Salesforce GROW

Module : Exceptions and Debugging

OurUniversity
EFMD CLIP ACCREDITED

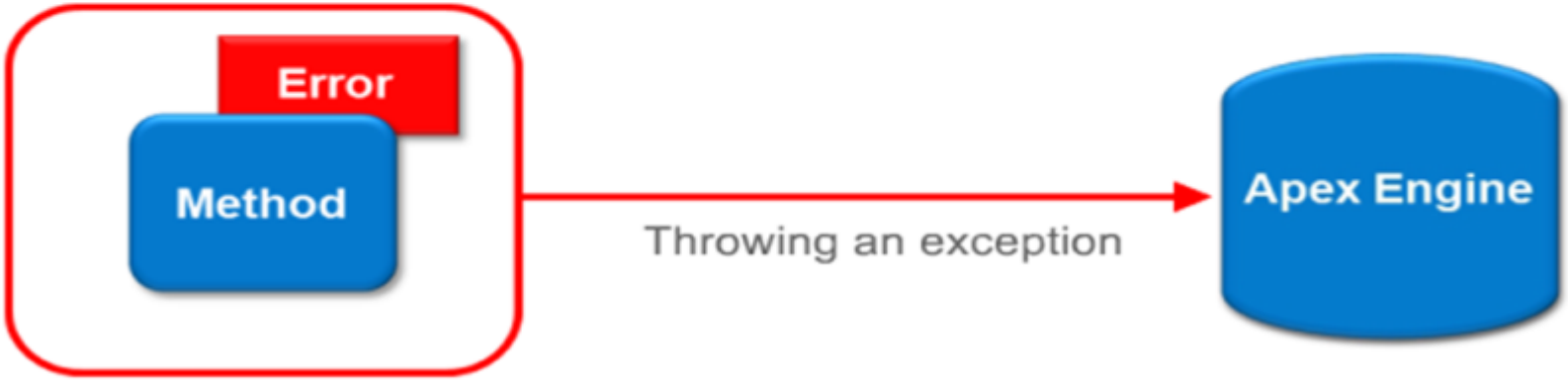
Exceptions and Debugging

Capgemini

Exceptions

Apex uses exceptions to record:

