Unit: Platform Development Basics - Notes.docx

* Understand which key features make up the Lightning Platform and the Apex programming language
* Identify similarities and differences between .NET and the Lightning Platform
* Use Developer Console to create your first Apex class
* Use Anonymous Apex to invoke a method from an Apex class

Platform Basics

One of the things that separates the platform from other software-as-a-service (SaaS) offerings is that it relies on a metadata-driven architecture. Everything, including the code, configuration, and apps, is specified as metadata.

The Lightning Platform is tightly integrated with the database. You also get all sorts of things, like user interface, security, and reporting, built right in with the platform. This integration is what lets you build apps super fast.

On the Lightning Platform, you don’t have to worry about setting up nodes or management tasks. You don’t have to worry about upgrading, tuning, or scaling anything because instead you can just focus on quickly building great apps.

Apex Basics

Because the Lightning Platform is so tightly integrated and relies on a metadata architecture, you can accomplish an awful lot using declarative development, or what is known as “point-and-click” app building.

Article - [Visual Development – When to Click Instead of Write Code](https://developer.salesforce.com/blogs/engineering/2014/12/forcedotcom-declarative-development.html)

What Is Similar?

The Apex programming language is similar to C# and like C# is object oriented.

Object-oriented Design

Apex supports many of the object-oriented principles that you’re used to such as encapsulation, abstraction, inheritance, and even polymorphism. The Apex language also includes many language constructs you’re already familiar with, including classes, interfaces, properties, and collections.

Data Types

There are primitive types, such as Integer, Double, Long, Date, Datetime, String, and Boolean. There is also an ID data type that is used for any valid 18-character Lightning Platform record identifier assigned by the system.

Value and reference types work the same, but in Apex, all variables are initialized to null by default. In Apex, strings are always treated as a primitive value type.

Besides primitives, supported data types include sObjects, either as a generic sObject or a specific one, such as an Account or Contact. Remember, an sObject is just a Salesforce object. The sObject can be either a standard one that comes built in with Salesforce or a custom one that you define yourself.

Working with Collections

Apex has only the following three collections:

* ***List*** - A list in an ordered collection of elements that works much the same as a traditional array. In fact, arrays in Apex are synonymous with lists, and you can use them interchangeably.
* ***Set*** - A set is an unordered collection of elements that does not contain duplicates. A set is commonly used to store ID values because the value are always unique.
* ***Map*** - A map is a collection of key-value pairs. Each key maps to a single value. A map is useful when you need to quickly find something by a key. The key values must be unique, so you could have a map that contained ID values for the key and then mapped to an sObject.

ASP.NET to Visualforce

Visualforce is a framework for rendering HTML pages using an MVC paradigm. Viewstate is just as much a pain with Visualforce as it is with ASP.NET due to the fact that HTTP is stateless. You can use both ASP.NET MVC and Visualforce to render web pages, and both separate the application logic from the markup and the database model, but they do so in different ways.

What Is Different?

Unlike C#, Apex is not case sensitive.

Apex and Database are Tightly Coupled

* Each standard or custom object in the database has a representation via an Apex class that provides many functionalities to make interacting with the database easy.
* The class and its underlying object are essentially a mirror image of one another that is constantly in sync.

*For instance, whenever you create a new field in an object, a class member is automatically surfaced to reference the values in the database. It's also impossible to add a reference in your Apex code to a field that doesn't exist;*

Different Design Patterns

Most of the design patterns from .NET don’t work on the Lightning Platform.

It’s important to understand that if you try to apply the same design strategies that you use in .NET to the Lightning Platform, you’ll likely encounter problems when you go to test and deploy your solutions.

Unit Tests Are Required

What is different on the Lightning Platform is that you must have 75% test coverage to deploy your Apex code to a production org.

No Solution, Project, or Config Files

* An application on the Lightning Platform is just a loose collection of components, such as tabs, reports, dashboards, and pages.
* All your code resides and executes in the cloud.
* There is also no such thing as a config file in the Lightning Platform world. Because the database is baked right in, you don’t need connection strings.
* And unlike ASP.NET MVC, you don’t need to configure routes.

A Much Smaller Class Library

The Apex class library is considerably smaller than the .NET Framework class library, so it’s easier and faster for you to come up to speed with Apex. The Lightning Platform is built with the idea of providing rapid application development.

Development Tools

[Free Developer Edition (DE) org](https://developer.salesforce.com/signup)

[Salesforce Extensions for VS Code](https://developer.salesforce.com/tools/vscode)

[Get Started with Salesforce DX](https://trailhead.salesforce.com/en/content/learn/trails/sfdx_get_started)

[Salesforce CLI](https://developer.salesforce.com/tools/sfdxcli)

Handling Security

* In the Lightning Platform you don’t have to worry about authentication or storing passwords and database connection strings
* You can control access to data at many different levels, including object level, record level, and field level
* Security is also handled declaratively.

Learn more in the [Data Security trail](https://trailhead.salesforce.com/content/learn/modules/data_security)

What About Integration?

* You can integrate with the platform in a number of ways, but you’ll probably use SOAP and REST the most.
* You can create and expose web services using the Apex programming language, as well as invoke external web services from Apex.
* Salesforce offers both SOAP and REST APIs that provide direct access to the data in your org. Toolkits that wrap around the APIs are available - you can use whatever language you prefer: .NET, Java, PHP, Objective C, Ruby, and JavaScript.

You can learn more about all the integration points by completing the [Apex Integration module](https://trailhead.salesforce.com/content/learn/modules/apex_integration_services).

[Introducing Apex](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro.htm) in the Apex Code Developer’s Guide

**Apex first challenge**

//Create an Apex class that returns accounts

//Create an Apex class that returns a List of Account objects for a user-specified state.

//Create an Apex class that contains a static method:

//Name: AccountUtils

//Method name: accountsByState

//The method must return the ID and name of all Account objects that match the BillingState for the state abbreviation passed to the method

public with sharing class AccountUtils {

// public method

public static void accountsByState(String stateAbbrv) {

List<Account> accountList = [SELECT id, Name FROM Account WHERE BillingState = :stateAbbrv];

return accountList;

}

}