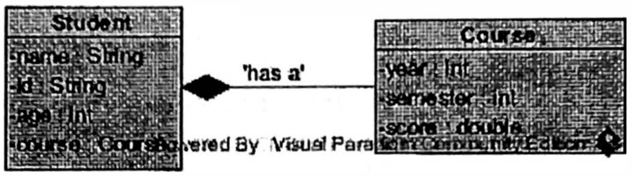
Solver: Pham Vu Tuan

Email Address: tuan007@e.ntu.edu.sg

1. (a)

- Object composition: An object can include other objects as its data member. In OO, this is called the "has-a" relationship.
- Example in Java: (I use an example from the lecture note)



Each Student 'has a' Course

```
public class Course{

        private int year;

        private int semester;

        private double score;

        public Course(int year, int semester, double score){

                this.year = year;

                this.semester = semester;

                this.score = score;

        }

        // getters and setters

}

public class Student{

        private String name;

        private String id;

        private int age;

        private Course course;  // this is composition.

        public Student(String name, String id, int age, Course
    course){

                this.name = name;

                this.id = id;

                this.age = age;

                this.course = course;
        }

        // getters and setters

}
```

Creating objects:

```
Course oodp = new Course(2, 1, 90);

Student student1 = new Student("John", "ABC123", 20, oodp);
```

(b) 3 ways of method overriding:

- Methods of a subclass override methods of a superclass
- Methods of a subclass implement the abstract methods of an abstract class
- Methods of a concrete class implements the methods of the interface

(c)
(i)

- When a message is sent (method call) to an object, the search for a matching method begins at the class of the object.
- If not found, the search continues to the immediate superclass.
- This proceeds through each immediate superclass until a matched method is found or no further superclass remain, where an error is reported.

(ii) `ClassF objectB = new ClassF();`

Code	Class involved	Error?	Explain
`objectB.print("OODP");`	ClassB	NO	
`objectB.print(2002, "OODP");`	null	YES	There is no print(int, String) methods in the class hierarchy
`objectB.print("HELLO", "OODP");`	ClassD	NO	
`objectB.print(2002);`	ClassE	NO	
`objectB.print(2002, 2015);`	ClassA	NO	

2. (a)

```
import java.util.*;
public abstract class Account{
        private String name;
        private String accountNo;
        protected double amount;

        public Account(String name, String accountNo, double amount){
                this.name = name;
                this.accountNo = accountNo;
                this.amount = amount;
        }
```

```
      public double getAmount(){ return amount;}
      public void deposit(double money){ amount += money;}
      public abstract void withdraw(double money);

}
```

(b)

```
import java.util.*;

public class NormalAccount extends Account{

      public NormalAccount(String name, String accountNo,

         double amount){

            super(name, accountNo, amount);

      }

      public void withdraw(double money){

            if(money > super.getAmount())

                  System.out.println("Over Limit!");

            else super.amount -= money;

      }

}
```

(c)

```
import java.util.*;
public class PrivilegedAccount extends Account{
      private double limit;

      public PrivilegedAccount(String name, String accountNo, double
amount, double limit){
            super(name, accountNo, amount);
            this.limit = limit;
      }

      public void withdraw(double money){
            if(money > super.getAmount() + limit)
                  System.out.println("Over Limit!");
            else super.amount -= money;
      }
}
```

(d)

Static binding:

```
import java.util.*;
public class AccountApp{
	public static void main(String[] args){
		String name = "Nicholas";
		String accountNo = "12345678";
		double amount = 3000;
		double money = 500;
		double limit = 500;
		NormalAccount normal = new NormalAccount(name, accountNo, 
amount);
		PrivilegedAccount privileged = new 
PrivilegedAccount(name, accountNo, amount, limit);
		withdraw(normal,  money);
		withdraw(privileged, money);
	}

	public static void withdraw(NormalAccount normal, double 
money){
		normal.withdraw(money);
	}

	public static void withdraw(PrivilegedAccount privileged, 
double money){
		privileged.withdraw(money);
	}
}
```

<u>**Dynamic binding**</u>

```
import java.util.*;
public class AccountApp{
	public static void main(String[] args){
		String name = "Nicholas";
		String accountNo = "12345678";
		double amount = 3000;
		double money = 500;
		double limit = 500;
		Account normal = new NormalAccount(name, accountNo, 
amount);
		Account privileged = new PrivilegedAccount(name, 
accountNo, amount, limit);
		withdraw(normal,  money);
		withdraw(privileged, money);
	}

	public static void withdraw(Account account, double money){
		account.withdraw(money);
	}
}
```

