APEX HOURS

20 Salesforce Apex Code Best Practices

swipe <<

1. Bulkify Apex Code

The first principle is to write code for multiple records at a time. We should write scalable code and avoid hitting the governor. Let’s understand with an example. We use a hard-coded index in the code below [0]

```
Trigger AccountTrigger on Account(before insert){
  Account acc= Trigger. New[0];
  if(acc.Name != null){
    // DO someything
  }
}
```

swipe <<

2. Avoid SOQL & DML inside for Loop

```
For(Account acc: Trigger.new){
 for(Contact con:[select id from contact where accountId = :acc.Id]){
 }
}
```

Solution

- Move SOQL/DML out of loops.
- **Query**: If you need query results, get all the records using a single query and iterate over the resultset.
- **Update**: If you need to update, batch up the data into a collection and invoke DML once for that collection

swipe <<

3. Querying Large Data Sets

Solution

1. Building Efficient & Selective Queries.
2. Avoid Common Causes of Non-Selective SOQL Queries.
3. Use Query Optimizer.
4. Avoiding querying on formula fields.
5. Custom Indexes Containing null Rows.
6. Avoid SOQL injection.
7. Avoid using negative filter operators

swipe <<

4. Use of Map of Sobject

Solution

1. Use Apex Collections to query data and store the data in memory efficiently.
2. A combination of using collections and streamlining SOQL queries can substantially help write efficient Apex code and avoid governor limits

```
Map<Id, Account> accountMap=new Map<Id, Account> ([select id, name from Account
where name like ‘%Inc.%’]);

for(Contact con: [select id,AccountId, name from Contact where AccountId
in:accountMap.keyset()]{
    //Get Related Account data using Contact field
    Account acc=accountMap.get(con.AccountId);
}
```

swipe <<

5. Use of the Limits Apex Methods

Solution

Use Apex Limits Methods to Avoid Hitting SF Governor Limits. Many of us face governor limit errors in trigger/classes/test classes

```
System.debug('Total Number of SOQL allowed in this Apex code: ' +
Limits.getLimitQueries());

Account acc=[select id, name from Account where BillingCity!=null  order by
createdDate desc limit 1];

System.debug('1. Number of Queries used in this Apex code so far: ' +
Limits.getQueries());
```

swipe <<

6. Avoid Hardcoding IDs

Don’t hard code ID in Apex Code. Deploying apex code between sandbox and production may fail because id may not be the same, for putting recordType Id.

Solution

```
Id accountRTId=
Schema.sobjectType.Account.getRecordTypeInfosByName().get(‘RTName’).getRecordType
Id();
```

Use of custom settings/custom metadata to avoid hard coding IDs

7. Use Database Methods

Solution

An Apex transaction represents a set of operations executed as a single unit. All DML operations in a transaction are completed successfully, or if an error occurs in one operation, the entire transaction is rolled back, and no data is committed to the database. Apex allows you to generate a savepoint, a point in the request that specifies the state of the database at that time

```
Database.SaveResult[] accountResults=Database.Insert(accountsToBeInsert, false);
// Using above code, we can work on failure records
  For(Database.SaveResult sr:accountResults) {
    If(!sr.isSuccess()) {
    // add your logic here
    for(Database.Error err : sr.getErrors()) {
    }
  }
}
```

swipe <<

8. Exception Handling in Apex

Solution

DML statements return run-time exceptions if something goes wrong in the database during the execution of the DML operations. Don't forget to use Try catch blocks for exception handling. With Apex, you can write code that responds to specific exceptions

