Software Cloud Computing

isobar

Index

Week #4 – Lesson #4

isobar

Review last lesson

isobar

Flows

- Reduce repetitive, labor intensive, and computer related effort;
- Reduce errors and inconsistences in data and process;
- Reduce the cost of maintaining expensive scripts;
- Save your users' time and make sure required tasks are being done;
- Improve the quality of your data;
- Produtivity: provides increase in productivity through automation;
- More fast deployment (can create directly in Prod org);

Flows

- Can execute logic,
- Interact with the Salesforce database,
- Call Apex classes,
- and guide users through screens for collecting and updating data.
- You can do all of this without using any code!
- Can automate repetitive task or process;

Flows

- **Visual coding** – they’re declarative, no coding needed;
- Lots of **automation tools**:
    - Formulas;
    - Validation Rules;
    - Processes;
    - Flows
    - Apex
- Flows are **useful for** two major use cases:
    - Behind the scenes automation;
    - Guided visual experiences;

APEX
What Is?

isobar

APEX

- Object-oriented programming language that uses syntax very similar to Java syntax;
- Add business logic to system events;
- Build complex business processes;
- Customized user interfaces;
- Customize the prebuilt applications;
- and integrations with third-party systems.

APEX

- **Hosted** - Apex is saved, compiled, and executed on the server - the Lightning Platform.
- **Object oriented** - Apex supports classes, interfaces, and inheritance.
- **Strongly typed** - Apex validates references to objects at compile time.
- **Multitenant aware** - Because Apex runs in a multitenant platform, it guards closely against runaway code by enforcing limits, which prevent code from monopolizing shared resources.

APEX

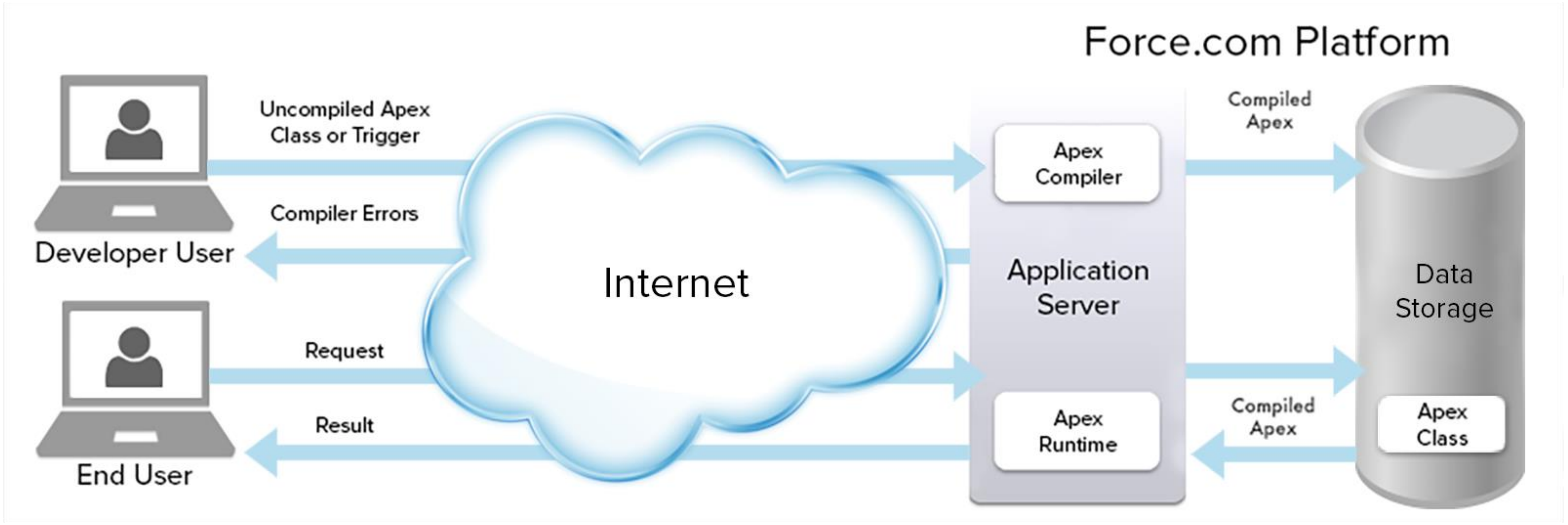
- **Integrated with the database** - It is straightforward to access and manipulate records. Apex provides direct access to records and their fields, and provides statements and query languages to manipulate those records.
- **Data focused** - Apex provides transactional access to the database, allowing you to roll back operations.
- **Easy to use** - Apex is based on familiar Java idioms.

APEX

- **Easy to test** - Apex provides built-in support for unit test creation, execution, and code coverage. Salesforce ensures that all custom Apex code works as expected by executing all unit tests prior to any platform upgrades.
- **Versioned** - Custom Apex code can be saved against different versions of the API.

isobar

APEX


Force.com Platform
Uncompiled Apex Class or Trigger
Compiler Errors
Developer User
Request
Result
End User
Internet
Apex Compiler
Application Server
Apex Runtime
Compiled Apex
Compiled Apex
Data Storage
Apex Class

Invoking Apex

You can run Apex code with triggers, or asynchronously, or as SOAP or REST web services:

- **Anonymous Blocks**: code that does not get stored in the metadata, but that can be compiled and executed.
- **Triggers**: enable you to perform custom actions before or after changes to Salesforce records, such as insertions, updates, or deletions.
- **Asynchronous Apex**: running your Apex code asynchronously (`Queueable`, `Scheduled`, `Batch`, `Future`).
- **Methods as SOAP Web Services**: expose Apex methods as SOAP Web Services so that external applications can access your code and your application.

isobar

Invoking Apex

- **Classes as REST Web Services**: expose your Apex classes and methods so that external applications can access your code and your application through the REST architecture.
- **Apex Email Service**: can use email services to process the contents, headers, and attachments of inbound email.
- **Visualforce Classes**: Apex can also be used to provide custom logic for Visualforce pages through custom Visualforce controllers and controller extensions.
- **JavaScript Remoting**: Use JavaScript remoting in Visualforce to call methods in Apex controllers from JavaScript.
- **Apex in AJAX**: The AJAX toolkit includes built-in support for invoking Apex through anonymous blocks or public `webservice` methods.

