Student ID #:_____

Question 1 [16 marks]

You are to design part of a lottery system. The core of the system is a random-number generator represented by the class `Generator`. In this system, objects of a `LotteryGame` class will request random numbers from a `Generator`. Here is a *very incomplete* implementation of `Generator`.

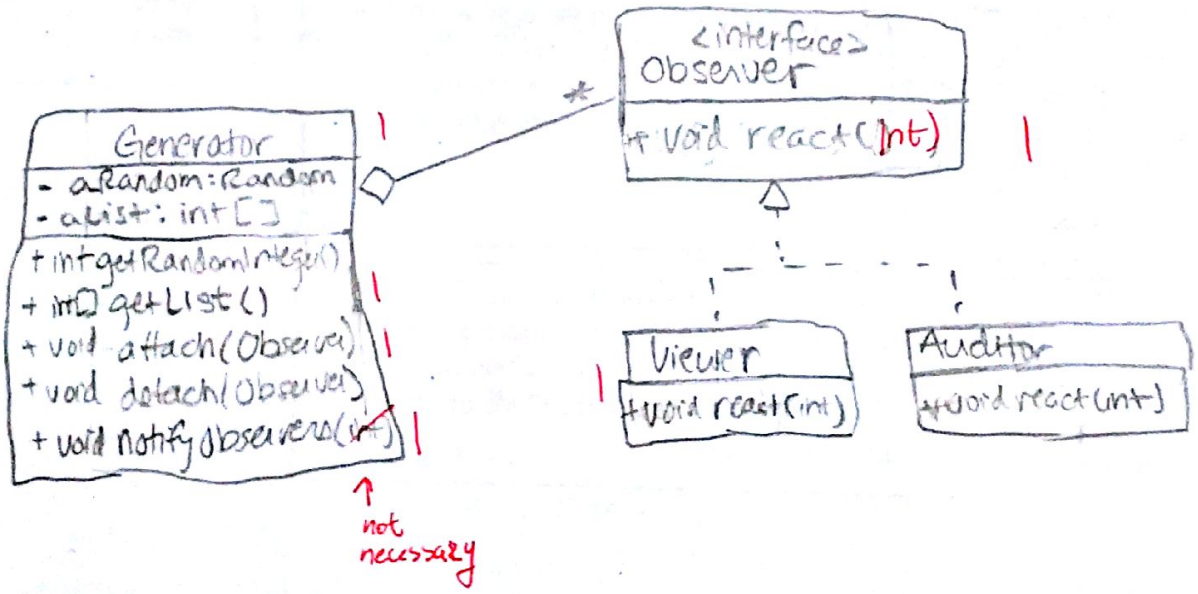
```
public class Generator {
   private Random aRandom = new Random();

   public int getRandomInteger() { return aRandom.nextInt(); } // incomplete
}
```

In addition, a `Generator` should keep the list of all numbers it generates. These recorded numbers should be available to clients of `Generator` through a mechanism of your choosing. Finally, whenever a new number is generated, objects of classes `Viewer` and `Auditor` should react (the `Viewer` should display the number, the `Auditor` should check for various vulnerabilities in the random sequence. The details are not important). The system should use the OBSERVER design pattern to implement this last requirement.

a) Draw a UML class diagram representing the main interactions between all the classes mentioned in the question to support the requirements above, plus any additional class or interface required to complete the design. Include the methods relevant to the requirements. Use meaningful names. [7]

7



2

b) Which data-flow model did you use? Why? [2]

I used the push data flow model. I passed to the callback number the new random integer that was generated. Push will work in this situation because Generator knows what information its observers want to know (the new random integer).

Student ID #:_____

c) Write a small sample of Java code that shows how observers would be attached to the subject in your design. To indicate that a variable is of a certain type, use the convention `lTheType` for an object of type `TheType`. [1]

