Queueable Apex

- Take control of asynchronous Apex Process by using the Queueable interface.
- This interface enables to add jobs in the queue and monitor them.
- Using the interface is an enhanced way of running asynchronous apex code compared to using future methods.
- Apex processes that runs for a long time , such as extensive database operations or external web service callouts, can be run asynchronously by implementing Queueable interface and adding a job to the apex job queue.
- In this way asynchronous apex jobs runs in the background in its own thread and does not delay the execution of main logic.
- Each queued job runs when the System resources are available.
- A benefit of using queueable resource is that Some governor limits are high than for synchronous apex, such as heap size limit.

Note : If an apex transaction rolls back , any jobs queued for execution by the transaction aren't processed.

Queueable jobs are similar to future methods in that they're both queued for execution, but queueable provides with additional benefits:

- Getting an Id of the job when submitted by invoking System.enqueueJob method , the method returns the Id of the new job. This Id corresponds to the Id of the AsyncApexJob record. Use this Id to identify and monitor the job either through salesforce UI (Apex Job Page) or programmatically by querying the records from the AsyncApexJob.
- Using non-primitive types : queueable classes contain member variables of non-primitive data types, such as sobjects or Custom Apex Types. These Objects can be accessed when the job executes.
- Chaining Jobs : can chain one job to another job by starting a second job from a running job. Chaining jobs if process depends on another process to have run fast.

Note : Variables that are declared as "transient" are ignored by serialization and deserialization and the value is set to null in Queueable apex.

<u>Adding a queueable job to the asynchronous execution :</u>

EX :

```
Public class AsyncExecutionExample implements Queueable{
        Public void execute(QueueableContext context){
                Account acc = new Account(Name='Test');
                Insert acc;
        }
}
```

To add this class as a job on the queue , call this method.

```
Id jobId = System.enqueueJob(new AsyncExecutionExample());
```

After submitting queueable class for execution , the job is added to the queue and will be processed when system resources become available. Can monitor the status of the job programmatically by querying AsyncApexJob or through the UserInterface in the setup by entering Apex Jobs in quick find box , then selecting Apex Jobs.

To query information about submitted job, perform a SOQL query on AsyncApexJob by filtering on the JobId that System.enqueueJob method returned.

```
AsyncApexJob jodDetail = [SELECT Id, Status , NumberOfErrors from AsyncApexJob where Id =:jobId];
```

** In queueable execute method Batch class can be called.

<u>Adding a Queueable job with a specified minimum delay :</u>

Use the System.enqueueJob(queueable , Delay) method to add queueable jobs to the asynchronous execution queue with a specified delay (0-10 minutes). The delay is ignored during testing in test class.

Note :

When delay is set to 0(Zero) , the queueable job runs as quickly as possible. With chained jobs, implement a mechanism to slow down or halt the job if necessary. Without such a fail-safe mechanism in place, can rapidly reach the daily async apex limit.

In following case, it would be beneficial to adjust the timing before the queueable job is run.

- If the external System is rate-limited and can be overloaded by chained jobs that are making rapidly callout.
- When polling for results, and executing too fast can cause wasted usage of daily async apex limits.

Ex: Integer delayInMinutes = 5;

System.enqueueJob(new MyQueueable() , delayInMinutes);

Admins can delay by using default org-wide delay (1-600 seconds) in scheduling queueable jobs that were scheduled without a delay parameter. Use the default setting as mechanism to slow down default queueable job execution. If the setting is omitted , Apex uses the standard queueable timing with no delay added.

- ⇨ In Quick Find -> Apex Settings -> Enter a value (1-600 Seconds) for default minimum enqueue delay (in seconds) for queueable jobs that do not have a delay parameter.
- ⇨ Or can be done this programmatically by using Apex Setting in metadata API developer guide.

<u>Adding a Queueable job with a specified stack depth:</u>

The System.AsyncInfo class has the properties which determines the current and maximum stack depths and the minimum queue delay .

The methods f AsyncInfo Class are :

- ⇨ getCurrentQueueableStackDepth()
- ⇨ hasMaxStackDepth()
- ⇨ getMaximumQueueableStackDepth()
- ⇨ getMinimumQueueableDelayInMinutes()

Ex :

```
Public class AsyncInfoDemo Implements Queueable{

        Public void execute(QueueableContext qc){

                System.debug(AsyncInfo.getCurrentQueueableStackDepth());

                System.debug(AsyncInfo.hasMaxStackDepth());

                System.debug(AsyncInfo.getMaximumQueueableStackDepth());

                System.debug(AsyncInfo.getMinimumQueueableDelayInMinutes());

        }

}
```

o/p :

```
false

1

Null

Null

Maximum stack depth is 5.
```

<u>Testing Queueable jobs :</u>

Queue it between Test.startTest() and Test.stopTest() block to test the Queueable class. System executes all asynchronous processes started in a test method synchronously after the Test.stopTest() method/statement. Next , the test method verifies the result of the queueable job by querying the account that the job created.

<u>Chaining jobs :</u>

To run a job after some other processing is done first by another job, can chain queueable jobs. To chain a job to another job, which means that only one child job can exist for each parent job . Ex: If you have a second class called secondJob that implements the queueable interface, can add this class to queue in the execute() method as follows.

Ex :

```
public class AsyncEx implements Queueable{

        public void execute(QueueableContext qc){

                System.enqueueJob(new SecondJob());

        }

}
```

<u>Queueable Apex Limits :</u>

- ➔ can add upto 50 jobs to the queue with System.enqueueJob in a single transaction. In asynchronous transaction (Ex: from a batch apex) , can add only one job to the queue with System.enqueuejob. To check how many queueable jobs have been added in a transaction, call Limits.getQueueableJobs().
- ➔ Because no limit is enforced on the depth of chained jobs, can chain one job to another. Can repeat this process with each new child job to link it to a new child job.

The maximum depth for the chained job is 5, which means you can chain jobs 4 times. The maximum number of jobs in the chain is 5, including the initial parent queueable job.

- ➔ When chaining jobs with System.enqueueJob, can add only one job from an executing job. Only one child job can exist for each parent queueable job. Starting multiple child jobs from the same queueable isn't supported.
- ➔ If add 2 child jobs – gets too many queueable jobs added to the queue error.
- ➔ QueueableContext has getJobId() :- Returns the Id of the Submitted job that uses the Queueable interface.