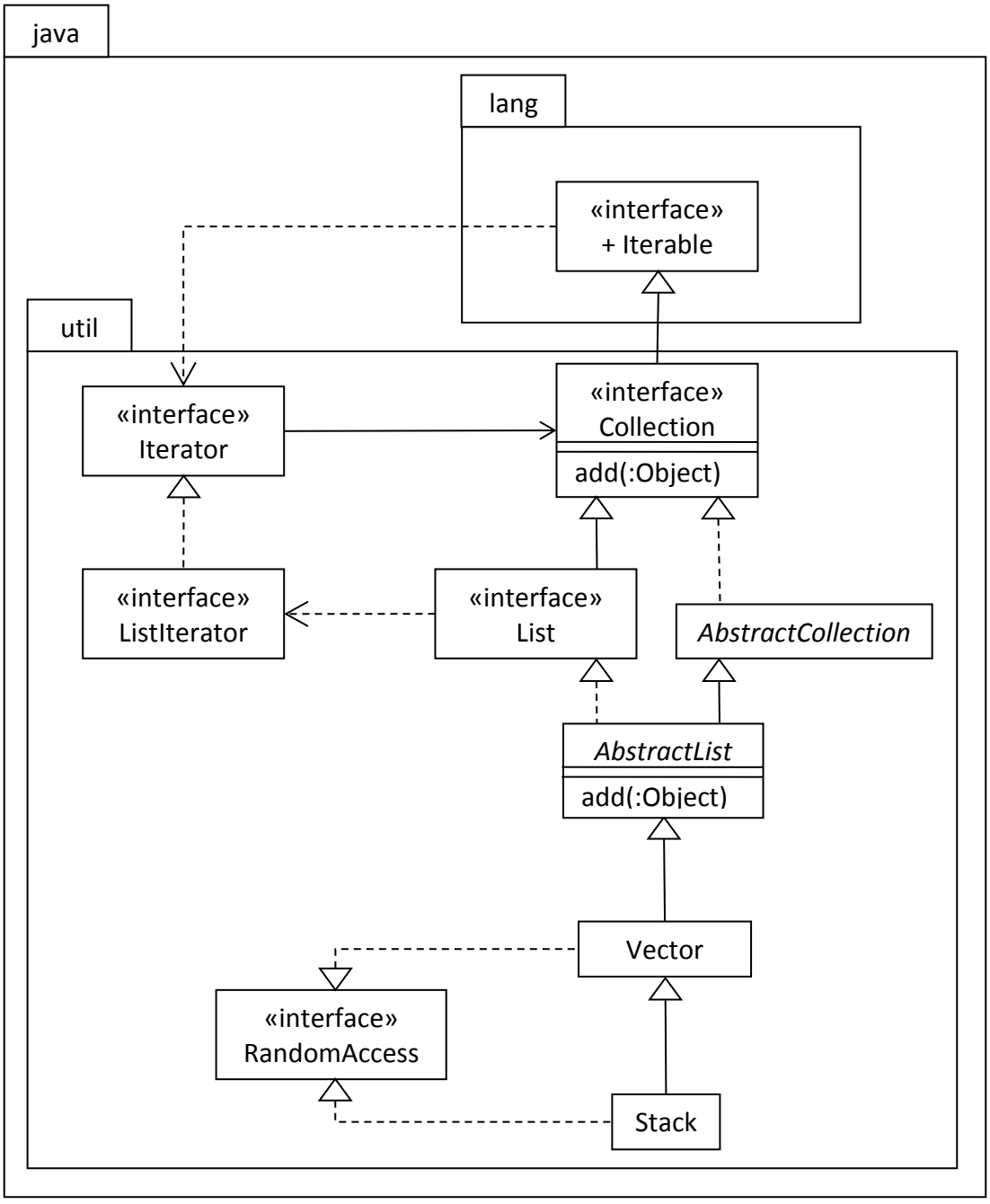
SENG 301 Software Analysis and Design

Examination Practice Questions

Below are a set of questions off old exams that are pertinent to this course. For simplicity, I have put them all here; make sure you pay attention to which topics are to be covered by the exam for which you are studying. If you are studying for a multiple choice exam, you need to be prepared to compare your answer against a bunch of possible answers from me… Crossed out questions aren't pertinent for your version of the course.

Use the diagram below to answer Questions 1 through 10.



For Questions 1 through 10, consider the diagram on page 2 then circle the letter of the **best answer**. Incorrect answers will treated as worse than blank answers.

1. Is Stack a subclass of Iterable?
   a. No, because there is no relationship between Stack and Iterable.
   b. There is no relationship between Stack and Iterable shown. This may mean that it has been abstracted away or that it does not exist.
   c. Yes, because all types here are subclasses of Iterable.
   d. Yes, because there is an inheritance hierarchy shown in which Stack is a descendent of Iterable.

2. Assume that information that is absent in the diagram has **not** simply been abstracted away. When Stack is instantiated, how many other objects will be produced as well?
   a. 0
   b. 3
   c. 7
   d. 9

3. Is there a problem with AbstractList?
   a. Yes, a class cannot have two parents.
   b. Yes, it cannot be abstract or else we will not be able to instantiate it.
   c. No, it can have two copies of Collection; Java will sort out which one to use.
   d. No, because interface inheritance is different from implementation inheritance.

4. If a program wants to treat instances of Vector and Stack as objects of RandomAccess, can this cause problems?
   a. Yes, because these are different types.
   b. Yes, because Stack and Vector are classes and not interfaces.
   c. Yes, because the program will have to be specific about when it wants to use Vector's version of RandomAccess and when it wants to use Stack's version of RandomAccess.
   d. Yes, because RandomAccess might hide some of the methods of Vector and Stack.

5. Does the relationship between Iterator and Collection make sense?
   a. No, it is not possible for an interface to have this relationship with another interface.
   b. Yes, it means that every instance of Iterator will have a field of type Collection.
   c. Yes, it means that every instance of Iterator will have a field of type Collection and not vice versa.
   d. Possibly, but too many details have been abstracted away to be sure.

6. Are there any (potential) problems with the explicit relationships shown involving Iterable?
   a. Yes, they are inside separate file folders, so that could make things complicated.
   b. No, these are all legal relationships between interfaces.
   c. Yes, Iterator may be package-protected.
   d. No, since they are all inside the java package.

7. Imagine that some Iterator makes a call to add(:Object) on a variable of type Collection. Which implementation will be executed at run-time?
   a. The relationship between Iterator and Collection is not valid, so such a call is impossible.

b. The one on AbstractList, but ONLY if it is not overridden by the class of the object in that variable.
c. The one on Collection, since it is called on Collection.
d. The one on AbstractList, because of polymorphism.

8. Vector is shown as being involved in three relationships. Are there any other ones implied or possible?
a. In addition to hidden ones, it inherits the two dependencies shown.
b. There might be some other ones that are not shown on the diagram, due to abstraction.
c. Vector has a relationship with every type in this diagram.
d. It might be inside a subpackage of util.

