OOC

features of OOC

1. Abstraction - It is a process on removing unnecessary data and considering only essential characteristics that needed for the particular system.

2. Encapsulation - It is a process of grouping related attributes and methods together and It's hides the details of the implementation of an objects.

3. Polymorphism - Simple meaning → It is one name many forms. That is same methods or attributes differently in different scenarios.

4. Inheritance - The ability to inherit the properties of one class to another class, or inherit the properties from a base class to an inherited class.

5. Class - class is a blueprint for creating objects. It consist of member variables and member functions.

6. object - An object is an instance of a class, without an object no memory is allocated to a class's data member or member functions.

* OOP - OOP is a method of implementation in which programs are organized as a collection of objects which cooperate to slove a problem.

Date No

CRC cards - class Responsibility and colaboration cards

* using CRC cards we can identify actions of given class as well as we can identify relation-ship ~~ano~~ among thees classes.

class name	
Responsibilities	colaborations

Analysis classes

* Entity classes - classes, that we can directly identify looking at user requirements.

* Boundary classes - certain activities can not be done by ~~a~~ the class itself. we need an outside ~~a~~ class to do it. Those type of classes are boundary clz. (Actors)

* control classes - There are some situations that you need to consider system itself as a clz, so these classes are called to control clzs.

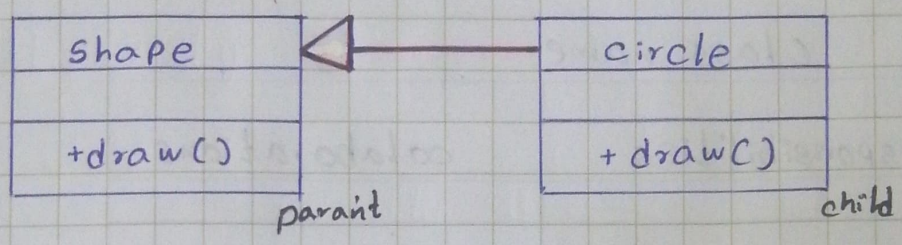
- An Entity class is typically responsible for handling one object at a time.

Relationships

Date No

1) Inheritance (is a) ⟶

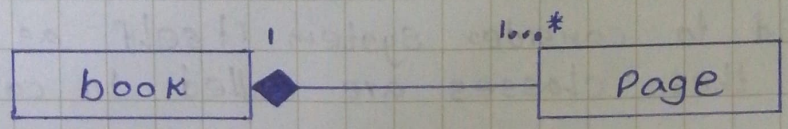
* when one class is a type of another class.
* Known as Generalization.
* in a UML notation arrow head is alway pointed to ~~sp~~ super clz / parent clz.



* All the commen details include in the parent clz and the will be automaticaly inherited by child clzes.

2) Composition (Part-of) ⟶◆

* when parts can not exist without the whole.
* Always larger clz is a whole clz.
* An objects ~~of~~ of the whole has objects of the part.
* This should have a relationshin with a multiplicity of "1..."

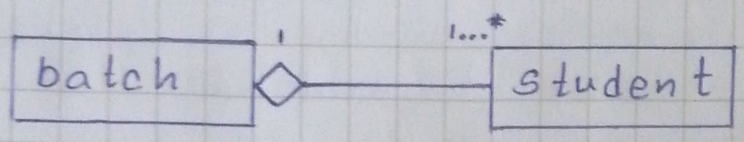


* page cannot exist with out a book.

Date No

3) Aggregation

* part can exist without the whole.
* can also have a multiplicity of " 0... ".



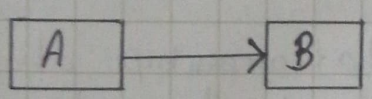
* student can exist without the batch.

4) Association. (Has a)

- bi-directional (two way)

* both classes will has an instence of the other clz.
* both classes will has 'has a' relationship.

- uni-directional (one way)



* class A uses and contains one instance of class B, but B doesn't know about it or contain ant instances of class A.

