**Q:** **Can a workflow rule be triggered from a field update event happening in an approval process?**

**A:** You have to select the "Re-evaluate Workflow Rules after Field Change" box in order for the Approval process to trigger the workflow.

An approval process can specify a field update action that re-evaluates workflow rules for the updated object. If, however, the re-evaluated workflow rules include a cross-object field update, those cross-object field updates will be ignored.

**Q:** **A process builder is set to re-evaluate the record, which make it iterate multiple times. Does the process builder iterate indefinitely or stop after a few iterations?**

A: The process will iterate for a stack depth of 5. Which means 5 times the process will iterate and then it will halt. So, the chain will continue for 5 times. If infinite loop will happen then the org will hit governor Limits. And in order to avoid such conditions, Salesforce has kept the stack depth of 5 and post that process will halt.

**Q: We have an approval process on a custom object, when the approver rejects the record, that should be automatically deleted. How to achieve this?**

**A:** Create a custom checkbox on that custom object and then Create a Approval Process, as soon as the approver rejects the record then do a field(checkbox) update. After that create a **Autolauched (No trigger) flow.** Now use this flow to delete records based on that checkbox value.

So, in this way as soon as that record is rejected by approver, field update will happen. Once the field is updated, that field will be picked by Flow and then that record will be deleted automatically.

**Q:** **Why do we prefer custom object over custom settings?**

**A:** No triggers on Custom Settings but can have triggers on custom Objects.

Custom Settings are stored in the Cache Memory whereas Custom Objects are stored in Force.com Database.

Custom Settings will have only limited Data types, but custom Objects have large variety of data types.

Custom Settings cannot have Relationships like MD or Lookup but custom objects can have.

Custom Settings cannot have its own custom tab in app but a custom object can have its custom tab in app.

No formula fields or RollUp summary in Custom Settings. But in custom Objects we have.

**Q: can I call a future method from batch class?**

A: Future method and Batch apex both are Asynchronous apex. So, we cannot call a async method from another async methods. So, we cannot directly call a future method from batch class But there can be a indirect or substitute way.

1st: Call the future method from finish() of batch class. Since, finish() is used for post processing logics. We can call a future method from finish().

2nd: Call a webservice method from Batch class, which in return will call a future method.

**Q: write a test class for batch class?**

**A:** The only catch here is to use **@testSetup** to create test records and  **Test.startTest() and Test.stopTest()**. This is used to make all the async calls between this block of code as sync in test context.

@isTest

Public class batchTest {

@testSetup

Public static void setup(){

/\*Create Records here\*/

}

Private static testMethod void(){

/\*Some logic\*/

Create a user and user System.runAs

Test.startTest()

Call the Batch Apex class

Assert statements

Test.stopTest()

}

}

**Q: OOPs Concepts in Salesforce?**

**A:** Oops concepts that we have in Apex are:

1. Objects
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

**Objects:** It is a real life entity. Like Car, Bike, book, Customer. It is an instance of class. A class can have multiple instances. Objects has methods and member variables or instance variables.

Eg: Account, Contact, Opportunity, Lead, etc.

**Class:** A class is just like a collection of real life entity. Like a showroom of cars, A compant having employees, etc. It is a collection of similar entities.

Eg: public class AccountHandler{

}

**Inheritance:** When the object acquires the behaviour/properties of its parent’s class then it is called Inheritance. Inheritance is possible when a child class extends the Parent class. And the Child class inherits the properties of parent or super class.

**Polymorphism:** Polymorphism is a concept in oops. Polymorphism means Many forms/shapes. Polymorphism can be of 2 types static or dynamic.

Static Polymorphism happens at compile time only. Meaning that the compiler knows at compile time only that which method will be called.

Suppose there are two methods in a class and they are having different signatures then it will be a static polymorphism. This is also called method Overloading.

Eg: public class CalculateArea {

Integer side;

Integer length;

Integer breadth;

Public void **Area**(Integer side) {

Double area = side\*side;

System.debug(area);

}

Public vid **Area**(Integer length, Integer breadth) {

Double area = length \* breadth;

System.debug(area);

}

}

In the above example the method area is Overoaded. And this method has different method signatures.

Dynamic Polymorphism happens at runtime. Meaning that at runtime it will be decided that which method will be called. Dynamic Polymorphism is also called Method Overriding.

Eg: Public class car {

Public void color() {

System.debug(‘Generic Red car’);

}

}

Public class TataCar extends Car {

Public void color() {

System.debug(‘Tata Red car’);

}

}

Now when this color() will be called/invoked, at runtime it will be decided that which class’s method will be invoked.

**Abstraction:** Hiding the data and showing/exposing only the relevant/necessary part is called Abstraction. Hiding internal details and showing functionality is abstraction. We use inheritance for implementing abstraction. Access Specifiers(public, private, protected) are also used to achieve abstraction. Hiding internal complexity and showing functionality is known as abstraction.

**Encapsulation:** Capsulating/binding the data into a single unit is called Encapsulation. We use classes and Interfaces to implement encapsulation. Encapsulation is used to provide a better security to data. Moreover, we can manipulate the data using encapsulation.

Global class BatchClass implements Database.Batchable {

}

**Q: Constructor Overloading in Apex?**

**A:** public class TestClass {

Public TestClass(){

//Default constructor or ParameterLess Constructor

}

Public TestClass(Boolean isFalse){

//Paramaterized Constructor

}

Public TestClass(Boolean isTrue, String Name) {

//Paramaterized Constructor

}

}

**Q: Assert statements in Apex**

**A:** System.assert()

This statement has 2 parameter. 1. Is the condition. 2nd(Optional) is the message.

System.assert() is used to assert that the specified condition is true. If the condition is false then a error is thrown.

System.AssertEquals()

This statement takes 3 parameters 1st The actual result, 2nd the expected result, 3rd is the message(optional).

System.AssertEquals() is used to compare whether the actual result and the expected result are equal or not. If not equal then it will throw an Error.

System.AssertNotEquals()

This statement takes 3 parameters 1st The actual result, 2nd the expected result, 3rd is the message(optional).

System.AssertNotEquals() is used to compare whether the actual result and the expected result are equal or not. If equal then it will throw an Error.

**Q: How can we maintain the chain of batch classes?**

**A:** In order to maintain the chain of batch classes. We first need to understand the 3 methods inside Database.batchable.

1. Start() this method is executed once to send the list of sobjects to execute().
2. Execute() this method is executed multiple times depending upon the batch size, taking the sObjects on chunks.
3. Finish() is executed once and is used to do post processing implementations like sending email alerts etc.

So, the batch apex chaining can happen through the finish(). We can call another batch class from finish() method. Important point to remember here is that we can have 5 jobs running concurrently. So the only 5 jobs can be chained. **Ex:**

Public void finish(Database.BatchableContext bc) {

Id firstBatchId = bc.getJobId();

Id job2Id = Database.executeBatch(new otherBatchJob(), 200);

}