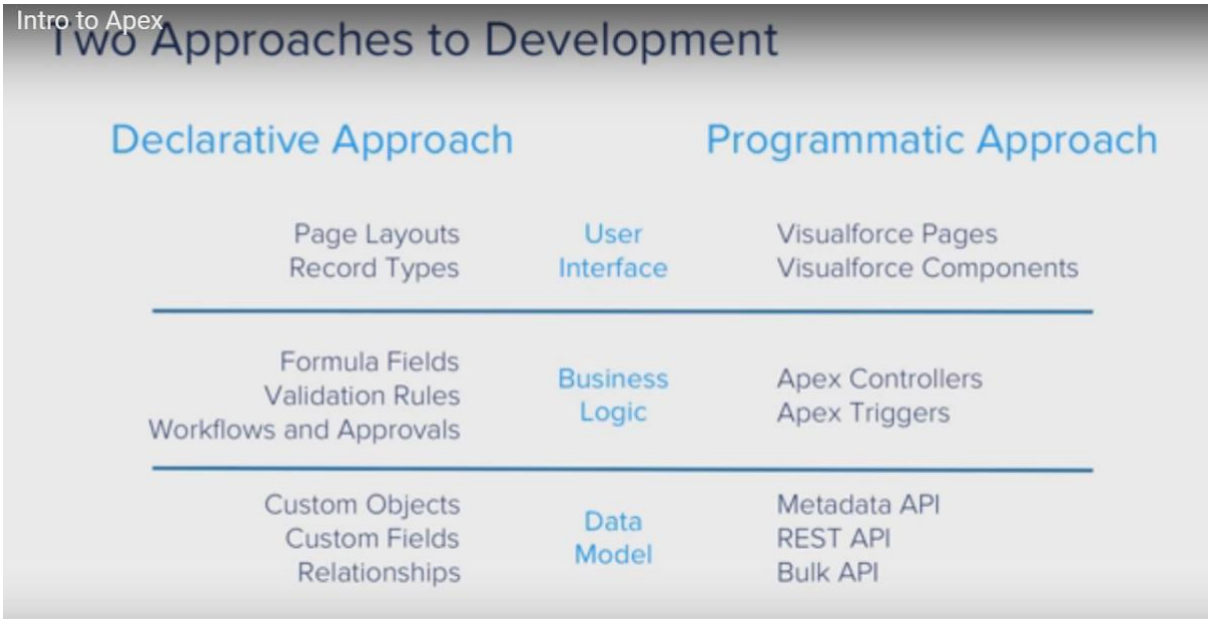
Brahmastra Cloud Guru

Salesforce Developer Material

Chapter#1 | Introduction to OOPL Technologies

Model-View-Controller

Salesforce Development Architecture is designed by MVC.
We have two approaches: Declarative & Programmatic Approach



Object Oriented Programing OOPs

Unlike procedural programing, here in the OOP model, Programs are organized around OBJECTs and DATA rather than action and logic.

- Main OOPs Principles are:

1.Encapsulation:

The wrapping up of DATA and METHODs

Hides data and behavior that is not necessary to the user.

Eg:- Class is an example for encapsulation.

2.Inheritance: It **creates new classes from existing classes**, so that the new classes will **acquire** all features of the existing classes

It is a concept of re-usability.

It is a concept acquiring the properties of one class by another .

We can only inherit those class which are defined virtual.

```
public virtual class ClassName{
}
```

If a class want to inherit another class then we use extends keyword

```
class Example extends Demo{
}
```

==> When a class extends another class, all the properties of the parent class can be access by child class.

==> Private members of the parent class are not accessible to the child class.

==> Whenever we are creating a object for the child class.

