

Smart Contract Bank Explanation



```

/*
Steps
go to remix ide
click on create file SimpleBank.sol
paste the below code from line 23
Click on solidity compiler ,compile the code
click on deploy and run transaction
after deploying a new deployed contract is seen click that
to deposit above enter 1 ether and click deploy
to withdraw enter numeric value
while checkBalance to get the balance
the outputs are seen the console
Adios
*/
/*
Write a smart contract on a test network, for Bank account of a customer for following
operations:
□ Deposit money
□ Withdraw Money
□ Show balance
*/

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleBank {
    // Mapping to store the balance of each account holder (address)
    mapping(address => uint256) private balances;

    // Event to log deposits
    event Deposit(address indexed account, uint256 amount);

    // Event to log withdrawals
    event Withdraw(address indexed account, uint256 amount);

    // Deposit function to add money to the sender's account
    function deposit() public payable {
        require(msg.value > 0, "Deposit amount must be greater than zero.");
        balances[msg.sender] += msg.value;
        emit Deposit(msg.sender, msg.value);
    }

    // Withdraw function to withdraw money from the sender's account
    function withdraw(uint256 withdrawAmount) public {
        require(balances[msg.sender] >= withdrawAmount, "Insufficient balance.");
    }
}

```

```
balances[msg.sender] -= withdrawAmount;
payable(msg.sender).transfer(withdrawAmount);
emit Withdraw(msg.sender, withdrawAmount);
}

// Function to check the balance of the sender's account
function checkBalance() public view returns (uint256) {
    return balances[msg.sender];
}
}
```

EXPLAIN LINE BY LINE CODE BY CODE



Here's a line-by-line explanation of the Solidity smart contract for a simple bank:

solidity

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

- ``// SPDX-License-Identifier: MIT``: This is a license identifier. It ensures the code is publicly available and free to use, modify, and distribute under the MIT License.
- ``pragma solidity ^0.8.0``: This specifies the version of Solidity to be used for the contract. It ensures the code is compiled with Solidity version 0.8.0 or later but not higher than the next major version (e.g., 0.9.0).

solidity

```
contract SimpleBank {
```

- ``contract SimpleBank {``: This defines the start of a contract named ``SimpleBank``. A contract in Solidity is similar to a class in other programming languages.

solidity

```
// Mapping to store the balance of each account holder (address)
mapping(address => uint256) private balances;
```

- ``mapping(address => uint256) private balances``: This line declares a mapping (a key-value pair) where the key is an Ethereum address (``address``) and the value is a uint256 (the balance of the account). The ``private`` keyword means the mapping is not accessible from outside the contract.

solidity

```
// Event to log deposits
event Deposit(address indexed account, uint256 amount);
```

- ``event Deposit(address indexed account, uint256 amount);``: This defines an event ``Deposit`` that logs whenever a deposit is made. Events are used for logging data to the blockchain and can be retrieved off-chain. The ``indexed`` keyword allows the account to be filtered more easily in logs.

solidity

```
// Event to log withdrawals
event Withdraw(address indexed account, uint256 amount);
```

- ``event Withdraw(address indexed account, uint256 amount);``: Similarly, this defines an event ``Withdraw`` to log whenever a withdrawal occurs, with the address of the account and the withdrawal amount.

solidity

```
// Deposit function to add money to the sender's account
function deposit() public payable {
    require(msg.value > 0, "Deposit amount must be greater than zero.");
    balances[msg.sender] += msg.value;
    emit Deposit(msg.sender, msg.value);
}
```

- ``function deposit() public payable {``: This is a function called ``deposit`` that can be called by any external address. The ``payable`` keyword means this function can receive Ether.
- ``require(msg.value > 0, "Deposit amount must be greater than zero.");``: The ``require`` statement checks that the amount of Ether sent (``msg.value``) is greater than zero. If not, the transaction reverts with the message "Deposit amount must be greater than zero."
- ``balances[msg.sender] += msg.value;``: This line updates the sender's balance by adding the deposited amount (``msg.value``) to their current balance.
- ``emit Deposit(msg.sender, msg.value);``: This emits the ``Deposit`` event with the sender's address (``msg.sender``) and the amount deposited (``msg.value``).

solidity

```
// Withdraw function to withdraw money from the sender's account
function withdraw(uint256 withdrawAmount) public {
    require(balances[msg.sender] >= withdrawAmount, "Insufficient balance.");
    balances[msg.sender] -= withdrawAmount;
    payable(msg.sender).transfer(withdrawAmount);
    emit Withdraw(msg.sender, withdrawAmount);
}
```

- ``function withdraw(uint256 withdrawAmount) public {``: This is a function named ``withdraw`` that allows the sender to withdraw a specified ``withdrawAmount`` in Ether.