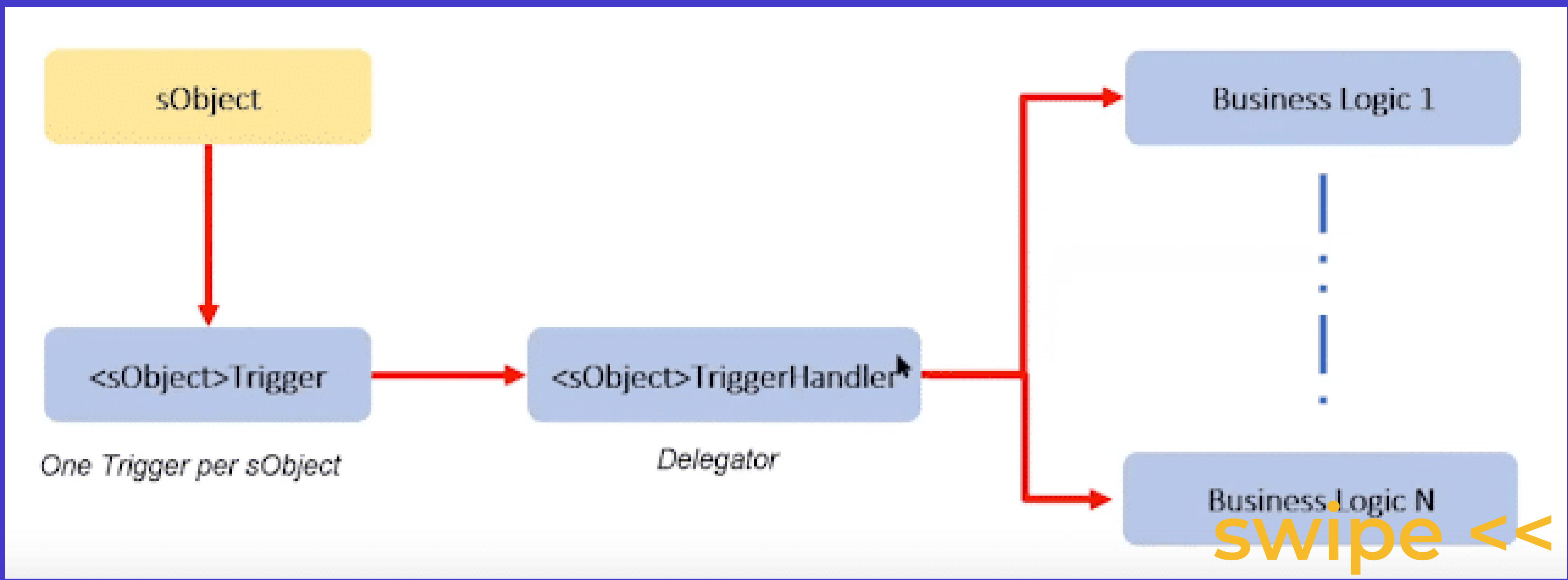
```
try{
    // Apex Code
}Catch(Exception e){
}
```

swipe <<

9. Write One Trigger per Object

Solution

Apex triggers within Salesforce are designed to help you automate certain tasks. Apex triggers allow you to perform custom actions before and after events in Salesforce. While writing Apex Trigger, you should follow the best practice and create one Trigger per object.