1

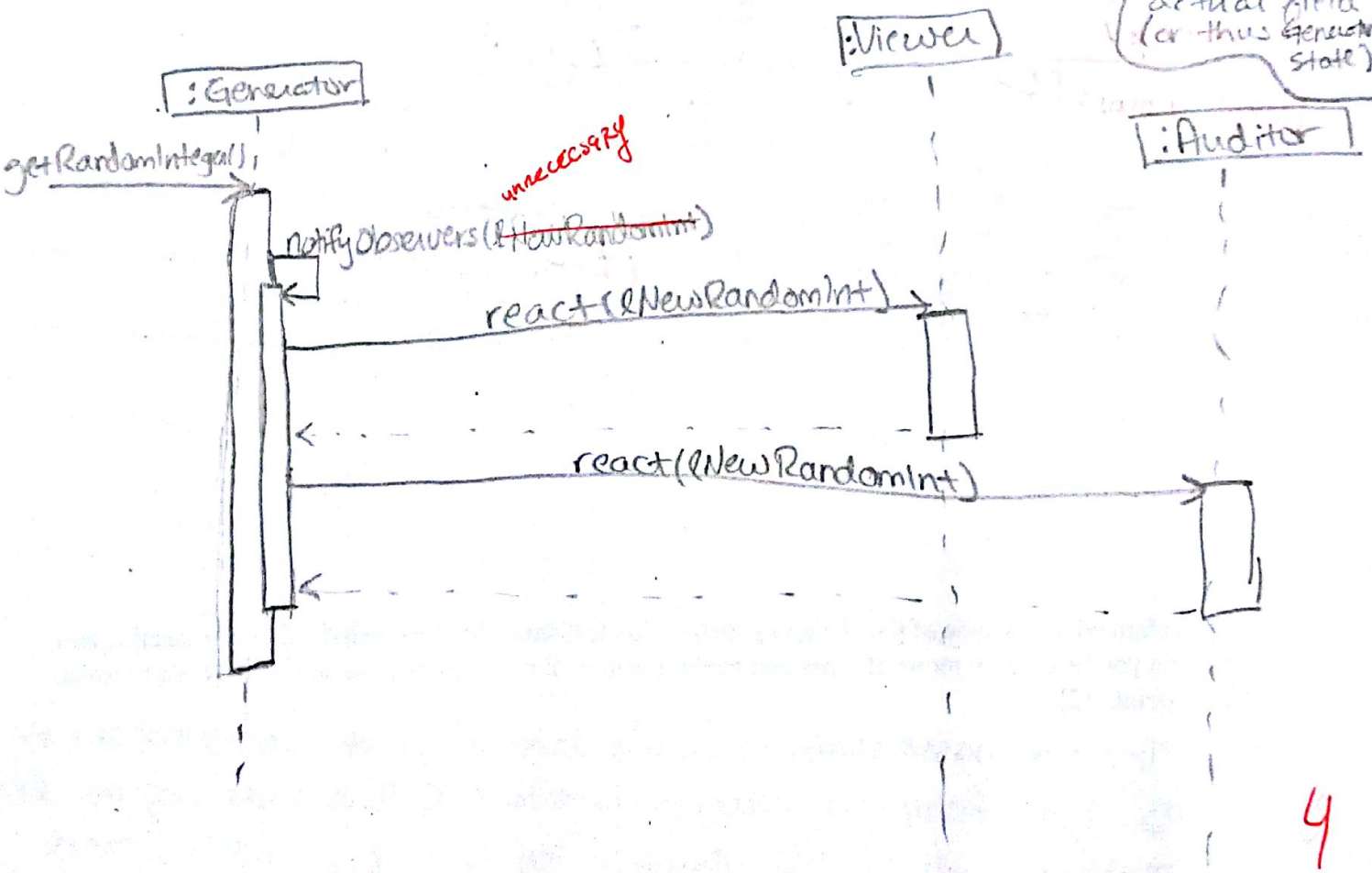
```
lGenerator.attach(lObserver);
```

d) Briefly explain how your design supports obtaining previously-generated numbers recorded by the `Generator`. Comment on the encapsulation? [2]

2

The Generator class has a getList() method which is public so any client can access it. The getList() method returns a clone of the field aList which stores all the previously-generated numbers. This is well encapsulated because it returns a clone so clients can't change Generator's actual field (or thus Generator's state).

d) Draw a UML sequence diagram that illustrates what happens when `getRandomInteger` gets called. Include all interactions related to the OBSERVER pattern. Make sure the names are consistent with the diagram in a). [4]



