İTÜ
Computer Engineering Department
Assoc.Prof. Feza BUZLUCA

Object Oriented Modeling and Design Exemplary Final Questions and Solutions

QUESTION 1:

Parts (a, b, and c) can be solved independently.

a) Write the **two** main benefits of the domain (analysis) model.
Solution (Write two sentences)**:**

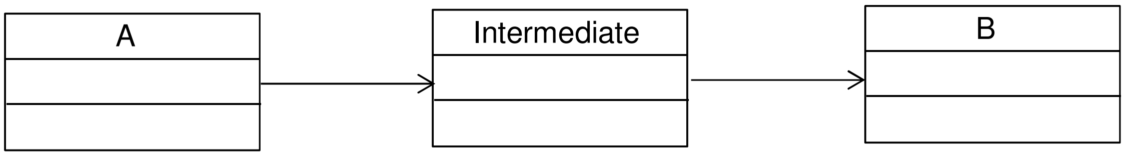
1. It helps to understand the system (conceptual classes in the real world).

2. It acts as a source to define software classes in design level.

b) What is the benefit of the **operational contracts** in design phase?
Solution (Write one sentence)**:**

Postconditions of the contracts give us the responsibilities (in the form of object creation/deleting, association forming/breaking, attribute modification) that we must assign to the objects in design.

c) Draw a UML class diagram to show the "**Indirection**" pattern and explain shortly its **benefits**.
Solution:



- A is not directly coupled to B. Changes in B will not affect A.
- In the future we can replace B with another class C. This replacement will not affect A.
- We can reuse the class in a new project independently from B.

Note: Although the Adapter pattern (GoF) covers the Indirection pattern, it also includes other patterns and mechanisms such as inheritance and polymorphism. For indirection we don't need them.

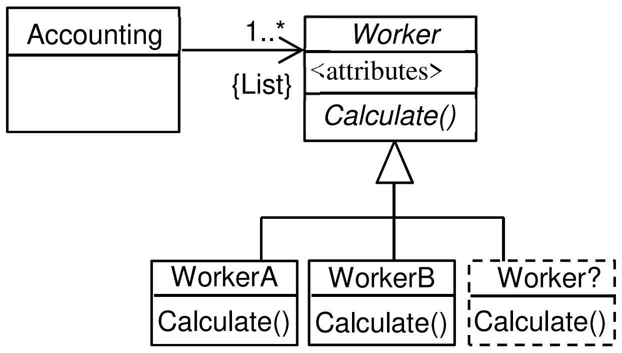
QUESTION 2:

In a company, salaries of some workers are calculated using the method A, and salaries of other workers are calculated using the method B. These methods depend on some attributes of the works such as age, number of children etc.

The accounting system maintains a list of workers. Each month salaries of workers in this list are calculated.

In the future, these methods can change, a new method can be added to the system, and the method of an existing worker can be replaced by another method.

The software architect decides to use inheritance and constucts the given design.



a. What are the possible drawbacks of the given design?

b. To overcome these problems construct a better design and draw the UML class diagram. Show parameters of the methods. Mention the design principles and patterns used in your solution.

c. Explain what will hapen in your system if the calculation method of an existing worker is replaced by another existing calculation method?

SOLUTION:

a.

Actually, there are not different types of workers in the system; only the method to calculate the salary changes.

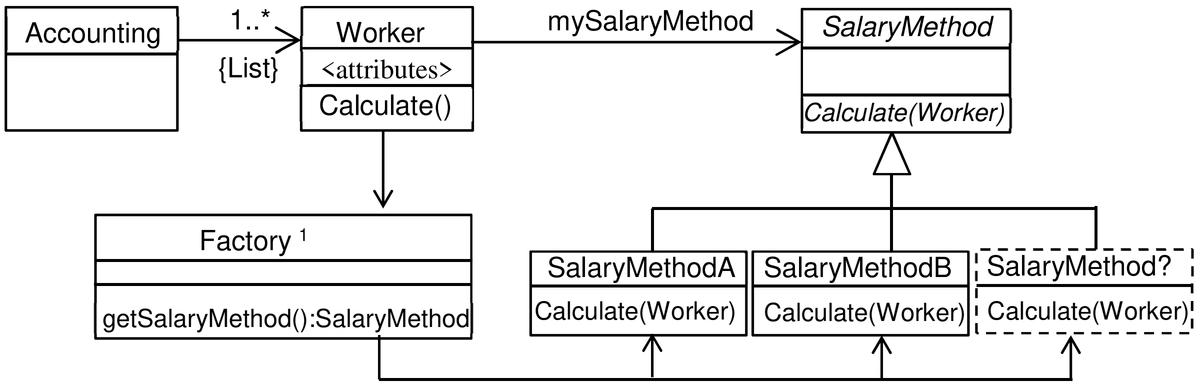
Varying parts (calculation methods) are inserted into stable class (worker).

