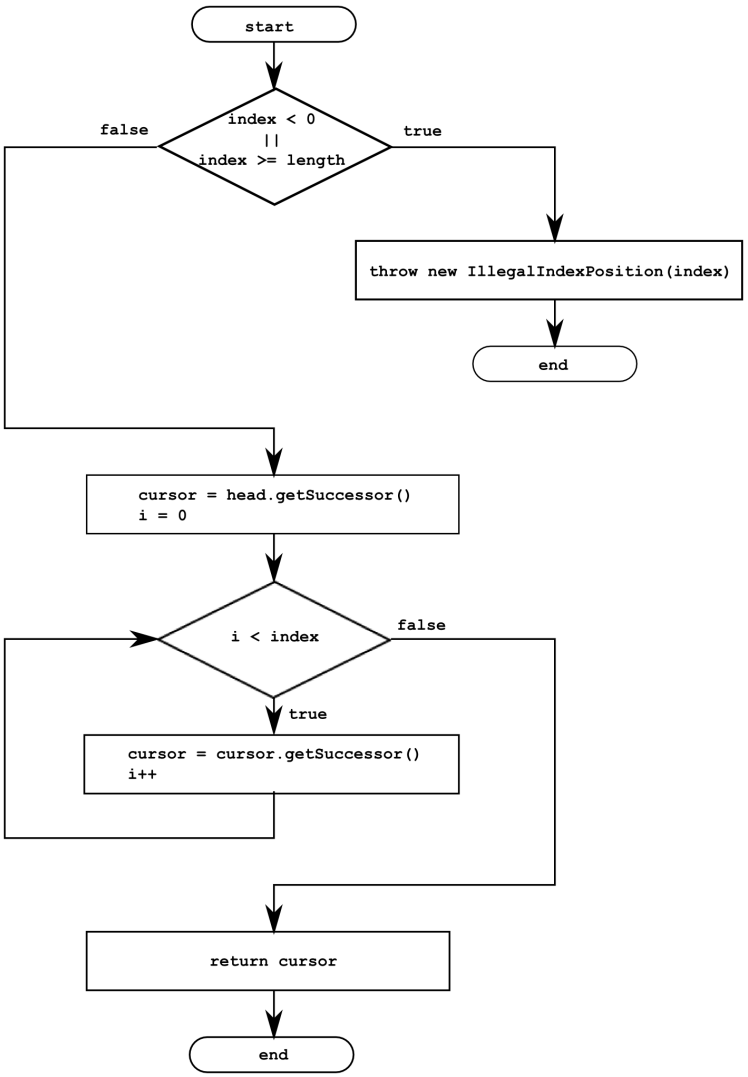
CPSC 210
Sample Final Exam Questions - Solution

Don't even think about looking at these solutions until you've put significant effort into developing your own solution to these problems!!!

