20th CSEC – Past Year Paper Solution *2016-2017 Sem 2*
CE/CZ 2002 – Object Oriented Design & Programming

1) a) An object is an entity that contains both the attribute that describes the state of a real-world object and the actions that are associated with the real-world object.

b) State of an object, attributes, behaviour of an object

c) *Editor's note:* Answer has been modified as question is asking for naming conventions.

- Lowercase is where all the letters in a word are written without any capitalisation (e.g., packagename). It is used for package name.
- Uppercase is where all the letters in a word are written in capitals. When there are more than two words in the name use underscores to separate them (e.g., MAX_HOURS, FIRST_DAY_OF_WEEK). It is used for constants, enums.
- CamelCase (also known as Upper CamelCase) is where each new word begins with a capital letter (e.g., CamelCase, CustomerAccount, PlayingCard). It is used for classes and interfaces.
- Mixed case (also known as Lower CamelCase) is the same as CamelCase except the first letter of the name is in lowercase (e.g., hasChildren, customerFirstName). It is used for methods, variables.

d) i) Compile-time error, InterfaceL does not have a moveIt method
ii) Compile-time error, instance of ClassBP cannot be cast to ClassB
iii) ClassBP's feedIt(v: ClassV, z: ClassM)
iv) ClassM's feedIt(o:Object)
v) ClassM's feedIt(o:Object)
vi) ClassP's moveIt(o:Object)

2
```
class HourlySaleApp {
    public static void main(String arg[]){
        Sale normalSale = new Sale("Samsung Galaxy Note 7", 600.00);
        DiscountSale discountSale = new DiscountSale("Samsung Galaxy
            Note 7", 600.00, 0.5);
        System.out.println(normalSale.lessThan(discountSale));
    }
}
class Sale {
    private String name;
    private double price;
    public Sale(String name, double price) {
        this.name = name;
        this.price = price;
    }
    public double getPrice() {return price;}
    public boolean lessThan(Sale otherSale) {
        if(price < otherSale.getPrice()) return true;
        else return false;
    }
    public double bill() {System.out.println("price: " + price);}
}
```

```
class DiscountSale extends Sale {
    private double discount;
    public DiscountSale(String n, double p, double d) {
        super(n, p);
        this.discount = d;
    }
    public double bill() {
        System.out.println("price: " + (getPrice() * discount));
    }
}
```

3) a) i)

