Home Work 1

Shanbo Li	840810-A478	shanboli@Gmail.com	0704646157
Sike Huang	850414-A394	sike.huang@gmail.com	0762320173

Question 1: What is the difference between an abstract class and an interface?

Answer:

The difference between an abstract class and an interface is shown on the *table 1*

Previously define:

```
abstract class A {
int x, y;
static x1 = 100;
    void a(int newX, int newY) {
        ...
    }
    abstract int b();
    abstract void c();
}

abstract class B{
    Abstract void d()
}
```

```
public interface I {
    final static z = 1124;
    int e();
    String f();
}

public interface J{
…
}
```

Feature	Abstract Class	Interface

Multiple Inheritance	A class may extend only one abstract class.
	
	e.g.
	Follows are wrong:
	class C extends A,B{
	…
	}
		A class may implement several interfaces.
		
		e.g.
		Follows are ok:
		class C implements I,J{
		…
		}

Default Implementation	An abstract class can provide complete code, default code, and/or just stubs have to be overridden.
	
	e.g.
	see definition of abstract class A
		An interface cannot provide any code at all, much less default code.
		
		e.g.
		see definition of interface I
		if we write follows in the definition of interfaces I, it is wrong:
		void foo(int j, int j) {
		...do something…
		}

constants	Both instance and static constants are possible. e.g. see definition of abstract class A	static final constants only e.g. see definition of interface I if we write follows in interface I, it will be wrong: int x;
default field	friendly default field can be redefined as well as given a new value e.g. Give a new value to field x in A is ok.	public static final static cannot be changed e.g. as defined z in interface I cannot be changed
method	Can have predefined method, all abstract method must be implemented. e.g. see definition of abstract class A	Cannot have predefined method, all method must be implemented. e.g. see definition of interface I
adding functionality	If you add a new method to an abstract class, you have the option of providing a default implementation of it. Then all existing code will continue to work without change. e.g. like method a() in abstract class A.	If you add a new method to an interface, you must track down all implementations of that interface in the universe and provide them with a concrete implementation of that method. e.g. implement all methods is necessary.
speed	Fast	Slow, requires extra indirection to find the corresponding method in the actual class.

Table 1 difference between an abstract class and an interface

Question 2:

i). Give an account of advantages and disadvantages of inheritance and composition.

Answer:

Composition

Advantages:

- Contained objects are accessed by the containing class solely through their interfaces
- “Black-box” reuse, since internal details of contained objects are not visible
- Good encapsulation
- Fewer implementation dependencies

- Each class is found on just one task
- The composition can be defined dynamically at run-time through objects acquiring references to other objects of the same type

Disadvantages:

- Resulting systems tend to have more objects
- Interfaces must be carefully defined in order to use many different objects as composition blocks

Inheritance

Advantages:

- New implementation is easy, since most of it is inherited
- Easy to modify or extend the implementation being reused

Disadvantages:

- Breaks encapsulation, since it exposes a subclass to implementation details of its superclass
- "White-box" reuse, since internal details of superclass are often visible to subclass
- Implementations inherited from superclass cannot be changed at run-time

ii) Give an example of when inheritance is better and when composition is better

Answer:

Composition is used in such way that you want the features of a ready-made class inside your class, which means you embedded an object and use it to implement functionally in your new class. By making embed objects private, the user of your new class sees the interface you've defined, instead of the interface from embedded object.

On the other hand, inheritance utilizes an existing class and makes a particular version of it. You take a general-purpose class and specialize it for certain need.

In all, inheritance stands for the "is a" relationship, and composition presents the "has a" counterpart.

Question 3:

i). What are some of their advantages and disadvantages over each other?

Answer:

The advantages and disadvantages over each other is shown on the *table 2*

	winner	Description
Easy to understand and use	waterfall	Waterfall model is the easiest way to understand and use. The developer can easy to know what stage he is working now. And what he will do next. On the other hand, spiral model is much more complicate
Face the wrong of design or documents	spiral	To spiral model there is particular stage to face the wrong design and can easily correct the wrong design. To waterfall model it is hard to change the design or documents, for the waterfall model is a step-by-step model and never go back.
Face the objectives alternatives	spiral	Spiral model is an activity-centered life cycle model that was devised to address the source of weaknesses in the waterfall model, in particular, to accommodate infrequent change during the software development. As described above, objectives must be declared clearly, and cannot be changed.
Risk management	spiral	Risk management is an activity of spiral mode. The more work the finished, the less risk they faced. To waterfall model, it just goes the opposite way. The designer cannot calculate all the risk and as the process the development new risk may come out. But waterfall model do not have a stage to salve the new risk.
Active with customers	spiral	“AIM it, DO it”, waterfall model do not have any activities between the customers and the developers. To spiral model, it increased the communication between customers and developers, the customers can adjust their needs during software development. And it can make the software fit their demand more exactly.
Small work	waterfall	Easy to do, easy to management.
Huge work	Spiral	Less risk, more consummate.

Table 2 advantages and disadvantages over Waterfall model and Spiral model

ii). Describe development circumstances under which you would prefer to follow waterfall method of development, instead of spiral model?

Answer:

If I meet a development circumstances that has most of the follow feature, I will use waterfall model instead of spiral model.

- Small Application with limited functions
- A specifically goal and the goal will not changed.

- Clearly demands.
- A roadmap with clearly timestamp.
- We are sure to design the documents perfectly and plan perfectly.
- The customer wants us to use waterfall model so he can track what we are doing now.

e.g.

A customer found us and need a calculator application. Which only need to calculator +, -, *, / and can run on both windows and MacX. He wants us do it as soon as possible.

Waterfall model is the best selection.

Question 4: Draw a class diagram for the case of our course. We assume that there are at least the following classes: Person, Student, Instructor and Assistant (you can add as many additional classes as you would like). You should draw associations between them, possible attributes and operations.

