option>
</select></li>
</ul>
```

Figure 6. index.jsp (part 1 of 2)

```
<h2>Chips</h2>
<ul>
<li>brand:
<select name="chipsBrand">
<option value="Ruffles" selected>Ruffles</option>
<option value="Lays">Lays</option>
<option value="Doritos">Doritos</option>
</select></li>
<li>low fat?
<select name="chipsLowFat">
<option value="Yes">Yes</option>
<option value="No" selected>No</option>
</select></li>
</ul>
<h2><input type="submit" value="Place Order"></h2>
</form>

<!-- STANDARD ADVERTISEMENT BEGIN -->
<div style="width: 100%; background-color: #CCC; margin: 0; padding: 2em;">
Visit our sponsors:
<a href="http://www.coke.com">COKE</a> &bull;
<a href="http://www.faygo.com">FAYGO</a> &bull;
<a href="http://www.7up.com">7-UP</a> &bull;
<a href="http://www.fritolay.com/our-snacks/ruffles-original.html">RUFFLES</a> &bull;
<a href="http://www.lays.com">LAYS</a> &bull;
<a href="http://www.doritos.com">DORITOS</a>
</div>
<!-- STANDARD ADVERTISEMENT END -->

</body>
</html>
```

Figure 7. index.jsp (part 2 of 2)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" import="com.snack.model.*" import="java.util.Vector" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Order Summary</title>
</head>
<body>

<!-- STANDARD ADVERTISEMENT BEGIN -->
<div style="width: 100%; background-color: #CCC; margin: 0; padding: 2em;">
Visit our sponsors:
<a href="http://www.coke.com">COKE</a> &bull;
<a href="http://www.faygo.com">FAYGO</a> &bull;
<a href="http://www.7up.com">7-UP</a> &bull;
<a href="http://www.fritolay.com/our-snacks/ruffles-original.html">RUFFLES</a> &bull;
<a href="http://www.lays.com">LAYS</a> &bull;
<a href="http://www.doritos.com">DORITOS</a>
</div>
<!-- STANDARD ADVERTISEMENT END -->

<h1>Your Order</h1>

<ul>
<%
SnackOrder order = (SnackOrder) request.getAttribute("order");
Vector<SnackItem> items = order.getItems();
%>
<li>Drink: <%= items.get(0).getName() %>
<li>Chips: <%= items.get(1).getName() %>
</ul>

<!-- STANDARD ADVERTISEMENT BEGIN -->
<div style="width: 100%; background-color: #CCC; margin: 0; padding: 2em;">
Visit our sponsors:
<a href="http://www.coke.com">COKE</a> &bull;
<a href="http://www.faygo.com">FAYGO</a> &bull;
<a href="http://www.7up.com">7-UP</a> &bull;
<a href="http://www.fritolay.com/our-snacks/ruffles-original.html">RUFFLES</a> &bull;
<a href="http://www.lays.com">LAYS</a> &bull;
<a href="http://www.doritos.com">DORITOS</a>
</div>
<!-- STANDARD ADVERTISEMENT END -->

</body>
</html>
```

Figure 8. orderSummary.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Order Error</title>
</head>
<body>

<!-- STANDARD ADVERTISEMENT BEGIN -->
<div style="width: 100%; background-color: #CCC; margin: 0; padding: 2em;">
Visit our sponsors:
<a href="http://www.coke.com">COKE</a> &bull;
<a href="http://www.faygo.com">FAYGO</a> &bull;
<a href="http://www.7up.com">7-UP</a> &bull;
<a href="http://www.fritolay.com/our-snacks/ruffles-original.html">RUFFLES</a> &bull;
<a href="http://www.lays.com">LAYS</a> &bull;
<a href="http://www.doritos.com">DORITOS</a>
</div>
<!-- STANDARD ADVERTISEMENT END -->

<h1>Sorry! Your order could not be processed due to an error.</h1>

<!-- STANDARD ADVERTISEMENT BEGIN -->
<div style="width: 100%; background-color: #CCC; margin: 0; padding: 2em;">
Visit our sponsors:
<a href="http://www.coke.com">COKE</a> &bull;
<a href="http://www.faygo.com">FAYGO</a> &bull;
<a href="http://www.7up.com">7-UP</a> &bull;
<a href="http://www.fritolay.com/our-snacks/ruffles-original.html">RUFFLES</a> &bull;
<a href="http://www.lays.com">LAYS</a> &bull;
<a href="http://www.doritos.com">DORITOS</a>
</div>
<!-- STANDARD ADVERTISEMENT END -->

</body>
</html>
```

Figure 9. orderError.jsp

```
@WebServlet("/orderSnack.do")
public class OrderSnackServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		DrinkItem drink = new DrinkItem();
		drink.setFlavor(request.getParameter("drinkFlavor"));
		drink.setSize(request.getParameter("drinkSize"));
		String drinkDietStr = request.getParameter("drinkDiet");
		if (drinkDietStr.equals("Yes")) {
			drink.setDiet(true);
		} else {
			drink.setDiet(false);
		}

