**Exceptions in Salesforce:**

Exceptions note errors and other events that disrupt the normal flow of code execution. throw statements are used to generate exceptions, while try, catch, and finally statements are used to gracefully recover from exceptions. There are many ways to handle errors in your code, including using assertions like System.assert calls, or returning error codes or Boolean values.

The advantage of using exceptions is that they simplify error handling. Exceptions bubble up from the called method to the caller, as many levels as necessary, until a catch statement is found that will handle the error. This relieves from writing error handling code in each of your methods. Also, by using finally statements, you have one place to recover from exceptions, like resetting variables and deleting data.

Example of how the try, catch, finally look:

try {

code of database operations that might cause an exception.

} catch(DmlException e) {

DmlException handling code here.

}

catch(Exception e) {

Generic exception handling code here

. }

finally { Perform some clean up. }

**Example:**

Merchandise\_\_c m = new Merchandise\_\_c();

insert m;

Here the insert DML statement in the example causes a DmlException because we’re inserting a merchandise item without setting any of its required fields.

We can write this using following code without any errors

try {

Merchandise\_\_c m = new Merchandise\_\_c();

insert m;

}

catch(DmlException e) {

System.debug('The following exception has occurred: ' + e.getMessage());

}

**Some imp Exceptions used in salesforce**

**DmlException**

Any problem with a DML statement, such as an insert statement missing a required field on a record.

**ListException**

Any problem with a list, such as attempting to access an index that is out of bounds

**NullPointerException**

Any problem with dereferencing a null variable

**QueryException**

Any problem with SOQL queries, such as assigning a query that returns no records or more than one record to a singleton sObject variable.

**SObjectException**

Any problem with sObject records, such as attempting to change a field in an update statement that can only be changed during insert.

**Apex Transaction:**

An Apex transaction represents a set of operations that are executed as a single unit. All DML operations in a transaction either complete successfully, or if an error occurs in one operation, the entire transaction is rolled back and no data is committed to the database.

For example, consider the following: a custom Apex Web service method causes a trigger to fire, which in turn calls a method in a class. In this case, all changes are committed to the database only after all operations in the transaction finish executing and don’t cause any errors. If an error occurs in any of the intermediate steps, all database changes are rolled back and the transaction isn’t committed.

These are useful in cases when several operations are related, and either all or none of the operations should be committed. This keeps the database in a consistent state.

Another common scenario is transferring funds from one bank account to another. It involves debiting the first account and crediting the second account with the amount to transfer. These two operations need to be committed together to the database. But if the debit operation succeeds and the credit operation fails, the account balances will be inconsistent.

Example:

public class MerchandiseOperations {

public static Id invoice( String pName, Integer pSold, String pDesc) {

// Retrieve the pencils sample merchandise

Merchandise\_\_c m = [SELECT Price\_\_c,Total\_Inventory\_\_c FROM Merchandise\_\_c WHERE Name = :pName LIMIT 1];

// break if no merchandise is found

System.assertNotEquals(null, m);

// Add a new invoice

Invoice\_Statement\_\_c i = new Invoice\_Statement\_\_c( Description\_\_c = pDesc);

insert i;

// Add a new line item to the invoice

Line\_Item\_\_c li = new Line\_Item\_\_c(

Name = '1',

Invoice\_Statement\_\_c = i.Id,

Merchandise\_\_c = m.Id,

Unit\_Price\_\_c = m.Price\_\_c,

Units\_Sold\_\_c = pSold);

insert li;

// Update the inventory of the merchandise

item m.Total\_Inventory\_\_c -= pSold; update m; return i.Id;

}

}

**Triggers:**

Triggers are Apex code that execute before or after an insert, update, delete or undelete event occurs on an sObject.

Trigger Syntax:

trigger TriggerName on ObjectName (trigger\_events) {

code\_block

}

**Trigger Events:**

before insert

before update

before delete

after insert

after update

after delete

after undelete