* Review
  + Pillars
    - Encapsulation
      * Controlling access to your data within a class
      * Tools
        + Access modifiers

Private

Protected

Public

Global

* + - * + Static
        + Properties

Get and set methodology

* + - Abstraction
      * Hiding away details of implementation
      * Like a contract with a developer rather than a blueprint
      * Tools
        + Abstract keyword
        + Interfaces
    - Inheritance
      * Parent child relationship of classes
      * Reuses existing functionality and builds upon it
      * Tools
        + Extends

Creates parent child relation of classes

The extending class in the child

* + - * + Implements

Grants access to interface methods that require implementation

* + - * + This

Refers to current class

Used for constructor chaining

* + - * + Super

Refers to the parent class

Used to access parent functionality and constructors

If you are going to use it, has to be the first one

* + - Polymorphism
      * The ability of an object to take on many forms
      * Allowing code to have multiple definitions/meanings
      * Allows object to be multiple types
      * Tools
        + Overloading

Same signature, different parameters

Usually different body (but not required)

* + - * + Overriding

Same signature

Same parameters

Different method body

* + - * + Constructors
        + Virtual
  + Inner classes
    - Same or more restrictive access modifier as outer class
    - Functionally indistinct from outer classes
      * Anything you can do in the outer class, you can do in the inner class
  + Exceptions
    - Not an error
    - Can be caught and handled
    - If not handled, will stop the program
    - Can create custom ones
      * Must end in word exception
      * Extends System.exception
    - Finally block is optional
    - Catch blocks should be ordered from most specific to least specific
    - Try block contains the code that will throw the exception
    - Methods
      * getMessage
      * getStackTrace
      * getLineNumber
      * setMessage
  + Error
    - Cannot be caught or handled
    - No matter what will end your program
  + Triggers
    - One trigger per object
      * Cannot control order of execution
    - Bulkification
      * Allow for any number of records
    - Avoid trigger recursion
      * Don’t do something that may call itself
    - Keep the trigger logicless
      * Makes it easier to maintain
    - Context variable
      * isDelete
      * isUpdate
      * Variables that hold information about the currently executing trigger
      * 2 types of triggers
        + Before

Perform complex validation

Can edit field values before DML operation completes

* + - * + After

Accessing system values

Manipulate other objects based on our finalized field values

* Testing
  + In order to deploy Apex to your org or to put it in a package for the AppExchange, your code coverage must be at least 75% for all Apex classes
    - This means 75% of your code needs to be run by your tests
    - Does not mean it is well tested
  + This metric does not include test classes
  + Triggers must have at least 1% code coverage
  + File structure
    - 
  + Precede with @isTest
    - Only use @isTest(SeeAllData=true)
    - Allows your tests to use org data
    - Do not do it
    - Ever
  + @testSetup
    - Cannot do this if you set SeeAllData
    - Is run one time before your tests are executed
    - After each test method, everything is rolled back to this state
  + Triggers still fire in any DML in your tests
  + IDs do not exist until inserted in the database
  + Helper class
    - Can insert data in the database
    - Less efficient
    - Still a test class
    - Can perform different setup for different cases
    - Test methods must be static and void
    - When appropriate,
      * Test positive
      * Test negative
      * Test single
      * Test bulk
        + Don’t exceed governor limits
        + Test with 151 and 201 so that it can ensure it does not exceed limits

These are baby numbers

* + - * Test restricted users
        + Pick a user that has restricted access and see if it works
        + Apex runs in system context

It will not take into account what user triggered the code

Will be more relevant later

* + - Will not always make sense
    - The name of the test should start with the letters in the beginning
      * TP\_
      * TN\_
      * TS\_
      * TB\_
      * TRU\_
  + All tests are part of the same transaction
    - We do not want to not exceed the limits though
    - We can use Test.startTest() and Test.stopTest() to reset limits
    - This also applies to the setup
      * Start will reset it to the point after the setup
    - We only get one set per test method
  + We want to test robustly
    - Test for anything and everything
  + Assert statements
    - We can use multiple system.assert() statements
    - System.assertExuals()
      * Most common
      * 2 parameters
        + First is expected
        + Second is actual value
      * Example
        + System.assertEquals(true, result);
    - System.assertNotEquals()
      * The same but for not equals as equals
    - Test for exceptions in a try catch block
      * Try{
      * KnownExceptionThrown
      * }catch(System.Exception e){
      * System.assert(true);
      * }
      * Text

        Description automatically generated
    - @testVisible
      * When used before a method, will make a method useable outside the class
    - System.runas(admin)
      * Only works in a test
  + There are a few ideologies with testing
    - Write a test rigjt after writing a method
    - Write all methods then write all tests
    - One group write all the code, another writes all the tests
    - Test driven development
      * More advanced technique