

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 interface IERC20 {
5     function decimals() external view returns (uint
6         8);
7     /**
8      * @dev Returns the amount of tokens in existence.
9      */
10    function totalSupply() external view returns (uint
11        256);
12    /**
13     * @dev Returns the amount of tokens owned by `acc
14     ount`.
15     */
16    function balanceOf(address account) external view
17        returns (uint256);
18    /**
19     * @dev Moves `amount` tokens from the caller's ac
20     ount to `recipient`.
21     *
22     * Returns a boolean value indicating whether the
23     operation succeeded.
24     *
25     * Emits a {Transfer} event.
26     */
27    function transfer(address recipient, uint256 amoun
28        t) external returns (bool);
29    /**
30     * @dev Returns the remaining number of tokens tha
31     t `spender` will be
32     * allowed to spend on behalf of `owner` through
33     {transferFrom}. This is
34     * zero by default.
35     *
36     * This value changes when {approve} or {transferF
37     rom} are called.
38     */
39    function allowance(address owner, address spender)
40        external view returns (uint256);
41    /**
42     * @dev Sets `amount` as the allowance of `spender
43     ` over the caller's tokens.
44     *
45     * Returns a boolean value indicating whether the
46     operation succeeded.
47     *
48     * IMPORTANT: Beware that changing an allowance wi
49     th this method brings the risk
50     * that someone may use both the old and the new a
51     llowance by unfortunate
52     * transaction ordering. One possible solution to
53     mitigate this race
54     * condition is to first reduce the spender's allo
55     wance to 0 and set the
56     * desired value afterwards:
57     * https://github.com/ethereum/EIPs/issues/20#issu
58     ecomment-263524729

```

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3 // Only change to generate diff
4
5 interface IERC20 {
6     function decimals() external view returns (uint
7         8);
8     /**
9      * @dev Returns the amount of tokens in existence.
10     */
11    function totalSupply() external view returns (uint
12        256);
13    /**
14     * @dev Returns the amount of tokens owned by `acc
15     ount`.
16     */
17    function balanceOf(address account) external view
18        returns (uint256);
19    /**
20     * @dev Moves `amount` tokens from the caller's ac
21     ount to `recipient`.
22     *
23     * Returns a boolean value indicating whether the
24     operation succeeded.
25     *
26     * Emits a {Transfer} event.
27     */
28    function transfer(address recipient, uint256 amoun
29        t) external returns (bool);
30    /**
31     * @dev Returns the remaining number of tokens tha
32     t `spender` will be
33     * allowed to spend on behalf of `owner` through
34     {transferFrom}. This is
35     * zero by default.
36     *
37     * This value changes when {approve} or {transferF
38     rom} are called.
39     */
40    function allowance(address owner, address spender)
41        external view returns (uint256);
42    /**
43     * @dev Sets `amount` as the allowance of `spender
44     ` over the caller's tokens.
45     *
46     * Returns a boolean value indicating whether the
47     operation succeeded.
48     *
49     * IMPORTANT: Beware that changing an allowance wi
50     th this method brings the risk
51     * that someone may use both the old and the new a
52     llowance by unfortunate
53     * transaction ordering. One possible solution to
54     mitigate this race
55     * condition is to first reduce the spender's allo
56     wance to 0 and set the
57     * desired value afterwards:
58     * https://github.com/ethereum/EIPs/issues/20#issu
59     ecomment-263524729

```

```

45  *
46  * Emits an {Approval} event.
47  */
48  function approve(address spender, uint256 amount)
    external returns (bool);
49
50  /**
51   * @dev Moves `amount` tokens from `sender` to `re
    cipient` using the
52   * allowance mechanism. `amount` is then deducted
    from the caller's
53   * allowance.
54   *
55   * Returns a boolean value indicating whether the
    operation succeeded.
56   *
57   * Emits a {Transfer} event.
58   */
59  function transferFrom(address sender, address reci
    pient, uint256 amount) external returns (bool);
60
61  /**
62   * @dev Emitted when `value` tokens are moved from
    one account (`from`) to
63   * another (`to`).
64   *
65   * Note that `value` may be zero.
66   */
67  event Transfer(address indexed from, address index
    ed to, uint256 value);
68
69  /**
70   * @dev Emitted when the allowance of a `spender`
    for an `owner` is set by
71   * a call to {approve}. `value` is the new allowan
    ce.
72   */
73  event Approval(address indexed owner, address inde
    xed spender, uint256 value);
74 }
75
76 contract StakingWarmup {
77
78     address public immutable staking;
79     IERC20 public immutable MEMORies;
80
81     constructor ( address _staking, address _MEMORie
    s ) {
82         require( _staking != address(0) );
83         staking = _staking;
84         require( _MEMORies != address(0) );
85         MEMORies = IERC20(_MEMORies);
86     }
87
88     function retrieve( address _staker, uint _amount
    ) external {
89         require( msg.sender == staking, "NA" );
90         MEMORies.transfer( _staker, _amount );
91     }
92 }

```

```

46  *
47  * Emits an {Approval} event.
48  */
49  function approve(address spender, uint256 amount)
    external returns (bool);
50
51  /**
52   * @dev Moves `amount` tokens from `sender` to `re
    cipient` using the
53   * allowance mechanism. `amount` is then deducted
    from the caller's
54   * allowance.
55   *
56   * Returns a boolean value indicating whether the
    operation succeeded.
57   *
58   * Emits a {Transfer} event.
59   */
60  function transferFrom(address sender, address reci
    pient, uint256 amount) external returns (bool);
61
62  /**
63   * @dev Emitted when `value` tokens are moved from
    one account (`from`) to
64   * another (`to`).
65   *
66   * Note that `value` may be zero.
67   */
68  event Transfer(address indexed from, address index
    ed to, uint256 value);
69
70  /**
71   * @dev Emitted when the allowance of a `spender`
    for an `owner` is set by
72   * a call to {approve}. `value` is the new allowan
    ce.
73   */
74  event Approval(address indexed owner, address inde
    xed spender, uint256 value);
75 }
76
77 contract StakingWarmup {
78
79     address public immutable staking;
80     IERC20 public immutable MEMORies;
81
82     constructor ( address _staking, address _MEMORie
    s ) {
83         require( _staking != address(0) );
84         staking = _staking;
85         require( _MEMORies != address(0) );
86         MEMORies = IERC20(_MEMORies);
87     }
88
89     function retrieve( address _staker, uint _amount
    ) external {
90         require( msg.sender == staking, "NA" );
91         MEMORies.transfer( _staker, _amount );
92     }
93 }

```