# SalesForce Development

## Module 2

## Apex Fundamentals

# Topics to be covered

- Variables in Apex
- Apex Programs
- Data Types in Apex
- String Methods
- Expressions and Operations
- VS Code

# Variables in Apex

# Variables in Apex

- A variable is a container which holds the value while program is executed
- Local variables are declared with Java-style syntax.

- Ex: Variable declaration

| | | | |
|---|---|---|---|
| Integer | age | = | 20; |
| (DataType) | (Variable Name) | | (Value) |

  - In above example **Integer** is a DataType, **age** is a variable name and **20** is a value of that variable

# Variables in Apex

- You can declare variable and then assign value to it:
  - `integer age;`
  - `age = 20;`
- Multiple Variables can be declared and initialized in a single statement.
  - `integer a,b,c;`
- If you declare a variable and don't initialize it's value, then it will be null.
  - `integer dd;`
- You can also assign null to the variable as well:

  `integer numberOfPencils = null;`

# Variables in Apex

- Apex variables are case-insensitive .
- Ex.

```
integer a = 10;
integer A = 20;
```

  - Here Apex consider ‘a’ and ‘A’ as same variables.

- In Apex, it is not allowed to redefine the same variable again in code block.

# Variables in Apex

- **Code :**

Enter Apex Code

```
integer balls = 5;
Integer balls = 10;
```

- This code throw an error as “Duplicate field”.

**Error**:

# Comments

- To exclude part of the code from executing you can comment it out.
- To comment a code out we use: //
- In the below example only Hi will print.

Enter Apex Code

```
1  //System.debug('Hello');
2  System.debug('Hi');
```

# Syntax rules to watch out

- Apex is case insensitive. A = a
- Every statement ends with semicolon ;
- Apex reads and executes our statements: Left to Right, Top to Bottom
- A variable should be declared before use
- Do not declare variable twice
- A variable that are not initialized will be null by default
- Do not leave space between code

# Adding comments in Apex

- Comments are text notes that are added to the code to give explanations for the source code.
- They are used in a programming language to document the code.
- They also help in the understanding and maintenance of code for other teammates.
- These are considered non-executable statements by the compiler.

# Adding comments in Apex

- **Single line Comments :**
  - All characters on the same line to the right of the // are ignored by the parser.

```
Integer myInt = 50;// This line will be igniored by compiler
```

- **Multiline Comments :**
  - All characters between '/*' and '*/' are ignored by the parser.

```
/* A Multiline Comment --
        Define a variable of string type
        and assign value to it.
*/
String msg = 'Hello World';
```

# Escape characters in Apex

- Backslash character (\) is a special character and gets treated differently inside the String.
- It is used to escape characters in column names and string values in a predicate expression.
- \' escape sequence to escape a single quote in a column name.
  - Ex. Country\'s Name/' == "India"

# Escape characters in Apex

| | |
|---|---|
| \n | New line |
| \t | Tab |
| \\ | Single backslash |
| \’ | Single Quote |
| \” | Double Quote |

# Escape characters in Apex

```
String str = 'Alex\nBiden' ;
System.debug('str ::::::::: ' + str) ;
```

| Timestamp | Event | Details |
|---|---|---|
| 20:57:16:004 | USER_DEBUG | [2]\|DEBUG\|str ::::::::: Alex |
| 20:57:16:000 | USER_DEBUG | Biden |

```
String str = 'Alex\\Biden' ;
System.debug('str ::::::::: ' + str) ;
```

| Timestamp | Event | Details |
|---|---|---|
| 21:00:49:003 | USER_DEBUG | [2]\|DEBUG\|str ::::::::: Alex\Biden |

```
String str = 'Alex \"Biden\"' ;
System.debug('str ::::::::: ' + str) ;
```

| Timestamp | Event | Details |
|---|---|---|
| 21:04:21:002 | USER_DEBUG | [2]\|DEBUG\|str ::::::::: Alex "Biden" |

# Assignment

- Create an Integer variable and assign a value 10 to it and print it using Anonymous Window.
- Print below text from developer console:
  “Salesforce “assignments” are
  too \ easy !!”

