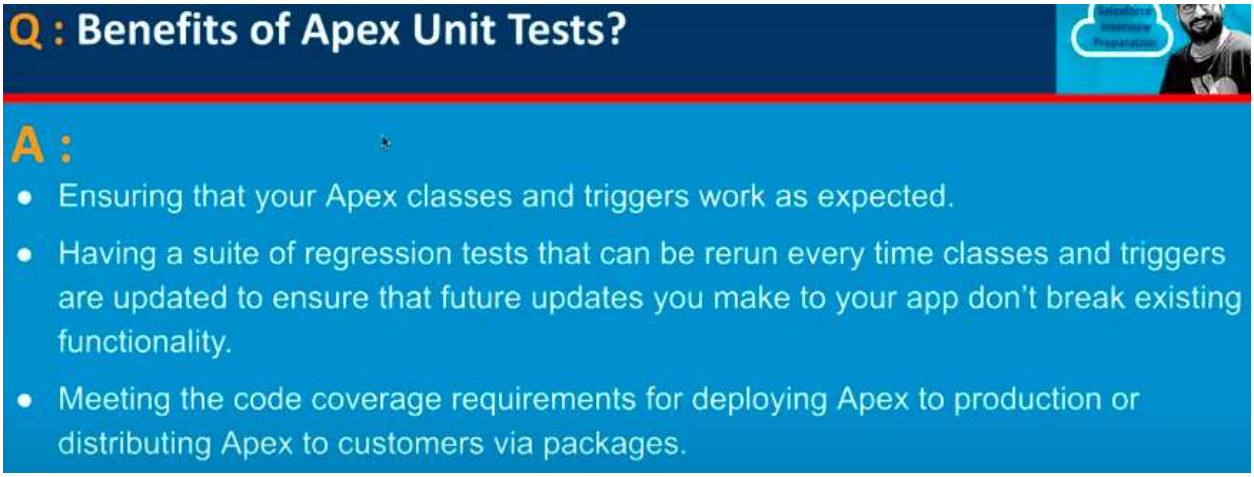
1.


Q : What is Apex Test Class?
A :
The Apex testing framework enables you to write and execute tests for your Apex classes and triggers on the Lightning Platform.
Apex unit tests ensure high quality for your Apex code and let you meet requirements for deploying Apex.
Apex code can only be written in a sandbox environment or a Developer org, not in production.
Apex unit tests are required for deploying and distributing Apex.

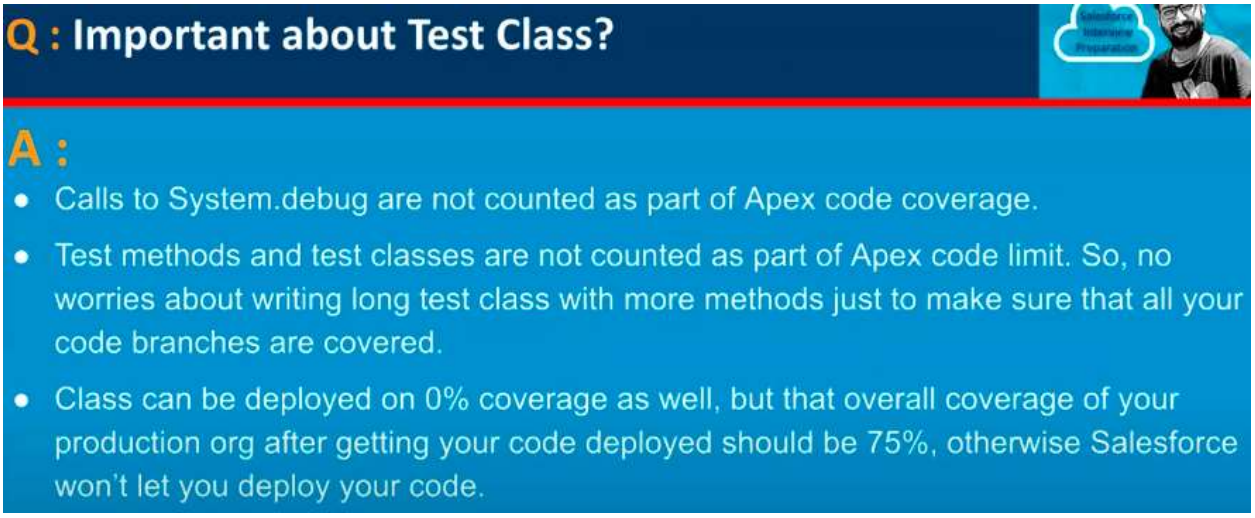
2.