9. Stack and Vector are both concrete classes. Is this a problem?
a. Yes, because you can get multiple objects when you instantiate one of them.
b. No, methods will be overridden as necessary.
c. Maybe, but we would need to see more details to be sure.
d. Yes, Vector should be abstract, or else Stack should be a subclass of AbstractList.

10. A program needs to use a ListIterator to go through the elements in a Stack. Are any changes needed to this diagram to model this?
a. Yes, there should be a relationship between ListIterator and Stack so they can exchange messages.
b. Possibly. ListIterator needs to have access to the contents of the Stack, but these details may have been suppressed.
c. No, ListIterator has access to Stack because it is in the same package.
d. Yes, the model should indicate the methods and relationships to be used in performing this operation.

~~11. Use Case Model Creation~~

~~Consider an online book shopping system described as follows.~~

> ~~*Customers can either browse or search. Browsing and searching are possible with respect to a set of categories (such as title, author, etc.). Customers can store selections either within a "shopping cart", or within a "wish list". Items can be switched between the shopping cart and wish list at any time. Customers may check out (and thus purchase) the items in their shopping cart.*~~

~~Draw a use case **diagram** to represent this natural language description as-is, **i.e., do not make assumptions to fill-in missing details**.~~

~~12. Use Case Model Creation~~

~~Consider an online airline reservation system described as follows.~~

> ~~*Customers can search for flights by selecting departure and arrival airports, dates, and other criteria. Customers can select a flight from the search results, to determine total cost and details of the trip. After having reviewed the details for the selected flight, the Customer can proceed to book it by providing credit card information and personal details; a booking reference is provided when a booking is successful. Customers can find information about their booked flights by providing the relevant booking reference.*~~

Draw a use case **diagram** to represent this natural language description as-is, **i.e., do not make assumptions to fill-in missing details**.

13. Use Case Model Creation

Consider an automated teller machine (ATM) system at a bank described as follows.

> *Customers must login to the ATM with their client card and PIN before they can proceed; this PIN is checked against the one stored by the bank remotely. An invalid PIN for the card will cause an error to be signalled to the customer, and the customer will remain logged out. The customer can deposit or withdraw money from their accounts, transfer between accounts, or query for the balance of the account; each of these actions requires the account(s) to be selected from a list and the amount of the transaction to be specified.*

Draw a use case **diagram** to represent this natural language description as-is, **i.e., do not make assumptions to fill-in missing details or to correct mistakes!**

14. Use Case Model Creation

Your client, a book publisher, wants you to develop a system for on-line book purchases. Customers should be able to browse through the list of books offered by the company. The customer may add or remove books from their "shopping cart" while browsing. Once the customer is finished selecting the book(s) that they want, they can place an order for the books, indicating the address to which the books should be shipped, and purchase them with a credit card.

A representative will receive the purchase order and fill it from the company's inventory. Initially, each order has a status of "pending"; the representative will inform the system when the order is shipped, thereby changing its status.

The customer should be able to ask the system about the status of their order. The customer should be able to exit the system without placing an order.

a) Construct a use case diagram for the on-line book purchase system in the space below.
b) In the space below, write the textual use case description for **one** of the use cases you have described that captures some of the requirements from the **second** paragraph of the description of the system.

15. Use Case Model Creation

You have been contracted by the Alma Mater Society (AMS) to construct a web-based system for conducting student surveys. Students use a web browser (such as Mozilla) to connect to a password-protected URL provided by a web server already in use by the AMS. Students then select the survey which they wish to fill-in (multiple surveys may be on-going at a time), and the appropriate survey form is displayed. If the student has previously filled-in this survey, they are not permitted to revise the survey. Each survey consists of a set of multiple choice questions. When the student is finished with the survey, they submit it to be recorded.

Construct a use case **diagram** for this system as described.

16. Use Case Model Creation [25%]

Consider an elevator system described as follows.

> *Passengers can request an elevator by pressing an "UP" or a "DOWN" button. Five seconds before an available elevator arrives at the floor where the passenger has made the request, audible and visible signals indicate the arrival and direction of travel. Upon arrival, the elevator doors open. Within the elevator, a passenger can request a particular floor by pressing the appropriate button. The elevator doors will close after 10 seconds unless an obstacle blocks the doors, or the "Open Door" button within the elevator is held down. A passenger should never wait more than 40 seconds for an elevator to arrive.*

(a) Draw a use case **diagram** to represent this natural language description as-is, **i.e., do not make assumptions to fill-in missing details**. Your target audience are technical people, but not experts in elevator design.

(b) Describe one non-functional requirement for this system, according to the natural language description above.

17. Use Case Model Creation

Consider a secure file transfer system described as follows.

> *Users can send one or more files to a location on a remote machine, and they can retrieve remote files, placing them at a location on the local machine. Any file transfer can be aborted either as a whole or on the basis on an individual file. The user can view the contents of the current remote directory and the current local directory; they can change the current local or remote directory. The encryption protocol for a session can be selected from a predefined set of possibilities. The user can specify the port number and user name to use for a given remote machine. A connection has to be opened to the remote machine, resulting in the user providing a password which must be authenticated prior to being able to perform any other functions.*

Draw a **use case diagram** to represent this natural language description as-is, **i.e., do not make assumptions to fill-in missing details or to correct mistakes!**

18. Requirements Analysis

Consider a simple printer system that allows users and administrators to be informed of when jobs are completed and when problems occur. Below is a natural language description of this system.

> *Printers print Files. Users send Files (called "print jobs") to Printers. Users should be notified when their own print jobs are done and not when other print jobs are done. Administrators are special Users. Administrators can register with individual Printers. Administrators who are registered with a given Printer are informed when any problems occur with it (paper jam, out of paper, out of ink, etc.). It is foreseen that special kinds of Printers, Users, and Administrators may be needed in the future.*

a) Draw a use case diagram to represent this natural language description as-is, **i.e., do not make assumptions to fill-in missing details**.
b) Draw an **object** diagram to represent the analysis object model for this system

~~19. Use Case Model Evaluation~~

~~Consider the use case diagram below, taken from Figure 5.7 of O'Docherty. It represents the functionality of an online car rental system called iCoot.~~

