2.3.4.8.4.7 Visibility

Return to Operation Data

Overview

Return to Top

Visibility is a major feature of Object-Oriented Programming (OOP) allowing architects and engineers to design rules covering the visibility of an Object's state variables and operations to other objects within the application or system.

For instance, to prevent a certain **`State Variable`** to be modified from outside the **`class`**. The default **Visibility** is for many OOP Languages (PHP, C++. Java, etc.) are often **`public`** meaning the class members are accessible from anywhere (internal or external objects). In C++, there are two ways to declare objects: **`class`** and **`struct`**. The main difference is the assumption of the member **Field** and **Operational** Data visibility. In C++, in **`struct`**s, all **Field** and **Operational** data are by default **`public`** unless otherwise specified. In a **`class`**, all **`Field`** and **`Operational`** Data are considered **`private`** unless specified differently.

Visibility of a **`class`** member determines where the **Field** or **Operational** Data is accessible or modifiable, or from where the member function can be called.

Visibility	Description
public	The member variable/function can be accessed/called in any statement within any function.
protected	The member variable/function can only be accessed/called in the statements of functions of the class to which the variables/functions belong and any derived classes (subclasses).
private	The member variable/function can only be accessed/called in the statements of member functions of the class to which the variables/functions belong.

> **Note: Visibility** is generally implemented by adding labels to the member variables or member operations.

Figure 1, graphically tries to represent the various **Visibility** characteristics found in OOP. **`public`**, **`internal`**, and **`private`** Visibility attributes can be associated with **`State Variables`** or with **`Functions`**.

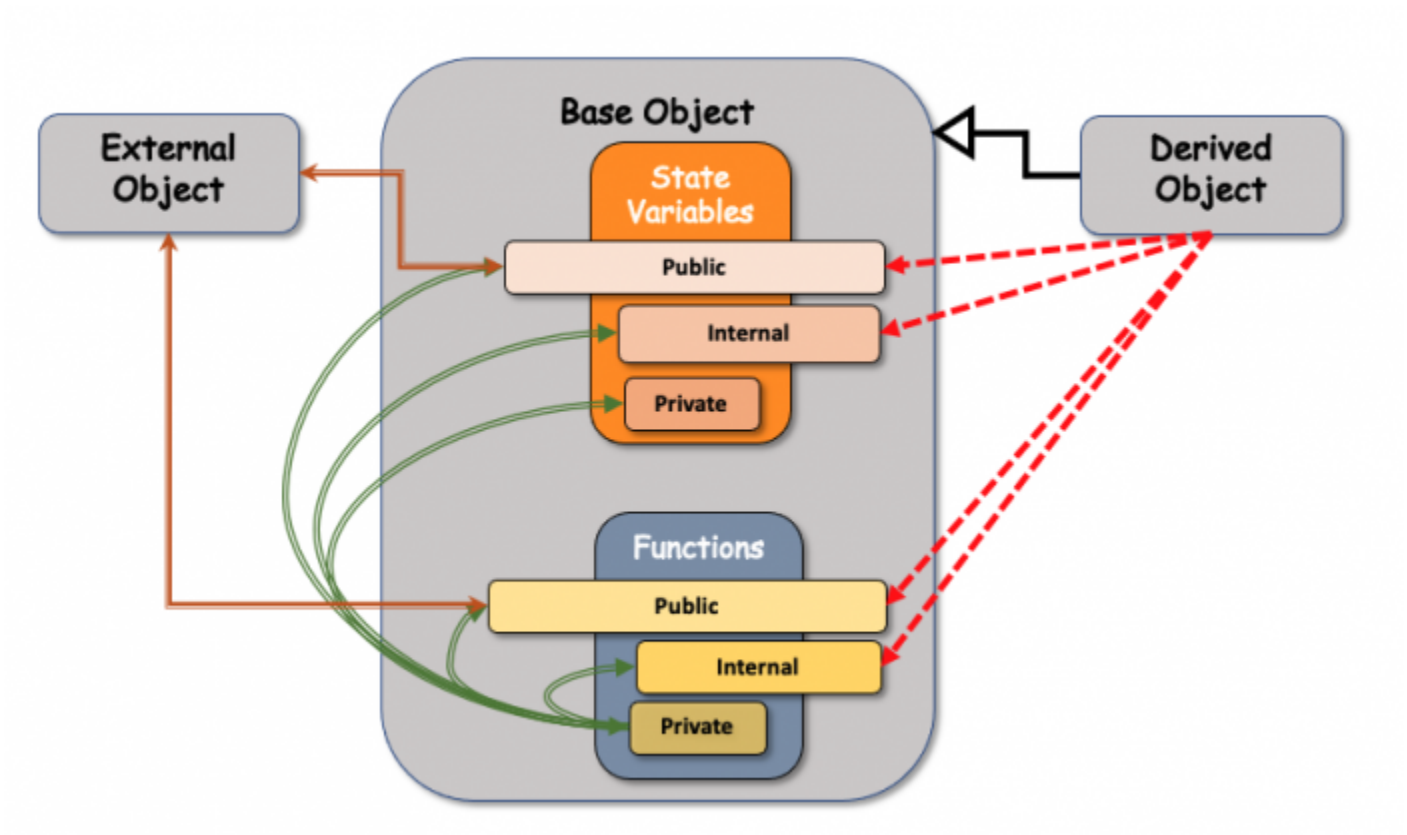


Figure 1: The Object-Oriented Programming (OOP) Visibility Kinds

See the following on C++ and visibility[1)].

