Session 1
Introduction to Apex

Apex is Multitenant on demand programming language for developers for building next generation business applications.

Apex is strongly typed Object oriented language that allows developers to execute flow & transaction control statements on salesforce servers in conjunction with call to API.

Apex programming language is world's first cloud-based object oriented programming language.

Each business logic should be implemented in terms of "Classes and Objects".

It can be used to implement complex validation rules or complex transactional flow.

Apex is tightly coupled programming language. Tightly coupled programming language means, each and every variable must be declared before using it.

By using Apex, we can perform all DML (INSERT, UPDATE, DELETE, UNDELETE, UPSERT) manipulations.

Apex programming is tightly integrated with Sales force objects.

By using Apex programming, we can retrieve records from Salesforce objects with the help of "SOQL" queries.

By using Apex programming, we can search for content inside Salesforce instance with the help of "SOSL".

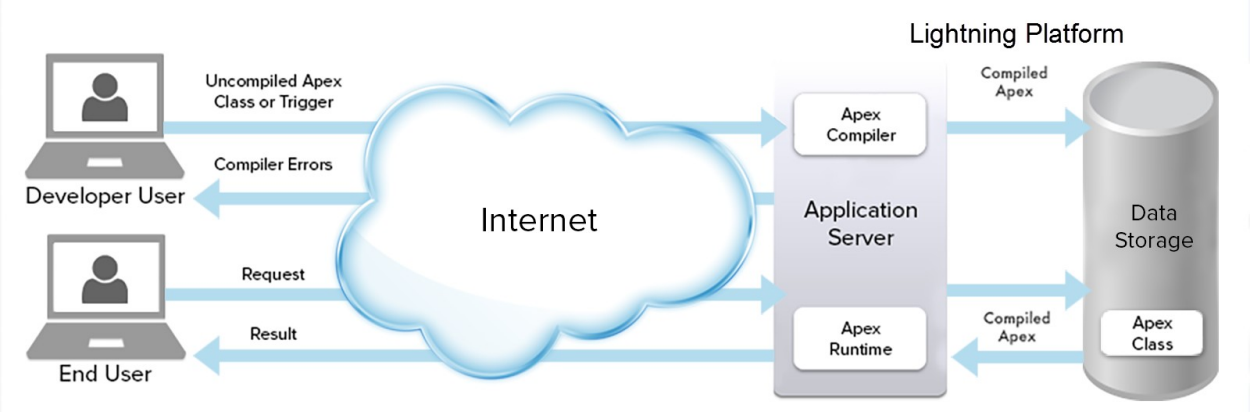
By using Apex programming, we can integrate Sales force application with external system with the help of REST API and SOAP API.

Each Apex code is compiled and executed by application server inside Force.com platform.

For each Apex program execution, a separate debug log file is generated. Using which, we can track progress of our Apex code.

When object is created, Sales force creates two classes. One is with same name as object name for DML operation, and other is with object name postfixed with "__share" for sharing records programatically.

Compilation and execution of Apex Program:



Apex is included in Unlimited Edition, Developer Edition, Enterprise Edition, and Database.com

Every application is a three-tier application. ie. application has three layers:

1. FrontEnd
2. Business Logic
3. BackEnd

Force.com provides following features:
1. Apex: It is object oriented programming language, which is used build business logic for the application.

2. Visualforce: It is a web development framework, which is used to design attractive and dynamic user interface.

3. Integration(REST API / SOAP API): It is used to integrate Salesforce with any external system.

Types of Customizations in Salesforce:

1. Declarative Customizations: It can be done using point and click. ie, we can implement the features by using mouse pointer.

