

OOD COURSEWORK

PART 1 - As the first step of development of the system, you will be required to **describe the following terms with code or diagram** where it is applicable for the following topics:

Question 1

1. Characteristics of the OO design paradigm

a. Classes

- A class is a blueprint and it defines the structure of the objects that belongs to the class

1. Instance variables

- These are the variables that are declared inside the class but outside the scope of any method.

2. Instance methods

- It is a method defined in a class which is only accessible through the object of the class

3. static variables

- one copy of this variable is shared by all the objects that belongs to that class .

4. static methods

- these are methods in java that can be called without creating an object of a

class

5. Constructors

- A constructor is a method used to initialize the instance variables of the object at the creation of the object.

6. Getters and setters

- Getters and setters are methods which are used to protect the data and make the code more secure.

Let me use a diagram in order to give an example for all the above discussed topics

Im going to have a class called Shapes

```
public class Shapes {
```

```
    private double length ;  
    private double width ;
```

these are instance
variables

```
    public Shapes(double length, double width){
```

```
        this.length = length;
```

```
        this.width = width;
```

```
    }
```

This is the
constructor

```
    void area () {
```

```
        length = 10;
```

```
        width = 20;
```

```
        area = length * width
```

```
        System.out.println("the area of shape is : " + area)
```

```
    }
```

This is an instance
method

```
Static int Count(){  
  
    Return 15;  
  
}
```

This is a Static variable

```
public void setLength(double length){  
  
    this.length = length  
  
}
```

```
public void setWidth(double width){  
  
    this.width = width;  
  
}
```

These are setter
methods

```
public double getLength(){  
  
    return length;  
  
}
```


```
public double getWidth(){
```

These are getter
methods

methods

```
        return width;
    }

//main algorithm
    public static void main(String[] args) {
static method
        Shapes shape = new Shapes();
        System.out.println("Hi this is my main program")
    }
}
```



This is a Static method

b.Objects

- An object is an instance of the class

c. OOP Concepts

1. Abstraction

Abstraction is the process of hiding certain details and showing only essential information to the user . This can be achieved with the help of abstract classes and methods.

Example :

```
Abstract class Human {  
  
    public abstract void humanWalk();  
  
    public void walking(){  
  
        System.out.println("krkrkrk")  
  
    }  
  
}
```

2. Encapsulation

- Encapsulation is the process of wrapping code and data together in a single unit

Example :

```
Public class Movies {  
  
    private String movieTitle;  
  
    private date  movieDate;  
  
    private String movieDirector;  
  
    public void setMovieTitle (String movieName) {  
  
        movieTitle= movieName ;  
  
    }  
  
}
```

```
public void setMovieDirector(String directorName) {  
  
    movieDirector= directorName ;  
}  
  
public void setMovieDate (date launchDate) {  
  
    movieDate= launchDate ;  
}  
  
public String getMovieTitle () {  
  
    return movieTitle;  
}  
  
public String getMovieDirector () {  
  
    return movieDirector ;  
}  
  
public date getMovieDate () {  
  
    return movieDate ;  
}  
  
}
```

3. Inheritance

- Inheritance allows a class to have a parent class from which it can inherit variables and methods.

Example:

```
public class Employee{

    private String employeeName;
    private String employeeID;

    public Employee (String name , string id){

        employeeName = name ;
        employeeId = id ;
    }

    public void setEmployeeName (String nameForEmployee){

        employeeName = nameForEmployee;

    }

    public void setEmployeeId (String idForEmployee){

        employeeId= idForEmployee;
```

```
}
```

```
Public String getEmployeeName(){  
  
    return employeeName ;  
}
```

```
Public String getEmployeeId(){  
  
    return employeeId ;  
}  
}
```

```
public class Doctor extends Employee{  
  
    private String department ;  
  
    private double salary;  
  
    public Doctor(String nameOfEmployee, String idOfEmployee, double  
salaryAmount , String nameDepartment) {  
  
        super(nameOfEmployee , idOfEmployee) ;  
        department = nameDepartment  
        salary = salaryAmount  
    }  
  
}
```


4. Polymorphism

- Using Polymorphism we can use the same code to process objects of various types and classes , as long as they have a common super class

Example ;

```
class Animal {  
  
    public void animalBehavior() {  
  
        System.out.println("Now the animal is doing something")  
    }  
}
```

```
class Lion extends Animal {  
  
    public void animalBehavior() {  
  
        System.out.println("The lion is hunting for its prey")  
    }  
  
}
```

```
class Cat extends Animal {  
  
    public void animalBehavior() {  
  
        System.out.println("The cat is sleeping")  
    }  
  
}
```

```
class Elephant extends Animal {
```

```

    public void animalBehavior() {

        System.out.println("The elephant is bathing")
    }
}

```

5. Abstract classes and abstract method

- Abstract classes are used to hide unwanted information and we generally cant create objects from these classes . The Abstract class may have abstract methods or normal methods or it can have both.
- In an abstract method , we don't have a body. However if there is an abstract method in the class , the class should be declared as abstract

Example :

```

abstract class NationalityGreeting {

    abstract void sayGreeting();

    public void hello{

        System.out.println("Hello!")

    }

}

class SriLankan extends NationalityGreeting {

```

```

        public void sayGreeting(){

            System.out.println("Ayubowan!")

        }

    }

class Spanish extends NationalityGreeting{

    public void sayGreeting(){

        System.out.println("Hola!")

    }

}

```

6. Interface

- An interface is a fully abstract class which contains abstract methods.
- We can identify if a class is an interface using the interface keyword.

```

Interface NationalGreeting {

    public void sayGreeting();
    public void hello();

}

```

Example

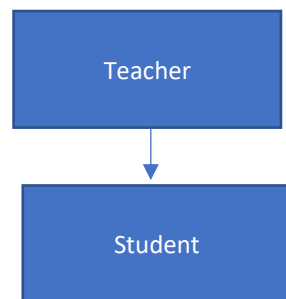
Question 2

2. Different types of relationships between classes in the UML class diagram

- Association

An association indicates that the use case somehow interacts and communicates with each other.