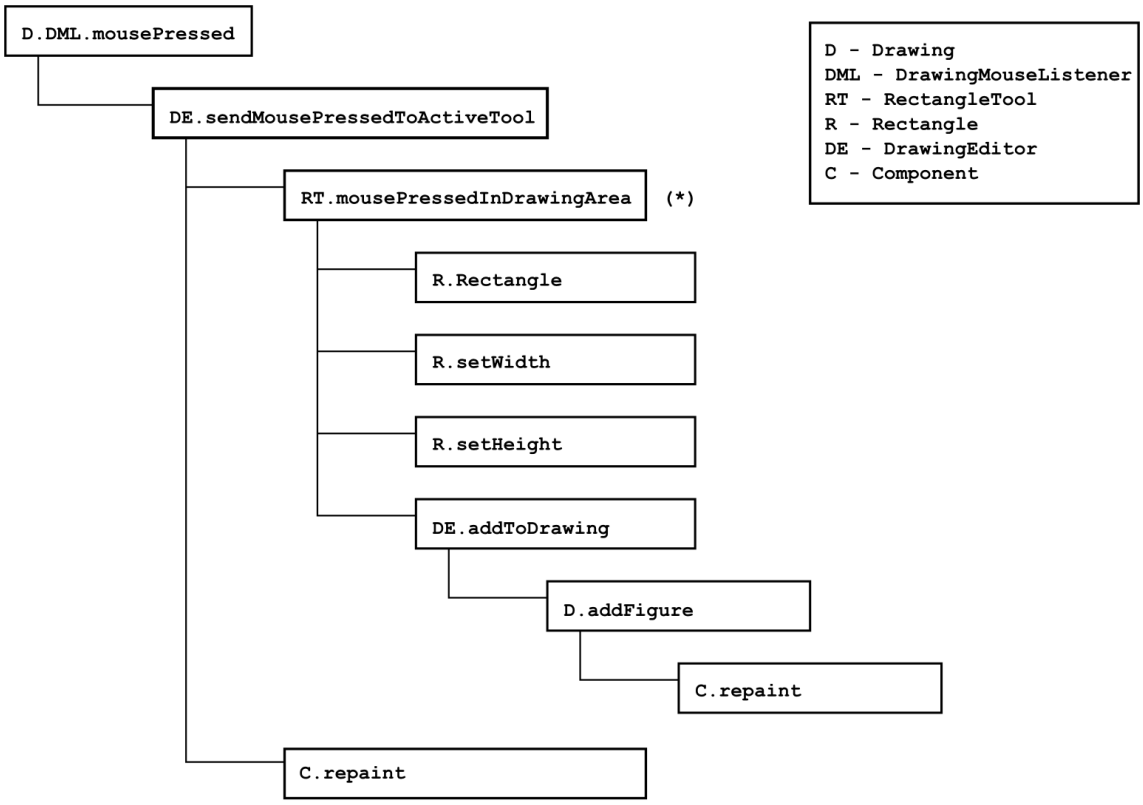
Q1. Draw an intra-method control flow diagram for the *iterative version* of the method:

```
MyListNode<E> find(int index) throws IllegalIndexPosition;
```

in the `ubc.cpsc210.list.linkedList.MyLinkedList` class of the project `\lectures\LinkedListComplete`



Q2. Draw a call graph starting at `DrawingMouseListener.mousePressed` in the `Drawing` class of the `\lectures\GUI4Complete` project. Assume that the active tool is a `RectangleTool` and use this information to resolve any calls to abstract methods in the `Tool` class. Include calls to methods in the `ca.ubc.cpsc210.drawingEditor.*` packages only, including methods inherited from super-types.



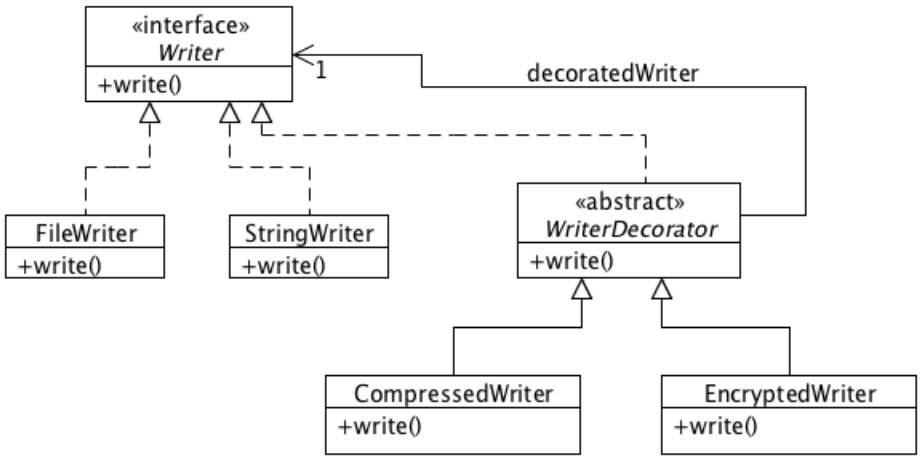
(*) Here's where we use the fact that the active tool is a RectangleTool

Q3. Consider the type hierarchy shown in the UML diagram in part b) below. The `FileWriter` class enables us to write data to a file while the `StringWriter` class enables us to write to a string.

a) Suppose we want to be able to compress the output that is written to file or string, or encrypt that output, or both. However, we don't want to have to make any changes to the existing hierarchy, as it took considerable effort to develop, test and debug it. What design pattern will you use?

