```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.8.0) (finance/PaymentSplitter.sol)

pragma solidity ^0.8.0;

import "../token/ERC20/utils/SafeERC20.sol";
import "../utils/Address.sol";
import "../utils/Context.sol";

/**
 * @title PaymentSplitter
 * @dev This contract allows to split Ether payments among a group of accounts. The sender does not need to be aware
 * that the Ether will be split in this way, since it is handled transparently by the contract.
 *
 * The split can be in equal parts or in any other arbitrary proportion. The way this is specified is by assigning each
 * account to a number of shares. Of all the Ether that this contract receives, each account will then be able to claim
 * an amount proportional to the percentage of total shares they were assigned. The distribution of shares is set at the
 * time of contract deployment and can't be updated thereafter.
 *
 * `PaymentSplitter` follows a _pull payment_ model. This means that payments are not automatically forwarded to the
 * accounts but kept in this contract, and the actual transfer is triggered as a separate step by calling the {release}
 * function.
 *
 * NOTE: This contract assumes that ERC20 tokens will behave similarly to native tokens (Ether). Rebasing tokens, and
 * tokens that apply fees during transfers, are likely to not be supported as expected. If in doubt, we encourage you
 * to run tests before sending real value to this contract.
 */
contract PaymentSplitter is Context {
    event PayeeAdded(address account, uint256 shares);
    event PaymentReleased(address to, uint256 amount);
    event ERC20PaymentReleased(IERC20 indexed token, address to, uint256 amount);
    event PaymentReceived(address from, uint256 amount);

    uint256 private _totalShares;
    uint256 private _totalReleased;

    mapping(address => uint256) private _shares;
    mapping(address => uint256) private _released;
    address[] private _payees;

    mapping(IERC20 => uint256) private _erc20TotalReleased;
    mapping(IERC20 => mapping(address => uint256)) private _erc20Released;

    /**
     * @dev Creates an instance of `PaymentSplitter` where each account in `payees` is assigned the number of shares at
     * the matching position in the `shares` array.
     *
     * All addresses in `payees` must be non-zero. Both arrays must have the same non-zero length, and there must be no
     * duplicates in `payees`.
     */
    constructor(address[] memory payees, uint256[] memory shares_) payable {
        require(payees.length == shares_.length, "PaymentSplitter: payees and shares length mismatch");
        require(payees.length > 0, "PaymentSplitter: no payees");

        for (uint256 i = 0; i < payees.length; i++) {
            _addPayee(payees[i], shares_[i]);
        }
    }

    /**
     * @dev The Ether received will be logged with {PaymentReceived} events. Note
```

```
     that these events are not fully
62   * reliable: it's possible for a contract to receive Ether without triggering
     this function. This only affects the
63   * reliability of the events, and not the actual splitting of Ether.
64   *
65   * To learn more about this see the Solidity documentation for
66   *
     https://solidity.readthedocs.io/en/latest/contracts.html#fallback-function[fallba
     ck
67   * functions].
68   */
69  receive() external payable virtual {
70      emit PaymentReceived(_msgSender(), msg.value);
71  }
72
73  /**
74   * @dev Getter for the total shares held by payees.
75   */
76  function totalShares() public view returns (uint256) {
77      return _totalShares;
78  }
79
80  /**
81   * @dev Getter for the total amount of Ether already released.
82   */
83  function totalReleased() public view returns (uint256) {
84      return _totalReleased;
85  }
86
87  /**
88   * @dev Getter for the total amount of `token` already released. `token` should
     be the address of an IERC20
89   * contract.
90   */
91  function totalReleased(IERC20 token) public view returns (uint256) {
92      return _erc20TotalReleased[token];
93  }
94
95  /**
96   * @dev Getter for the amount of shares held by an account.
97   */
98  function shares(address account) public view returns (uint256) {
99      return _shares[account];
100 }
101
102 /**
103  * @dev Getter for the amount of Ether already released to a payee.