Answer:

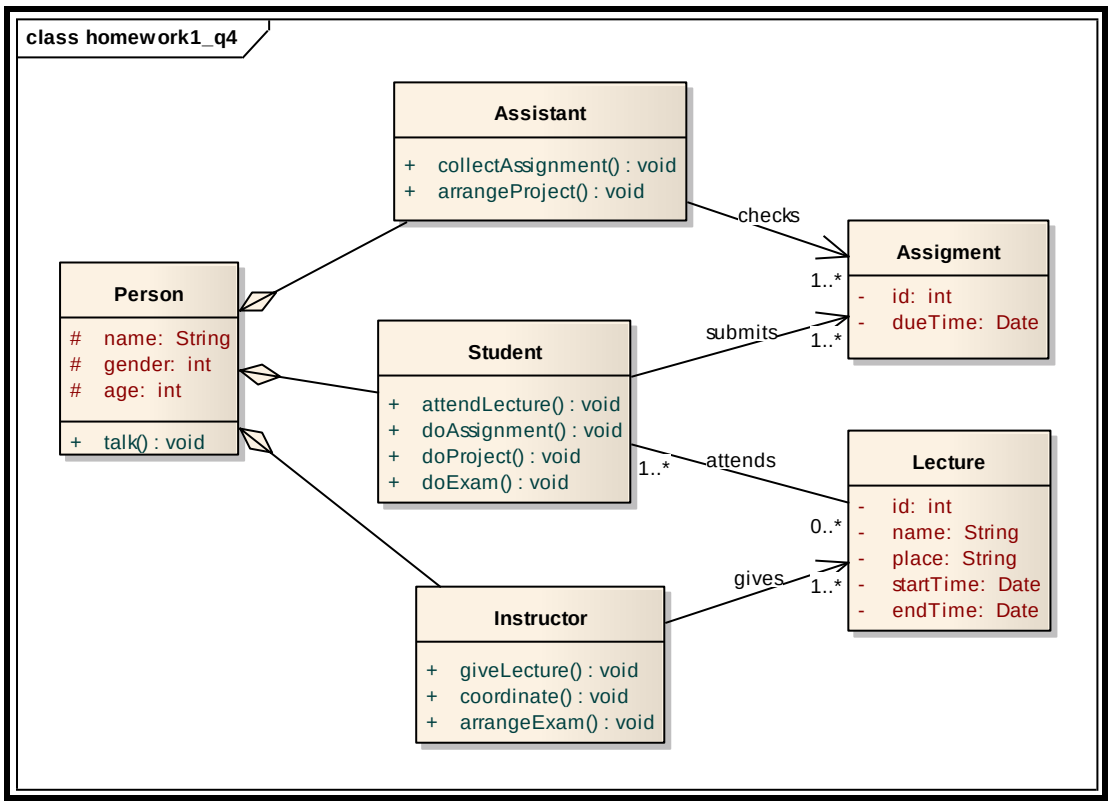

Figure 1 class diagram of our course

The class diagram shows on the *Figure 1*. There are two reasons, we flavor composition over inheritance. First of all, a student is a role a person plays (rather than “is a special kind of “), so are instructor and assistant. Secondly, an instance of a subclass of Person could change from Student to Instructor or assistant over time, and vice versa.

Question 5:

Answer:

As shown on the class diagram *Figure 2*, class InsurancePolicy has two association classes. DateInfo and PolicyInfo. All the fields and method named with their functions. We separate date from other info of policy, for there are some special operation to date. No currentYear field needed, for it could be calculated according to renewalDate by the method getCurrentYear(). The customer pays the Insurance by method payForThisYear(money:double):boolean. If paid successful, UpdateRenewDate() will be called to update the renewdate i.e. renewalDate plus one year.

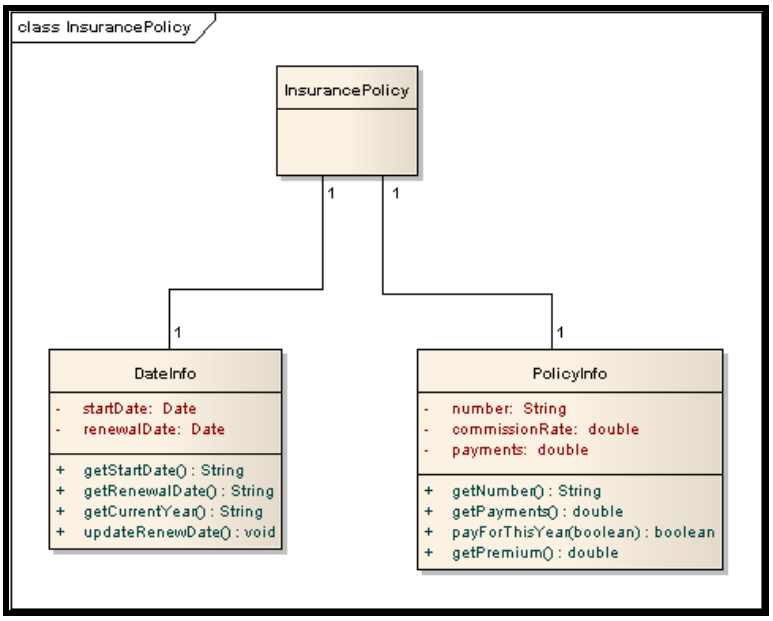

Figure 2 class diagram for InsurancePolicy

Question 6:

Answer:

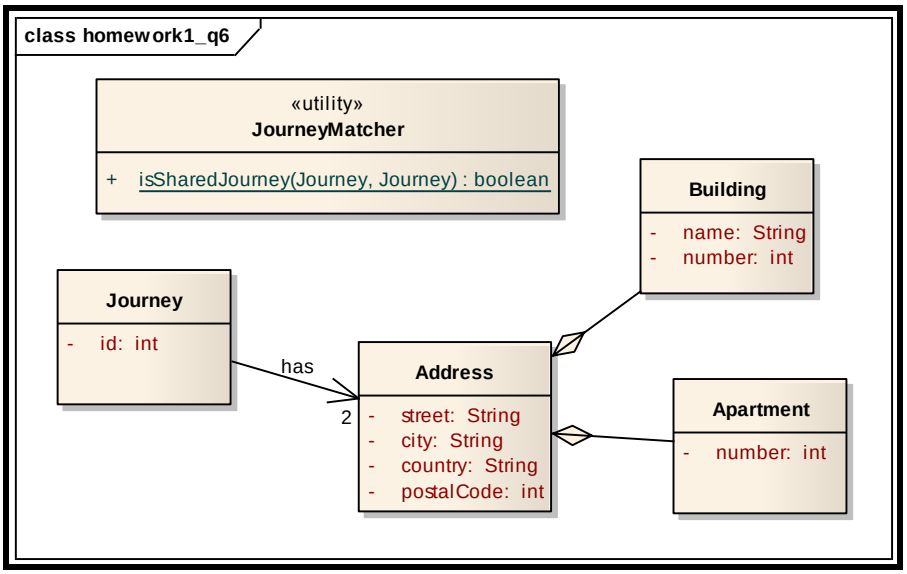

Figure 3 a part of Car Sharing System class diagram

The class diagram shows on *Figure 3*. Each journey is composed of two addresses: start and destination, while each address owns a specific building and apartment to further identify its uniqueness, and a utility class is introduced to accomplish the task of matching shared journey.

Question 7: Draw the Publish/Subscribe protocol as sequence diagrams (publisher, subscriber and Subscription consumer). "Subscriber subscribes consumer at a publisher to receive notification message and publisher sends confirmation to subscriber and notification to consumer (who may be the same or different)."

Answer:

The sequences diagram *Figure 4* shows the Publish/Subscribe protocol.

According to the subscribe(receiver:Customer). The Publisher will choose the target to send notification (Subscriber or Consumer). The argument receiver is just a flag to inform who should receive the notification.

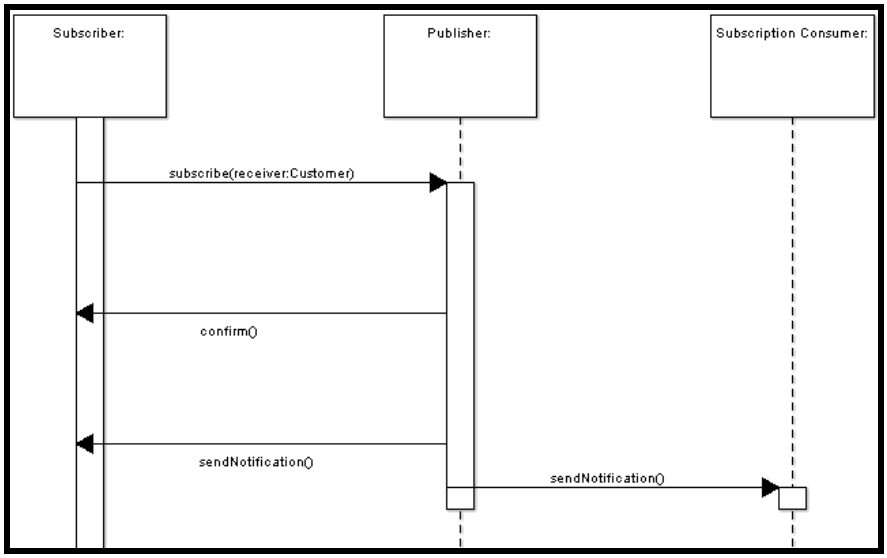

Figure 4 sequence diagram of Publish/Subscribe protocol

Question 8: Draw a sequence diagram of scenario "WarehouseOnFire" given in figure 2-15 in Book. Include Objects bob, Alice, john, FRIEND and instances of other classes you may need.

Answer:

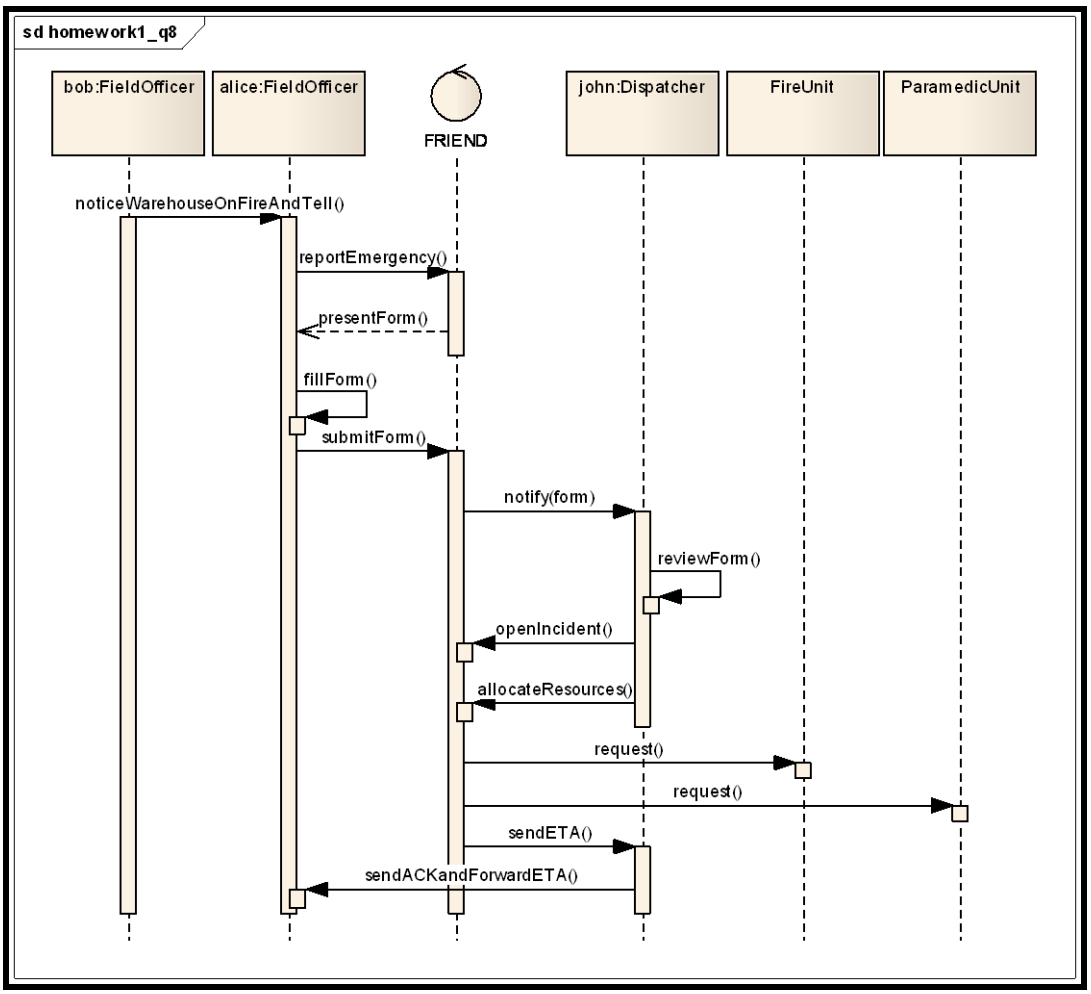

Figure 5 sequence diagram of scenario "WarehouseOnFire"

The sequence diagram shows on *Figure 5*. The "WarehouseOnFire" scenario begins with the notification of fire by Bob, ends when Alice receives the acknowledgment and the ETA. Since "FRIEND" is an accident management system, we denote it as a control, which is active component that controls what work gets done, when and how. It interacts with the fieldofficer and the dispatcher, and communicates with other resources/systems (FireUnit, ParamedicUnit).

Question 9:

i). Draw a Collaboration diagram of the similar scenario as of Question 8.

Answer:

The collaboration diagram is shown on *Figure 6*:

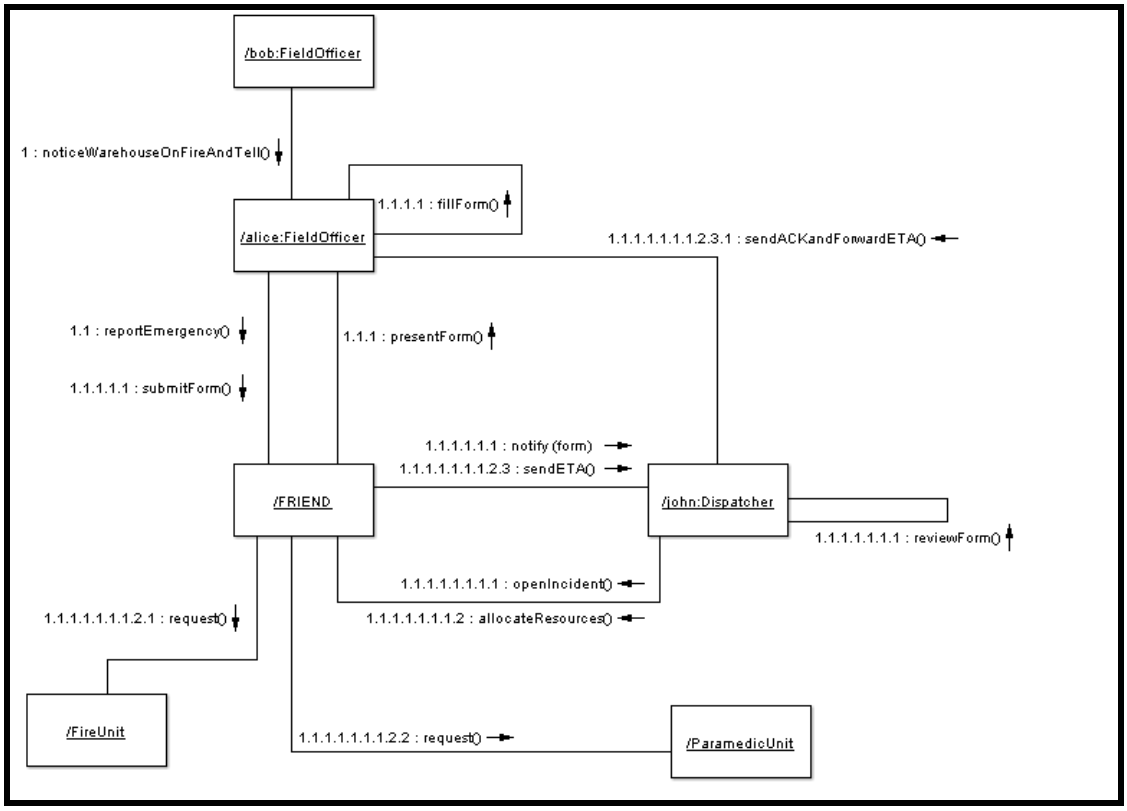

Figure 6 collaboration diagram of scenario "WarehouseOnFire"

ii). Analyze which type of Interaction diagram you would prefer.

Answer:

I prefer sequence diagram to collaboration diagram with this scenario.

