SUB PROGRAMS

Upon implementing the business logic as part of the application requirement, we have to divide the business logic into the various small pieces called as "Sub Programs".

Sub Program contains one / more statements, used to perform certain operations inside the application.

By using sub programs, we can make the code re-usable, and avoid the redundant code inside the application. Hence, the application performance can be improved.

Apex provides 2 types of Sub Programs.

1. **Procedure**: Procedure contains one or more statements, used to perform the operations inside the application.

These are Fire & Forget feature, will execute the specified business logic, but will not return any value back to the calling environment.

2. **Function**: Functions are like "Request-Reply" feature, which contains one or more statements inside it, performs certain operation inside the application.

Once the function executes, it returns the value back to the calling environment.

Note: Sub Programs should be defined always inside the class. A class can have "n" number of procedures and functions inside it.

PROCEDURES

Procedure contains one or more statements, to be used to perform the operations inside the application.

Apex provides 2 types of Procedures.

1. Non-Parameterized Procedures.

2. Parameterized Procedures.

Non-Parameterized Procedures:

In these procedures, the Procedure does not expect input values from the end user. Upon invoking the procedure, it will execute the business logic automatically.

Note: Procedure won't return any value back to the calling environment. Hence the procedure return type should be always "void".

Syntax: [Access Specifier] void <procedureName> ()

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

Ex:

```
Public Class AccountsHelper
{
        Public void InsertAccounts ()
        {
                // Write the Business Logic
        }
```

```
        Public void DeleteInActiveAccounts()
        {
                // Write the Business Logic
        }
        Public void GetAllAccounts()
        {
                // Write the Business Logic
        }
    }
```

INVOKING PROCEDURE

Once the procedure is ready, it can be invoked by creating the object of the class.

Syntax: <objectName>.<procedureName>();

Ex:
```
AccountsHelper accHelper = new AccountsHelper();
accHelper.InsertAccounts();
accHelper.DeleteInActiveAccounts();
accHelper.GetAllAccounts();
```

Use Case: Write an Apex program, to insert an account record inside the object.

Class Code: (Developer Console)

```
public class AccountsHelper
{
        Public void InsertAccountRecord()
   {
        Account acc = new Account();
        acc.Name = 'Sudheer Kumar';
        acc.Rating = 'Hot';
        acc.Industry = 'Education';
        acc.AnnualRevenue = 3100000;
        acc.Type = 'Prospect';
        acc.Ownership = 'Public';
        acc.Website = 'www.gmail.com';
        acc.Phone = '9900445577';
        acc.Fax = '9900444444';
        acc.BillingCity = 'Bangalore';
        acc.BillingState = 'Karnataka';
```

```
        acc.BillingCountry = 'India';
        acc.CustomerPriority__c = 'High';
        acc.Active__c = 'Yes';
        Insert acc;
        if(acc.Id != Null)
        System.debug('Account Record Inserted Successfully with the Id ...: '+ acc.Id);
    }
}
```

Execution: (Execute Anonymous Window)

```
// Create the object of the Class
AccountsHelper accHelper = new AccountsHelper();
// Invoke the Procedure
accHelper.InsertAccountRecord();
```

Parameterized Procedures:

Parameterized procedure takes the input values from the end users, so that it will execute the business Logic and will perform the operations inside the application.

Using parameterized procedures, we can make the functionality dynamic. A Procedure can have "n" number of Parameters.

Note: Upon invoking the Procedure, we have to supply the values at runtime.

Syntax: [Access Specifier] void <ProcedureName>(<DataType> <param1>, <DataType> <param2>, <DataType> <param3>.. <DataType> <param n>)

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

Ex:

```
Public Class CommonHelper
        {
                Public void Addition (Integer fNumber, Integer sNumber)
                {
                        // Write the Business Logic.
                }

                Public void Multiply (Integer fNumber, Integer sNumber, Integer tNumber)
```

```
        {
                // Write the Business Logic
        }

        Public void Check Equals (String str1, String str2)
        {
                // Write the Business Logic.
        }

        Public void GetAccountDetails(Id accRecordID)
        {
                // Write the Business Logic...
        }
    }
```

Invoking the Procedure

Once the procedure is ready with the required input parameters, we can invoke the procedure by using the below syntax.