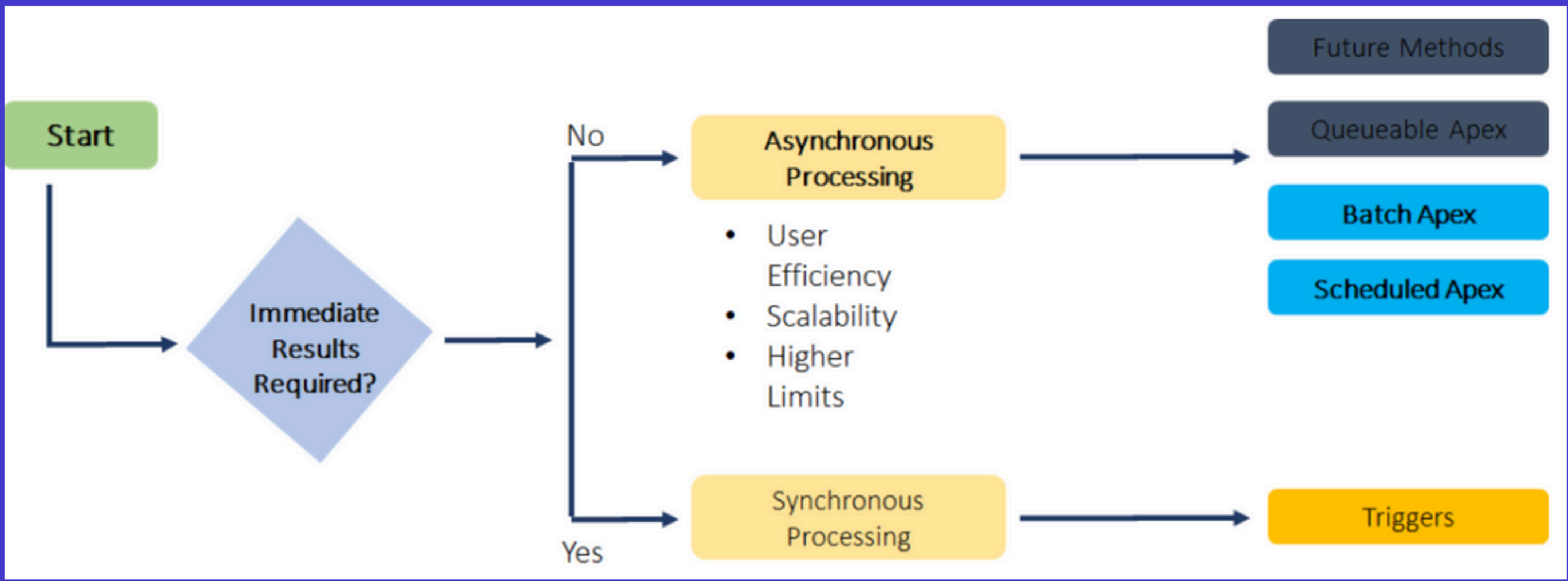


swipe <<

10. Use Asynchronous Apex

Solution

Apex, written within an asynchronous method, gets its own independent set of higher governor limits. For example, the number of SOQL queries doubles from 100 to 200 when using asynchronous calls. The total heap size and maximum CPU time are larger for asynchronous calls.



11. Security and Sharing in Apex

Solution

Enforcing Object & FLS Permissions in Apex
Apex doesn’t enforce object-level and field-level permissions by default. Let see how we can enforce the CRUD & FLS in Apex.

- Schema methods
- WITH SECURITY_ENFORCED
- Security.stripInaccessible()
- Database operations in user mode **(pilot)**

SHARING CLAUSES FOR APEX CLASSES

1. WITH SHARING
2. WITHOUT SHARING
3. INHERITED SHARING

swipe <<

12. Make reusability of Apex Code

Solution

The best code is written away from the keyboard. Break methods into multiple smaller methods if possible, making your code more modular and easier to read and maintain. A critical best practice with Apex Code is always to write scalable and reusable code so that it can be reused in another functionality or module. This entails making code generic so it can be reused with minimum modification.

13. Use Custom Setting and Custom Metadata

Solution

Custom Settings and Custom Metadata Types allow you to store configuration data that can be easily accessed in Apex.

```
// Custom Setting Example
My_Custom_Setting__c setting = My_Custom_Setting__c.getInstance();
String configValue = setting.Config_Value__c;

// Custom Metadata Type Example
My_Metadata_Type__mdt metadata = [SELECT Label, Value__c FROM
My_Metadata_Type__mdt WHERE Label = 'Config'];
String metadataValue = metadata.Value__c;
```

