Practice Questions

- Design, UML

(a) [2] Suppose there is a class `Item` and two corresponding container classes `ItemList` and `ItemArray`. In a UML class diagram, is it possible to have a composition relationship from both `ItemList` to `Item` and `ItemArray` to `Item` at the same time? Explain clearly why or why not.

(b) [2] Explain the similarities and differences between *abstract classes* and *interfaces* in Java. In what situation(s) would you use abstract classes and in what situations would you use interfaces?

Main difference - Java interface are implicitly abstract and cannot have implementations.

Abstract class can have instance methods that implements a default behavior.

(c) [2] In UML use case diagrams, depict and explain clearly the main difference between extend and include relationships. Give examples of your choice (but not one used in class).

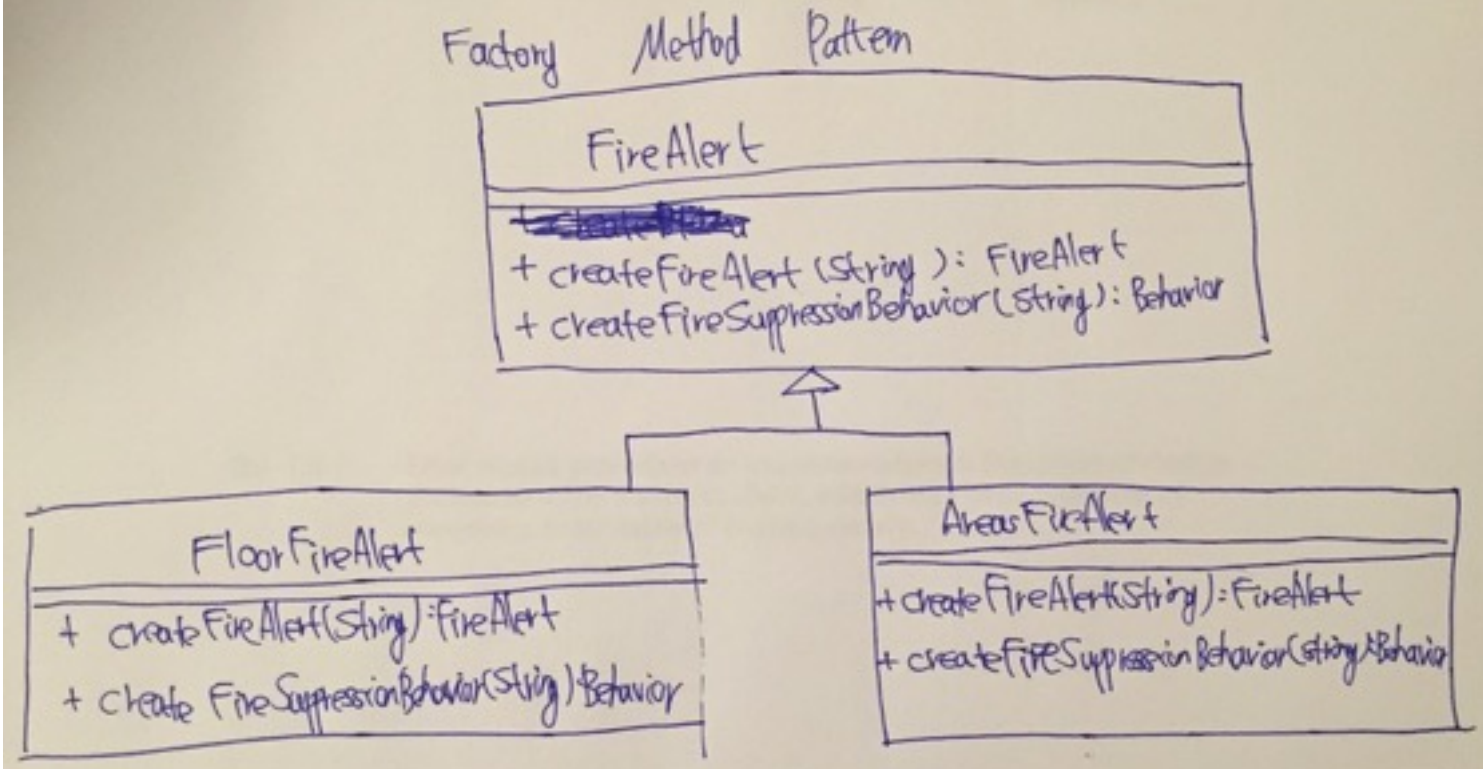
Extend is used when a use case adds steps to another first class use case.

Include is used to extract use case fragments that are duplicated in multiple use case.

- [4] Design patterns

You are modeling an integrated fire protection system for a building with floors, each floor with areas. Each kind of unit (floor or area) has an appropriate fire alert and fire suppression behavior to be triggered consistently. Describe clearly how to represent this configuration using a suitable design pattern.

Apply the pattern, and outline the structure of the design using a correct UML class diagram.