Language Highlights

Like other object-oriented programming languages, support:

- **Classes**, **interfaces**, **properties**, and **collections** (including arrays).
- **Object** and array notation.
- **Expressions**, **variables**, and **constants**.
- **Conditional** statements (if-then-else) and **control flow** statements (for loops and while loops).

Language Highlights

Unlike other object-oriented programming languages, support:

- Cloud development as Apex is stored, compiled, and executed in the cloud.
- Triggers, which are similar to triggers in database systems.
- Database statements that allow you to make direct database calls and query languages to query and search data.

Language Highlights

Unlike other object-oriented programming languages, support:

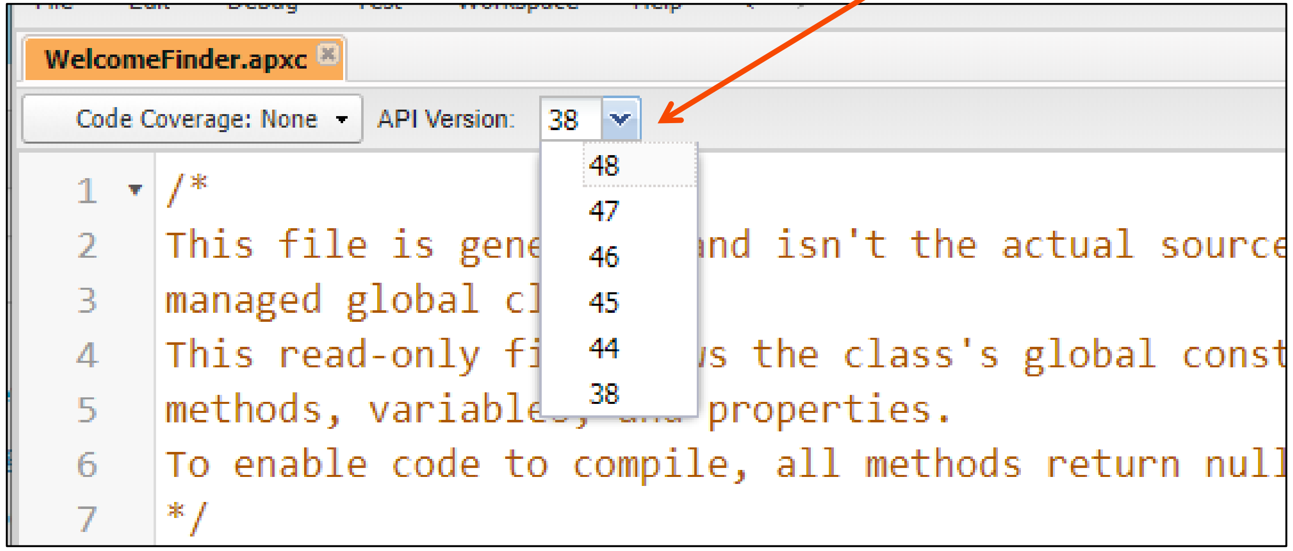
- Transactions and rollbacks.
- The ***global*** access modifier, which is more permissive than the ***public*** modifier and allows access across namespaces and applications.
- Versioning of custom code.
- In addition, Apex is a case-insensitive language.

APEX
Core Concepts

isobar

API Version

Specify the Salesforce.com API (Apex version) against which to save your Apex code.



```
/*
This file is gene        and isn't the actual source
managed global cl
This read-only fi        s the class's global const
methods, variable        properties.
To enable code to compile, all methods return null
*/
```

Data Types

Apex supports various data types, including a data type specific to Salesforce - the **sObject** data type.

- A primitive, such as an **Integer**, **Double**, **Long**, **Date**, **Datetime**, **String**, **ID**, **Boolean**, among others;
- An **sObject**, either as a generic sObject or as a specific sObject, such as an Account, Contact, or *MyCustomObject__c*;

isobar

Data Types

- A collection, including:
  - A **List** (or array) of primitives, sObjects, user defined objects, objects created from Apex classes, or collections;
  - A **Set** of primitives;
  - A **Map** from a primitive to a primitive, sObject, or collection;
- A typed list of values, also known as an **enum**;
- User-defined Apex **classes**;
- System-supplied Apex classes;

Programming Elements

Variable Declaration

SOQL Query

Control Structure

Array (list)

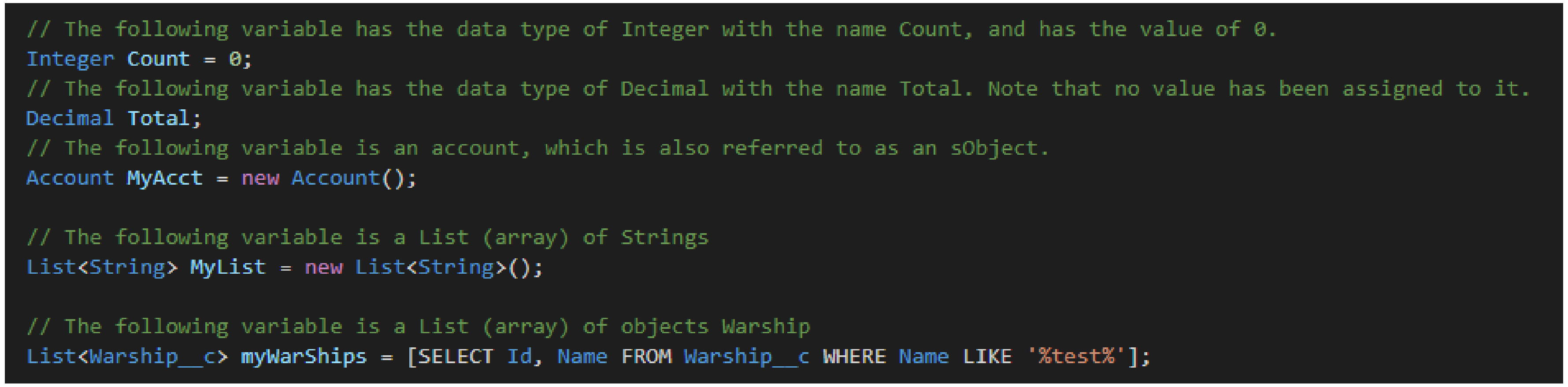
Data (DML) Operation

```
Integer NUM = 10;
Account[] accs;
// Clean up old data
Accs = [SELECT Id FROM Account WHERE name LIKE ‘text%’];
Delete accs;

Accs = new Account[NUM];
FOR (Integer i=0; I < NUM; i++){
   accs[i] = new Account(Name=‘test ‘+I, outstandingshare__c = i);
}
Insert accs;
Contact[] cons = new Contact[0];
FOR (Account acc : accs) {
   cons.add(new Contact(LastName = acc.Name + ‘1’, AccountId = acc.Id));
   cons.add(new Contact(LastName = acc.Name + ‘2’, AccountId = acc.Id));
}
Insert cons;
```