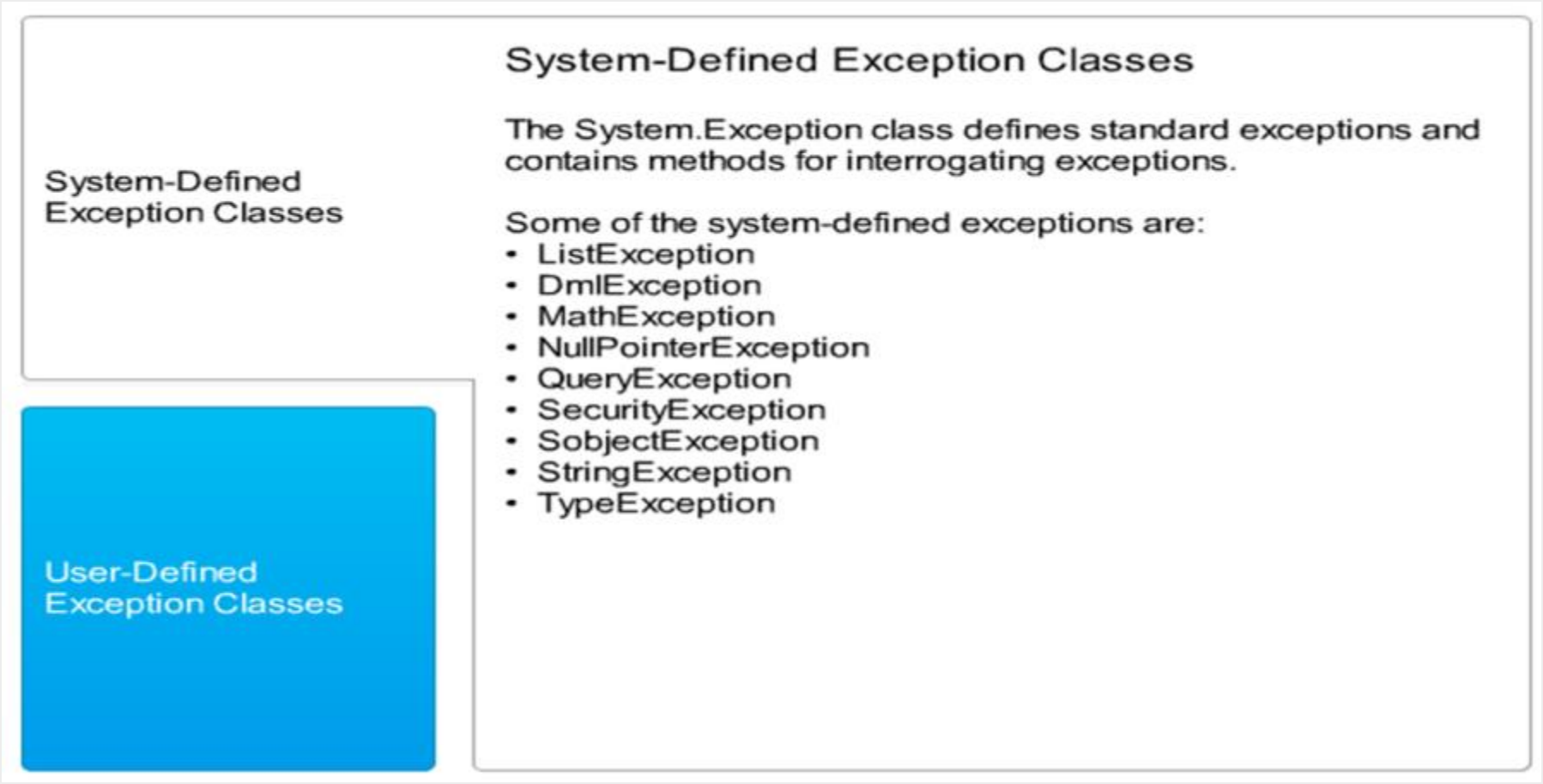
- Errors and other events that disrupt the execution of code.
- Information about:
  - The error.
  - The type of error.
  - The state of the script or program when the error occurred.



Exceptions(Cont.)

- `throw`: Indicates that an error has occurred and provides an exception object.
- `try`: Identifies the block of code where the exception can occur.
- `catch`: Identifies the block of code that can handle a specific exception.
- `finally`: Identifies a block of code that executes after a try block.