Syntax: <objectName>. <procedureName>(<value1>, <value2>,<valueN>);

Note: While invoking the procedure, we have to supply the required input values in the same order in which they have defined.

Ex:

```
CommonHelper cHelper = new CommonHelper();
cHelper.Addition(3400, 560);
cHelper.Multiply(3400, 14, 19);
cHelper.CheckEquals('Welcome','Hyderabad');
```

Use Case: Write an apex program, to perform the Mathematical Operations using Procedures.

Class Code:

```
public class MathOperationsHelper
{
    Public void Addition(Integer fNumber, Integer sNumber)
  {
     System.debug('Addition Result is....: '+ (fNumber + sNumber));
  }
  Public void Multiply (Integer fNumber, Integer sNumber, Integer tNumber)
  {
```

```
        System.debug('Multiplication Result is....: '+ (fNumber * sNumber * tNumber));
    }
    Public void CheckLargestValue(Integer fNumber, Integer sNumber)
    {
        if(fNumber == sNumber)
            System.debug('Both the Numbers are Identical.');
        else if(fNumber > sNumber)
            System.debug('First Number is the Biggest One.');
        Else
            System.debug('Second Number is the Biggest One.');
    }
}
```

Execution:

```
        // Create the Object of the Class.
        MathOperationsHelper mHelper = new MathOperationsHelper();
        // Invoke the Procedure.
        mHelper.Addition(45000, 1945);
        mHelper.CheckLargestValue(194500, 194500);
```

Use Case: Write an apex program, to Insert a Hiring Manager Record inside the Object, through Parameterized Procedures.

Class Code:

```
public class HiringManagerHelper
{
        Public void CreateHRRecord(String hrName, String hrCity, String hrPhone, String hrEmail, String hrTitle, Decimal salary)
    {
        Hiring_Manager__C hrRecord = new Hiring_Manager__C();
            hrRecord.Name = hrName;
            hrRecord.Contact_Number__c = hrPhone;
            hrRecord.Email_Address__c = hrEmail;
            hrRecord.Designation__c = hrTitle;
            hrRecord.Location__c = hrCity;
            hrRecord.Current_CTC__c = salary;
        Insert hrRecord;
```

```
        if(hrRecord.Id != Null)
            System.debug('Hiring Manager Record Inserted Successfully. '+ hrRecord.Id);
    }
}
```

Execution:

```
        HiringManagerHelper hrHelper = new HiringManagerHelper();
        hrHelper.CreateHRRecord('PraveenRathode','Bangalore','9900334477','praveen@gmail.com','Recruitment Specialist', 1200000);
```

Static Members: While implementing the business logic inside the class, we can define "Static Members" also (i.e., Static Variables, Procedures, Functions, Properties).

Static Members doesn't require object creation.

Syntax: <ClassName>.<variableName>;

<ClassName>.<ProcedureName>();

Note: 1. A Class can have both non-static procedures and static procedures.

2. We can implement the chaining mechanism between the procedures inside the class.

3. We can call a static procedure from a non-static procedure but a non-static procedure cannot be invoked from static procedure.

Use Case: Write an apex program, to insert a Hiring Manager record, and a related position record inside the Object.