Variables

- Cannot use any of the Apex reserved keywords when naming variables, methods or classes (ex: ***list***, ***test***, ***account***, etc - check *Reserved Keywords** list);
- You must declare the data type of a variable when you first refer to it;
- Variables are declared with a name and a data type (`datatype variable_name [ = value];`) – end statement with a semi-colon;



```
// The following variable has the data type of Integer with the name Count, and has the value of 0.
Integer Count = 0;
// The following variable has the data type of Decimal with the name Total. Note that no value has been assigned to it.
Decimal Total;
// The following variable is an account, which is also referred to as an sObject.
Account MyAcct = new Account();

// The following variable is a List (array) of Strings
List<String> MyList = new List<String>();

// The following variable is a List (array) of objects Warship
List<Warship__c> myWarShips = [SELECT Id, Name FROM Warship__c WHERE Name LIKE '%test%'];
```

* https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_reserved_words.htm

COLLECTIONS

Collections: **`List`**

Apex has the following types of collections:

- Lists (arrays)
- Maps
- Sets;

- A **List** is a collection of elements, such as Integers, Strings, objects, or other collections.
- Use a **List** when the sequence of elements is important.
- You can have duplicate elements in a **List**.
- The first index position in a **List** is always 0.

Collections: List

To create a **List**:

- Use the new keyword
- Use the List keyword followed by the element type contained within <> characters.

Syntax:

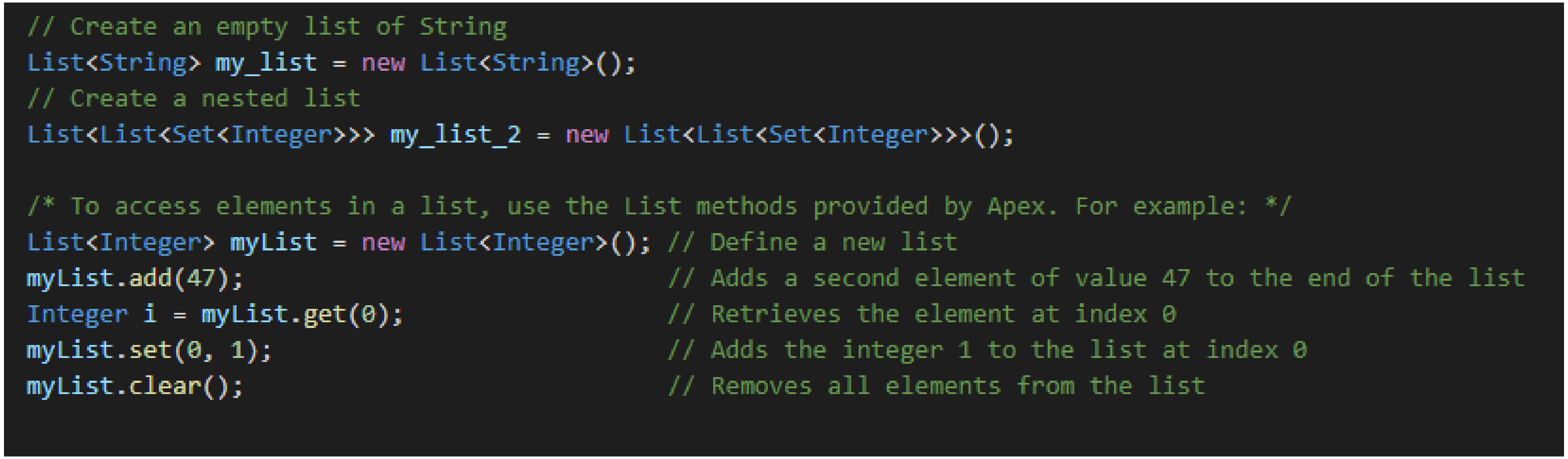
```
List <datatype> list_name
   [= new List<datatype>();] |
   [= new List<datatype>{value [, value2. . .]};] |
   ;
```

Examples:

- `List<Integer> My_List1 = new List<Integer>();`
- `List<Integer> My_List2 = new List<Integer>{1, 2, 3, 4, 5};`

Collections: `List`

Examples:



```
// Create an empty list of String
List<String> my_list = new List<String>();
// Create a nested list
List<List<Set<Integer>>> my_list_2 = new List<List<Set<Integer>>>();

/* To access elements in a list, use the List methods provided by Apex. For example: */
List<Integer> myList = new List<Integer>(); // Define a new list
myList.add(47);                             // Adds a second element of value 47 to the end of the list
Integer i = myList.get(0);                  // Retrieves the element at index 0
myList.set(0, 1);                           // Adds the integer 1 to the list at index 0
myList.clear();                             // Removes all elements from the list
```

Collections: `Set`

- A **Set** is a collection of **unique** collection – use if you want to make sure that your collection should not contain Duplicates;
- A **Set** is an unordered collection – you can’t access a set element at a specific index. You can only iterate over set elements;
- The iteration order of **Set** elements is deterministic, so you can rely on the order being the same in each subsequent execution of the same code;
- It can contains primitive data types, such as Integers, Strings, Date, and so on. Can contain also sObjects;

Collections: Set

To create a **Set**:

- Use the new keyword
- Use the Set keyword followed by the element type contained within <> characters.