Drawbacks:

- If we want to inset a new calculation method a new type (class) of worker must be written.
- Calculation method of a worker cannot be changed in run time. In such a case, we must destroy (delete) the existing worker object and then create a new one.

b.

To have a proper solution we will apply the Strategy pattern, which supports the "Favor composition over inheritance" principle.



Principles:

"Favor composition over inheritance" principle.

"Design to interface"

Patterns:

Strategy, Factory (singleton)

c. If we want to replace the calculation method of an existing worker by another existing calculation method, we only need to change the address (reference) of the SlaryMethod pointed

by the pointer mySalaryMethod of the Worker object. We can change it in run time, without interrupting the system.
Details:
Worker object asks the Factory for a SalaryMethod pointed by calling the getSalaryMethod(). The Factory will decide the new method and return its address.

QUESTION 3 :
To do: An object of class A must be created and an object of class B will use it.
According to GRASP Creator, "Assign class B the responsibility to create an instance of class A if B closely uses A".
In what cases this advice does not lead to proper solutions?
Create a better design to overcome this problem and express your solution by drawing a UML collaboration diagram.

SOLUTION:
This advice can violate the "low coupling" principle. If B creates objects A then B is coupled to A. Class B must also know the details how to create objects of A (when, which parameters?).
It may also lower the cohesion of B. Class B may have different responsibilities from object creation.
Class A can be a subclass of a general type and according to some conditions objects of different subclasses can be creates such as A1, A2. Class B must include the decision logic. If subclasses or the decision rule change class B must also change.
It is better to apply the "Factory" pattern.

QUESTION 4:
In a software system there is a counter object (:Counter) that counts some events. There are also other objects which perform some actions if the counter value exceeds two predetermined threshold values. For example, objects :A and :B perform their actions if the counter exceeds the threshold1 value; objects :C and :D perform their operations if the counter exceeds the threshold2 value. In run-time, the number of objects that are interested in the counter value can change. While objects (:A , :B , :C , :D) are created they get the address of the :Counter object in which they are interested. Classes (A,B,C,D) include two methods, namely startCount() and stopCount() to start and terminate their interest in the counter value, respectively.
In the future new classes (E, F) can be added to system that may be interested in the counter value.

a) Design the system to achieve the required flexibility and draw the UML class diagram. Which GoF design patterns did you use in your design?
b) Write the known parts of the program of class A in C++ or Java. Include necessary attributes, constructor, and methods startCount(), stopCount(), Operation() that is invoked if the counter value exceeds the threshold.
c) Assume that the class Conter includes a method eventHappened() that increments the counter value. Draw a UML sequential interaction diagram, that show interactions, which occur in the system if when the eventHappened() method is called.

SOLUTION:
Observer pattern will be used.
Counter is publisher (:A , :B , :C , :D) are subscribers (listeners or observers).
One solution is to create two listeners-lists, namely one for threshold1 and the other one for threshold2.
Ore only one list can be created and the notified objects can check the reason for the notification (threshold1 or threshold2).

QUESTION 5:

We are designing a class **A** that has a method calculateX(), which calculates the value of a variable x by calling different functions depending on some conditions as shown on the right.

```
void A::CalculateX()
{
...
     if (condition1)  x= f1(attribute1);
     else
     if (condition2)  x= f2(attribute1, attribute2);
     else
     if (condition3)  x= f1(attribute1) + f3();
     else
     if (condition4)  x= f2(attribute1, attribute2) + f3();
   ......
```

