BCT CONTRACT

The EtherWallet smart contract is a simple Ethereum wallet that allows an owner to store, send, and check Ether balance within the contract. Here's a breakdown of each part:

Code Explanation

SPDX License Identifier

solidity
Copy code
// SPDX-License-Identifier: MIT

The SPDX license identifier is a comment line that specifies the license type for the code. Here, it is set to "MIT", which is a commonly used open-source license. This identifier is necessary for Solidity to meet licensing requirements, making it easier for developers to know the license under which the code is shared.

Pragma Directive

solidity
Copy code
pragma solidity >=0.5.0 <0.9.0;

The pragma directive specifies the Solidity compiler version range that is compatible with this contract. This ensures the code is compiled with Solidity versions between 0.5.0 and 0.9.0, ensuring compatibility with the syntax and features used.

Contract Definition

solidity
Copy code
contract EtherWallet {
    address payable public owner;

The contract EtherWallet defines a simple wallet functionality on the Ethereum blockchain. It has a state variable owner, which is of type address payable. The payable modifier allows this address to send and receive Ether, enabling it to execute transactions involving Ether.

Constructor

solidity
Copy code
    constructor() payable {
        owner = payable(msg.sender); // Mark owner as payable
    }

The constructor is a function that is only run once, at the time of contract deployment. Here, it sets the contract's owner to be the Ethereum address of the account deploying the contract

(msg.sender). The payable keyword in the constructor allows the contract to receive Ether during deployment if an initial balance is sent.

Withdraw Function

```
solidity
Copy code
   function Withdraw(uint _amount) public {
      require(msg.sender == owner, "Only the owner can invoke this function");
      payable(msg.sender).transfer(_amount);
   }
```

The Withdraw function allows the owner to withdraw a specified _amount of Ether from the contract. It has the following steps:

1. **Access Control**: The require statement checks if the caller (msg.sender) is the owner. If not, it reverts with an error message, "Only the owner can invoke this function."
2. **Transfer**: If the caller is the owner, the function transfers the specified _amount to the msg.sender. The payable(msg.sender).transfer(_amount); statement ensures Ether is sent only to the owner.

getBalance Function

```
solidity
Copy code
   function getBalance() external view returns (uint) {
      return address(this).balance;
   }
```

The getBalance function is a view function that allows anyone to check the balance of the contract in Ether. It returns the balance using address(this).balance, which provides the current Ether amount stored in the contract's address.

receive() Function

```
solidity
Copy code
   receive() external payable {}
```

The receive function is a special function that allows the contract to accept Ether sent to it directly, without calling any specific function. It is marked as payable, enabling it to receive Ether. This function will be triggered automatically whenever Ether is sent to the contract's address without any additional data or function call.

Summary

- **EtherWallet** allows an owner to manage Ether within the contract.
- **Constructor** sets the deploying address as the owner.

- **Withdraw** function allows only the owner to withdraw specified Ether amounts.
- **getBalance** lets anyone check the Ether balance in the contract.
- **receive** function enables the contract to receive Ether directly.

This is a basic wallet contract suitable for storing and managing Ether securely by the owner.

BCT STUDENT

This StudentRegister contract is a Solidity-based smart contract designed to store and retrieve student information, including their ID, name, course, and marks, on the Ethereum blockchain. The contract restricts access to specific functionalities so that only the owner can register student details.

Code Explanation

SPDX License Identifier

solidity
Copy code
```
// SPDX-License-Identifier: MIT
```

This comment specifies the license type for the code, here being the MIT License, which is an open-source license commonly used in software development.

Pragma Directive

solidity
Copy code
```
pragma solidity >=0.4.0 <=0.9.0;
```

The pragma directive specifies that the contract is compatible with Solidity compiler versions from 0.4.0 to 0.9.0. This allows the code to be compiled across multiple versions of Solidity, ensuring greater compatibility and flexibility.