Syntax:

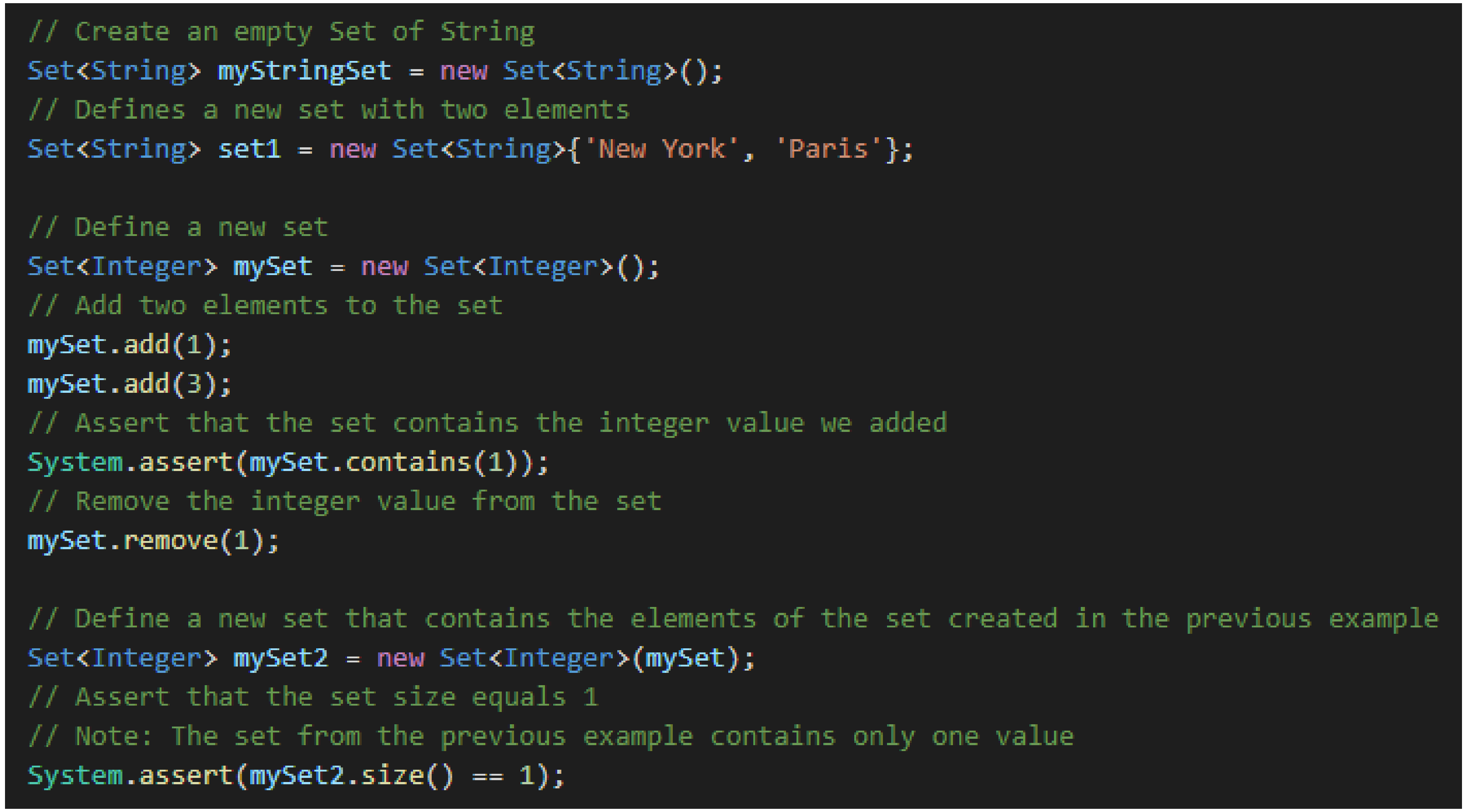
```
Set <datatype> set_name
   [= new Set<datatype>();] |
   [= new Set<datatype>{value1 [, value2...]};] |
   ;
```

Examples:

- `Set<String> My_Set1 = new Set<String>();`
- `Set<String> My_Set2 = new Set<String>{‘a’, ‘b’, ‘c’, ‘d’, ‘e’};`

Collections: Set

Examples:



```
// Create an empty Set of String
Set<String> myStringSet = new Set<String>();
// Defines a new set with two elements
Set<String> set1 = new Set<String>{'New York', 'Paris'};

// Define a new set
Set<Integer> mySet = new Set<Integer>();
// Add two elements to the set
mySet.add(1);
mySet.add(3);
// Assert that the set contains the integer value we added
System.assert(mySet.contains(1));
// Remove the integer value from the set
mySet.remove(1);

// Define a new set that contains the elements of the set created in the previous example
Set<Integer> mySet2 = new Set<Integer>(mySet);
// Assert that the set size equals 1
// Note: The set from the previous example contains only one value
System.assert(mySet2.size() == 1);
```

Collections: Map

- A Map is a collection of key-value pairs where each unique key maps to a single value;
- Keys and values can be any data type – ***primitive types***, ***collections***, ***sObjects***, ***user-defined types***, and built-in Apex types;

Country (Key)	'United States'	'Japan'	'France'	'England'	'India'
Currency (Value)	'Dollar'	'Yen'	'Euro'	'Pound'	'Rupee'

- Map keys and values can contain any collection, and can contain ***nested*** collections;
- You can use the generic or specific sObject data types with Maps.
- You can also create a generic instance of a Map;

Collections: `Map`

- The iteration order of map elements is deterministic;
- It's recommend to always access map elements by key;
- A map key can hold the `null` value;
- Adding a map entry with a **key** that *matches* an **existing key** in the **Map** *overwrites* the *existing entry* with that key with the new entry;
- **Map** keys of type String are case-sensitive;
- A Map object is serializable into JSON only if it uses one of the following data types as a key:
    - Boolean; Date; DateTime; Decimal; Double; Enum; Id; Integer; Long; String; Time.

isobar

Collections: Map

To create a **Map**:

- Use the new keyword
- Use the Map keyword followed by a key-value pair, delimited by comma and enclosed in <> characters.