Exception Types



Exception Types

System-Defined Exception Classes

User-Defined Exception Classes

User-Defined Exception Classes

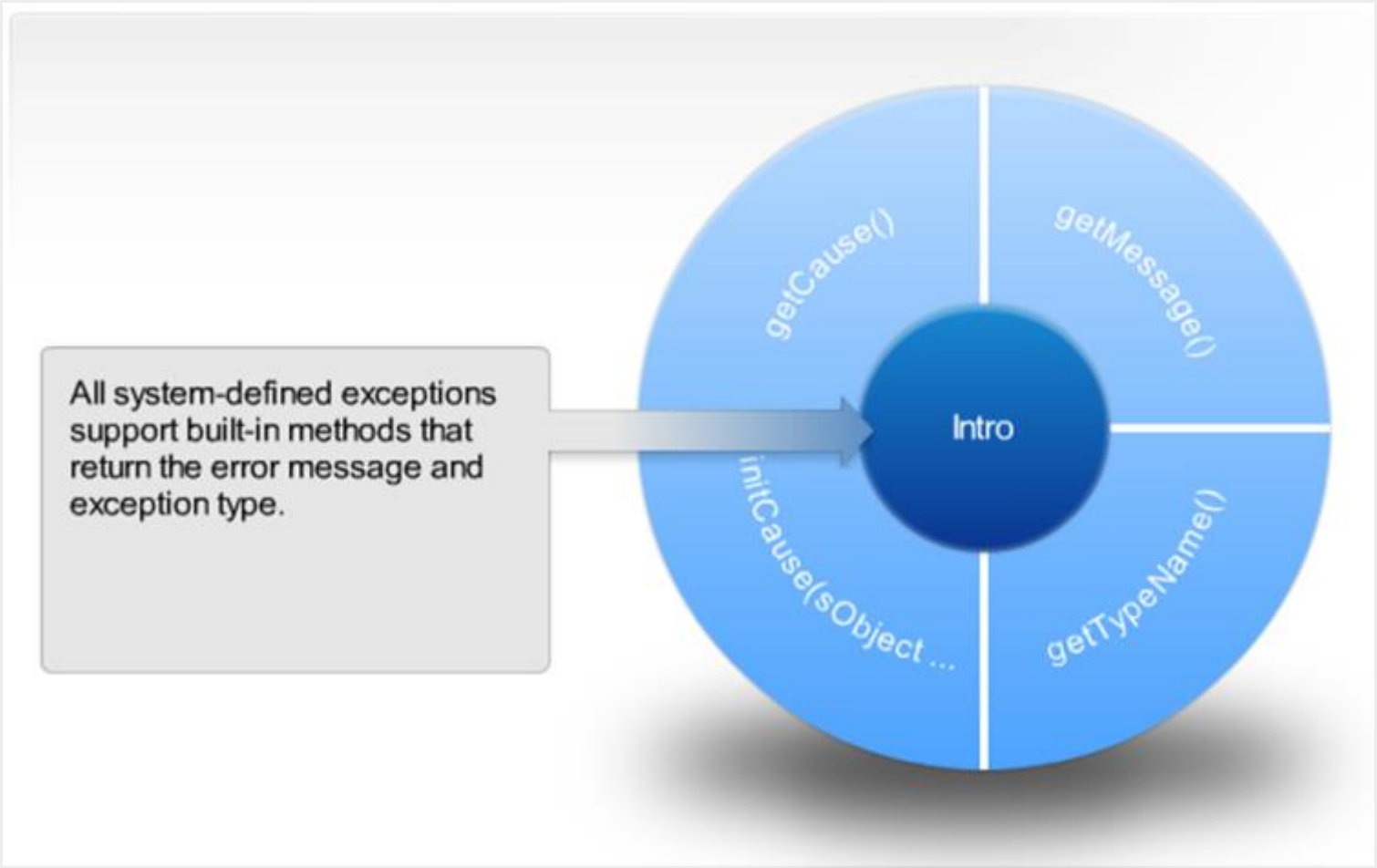
- Have their own characteristics.
- Behave as any other exception type.
- Are created if the system-defined exception classes do not provide the required behavior.
- Provide more control over the program behavior.
- Must end with the string `Exception`.
- Must extend the system-defined Exception class.

The Exception class:

- Is a catch-all type of class.
- Should be placed at the end of all catch blocks.

```
1  public virtual class BaseException extends Exception {}
2  public class OtherException extends BaseException {}
```

System-Defined Exception Methods



System-Defined DML Exception Methods

- Methods that have DML in their name are only supported for DML exceptions.
- Most DML exceptions take an integer parameter of the *i*th failed row.