Contract Definition

solidity
Copy code

```
contract StudentRegister {
   address public owner;

   mapping (address => student) students;
```

The contract StudentRegister is defined to manage a student registration system. It contains:

- **owner**: An address variable representing the contract's owner (the account that deployed the contract).
- **students**: A mapping that associates each student's address (address) with their information stored as a student struct.

Constructor

```
solidity
Copy code
   constructor() {
      owner = msg.sender;
   }
```

The constructor is executed only once at the time of deployment. It sets the owner of the contract to the Ethereum account that deployed it (msg.sender). This ensures that only the deploying account has the authority to add student records.

onlyOwner Modifier

```
solidity
Copy code
   modifier onlyOwner {
      require(msg.sender == owner);
      _;
   }
```

This onlyOwner modifier is used to restrict access to specific functions. It checks if the function caller (msg.sender) is the owner. If not, it will revert the transaction. If the check passes, it continues to execute the function code. This is used for security to ensure that only the contract owner can register student details.

Student Struct

```
solidity
Copy code
   struct student {
      address studentId;
      string name;
      string course;
      uint256 mark1;
      uint256 mark2;
      uint256 mark3;
      uint256 totalMarks;
      uint256 percentage;
```

```
    bool isExist;
  }
```

The student struct defines the data structure to store a student's information, including:

- studentId: The unique address ID of the student.
- name: The student's name.
- course: The course the student is enrolled in.
- mark1, mark2, mark3: Marks obtained in three subjects.
- totalMarks: Total marks obtained by the student.
- percentage: Calculated percentage.
- isExist: A boolean flag that confirms if the student record already exists.

register Function

```
solidity
Copy code
  function register(address studentId, string memory name, string memory course, uint256 mark1, uint256 mark2,
uint256 mark3) public onlyOwner {
    require(students[studentId].isExist == false, "Fraud Not Possible, student details already registered and cannot
be altered");

    uint256 totalMarks;
    uint256 percentage;

    totalMarks = (mark1 + mark2 + mark3);
    percentage = (totalMarks / 3);

    students[studentId] = student(studentId, name, course, mark1, mark2, mark3, totalMarks, percentage, true);
  }
```

The register function is used to add a student's details. It has the following steps:

1. **Access Control**: The onlyOwner modifier restricts this function so only the owner can call it.
2. **Duplicate Check**: The require statement checks that a student with the same studentId hasn't already been registered.
3. **Marks and Percentage Calculation**:
    - totalMarks is calculated by summing up mark1, mark2, and mark3.
    - percentage is calculated by dividing totalMarks by 3 (assuming there are three subjects).
4. **Student Registration**: If all checks pass, a new student struct is created and added to the students mapping using studentId as the key.

getStudentDetails Function

```
solidity
Copy code
  function getStudentDetails(address studentId) public view returns (address, string memory, string memory,
uint256, uint256) {
```

```
        return (
            students[studentId].studentId,
            students[studentId].name,
            students[studentId].course,
            students[studentId].totalMarks,
            students[studentId].percentage
        );
    }
```

The `getStudentDetails` function allows anyone to retrieve specific details of a registered student using their `studentId`. It is a `view` function, meaning it does not modify the contract state and can be called without incurring a transaction fee. The function returns:

- `studentId`: The unique address ID of the student.
- `name`: The student's name.
- `course`: The course they are enrolled in.
- `totalMarks`: The calculated total marks.
- `percentage`: The calculated percentage.

Summary

- **Owner Access Control**: Only the owner of the contract can register student details.
- **Student Registration**: The owner can register a student by specifying their ID, name, course, and marks.
- **Duplicate Prevention**: The contract checks to prevent duplicate registrations of the same student.
- **View Function**: Anyone can view a student's basic details, including total marks and percentage.

This contract is designed to securely store student records and restrict unauthorized modifications. The Ethereum blockchain ensures data transparency, making this contract suitable for tamper-proof student record systems.