Syntax:

```
Map <key_datatype, value_datatype> map_name
   [= new Map< key_datatype, value_datatype>();] |
   [= new Map< key_datatype, value_datatype>{key1 => value1 [, key2 => value2…]};] |
   ;
```

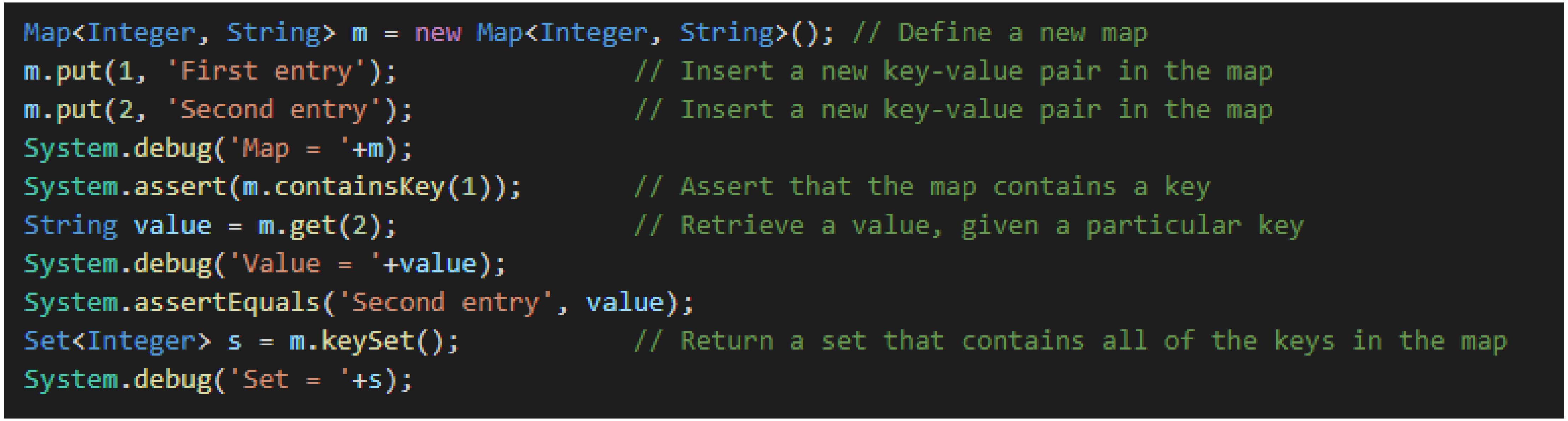
Collections: Map

Examples:

- `Map<Integer, String> My_Map1 = new Map<Integer, String>();`
- `Map<Integer, String> My_Map2 = new Map<Integer, String>{1=>’a’, 2=>‘b’};`
- `Map<Id, Account> My_Map3 = new Map<Id, Account>([SELECT Id, Name, Type FROM Account WHERE Id IN: accIdList]);`

Collections: Map

Examples:



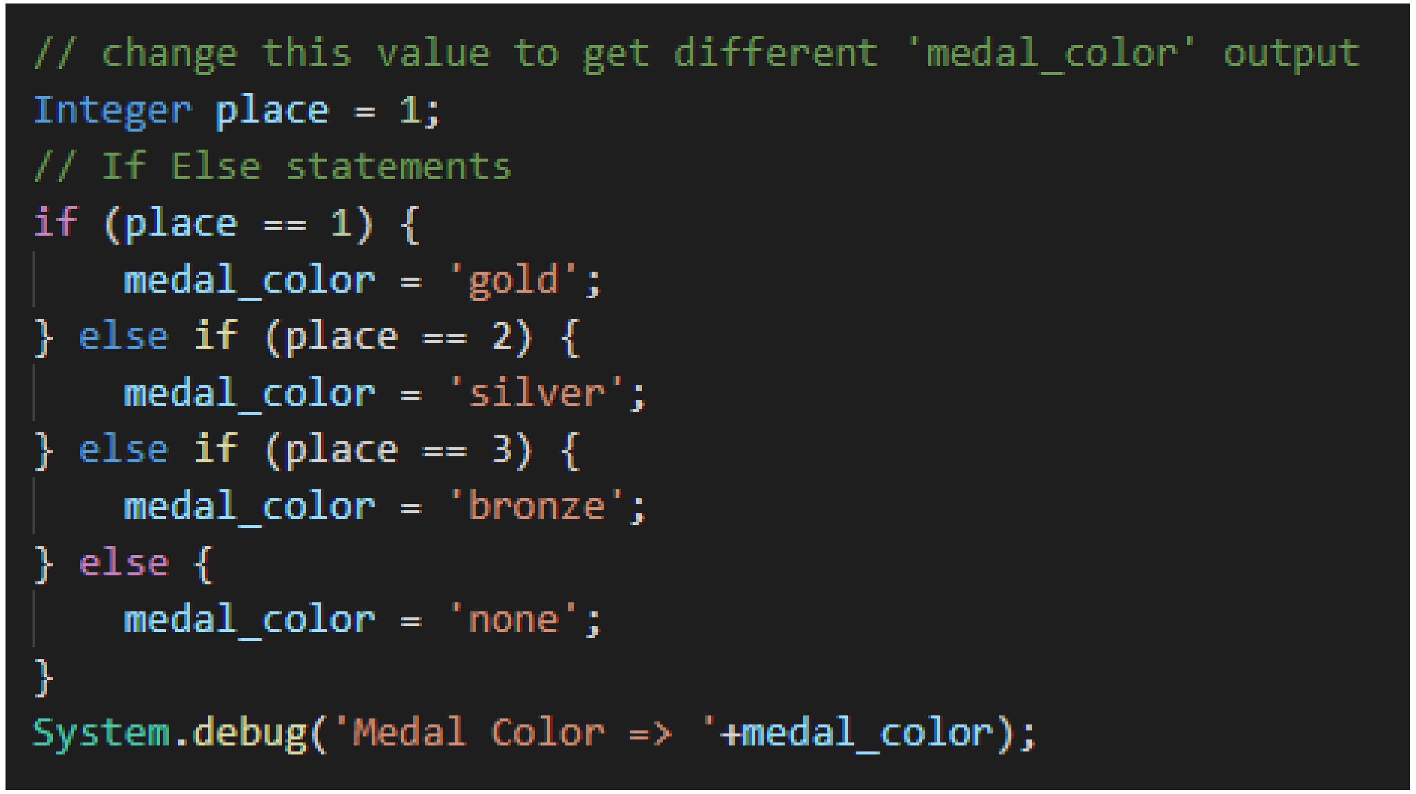
```
Map<Integer, String> m = new Map<Integer, String>(); // Define a new map
m.put(1, 'First entry');               // Insert a new key-value pair in the map
m.put(2, 'Second entry');              // Insert a new key-value pair in the map
System.debug('Map = '+m);
System.assert(m.containsKey(1));       // Assert that the map contains a key
String value = m.get(2);               // Retrieve a value, given a particular key
System.debug('Value = '+value);
System.assertEquals('Second entry', value);
Set<Integer> s = m.keySet();           // Return a set that contains all of the keys in the map
System.debug('Set = '+s);
```