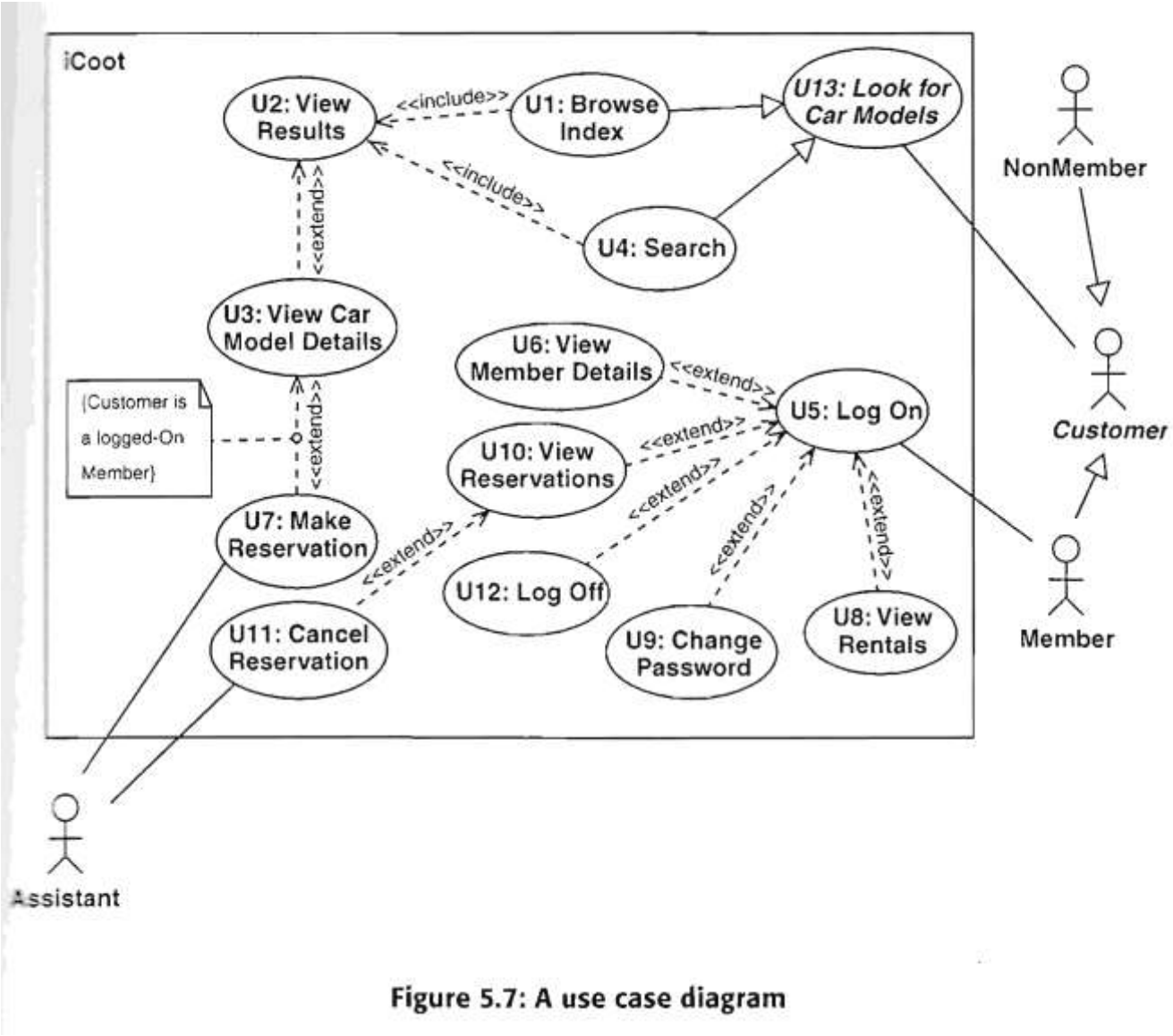
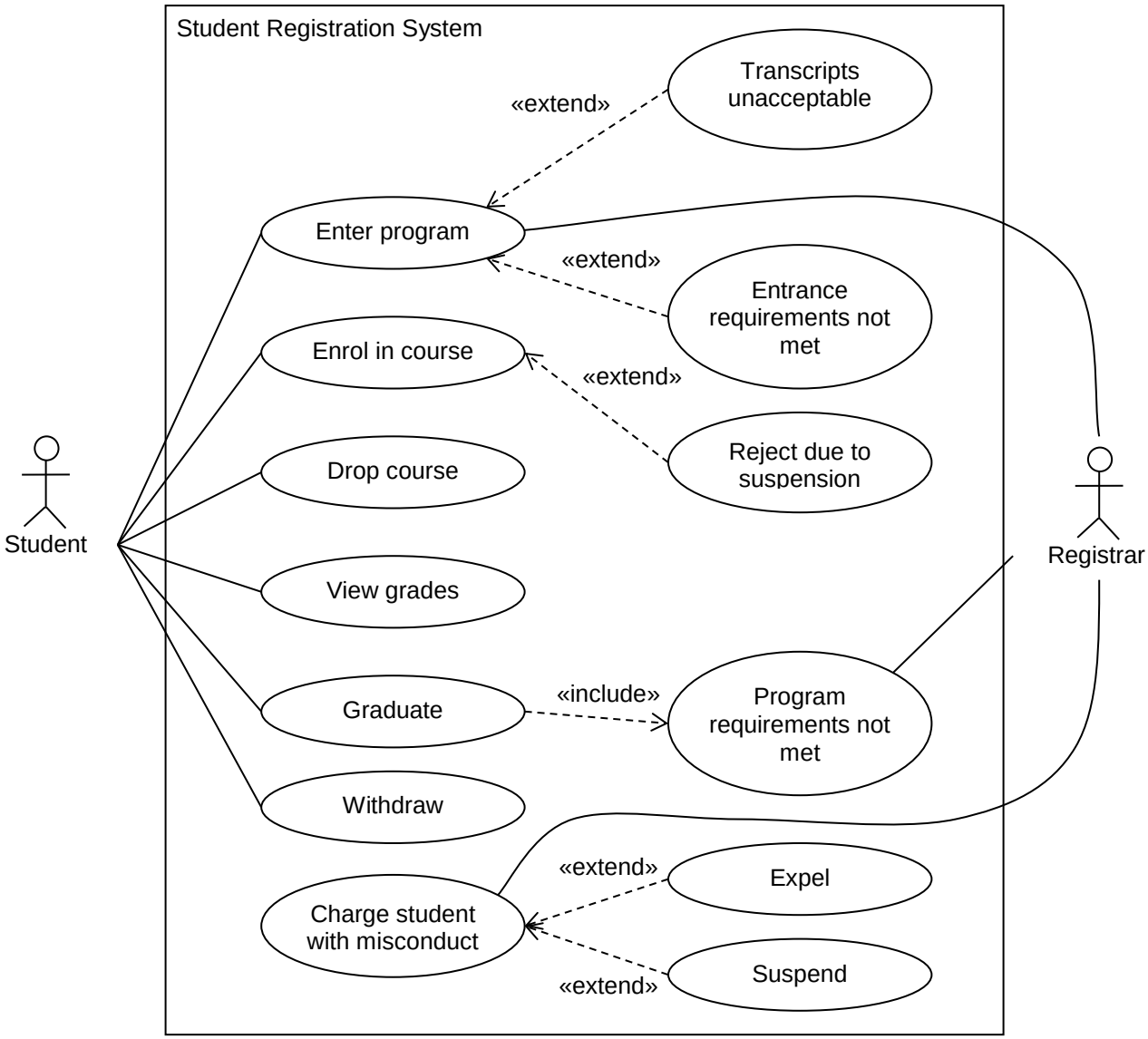