This WarehouseOnFire scenario describes a sequence from field officer notice fire to send fire unit and paramedic unit. The whole sequence is an entirety. The sequence diagram can show a sequence very clearly and make reader know which happens first and which happens then. On the other hand, collaboration diagram looks upset though it describes the same scenario of sequence diagram.

iii). If you are given a scenario with numerous concurrent flows of control then which Interaction diagram will you opt for.

Answer:

If I was given a scenario with numerous concurrent flows of control, I will choose collaboration diagram.

Collaboration diagram will make the diagram tidier if it describes concurrent flows. No more association needed when add a new flow between two objects. To sequence diagram, it will cost a lot of paper to draw

the association. And most important, collaboration diagram can easily tell the concurrent relationship with flows which sequence diagram is hard to do.

e.g. Mine and my friend's laptop set up a peer-to-peer connection, I can download files from his laptop, meanwhile, I'm listening music from his music album in his laptop, moreover I'm reading his articles from his laptop as well. So as the scenario described above, if I use sequence diagram, it's hard to explain the concurrent operations with his laptop, whereas, the collaboration diagram is the best way to present such scenario.

Question 10: Draw an activity: "Opening account, deposit/withdraw iterations (with conditions checking) and closing account" as state chart diagram(s).

The three activities are illustration in three separate state chart diagrams, as following:

1) Opening accont

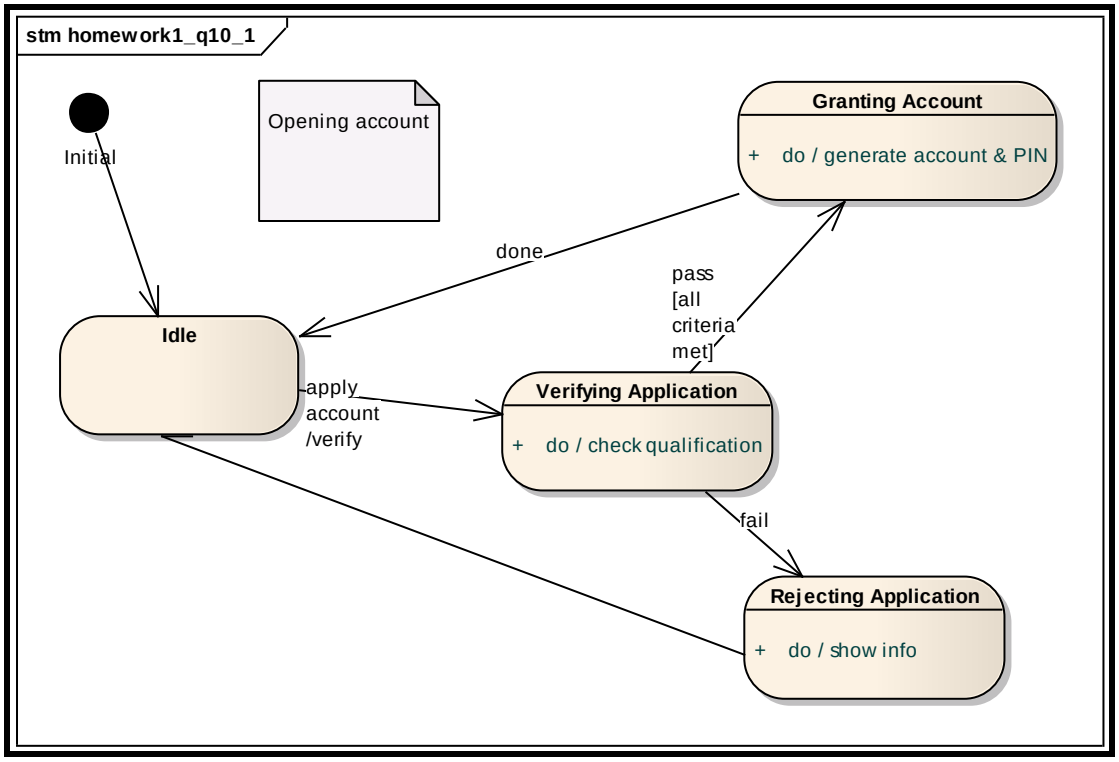

Figure 7 state chart diagram of opening account

Assume there is account opening subsystem, and normally it is idle, waiting for request. When the event of applying a new account is coming, it triggers the verification action, thereby enters the verifying state. Then the system checks the qualification of applicant again some criteria, if it matches the account number and PIN code will be generated in granting account state, otherwise the application is rejected, and eventually the system returns to idle state.