An example of using **Visibility** labels in a simple C++ program.

```
// C++ implementation to show
// Visibility modes
#include <bits/stdc++.h>
using namespace std;
class BaseClass
{ //===== Public
  public:
    int x;
  //===== Protected
  protected:
    int y;
  //===== Private
  private:
    int z;

}; // End BaseClass
// DerivedClass inherits from BaseClass
class DerivedClass : public BaseClass
{
}; // End DerivedClass
```

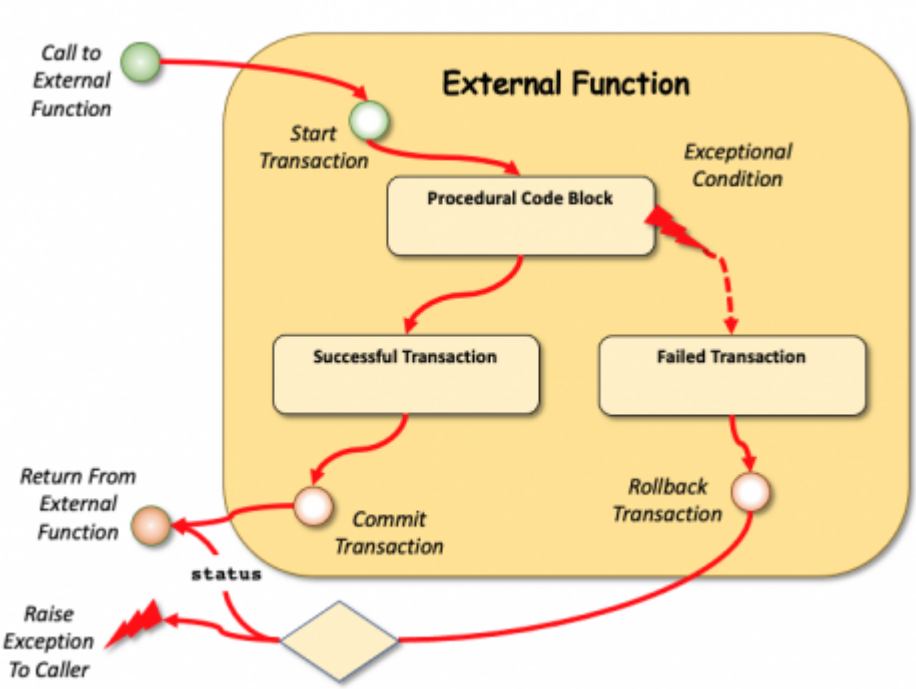
```
// main function
int main()
{
    DerivedClass derivedClass;
    // x is labeled public, therefore it's value is displayed in the standard
output stream
    cout << derivedClass.x << endl;
    // y is labeled protected, therefore the compiler displays a visibility
error
    cout << derivedClass.y << endl;
    // z is labeled private within BaseClass, therefore the compiler displays
a visibility error
    cout << derivedClass.z << endl;

}; // End main
```

> **Note:** Often the coding styles specify the attributes with a particular label be listed together (i.e., all public state variables listed together, etc.) and they are listed alphabetically within the group. The Labels are listed from least restrictive to most restrictive (i.e., **public**, **protected**, and **private**)

DIDO Specifics

Return to Top

Solidity has similar visibility to those used in more traditional Object-Oriented Programming (OOP), however, it introduces a new visibility attribute **External**. All the regular OOP **Visibility** attributes are applicable to **Functions** and **State Variables**. The **External** visibility attribute can only be used with **functions**[2)].

Visibility	Description
external	External functions are part of the contract interface, which means they can be called from other contracts and **via transactions**. An external function **f** cannot be called internally (i.e. **f()** does not work, but **this.f()** works). When an **external** function is called, a transaction block is automatically started covering all the statements in the body of the function. If the block of code in the function executes without exception, control is returned to the calling contract. If any statement raises an exception, then all the changes made to the **state variables** are reverted to the original values existing before the transaction was started. **Note:** Depending on how the external function was called, either a **status** is returned to the calling contract or an exception is raised to the caller (i.e., and the process is repeated for that contract).  Figure 2: External Functions imply a Transaction.
public	Public functions are part of the contract interface and can be either called internally or via messages. For public state variables, an automatic getter function (see below) is generated.
internal	Those functions and state variables can only be accessed internally (i.e. from within the current contract or contracts deriving from it), without using this. This is the default visibility level for state variables.
private	Private functions and state variables are only visible for the contract they are defined in, and not in derived contracts.