Q : Benefits of Apex Unit Tests?
A :
Ensuring that your Apex classes and triggers work as expected.
Having a suite of regression tests that can be rerun every time classes and triggers are updated to ensure that future updates you make to your app don't break existing functionality.
Meeting the code coverage requirements for deploying Apex to production or distributing Apex to customers via packages.

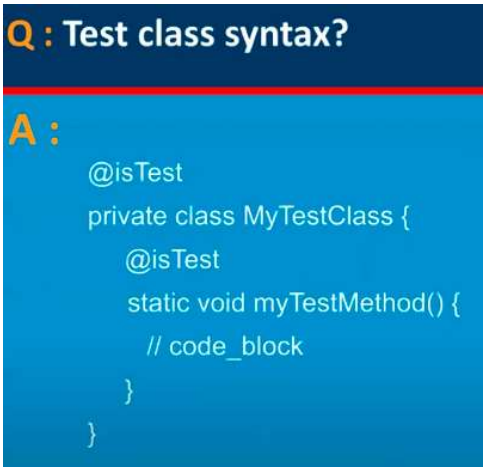
3.


Q : Code Coverage requirement for Deployment?
A :
• To deploy code or package it for the Lightning Platform AppExchange, at least 75% of Apex code must be covered by tests, and all those tests must pass.
• In addition, each trigger must have some coverage. Even though code coverage is a requirement for deployment, don't write tests only to meet this requirement.
• Make sure to test the common use cases in your app, including positive and negative test cases, and bulk and single-record processing.

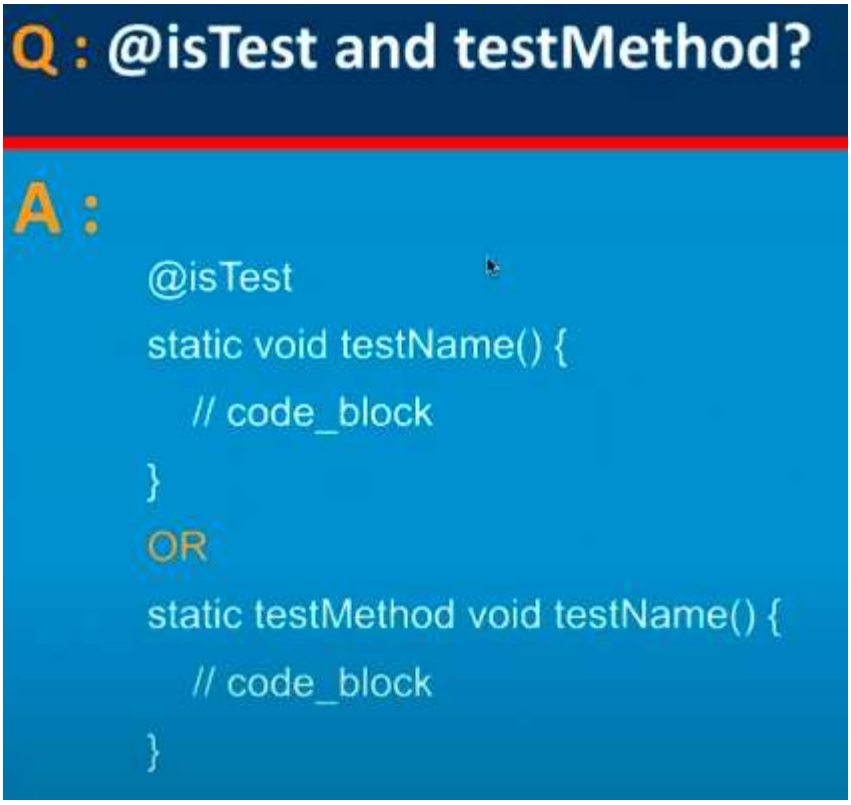
4.


Q : Important about Test Class?
A :
• Calls to System.debug are not counted as part of Apex code coverage.
• Test methods and test classes are not counted as part of Apex code limit. So, no worries about writing long test class with more methods just to make sure that all your code branches are covered.
• Class can be deployed on 0% coverage as well, but that overall coverage of your production org after getting your code deployed should be 75%, otherwise Salesforce won't let you deploy your code.