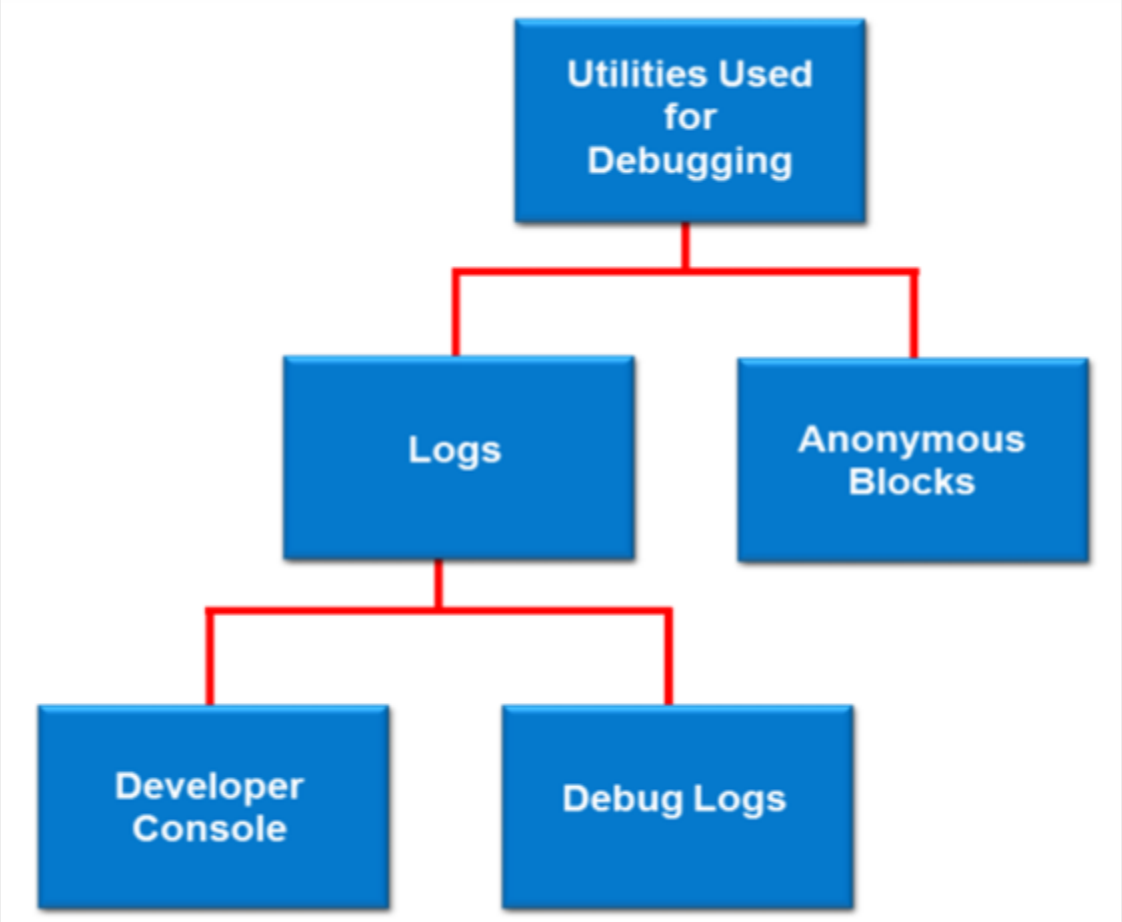
DML Exception Method	Returns
`getDmlFields(int)`	The names of the field(s) caused by the *i*th failed row
`getDmlIndex(int)`	The original row position of the *i*th failed row
`getDmlMessage(int)`	The user message for the *i*th failed row
`getDmlStatusCode(int)`	The Apex failure code for the *i*th failed row
`getNumDml()`	The number of failed row

User-Defined Exception Types

- Developers can extend the exception classes, allowing user-defined exception types to form an inheritance tree.
- `catch` blocks ensure that the lowest or deepest level of exception or any of its parent exceptions are caught.

```
1  public virtual class BaseException extends Exception {}
2  public class OtherException extends BaseException {}
3  try {
4    Integer i;
5    // Your code here
6    if (i < 5) throw new OtherException('This is bad');
7  }
8  catch (BaseException e) {
9    // This catches the OtherException
10 }
11 catch (Exception e) {
12  // This catches the general Exception
13 }
```

Debugging



System Log

- A System log:
  - Records errors and system processes that occur in an organization.
  - Records entries for every user whenever an Apex script, a rule, or an approval process executes.
  - Can be viewed using the Salesforce UI or the Force.com IDE.
  - Is limited to 2 MB per log.
- Each organization can retain up to 50 MB of logs.