1. First Parent Class Constructor will run
2. Child Class Constructor will run

Example:

```
public virtual class Parent_Example {
        public Parent_Example(){
                System.debug('Parent _Example');
        }
}

public class Child_Example extends Parent_Example{
        public Child_Example(){
                System.debug('Iam a child Class');
        }
}
```

Execute : Child_Example e =new Child_Example();

O/P: Parent_Example

: Iam a child class.

Types of inheritence :

1.Single Inheritence

```
class Example extends Demo{
}
```

2. Multiple Inheritence

Class Example extends Demo, AccountWrapper{

}

3. Multi Level Inheritence

4. Hybrid Inheritence

3.Polymorphism: **"Having multiple forms"** - To allow an entity such as a variable, a function, or an object to have more than one form.

4.Abstract: Through the process of abstraction, a programmer **hides all irrelevant data about object** to reduce complexity and **increase efficiency**.

An abstract class has at least one abstract method. **An abstract method will not have any code in the base class**; the code will be added in its derived classes.

5.Class:

Class is a template definition of the methods and variables.

6.Object: (Class Instance/Class Object)

It refers to a particular instance(single occurence) of a class.

It is a instance of a class.

Syntax:

ClassName varaiable = new ClassName(parameters) ;

new: new is a keyword

It is used to allocate the memory for data members of the class.

Constructor: It will be invoked automatically once the memory is allocated

It is used to initialize the data members of the class.

variable: It is a reference variable.

: It is used to reffer to the memory allocated by the new keyword.

: This variable is called as object.

Example : object.VariableName;

object.method();

c. All the members class are reffered using object

Note : Using the object we can access only global and public members of class

7.Constructors:

Special method that initializes an object of Class type.

Has the same name of the class

Set the values of the members of an object

8.Overloading:

Creating **methods with same name** but different parameters.

It is a concept of defining more than one method, in the same class with same name ,with different no of parameters and different order of parameters, different type of parameters then we call it as method overloading.

```
public class Example{
    public void show(){
        System.debug('Name:Srinivas');
        System.debug('Age :29);
    }
    public void show(String name){
        System.debug(name);
    }
    public void show(Integer age){
        System.debug(age);
    }
    public void show(Integer age,String name){
        System.debug('Name:'+name);
        System.debug('Age:'+age);
    }
    public void show(String name,Integer age){
        System.debug('Name:'+name);
        System.debug('Age:'+age);
```

```
        }
    }
    OverLoading_Example ol=new OverLoading_Example();
ol.show();
ol.show('Kiran');
ol.show(40);
ol.show(40,'Prasad');
```

9.Overriding: **Re-defining body of a method** of superclass in a subclass to change behavior of a method.

1. If child class want to override the method in the parent class then we use overridding
2. Method in the parent class should be defined as virtual.
3. Method in the child class should be defined as override.
4. When ever there is a conflict between parent and child class then child class will dominate the parent class .

Example ;

```
global virtual class Parent_Example {
    public Integer age ;
    protected Decimal salary;
        global String name ;
    private String city;
    global Parent_Example(){
    System.debug('Parent _Example');
    age=40;
        salary=50000;
        name='Satish';
    city='Hyd';
    }
    public virtual void show(){
    System.debug('Age :'+age);
    System.debug('Name:'+name);
    System.debug('Salary'+salary);
    System.debug('City:'+city);
    }
    }
```

Child :

```
public class Child_Example extends Parent_Example{
        public String phone;
    public String industry;
    public Child_Example(){
    System.debug('Iam a child Class');
    System.debug(age);
    System.debug(name);
    System.debug(salary);
    phone='040-1111';
    industry='Banking';
      }
 public override void show(){
   System.debug('Age :'+age);
   System.debug('Name:'+name);
   System.debug('Salary'+salary);
   System.debug('Phone:'+phone);
   System.debug('Industry:'+industry);

    }
    }
```

10.Arrays: A series of objects all of which are the **same size and type**. Each object in an array is called an array element.

It is a collection of simillar elements .

Memory will be allocated sequentially.

Elements in the array are reffered using index .