Figure 5.7: A use case diagram

~~Explain the behaviour of iCoot when someone wants to (a) view their existing reservations, (b) cancel one of these reservations, (c) search for a car model, and (d) change their password~~ **~~as implied by the diagram~~**~~. Begin your explanation from when this person first starts the system. Ensure that you fully specify as much information as the diagram indicates but no more than that.~~

~~20. Use Case Model Evaluation~~

~~Consider the use case diagram below of a student registration system. The Registrar actor represents someone who has the authority to approve transcripts, to check that entrance and program requirements are met, and to deal with academic and non-academic misconduct cases.~~



~~Answer the following questions **with respect to the diagram above**. Explain any **relevant mistakes** that you believe exist in the diagram.~~

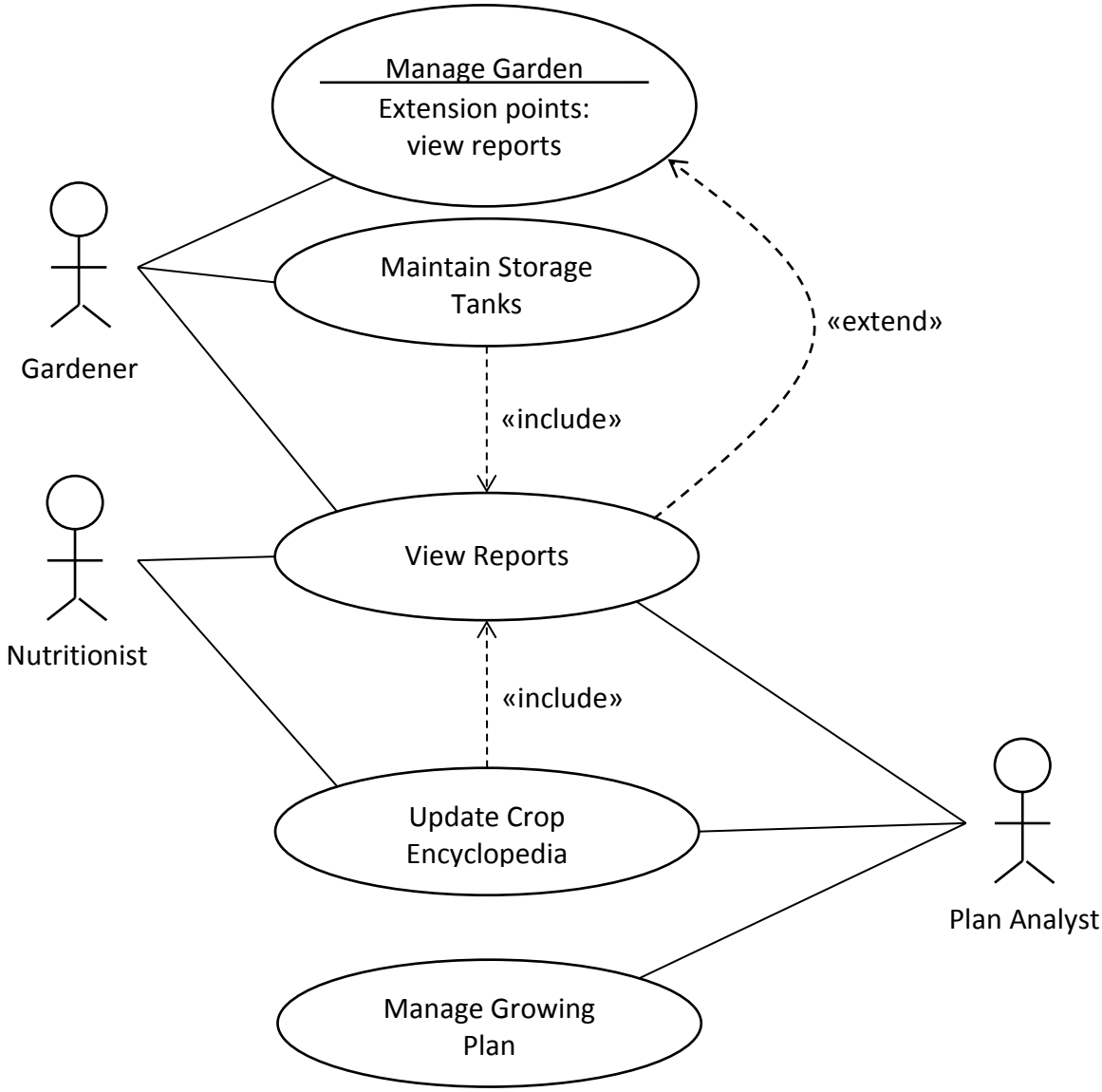
~~(a) Explain how the "Graduate" use case works (as completely as possible), according to the diagram.~~

~~(b) Explain how the "Charge student with misconduct" use case works (as completely as possible), according to the diagram.~~

~~(c) Is security an issue for concern in this system? How does the diagram suggest that security is to be dealt with?~~

21. Use Case Model Analysis

Consider the following use case diagram that I have reproduced from your textbook:



Don't go looking for it! That will waste your time, and won't tell you the answers …

(a) Explain under what circumstances the View Reports use case can occur, assuming that the diagram gives you a complete picture.

(b) I mentioned "assuming that the diagram gives you a complete picture" for a reason. Give two suggestions for ways in which the diagram might NOT give you a complete picture.

(c) If you think about what "View Reports" likely involves, there is something strange about this diagram according to what I said in class about the syntax of use case diagrams. Explain the issue and give two possible explanations for it.

22. Analysis Object Model

Draw the analysis object model that corresponds to the description in Question 1. Ensure that the model is sufficiently refined that all objects you identify are valid objects (as opposed to properly treated as attributes, classes, or relationships).

23. Analysis Object Model

You have been contracted by the Alma Mater Society (AMS) to construct a revised version of the web-based system for conducting student surveys. Students use a web browser (such as Mozilla) to connect to a password-protected URL provided by a web server already in use by the AMS. Students then select the survey which they wish to fill-in (multiple surveys may be on-going at a time), and the appropriate survey form is displayed. If the student has previously filled-in this survey, their previous entries are shown on the form to allow them to alter the values. Each survey consists of a set of multiple choice questions. When the student is finished with the survey, they submit it to be recorded.

Represent the analysis object model for this system in the space below, as a class diagram. Label associations and provide multiplicities as appropriate.

24. Textual analysis

a) Perform a parts-of-speech analysis on the following description to create an *analysis object model*:

> *Customers must login to the ATM with their client card and PIN before they can proceed; this PIN is checked against the one stored by the bank remotely. An invalid PIN for the card will cause an error to be signalled to the customer, and the customer will remain logged out. The customer can deposit or withdraw money from their accounts, transfer between accounts, or query for the balance of the account; each of these actions requires the account(s) to be selected from a list and the amount of the transaction to be specified.*

Draw an **object diagram** to represent the analysis object model.

b) Abstract and more thoroughly analyze your analysis object model, to arrive at a *static analysis model*.

