OOC Definitions

Object Oriented Programming

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

Main Features of OOP

* Abstraction * Encapsulation * Information Hiding * Relationships * polymorphism
* class * object

Class

* class is a user defined prototype that defines a set of attributes that characterize any object of the class.
  Ex: blueprint of a house

Object

* Object is an unique instance of a class. An object comprises both data members and methods.
  Ex: houses that built using a blueprint.

Abstraction

* An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perespective of the viewer.

Encapsulation

* Encapsulation is the process of grouping related attributed and methods togather, giving a name to the unit and providing an interface for outsiders to communicate with the unit.

Information Hiding

* Hide certain information or implementation decision that are internal to the encapsulation structure.

Steps of developing an OOP

* Analyse the problem
* Identify Objects that are needed to solve the problem.
* Identify classes through abstraction.
* Create objects from identified classes.
* Assemble objects to create the solution.

Structured programming vs OOP

Structured Programming	OOP
* Difficult to modify structured programs	* Easier to modify OOP
* Difficult to reuse code	* Easier to reuse code
* There are no access specifiers	* There are access specifiers such as private, public and protected.
* Data is not secured	* Data is secure.
* Difficult to develop complex programs	* Easier to develop complex programs.

Setters (Mutators)

Getters (Accessors)

Function Overloading

* Having multiple functions with the same name, but having different types of parameters.

Constructor

* Constructor is used to initialize the object when it is declared. It has no return type (not even void) and has the same name as the class.

→ Default Constructor

* Can be used to initialize attributes to default values.

→ Overloaded Constructor

* Can be used to assign values sent by the main program as arguments.

Destructor

* A destructor can be used to release memory of attributes that were created dynamically when the object was created. (~tilde)

Static Objects

* Rectangle R1;

Overload constructor

* Rectangle R2(100,50);

Accessing method.

* R1.setLength(100);

↑ dot operator

Dynamic Objects.

* Rectangle *r;
  r = new Rectangle(); } method 1

overloaded can be done here

* Rectangle *r = new Rectangle(); Method 2

Accessing Method.

* r-> setWidth(100);

↑ arrow operator

Types of Analysis Class

→ Entity Classes - Classes that we have identified using noun verb analysis.

→ Boundary Classes - Interaction Classes, Forms, and reports. (lists)

→ Control Classes - In a complex use case, the use case itself can be a class. Typically we can have one control class per complex use case.

Super Class (parent / ancestor)

* In an inheritance another class takes on the properties (derives) of an existing base class. That base class is called 'super class'

Sub Class (child / descendant)

* The class that derives attributes fom the superclass is called 'sub class'.
* The class that inherits from super class.

* super-class to sub-class → Specialization of super class
* sub-class to super-class → Generalization. of subclass

Association Class

* An association relationship between two classes that has been promoted as a class is called 'Association Class'.
* It has the responsibility for maintaining information pertaining to the association with the ownership class. ← has information related to the association.

Deciding Relationships.

* Is-A Inheritance
* Part of - part cannot exist without the whole Composition
  * Pages are part of a book
* Part of - part can exist without the whole Aggregation.
  * Students are Part of a Batch
* Has a - Not a direct part of a relationship Association
  * Customer has multiple orders.
* Uses - Depends on another because it uses it at some point Dependency

Virtual / Virtual functions.

* By defining the function that we are overriding as a virtual function, it enables dynamic binding where the overidden methods are called correctly at runtime.
* Virtual functions support dynamic polymorphism.

```
Virtual int area() {
            return 0;
};
```

Polymorphism.

* Polymorphism is allowing an entity such as a variable, a function, or an object to have more than one form.
* Overriding is a type of polymorphism.
* Polymorphism occurs wher there is a hierarchy of classes and they are related by inheritance

Abstract Class.

* Classes that (made to prevent) are restricted to create objects are called 'Abstract class'.
* we can create an abstract class by including at least one pure virtual method.

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

* Cannot create objects of an abstract class and only be used as a base class when declaring other classes.