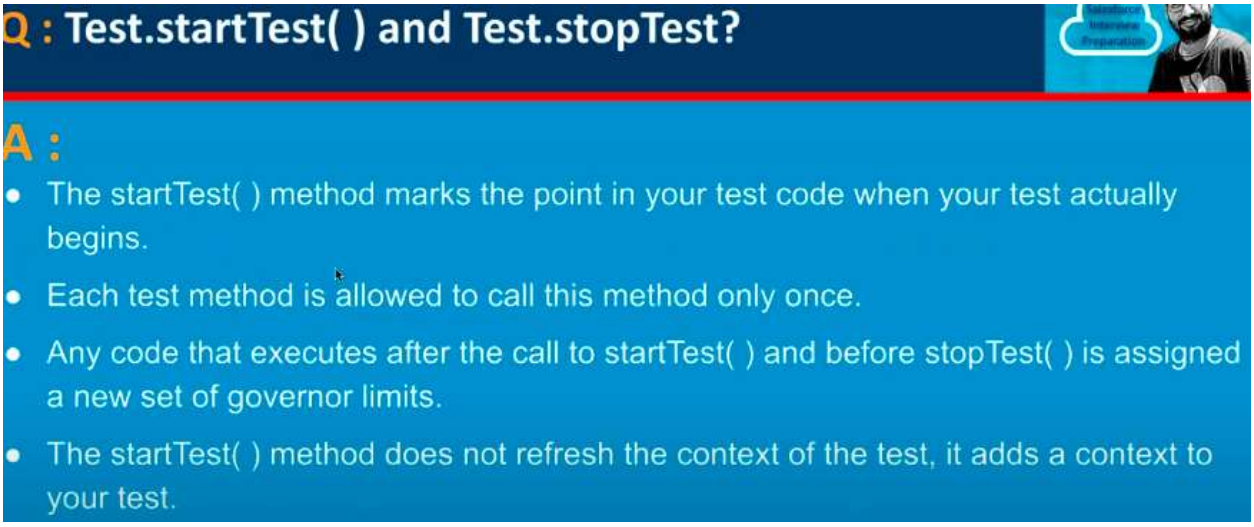
5.


Q : Test class syntax?
A :
@isTest
private class MyTestClass {
@isTest
static void myTestMethod() {
// code_block
}
}

6.


Q : @isTest and testMethod?
A :
@isTest
static void testName() {
// code_block
}
OR
static testMethod void testName() {
// code_block
}

7.


Q : Test.startTest( ) and Test.stopTest?
A :
• The startTest( ) method marks the point in your test code when your test actually begins.
• Each test method is allowed to call this method only once.
• Any code that executes after the call to startTest( ) and before stopTest( ) is assigned a new set of governor limits.
• The startTest( ) method does not refresh the context of the test, it adds a context to your test.

- For Example: if your class makes 98 SOQL queries before it calls startTest( ), and the first significant statement after startTest is a DML statement, the program can now make an additional 100 queries. Once stopTest( ) is called, however, the program goes back into the original context, and can only make 2 additional SOQL queries before reaching the limit of 100.
- All asynchronous calls made after the startTest method are collected by the system. When stopTest is executed, all asynchronous processes are run synchronously.

8.