Log Filters

Log Filters	Includes
`Database`	Log messages generated by calls to the `System.debug` method, DML statement, or inline SOQL or SOSL query
`Workflow`	Information for workflow rules, assignment rules, auto-response rules, escalation rules, and approval processes
`Validation`	Information for validation rules
`Callout`	Request-response XML that the server sends and receives from an external Web service
`Apex Code`	Information about Apex scripts, DML statements, inline SOQL or SOSL queries
`Apex Profiling`	Cumulative profiling information
`Visualforce`	Information about Visualforce events
`System`	Information about calls to all System methods

Debug Log and Filters

Debug Log Filters

With debug log filters, you can:

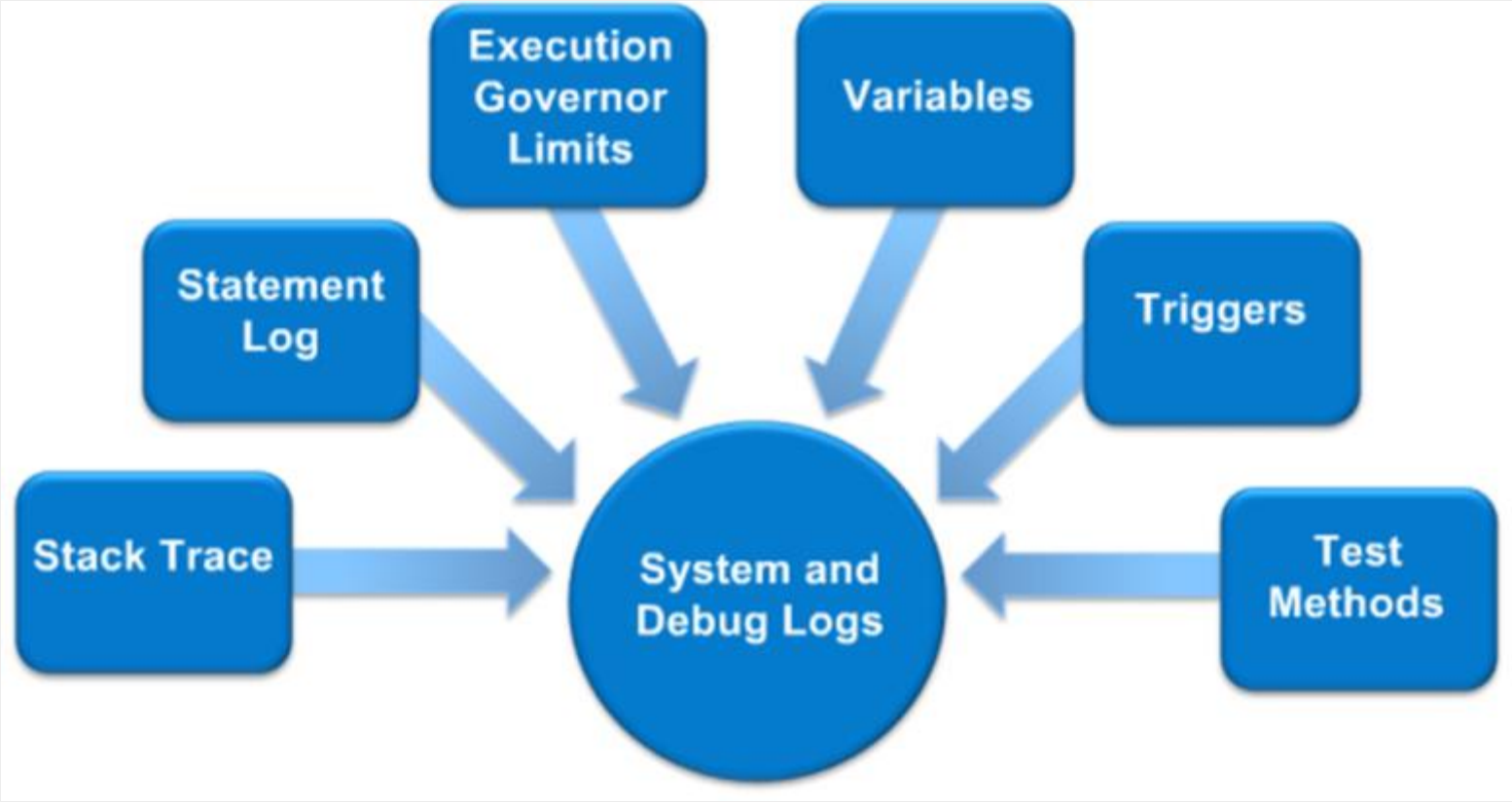
- Filter unwanted information from the debug logs.
- Set appropriate log levels for classes and triggers:
  - The log levels specified for a class apply to all the classes, if falling in the same execution path.
  - The log levels can also be set for individual classes and triggers.