> **Note:** All data on a public blockchain should be considered **public**, even **private** state variables. **Private** state variables are not available for another contract to read, however, these variables can be read using web3.js. **Never store sensitive data on the public blockchain**. [3].

Figure 3, graphically tries to represent the various **Visibility** characteristics found in Solidity. **public**, **internal**, and **private** Visibility attributes can be associated with **State Variables** or with **Functions**, however, the **external** visibility attribute can only be associated with **functions**.

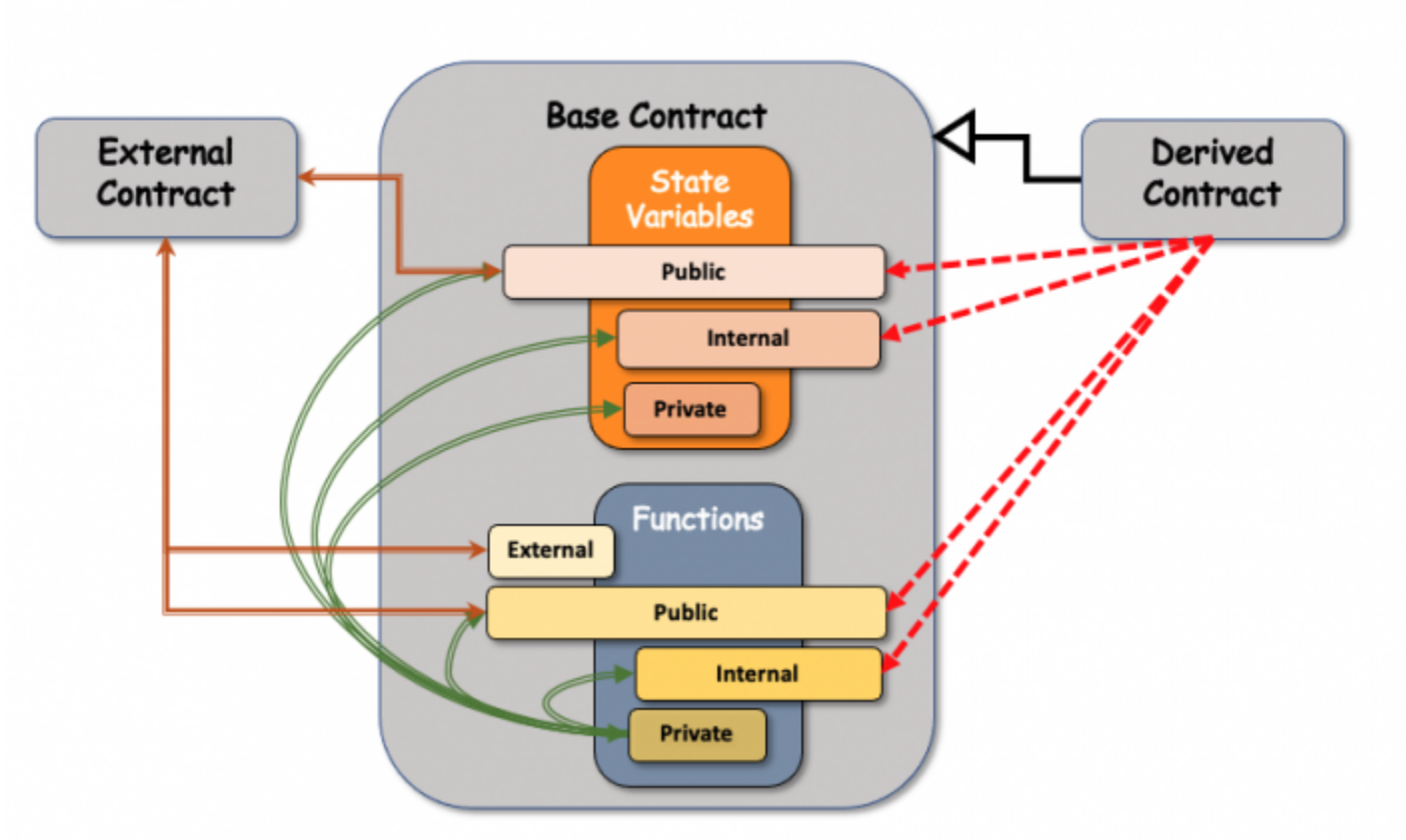


Figure 3: The Solidity Visibility Kinds

☐ [char]Review

1)
geeksforgeeks, Visibility Modes in C++ with Examples, 4 March 2020, Accessed 14 December 2021, https://www.geeksforgeeks.org/visibility-modes-in-c-with-examples/
2)
Solidity Language, Contracts : Visibility and Getters, Published: version 0.8.10, Accessed: 20 December 2021, https://docs.soliditylang.org/en/v0.8.10/contracts.html
3)
Cyrpto Market Pool - Blockchain Engineer Resource, Accessing Solidity smart contract data that is declared private, Accessed: 29 January 2022, https://cryptomarketpool.com/access-private-data-on-the-eth-blockchain/