- ``require(balances[msg.sender] >= withdrawAmount, "Insufficient balance.");``: The ``require`` statement checks if the sender has enough balance to withdraw the specified amount. If not, it reverts the transaction with the message "Insufficient balance."
- ``balances[msg.sender] -= withdrawAmount;``: This line deducts the ``withdrawAmount`` from the sender's balance.
- ``payable(msg.sender).transfer(withdrawAmount);``: This sends the specified ``withdrawAmount`` of Ether back to the sender using the ``transfer`` method.
- ``emit Withdraw(msg.sender, withdrawAmount);``: This emits the ``Withdraw`` event with the sender's address and the withdrawn amount.

solidity

```
// Function to check the balance of the sender's account
function checkBalance() public view returns (uint256) {
    return balances[msg.sender];
}
```

- ``function checkBalance() public view returns (uint256) {``: This function returns the balance of the sender's account. It is marked as ``view``, meaning it doesn't modify the blockchain state but only reads it.
- ``return balances[msg.sender];``: This line returns the balance of the sender (``msg.sender``).

Summary of contract functionality:

- **Deposit:** The ``deposit()`` function allows the user to send Ether to the contract and store it in their balance.
- **Withdraw:** The ``withdraw(uint256)`` function allows users to withdraw a specified amount of Ether from their balance.
- **Check Balance:** The ``checkBalance()`` function allows users to check how much Ether they have deposited in the contract.

This is a simple bank contract that allows basic banking operations like depositing, withdrawing, and checking balances for users on the Ethereum blockchain.



```
/*
Steps
go to remix ide
click on create file Student.sol
paste the below code from line 41
Click on solidity compiler ,compile the code
click on deploy and run transaction
enter student as bc,23 this is the name and age
get details
getStudent by passing index 0,1,2 so on
Adios
*/
/*
```

Write a program in solidity to create Student data. Use the following constructs:

- Structures
- Arrays
- Fallback

Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.

Theory

A fallback function is a special type of function in Solidity that is automatically called under certain conditions, particularly when a contract receives Ether or when a function that does not exist in the contract is called. It has the following characteristics:

It has no name.

It does not take any arguments and does not return anything.

It's marked as either payable or non-payable, depending on whether it can accept Ether.

There are two types of fallback functions in Solidity:

fallback(): Invoked when a contract receives Ether and no other function matches the call data.

receive(): A simplified fallback function that is only used to receive Ether when no data is sent.

When is Fallback Triggered?

When Ether is sent to the contract without any data.

When a non-existent function is called.

Conclusion

Fallback: Used when a contract receives Ether or invalid function calls.

Transaction Fees and Gas: You can observe varying gas usage for deploying, adding students, and invoking the fallback.

*/

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.22;
```

```
contract StudentData {
```

```
    struct Student {
```

```
        string name;
```

```
        int age;
```

```
    }
```

```
    Student[] public studentArray;
```

```
    // Add new student details to the array
```

```
    function addStudent(string memory name, int age) public {
```

```
        for (uint i = 0; i < studentArray.length; i++) {
```

```
            if (studentArray[i].age == age) {
```

```
                revert("Roll No Exists");
```

```
            }
```

```
        }
```

```
        studentArray.push(Student(name, age));
```

```
    }
```

```
    // Returns all students in the array
```

```
function displayAllStudents() public view returns (Student[] memory) {
    return studentArray;
}

// Returns the length of the student array
function getStudentLength() public view returns (uint256) {
    return studentArray.length;
}

// Get a student by index
function getStudent(uint128 index) public view returns (Student memory) {
    require(studentArray.length > index, "Out of Index");
    return studentArray[index];
}

// Fallback function for handling unknown function calls
fallback() external payable {
    // Optionally, handle the call or log it
}

// Receive function for handling direct ETH transfers
receive() external payable {
    // Handle ETH transfers to the contract
}
}
```

EXPLAIN LINE BY LINE CODE BY CODE



Here's a detailed explanation of the Solidity smart contract `StudentData`, which includes structures, arrays, and fallback functions:

```
solidity

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.22;
```

- `// SPDX-License-Identifier: MIT`: This is the SPDX license identifier, ensuring the contract is licensed under the MIT License.
- `pragma solidity ^0.8.22;`: This specifies that the contract is written for Solidity version 0.8.22 or later, ensuring compatibility with that version.