Index value starts with zero.

Syntax :

```
DataType[] arrayName =new DataType[size];
```

Example :

```
Integer[] ages =new Integer[4];
Decimal[] salaries=new Decimal[30];
String[] names =new String[40];
Account[] accounts =new Account[4];
Contact[] cons=new Contact[10];
Student_Wrapper[] stds =new Student_Wrapper[3];
```

Static Initialization of values .

Example :

Integer[] ages =new Integer[]{10,20,30,40};

Drawbacks of an array :

1. Size of an array is fixed .
2. We cannot increase or decrease the size based on run time requirement.

Examples:

=======

Create array of Opportunities

Create array of City names and display the cities

Create a array of Accounts and display the data

Create array of Accounts and display the data

```
Account[] accounts =new Account[4];
for(Integer i=0;i<accounts.size();i++){
    System.debug('Name :'+accounts[i].Name);
    System.debug('Phone:'+accounts[i].Phone);
    System.debug('Industry:'+accounts[i].industry);
    }
```

11.Collections: is a **group of data** manipulate as a single object. Corresponds to a bag.

Array	Collections
1.Array is a collection of Homogeneous (Similar) elements	1.It is a collection of Homogeneous &Heterogeneous elements
2.Arrays can not grow and shrink dynamically	2.It can grow and shrink dynamically
3.Arrays can be accessed faster and less memory	3.Collections are slow compared to arrays &consume more memory

3 types of Collections, List, Set and Map.

Chapter#2 | Introduction to Apex

About Apex

Apex is an Object-oriented, on-demand programming language.

- Object-Oriented
- On-Demand: Functionality that you need available in the cloud
- Java-like
- Built on the Force.com platform

Benefits of Apex

- Secure,proven, and trusted, running natively on Salesforce servers.
- Your data structures already exists, so you can reference the same sObjects and fields created through the declarative Setup menu.
- It has a built-in framework for testing & deployment.

Apex Vs. Java

Commonalities:-

- Classes, Inheritance, Polymorphism, and other common OOP features.
- similar syntax and notation
- compiled, strongly-typed, and transactional.

Differences:-

- Runs on multi-tenant environment and is controlled by governor limits.
- Case-insensitive
- On-Demand and is compiled and executed **in the Cloud**
- Not general purpose programming language
- Requires **unit testing** for deployment into a production environment.

Invoking Apex

Implicit (within Salesforce Cloud)	Explicit (from outside Salesforce Cloud/external)
1.Triggers	1.Saving records via API (Dataloader)
2.Saving a record via UI	2.Sending an email (Email Services)

3.Interacting with a Visualforce page.	3.Submitting an anonymous block via API.
4.Invoking an Apex Web Service	4.Web Service client makes a call.
5.Creating a Job via Apex Scheduler.	

Development & Execution Environments

Developer Edition	
Enterprise Edition	Enterprise Edition Sandbox
Unlimited Edition	Unlimited Edition Sandbox
Force.com Free Edition	Force.com Free Edition Sandbox

Apex Class and Apex Trigger

All Apex code is saved as one of two things:

Class: An Apex library of attributes and methods that can be instantiated into an object.

Trigger: An Apex script that executes before or after a specific DML event on a particular Salesforce object.

And Interface also.

Tools

- Browser based Tools:
  - Setup→Build→Develop→Apex Classes
  - Developer Console
- Force.com IDE (Plug-in to Eclipse)

Use Cases

- Field Updates on other Records {Workflow Rule Field Update à Apex Trigger}

- Automate record sharing based on dynamic data-driven record criteria.
- Callout to External Web Service
- Custom Web Service
- Automate Record Creation/Deletion
- Cleanse/Repair Existing Records
- Custom Email Handler
- Custom Button Action
- Complex Validation Rule

Apex Class and Apex Trigger

All Apex code is saved as one of two things:

- ✓ Class: An Apex library of attributes and methods that can be instantiated into an object.
- ✓ Trigger: An Apex script that executes before or after a specific DML event on a particular Salesforce object.
- ✓ And Interface also.