- Design patterns

The observer design pattern is used to define a dependency between objects so that when one subject object changes state, all its dependent observer objects are notified and updated automatically.

(a) [2] How would you modify an implementation of this pattern to handle the case where an observer object may need to be notified of changes in many subject objects (not just one)?

Give a parameter to Notify and Update.

(b) [2] How would you adjust an implementation of the observer design pattern to allow a subject object, which may have a number of observers, to be deleted? Explain clearly.

- Design patterns

In the following situations, explain which design pattern is most appropriate for addressing the problem.

(a) [2] You want to develop an application to count the total size of a file-system directory. Directory sizes are the sum of the sizes of their contents.

Composite Pattern

(b) [2] You are developing a spreadsheet application that allows cells to be calculated automatically based on formulas depending on other cells.

Factory Method.

(c) [2] You want to develop a kids' calculator for integers and you have a college math calculator.

Decorator Pattern.

(d) [2] You want to develop a file reader that is capable of reading a file, which can possibly be (1) zipped, (2) encrypted (3) zipped and encrypted or (4) encrypted then zipped and encrypted again.

Command pattern

- Refactoring

Suppose there is an `Employee` class and a `PayType` class. Different employees may be paid differently. For example, a salesperson may get a commission beyond their usual monthly salary. A manager may get a management bonus.

Assume an `Employee` object has a `PayType` object that is responsible for such types of pay. Consider the following (partial) implementation of a `PayType` class.

```
class PayType {
    …
    int payAmount( Employee emp ) {
        switch (this.getTypeCode()) {
        case ENGINEER:
            return emp.getMonthlySalary();
        case SALESPERSON:
            return emp.getMonthlySalary()
                + emp.getCommission();
        case MANAGER:
            return emp.getMonthlySalary()
                + emp.getBonus();
        …
        }
    }
}
```

(a) [3] Outline how to refactor this code to use polymorphism instead of the above switch statement.

Create subclass for every condition.

(b) [3] Draw the UML class diagram after the refactoring. State any further assumptions.

- [4] Refactoring

Consider the following (partial) implementation of a Person class.

```
class Person {
    …
    public String getName() {
        return _name;
    }
    public String getTelephoneNumber() {
        return "(" + _officeAreaCode + ") "
            + _officeNumber;
    }
    String getOfficeAreaCode() {
        return _officeAreaCode;
    }
    void setOfficeAreaCode( String areaCode ) {
        _officeAreaCode = areaCode;
    }
    String getOfficeNumber() {
        return _officeNumber;
    }
    void setOfficeNumber( String officeNumber ) {
        _officeNumber = officeNumber;
    }
    private String _name;
    private String _officeAreaCode;
    private String _officeNumber;
}
```

should be object

should be object

Draw the UML class diagram for the refactored design.

• [6] User interface design

The Unix `diff` command displays line-by-line differences between pairs of text files.

The command has the form:
`diff` *options file1 file2*

where the *options* can be a sequence of zero or more of the following four flags:
`-b` ignore trailing spaces and tabs
`-i` ignore case
`-t` expands tab characters in the output
`-w` ignores all spaces and tabs
(followed by zero or one other flags which will not be of concern in this design).

Example files and output:

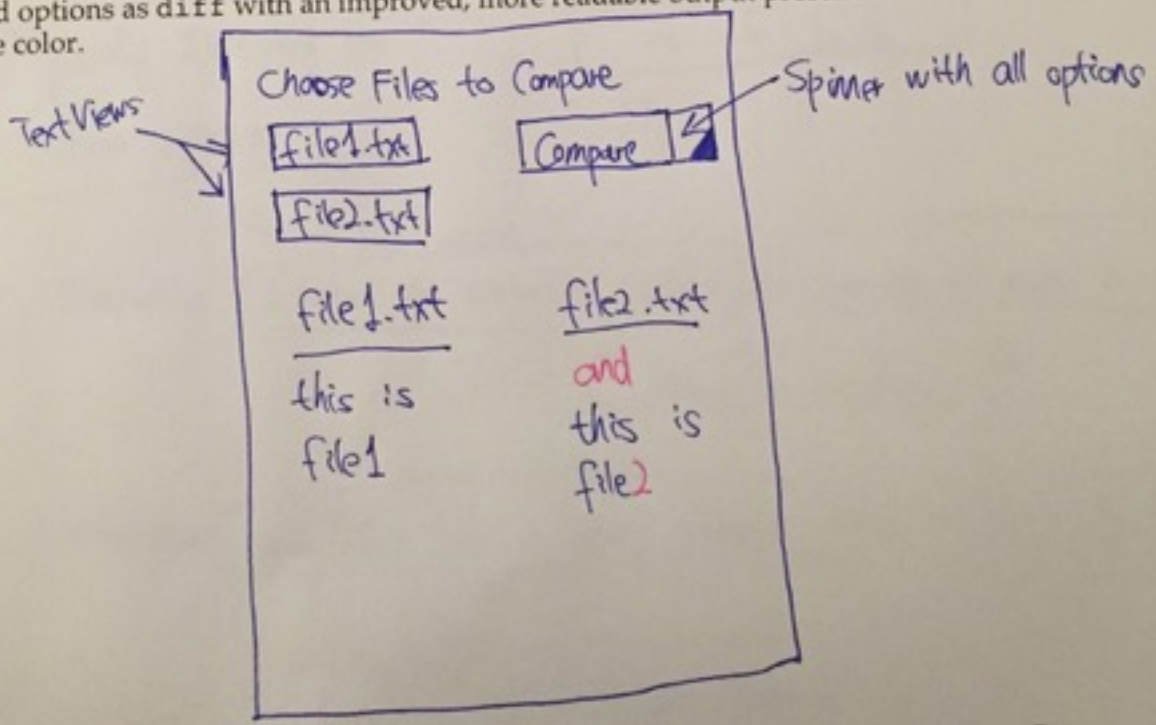
```
file1.txt:
this is
file1
```

```
file2.txt:
and
this is
file2
```

```
diff file1.txtfile2.txt
0a1
> and
2c3
< file1
---
> file2
```