		ChipsItem chips = new ChipsItem();
		chips.setBrand(request.getParameter("chipsBrand"));
		String chipsLowFatStr = request.getParameter("chipsLowFat");
		if (chipsLowFatStr.equals("Yes")) {
			chips.setLowFat(true);
		} else {
			chips.setLowFat(false);
		}

		SnackOrder order = new SnackOrder();
		order.addDrinkItem(drink);
		order.addChipsItem(chips);
		request.setAttribute("order", order);

		SnackOrderDao dao = new SnackOrderDao();

		if (dao.insertOrder(order) == -1) {
			request.getRequestDispatcher("WEB-INF/orderError.jsp").forward(request,
response);
		} else {
			request.getRequestDispatcher("WEB-INF/orderSummary.jsp").forward(request,
response);
		}
	}

}
```

Figure 10. OrderSnackServlet.java

```
public class SnackOrder {

	private DrinkItem drink = null;
	private ChipsItem chips = null;

	public void addDrinkItem(DrinkItem drink) { this.drink = drink; }
	public void addChipsItem(ChipsItem chips) { this.chips = chips; }

	public Vector<SnackItem> getItems() {
		Vector<SnackItem> result = new Vector<SnackItem>();
		result.addElement(drink);
		result.addElement(chips);
		return result;
	}

}
```

Figure 11. SnackOrder.java

```
public class SnackOrderDao {

	/**
	 * Returns newly created order number or -1 on error.
	 */
	public int insertOrder(SnackOrder order) {
		...
	}

	...
}
```

Figure 12. SnackOrderDao.java

```
public interface SnackItem {
	public String getName();
}
```

Figure 13. SnackItem.java

```
public class DrinkItem implements SnackItem {

	private String flavor = "";
	private String size = "";
	private boolean diet = false;

	public void setFlavor(String flavor) { this.flavor = flavor; }
	public void setSize(String size) { this.size = size; }
	public void setDiet(boolean diet) { this.diet = diet; }

	public String getFlavor() { return flavor; }
	public String getSize() { return size; }
	public boolean isDiet() { return diet; }

	public String getName() {
		return size + " " + (diet?"Diet ":"") + flavor;
	}

}
```

Figure 14. DrinkItem.java

```
public class ChipsItem implements SnackItem {

	private String brand = "";
	private boolean lowFat = false;

	public void setBrand(String brand) { this.brand = brand; }
	public void setLowFat(boolean lowFat) { this.lowFat = lowFat; }

	public String getBrand() { return brand; }
	public boolean isLowFat() { return lowFat; }

