OBJECT ORIENTED PROGRAMMING

Apex is a cloud-based Object-Oriented Programming, where each business logic should be implemented in the form of Classes and Objects.

Apex supports all the Object-Oriented Programming Principles.

CLASS

Class is a Prototype, which contains collection of members inside it, which includes variables, procedures, functions, properties and constructors.

OBJECT

Once the class has been defined with the required members, then to access the class members we need a reference / instance / key called as "Object".

Object is an entity which holds the values for the class members.

By using object, we can assign/retrieve the values from the class members.

A class can have one/more objects. Memory will be always allocated for the objects.

Each object contains its own memory location. Each object can have a separate amount of memory, as objects support dynamic memory allocation.

PILLARS OF OBJECT-ORIENTED PROGRAMMING

Object Oriented Programming provides the below 4 principles.

1. **Encapsulation**: Encapsulation allows us to group a set of related members together into a single entity.

Encapsulation can be achieved with the help of Classes and Interfaces.

2. **Abstraction**: Abstraction provides the user Interface to the users, but without providing the complexity of implementation.

By using Abstraction, we can achieve Data Hiding i.e. Security.

3. **Inheritance**: Inheritance allows us to acquire/access the features of one class into another class so that we can avoid the redundant code and make the code generic.

By using Inheritance, we can achieve the "Re-Usability".

4. **Polymorphism**: Polymorphism allows us to prepare the multiple functions / procedures with the same name, but with different signature.

We have 2 types of Polymorphism.

1. Compile Time Polymorphism / Early Binding / Static Binding:

|

----> Achieved by using "Overloading".

|

----> 2 Types of Overloading.

i. Method Overloading

ii. Constructor Overloading

When two or more methods in the same class have the same name but different parameters, it is called Overloading.

2. Runtime Polymorphism / Late Binding / Dynamic Binding:

|

----> Achieved by using "Overriding".

|

----> Method Overriding.

Overriding enables child class to provide different implementation a method that is already defined in its parent class. In overriding, methods or functions must have same name and the same signature.

Class: Class is an entity, which contains a set of business logic that can be referenced inside the application.

Classes are also called as "Controllers" which contains the business logic.

A Class can include one /more variables, procedures, functions, properties and constructors inside it.

Note: 1. We can create "n" number of apex classes inside the organization.

2. Each apex class code will get resides inside the file, with the extension ".apxc".

3. All the apex classes information will get resides inside the "Apex Class" object.

Syntax: [Access Specifier] [With Sharing / Without Sharing] Class <Class name>

```
{
        // Write the Business Logic
}
```

Note: Use the with sharing or without sharing keywords on a class to specify whether sharing rules must be enforced. By default, the Apex class will be executed in without sharing mode.

Access Specifier: Access Specifier defines the level of access of the class and its members. It can be applicable at both "Class" level, and "Member" level. i.e. class can have its own access specifier, and member can have its own access specifier.

Apex provides the below 4 access specifiers.

1. **Private**: Private members of the class can be accessible within the Class only. These can't be accessible from outside the class.

Private Access Specifier can be applicable for "Class Members" and for "Test Classes".

A normal business logic class, should be always defined as either "Public / Global".

Note: If the user doesn't provide any access specifier for the class member, then Apex will consider the member as "Private" by default.

2. **Public**: Public Members of the Class can be accessible within the Class and from any other Class inside the Organization. But these can't be accessible from outside of the Organization.

3. **Protected**: This Access Specifier has been defined especially for Inheritance feature.

Protected Access specifier can be applicable only for the class members, not for the classes.

Protected Members of the class, can be accessible within the class and from other related child classes. But it cannot be accessible from other independent classes inside the Organization.

4. **Global**: Global Members of the class can be accessible within the organization, and from outside of the Organization also from any third-party system, during Integration.

Note: All the Batch Classes, Schedule Classes, Webservices and API's should be defined as "Global".

WAYS TO DEFINE AN APEX CLASS

Salesforce provides the below 2 ways to define an Apex Class.

1. By using "Standard Navigation":

    Click on the "Setup" menu.

        1. Search for the option "Apex Classes" in the Quick Find box.
        2. Click on "New" button.
        3. Write the Apex Class Code inside the Editor.
        4. Click on "Save / Quick Save" button, to Save the Code.

Use Case: Define an Apex Class, to manage the Employee Details.

```
Public Class EmployeeHelper
{
    Public Integer employeeId, employeeAge;
    Public String employeeName, designation, contactNumber, emailAddress, department;
    Public Decimal currentCTC;
    Public Date birthDate, joiningDate, relievingDate;
    Public Boolean isActive;
}
```

2. By using "Developer Console".

    Click on "Setup" menu.

