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

pragma solidity ^0.8.2;

import "../../utils/Address.sol";

/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that will be deployed
 * behind a proxy. Since proxied contracts do not make use of a constructor, it's common to move constructor logic to an
 * external initializer function, usually called `initialize`. It then becomes necessary to protect this initializer
 * function so it can only be called once. The {initializer} modifier provided by this contract will have this effect.
 *
 * The initialization functions use a version number. Once a version number is used, it is consumed and cannot be
 * reused. This mechanism prevents re-execution of each "step" but allows the creation of new initialization steps in
 * case an upgrade adds a module that needs to be initialized.
 *
 * For example:
 *
 * [.hljs-theme-light.nopadding]
 * ```
 * contract MyToken is ERC20Upgradeable {
 *     function initialize() initializer public {
 *         __ERC20_init("MyToken", "MTK");
 *     }
 * }
 * contract MyTokenV2 is MyToken, ERC20PermitUpgradeable {
 *     function initializeV2() reinitializer(2) public {
 *         __ERC20Permit_init("MyToken");
 *     }
 * }
 * ```
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be called as early as
 * possible by providing the encoded function call as the `_data` argument to {ERC1967Proxy-constructor}.
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer twice, or to ensure
 * that all initializers are idempotent. This is not verified automatically as constructors are by Solidity.
 *
 * [CAUTION]
 * ====
 * Avoid leaving a contract uninitialized.
 *
 * An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation
 * contract, which may impact the proxy. To prevent the implementation contract from being used, you should invoke
 * the {_disableInitializers} function in the constructor to automatically lock it when it is deployed:
 *
 * [.hljs-theme-light.nopadding]
 * ```
 * /// @custom:oz-upgrades-unsafe-allow constructor
 * constructor() {
 *     _disableInitializers();
 * }
 * ```
 * ====
 */
abstract contract Initializable {
    /**
     * @dev Indicates that the contract has been initialized.
     * @custom:oz-retyped-from bool
```

```
     */
    uint8 private _initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private _initializing;

    /**
     * @dev Triggered when the contract has been initialized or reinitialized.
     */
    event Initialized(uint8 version);

    /**
     * @dev A modifier that defines a protected initializer function that can be invoked at most once. In its scope,
     * `onlyInitializing` functions can be used to initialize parent contracts.
     *
     * Similar to `reinitializer(1)`, except that functions marked with `initializer` can be nested in the context of a
     * constructor.
     *
     * Emits an {Initialized} event.
     */
    modifier initializer() {
        bool isTopLevelCall = !_initializing;
        require(
            (isTopLevelCall && _initialized < 1) || (!Address.isContract(address(this)) && _initialized == 1),
            "Initializable: contract is already initialized"
        );
        _initialized = 1;
        if (isTopLevelCall) {
            _initializing = true;
        }
        _;
        if (isTopLevelCall) {
            _initializing = false;
            emit Initialized(1);
        }
    }

    /**
     * @dev A modifier that defines a protected reinitializer function that can be invoked at most once, and only if the
     * contract hasn't been initialized to a greater version before. In its scope, `onlyInitializing` functions can be
     * used to initialize parent contracts.
     *
     * A reinitializer may be used after the original initialization step. This is essential to configure modules that
     * are added through upgrades and that require initialization.
     *
     * When `version` is 1, this modifier is similar to `initializer`, except that functions marked with `reinitializer`
     * cannot be nested. If one is invoked in the context of another, execution will revert.
     *
     * Note that versions can jump in increments greater than 1; this implies that if multiple reinitializers coexist in
     * a contract, executing them in the right order is up to the developer or operator.
     *
     * WARNING: setting the version to 255 will prevent any future reinitialization.
     *
     * Emits an {Initialized} event.
     */
    modifier reinitializer(uint8 version) {
        require(!_initializing && _initialized < version, "Initializable: contract is already initialized");
        _initialized = version;
        _initializing = true;
        _;
```

```solidity
        _initializing = false;
        emit Initialized(version);
    }

    /**
     * @dev Modifier to protect an initialization function so that it can only be invoked by functions with the
     * {initializer} and {reinitializer} modifiers, directly or indirectly.
     */
    modifier onlyInitializing() {
        require(_initializing, "Initializable: contract is not initializing");
        _;
    }

    /**
     * @dev Locks the contract, preventing any future reinitialization. This cannot be part of an initializer call.
     * Calling this in the constructor of a contract will prevent that contract from being initialized or reinitialized
     * to any version. It is recommended to use this to lock implementation contracts that are designed to be called
     * through proxies.
     *
     * Emits an {Initialized} event the first time it is successfully executed.
     */
    function _disableInitializers() internal virtual {
        require(!_initializing, "Initializable: contract is initializing");
        if (_initialized != type(uint8).max) {
            _initialized = type(uint8).max;
            emit Initialized(type(uint8).max);
        }
    }

    /**
     * @dev Returns the highest version that has been initialized. See {reinitializer}.
     */
    function _getInitializedVersion() internal view returns (uint8) {
        return _initialized;
    }

    /**
     * @dev Returns `true` if the contract is currently initializing. See {onlyInitializing}.
     */
    function _isInitializing() internal view returns (bool) {
        return _initializing;
    }
}
```