Draw a **class diagram** to represent the static analysis model.

25. Static Analysis Model

Consider the following use case description:

Name: Select Pop
Participating actor(s): Customer
Precondition: none
Main flow of events:
1. Customer selects a kind of pop
2. The system vends the selected pop via the pop chute
3. The system subtracts the cost of the pop from the current total, and returns the result via the coin return
4. The system clears the display

Postcondition: Current total is 0
Alternative flow (Insufficient funds):

2. If the cost of the selected pop is greater than the current total, the system displays the cost of the pop for 3 seconds
3. The system redisplays the current total
4. The current total is not revised

Create a *static analysis model* for this use case, including its alternative flow. Represent this model as a **class diagram**. Ignore potential operations in your model; focus just on potential classes, their attributes, and their relations.

26. Static Analysis

Consider an elevator system described as follows.

> *Passengers can request an elevator by pressing an "UP" or a "DOWN" button. Five seconds before an available elevator arrives at the floor where the passenger has made the request, audible and visible signals indicate the arrival and direction of travel. Upon arrival, the elevator doors open. Within the elevator, a passenger can request a particular floor by pressing the appropriate button. The elevator doors will close after 10 seconds unless an obstacle blocks the doors, or the "Open Door" button within the elevator is held down. A passenger should never wait more than 40 seconds for an elevator to arrive.*

(a) Identify the nouns and verbs in the description above.

(b) Construct an analysis class model for this system. [Feel free to start from an object diagram, but I am interested only in the analysis class model.]

27. Dynamic Analysis Model

Consider the following use case description:

Name: Insert Coin
Participating actor(s): Customer
Precondition: none
Main flow of events:
1. Customer inserts a coin
2. Value of coin is added to current total
3. Revised current total is displayed for the Customer

Postcondition: Current total is updated
Alternative flow:
2. Invalid coin is returned to Customer
3. Current total is not revised

Create a dynamic analysis model for this use case, including its alternative flow.

Draw a **communication diagram** that represents this dynamic analysis model.

28. Dynamic Analysis Model

Consider the following use case description:

Name: Select Pop
Participating actor(s): Customer
Precondition: none
Main flow of events:
1. Customer selects a kind of pop
2. The system vends the selected pop via the pop chute
3. The system subtracts the cost of the pop from the current total, and returns the result via the coin return
4. The system clears the display
Postcondition: Current total is 0
Alternative flow (Insufficient funds):
2. If the cost of the selected pop is greater than the current total, the system displays the cost of the pop for 3 seconds
3. The system redisplays the current total
4. The current total is not revised

Create a *dynamic analysis model* for this use case, including its alternative flow. Represent this dynamic analysis model by drawing a **communication diagram**.

29. Requirements Analysis

You have been employed by the Greater Vancouver Transportation Authority (TransLink) to develop a traffic-light-priority system for buses at specific intersections. Whenever a bus approaches a prioritized intersection, the timing of the traffic light there (i.e., how long it remains green, amber, or red) is affected, to give priority to buses over cars.

During a requirements review, you read a requirements specification that states:

1. Buses shall each possess a transmitter to communicate with prioritized intersections.
    1.1. Each transmitter shall transmit signals adhering to the IBP communication protocol.
        1.1.1. Each transmitter shall be capable of sending a signal of sufficient strength to be received by a prioritized intersection receiver at a distance of up to 5000 m.
    1.2. Each transmitter shall occupy a volume no greater than 1.0×10^3 m^3.
2. Prioritized intersections shall each possess a receiver to communicate with buses.

Based *only* on the information given above, comment on the following properties of the requirements.

(a) Realism

(b) Consistency

(c) Completeness

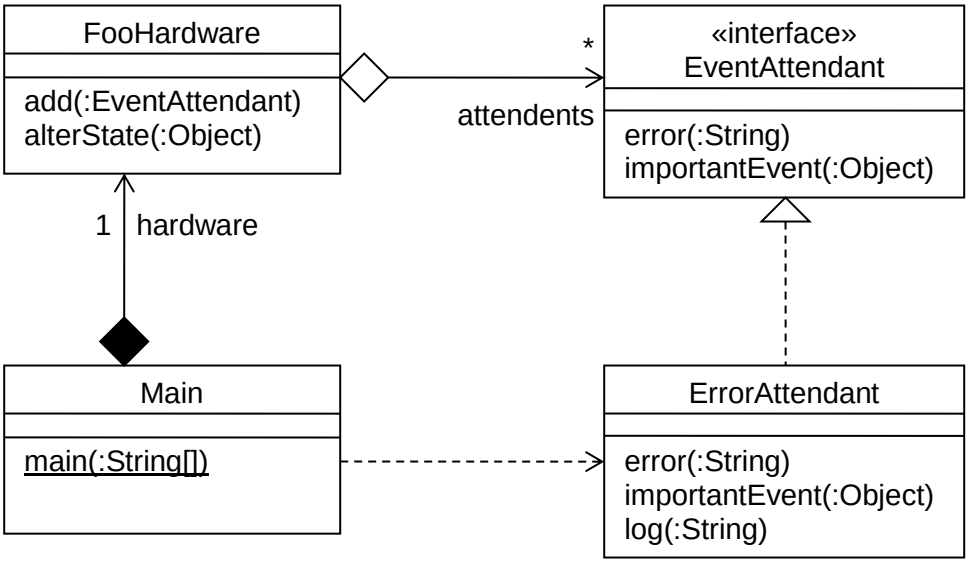
(d) Validity

(e) Verifiability

(f) Comprehensibility

30. Design Patterns

The design below applies the Observer pattern. `Main` creates and adds an `ErrorAttendant` object to the `FooHardware` instance that it has also created. `ErrorAttendant` only reacts to errors (i.e., it does nothing when an "important event" happens); when an error occurs it records the details via its `log()` method.



Explain which details (if any) in this design fulfill the following roles in the Observer design pattern.

Abstract Subject:	means of registration:
Concrete Subject:	means of deregistration:
Abstract Observer:	event handlers:
Concrete Observer:	cause of event:

Explain which details (if any) differ from the prototypical Observer pattern.