CONDITIONAL STATEMENTS

Conditional Statements: If-Else

```
If ([boolean_condition])
   // Statement when true
Else
   // other Statement
```

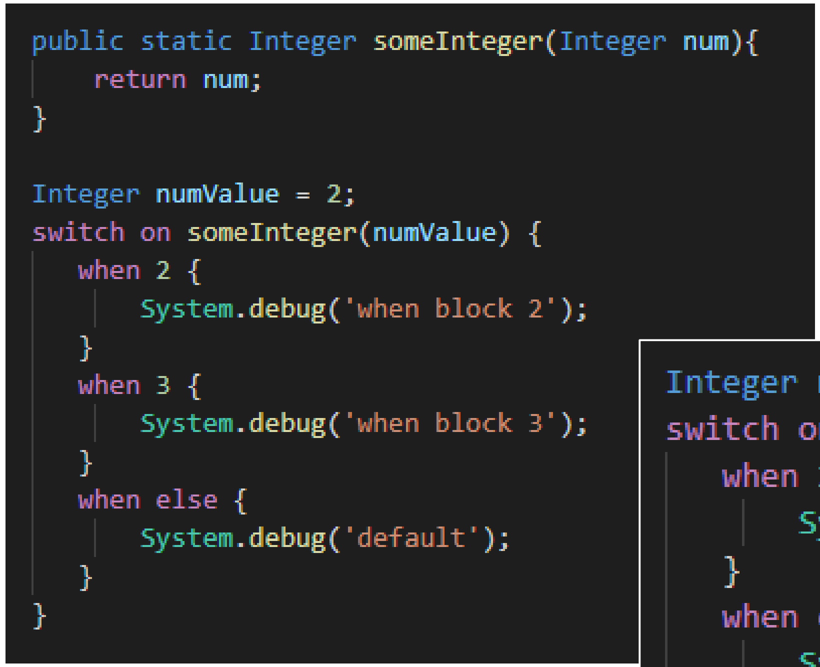
```
If ([boolean_condition])
    // Statement when true
Else If ([boolean_condition])
    // other Statement true
Else
    // other Statement
```



```
// change this value to get different 'medal_color' output
Integer place = 1;
// If Else statements
if (place == 1) {
    medal_color = 'gold';
} else if (place == 2) {
    medal_color = 'silver';
} else if (place == 3) {
    medal_color = 'bronze';
} else {
    medal_color = 'none';
}
System.debug('Medal Color => '+medal_color);
```

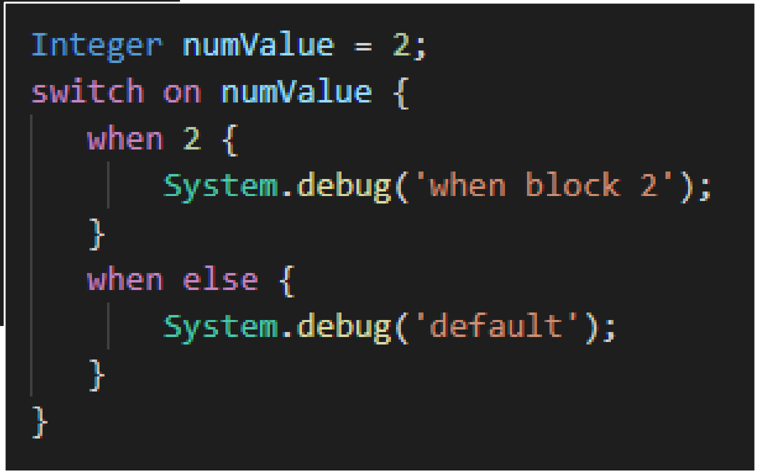
Conditional Statements: switch

```
switch on expression {
   when value1 {
     // code block 1
   }
   when value2, value3 {
     // code block 2
   }
   when TypeName VariableName {
     // code block 3
   }
   when else {
     // code block 4
   }
}
```



```
public static Integer someInteger(Integer num){
    return num;
}

Integer numValue = 2;
switch on someInteger(numValue) {
   when 2 {
       System.debug('when block 2');
   }
   when 3 {
       System.debug('when block 3');
   }
   when else {
       System.debug('default');
   }
}
```



```
Integer numValue = 2;
switch on numValue {
   when 2 {
       System.debug('when block 2');
   }
   when else {
       System.debug('default');
   }
}
```

LOOPS

Loops: Do-While

The Apex **do-while** loop repeatedly executes a block of code as long as a particular Boolean condition remains true, but does not check the Boolean condition statement until after the first loop is executed.

Syntax:

```
do {
    code_bock
} while (condition);
```

Example:

```
Integer count = 1;
do {
    System.debug(count);
    count++;
} while (count < 11);
```

Loops: While

The Apex **while** loop repeatedly executes a block of code as long as a particular Boolean condition remains true, but checks the Boolean condition statement before the first loop is executed.

Syntax:

```
while (condition) {
    code_bock
};
```

Example:

```
Integer count = 1;
while (count < 11) {
    System.debug(count);
    count++;
}
```

isobar

Loops: For

The **traditional** for loop:

Syntax:

```
for (init_stmt; exit_condition; increment_stmt) {
    code_bock
}
```

Example:

```
for (Integer i = 0; i < 10; i++) {
    System.debug(i+1);
}
```

isobar

Loops: For

The **list** or **set** iteration for loop:

Syntax:

```
for (variable: list_or_set){
    code_bock
}
```

Example:

```
Integer[] myInts = new Integer[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for (Integer i : myInts) {
    System.debug(i);
}
```

Loops: For

The **SOQL** for loop:

Syntax:

```
for (variable: [soql_query]){
     code_bock
}
```

Example:

```
String s = 'Acme';
for (Account a : [SELECT Id, Name FROM
Account WHERE Name LIKE :(s+'%')]) {
    System.debug('Account Name = '+
a.Name + ' (Id:'+a.Id+') ');
}
```

```
// Create a list of account records from
a SOQL query
List<Account> accs = [SELECT Id, Name
FROM Account WHERE Name = 'Siebel’];
// Loop through the List and update Name
for (Account a : accs) {
    a.Name = ‘Oracle’;
}
update accs; // Update database
```

isobar

CLASSES

Apex Classes

What Does Object-Oriented Mean?