# Apex Programs

# Apex Programs

- Creating Programs in Apex is equivalent to creating Apex Classes
- Unlike Anonymous Block, Codes written in Programs are reusable
  - Apex Programs (Classes) are saved and available in our org for future use.
    - *Go to Org Setup Page → Apex Classes*
  - Code written in Anonymous block are temporary, once a new code is written in it it will loose the old ones

# Apex Programs

- Each Apex Class should contain at least one method to provide codes
- To run the program just follow the basic syntax of method for now

```
public static void <method_name>(){
}
```

- To create an Apex Class
  1. *File → New → Apex Class*
  2. *Provide Name to the Apex Class*
  3. *Click Ok*

# Apex Programs

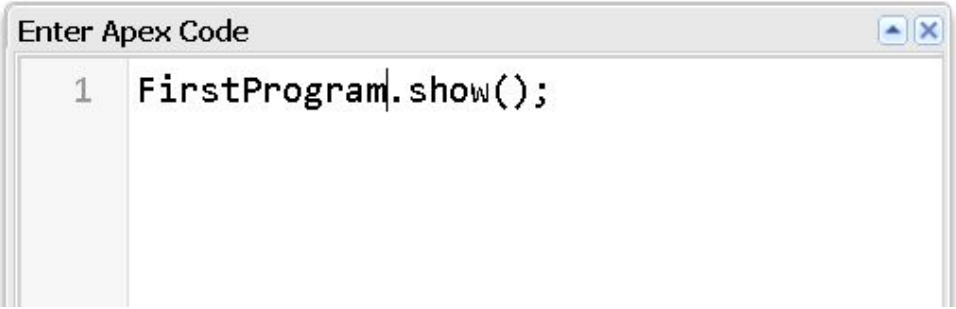
FirstProgram.apxc

Code Coverage: None API Version: 55

```
1 public class FirstProgram {
2     public static void show(){
3         Integer var = 10;
4         System.debug(var);
5     }
6 }
```

# Apex Programs

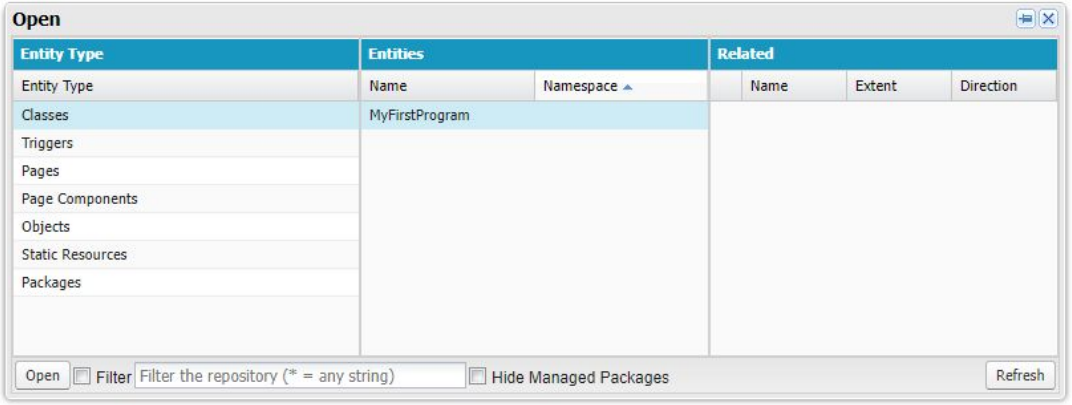
- Save your class by going File → Save
- To Run the Program
  - Debug → Open Execute Anonymous Window
  - Call the method of the Apex Class with Class Name (For example: We call **debug** method of **System** Class)
  - Click on Execute



# Open Saved Class

To Open saved class:

- File → Open → Classes → Double click <*Your Class Name*>



# Assignment

- Create an Apex Class with name “MyClass1”, create two integer variables in it, assign different values to them, print these values. Close the Class or Program.
- Create an Apex class called: “MyClass2”, create an integer variable, assign a value to it and print it, now increase the value of the variable by 10 and print the new value.
- Open the Apex class “MyClass1”, create a third integer variable in it, add the two variables that you have created earlier and store the result in the third variable and print it as well.