4

Student ID #: 26044864

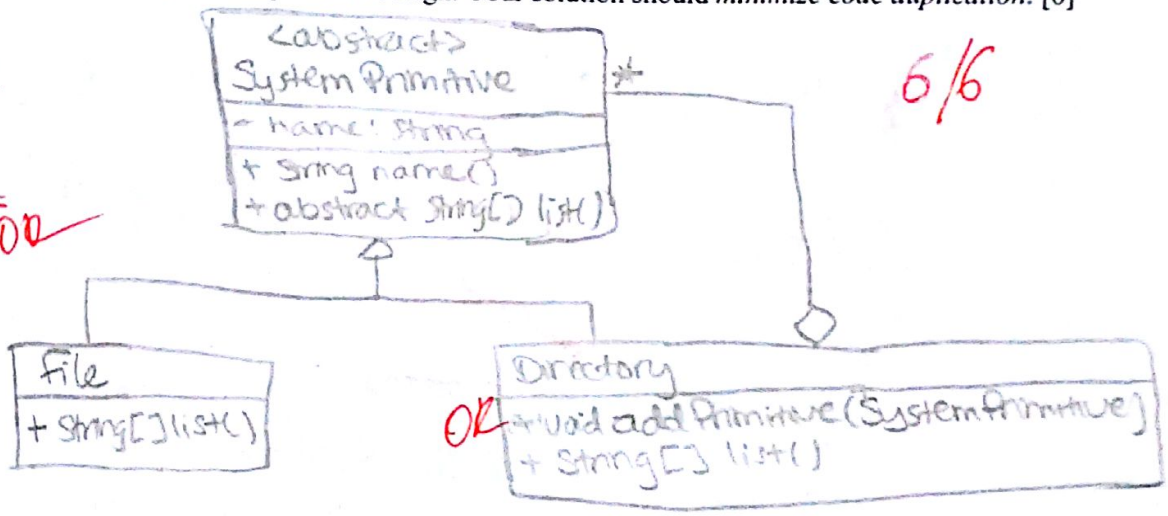
Question 2 [12 marks]

You are implementing a file system, and your design includes a `File` class and a `Directory` class. A `Directory` contains `File` or `Directory` objects in a recursive way. Both `File` and `Directory` classes have methods `list()` and `name()`. The `name()` method returns a `String` (the name of the file or directory), and `list()` returns a `String` array of all the names of the files in a directory and its subdirectories (recursively). Applying `list()` to a `File` just returns the name of the file (seems weird? That what UNIX `ls` does).

a) Draw a UML class diagram that illustrates how to implement this system using the COMPOSITE design pattern. Your diagram should include the `File` and `Directory` types, along with any new type necessary to complete the design. Make sure to include the `list()` and `name()` methods, and all additional methods necessary to complete the design. Your solution should *minimize code duplication.* [6]

Abstract OK
Aggregation OK
superlype
Design almost OK

6/6



b) Explain what the code of the `list()` method in the Composite class will do. Be very specific, and explain your answer in terms of types and method names. You can even write some Java code to make your point. [2]

2

The composite class is called Directory. It contains a list of SystemPrimitives (Files or Directories). The code of its list() method will iterate through its list of primitives and call list() on each of them. If the primitive is a File calling list() will return the name; if the primitive is a Directory calling list will start a recursive call to go through all that directory's SystemPrimitives. Then the code returns the list of names it has obtained.