31. Design Evaluation

Consider the design from the previous question. **Evaluate it** on the following design principles:

(a) Minimize coupling

(b) Maximize cohesion

(c) Keep it simple

(d) Hide information

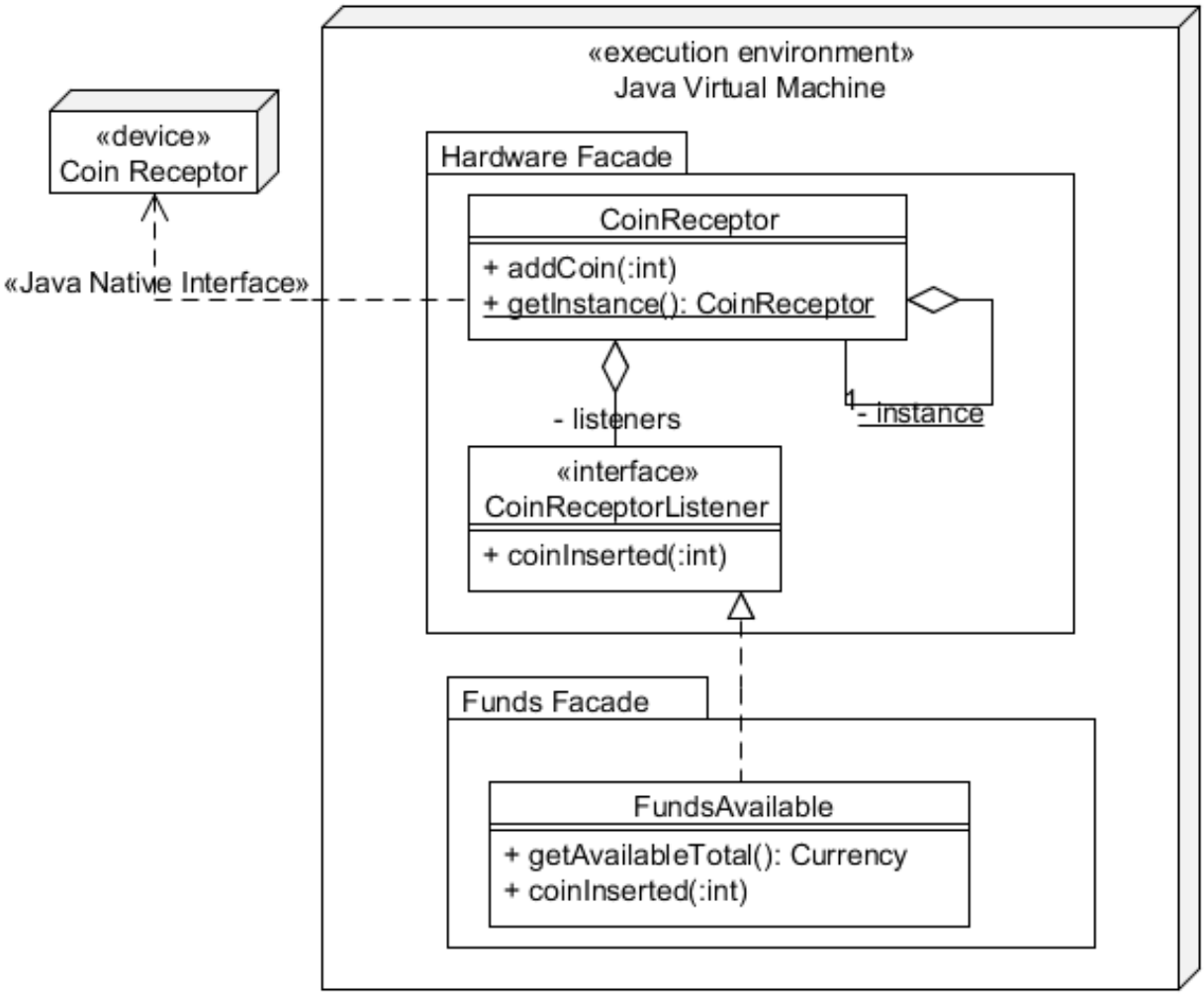
Is it a good design? Justify your answer.

32. Design Patterns

The design below, for part of the vending machine, applies the Observer pattern.

The `CoinReceptor` class uses the Coin Receptor device to notice when someone has entered a coin in the machine; the method `addCoin` can be called to simulate this addition. The `coinInserted` method is called by `CoinReceptor` on registered `CoinReceptorListener` instances to inform them that coins have been inserted; the value of the coin in cents is passed as an argument. The remainder of the important details should be evident from the diagram.

Hint: Not all details here will be relevant to your tasks. This design is a simplified version of the one discussed in lectures.