        1. Click on "Developer Console" link.
        2. Go to the "Developer Console" Editor.

3. Click on "File ---> New ---> Apex Class".

4. Enter the Apex Class Name inside the textbox provided by the dialog box.

5. Click on "OK" button.

6. Write the Class Code inside the Editor.

7. Click on "CTRL+S" to save the Code.

Use Case: Define an apex class, to manage the Projects Details. Define the required variables to store the Project Details.

```
public class ProjectsHelper
{
    Public Integer projectCode, teamSize;
    Public String projectTitle, clientName, contactNumber, location, emailAddress;
    Public Date startDate, endDate;
    Public Decimal budgetCost;
    Public Boolean isInProgress;
}
```

Use Case: Define an Apex Class, to manage the Patient Details. Define the required variables to store the Patient Details.

```
public class PatientHelper
{
    Public Integer patientID, patientAge;
    Public String patientName, location, contactNumber, emailAddress, doctorName;
    Public Date appointmentDate, visitingDate;
    Public Time appointmentTime;
    Public Decimal consultationFee;
    Public Boolean isVisited;
}
```

OBJECT

Once a class has been defined with the required members, we can access the class members with the help of an "Object".

Object is an instance of the class, which is used to assign/retrieve the values from the class members.

A Class can have one or more objects. Each object will hold the different set of values for the class members.

Objects support "Dynamic Memory Allocation", hence no memory wastage.

Syntax: <ClassName> <objectName> = new <ClassName>();

Ex: ProjectsHelper pHelper = new ProjectsHelper();

(OR)

ProjectsHelper pHelper;

pHelper = new ProjectsHelper();

Note: We can define multiple objects for the Same Class. But, object names should be always Unique.

Ex: ProjectsHelper pHelper = new ProjectsHelper();

ProjectsHelper pHelper1 = new ProjectsHelper();

Assigning Values: We can assign the values for the Class members with the help of the object as below.

Syntax: <objectName>.<memberName> = <Value>;

Ex:
```
pHelper.projectCode = 9002391;
pHelper.projectTitle = 'Pharma Automation.';
pHelper.clientName = 'Reddy Labs Inc.';
pHelper.budgetCost = 25000000;
```

Retrieving Values: We can retrieve the values from the class Members as below.

Syntax: <objectName>.<memberName>;

Ex:
```
System.debug('Project Code is.....: '+ pHelper.projectCode);
System.debug('Project Title is....: '+ pHelper.projectTitle);
System.debug('Client Name is......: '+ pHelper.clientName);
System.debug('Budget Cost is......: '+ pHelper.budgetCost);
```

Use Case: Write an apex program, to define a class to manage the Product details. Define the required variables to store the Product details and assign the values and print the values on the log file.

Class Code: (Developer Console)

```
public class ProductDetailsHelper
{
   Public Integer productCode, quantity;
   Public String productName, manufacturer, location, emailId, contactNumber;
```

```
    Public Date manufacturingDate, expiryDate;
    Public Decimal unitPrice, discountedPrice;
    Public Boolean isInStock;
}
```

Execution: (Execute Anonymous Window)

```
// Creating the Object of the Class.
        ProductDetailsHelper pHelper = new ProductDetailsHelper();

// Print the Product Details.
        System.debug('Product Details are.....: '+ pHelper);

// Assigning the Values.
        pHelper.productCode = 15004511;
        pHelper.productName = 'Desktop Device';
        pHelper.quantity = 250;
        pHelper.unitPrice = 75000;
        pHelper.discountedPrice = 71000;
        pHelper.manufacturer = 'DELL Inc.';
        pHelper.location = 'India';
        pHelper.contactNumber = '+1 (800) 898-3321';
        pHelper.emailID = 'sales@dell.com';
        pHelper.manufacturingDate = System.today();
                //Date.newInstance(year, month, day)
        pHelper.expiryDate = System.today().AddMonths(60);
        pHelper.isInStock = true;
// Print the Values on the Log File..
        System.debug('Product Code is....: '+ pHelper.productCode);
        System.debug('Product Name is....: '+ pHelper.productName);
        System.debug('Unit Price is....: '+ pHelper.unitPrice);
        System.debug('Manufacturer Name is...: '+ pHelper.manufacturer);
        System.debug('Contact Number is.....: '+ pHelper.contactNumber);
        System.debug('Expiry Date is......: '+ pHelper.expiryDate.format());
        System.debug('Is Available in Stock? ....: '+ pHelper.isInStock);
```