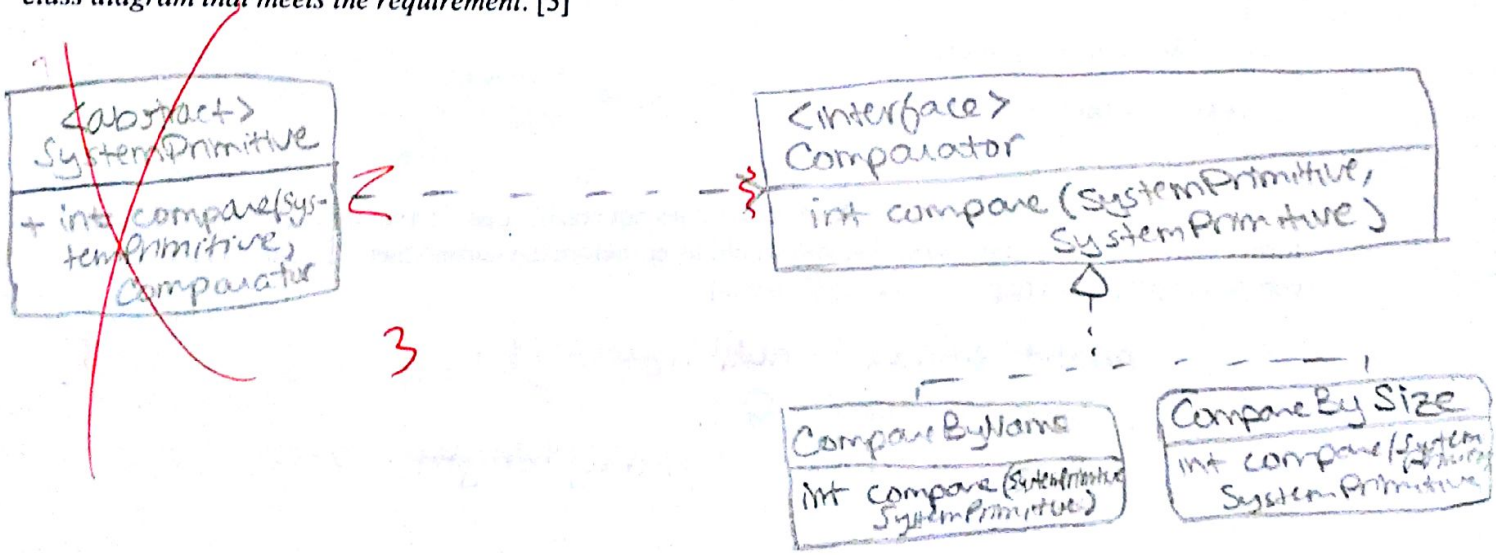
Student ID #: 260442644

You want to make it possible to compare `File` and/or `Directory` objects in different ways. For example, two `Files` or a `File` and a `Directory` are the same if they have the same name. Or, two directories are the same if they have the same list of files (either recursively or not). The number of ways to compare `File` and `Directory` objects with each other is open-ended.

c) What is a good design pattern to support this requirement? [1]

Strategy 1

d) Draw a class diagram to illustrate how you would realize this requirement. Only include the relevant parts from question a). For illustration, include at least two different ways to compare files and/or directories. *Tip: You can do this question even if you don't know the answer to question a). Just draw the class diagram that meets the requirement.* [3]



----- YOU SHOULD NOT NEED THE SPACE BELOW THIS LINE -----

Question 3 [10 marks]

The following `concat3` method returns a `String` that is a concatenation of the 3 `Strings` in `pArray` starting at `pIndex` and ending at `pIndex+3` (exclusive). Note that appending a null-valued variable of type `String` to another `String` results in the `String` "null" being appended. This is acceptable behavior for this program. For example with the array `String[] a = {"A","B",null,"C","D"}, concat3( a, 1 )` returns `BnullC`.

```
class Util {
public static String concat3( String[] pArray, int pIndex )
{
   String lReturn = "";
   for( int i = pIndex; i < pIndex + 3; i++ )
   {
      lReturn += pArray[i];
   }
   return lReturn;
}
```

a) What preconditions would ensure this method does not crash? List all appropriate preconditions in the form of Java `assert` statements that you would insert before the normal method code. Do *not* use compound predicates (e.g., `A && B`) [3 marks].

assert pArray != null
assert pIndex >= 0
assert pIndex + 3 <= pArray.length

3

b) Let's say we are not satisfied about the fact that the method supports null strings in the input array. Would it make sense to *override* the method to create an implementation that works in a similar way, but does not allow null strings? State yes or no, and provide and explain the major argument for or against your choice [3]

NO, because this would break the Liskov Substitutability Principle. Creating a subclass that limits what clients of the superclass can do is against this rule, and making the concat3 method have stricter preconditions would limit what the clients can do.

3

Student ID #: 260448644

Question 4: Very short answers [2 marks each, total of 12]

Answer each sub-question using a maximum of **two sentences**. Make sure your answer is **clear** and **precise**.

(a) Briefly state one argument against the use of overloading.

2 It makes code more complicated and can make it difficult to know which method will be called (for example, if you pass a null parameter to a method overloaded to take two different objects).

(b) What is the purpose of the ITERATOR design pattern?

2 It allows clients of a class to iterate through a list of that class without knowing the details of how the class has implemented its list.

(c) If all the fields of a class are `final`, are the instances of the class immutable? Justify briefly.

2 No, because if you have a final field of a type that's mutable you can change what is at the memory location that field is pointing to (though you can't change where it points).

(d) What does the CVS `commit` operation accomplish?

2 It updates the files in the repository that you have changed so that they match your changes and it alerts you if there's a conflict. Afterwards, others using that repository can access your changes.

(e) What does `super.aMethodName` do?

2 It calls the aMethodName method as it is implemented in the super class (as opposed to how it's implemented in the subclass which is writing "super.aMethodName").

(f) Name two practical benefits of polymorphism (maximum two words each).

2 - flexability
- code reuse