	public String getName() {
		return (lowFat?"Low-Fat ":"") + brand;
	}

}
```

Figure 15. ChipsItem.java

Question Group #4

For each of the following multiple-choice questions, circle the (one) response that best answers the question.

1. [2pts] What is the typical relationship between coupling and cohesion?

   a. There is no relationship between coupling and cohesion.

   b. As cohesion increases, coupling increases.

   (c.) As cohesion increases, coupling decreases.

2. [2pts] All else being equal, which is more desirable?

   a. Higher cohesion and higher coupling

   (b.) Higher cohesion and lower coupling

   c. Lower cohesion and lower coupling

   d. Lower cohesion and higher coupling

   e. None of the above is more desirable than the others.

3. [2pts] All "real" software contains bugs. True or false?

   (a.) True

   b. False

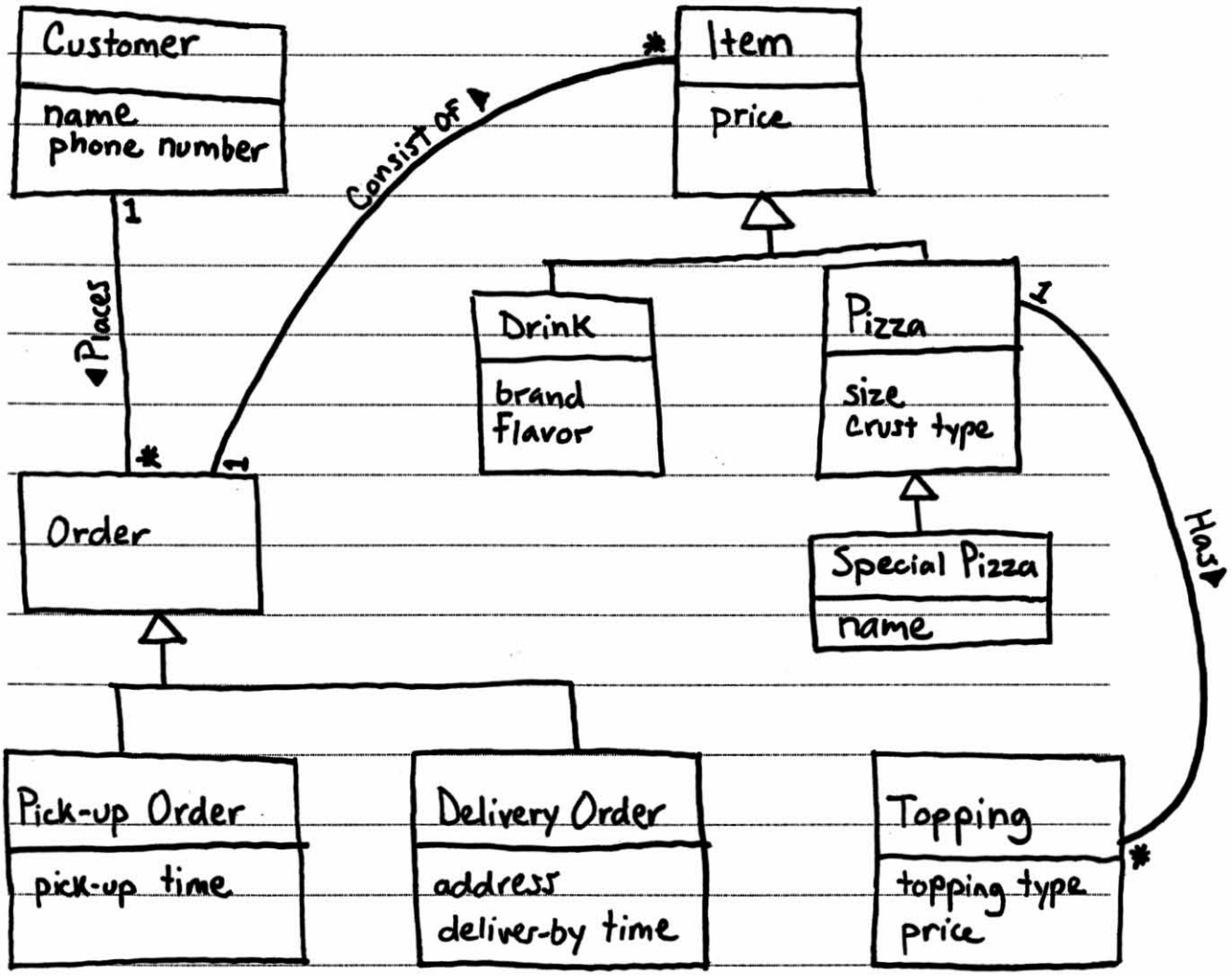
4. [2pts] Generally, in practice, developers exhaustively test software. True or false?

   a. True

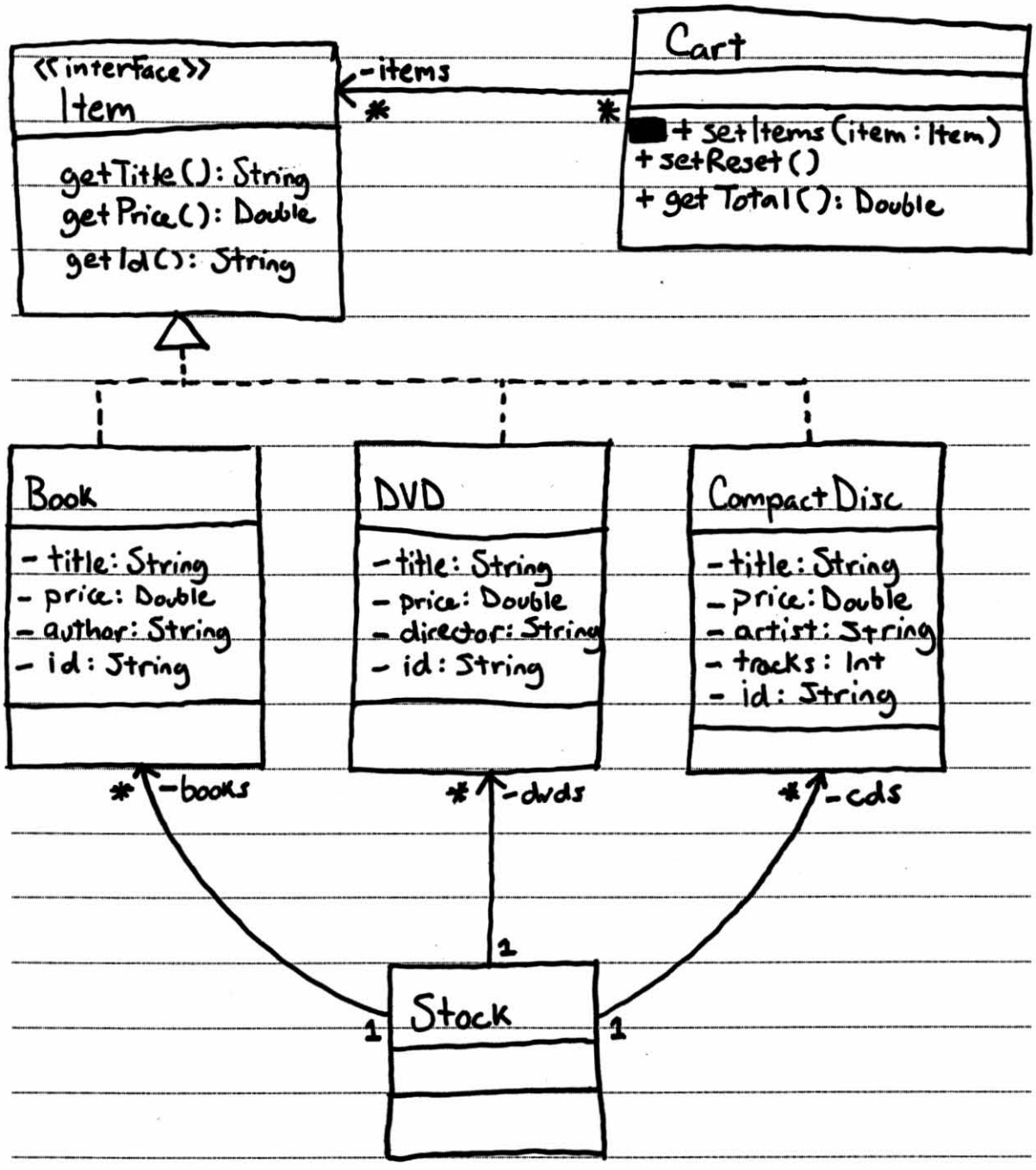
   (b.) False

6. [10pts] Imagine that you are tasked with developing a system for a pizza shop. Given the following description, create a Domain Model (in the form of a UML class diagram). Include all conceptual classes, attributes, associations, and generalization relationships mentioned in the descriptions. Label all associations and include all multiplicities.

