Getters & Setters

Table of Content

Getters & Setters | Functions

Getters retrieve data from a contract, while setters modify the contract's state variables. Functions in Solidity allow these operations. In Solidity, functions facilitate the execution of specific tasks within a contract. Getters and setters are types of functions used to read and modify the contract's state variables, respectively.

Getter and setter functions within Solidity serve the purpose of retrieving and updating the stored values in a smart contract. Getters retrieve and return values, while setters modify the values of specific variables. The correct declaration of visibility is crucial for the proper functioning of these functions.

Preferring getters and setters over public variables is advisable due to the enhanced control they offer. Public variables can pose challenges in control, whereas getters and setters provide a structured approach to accessing and modifying data. This structured approach fosters improved security measures within the contract. Moreover, opting for getters and setters' aids in curbing unnecessary code deployment, leading to potential reductions in gas costs.

In Solidity, getters and setters are declared using the keyword "contract". For example, to create a getter and setter for a score variable, one would declare it as follows:

```
contract Score {
   uint score = 5;
   function getScore() public returns (uint) {
      return score;
   }
   function setScore(uint new_score) public {
      score = new_score;
   }
}
```

Code:

```
pragma solidity ^0.8.0;
contract GetterSetter {
    string storedValue;
    // Setter function to update the stored value
    function setValue(string memory newValue) public {
        storedValue = newValue;
    }
    // Getter function to retrieve the stored value
    function getValue() public view returns (string memory) {
        return storedValue;
    }
}
```

Explanation of the Code:

setValue Function:

- **Purpose:** Modifies the storedValue state variable.
- **Parameters:** Takes a string argument (newValue) to update storedValue.

getValue Function:

- **Purpose:** Retrieves the current value of storedValue.
- **Return Type:** Returns a string value (storedValue) using the view keyword, ensuring it doesn't modify contract state.

Getters & Setters: Usage and Real-Life Use Case

In programming, Getters and Setters are like the gateway keepers to information stored in a safe. Think of your favorite toy chest: a Getter helps you peek inside to see what's already there, while a Setter allows you to add new toys or replace existing ones.

Usage:

Getters are like magnifying glasses that let us look into a box without touching anything inside. They're used to fetch or retrieve specific information or values stored in a program or a smart contract. For example, imagine you have a digital piggy bank; a Getter function could help you check how much money is currently inside without changing anything.

Setters, on the other hand, are the caretakers of this toy chest. They manage what goes in or out. They're used to update or modify the contents within a program or contract. Going back to the toy chest analogy, a Setter function could help you add a new toy to the chest or replace an old one with a new favorite.

Real-Life Use Case:

Let's relate this to a library. The librarian acts as the Getter by allowing you to see the available books without altering their places. You check what's on the shelf and decide which book you want to borrow. When you return a book or borrow a new one, the librarian becomes the Setter, managing the library's collection. They add new books to the shelves or replace old ones as per the library's needs without revealing what other people borrowed.

In programming, especially in smart contracts on blockchains like Ethereum, Getters and Setters ensure controlled access to data. They maintain transparency by letting users view information without changing it (Getters) and manage how data is updated or added (Setters) while maintaining security and integrity within the system.