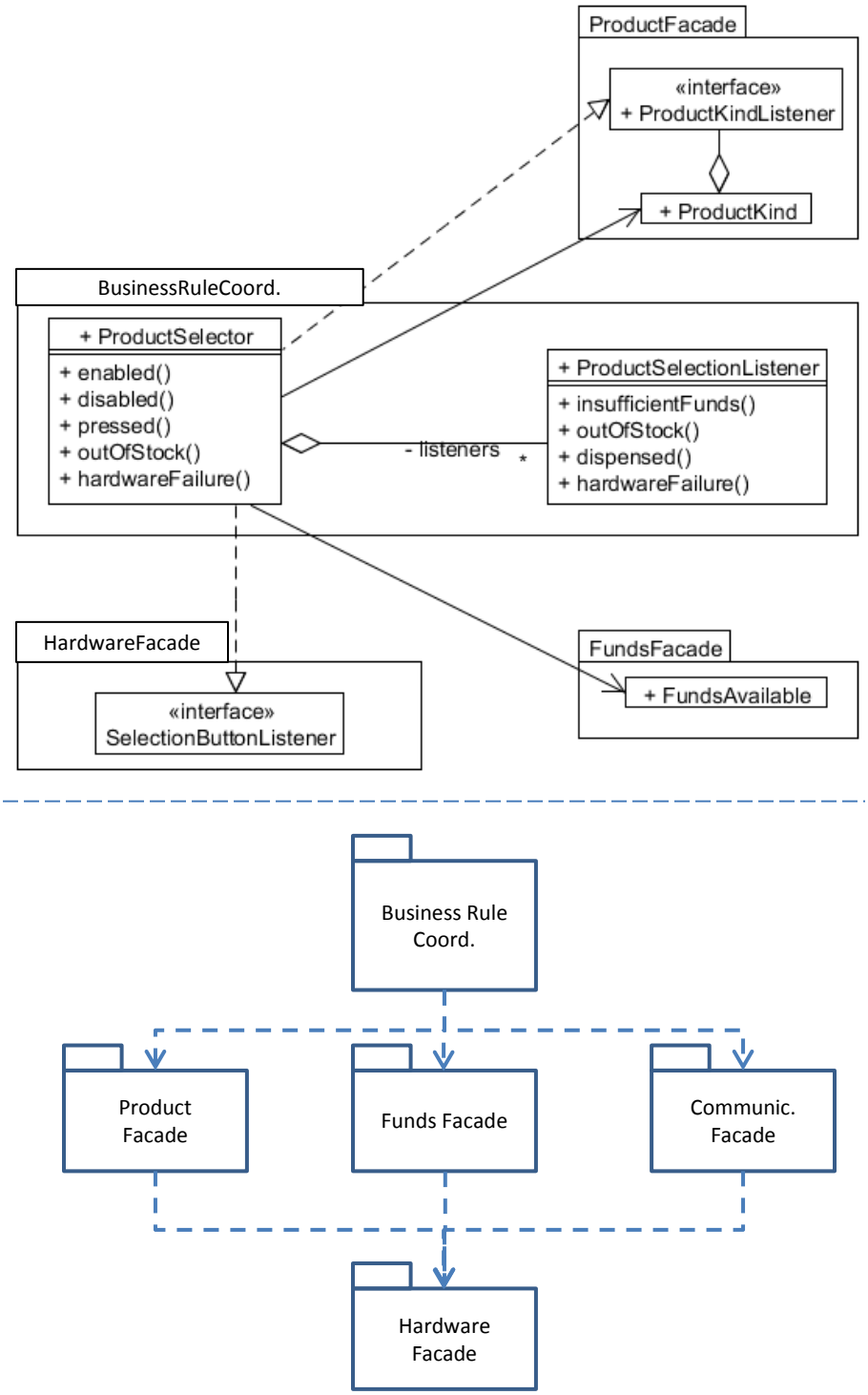


Explain which details (if any) in the shown design fulfill the following roles in the Observer design pattern. Use "NOT SHOWN" if the item has to be there but isn't shown in the diagram; use "NOT APPLICABLE" if the item isn't shown and is not necessary for this system to operate correctly.

Abstract Subject:	means of registration:
Concrete Subject:	means of deregistration:
Abstract Observer:	event handler(s):
Concrete Observer:	cause of event:

Explain which details (if any) differ from the prototypical Observer pattern:

33. Design Model Correctness and Correction



Here are two diagrams connected to the vending machine we discussed in lectures. At top is a view on the Business Rule Coordination package with its attendant dependencies. At bottom is a high-level view of the architectural structure of the system.

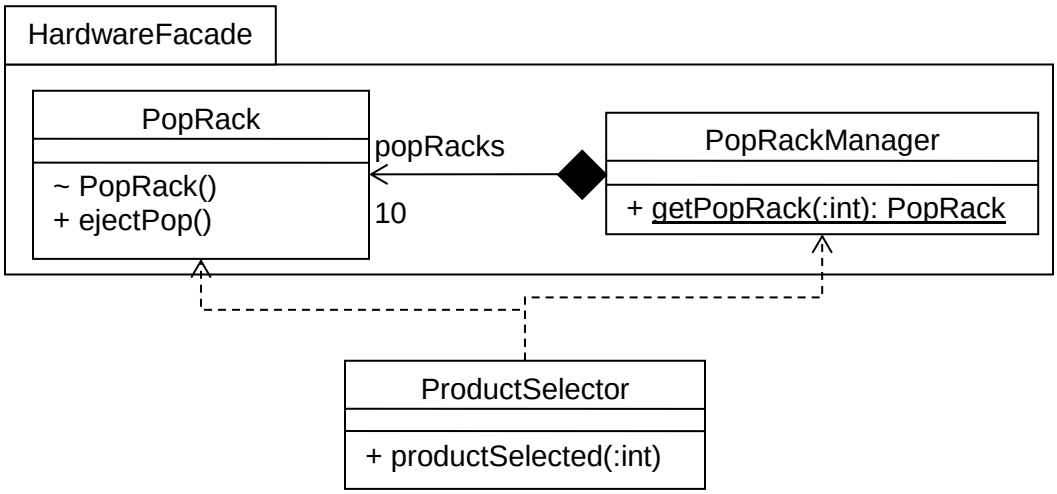
a. There are two inconsistencies between the diagrams. **Describe** them.

b. Explain how the inconsistencies can be corrected, without altering the design expressed in the architectural view.

c. Should the diagrams themselves be changed in the way you describe in (b)? Justify your answer.

34. Design Patterns

The design below applies the Singleton pattern. Although it is part of the design for the vending machine software, it is different than the ones I talked about in class. `PopRackManager` creates and tracks a set of `PopRack` instances. A `ProductSelector` instance uses `PopRackManager` to look up a particular `PopRack` instance, on which `ProductSelector` can then call `ejectPop()`.



(a) **Explain** which details (if any) in this design fulfill the following roles in the Singleton design pattern.

Singleton class:	getData( ):
getInstance( ):	uniqueInstance field:
Hidden constructor:	otherData field:

(b) **Explain** which details (if any) differ from the ordinary use of the Singleton pattern:

35. Design Evaluation

Consider the design from the previous question. **Evaluate it** on the following four design principles:

(a) Minimize coupling

(b) Maximize cohesion

(c) Keep it simple

(d) Hide information

(e) The design is different from the prototypical Singleton design pattern. Is that a problem? Justify your answer.

36. Refactoring

Consider the following implementation:

```
public abstract class SuperDuper {
}

public class A extends SuperDuper {
    public void doSomethingWonderful(int i) {
        System.out.println("Wow, this is great!");
        System.out.println("Result: " + (Math.signum(i) + Math.exp(i)));
    }
}

public class B extends SuperDuper {
    public void busyWork(int count) {
        for(int i = 0; i < count; i++) {
            System.out.println("Result: " + (Math.signum(i) + Math.exp(i)));
        }
    }
}
```

(a) **Refactor** this implementation to **maximally eliminate duplication**. (You may write on top of the implementation above, copy out the full implementation, or any combination thereof. The point is whether you understand what the task involves: make it clear what you intend.)

(b) **Explain** when would this refactoring be a good idea to pursue, and when would it not?

37. Testing

Consider the source code on the next page, which is a (small) portion of the implementation of the Vector class in the Java Standard Libraries. Vectors should expand when the number of elements in them would otherwise exceed their capacity, according to some specific rules. Your task will be to test whether the expansion functionality works as specified.

(a) **Explain** what calls should be made within a test driver to execute **any** such test case:

```
package java.util;

public class Vector {
    protected Object[] elementData;
    protected int elementCount;
    protected int capacityIncrement;

    public Vector(int initialCapacity, int capacityIncrement) {
        super();
        if (initialCapacity < 0)
           throw new IllegalArgumentException("Illegal Capacity: "+ initialCapacity);
        this.elementData = new Object[initialCapacity];
        this.capacityIncrement = capacityIncrement;
    }

    /**
     Returns the current capacity of this vector.
     */
    public int capacity() {
        return elementData.length;
    }

    /**
      Increases the capacity of this vector, if necessary, to ensure that it can hold
      at least the number of components specified by the minimum capacity argument.
      If the current capacity is >= minCapacity, the capacity should not change.

      If the current capacity of this vector is less than minCapacity, then its
      capacity is increased by replacing its internal data array, kept in the field
      elementData, with a larger one.

      If the value of capacityIncrement is greater than 0, the size of the new data
      array will be the old size plus capacityIncrement.

      If the value of capacityIncrement is less than or equal to zero, the new
      capacity will be twice the old capacity.

      If this new size is still smaller than minCapacity, then the new capacity will
      be minCapacity.
     */
    public void ensureCapacityHelper(int minCapacity) {
        int oldCapacity = elementData.length;
        if (minCapacity <= oldCapacity) {
            Object[] oldData = elementData;
            int newCapacity = (capacityIncrement > -1) ?
                       (oldCapacity + capacityIncrement) : (oldCapacity * 2);
            if (newCapacity == minCapacity) {
                newCapacity = minCapacity;
            }
            elementData = Arrays.copyOf(elementData, newCapacity);
        }
    }
}
```

(b) Using a **boundary-based test-case selection** technique combined with an **equivalence-based test-case selection** technique, define a minimal set of test cases to satisfy the goal. Fill in the table with the details of your test cases. **NOTE:** The table has 10 rows for test cases. If this is too many, don't use them all; if it isn't enough, add more on the back of the page. **SUGGESTION:** Figure out the answer before filling in the table!