2) Deposit/withdraw iterations

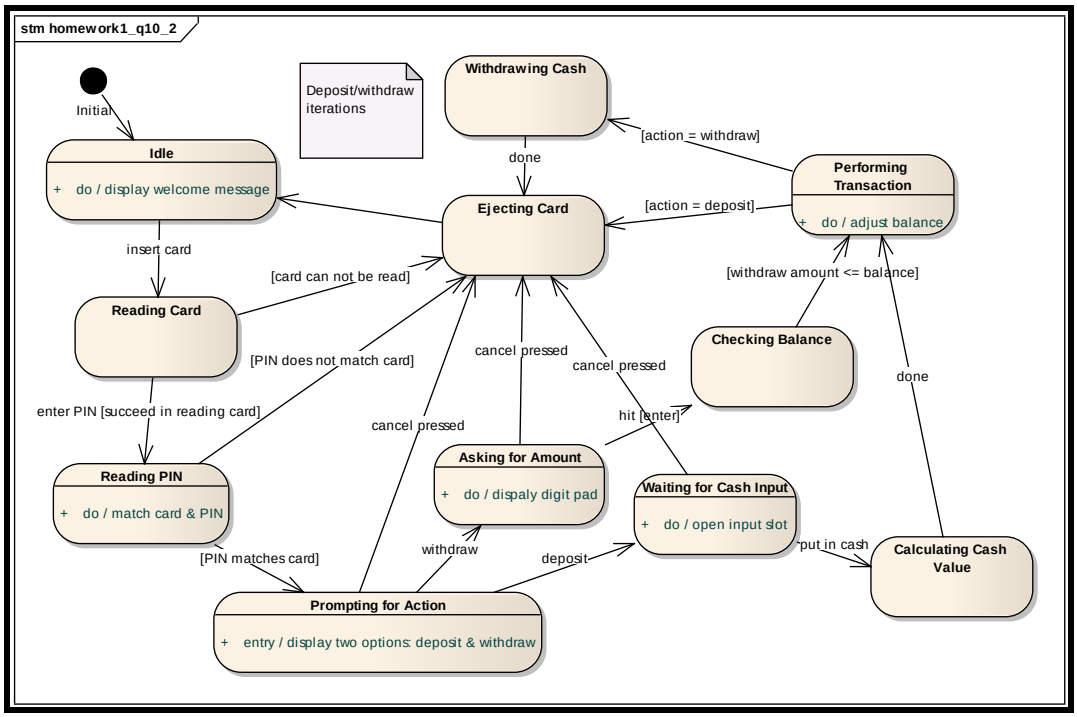

Figure 8 state chart diagram of Deposit/withdraw iterations

Unlike opening or closing account, here let's think about an ATM machine, on which we can withdraw and deposit (thought it is weird). The state chart diagram *Figure 8* is detailed enough to complain itself, the two iterations start with reading card and PIN code, set apart after prompting for action state (asking user to select between withdraw action and deposit action), and finally converge when coming to ejecting card state.

3) Closing account

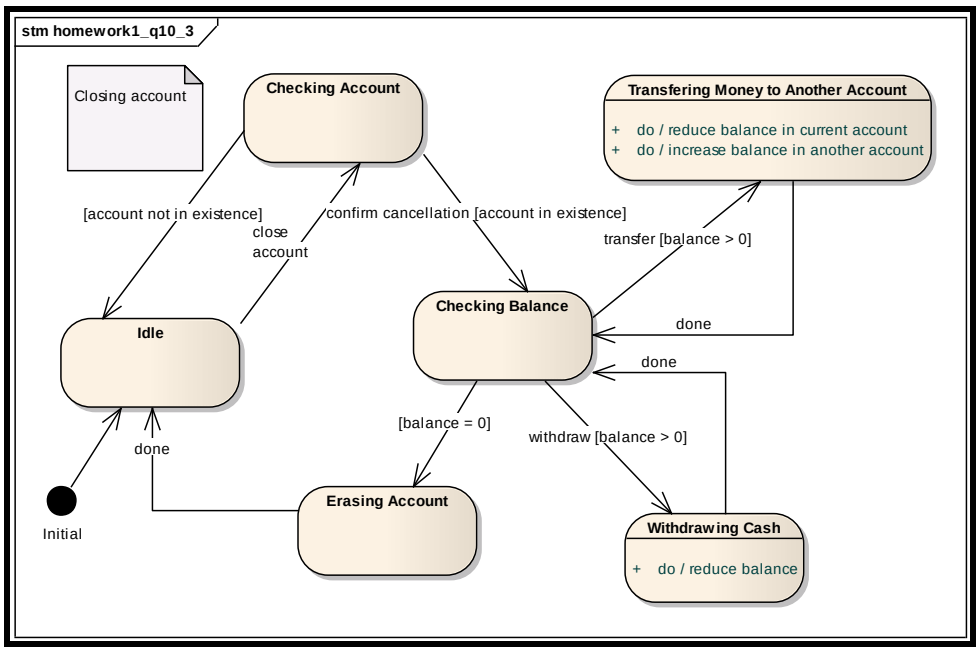

Figure 9 state chart diagram of closing account

Similar to opening account scenario, we figure out an account closing subsystem, as usual it settles in idle state at the beginning. When the system receives a close account event, it first checks whether the specific account exits or not, if there is no such account, the system goes back to idle, otherwise it waits for confirmation of cancellation, for safety sake, to proceed to balance checking state, where the system tries to clear the balance by either transferring or withdrawing, once balance becomes zero, it enters the state where all necessary operations are carried out to erase the account indeed.

Question 11: Consider the process of ordering a pizza over the phone. Draw an activity diagram representing each step of the process, from the moment you pick up the phone to the point where you start eating the pizza.

Answer:

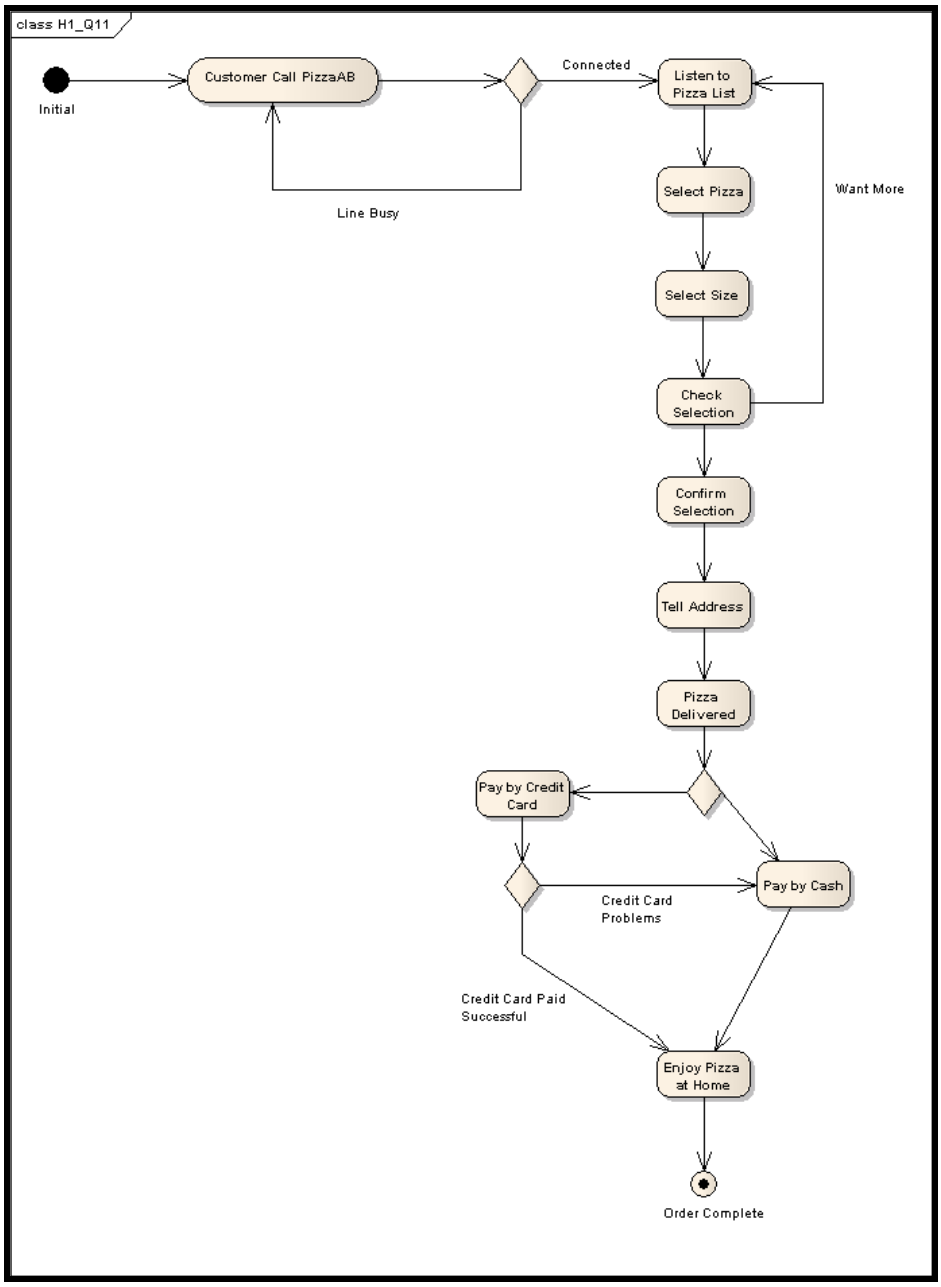

Figure 10 activity diagram of ordering pizza over the phone

As shown on the Activity Diagram *Figure 10*, customer orders pizza by telephone. We considered that maybe one pizza is not enough, so we all a function that allow customer buying more pizza. When customer pays for the pizza, we consider that the customer may want to pay by credit card or cash. And also if the credit card does not work customer has to pay by cash.

Question 12: Draw activity diagram of Boehm's Spiral Model in Figure 15-10. (01 Bonus Point)

Answer:

We depict Boehm's spiral model by a parent and a child activity diagrams.