Apex Classes

Apex Classes:

- are Blueprints used to create objects in code
- It is collection of data members and methods
- Concept of class is derived from Encapsulation
- Contain characteristic attributes and action-performing methods.
- Support implicit and explicit constructors

Syntax:

Class ClassName [implements InterfaceNameList | (none)] [extends ClassName | (None)] {

//Class body

}

Class Access Modifiers

Apex supports 3 access modifiers for classes:

Private: Classes cannot be referenced by any Apex code outside of the class in which they are defined.

Public: These can be called within the same org

Global: These classes can be called outside of the org.

Use cases : Classes Modifiers

- Global:
  - Batch Apex
  - Inbound Email Services
  - Web Services
  - Scheduled Classes
  - Accessible classes in managed packages.
- Private:
  - Test Classes
  - Inner Classes
- Public:
  - Everything Else

Class Declaration Keywords

- Data in the database is accessed in one of the to contexts, depending on the situation:

❖ **System Context:**

✓ Implemented using the "without sharing" keyword.

✓ Allows the code **access to all records** in the database for all objects even if the user executing the code **does not have access** to that data.

✓ **Ignores the sharing model**

Ex:- public without sharing class seeAllDataClass{}

❖ **User Context:**

✓ Implemented using the "with sharing" keyword

✓ Limits the code to only **access** the records that the user executing the code normally has access to.

✓ **Respects the sharing model**

Example:

Public with sharing class seeRestrictedDataClass{

}

Access Modifiers : Class Members

Class Members:

Variables

Methods

Syntax of a variable :

accessmodifier DataType variableName [=value];

public Integer age ;

global String name;

String city ;

protected Decimal value;

private String colgName;

Note : Accessmodifiers for the variables are defined only when we define variables in class or interface.

Function:

1. It is a set of instructions running together to perform a particular task.
2. Fuctions are classified into four types
   a. Method without parameters and without ReturnTypes
   b. Method with parameters and without ReturnTypes
   c. Method without parameters and with ReturnTypes
   d. Method with parameters and with ReturnTypes.
3. Method without returnType and without parameters
   a. syntax:

```
void methodName(){
        logic
}
```

Ex:

1.Create a method calculate which will take internal marks and external marks as parameters if total is more or equals to 40 print pass ,otherwise fail

2.Create a method calculate which will take amount and years as input parameters calculate the interest with rate as 12%

static :

1. It is a keyword
2. It can be used along with variables and methods
3. Static Variables :
    a. If you define any variable as static ,memory will be allocated only once .
    b. Memory will be allocated on the name of class.
    c. Static variables will be reffered using className .
    d. Memory for the static variables is allocated at the time of loading a class.

Example :

```
public class Static_Example {
    public static Integer aval=10;
    public integer bval=0;
    public void add(){
  Integer aval=aval+10;
  Integer bval=bval+10;
 }
}
```

Apex supports 4 Access Modifiers for Class members:

- Global
- Public
- Protected
- Private

These will specify scope and visability of the variables ,methods,classes and interfaces .

private:

a. If you define any varible,method or class as private ,they can be accessed only with in the scope of the class where they are defined .

b. Private members cannot called outside the class .

c. If you dont define any access modifier ,default modifier is private.

protected:

a. If you define any variable ,method as protected ,they can be accessed with in the class in which they are defined .
b. Protected members can also be accessed with in the child classes or inner classes.

public:

a. If you define any variable ,method ,class ,interface as public they can be accessed with in the class in which they are defined and any where with in the application.

global:

a. If you define any variable ,method ,class ,interface as global ,they can be accessed with in the class, any where in the application and outside the application.

b. If you want to define webservice classes then they should global

Note: A class **cannot** have a more **restrictive access** modifier than one of its members or attributes.