The Decorator Pattern

b) Modify the UML diagram below to show how you will apply the design pattern that you identified in a). (Parameters for `write()` have been omitted.)



The following was not required but provides you with a bit more detail about how this will work. Suppose we want to compress some data and write it to file…

```
CompressedWriter cw = new CompressedWriter(
                                new FileWriter(/* path to file */));
cw.write(/* some data */);
```

The implementation of the write method in `CompressedWriter` goes something like:

```
public void write(/* some data */ ) {
   // - first compress the data that was passed in as a parameter
   // - in the context of the example above, the line below calls
   //   FileWriter's write method which writes the compressed
   //   data to file
   decoratedWriter.write( /* pass compressed data */ );
}
```

Q4a) What does it mean for the design of a class to be robust?

First, all of the methods in the class must be robust. A method is robust if its specification covers all possible input values to that method.

In addition, we must specify class invariants and ensure that they hold before any method in the class is called and immediately after any of those methods executes. The invariants specify what an operation can assume about the state of an instance of that class at any time.

b) Design and implement a class that represents an *unchecked* exception that will be thrown by the following method of the `MyArrayList<E>` class when the index is not valid:

```
public E get(int index);
```

The class must provide a constructor that takes the invalid index as a parameter and uses it to construct a meaningful error message that can be displayed when the exception is caught.

```
/**
 * Represents an exception raised when an illegal index
 * is used.
 */
public class IllegalIndexPosition extends RuntimeException {
     /**
      * Constructor
      * @param index  the illegal index
      */
     public IllegalIndexPosition(int index) {
          super("The index " + index + " is not valid.");
     }
}
```

Q5. Define the Open-Closed Principle and provide an example of how it might be applied.

The Open-Closed Principle states that a system must be open for extension but closed for modification. This principle can be applied in the context of framework/library design. For example, the Java Swing library is a framework for developing GUI applications. This framework is open for extension but closed for modification because we can extend the basic functionality provided by the framework to develop new GUI applications without having to modify the framework itself. The framework is considered closed to modification, as we would never attempt to change the code provided in the framework - it's extremely complicated and it works well, so let's not mess with it!

Q6. Suppose that a friend of yours has designed a type hierarchy. At one point in the hierarchy a subtype overrides a method and throws a checked exception that is not thrown by the overridden method in the super-type. The code does not compile. In terms of a design principle studied this term, explain why you would not expect such code to compile.

In light of the Liskov Substitution Principle (LSP), we would not expect the code to compile because the overriding method in the subtype requires more of the client than the corresponding method in the super-type. Given that the overriding method in the subtype throws a checked exception, the client is required to handle the exception when calling that method. This involves *adding* code to the client that either catches the exception or propagates it and therefore requires more of the client code.

Q7. Suppose you have been asked to design software for a company called *WhereNotToStay* that allows users to write reviews for hotels, motels and hostels at which they have stayed. Hotels, motels and hostels provide different types of accommodation and so the way that they are described is different. Hotels, for example, include a rating on a 5-star scale, whereas motels include information about the number of parking spaces available and hostels about the number of beds per dorm. The system can print a description for any of these different types of accommodation. The review that users submit has the same form for all of these different accommodation types.

Users must register with the system before they are allowed to post a review. Hotels, motels and hostels can be added to or removed from the system. When any of them is removed from the system, the corresponding reviews are not removed. If a user is removed from the system, their reviews are not removed. For any given hotel, motel or hostel, we must be able to list the corresponding reviews. For any given user, we must be able to list the reviews that they have posted.

a) Identify candidate classes. Write a brief description of the purpose of each and list the major responsibilities of each.