# Homework 1

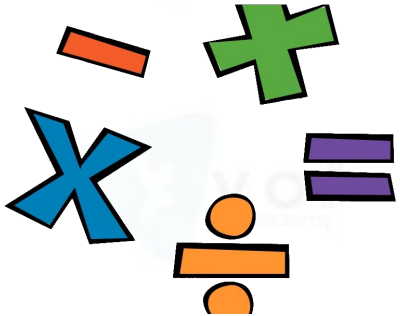
- Create an Apex class named “MyClass4”.
- Create 2 Integer variables in it.
- Assign a value to one of the variable.
- Make the second variable 2 times the value of the first variable.
- Print the variables.
- Execute the program to see the output.

# Assignment Operator

- Need to pay attention to the usage of = assignment operator
- If = sign is not used that means the value of the variable hasn't changed
  - ```
    Integer a = 10;
    Integer b = 20;
    System.debug(a+b);
    ```
    ⟶ This doesn't change the value of a or b.
- You can assign value of one variable to another
  - ```
    Integer a = 10;
    Integer b = 20;
    Integer a = b;
    System.debug(a);
    ```
    ⟶ 20;
  - Now both a and b have the value of 20.

# Basic Arithmetic Operators

- + → Addition
  - ```
    Integer total = num1 + num2;
    Integer num = 30 + 10;
    Integer num = num1 + 5;
    ```
- - → Subtraction
  - ```
    Integer total = num1 - num2;
    Integer num = 30 - 10;
    Integer num = num1 - 5;
    ```
- * → Multiplication
  - ```
    Integer total = num1 * num2;
    Integer num = 30*10;
    Integer num = num1 * 5;
    ```
- / → Division
  - ```
    Integer total = num1 / num2;
    Integer num = 30/10;
    Integer num = num1 / 5;
    ```



# Data Types

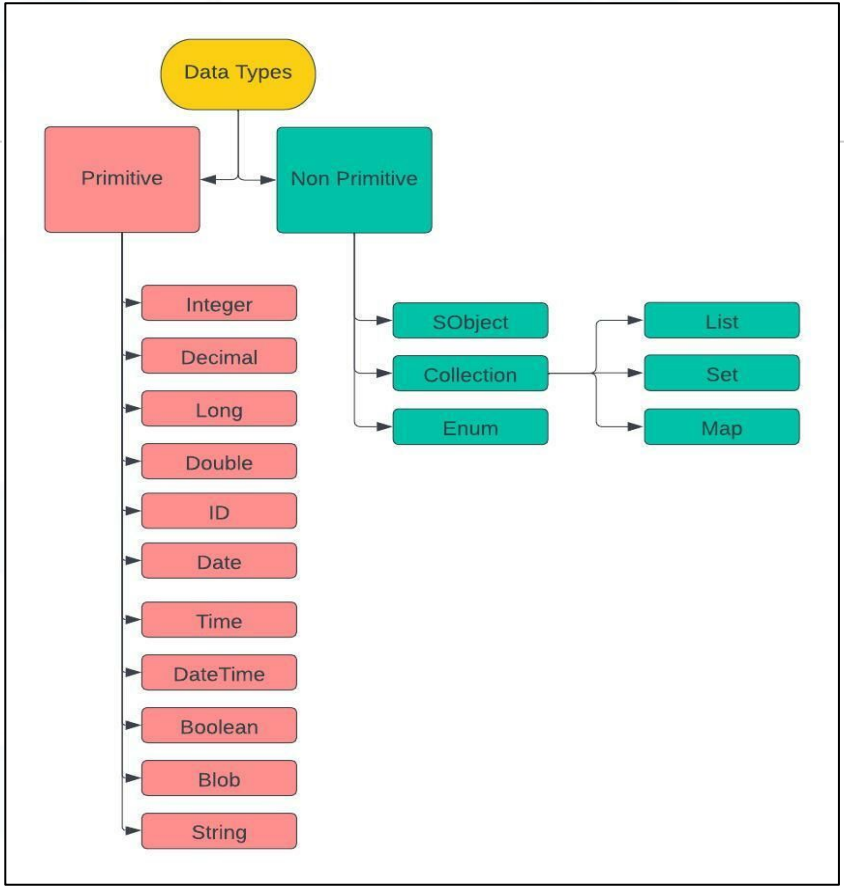
# Data Types in Apex

- Because the Apex language is **Strongly Typed**, each variable in Apex will be defined with the appropriate data type.
- Initially, all apex variables are set to null.
  - Below code will produce **null** as result

Enter Apex Code

```
integer a;
system.Debug(a);
```

- It is recommended to ensure that appropriate values are provided to variables.
  - Otherwise, using such variables will result in **null pointer exceptions** or other **unhandled exceptions**.