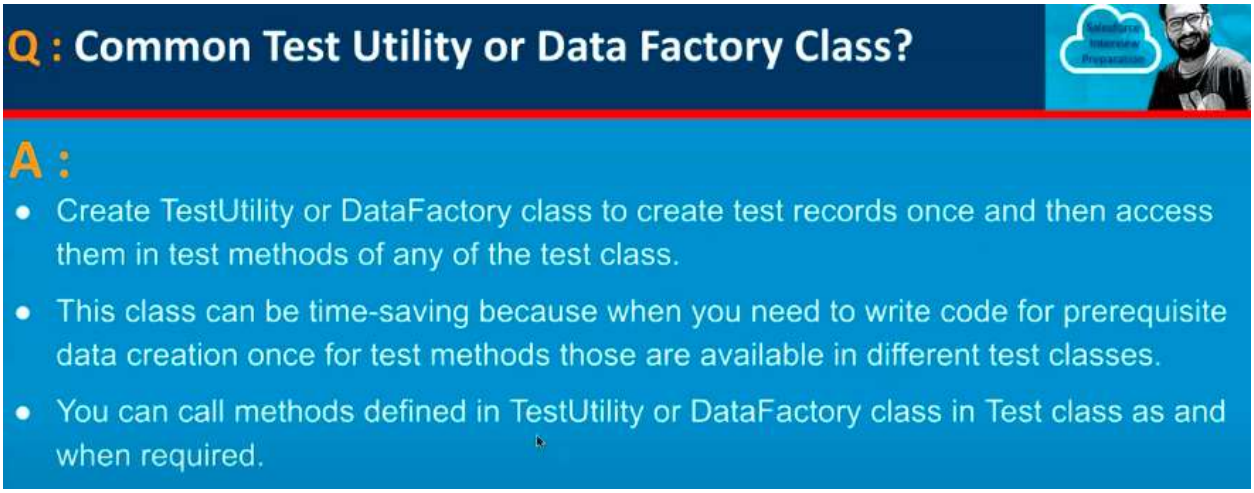
Q : @testSetup?

A :

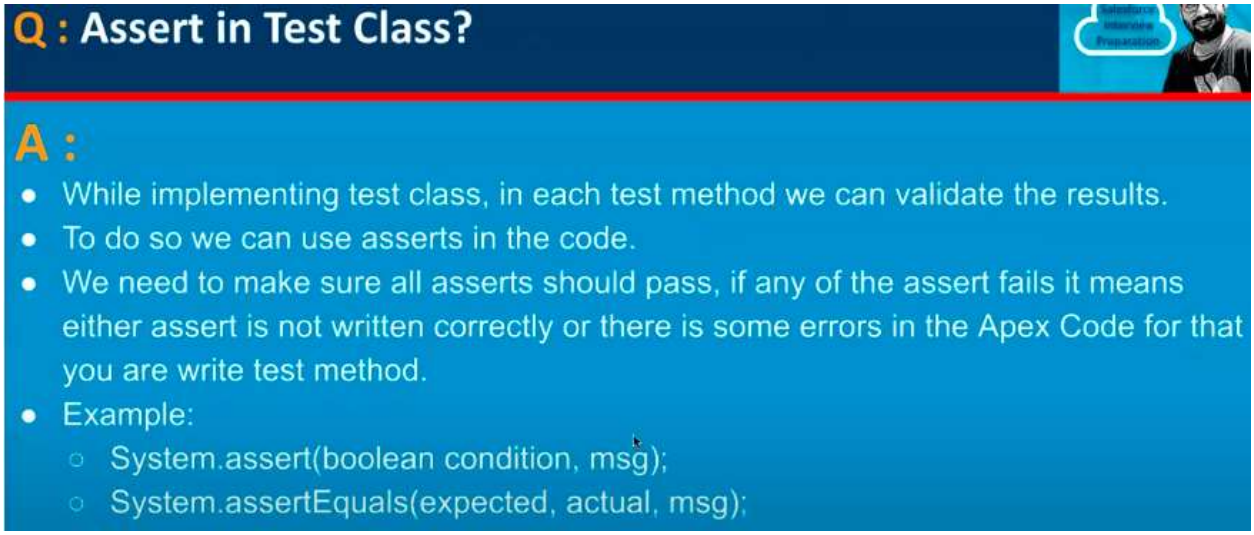
- Use test setup methods (methods that are annotated with @testSetup) to create test records once and then access them in every test method in the test class.
- Test setup methods can be time-saving when you need to create reference or prerequisite data for all test methods, or a common set of records that all test methods operate on.
- If a test class contains a test setup method, the testing framework executes the test setup method first, before any test method in the class.

- Records that are created in a test setup method are available to all test methods in the test class and are rolled back at the end of test class execution.
- If a test method changes those records, such as record field updates or record deletions, those changes are rolled back after each test method finishes execution. The next executing test method gets access to the original unmodified state of those records.