FrontEnd:

a. Page Layout
b. Tabs
c. Record Types

Business Logic:

a. Validation Rule
b. Formula fields
c. Workflow Rules, Assignment Rules, Sharing Rules
d. Approval Process, Process Builders

BackEnd:

a. Objects
b. Fields
c. Association (Relationship)

2. Programmatic Customizations: It requires programming language to be used to implement additinal functionality.

FrontEnd: Visualforce
Business Logic: Apex
BackEnd: Objects, Fields, Association (Relationship)

As a language, Apex is:

Integrated

Apex provides built-in support for common Lightning Platform idioms, including:

Data manipulation language (DML) calls, such as `INSERT`, `UPDATE`, and `DELETE`, that include built-in `DmlException` handling

Inline Salesforce Object Query Language (SOQL) and Salesforce Object Search Language (SOSL) queries that return lists of sObject records

Looping that allows for bulk processing of multiple records at a time

Locking syntax that prevents record update conflicts

Custom public API calls that can be built from stored Apex methods

Warnings and errors issued when a user tries to edit or delete a custom object or field that is referenced by Apex

Easy to use

Apex is based on familiar Java idioms, such as variable and expression syntax, block and conditional statement syntax, loop syntax, object and array notation.

Where Apex introduces new elements, it uses syntax and semantics that are easy to understand and encourage efficient use of the Lightning Platform. Therefore, Apex produces code that is both succinct and easy to write.

Data focused

Apex is designed to thread together multiple query and DML statements into a single unit of work on the Salesforce server. Developers use database stored procedures to thread together multiple transaction statements on a database server in a similar way. Like other database stored procedures, Apex does not attempt to provide general support for rendering elements in the user interface.

Rigorous

Apex is a strongly typed language that uses direct references to schema objects such as object and field names. It fails quickly at compile time if any references are invalid. It stores all custom field, object, and class dependencies in metadata to ensure that they are not deleted while required by active Apex code.

Hosted

Apex is interpreted, executed, and controlled entirely by the Lightning Platform.

Multitenant aware

Like the rest of the Lightning Platform, Apex runs in a multitenant environment. So, the Apex runtime engine is designed to guard closely against runaway code, preventing it from monopolizing shared resources. Any code that violates limits fails with easy-to-understand error messages.

Easy to test

Apex provides built-in support for unit test creation and execution. It includes test results that indicate how much code is covered, and which parts of your code could be more efficient. Salesforce ensures that all custom Apex code works as expected by executing all unit tests prior to any platform upgrades.

Versioned

You can save your Apex code against different versions of the API. This enables you to maintain behavior.

When to use Apex:
1)when you want to do some automation that cannot be implemented using workflows or process builders in that case we need to write apex code or basically apex triggers
2)when ever you want to do some complex validation rules like..
3)To create web services to interact with external application or if we want to create email services also we need to write apex code.
4)To do some transactions and want to control those transactions using some save points & rollbacks we can use apex.

We cannot implement using Apex
1) We cannot show anything on UI except error meassage
2)We cannot change any salesforce functionality we can only add new functionalities and stop the execution of the exisiting functionality.
3) It is not used to create a temporatory file,so no file handling
4) We cannot create multiple threads using salesforce so threading is not possible.

Features of Apex:
1)Apex is not case sensitive
2)It upgrades auomatically
3)It is object oriented programming language like java synatax
4) Its very easy to test since it provides built in supported
5)Which runs on multi-tenant environment i,e no need to worry about own infrastructure and maintains.

Datatypes:

1) As Apex is strictly types each variable should be declared first, before using it.

2) Datatype describes two things:
1. What type of data the variable can hold.
2. To store the data, how much memory needs to get allocated.

1**. Integer :** It is used to store whole number only.
It allocates 4 bytes of memory.
It can store the 32 bit number ranging from "-2,147,483,648" to "2,147,483,647".

Ex:
Integer i = 1;

2. Long:
- It is used to store whole number only.
- It allocates 8 bytes of memory.
- It can store the 64 bit number ranging from "-2^63" to "2^63-1".

Ex:

Long l = 2147483648L;

3. Decimal:
- It is used to store floating point numbers.
- It allocates 8 bytes of memory.
- "Number", "Currency" and "Percent" fields are referenced to Decimal datatype.

4. Double:
- It is used to store bigger floating point numbers along with more digits after decimal point.
- A 64-bit number that includes a decimal point.
- Doubles have a minimum value of -2^63 and a maximum value of 2^63-1 including decimal point.