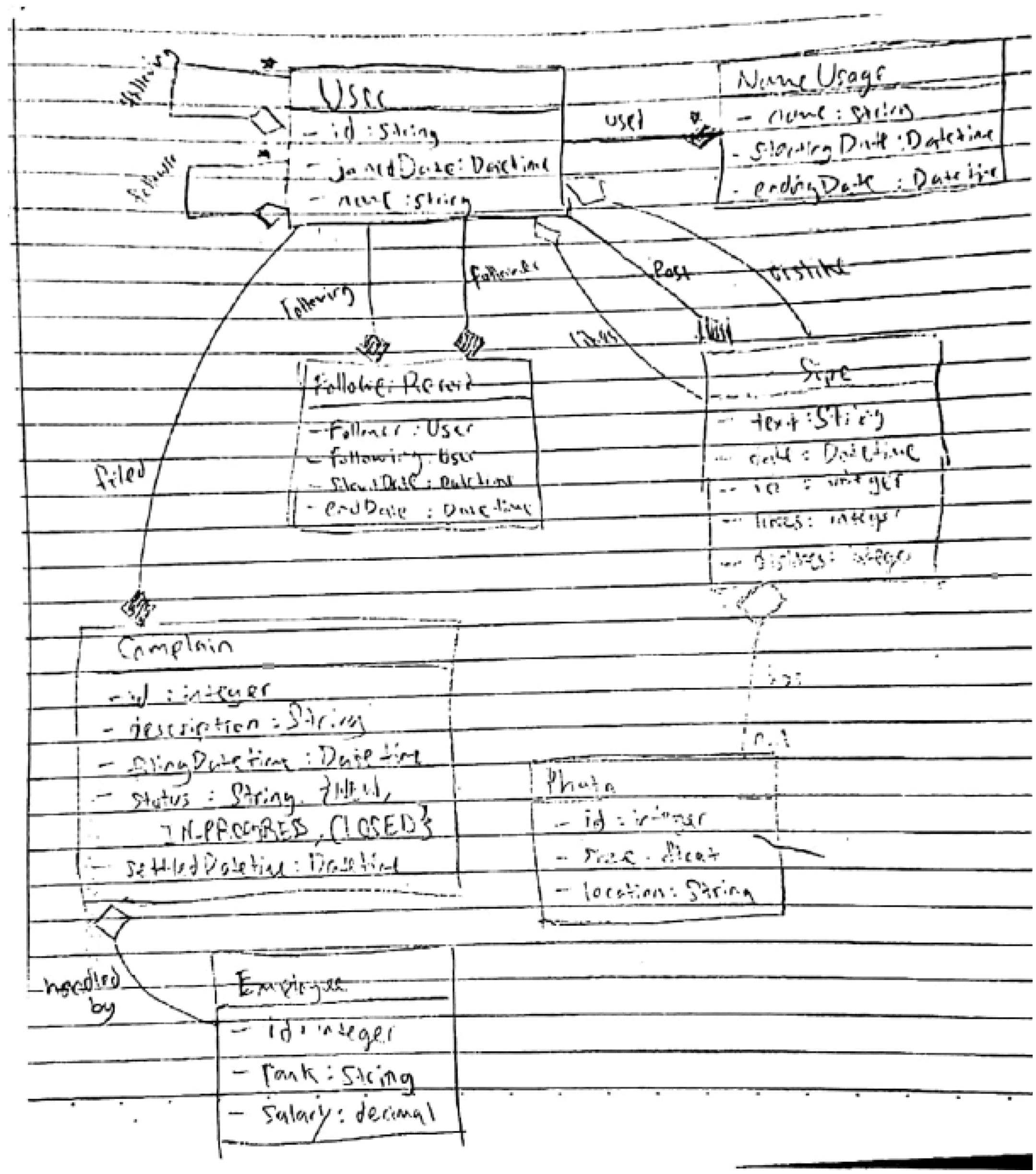
```
//in vector3d.h
#include <vector2d.h>
#include <fraction.h>

class Vector3D : public Vector2D {
  protected:
    Fraction z;
  public:
    Vector3D(Fraction x, Fraction y, Fraction in_z);
    Vector3D(int nx, int dx, int ny, int dy, int nz, int dz);
    void print();
    Vector2D& operator+(Vector2D &v);
}
```

ii)

