```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.8.0) (token/ERC20/utils/SafeERC20.sol)

pragma solidity ^0.8.0;

import "../IERC20.sol";
import "../extensions/IERC20Permit.sol";
import "../../../utils/Address.sol";

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using Address for address;

    function safeTransfer(
        IERC20 token,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(
        IERC20 token,
        address from,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(
        IERC20 token,
        address spender,
        uint256 value
    ) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        require(
            (value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(
        IERC20 token,
        address spender,
        uint256 value
    ) internal {
        uint256 newAllowance = token.allowance(address(this), spender) + value;
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }
```

```
69
70      function safeDecreaseAllowance(
71          IERC20 token,
72          address spender,
73          uint256 value
74      ) internal {
75          unchecked {
76              uint256 oldAllowance = token.allowance(address(this), spender);
77              require(oldAllowance >= value, "SafeERC20: decreased allowance below zero"
                );
78              uint256 newAllowance = oldAllowance - value;
79              _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector,
                spender, newAllowance));
80          }
81      }
82
83      function safePermit(
84          IERC20Permit token,
85          address owner,
86          address spender,
87          uint256 value,
88          uint256 deadline,
89          uint8 v,
90          bytes32 r,
91          bytes32 s
92      ) internal {
93          uint256 nonceBefore = token.nonces(owner);
94          token.permit(owner, spender, value, deadline, v, r, s);
95          uint256 nonceAfter = token.nonces(owner);
96          require(nonceAfter == nonceBefore + 1, "SafeERC20: permit did not succeed");
97      }
98
99      /**
100      * @dev Imitates a Solidity high-level call (i.e. a regular function call to a
         contract), relaxing the requirement
101      * on the return value: the return value is optional (but if data is returned, it
         must not be false).
102      * @param token The token targeted by the call.
103      * @param data The call data (encoded using abi.encode or one of its variants).
104      */
105     function _callOptionalReturn(IERC20 token, bytes memory data) private {
106         // We need to perform a low level call here, to bypass Solidity's return data
            size checking mechanism, since
107         // we're implementing it ourselves. We use {Address-functionCall} to perform
            this call, which verifies that
108         // the target address contains contract code and also asserts for success in
            the low-level call.
109
110         bytes memory returndata = address(token).functionCall(data, "SafeERC20:
            low-level call failed");
111         if (returndata.length > 0) {
112             // Return data is optional
113             require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did
                not succeed");
114         }
115     }
116 }
117
```