Governor Limits

- Governor Limits are **runtime limits enforced by the Apex runtime engine.**

Governor Limits can:

- Monitor and Manage platform resources such as **Memory, Database resources**, etc.
- **Enable multitenancy** by ensuring that resources are available for all tenants on the platform
- Can issue program-terminating **runtime exceptions** when limits are exceeded.
- Are typically **reset per transaction**
- Are **Subject to change** from release to release.

Governor Limits Examples

Govenor Limit Description	Limit
1.SOQL Queries per Class	100
2.SOQL Queries per Batch Apex & Future Methods	200
3.Records retrieved by SOQL queries	50,000
4.Records retrieved by Database.getQueryLocator	10,000

5.SOSL queries per class	20
6.Records retrieved by a Single SOSL query	200
7.DML statements per class	150
8.Records processed by DML statements	10,000
9.No.ot executed statements	2,00,000
10.No.of executed statements per Batch Apex & Future methods	10,00,000
11.Total heap size	6MB
12. Webservices	100
13. Future Methods	50
14. Queueable	50
15. Schedulable	100

Chapter#3 | Data Types

Data Types Overview

- Identifiers:- **Name** given to a Class, Interface, Method or Variable
- Variable: It is the name given to a **temporary memory location**
- Datatypes: Amount of memory, what type of data and Values range
- Categories:-
    - **Primitive**
    - **sObject**
    - **Collections**
    - **Enum**
    - **User-defined Apex classes**
    - **System-supplied Apex classes**

Primitive Data Types

- Boolean

- Date, DateTime, Time
- ID
- Integer, Long, Double, Decimal
- String
- **Object** (Can be used as the base type for any other data type.)

sObject:

Sobject standard for salesforce object.

It refers to of object in the database of the application

It can be standard object or custom object.

Every object in the database of the application will have equilent pre-defined apex class

Create instance of a sobject

Example :

```
Account acc =new Account();
acc.Name='Wipro';
acc.Phone='040-1234';
acc.Industry='Banking';
acc.Rating='Hot';
```

```
Opportunity op2=new Opportunity();
    op.Name='Admin';
    op.CloseDate=System.today();
    op.StageName='Closed Won';
    op.Amount=50000;
    op.Type='New Customer';
    op.LeadSource='Web';
```

Note : SObject can be created in two other formats

```
    Account acc =new Account(Name='Wipro',Phone='040-123');
    Opportunity op=new Opportunity(Name='Testing',StageName='Closed Won');
```

```
    Account acc =new Account();
```

```
acc.put('Name','Wipro');
acc.Put('Phone','040-1111');
acc.put('Rating','Hot')
```

Putting & Getting Values

- Set a field on an sObject

  S.put('**Name**','Salesforce.com');

- Access a field on an sObject

  Object fieldvalue=s.get('**Name**');

Enum – Datatype

- Enum is an abstract data type that stores one value of a finite set of specified identifiers.

Syntax:-

Public **enum** Season {WINTER, SPRING, SUMMER, FALL}

Season e=Season.WINTER

- By creating this 'enum', you've created a new data type called 'Season' that can be used as any other data type.

Wrapper Class:

It is a user-defined datatype.

It is also an apex class,

It contains only data members.

Wrapper class doesn't contain any methods.

Example

```
public class ClassName{
    datamembers
}
```

Example :

```
public class Student{
    public String name;
    public String city;
    public String phone;
```

```
}
Student std;
```

Example : create a wrapper class Employee with name ,salary ,exp

```
public class Employee {
    public String name;
    public Decimal salary;
    public Decimal exp;
}
```

Example : Create a wrapper class Invoice with invoiceNo, invoiceDate, quantity, price

```
public class Invoice {
    public String invoiceNo;
    public Date invoiceDate;
    public Integer quantity;
    public Decimal price;
}
```