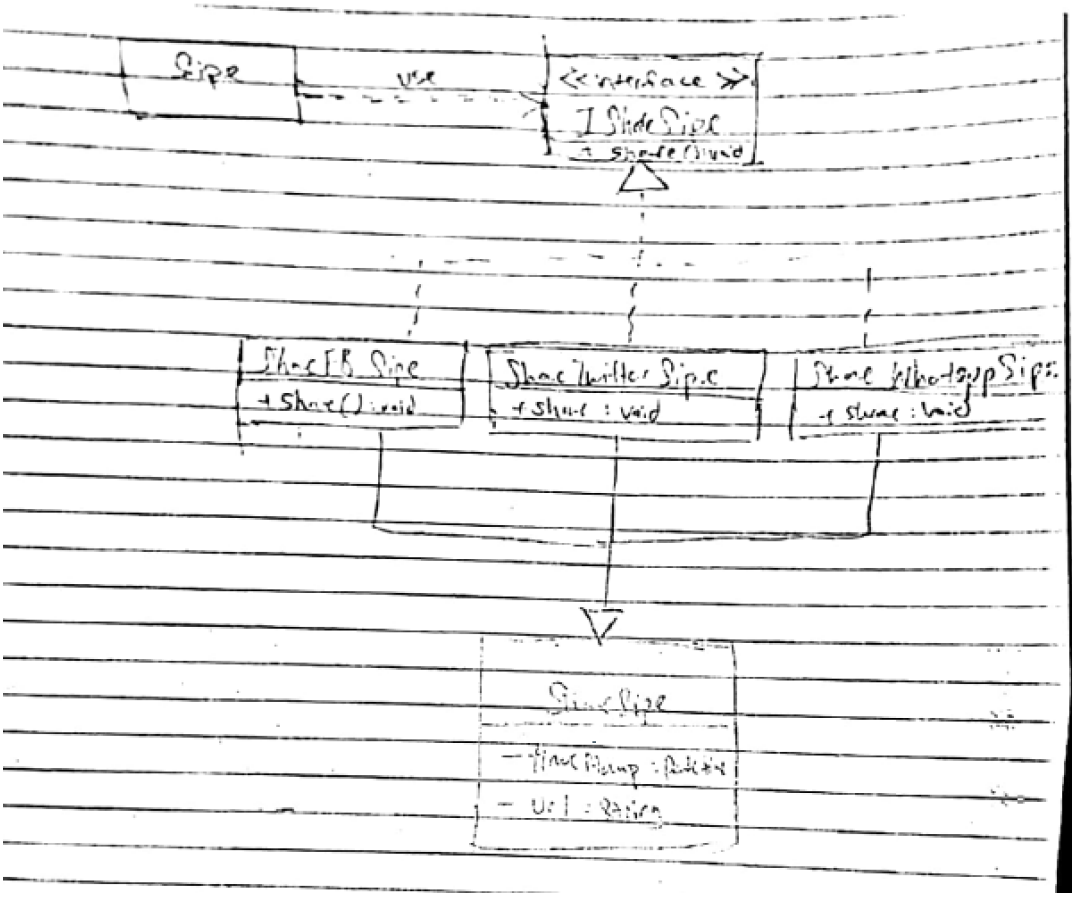
```
#include <vector3d.h>

Vector3D::Vector3D(Fraction x, Fraction y, Fraction in_z) :
Vector2D(x, y), z(in_z){};

Vector3D::Vector3D(int nx, int dx, int ny, int dy, int nz,
int dz) : Vector2D(nx, dx, ny, dy) {
  z = new Fraction(nz, dz);
}

Vector3D::print() {
  cout << Vector2D::x << "i " << Vector2D::y << "j " << z <<
    "k" << endl;
}

Vector3D::operator+(Vector2D &v) {
    Vector3D &a = dynamic_cast<Vector3D&>(v);
    if (a == NULL) {return *this;}
    else
      return new Vector3D(a.Vector2D::x + Vector2D::x,
        a.Vector2D::y + Vector2D::y, a.z + z);
}
```

b)
```
class Local {
    public boolean emit(String msg, String type) {
        if (type != null) {
            Emitter em = new Emitter(type, this);
            boolean status = false;
            for (int count = 0; count < 10; count++) {
                status = em.connect();
                if (status) {
                    em.send();
                    break;
                }
            }
            return status;
        }
        else {
            displayError();
            return false;
        }
    }
    public String getAddress() {return ""};
    public void displayError() {};
}

class Emitter {
    private String type;
    private Local local;
    private String addr;
    public Emitter(String type, Local local){
        this.type = type;
        this.local = local;
    }
    public boolean connect() {return true;}
    public void send() {addr = local.getAddress();}
}
```

4) a)


User
- id : String
- name : String
Name Usage
- name : String
Follower Record
- Follower : User
- endDate : DateTime
Complain
- id : integer
- description : String
- Status : String {NEW, IN-PROGRESS, CLOSED}
Photo
- id : integer
- location : String
Employee
- id : integer
- rank : String
- salary : decimal
filed
following
handled by
post

b)



ShareFBSipe, ShareTwitterSipe and ShareWhatsappSipe class contain corresponding implementation needed for sharing Sipe on Facebook, Twitter and Whatsapp. ShareSipe class contains the common attributes and functions that are required for sharing Sipe. This class allow reusing of common attributes and functions when sharing Sipe. As shown in the above drawing, ShareFBSipe, ShareTwitterSipe and ShareWhatsappSipe class inherited ShareSipe for the **reusability** and at the same time avoid duplication of code thus it will have better **maintainability**. Modification would cause less inconsistency.

As interface IShareSipe is used and implemented by different sharing strategy, SharingFBSipe, SharingTwitterSipe and SharingWhatsappSipe class, this helps to provide common interface so that new sharing strategies can be added by implementing the interface. Thus, provide a better **extensibility**. Interface also enforces all the implementation classes to implement the functions required. Thus, changes made in the implementation will not affect classes that are using them as long as the interface remains the same.

--End of Answers--

Solver: Shao Jie