A customer places orders. A customer has a name and phone number. There are two types of orders: pick-up and delivery. A pick-up order has a pick-up time. A delivery order has an address and deliver-by time. All orders consist of a set of items. There are two types of items: pizzas and drinks. All items have a price. A pizza has a size and a crust type. A pizza also has a number of toppings. A topping has a topping type and a price. Some pizzas are special pizzas that have a name (e.g., "Hawaiian" or "Meat Lovers"). A drink has a brand and a flavor.

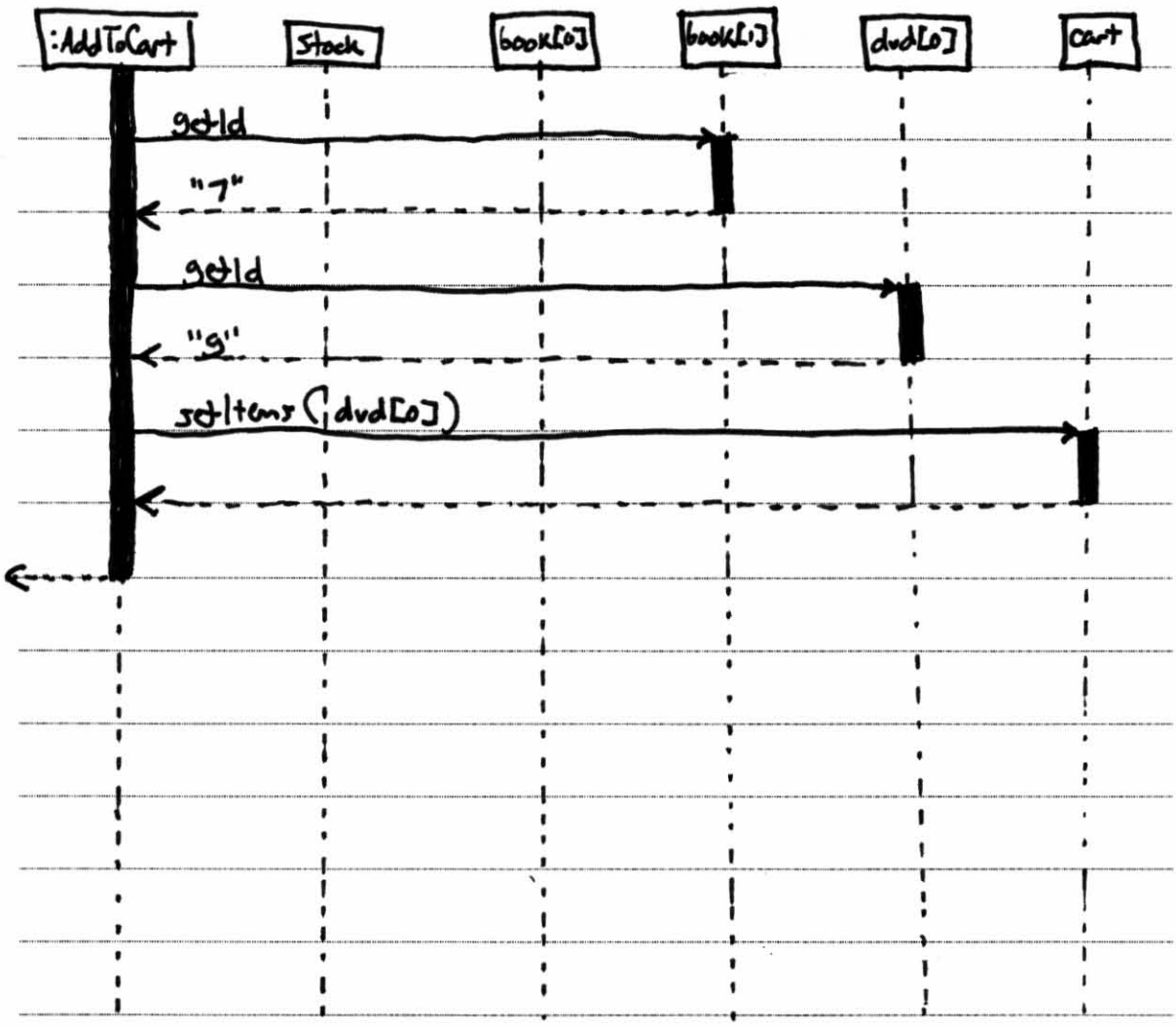


7. [13pts] Reverse engineer a design class diagram for the Media"R"Us.com system (page 16). Include only the following classes/interfaces in your diagram: Item, Book, DVD, CompactDisc, Cart, and Stock. Model completely all attributes, operations, associations, visibilities, multiplicities, generalizations, etc. You may omit constructors and simple getter/setter operations that do nothing more than return/assign an instance variable.



8. [12pts] Consider the object diagram (snapshot) of a running Media"R"Us system (page 19). Complete the following sequence diagram that models the interactions that would result from an execution of AddToCart's add() method when method parameter ids = ("7", "g"). Model only interactions among the objects given below—be careful not to miss any! Explicitly model returns and return values, and execution specifications (aka activation bars).