WhereNotToStay: tracks the accommodation and user managers.
Responsibilities: get accommodation manager, get user manager

AccommodationManager: manages accommodations registered with the system
Responsibilities: add/remove accommodation, get accommodation

UserManager: manages users registered with the system
Responsibilities: register user, remove user, get user

Accommodation: represents an accommodation that can be reviewed by users
Responsibilities: get/set description, add review, get review

Hotel: represents a hotel
Responsibilities: set description, get/set star rating

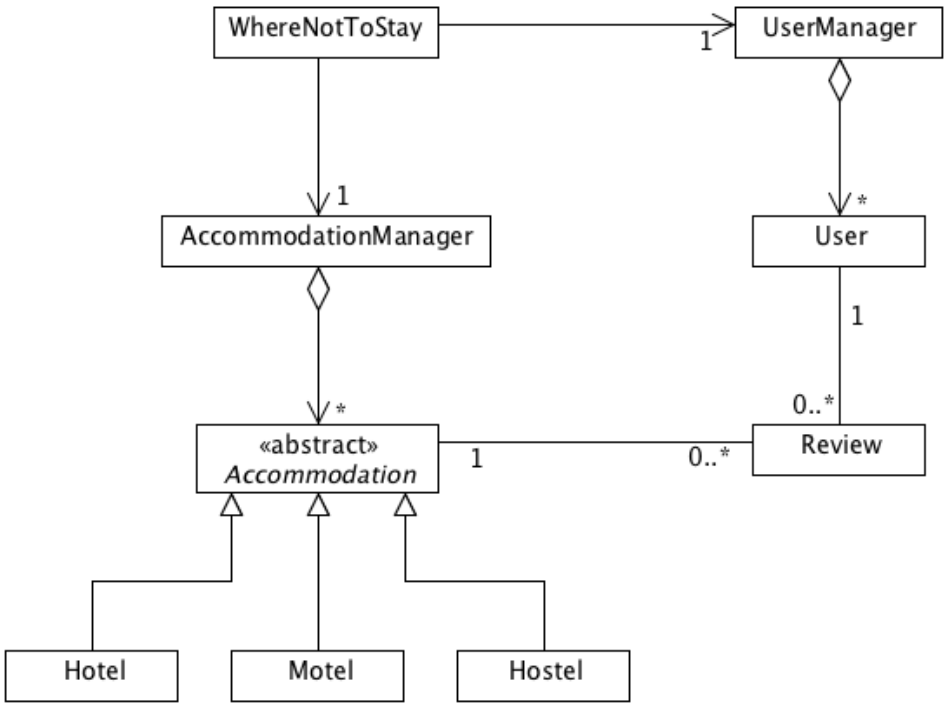
Motel: represents a motel
Responsibilities: set description, get/set number of parking spaces

Hostel: represents a hostel
Responsibilities: set description, get/set number of beds per room

User: represents a registered user
Responsibilities: add review, get review, get/set name

Review: represents a review of an accommodation by a user
Responsibilities: get/set user, get/set accommodation, get/set review content

b) Draw a UML diagram to represent your design. Include all classes and interfaces. Indicate interfaces and abstract classes using <<...>> annotations. Include multiplicities – any multiplicity not included will be assumed to have the value 1. Do not include fields or methods.



Note: on the assumption that we wish to display some information about the user that wrote a review whenever a review is displayed, the relationship between `User` and `Review` is bidirectional.

Q8. This question refers to the class `ubc.cpsc210.list.linkedList.MyLinkedList` from the `\lectures\LinkedListComplete` project.

