1. Consider the following interface.

```
interface Rectangle {
    int getWidth();
    int getLength();
    void move(int deltaX, int deltaY);
    void draw(Window w);
}
```

Would it now be reasonable to define the interface `Square` as a subinterface of `Rectangle`?

2. Suppose we added a `stretch` method to the interface `Rectangle`.

```
interface Rectangle {
    int getWidth();
    int getLength();
    void move(int deltaX, int deltaY);
    void draw(Window w);

    /* Multiplies the width of the rectangle by "factorX", 
     * and similarly multiplies the rectangle height by "factorY".
     */
    void stretch(float factorX, float factorY);
}
```

Would it now be reasonable to define `Square` as a subinterface of `Rectangle`?

Is there a principle of Object-Oriented Design that motivates your answer?
3. Consider the following interface for getting mail from a POP server.

```java
interface GetMail {
    /* Attempts to receive mail from the given "host" on "portNum",
    * using the specified "user" and "passwd".
    * Downloads all available email using the POP protocol.
    * Returns a linked list of MailMessages on success, or "null" on failure.
    */
    LinkedList protocolConnect(java.lang.String host, int portNum,
                              java.lang.String user, java.lang.String passwd);
}
interface MailMessage {
    ...
}
```

What limitations does this interface impose on an interactive email client?

4. Can you suggest a better return type for the above method `protocolConnect` from the collection classes (all from the package java.util): (a) List, (b) ArrayList, (c) Vector, (d) Set. Please justify your answer.

If you're not familiar with the Java collections library, the documentation is online at [http://java.sun.com/j2se/1.4.2/docs/api/index.html](http://java.sun.com/j2se/1.4.2/docs/api/index.html), and partly reproduced here.

```java
public class LinkedList
    extends AbstractSequentialList
    implements List, Cloneable, Serializable

Linked list implementation of the List interface. Implements all optional list operations, and permits all elements (including null). In addition to implementing the List interface, the LinkedList class provides uniformly named methods to get, remove and insert an element at the beginning and end of the list. These operations allow linked lists to be used as a stack, queue, or double-ended queue (deque).
```

5. Suppose you were designing an application that stored sensitive data in log files, and you wanted to given the user the option to either:

- (a) store the data as text;
- (b) compress the data;
- (c) encrypt the data;
- (d) compress then encrypt the data; or
- (e) encrypt then compress the data.

What design pattern might you use? Describe briefly how your design would work.

6. In the previous question, suppose you needed to create many such log files in different parts of the application, all of which follow the user’s global choice of storage format. Is there a second design pattern that would be useful here?