(There is more room on the next page.)


:AddToCart
Stock
book[0]
book[1]
dvd[0]
cart
getId
"7"
getId
"9"
setItems(dvd[0])

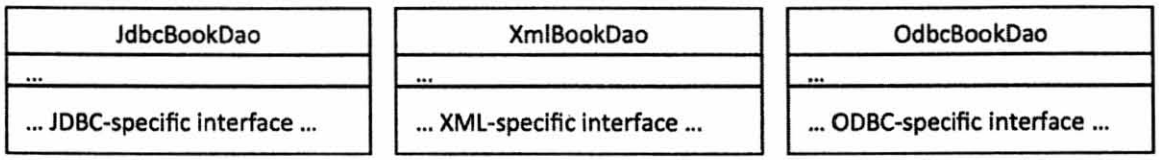
9. [5pts] Consider the total cost of customer's purchase of books, DVDs, and/or CDs. Which Media"R"Us class is an information expert with respect to the total cost? Why was that class chosen to be the information expert?

Cart is the information expert for total cost of purchase.
Cart was chosen to be information expert because it is the class with all the information necessary to fulfill that responsibility. That is, a Cart knows all the Items in a purchase, and each item knows its unit cost.

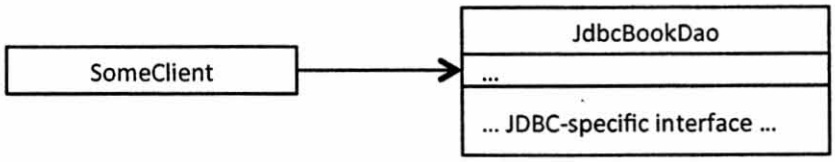
10. [2pts] To which component of an MVC architecture should the Item, Cart, and Stock classes belong? Give the full name for the component (not just the letter).

Model

There are many different ways that you might want to store Media"R"Us items in persistent storage. For example, you might store items to an XML file, or store them in a DB (via the JDBC interface), or transmit them via TCP to some other type of server. Imagine that the Book class has a corresponding data access object (DAO) class for each storage technology, and each DAO class has a slightly different interface. For example:



Now, imagine some client code implements a media sales system and uses the JdbcBookDao class to create/read/update/delete stored Book objects:

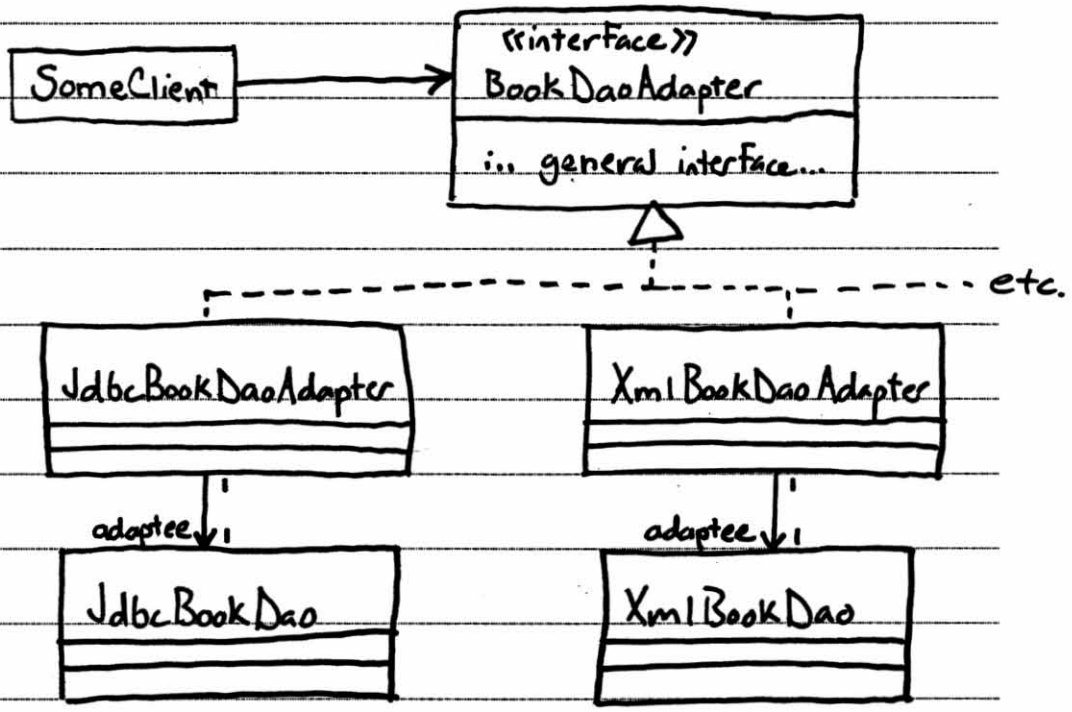


11. [5pts] What drawback does the SomeClient/JdbcBookDao design have if the choice of storage technology tends to be an unstable design decision (i.e., prone to change)?

The drawback is that you need to modify the SomeClient code wherever you change storage technology.

12. [10pts] How might you improve the design to overcome this drawback? Explain your design with both words and a design class diagram. Name two of the design patterns discussed in class, and explain how each contributes to your design.

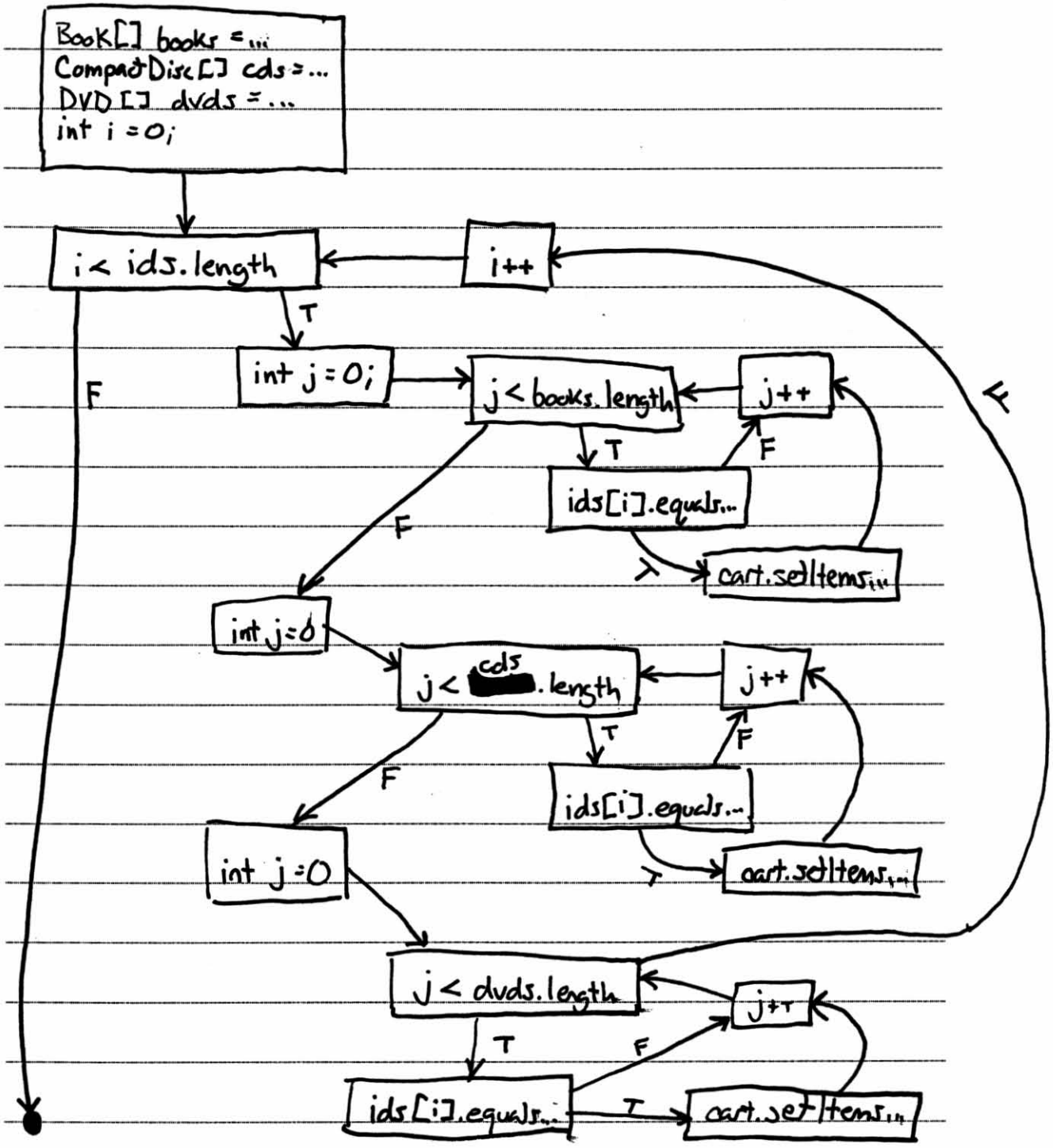
To improve the design, I would decouple the SomeClient and JdbcBookDao classes by inserting an adapter class, like this:



This design applies 3 main patterns:

- Indirection: It inserts a level of indirection between client and BookDao.
- Protected Variation: Creates stable adapter interface around unstable BookDao interfaces.
- Polymorphism: BookDaoAdapter interface operations are polymorphic, so different storage technologies can be swapped in and out.

13. [10pts] Draw a control flow graph (CFG) for AddToCart's add() method. (See the example of a for-loop CFG on page 20.) You may shorten long lines of code using ellipses (i.e., "..."s); for example: `Book[ ] books = ...`