```
solidity

contract StudentData {
```

- ``contract StudentData {``: This line defines the start of the contract ``StudentData``. The contract will handle storing and managing student data.

solidity

```
struct Student {
    string name;
    int age;
}
```

- ``struct Student {``: This defines a **struct** named ``Student``. A struct is a custom data type that allows you to group multiple variables under a single name. Here, the ``Student`` struct has two properties:
 - ``string name;``: A string to store the student's name.
 - ``int age;``: An integer to store the student's age (using ``int`` which can be both positive and negative, but for this case, age will be positive).

solidity

```
Student[] public studentArray;
```

- ``Student[] public studentArray;``: This line defines an array of ``Student`` structs called ``studentArray``. This array will hold all the student records. The ``public`` visibility modifier means the array is accessible from outside the contract.

solidity

```
// Add new student details to the array
function addStudent(string memory name, int age) public {
```

- ``function addStudent(string memory name, int age) public {``: This is the ``addStudent`` function, which allows adding a new student to the ``studentArray``. The function takes two parameters:
 - ``string memory name``: A string parameter for the student's name.
 - ``int age``: An integer parameter for the student's age. The function is ``public``, meaning it can be called by anyone.

solidity

```
    for (uint i = 0; i < studentArray.length; i++) {
        if (studentArray[i].age == age) {
            revert("Roll No Exists");
        }
    }
```

- ``for (uint i = 0; i < studentArray.length; i++) {``: This loop iterates through the entire ``studentArray`` to check if there is already a student with the same age.
- ``if (studentArray[i].age == age) {``: If a student with the same age is found, the function will **revert** the transaction.

- ``revert("Roll No Exists");``: This ``revert`` statement halts the transaction and throws an error message: "Roll No Exists". This ensures no duplicate students with the same age are added.

solidity

```
        studentArray.push(Student(name, age));
    }
```

- ``studentArray.push(Student(name, age));``: If no duplicate age is found, a new ``Student`` struct is created with the provided ``name`` and ``age`` and added to the ``studentArray`` using the ``push`` method.

solidity

```
// Returns all students in the array
function displayAllStudents() public view returns (Student[] memory) {
    return studentArray;
}
```

- ``function displayAllStudents() public view returns (Student[] memory) {``: This function returns the entire array of students (``studentArray``). The ``view`` modifier means the function doesn't modify the blockchain state, and the function returns the array.
- ``return studentArray;``: It simply returns the current list of students.

solidity

```
// Returns the length of the student array
function getStudentLength() public view returns (uint256) {
    return studentArray.length;
}
```

- ``function getStudentLength() public view returns (uint256) {``: This function returns the number of students currently stored in the ``studentArray``. It is marked as ``view`` because it only reads data.
- ``return studentArray.length;``: It returns the length (size) of the ``studentArray``.

solidity

```
// Get a student by index
function getStudent(uint128 index) public view returns (Student memory) {
    require(studentArray.length > index, "Out of Index");
    return studentArray[index];
}
```

- ``function getStudent(uint128 index) public view returns (Student memory) {``: This function retrieves a student by their index in the ``studentArray``. The index is passed as a ``uint128``.
- ``require(studentArray.length > index, "Out of Index");``: The ``require`` statement checks that the index is valid (i.e., the index is within the bounds of the array). If the index is out of range, it reverts the transaction with the message "Out of Index."
- ``return studentArray[index];``: It returns the student at the specified index.


```
solidity
```

```
// Fallback function for handling unknown function calls
fallback() external payable {
    // Optionally, handle the call or log it
}
```

- `fallback() external payable {}`: This is the **fallback** function. It is automatically invoked when the contract receives Ether or when a function that doesn't exist is called.
 - The function is marked `external`, meaning it can only be called from outside the contract.
 - The `payable` keyword means this function can accept Ether sent to the contract. This function does not return any data or take any parameters.
 - You can handle the unknown calls or log them in this function (it is empty here).

```
solidity
```

```
// Receive function for handling direct ETH transfers
receive() external payable {
    // Handle ETH transfers to the contract
}
```

- `receive() external payable {}`: This is the **receive** function, which is a simplified version of the fallback function. It is specifically designed to handle direct Ether transfers (without any data).
 - The `payable` modifier indicates the function can accept Ether.

```
solidity
```

```
}
```

- `}`: This closes the `StudentData` contract.

Key Concepts:

1. **Structures:** A custom data type is used to store student details (name and age) in the `Student` struct.
2. **Arrays:** The contract uses an array (`studentArray`) to store multiple student structs.
3. **Fallback Function:** The fallback function (`fallback()`) handles unknown function calls or transactions that send Ether without any function call data.
4. **Receive Function:** The receive function handles direct Ether transfers to the contract.

Conclusion:

This contract allows adding, retrieving, and managing student data, with additional functionalities like handling direct Ether transfers and invalid function calls using fallback and receive functions.

ChatGPT can make mistakes. Check important info.