Object-oriented means that the code focuses on describing objects.

An object can be anything that has unique characteristics, such as a person, an account, or even a flower.

Before you can truly understand what object-oriented means, there are two things that you need to understand: **classes** and **objects**.

isobar

Apex Classes

Classes

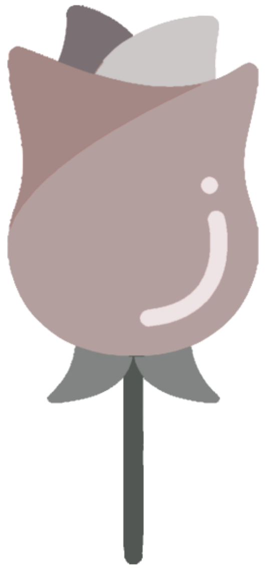
A ***class*** is a blueprint.

A class defines a set of **characteristics** and **behaviors** that are common to all objects of that class.

Objects are based on classes.

Characteristics are called ***variables*** and the **Behaviors** are called ***methods***.

Apex Classes



Characteristics (Variables)	Behaviors (Methods)
color	Grow
height	Pollinate
maxHeight	Wilt
numberOfPetals	

isobar

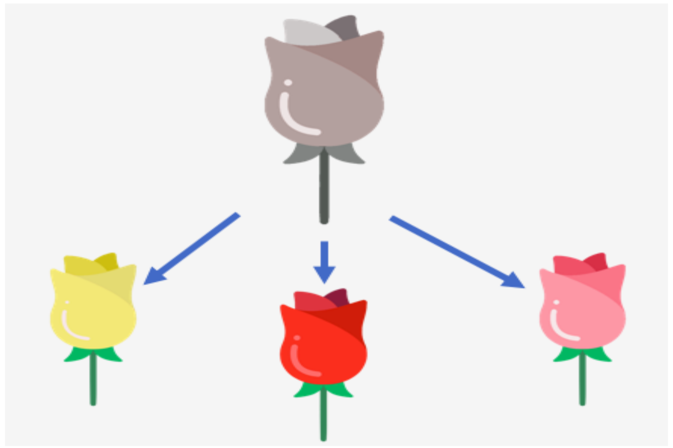
Apex Classes

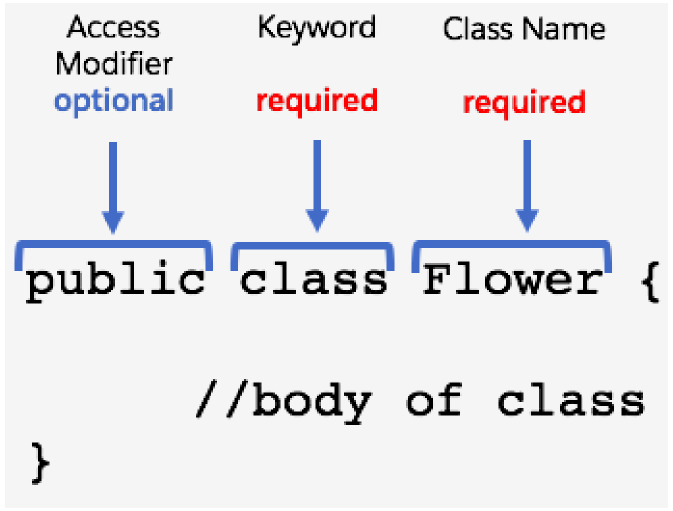
Declaring a Class

Classes are declared using four parts: the *access modifier*, the *keyword* "class", the *class name*, and the *class body*. The body (inside the curly braces **{ }**), is where *methods* and *variables* of the class are defined.

Access Modifier

An access *modifier* is a keyword in a class or method declaration. The access modifier determines what other Apex code can see and use the class or method.





```
public class Flower {
        //body of class
}
```

Apex Classes

Declaring a Class: Syntax

```
private | public | global
[virtual | abstract | with sharing | without sharing]
class ClassName [implements InterfaceNameList] [extends ClassName]
{
  // The body of the class
}
```

`private` declares that this class is only known locally, that is, only by this section of code.

The `public` access modifier declares that this class is visible in your application or namespace.

The `global` access modifier declares that this class is known by all Apex code everywhere.

Apex Classes

Declaring a Class: Syntax

The `with sharing` and `without sharing` keywords specify the sharing mode for this class.

For more information: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_keywords_sharing.htm

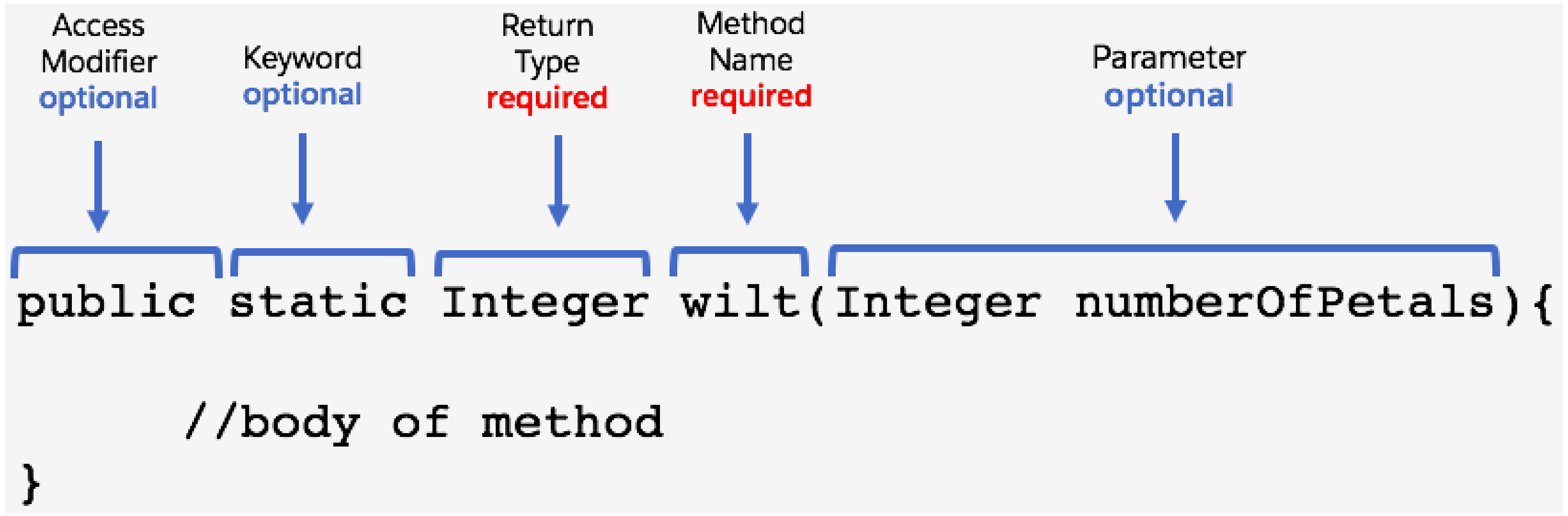
The `virtual` definition modifier declares that this class allows extension and overrides. You cannot override a method with the override keyword unless the class has been defined as virtual.