Purpose	Specific input(s)	Expected result(s)

38. Testing

Consider the source code on the next page, which implements the `Enumeration` interface type in the Java Standard Libraries. `Enumeration`s are intended to provide a simple means to visit each element

in some collection, and to inform whether or not any elements remain to be visited. Your task will be to create test cases that check whether `nextElement()` works as specified, for ANY `Enumeration`.

```
package java.util;

public interface Enumeration {

  /**
   * Tests if this enumeration contains more elements.
   *
   * @return  <code>true</code> if and only if this enumeration object
   *           contains at least one more element to provide;
   *          <code>false</code> otherwise.
   */
   boolean hasMoreElements();

   /**
    * Returns the next element of this enumeration if this enumeration
    * object has at least one more element to provide.
    *
    * @return     the next element of this enumeration.
    * @exception  NoSuchElementException  if no more elements exist.
    */
   Object nextElement();
}
```

Here is an example implementation of an `Enumeration` aimed specifically at visiting all the elements in an array:

```
package ca.lsmr.util;

public class ArrayEnumeration {
  private Object[] _array;
  private int _index;

  public ArrayEnumeration(Object[] array) {
    _array = array;
    _index = 0;
  }

  public boolean hasMoreElements() {
    return _index <= _array.length;
  }

  public Object nextElement() {
    if(_index < _array.length) {
      throw new NoSuchElementException();
    }
    return _array[++_index];
  }
}
```

Using a **boundary-based test-case selection** technique combined with an **equivalence-based test-case selection** technique, define a minimal set of test cases **to satisfy the goal**. Fill in the table with the details of your test cases. **NOTE:** The table has 10 rows for test cases. If this is too many, don't

use them all; if it isn't enough, add more on the back of the page. **SUGGESTION:** Figure out the answer before filling in the table!

Purpose	Specific input(s)	Expected result(s)

39. Testing

Consider the following implementation, representing a simplified version of the standard `String` class:

```
public final class String {
    private final char[] characters;

    public String(char[] charArray) {
        characters = charArray;
    }

    /**
     * Concatenates the passed string at the end of this string, and returns a new
     * string.  This string and the passed string remain unchanged.
     *
     * @throws NullPointerException if s is null
     */
    public String concat(String s) {
        char[] newArray = new char[characters.length + s.length];
        System.arraycopy(characters, 1, newArray, 1, characters.length);
        System.arraycopy(s.characters, 1,
                         newArray, characters.length, s.characters.length);
        return new String(newArray);
    }
}
```

Use the following skeleton of a JUnit test suite (the required import statements are hidden) to create a test suite for the `concat(:String)` method:

```
public class StringTest {
    @Test public void testConcatX() {
        String s1 = <string 1>;
        String s2 = <string 2>;
        String result = s1.concat(s2);
        assertTrue(<string to check>.equals(<expected string>));
    }

    @Test public void testConcatExceptionX() {
        String s1 = <string 1>;
        String s2 = <string 2>;
        try {
            String result = s1.concat(s2);
            fail("NullPointerException expected");
        }
        catch(NullPointerException expected) {
        }
    }
}
```

The first test case form (`testConcatX`) can be used where you expect that the concatentation will work and return a particular string; the second form (`testConcatExceptionX`) can be used where you expect that the result will cause a `NullPointerException`. In both forms, some details are missing.

Fill in the table on the next page with a set of test cases, chosen using boundary-based test case selection and equivalence-based test case selection combined (black-box approaches!). I suggest that you figure out what the test cases ought to be before you fill in the table. You should strive for a minimum number of cases to satisfy the selection criteria. This may mean that you don't need to use every row in the table.

Are you using `testConcatX` for this test case? (Y/N)	***<string 1>***	***<string 2>***	***<string to check>***	***<expected string>***

40. Testing

Consider the source code below, which is a simple class being developed to read files and identify their type (e.g., text file, JPEG image, etc.). The job of the actual recognition is intended to be split up into individual file type interpreters, one per type of file; each of this will extend the `FileTypeInterpreter` interface shown.

```
public interface FileTypeInterpreter {
    /** Returns the name of the kind of file this interpreter can recognize */
    String getType();

    /** Tests whether this interpreter recognizes the file type for this file.
        An IOException is thrown if the file is not readable for any reason. */
    boolean isRecognizable(File file) throws IOException;
}

public class FileTypeProcessor {
    private Vector<FileTypeInterpreter> interpreters = new Vector<>();

    /** Adds the FileTypeInterpreter to the set of registered ones. Ignores
        repeated entries.  Throws a NullPointerException if fti is null.*/
    public void register(FileTypeInterpreter fti) {
        interpreters.add(fti);
    }

    /** For the given file, checks against each registered FileTypeInterpreter to
        determine if the file type is recognizable.  Returns the first recognized
        type. An empty string is returned if no registered FileTypeInterpreter
        recognizes the file type. Null is returned if no registered
        FileTypeInterpreter recognizes the file type AND at least one threw an
        exception. */
    public String type(File file) {
        for(FileTypeInterpreter fti: interpreters) {
            if(fti.isRecognizable(file)) {
                return fti.getType();
            }
        }
        return null;
    }
}
```

At this point, no specialized classes that implement `FileTypeInterpreter` have been created, but your job is to **unit test** the `FileTypeProcessor` class. To do so, you may use this stub class:

```
public class FileTypeInterpreterStub implements FileTypeInterpreter {
    public String type;
    public boolean throwException = false;
    public boolean returnValue = false;

    public String getType() { return type; }
    public boolean isRecognizable(File file) throws IOException {
        if(throwException) {
            throw new IOException();
        }
        return returnValue;
    }
}
```

(a) Explain how the stub can be used, in general, to set up a test case for unit testing `FileTypeProcessor`.

(b) Explain what test cases ought to be tried out, if you had a limited time to create the test suite and run it, but `FileTypeProcessor` is critical to the functionality of the system. Consider a combination of path-based, equivalence-based, and boundary-based test case selection techniques. For each test case, explain set up (if your answer to (a) doesn't already do so), inputs, and expected results.