That is, the second file has an added line at the top and a changed line. In general, differences can be additions, changes, or deletions.

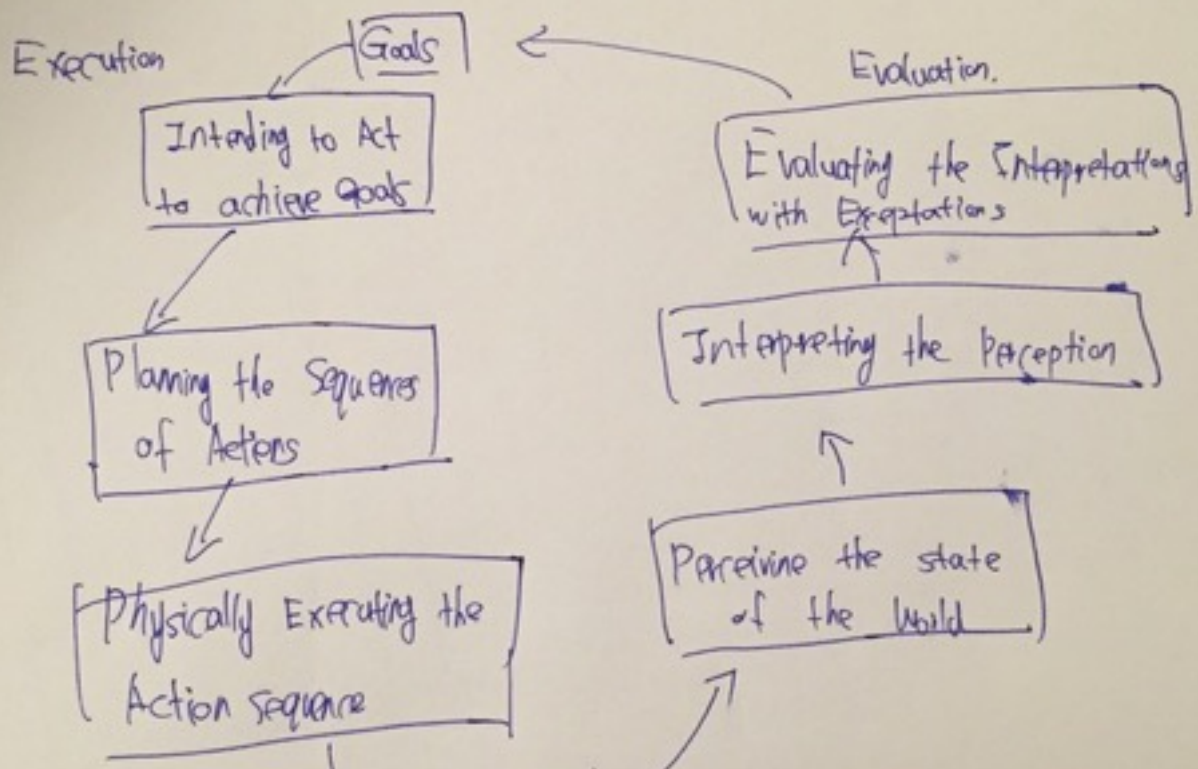
Sketch the layout of a graphical user interface with the same essential capabilities and options as `diff` with an improved, more readable output presentation. Hint: use color.



• [4] Human error

Norman devised a model of human-computer interaction with an execution-evaluation cycle. In demonstrating why some user interfaces caused problems to their users, he referred to two concepts: gulfs of execution and gulfs of evaluation.

Explain the concepts of *gulfs of execution* and *gulfs of evaluation* using user interface examples of your choice (but not used in class).



Example: Execution I want to make a sandwich → Planning of make a sandwich → Make a sandwich

Evaluation Search what does a normal sandwich online → Interpreting the normal sandwich → Compare the different between the normal sandwich with my sandwich