In the next two questions, you will use your CFG to define test suites. Because the id is the only part of an Item we care about in these problems, you may use the following shorthand notation:

- ids = ("id1", "id2", "id3")
  - Specifies an array of 3 id strings.
- cart = ("id4", "id5")
  - Specifies a cart with two items with id="id4" and id="id5", respectively.
- stock = (Books → ("id6", "id7"), CDs → ("id8"), DVDs → ("id9", "id10"))
  - Specifies a stock with 2 books, 1 CD, and 2 DVDs.

To simplify matters, you may omit the expected output from your text cases.

14. [5pts] Specify a test suite that achieves statement coverage of AddToCart's add() method.

Test case #1:

ids = ("a", "b", "c")

cart = ()

stock = (Books → ("a"), CDs → ("b"), DVDs → ("c"))

15. [5pts] Specify a test suite that achieves condition coverage of the AddToCart's add() method.

The suite From question 15 will work here.

16. [3pts] How may test cases would a suite have that does path coverage (with max of 1 iteration per loop) of add()?

a. 1

b. 2

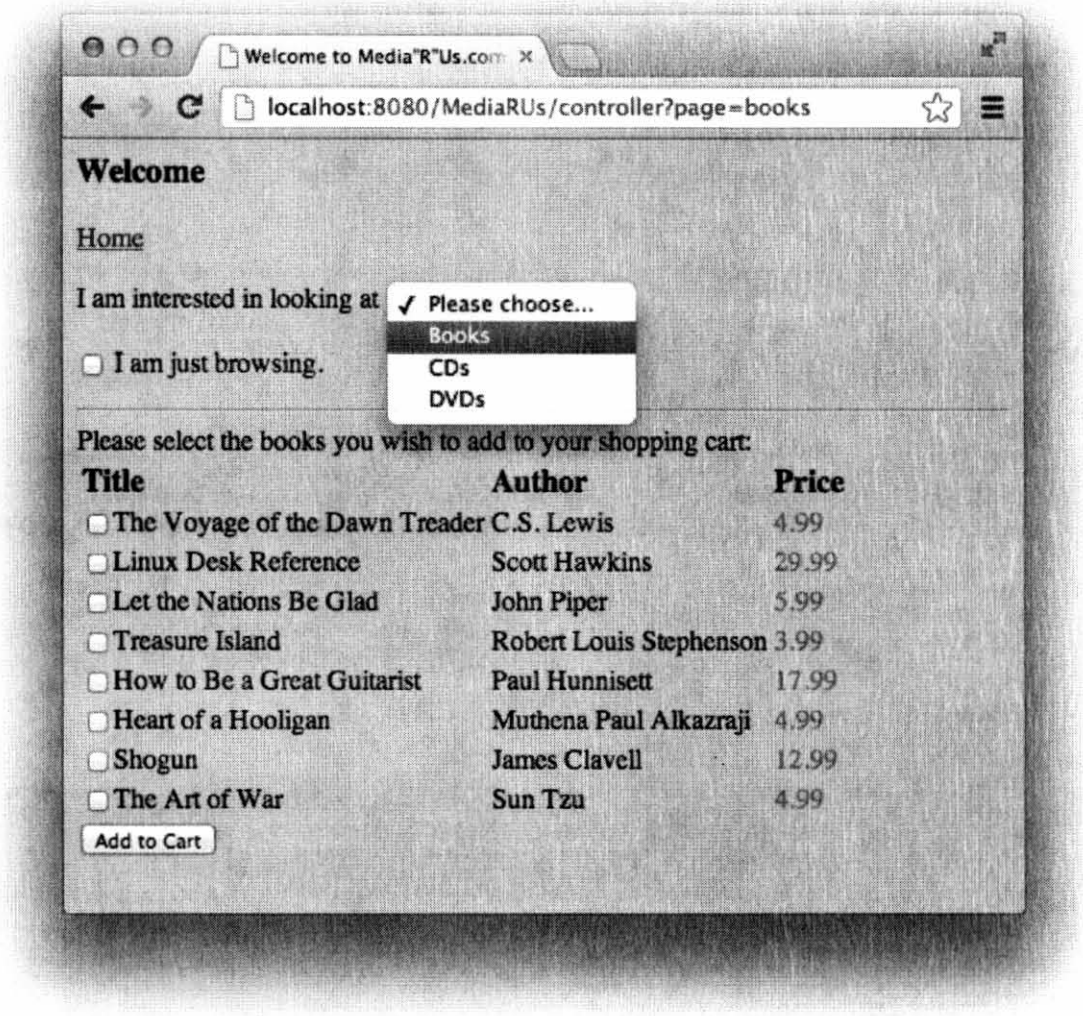
c. 4

d. 7

(e.) More than 7

Media"R"Us.com Source Code

The following source code belongs to the code base for a media sales system. You might use this code to build an online media store like this:



Media items include books, CDs, and DVDs. The code includes functionality for a cart to which shoppers can add their candidate purchases, and functionality for keeping track of the store's stock of media items.