Ex:
Double d=3.14159;

5. String:
- Any set of characters surrounded by single quotes.
- Each caracter occupied 2 bytes of memory.
- "Textfield", "Text Area", "Text Area Long", "Email", "Phone", "URL", "Text Encrypted", "Picklist", "Multiselect Picklist" fields are referenced to "String" datatype.

Ex:
String s = 'The quick brown fox jumped over the lazy dog.';

6. Boolean:
- A value that can only be assigned true, false, or null.
- It occupies 1 bit of memory.
- "Checkbox" field is referenced to "Boolean" datatype.

Ex:
Boolean isWinner = true;

7. Date:

- A value that indicates a particular day.
- It occupies 8 bytes of memory.
- Always create date values with a system static method.
- You can add or subtract an Integer value from a Date value, returning a Date value.
- Addition and subtraction of Integer values are the only arithmetic functions that work with Date values.
- Each "Date" field is referenced to "Date" datatype.

8. DateTime:

- A value that indicates a particular day and time, such as a timestamp.
- Always create datetime values with a system static method.
- You can add or subtract an Integer or Double value from a Datetime value, returning a Date value.
- Addition and subtraction of Integer and Double values are the only arithmetic functions that work with Datetime values.

9. Blob:

- A collection of binary data stored as a single object.
- You can convert this data type to String or from String using the toString and valueOf methods, respectively.
- Blobs can be accepted as Web service arguments,
- stored in a document (the body of a document is a Blob), or sent as attachments.
- It is used to store images, pictures, documents etc.
- Blob datatype will be able to store max 2 GB of content.

10.ID:

- Any valid 18-character Lightning Platform record identifier.
- Each object's "ID" field is referenced to "ID" datatype.
- It is case sensitive datatype.
- "Lookup" and "Master-Detail" relationship fields are referenced to "ID" datatype.

Ex:

ID id='00300000003T2PGAA0';

b. sObject Type:

An sObject is any object that can be stored in the Force.com platform database. These are not objects in the sense of instances of Apex classes; rather, they are representations of data that has or will be persisted.
These persisted objects can be treated as first class citizens within the Apex language, which makes the database integration particularly intuitive and easy to use.

- This datatype can be used to store entire record of an object.
- For every object, Salesforce creates a class with same name as object name.
- Datatype would be of object class name type.
- Ex: Account acc, Contact con, Position__c pos etc.

For primitive datatype, there is static memory allocation, and memory is allocated at "Stack" area.

For sObject datatype, there is dynamic memory allocation, and memory is allocated at "Heap" area.

Size of sObject datatype depends on number of members assigned a value.

Example:

```
Account a = new Account();
MyCustomObject__c co = new MyCustomObject__c();
```

Default Value in Salesforce:

Default value of any variable declared, is "NULL". Because every datatype in Salesforce is a "class".

if you declare a variable and don't initialize it with a value, it will be `null`. In essence, `null` means the absence of a value. You can also assign `null` to any variable declared with a primitive type. For example, both of these statements result in a variable set to `null`:

```
Boolean x = null;
Decimal d;
```

Reserved Words:

abstract	Activate	And	Any,	Asc	Autonomous	Begin	Bigdecimal	Bloc	Boolean
Break	Bulk	Byte	Case	Cast	Catch	Char	Class	Collect	Commit
Const	Continue	Currency	Date	Datetime	Decimal	Default	Delete	Desc	Do
Double	Else	End	Enum	Exception	Exit	Export	Extends	False	Final
Finally	Float	For	From	Global	Goto	Group	Having	Hint	If
Implements	Import	In	Inner	Insert	Instanceof	Int	Integer	Interface	Into
Join	Like	Limit	List	Long	Loop	Map	Merge	New	Not
Null	Number	Object	Of	On	Or	Outer	Override	Package	Parallet
Pragma	Private	Protected	Public	Retrieve	Return	Rollback	Select	Set	Short
Sobject	Sort	Static	String	Super	Switch	Synchronized	System	Thestmethod	Then
This	Throw	Time	Transaction	Trigger	True	Try	Undelete	Update	Upsert
Using	Virtual	Void	Webservice	When	Where	while			

Operators :

1) Arithmetic Operators:

1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)

5. Modulus (Math.mod())

Note: If "Plus(+)" is used with String datatype, then result will be concatination of Strings.