104  */
105 function released(address account) public view returns (uint256) {
106     return _released[account];
107 }
108
109 /**
110  * @dev Getter for the amount of `token` tokens already released to a payee.
     `token` should be the address of an
111  * IERC20 contract.
112  */
113 function released(IERC20 token, address account) public view returns (uint256) {
114     return _erc20Released[token][account];
115 }
116
117 /**
118  * @dev Getter for the address of the payee number `index`.
119  */
120 function payee(uint256 index) public view returns (address) {
121     return _payees[index];
122 }
123
124 /**
125  * @dev Getter for the amount of payee's releasable Ether.
126  */
127 function releasable(address account) public view returns (uint256) {
128     uint256 totalReceived = address(this).balance + totalReleased();
```

```
129             return _pendingPayment(account, totalReceived, released(account));
130         }
131 
132         /**
133          * @dev Getter for the amount of payee's releasable `token` tokens. `token`
             should be the address of an
134          * IERC20 contract.
135          */
136         function releasable(IERC20 token, address account) public view returns (uint256) {
137             uint256 totalReceived = token.balanceOf(address(this)) + totalReleased(token);
138             return _pendingPayment(account, totalReceived, released(token, account));
139         }
140 
141         /**
142          * @dev Triggers a transfer to `account` of the amount of Ether they are owed,
             according to their percentage of the
143          * total shares and their previous withdrawals.
144          */
145         function release(address payable account) public virtual {
146             require(_shares[account] > 0, "PaymentSplitter: account has no shares");
147 
148             uint256 payment = releasable(account);
149 
150             require(payment != 0, "PaymentSplitter: account is not due payment");
151 
152             // _totalReleased is the sum of all values in _released.
153             // If "_totalReleased += payment" does not overflow, then "_released[account]
                += payment" cannot overflow.
154             _totalReleased += payment;
155             unchecked {
156                 _released[account] += payment;
157             }
158 
159             Address.sendValue(account, payment);
160             emit PaymentReleased(account, payment);
161         }
162 
163         /**
164          * @dev Triggers a transfer to `account` of the amount of `token` tokens they are
             owed, according to their
165          * percentage of the total shares and their previous withdrawals. `token` must be
             the address of an IERC20
166          * contract.
167          */
168         function release(IERC20 token, address account) public virtual {
169             require(_shares[account] > 0, "PaymentSplitter: account has no shares");
170 
171             uint256 payment = releasable(token, account);
172 
173             require(payment != 0, "PaymentSplitter: account is not due payment");
174 
175             // _erc20TotalReleased[token] is the sum of all values in
                _erc20Released[token].
176             // If "_erc20TotalReleased[token] += payment" does not overflow, then
                "_erc20Released[token][account] += payment"
177             // cannot overflow.
178             _erc20TotalReleased[token] += payment;
179             unchecked {
180                 _erc20Released[token][account] += payment;
181             }
182 
183             SafeERC20.safeTransfer(token, account, payment);
184             emit ERC20PaymentReleased(token, account, payment);
185         }
186 
187         /**
188          * @dev internal logic for computing the pending payment of an `account` given
             the token historical balances and
189          * already released amounts.
190          */
191         function _pendingPayment(
192             address account,
193             uint256 totalReceived,
```

```
194             uint256 alreadyReleased
195         ) private view returns (uint256) {
196             return (totalReceived * _shares[account]) / _totalShares - alreadyReleased;
197         }
198 
199         /**
200          * @dev Add a new payee to the contract.
201          * @param account The address of the payee to add.
202          * @param shares_ The number of shares owned by the payee.
203          */
204         function _addPayee(address account, uint256 shares_) private {
205             require(account != address(0), "PaymentSplitter: account is the zero address")
                ;
206             require(shares_ > 0, "PaymentSplitter: shares are 0");
207             require(_shares[account] == 0, "PaymentSplitter: account already has shares");
208 
209             _payees.push(account);
210             _shares[account] = shares_;
211             _totalShares = _totalShares + shares_;
212             emit PayeeAdded(account, shares_);
213         }
214     }
215 
```