3. (a)

(i)

```
#include<iostream>

#include<string>

using namespace std;

class Weapon{

public:

        Weapon(int a, int w, string s) : weight(w),
        attackRate(a),sound(s) {};

        ~Weapon(){};

        int getAttackRate(){ return attackRate;}

        int getWeight(){ return weight;}

        void use(){ cout << sound << endl;}

        virtual int calDamage() = 0;

private:

        int weight;

        int attackRate;

        string sound;

};
```

(ii)

```
#include "weapon.cpp"

class Gun : public Weapon{

public:

        Gun(int a, int w, string s, int p, int r, int am) :
        Weapon(a,w,s), power(p), range(r), ammo(a){};

        int getRange(){ return range;}

        int loadAmmo(int am){

                ammo += am;
```

```
            return ammo;
        }
        int getAmmo(){ return ammo;}
        void use(){
            if(ammo != 0) ammo--;
            Weapon::use();
        }
        int calDamage(){ return Weapon::getAttackRate()*power;}
private:
        int power;
        int range;
        int ammo;
};
```

(III)

```
Player operator+(const Weapon gun){
        Player player;
        player.setRightWeapon(gun);
        return player;
}
```

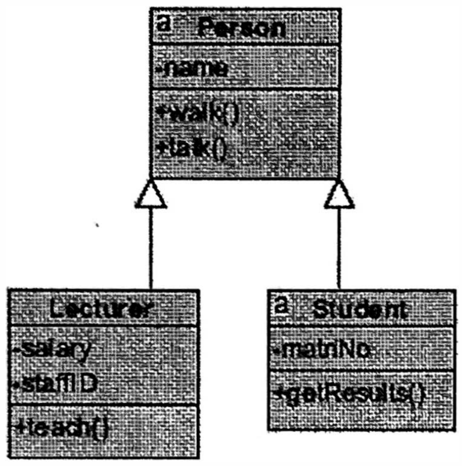
(b)

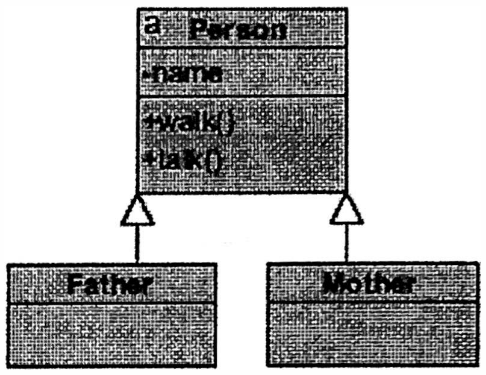
(I) Role versus Inheritance: About deciding whether it is appropriate/valid to subclass OR is it just an object instance (role) of a class rather than a subclass.

Both Inheritance and Role represent "is-a" relationship. But Inheritance, besides being a generalization, is also about subclass specialization. Subclass specializes via having addition attributes or methods with at least overriding.

I use the example from the lecture note:

- Student and Lecturer are specialization of Person
- Father and Mother are not specializations, both Father and Mother are more of a role – like role name in Class diagram, an instance of Person rather than a class





(I checked with the lecturer for this answer, it should be correct ☺)

(ii)

<u>Liskov Substitution Principle:</u> "if for each object o1 of type S there is an object o2