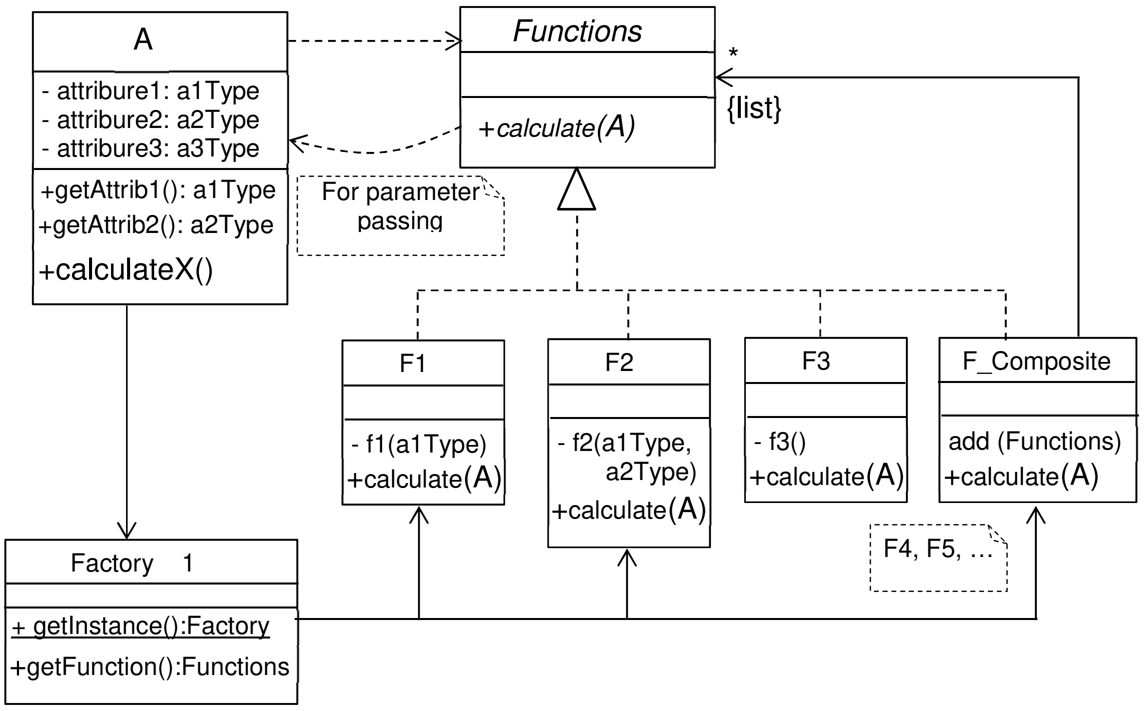
- Sometimes, it calls a single function such as f1(...), f2(...) or f3().
- For the same calculation, sometimes it calls a group of functions and the result is the sum of all return values such as (x= f1(...) + f3();).
- In the future, another group of functions can be called to calculate the value of x, such as (x= f2(...) + f3();).
- Functions take different attributes (data members) of the class A as parameters.
- In the future, new functions (f4(...), f5(...)) can be added to the system that are called under different conditions. Their parameters can be different attributes of the class A.

a) Design the system using GoF software design patterns to achieve the required flexibility so that the possible changes will not affect the class A. Draw the UML class diagram. Mention the GoF design patterns that are used in your solution.

b) Assume that the method calculateX() is called and condition3 is true. Hence, to calculate the value of x, the operation f1(attribute1) + f3(); must be performed. List step by step the operations (function calls) that are performed in the system at run-time for the calculation of the value of x.

Solution:

a) Class A sometimes gets a service form a single function (Atomic) and sometimes from a group of functions (Composite). GoF Patterns: Composite, Factory with Singleton



Note: The factory can create an object of F_Composite and can add atomic functions into its list. This solution is flexible, because the factory can create composite objects that have different functions in their lists. It is also possible to change the list in run time.

QUESTION 5:
a) (cont'd)

- Designing a separate strategy class for each condition is not a flexible solution. If we have new conditions or if we want to change them, we must rewrite the program. However, if we apply the Composite pattern, the factory needs only to change the functions in the list.
- Decorator pattern does not provide the proper design either. We don't need to call functions in a predetermined order. We need to call a group of functions and then add their return values.

b)

- Object of class A (:A) (in the calculateX method) calls the getInstance method of the Factory.
- :A calls getFunction() of the Factory.
- Because condition3 is true, getFunction() of the Factory decides to create an object of F_Composite (:F_Composite).
- getFunction() decides to create objects of F1 and F3 (:F1, :F3), if they were not previously created.
- getFunction adds objects :F1 and :F3 to the list of the :F_Composite by calling its add(Functions) method two times.
- getFunction returns the address of :F_Composite to :A.
- :A calls calculate(A) method of :F_Composite and sends its address: calculate(this).
- calculate(A) method of :F_Composite calls calculate(A) methods of the atomic objects in its list.
    - First it calls calculate(A) method of :F1.
    - calculate(A) method of :F calls getAttrib1() method of :A and gets the value of the attribute1. It calls f1(attribute1). f1 returns X.
    - Then it calls calculate(A) method of :F3. It calls f3(). It returns X.
- calculate(A) method of :F_Composite calculates and returns the sum.