```
public interface Item {
	String getTitle();
	double getPrice();
	String getId();
}
```

```
public class Book implements Item{
	//Instance variables
	private String title;
	private double price;
	private String author;
	private String id;

	//Methods
	public String getId(){ return id; }
	public String getTitle(){ return title; }
	public double getPrice(){ return price; }
	public String getAuthor(){ return author; }

	//constructors
	public Book(){}

	public Book(String t, double p,String a, String i){
		title=t;
		price=p;
		author=a;
		id=i;
	}
}
```

```
public class DVD implements Item{
	//Instance variables
	private String title;
	private double price;
	private String director;
	private String id;

	//Methods
	public String getId(){ return id; }
	public String getTitle(){ return title; }
	public double getPrice(){ return price; }
	public String getDirector(){ return director; }

	//constructors
	public DVD(){}

	public DVD(String t, double p,String d, String i){
		title=t;
		price=p;
		director=d;
		id = i;
	}
}
```

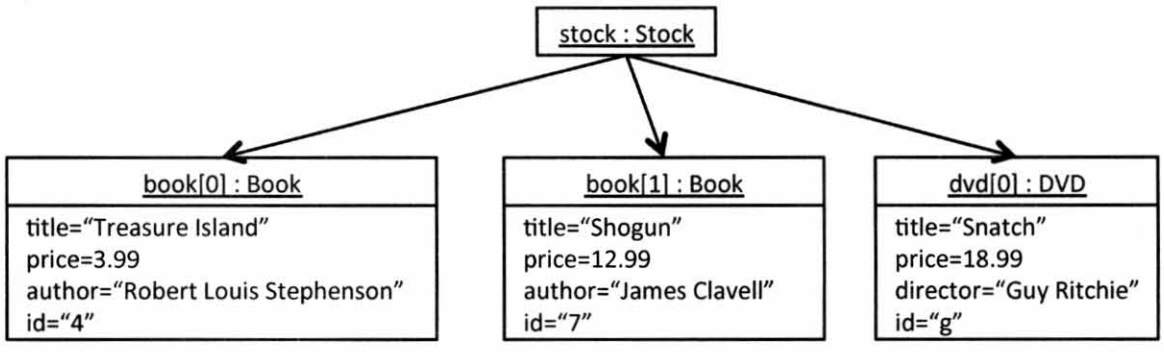
```
public class CompactDisc implements Item{
	//Instance variables
	private String title;
	private double price;
	private String artist;
	private int tracks;
	private String id;

	//Methods
	public String getId(){ return id; }
	public String getTitle(){ return title; }
	public double getPrice(){ return price; }
	public String getArtist(){ return artist; }
	public int getTracks(){ return tracks; }

	//constructors
	public CompactDisc(){}

	public CompactDisc(String t, double p, String a,
			int tr, String i){
		title=t;
		price=p;
		artist=a;
		tracks=tr;
		id = i;
	}
}
```

```
public class Cart {
	private Item[] items = new Item[0];

	public Item[] getItems() { return items; }

	public void setItems(Item item) {
		Vector v = new Vector(1);
		for (int i=0;i<items.length;i++){
			v.add(items[i]);
		}
		v.add(item);
		items = (Item[])v.toArray(items);
	}

	public void setReset() { items = new Item[0]; }

	public double getTotal() {
		double total = 0;
		for (int i=0;i<items.length;i++){
			total+=items[i].getPrice();
		}
		return total;
	}
}
```

```
public class Stock{
	private Book[] books;
	private CompactDisc[] cds;
	private DVD[] dvds;

	public Stock() { ... }

	public Book[] getBooks() { return books; }
	public CompactDisc[] getCds() { return cds; }
	public DVD[] getDvds() { return dvds; }
}
```

```
public class AddToCart {

	public void add(String[] ids, Cart cart, Stock stock) {
		Book[] books = stock.getBooks();
		CompactDisc[] cds = stock.getCds();
		DVD[] dvds = stock.getDvds();

		for (int i = 0; i < ids.length; i++) {
			for (int j = 0; j < books.length; j++) {
				if (ids[i].equals(books[j].getId())) {
					cart.setItems(books[j]);
				}
			}
			for (int j = 0; j < cds.length; j++) {
				if (ids[i].equals(cds[j].getId())) {
					cart.setItems(cds[j]);
				}
			}
			for (int j = 0; j < dvds.length; j++) {
				if (ids[i].equals(dvds[j].getId())) {
					cart.setItems(dvds[j]);
				}
			}
		}
	}
}
```

Media"R"Us Object Diagram



(Note: This is *not* a class diagram. It shows instances of classes.)

Control Flow Example

For Loop

```
before();
for (int i = 0; i < 10; ++i) {
  doSomething(i);
}
```

Control Flow Graph