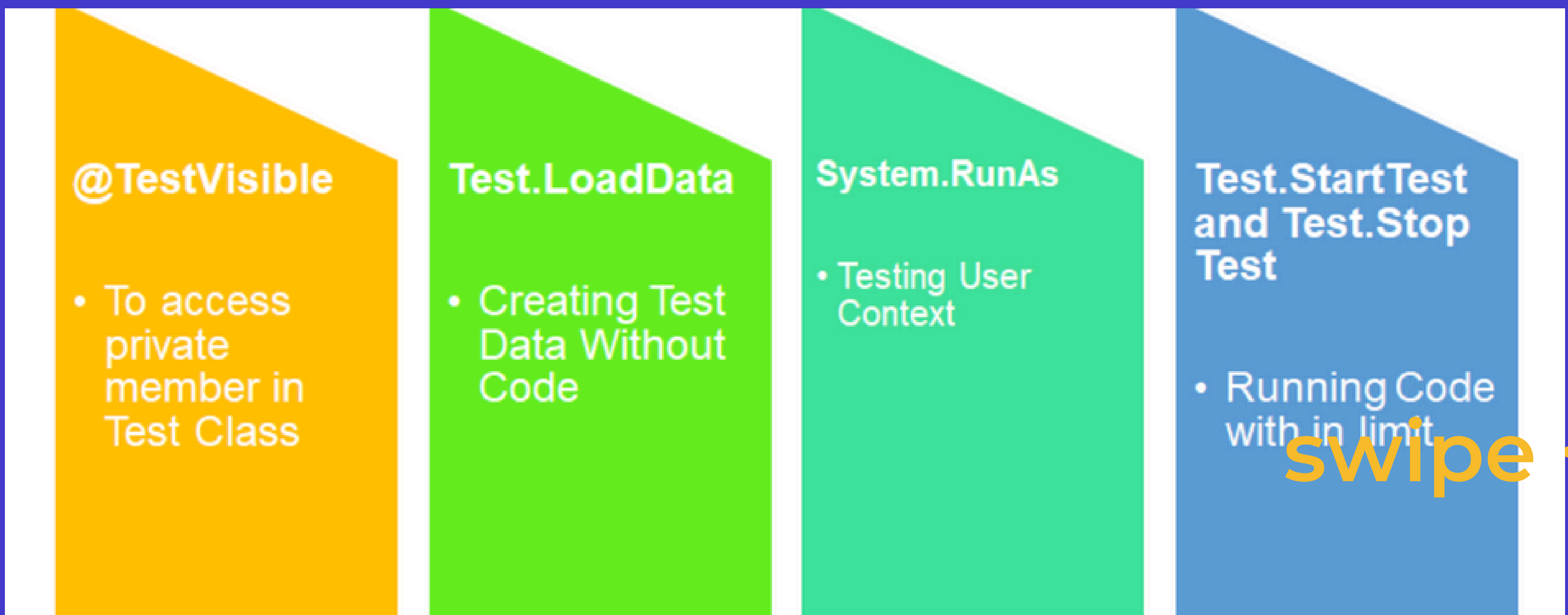
swipe <<

14. High Code Coverage

Solution

Unit tests are the tests performed by the developers to ensure that functionality is working as expected, considering Both positive and negative tests. We should not focus on the percentage of code coverage

1. Bulk Records
2. Positive & Negative testing.
3. Restricted User testing.
4. One Assert Statement per method.
5. should not use @isTest(seeAllData=true).



swipe <<

15. Return Early Pattern

Solution

Return early is the way of writing functions or methods so that the expected positive result is returned at the end of the function, and the rest of the code terminates the execution.

```
static void someAction(Case record)
    {
        if (/*condition 1*/){
            // do stuff
            return;
        }
        if (/*condition2*/)
        {
            // do stuff
            return;
        }
        // do stuff
        return;
    }
```

swipe <<

16. Avoid nesting loops within loops

Solution

Nested loops should be avoided in Apex controllers because they may slow down the processing of the page or may hit the governing limits for the page. One easy way, which gives some nice structure to the code, is to make the inner loop a separate function or minimize using loops altogether.

17. Don’t mix Apex & Automation

Solution

Don’t mix Apex, Process Builders, Workflow Rules, and Record-Triggered flows

1. Poor performance.
2. Unexpected results due to the inability to control the order of operations in the ‘stack’.
3. Increase in technical debt with admins/devs not collaborating.
4. Documentation debt.

18. Apex Naming Conventions

Solution

Another critical best practice surrounding Apex Code is following a proper naming convention. The naming convention in Salesforce is a rule to follow as you decide what to name your identifiers like class, variable, constant, method, etc. But it is not forced to follow. So, it is known as convention, not rule. Naming conventions make the application easier to read and maintain

- **camelCase** : Each word in the middle of the respective phrase begins with a capital letter. for example apexHours

```
String firstName;
```

- **PascalCase**: It is same like Camel Case where first letter always is capitalized. for example ApexHours

```
Class UserController{ }
```

- **kebab-case**: Respective phrase will be transferred to all lowercase with hyphen(-) separating words. for example apex-hours