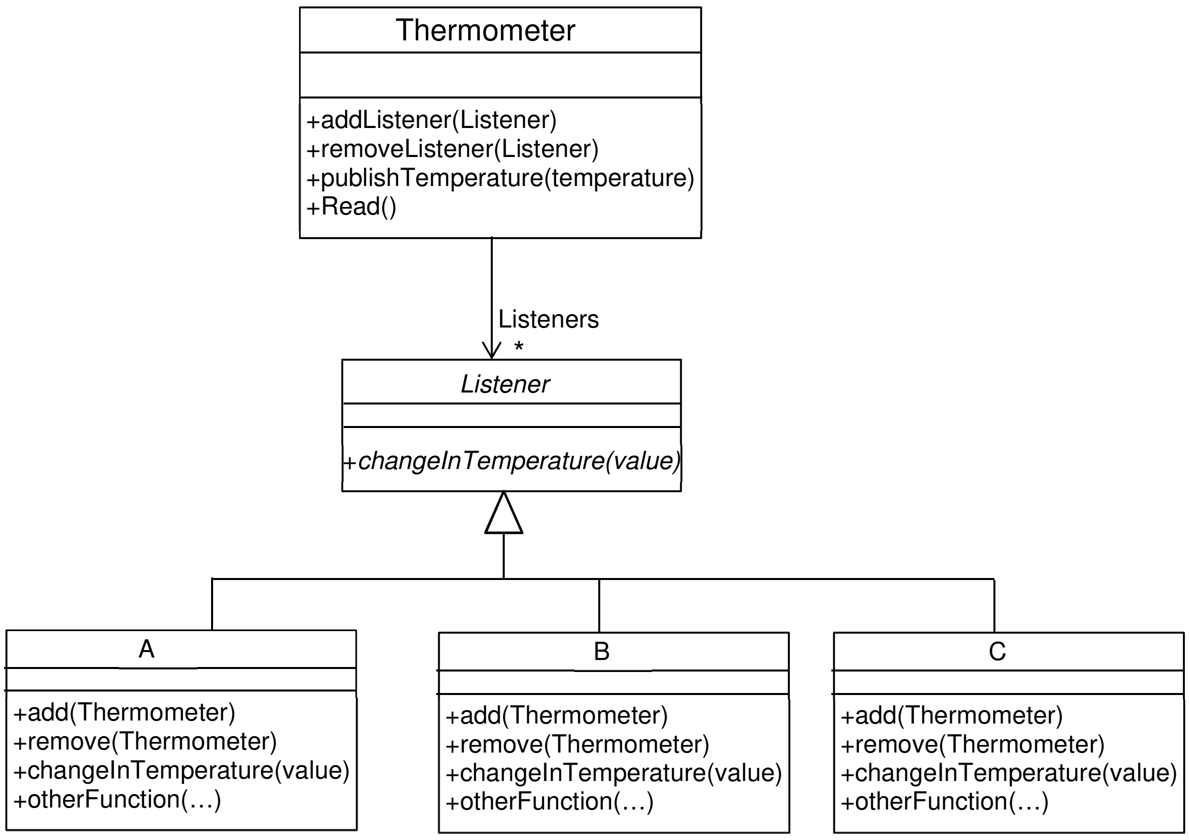
QUESTION 6 :
In a software system there is a class Thermometer with a method Read(), which reads temperature values from a heat sensor.
In this system, there are also other classes (A, B , C, etc.). Their objects perform different operations based on the changes in the temperature. For example, object of A turns on or off a heater; object of B sends messages to some users. In the future, other classes (D, E , etc.). may also need the temperature value for their jobs.
Objects of classes (A, B, C, etc.) need the temperature value only in certain times. For example, object of A needs the temperature value only on Mondays and Tuesdays. Object of B needs it every day between 10.00 and 14.00. These times may change.

d) Design the system to achieve the required flexibility and draw the UML class diagram. Which GoF design pattern did you use in your design?

e) Explain what happens (step by step) in your design in runtime, when an object of the class A needs the temperature value and how it gets this value.

Solution:

a) Thermometer will be designed as a publisher. When objects of classes (A, B, C, etc.) need the temperature value they will call addListener method of the Thermometer and will subscribe to its list. When they do need the temperature, they will call removeListener method to end the subscription. GoF pattern: Observer (publish-subscribe, listener)



b)

- When it is Monday or Tuesday, an initializer object of class A calls the add method of class A and gives the address of the Thermometer object (:Thermometer).
- :A calls (in the add method) addListener(Listener) of the Thermometer and gives its address (this).
- :Thermometer adds :A to its list Listeners.
- :Thermometer reads (Read()) the temperature value; if there is a change it calls its publishTemperature(temperature) method.
    - The publishTemperature(temperature) method iterates on the list Listeners and calls changeInTemperature(value) method of all listeners in the list. Because :A is in the list, it also calls its changeInTemperature(value) method and gives the temperature value as a parameter.
- :A gets the temperature value dose its predetermined job.
- On Wednesday, the initializer object of class A calls the remove method of class A. Hence :A does not get unnecessary temperature values.