The `abstract` definition modifier declares that this class contains abstract methods, that is, methods that only have their signature declared and no body defined..

Apex Classes

Methods

Methods are defined within a class. A method describes the behaviors inherited by objects of that class. A class can have one or more methods. The Flower class has three methods (behaviors): **grow**, **pollinate**, and **wilt**.



```
public static Integer wilt(Integer numberOfPetals){

     //body of method
}
```

Apex Classes

Methods: Syntax

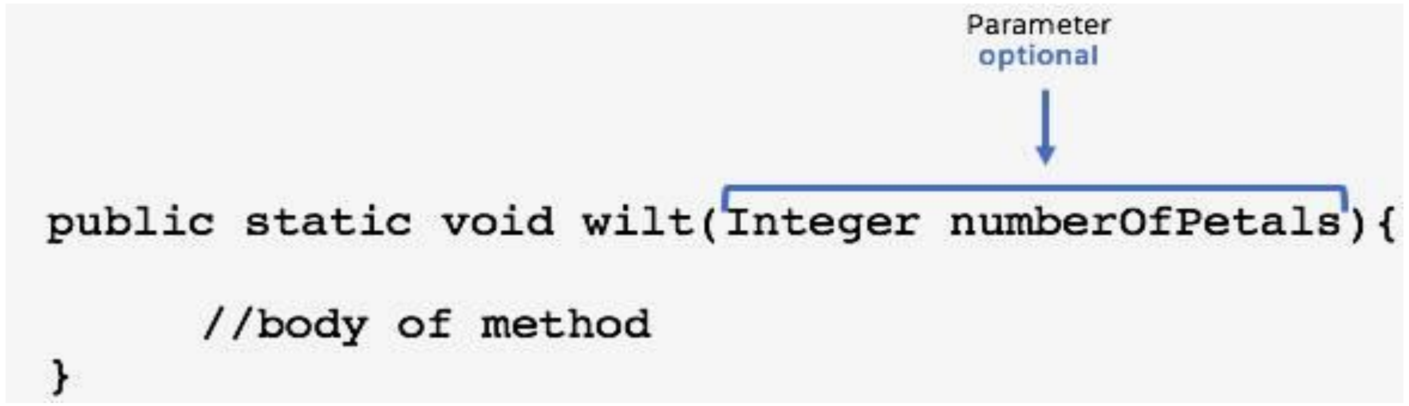
```
[public | private | protected | global] [override] [static] data_type
method_name
(input parameters)
{
   // The body of the method
}
```

Apex Classes

Parameters

A parameter is a variable that serves as a placeholder, waiting to receive a value.

Parameters are declared similarly to a variable, with a *data type* followed by the *parameter name*.



```
public static void wilt(Integer numberOfPetals){

      //body of method
}
```

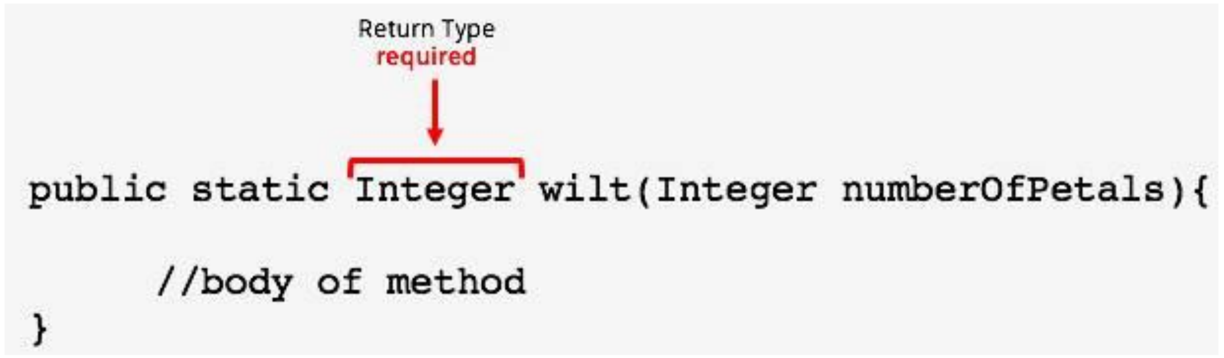
Apex Classes

Return Types

A Method declaration must explicitly state its expected *return type*.

A *returned variable* must *match* the *return type* specified by the method declaration.

The return type is a specific data type, such as **Boolean**, **String**, or **Account**, or it can be ***void*** (nothing), etc.



```
public static Integer wilt(Integer numberOfPetals){
      //body of method
}
```

isbar

References

Multi Tenant Architecture:
https://developer.salesforce.com/page/Multi_Tenant_Architecture

Developer Centers / Developer Experience:
https://developer.salesforce.com/developer-centers/developer-experience/

Apex Developer Guide
https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm

Apex: Data Types and Variables
https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_datatypes_variables_intro.htm

Apex: Classes, Objects and Interfaces
https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes.htm

References

- Object-Oriented Programming for Admins: https://trailhead.salesforce.com/en/content/learn/modules/object-oriented-programming-for-admins/create-classes-and-objects
- Quick Start: Apex https://trailhead.salesforce.com/en/content/learn/projects/quickstart-apex
- Quick Start: Apex Coding for Admins https://trailhead.salesforce.com/en/content/learn/projects/quick-start-apex-coding-for-admins
- Apex Testing https://trailhead.salesforce.com/en/content/learn/modules/apex_testing

isobar

isobar