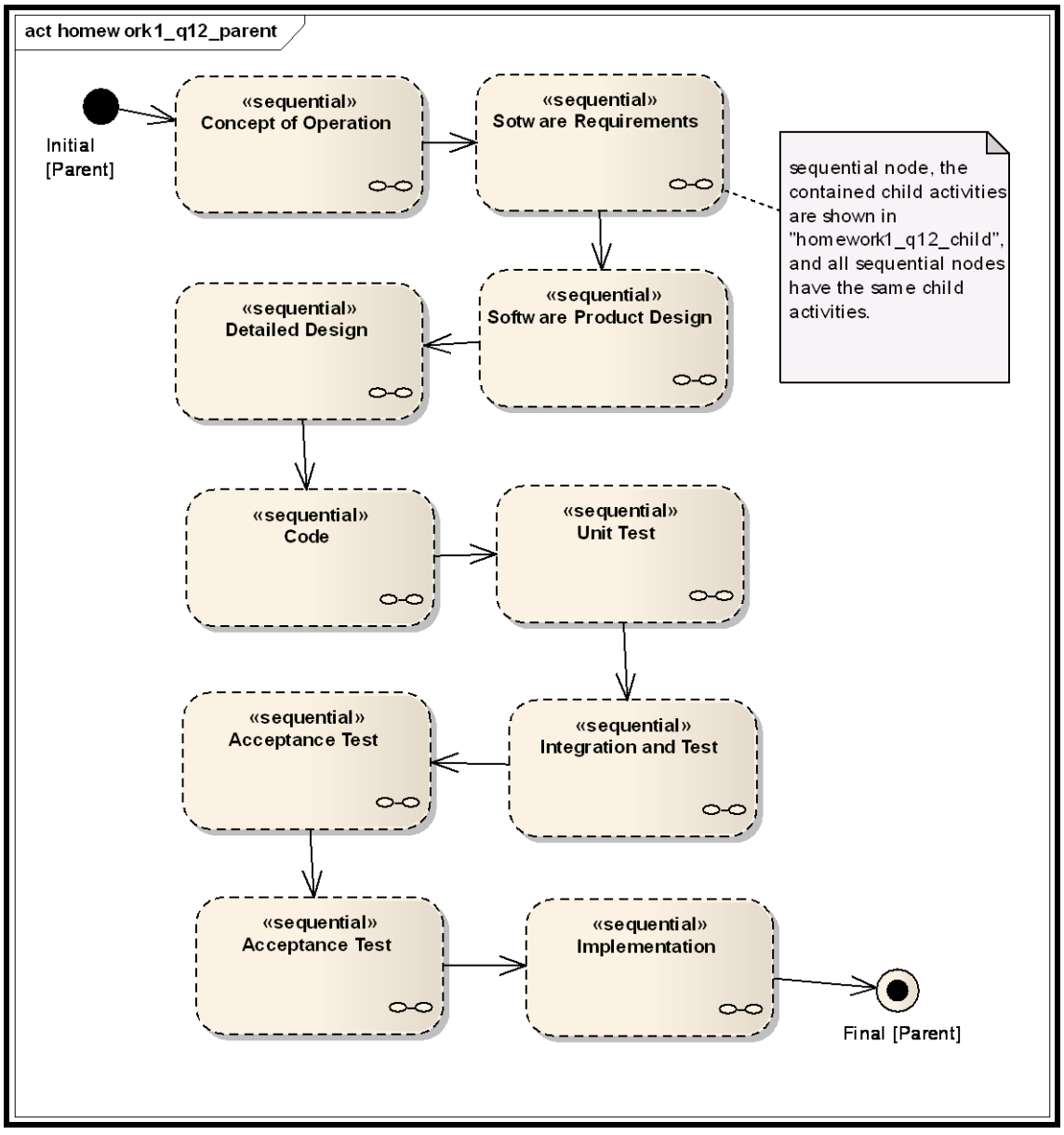

Figure 11 activity diagram of Boehm's Spiral Model (Parent)

In Boehm's spiral model, there're mainly 9 rounds, which are presented as a set of sequential nodes in the parent activity diagram *Figure 11*, starting from "Concept of Operation" till "Implementation". Each and every sequential node represents a sequential arrangement of activities, and in this case, all nodes have the same child activities as defined in Boehm's spiral model, which are depicted in the child activity diagram *Figure 12*.

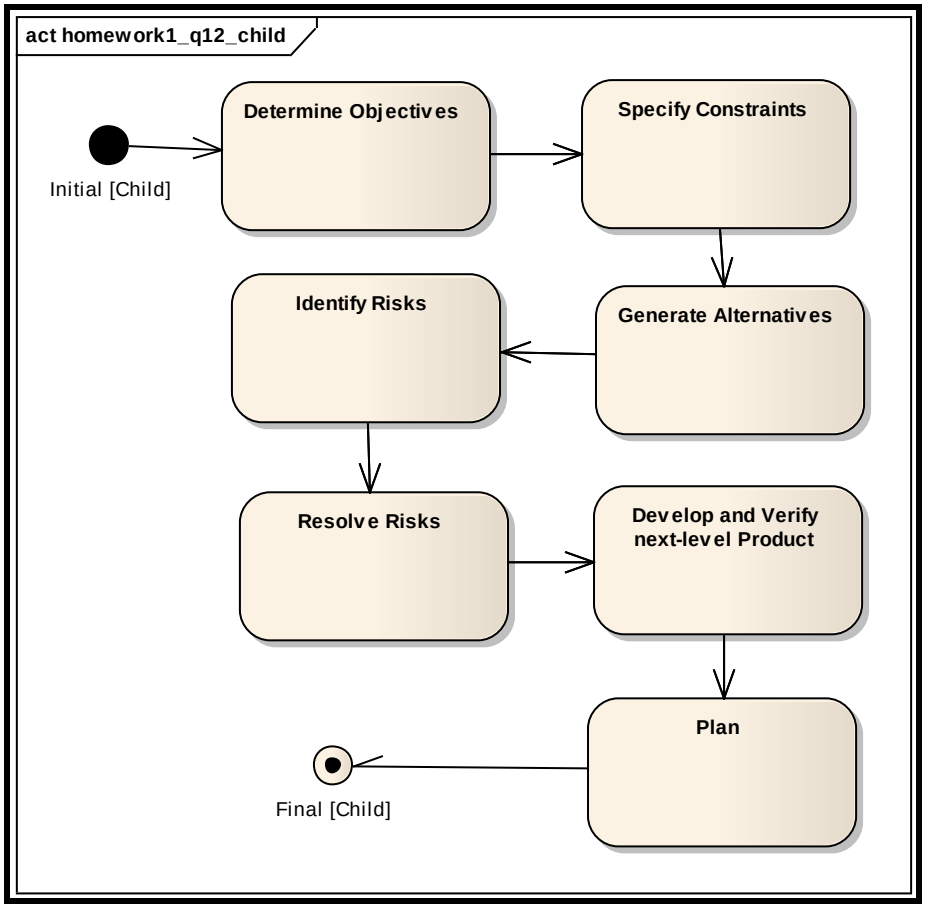

Figure 12 activity diagram of Boehm's Spiral Model (Child)

In Boehm's spiral model, each round goes through the waterfall model, including "Determine Objectives" to "Plan", so we define the above child activity diagram *Figure 12* to hold such flow, and this flow is embedded in every sequential node in parent diagram. Then by putting these two diagrams together, it reveals a gradual iteration as a spiral model.

Question 13: Give a scenario in which you would prefer to use Entity-Centered models over Activity-Centered Models. (01 Bonus Point)

Answer:

If a project comes with frequent changes, and issue is more important than activation, then we will prefer to use Entity-Centered Models over Activity-Centered Models.

If we meet a scenario as follow we will use Entity-Centered Models.

A customer comes to us and wants us to make a high-performance calculate application which can be used on the top-10 high-performance computer of the world. As new innovation comes out every day, top-10 high-performance computer can be instead by new model at any time. The customer wish the application can be very easily migrate to the new model of high-performance computer and works more effectively. We accept the order because the enticement of money. But we don't have much experience on this domain.

The model we choose should be Entity-Centered Model. For we need to solve several issues: Which platform we should use? Which implementation language should we use? How to deal with the easily migrate? What should be the initial team? How can we use top-10 high-performance computer and code on them?

If we use the Activity-Centered models maybe the new model of high-performance computer will not works well with our application. The issues we face is the key problem, not how to code can make the function more efficient nor evaluate the risk.

So we choose Entity-Centered Model rather than Activity-Centered Model.