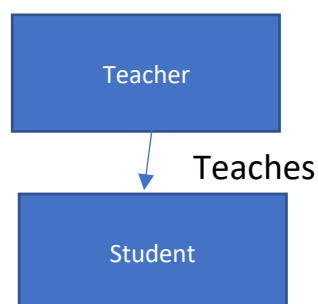
Eg



- Directed Association

In this type of association we are given a directional relationship which helps us find how the use case is interacting .

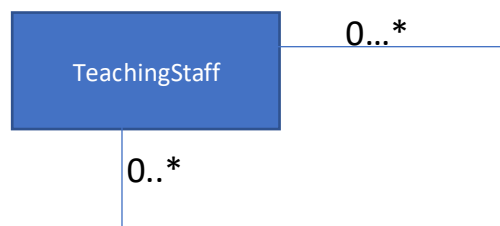
Eg.



- Reflexive Association

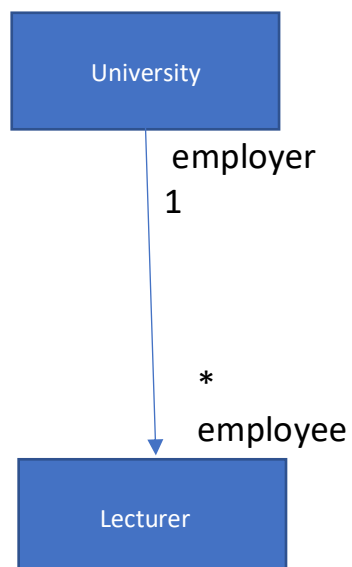
This type-of relationship indicates multiple functions or responsibilities.

For example a teacher in the teaching staff may be a computer science teacher ,math teacher , biology teacher or even a physical education teacher



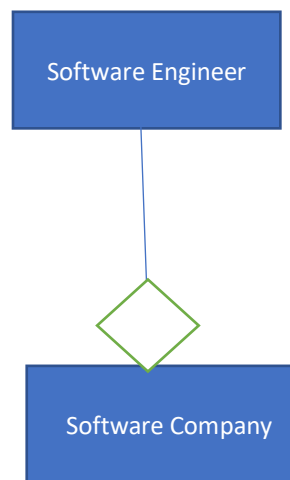
- Multiplicity

Multiplicity constrains the number of objects of a class that can be involved in a particular relationship at any point in time.



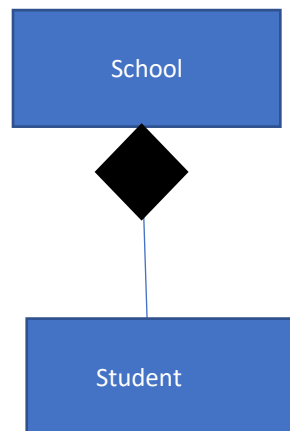
- Aggregation

Refers to the formation of a class as a result of one class being aggregated or built as a collection. For example we can say a software engineer has a relationship with the software company



- Composition

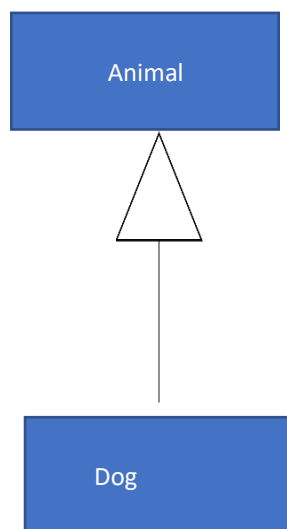
Composition relationship is similar to aggregation but it shows the dependency of a contained class in the life cycle of container class. For example a student will have no school if the school shuts down .



- Inheritance / Generalization

Inheritance or generalization refers to a relationship where one class is a child / sub class of a parent class and will have the same functionalities as that of its parent class

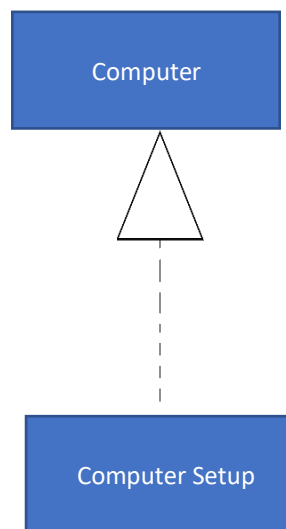
For example Dog is inherited from the Animal class .



- Realization

Realization relationship shows the implementation of the functionality defined in one class by another class .

For example the different features and likings that are set using the computer set up interface are been implemented by the computer.



Question 3

3. Design patterns and their relationship with the OO design principles

- Singleton Pattern

Singleton design pattern helps to make sure that there will be only one instance of an object created from your class and it will prevent any other objects been created

- Strategy Pattern

In this pattern , we separate the behaviors in a class which act different and only keep the ones which act in a similar manner and the ones which are separated are treated individually as a separate class that implements an interface.

For example traveler route . Lets say we have an application that will be able to provide a traveler with a route to his destination, but the issue is here one location can have many routes and each route maybe different from one another, for example a traveler can go to Hambantota from the highway or from the normal road .

- Command Pattern

This pattern focuses on providing a solution when there's a problem where an object has to perform some commands during run time.

For example read , write delete , edit in a word document .

- State Pattern

This design pattern is used when there are different states to navigate between more than one state during run time.

States are objects where each state looks into a different type of user communication. Eg washing machine

UML CLASS DIAGAM