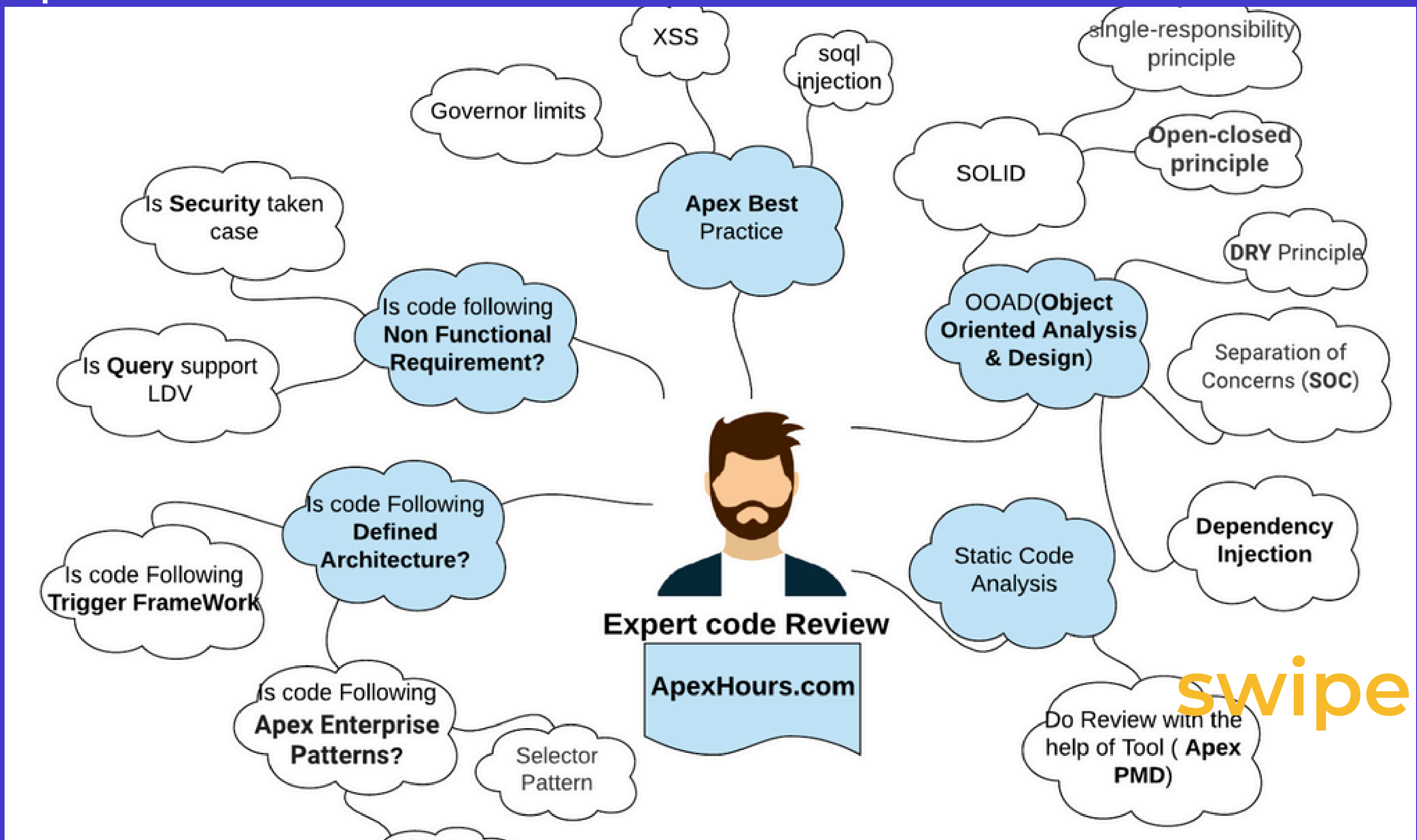
swipe <<

APEX HOURS

19. Code Review process

Solution

40%–80% of the lifetime cost of a piece of software goes to maintenance. To avoid future tech debt, we should follow the code review process in each project and should create one code review checklist for developers



swipe <<

20. Apex Performance

Solution

How to measure the performance of your code and understand where time is being used within a transaction, as well as the actions to take and some best practices to keep things running nicely. Some Tips are here.

1. Query only required fields in SOQL.
2. Retrieve only relevant data rows and apply filters and limits.
3. Remove debug statements from the code once testing is done.

swipe <<

APEX HOURS

DID YOU LIKE THIS?

Let me know
in the comments!