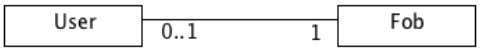
a) Complete the implementation of the following member of the `MyLinkedList` class. Assume that `Collection<E>` is from the `java.util` package.

```
/**
 * Returns true if this list contains all the elements in the
 * collection c, false otherwise.
 */
public boolean containsAll(Collection<E> c) {
    for (E next : c) {
        if (!contains(next))
            return false;
    }

    return true;
}
```

b) Provide an iterative and recursive implementation of the following member of the `MyLinkedList` class:

```
/**
 *  Returns the index of the given element or -1 in the case when the
 *  element is not in the list.
 */
// Iterative version
public int indexOf(E element) {
    MyListNode<E> cursor = head.getSuccessor();
    int index = 0;

    while (cursor != tail) {
        if (cursor.getElement().equals(element))
            return index;
        index++;
        cursor = cursor.getSuccessor();
    }

    return -1;
}

// Recursive version
public int indexOf(E element) {
    return indexOfHelper(head.getSuccessor(), element, 0);
}

private int indexOfHelper(MyListNode<E> cursor, E element, int acc) {
    if (cursor == tail)
        return -1;
    else if (cursor.getElement().equals(element))
        return acc;
    else
        return indexOfHelper(cursor.getSuccessor(), element, acc + 1);
}
```

Q9. Provide an implementation for the classes shown in the UML diagram below. You must include any fields or methods necessary to support the relationship between the classes but no other fields or methods are required.




```
/**
 * Represents a fob
 */
public class Fob {
    private User theUser;

    /**
     * Constructor sets this fob's user to null
     */
    public Fob() {
        theUser = null;
    }

    /**
     * Gets the user of this fob
     * @return the user
     */
    public User getUser() {
        return theUser;
    }

    /**
     * Sets the user for this fob
     * @param u  this fob's user
     */
    public void setUser(User u) {
        if (theUser != u) {
            theUser = u;
            theUser.setFob(this);
        }
    }
}
```

```
/**
 * Represents a user.
 */
public class User {
    private Fob theFob;

    /**
     * Constructor
     * @param f  this user's fob
     */
    public User(Fob f) {
        setFob(f);
    }

    /**
     * Gets this user's fob
     * @return the fob
     */
    public Fob getFob() {
        return theFob;
    }

    /**
     * Sets the fob for this user
     * @param f  this user's fob
     */
    public void setFob(Fob f) {
        if (theFob != f) {
            theFob = f;
            theFob.setUser(this);
        }
    }
}
```

Q10. For each of the scenarios below, identify which collection from the Java Collections Framework you would use and briefly justify your answer.

a) Suppose you want to simulate line-ups at a bank. There can be anywhere from one to several tellers available at any given time and each teller has their own line-up of customers. Tellers are numbered sequentially starting at position 0. You want to be able to get the line-up for a particular teller station by specifying the teller's position number. If the teller at position 2, for example, is absent, the line-up is `null`. Assume that there is a `Customer` class in the system. How would you represent the collection of line-ups?

`List<Queue<Customer>>` or `ArrayList<LinkedList<Customer>>`

Each line-up of customers can be represented by a `Queue<Customer>` which maintains entries in First-In First-Out (FIFO) order. Note that `Queue` is an interface that is implemented by `LinkedList`, so we could therefore choose `LinkedList<Customer>` to represent each line-up of customers. Given that we have more than one line-up and that we want to have positional access to the collection of line-ups, we choose `List<Queue<Customer>>`. We cannot use `Set<Queue<Customer>>` as a `Set` does not provide positional access. When choosing a particular implementation of `List`, we go with `ArrayList` as the total number of teller stations is not likely to change often.

b) Suppose you are designing a course registration system. Assume that there is a `Student` and a `Course` class in the system. How would you store the students and courses so that you can quickly retrieve the courses in which a given student is registered?

`Map<Student, Set<Course>>` or `HashMap<Student, HashSet<Course>>`

For each student we want to be able to quickly retrieve the courses in which he/she is registered. We therefore use a map with the student as key and the courses in which the student is registered as the corresponding value. Given that a student won't register in a course more than once at any given time, we represent the collection of courses using a `Set`. We use `HashMap` and `HashSet` rather than `TreeMap` and `TreeSet` because there is no requirement that the students or courses be ordered in any particular way.