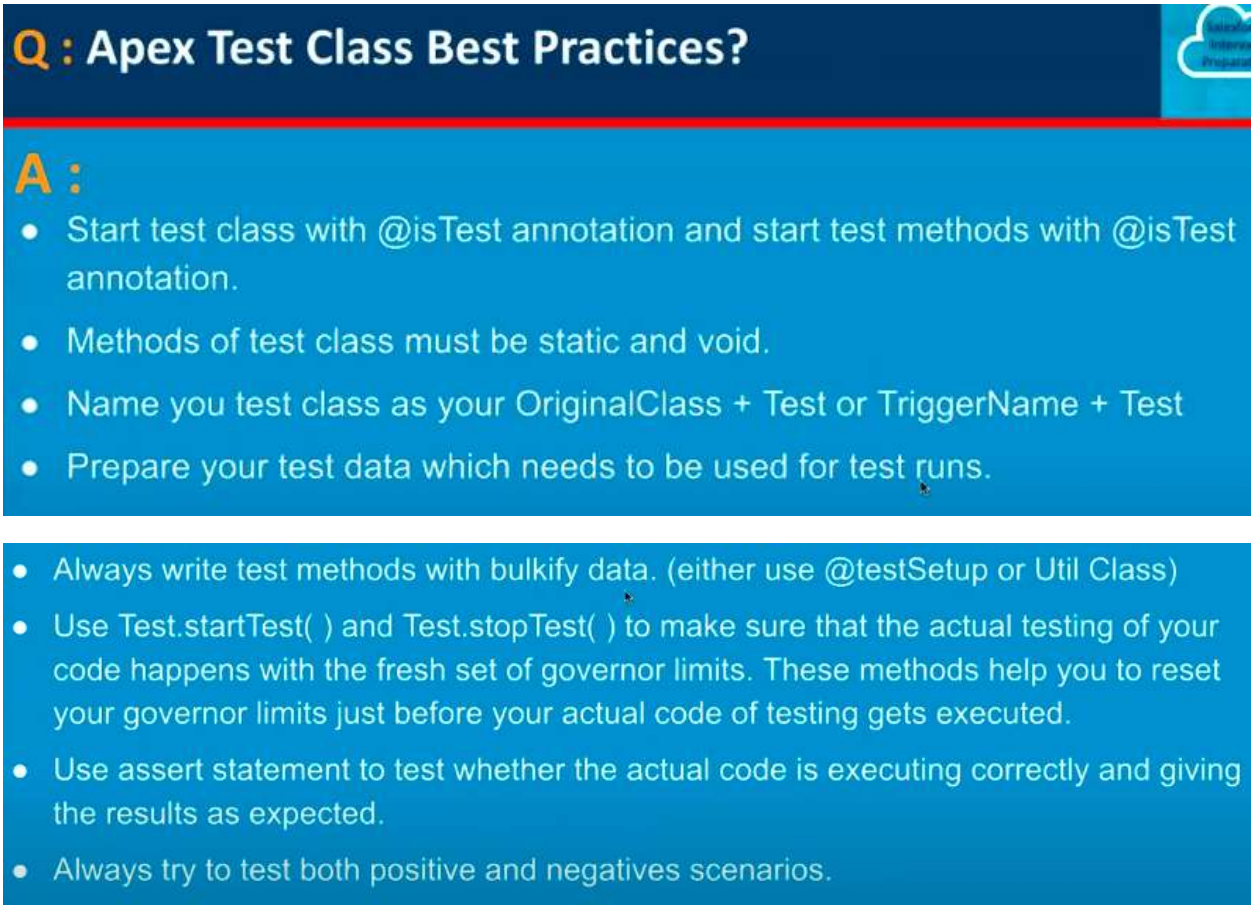
9.


Q : Common Test Utility or Data Factory Class?
A :
• Create TestUtility or DataFactory class to create test records once and then access them in test methods of any of the test class.
• This class can be time-saving because when you need to write code for prerequisite data creation once for test methods those are available in different test classes.
• You can call methods defined in TestUtility or DataFactory class in Test class as and when required.

