- 0 Removals + 1 Addition

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity 0.7.5;

interface IERC20 {
    function decimals() external view returns (uint8);
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

  /**
   * @dev Returns the amount of tokens owned by `account`.
   */
  function balanceOf(address account) external view returns (uint256);

  /**
   * @dev Moves `amount` tokens from the caller's account to `recipient`.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * Emits a {Transfer} event.
   */
  function transfer(address recipient, uint256 amount) external returns (bool);

  /**
   * @dev Returns the remaining number of tokens that `spender` will be
   * allowed to spend on behalf of `owner` through {transferFrom}. This is
   * zero by default.
   *
   * This value changes when {approve} or {transferFrom} are called.
   */
  function allowance(address owner, address spender) external view returns (uint256);

  /**
   * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * IMPORTANT: Beware that changing an allowance with this method brings the risk
   * that someone may use both the old and the new allowance by unfortunate
   * transaction ordering. One possible solution to mitigate this race
   * condition is to first reduce the spender's allowance to 0 and set the
   * desired value afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
```

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity 0.7.5;
// Only change to generate diff

interface IERC20 {
    function decimals() external view returns (uint8);
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

  /**
   * @dev Returns the amount of tokens owned by `account`.
   */
  function balanceOf(address account) external view returns (uint256);

  /**
   * @dev Moves `amount` tokens from the caller's account to `recipient`.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * Emits a {Transfer} event.
   */
  function transfer(address recipient, uint256 amount) external returns (bool);

  /**
   * @dev Returns the remaining number of tokens that `spender` will be
   * allowed to spend on behalf of `owner` through {transferFrom}. This is
   * zero by default.
   *
   * This value changes when {approve} or {transferFrom} are called.
   */
  function allowance(address owner, address spender) external view returns (uint256);

  /**
   * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * IMPORTANT: Beware that changing an allowance with this method brings the risk
   * that someone may use both the old and the new allowance by unfortunate
   * transaction ordering. One possible solution to mitigate this race
   * condition is to first reduce the spender's allowance to 0 and set the
   * desired value afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
```

```
    *
    * Emits an {Approval} event.
    */
   function approve(address spender, uint256 amount) external returns (bool);

   /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the
    * allowance mechanism. `amount` is then deducted from the caller's
    * allowance.
    *
    * Returns a boolean value indicating whether the operation succeeded.
    *
    * Emits a {Transfer} event.
    */
   function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

   /**
    * @dev Emitted when `value` tokens are moved from one account (`from`) to
    * another (`to`).
    *
    * Note that `value` may be zero.
    */
   event Transfer(address indexed from, address indexed to, uint256 value);

   /**
    * @dev Emitted when the allowance of a `spender` for an `owner` is set by
    * a call to {approve}. `value` is the new allowance.
    */
   event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract StakingWarmup {

    address public immutable staking;
    IERC20 public immutable MEMOries;

    constructor ( address _staking, address _MEMOries ) {
        require( _staking != address(0) );
        staking = _staking;
        require( _MEMOries != address(0) );
        MEMOries = IERC20(_MEMOries);
    }

    function retrieve( address _staker, uint _amount ) external {
        require( msg.sender == staking, "NA" );
        MEMOries.transfer( _staker, _amount );
    }
}
```

```
    *
    * Emits an {Approval} event.
    */
   function approve(address spender, uint256 amount) external returns (bool);

   /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the
    * allowance mechanism. `amount` is then deducted from the caller's
    * allowance.
    *
    * Returns a boolean value indicating whether the operation succeeded.
    *
    * Emits a {Transfer} event.
    */
   function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

   /**
    * @dev Emitted when `value` tokens are moved from one account (`from`) to
    * another (`to`).
    *
    * Note that `value` may be zero.
    */
   event Transfer(address indexed from, address indexed to, uint256 value);

   /**
    * @dev Emitted when the allowance of a `spender` for an `owner` is set by
    * a call to {approve}. `value` is the new allowance.
    */
   event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract StakingWarmup {

    address public immutable staking;
    IERC20 public immutable MEMOries;

    constructor ( address _staking, address _MEMOries ) {
        require( _staking != address(0) );
        staking = _staking;
        require( _MEMOries != address(0) );
        MEMOries = IERC20(_MEMOries);
    }

    function retrieve( address _staker, uint _amount ) external {
        require( msg.sender == staking, "NA" );
        MEMOries.transfer( _staker, _amount );
    }
}
```