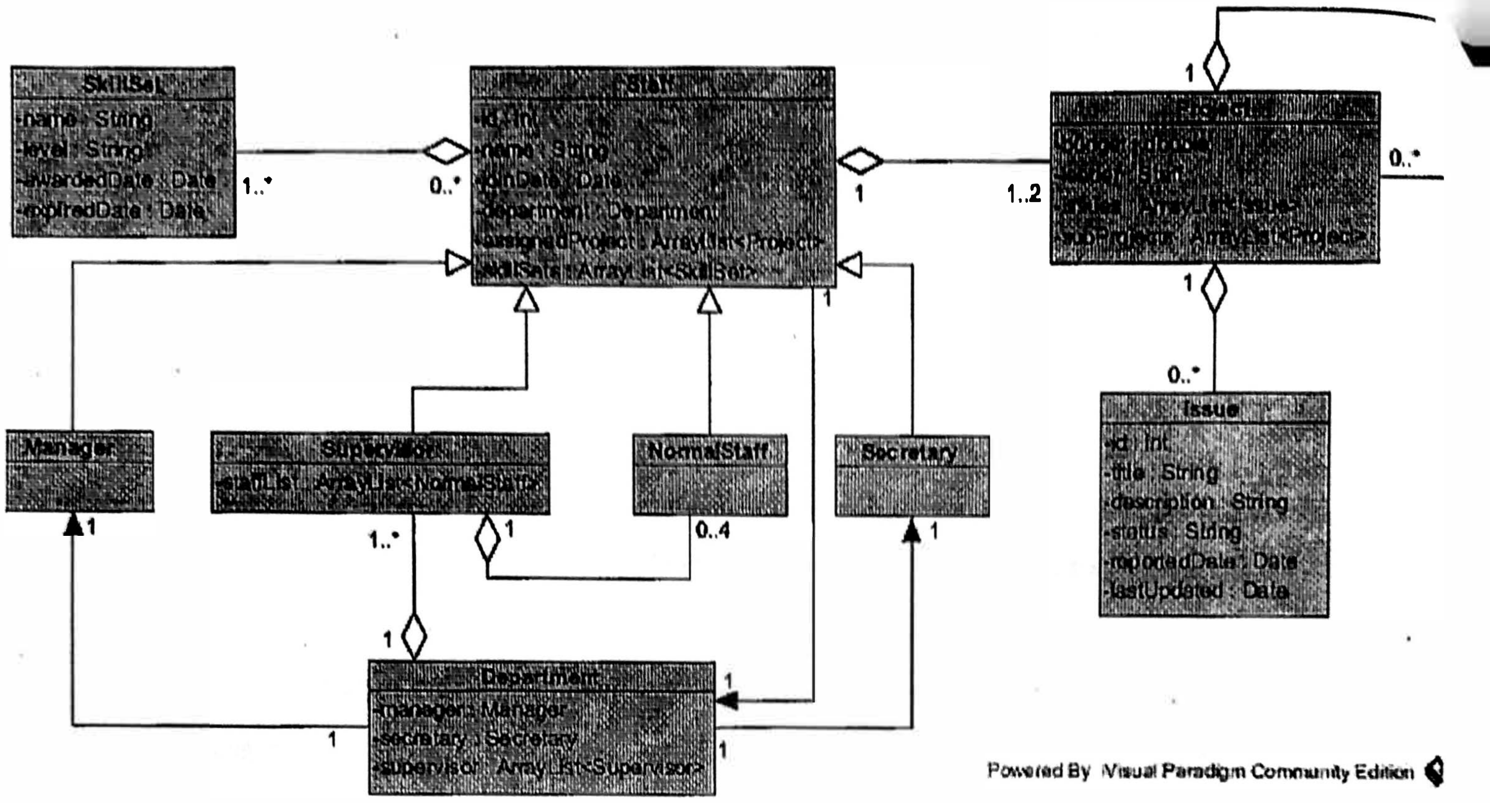
of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2thenSis a subtype of T."

- <u>Explain:</u>
  - Subtypes must be substitutable for their base types.
  - a user of a base class should continue to function properly if a derivative of that base class is passed to it.
- <u>Design by contract:</u> Methods (public interfaces) should specify their pre and post conditions: what must be true before and what must be true after their execution, respectively.
- Restating the LSP again, in terms contracts, a derived class is substitutable for its base class if:
  - Its pre-conditions are no stronger than the base class method. (expect no more)
  - Its post-conditions are no weaker than the base class method. (provide no less)

<u>Example:</u>

- Circle is a subclass of Ellipse with the length of major axis = length of minor axis.
- Ellipse class has 2 methods setMajor() and setMinor().
- Because Circle inherits from Ellipse, according to LSP, it must also have setMajor() and setMinor() method. But if we use one of these methods individually on Circle, the method will change the circle into something which is no longer a circle. This violates the LSP.
- You can read more at: https://en.wikipedia.org/wiki/Circle-ellipse_problem

4. (a)



This question has no right/wrong answer, draw the diagram clearly and you will get the score.

(b)

```
import java.util.*;
public class HandlerA extends Handler{
      private Handler h;

      public void setHandler(Handler h){
            this.h = h;
      }
      public void handleRequest(Request req){
            int value = req.getValue();
            if(value >= 0) handle(req);
            else if(h!=null) h.handleRequest(req);
      }
      public void handle(Request req){}
}
```

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