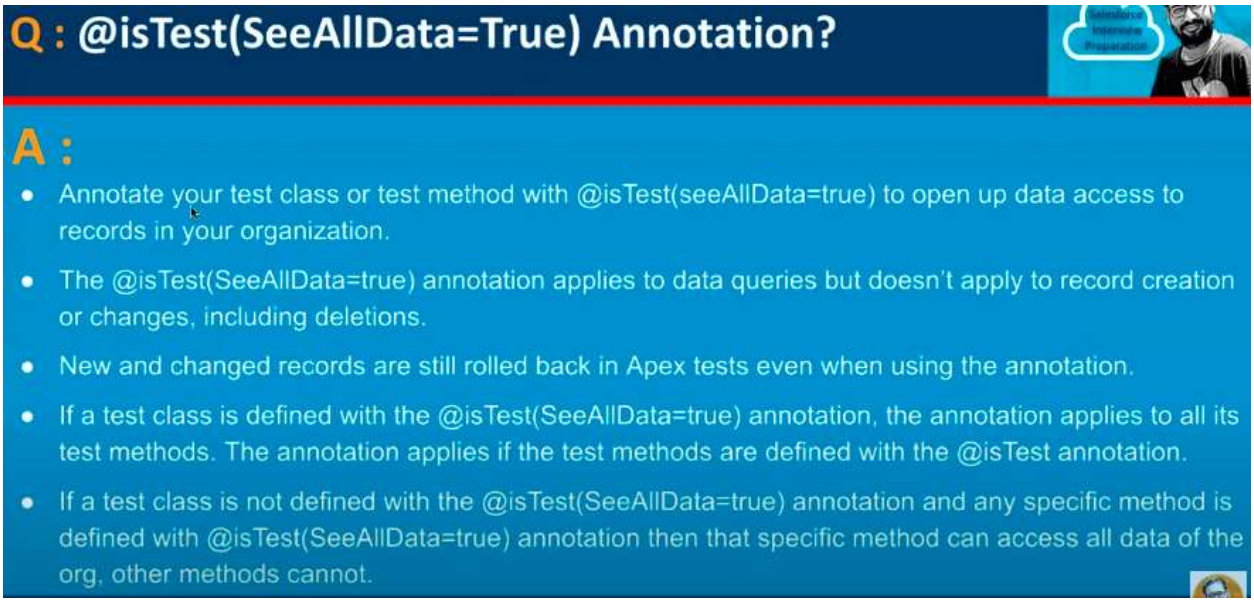
10.


Q : Assert in Test Class?
A :
• While implementing test class, in each test method we can validate the results.
• To do so we can use asserts in the code.
• We need to make sure all asserts should pass, if any of the assert fails it means either assert is not written correctly or there is some errors in the Apex Code for that you are write test method.
• Example:
○ System.assert(boolean condition, msg);
○ System.assertEquals(expected, actual, msg);

11.


Q : Apex Test Class Best Practices?
A :
Start test class with @isTest annotation and start test methods with @isTest annotation.
Methods of test class must be static and void.
Name you test class as your OriginalClass + Test or TriggerName + Test
Prepare your test data which needs to be used for test runs.
Always write test methods with bulkify data. (either use @testSetup or Util Class)
Use Test.startTest( ) and Test.stopTest( ) to make sure that the actual testing of your code happens with the fresh set of governor limits. These methods help you to reset your governor limits just before your actual code of testing gets executed.
Use assert statement to test whether the actual code is executing correctly and giving the results as expected.
Always try to test both positive and negatives scenarios.

12.


Q : @isTest(SeeAllData=True) Annotation?
A :
Annotate your test class or test method with @isTest(seeAllData=true) to open up data access to records in your organization.
The @isTest(SeeAllData=true) annotation applies to data queries but doesn't apply to record creation or changes, including deletions.
New and changed records are still rolled back in Apex tests even when using the annotation.
If a test class is defined with the @isTest(SeeAllData=true) annotation, the annotation applies to all its test methods. The annotation applies if the test methods are defined with the @isTest annotation.
If a test class is not defined with the @isTest(SeeAllData=true) annotation and any specific method is defined with @isTest(SeeAllData=true) annotation then that specific method can access all data of the org, other methods cannot.

13.

Q : Using the runAs Method?

A :

- Usually, all Apex code runs in system mode, where the permissions and record sharing of the current user aren't taken into account.
- The system method runAs enables you to write test methods that change the user context to an existing user or a new user so that the user's record sharing is enforced.
- The runAs method doesn't enforce user permissions or field-level permissions, only record sharing.
- You can use runAs only in the test methods. The original system context is started again after all runAs test methods complete.
- The runAs method ignores user license limits. You can create new users with runAs even if your organization has not additional user licenses.

- Example:

```
Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
User u = new User( initialize all required fields here);
System.runAs(u){
    //some code
}
```