Debug Log

A debug log:

- Captures the same information as the system logs.
- Contains:
  - The output Apex debug statement.
  - The results of testing.
  - Unique custom messages.
  - Default messages.

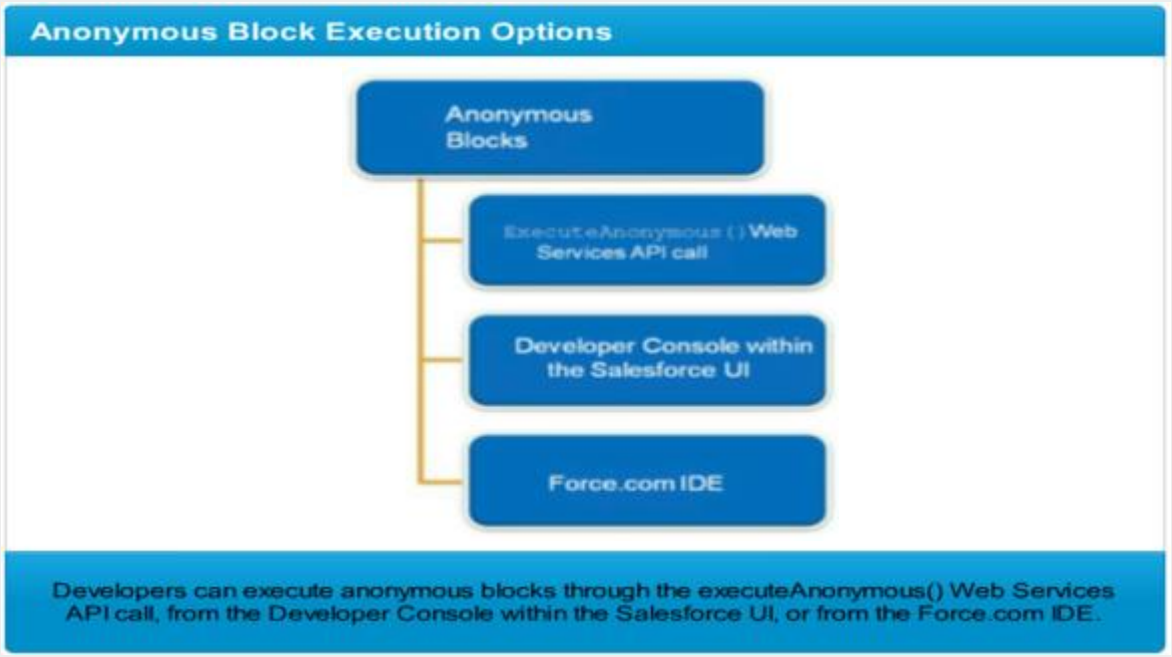
Debugging Apex In Logs



Anonymous Blocks and Execution Options

An anonymous block:

- Does not get stored in the metadata.
- Gets dynamically compiled and executed.
- Is executed using the full permissions of the current user.
- Returns results that include status information for the compile and execute phases of the call and any errors that occur.
- Can be used to evaluate Apex.
- Can be used to write scripts that change dynamically at runtime.
- Cannot include the `static` keyword.



Hands-on Exercise

Apex: Debugging
Exercise 6-1: Creating a Test Class

https://lms.cfs-api.com/v1/content/e1e000f7-2f7f-420c-8e5c-b77a6672cd04/presentation_content/external_files/exceptions,debugging,andtestingexerciseguide.pdf