1. Use different Procedures to implement the Business Logic.

2. Implement Chaining Mechanism between the Procedures.

Class Code:

```
public class HiringManagerUtility
{
        Public static void InsertHRRecord()
    {
        Hiring_Manager__C hrRecord = new Hiring_Manager__C();
        hrRecord.Name = 'Sampath Kumar';
        hrRecord.Location__c = 'Mumbai';
        hrRecord.Contact_Number__c = '9900334466';
        hrRecord.Email_Address__c = 'sampath@gmail.com';
        hrRecord.Designation__c = 'Recruitment Specialist';
        hrRecord.Current_CTC__c = 2100000;
      Insert hrRecord;
```

```
    if(hrRecord.Id != Null)
    {
        System.debug('Hiring Manager Record Id is....: '+ hrRecord.Id);
          // Invoke the Procedure
        InsertRelatedPosition(hrRecord);
    }
}
Public static void InsertRelatedPosition(Hiring_Manager__C hrRecord)
{
    if(hrRecord.Id != Null)
    {
        // Write the Code to Insert the Related Position Record.
                Position__c pos = new Position__C();
                        pos.Name = 'SFDC Project Manager';
                        pos.Location__c = hrRecord.Location__c;
                        pos.Position_Status__c = 'Open Approved';
                        pos.Number_of_Vacancies__c = 2;
                        pos.Open_Date__c = System.today();
                        pos.Milestone_Date__c = System.today().AddMonths(1);
                        pos.Minimum_Budget__c = 1400000;
                        pos.Maximum_Budget__c = 2000000;
                        pos.HR_Contact_Number__c = hrRecord.Contact_Number__c;
                        pos.HR_Email_ID__c = hrRecord.Email_Address__c;
                        pos.Travel_Required__c = true;
                        pos.Passport_Required__c = true;
                        pos.Position_Description__c = 'Required 10+ years of experience in
Salesforce Project Management.';
                        pos.Skills_Required__c = 'Required 10+ years of Experience in SFDC
Automation, Apex, Visualforce, Lightning, LWC.';
                pos.Hiring_Manager__c = hrRecord.Id;
                Insert pos;
                if(pos.Id != Null)
               System.debug('Position Record Id is....: '+ pos.id);
```

```
    }
  }
}
```

Execution

```
_HiringManagerUtility.InsertHRRecord();
```

FUNCTIONS

Functions are Request-Reply feature which contains one or more statements to perform the operations inside the application.

Once the operation is done, it will return the result back to the calling environment.

Function can return any type of value i.e., either Primitive / SObject/ Collection / Apex Type/ User Defined Type.

Function returns the value by using a "Return" statement.

Syntax: [Access Specifier] [Static] <ReturnType> <FunctionName>(<Parameters>)

```
    {
        // Write the Business Logic
            ....
            ....
        Return <Value>;
    }
```

Ex:

```
    Public Class StringOperationsHelper
    {
        Public static String Concatenate (String s1, String s2)
        {
            return s1 + s2;
        }

        Public static Boolean CheckEquals(String str1, String str2)
        {
            if(str1 == str2)
                return true;
            else
                return false;
        }
    }
```

Invoking the Functions

Upon invoking the function, it will return the value back to the calling environment. The value should be stored inside a variable.

Syntax: <DataType> <variableName> =<objectName>.<FunctionName>(<Values>);

Ex: String result = StringOperationsHelper.Concatenate('Welcome', 'India');

System.debug('Concatenation Result is....: '+ result);

Use Case: Write an apex program, to perform the Math Operations and String Operations.

Class Code

```
public class CommonHandler
{
        Public static Integer Addition(Integer fNumber, Integer sNumber)
   {
      return (fNumber + sNumber);
   }
  Public static string CheckBiggestNumber (Integer fNumber, Integer sNumber)
   {
      if(fNumber == sNumber)
         return 'Both the Numbers are Identical';
      else if(fNumber > sNumber)
         return 'First Number is the Biggest One.';
      else
         return 'Second Number is the Biggest One.';
   }
Public static Integer GetStringLength(String message)
   {
          return message.length();
   }
 Public static ID  CreateHiringManagerRecord(String hrName, String hrCity, String
hrPhone, String hrEmail, String hrtitle, Decimal hrSalary )
   {
      Hiring_Manager__C hrRecord = new Hiring_Manager__c();
         hrRecord.Name = hrName;
         hrRecord.Location__c = hrCity;
         hrRecord.Contact_Number__c = hrPhone;
```

```
        hrRecord.Email_Address__c = hrEmail;
        hrRecord.Designation__c = hrTitle;
        hrRecord.Current_CTC__c = hrSalary;
    Insert hrRecord;
    return hrRecord.Id;
  }
}
```

Execution:

```
Integer additionResult = CommonHandler.Addition(24000, 9400);
System.debug('Addition Result is : '+ additionResult);
System.debug('Operation Result is....: '+ CommonHandler.CheckBiggestNumber(34000, 34000));
System.debug('Number of Characters Count is...: '+ CommonHandler.GetStringLength('Welcome to Parameterized Functions.'));
Id recordId = CommonHandler.CreateHiringManagerRecord('Srikanth Kumar', 'Pune','9944550099','srikanth@gmail.com','HR Executive', 100000);
if(recordID != Null)
System.debug('Hiring Manager Record Inserted Successfully. '+ recordID);
```