```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (proxy/transparent/ProxyAdmin.sol)

pragma solidity ^0.8.0;

import "./TransparentUpgradeableProxy.sol";
import "../../access/Ownable.sol";

/**
 * @dev This is an auxiliary contract meant to be assigned as the admin of a {TransparentUpgradeableProxy}. For an
 * explanation of why you would want to use this see the documentation for {TransparentUpgradeableProxy}.
 */
contract ProxyAdmin is Ownable {
    /**
     * @dev Returns the current implementation of `proxy`.
     *
     * Requirements:
     *
     * - This contract must be the admin of `proxy`.
     */
    function getProxyImplementation(TransparentUpgradeableProxy proxy) public view virtual returns (address) {
        // We need to manually run the static call since the getter cannot be flagged as view
        // bytes4(keccak256("implementation()")) == 0x5c60da1b
        (bool success, bytes memory returndata) = address(proxy).staticcall(hex"5c60da1b");
        require(success);
        return abi.decode(returndata, (address));
    }

    /**
     * @dev Returns the current admin of `proxy`.
     *
     * Requirements:
     *
     * - This contract must be the admin of `proxy`.
     */
    function getProxyAdmin(TransparentUpgradeableProxy proxy) public view virtual returns (address) {
        // We need to manually run the static call since the getter cannot be flagged as view
        // bytes4(keccak256("admin()")) == 0xf851a440
        (bool success, bytes memory returndata) = address(proxy).staticcall(hex"f851a440");
        require(success);
        return abi.decode(returndata, (address));
    }

    /**
     * @dev Changes the admin of `proxy` to `newAdmin`.
     *
     * Requirements:
     *
     * - This contract must be the current admin of `proxy`.
     */
    function changeProxyAdmin(TransparentUpgradeableProxy proxy, address newAdmin) public virtual onlyOwner {
        proxy.changeAdmin(newAdmin);
    }

    /**
     * @dev Upgrades `proxy` to `implementation`. See {TransparentUpgradeableProxy-upgradeTo}.
     *
     * Requirements:
     *
     * - This contract must be the admin of `proxy`.
     */
    function upgrade(TransparentUpgradeableProxy proxy, address implementation) public virtual onlyOwner {
        proxy.upgradeTo(implementation);
    }

    /**
     * @dev Upgrades `proxy` to `implementation` and calls a function on the new
     implementation. See
     * {TransparentUpgradeableProxy-upgradeToAndCall}.
     *
     * Requirements:
     *
     * - This contract must be the admin of `proxy`.
     */
    function upgradeAndCall(
        TransparentUpgradeableProxy proxy,
        address implementation,
        bytes memory data
    ) public payable virtual onlyOwner {
        proxy.upgradeToAndCall{value: msg.value}(implementation, data);
    }
}
```