Data Types
Primitive
Non Primitive
Integer
Decimal
Long
Double
ID
Date
Time
DateTime
Boolean
Blob
String
SObject
Collection
Enum
List
Set
Map

# Data Types in Apex

- **Primitive**
  - Integer, Double, Decimal, Long, Date, Datetime, String, ID,Blob ,Boolean
- **Collections**
  - Lists, Sets and Maps
- **sObject**
- **Enums**
- **Classes, Objects and Interfaces**

# Integer

- **Integer :**
  - A 32-bit primitive number that does not include a decimal point.
  - Integers have a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647.


```
Integer rollNumber = 50;
System.debug('Value of rollNumber is ' + rollNumber);
```

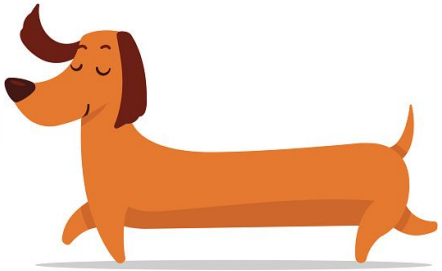
**Output:** Value of rollNumber is 50

# Long

- **Long:**
  - A 64-bit number that does not include a decimal point.
  - Longs have a minimum value of -2^(63) and a maximum value of 2^(63)-1 .
  - By default all whole numbers are considered as Integers
  - Therefore, we need to put L or l to the end of the Long value to specify that it is long

```
Long ll = 2147483648L;
System.debug(' Value of ll is : '+ll);
```

**Output:** Value of ll is 2147483648L



Below code will throw error.

Enter Apex Code

```
Long balls = 5324234234235;
System.debug(balls);
```

# Double

- **Double :**
  - It is 64 bit number(15-16 digit), which accept decimal value.
    - Ex. 1 is Integer and 1.0 is double.
  - By default  all decimal numbers are considered as double

```
double litreOfWater = 25.5;
System.debug(' Liters of water in jug is : '+litreOfWater);
```

**Output:** Liters of water in jug is 25.5

# Division

- When dividing 2 integers result will be integer by default and can be incorrect.
  - `System.debug(20/6);` //we expect this: 3.333333 --> but we get 3

- To solve this problem, we have to use decimal points in one of the sides.
  - Any of the below combinations will work
  - `System.debug(20.0/6);` //we get 3.333..
  - `System.debug(20/6.0);` //we get 3.333..
  - `System.debug(20.0/6.0);` //we get 3.333..

# Decimal

- **Decimal :**
  - It is another data type that is used for numbers with decimal point.
  - It comes with lot of built-in methods and rounding options.
  - Decimal are used in currencies by default.
  - Unlike double, the number of digits is not fixed for it.
    - If we won't explicitly set the number of places for a decimal, then the item from which it is created decides the places.
    - In comparison, for **Double** the maximum number of decimal places is **16**.

```
Double var1 = 2.0/3.0;
Decimal var2 = 2.0/3.0;

System.debug('For Double => '+var1);
System.debug('For Decimal => '+var2);
```

Details

[4]|DEBUG|For Double => 0.6666666666666666

[5]|DEBUG|For Decimal => 0.6666666666666666666666666666667

# Decimal

- **Decimal :**
  - We can set the number of decimal places.
  - setScale method is used to perform this task.
  - It performs the rounding when we call this method.
  - Also Decimal has a lot of built-in methods and rounding options, so it's helpful to use this instead of double

Enter Apex Code

```
Decimal var = 2.56789;
Decimal var1 = var.setScale(2);
Decimal var2 = var.setScale(3);

System.debug(var);
System.debug(var1);
System.debug(var2);
```

| Details |
| --- |
| [5]\|DEBUG\|2.56789 |
| [6]\|DEBUG\|2.57 |
| [7]\|DEBUG\|2.568 |

# Assignment

- Write a program in Apex, with a variable to store your age and print the following statement.

  ***I am <AGE> years old.***

- Write a program in Apex, with a variable to store '7977148450' and print the following statement.

  ***Currently there are more than 7977148450 people in the world.***

- Write a program in Apex, with a variable to store your height in cm (like 165.2) and print the following statement.

  ***I am <HEIGHT> cm long.***