* The dependent class (A) defines an instance of associated class (B).

5) Dependency (uses) ------->

* weekest relationship.
* Always uni directional.

Multiplicity

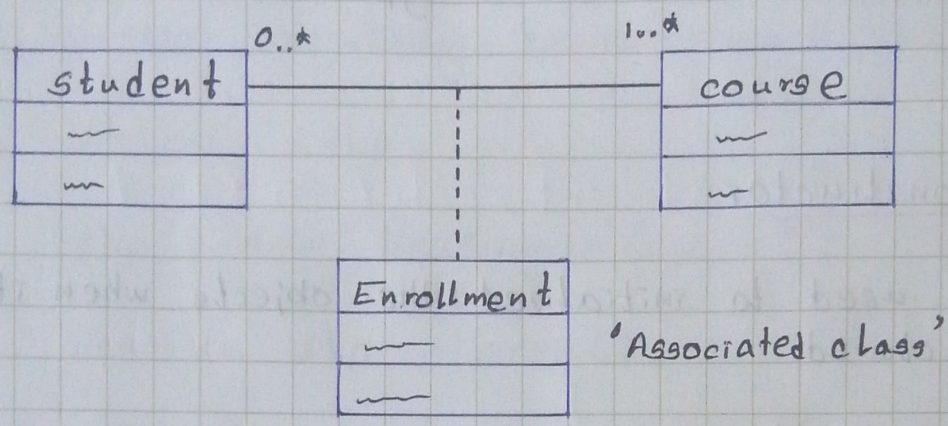
* 1 - Exactly one
* n - Exactly n
* 0..1 - Zero or one
* 0..* - Zero or more
* 1..* - one or more
* n..m - betwen n and m
* 0..1, 3..5 - zero or one, or between 3 and 5 (0, 1, 3, 4, 5)

* when multiplicity is Exacty one (1) we do not mention ~~in~~ it in a class diagram.

* In a composition relationship alway whole side is one. so we don't write muliplicity on that side.

Association class.

* class ~~ta~~ that is part of an association relationship between two other classes. we can attach an association class to an association relationship to provide additional informations about the relationship. { methodes , attributes }



- Super class

* The parent class whose properties are inherite by another class (child class)

* Also known as base class , Ancestor class.

* Generalization happens here.

- Derived class

  * child class.
  * Sub class.
  * Decendant class.
  * specialization happens here.

Overloading Function

In C++ we can have multple functions with the same name, but having differant types of parameters. It's called function overloading.

* Parameters of each function should be different.
* Fun. name and return type same.

Constructor

is used to initialized the objects when it is declared.

* default constructor
  used to initialized attributes to default values.

* Overloaded constructor
  used to assign values sent by the main program as arguments.

~Distructor

can be used to release memory of attributes that ~~wr~~ were created dinamically when the object was created.

Date No

Function overriding

when a derived class defines a function that is already defined in the base class, (same function in both classes) it is called function overriding. It helps us achieve runtime polymorphism.

But, when we are created dynamic objects this function overriding doesn't work properly.

```
shape *shp;

shp = new Rectangle(10, 5);

cout << shp -> area();
```

→ output = 0

In the above code first we are create a pointer to shape class and then create a object to Rectangle type. but when we run this shape area() function is called, not a Rectangle function area().

Because in the code 2nd line, Rectangle object in created at a run time. not the complie time. so the compiler finally run shape area() function.

to avoid it we can use virtual function.

Virtual Function

we can create functions that we are overriding as a virtual fuunctions. This enables dynamic blinding where the overriden methods are called correctly at run time.

These are ~~s~~ support dynamic polymorphism. vertual function is a member function that is declared with in the base clz and redifien by a derived clz.

```
virtual int calcArea () {
        return 0;
}
```

Abstract class

class ~~tat~~ that has at least one pure virtual function.

```
Virtual int area () = 0;
```

Some times we want to prevent create of objects of given class.

we can't declare an instance (object) of an abstract base clz. we can use it only as a base clz when declaring other clzes.