2) Relational / Comparison Operators:

1. Less Than (<)
2. Less Than or Equal (<=)
3. Greater Than (>)
4. Greater Than or Equal (>=)
5. Equal (==)
6. Not Equal (!=)

Note: If we compare two strings using "Equal (==)" operator, "Text Comparison" will be done. It compares both strings by ignoring case sensitivity. If we want to compare both strings, including case sensitivity, "equals()" method would be used which compares both strings with "Binary Comparison".

3) Assignment Operators:

1. +=
2. -=
3. *=
4. /=

4) Increment / Decrement Operators:

1. Increment (++) - It increments the value by 1.
2. Decrement (--) - It decrements the value by 1.

5) Logial Operators:

1. AND (&&)
2. OR (||)
3. NOT (!)

Operator Precedence

Operators are interpreted in order, according to rules.

Apex uses the following operator precedence rules:

Precedence	Operators	Description
1	`{} () ++ --`	Grouping and prefix increments and decrements
2	`! -x +x (type) new`	Unary negation, type cast and object creation
3	`* /`	Multiplication and division
4	`+ -`	Addition and subtraction
5	`< <= > >= instanceOf`	Greater-than and less-than comparisons, reference tests
6	`== !=`	Comparisons: equal and not-equal
7	`&&`	Logical AND

Precedence	Operators	Description
8	||	Logical OR
9	= += -= *= /= &=	Assignment operators

Printing Statement in Apex:

For each request, Salesforce generates a "Debug Log File" which describes the execution process of application.

For debugging the code, Salesforce provides a function, "System.debug()" which is used to write a message in debug log file.

Ways to write Apex Code:

1. Standard Navigation:
Setup -> Build -> Develop -> Apex Classes -> New -> Coding -> Save

2. Developer Console:

Note: Developer Console provides auto-save option, which will send the code to Force.com platform for each fraction of 10 seconds.

3. Anonymous Window:

Note: The code written in annonymous window, will not be saved in "Metadata Repository".

4. Eclipse IDE

5. Aside.io

Example 1:

```
public class HelloWorld {
    public static void sayYou()
    {
        System.debug('You');
    }
    public void sayMe(){
        System.debug('Me');
    }

}
```

HelloWorld obj=new HelloWorld();
obj.sayMe();
HelloWorld.sayYou();

Example 2:

```
public class BasicExample {
   private Integer id1;
   public String name;

   public void setID(Integer id2)
   {

      id1=id2;

   }
   public Integer display(){
      return id1;
   }

}


BasicExample obj=new BasicExample();
//obj.id1=20;
obj.setID(10);
System.debug(obj.display());
```

Enums

Use enumerations (enums) to specify a set of constants. Define a new enumeration by using the enum keyword followed by the list of identifiers between curly braces.

Each value in the enumeration corresponds to an Integer value, starting from zero and incrementing by one from left to right. Because each value corresponds to a constant, the identifiers are in upper case.

For example, this example defines an enumeration called Season that contains the four seasons:
public enum Season {WINTER, SPRING, SUMMER, FALL}
the Integer value of WINTER is 0, SPRING 1, SUMMER 2, FALL 3.

Once you define your enumeration, you can use the new enum type as a data type for declaring variables. The following example uses the Season enum type that is defined first and creates a variable s of type Season . It then checks the value of the s variable and writes a different debug output based on its value.

```
public enum Season {WINTER, SPRING, SUMMER, FALL}
Season s = Season.SUMMER;
if (s == Season.SUMMER) {
// Will write the string value SUMMER
System.debug(s);
} else {
```

```
System.debug('Not summer.');
}
```

Comments:
Comments are lines of text that you add to your code to describe what it does. Comments aren’t executable code. It’s good practice to annotate your code with comments as necessary.
This makes the code easier to understand and more maintainable. Apex has two forms of comments. The first uses the // token to mark everything on the same line to the right of the token as a comment. The second encloses a block of text, possibly across multiple lines, between the /* and */ tokens.

```
System.debug ('I will execute');
// This comment is ignored.
/*
I am a large comment, completely ignored as well.
*/
```