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

pragma solidity ^0.8.0;

/**
 * @dev https://eips.ethereum.org/EIPS/eip-1167[EIP 1167] is a standard for
 * deploying minimal proxy contracts, also known as "clones".
 *
 * > To simply and cheaply clone contract functionality in an immutable way, this standard specifies
 * > a minimal bytecode implementation that delegates all calls to a known, fixed address.
 *
 * The library includes functions to deploy a proxy using either `create` (traditional deployment) or `create2`
 * (salted deterministic deployment). It also includes functions to predict the addresses of clones deployed using the
 * deterministic method.
 *
 * _Available since v3.4._
 */
library Clones {
    /**
     * @dev Deploys and returns the address of a clone that mimics the behaviour of `implementation`.
     *
     * This function uses the create opcode, which should never revert.
     */
    function clone(address implementation) internal returns (address instance) {
        /// @solidity memory-safe-assembly
        assembly {
            // Cleans the upper 96 bits of the `implementation` word, then packs the first 3 bytes
            // of the `implementation` address with the bytecode before the address.
            mstore(0x00, or(shr(0xe8, shl(0x60, implementation)), 0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000))
            // Packs the remaining 17 bytes of `implementation` with the bytecode after the address.
            mstore(0x20, or(shl(0x78, implementation), 0x5af43d82803e903d91602b57fd5bf3))
            instance := create(0, 0x09, 0x37)
        }
        require(instance != address(0), "ERC1167: create failed");
    }

    /**
     * @dev Deploys and returns the address of a clone that mimics the behaviour of `implementation`.
     *
     * This function uses the create2 opcode and a `salt` to deterministically deploy
     * the clone. Using the same `implementation` and `salt` multiple time will revert, since
     * the clones cannot be deployed twice at the same address.
     */
    function cloneDeterministic(address implementation, bytes32 salt) internal returns (address instance) {
        /// @solidity memory-safe-assembly
        assembly {
            // Cleans the upper 96 bits of the `implementation` word, then packs the first 3 bytes
            // of the `implementation` address with the bytecode before the address.
            mstore(0x00, or(shr(0xe8, shl(0x60, implementation)), 0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000))
            // Packs the remaining 17 bytes of `implementation` with the bytecode after the address.
            mstore(0x20, or(shl(0x78, implementation), 0x5af43d82803e903d91602b57fd5bf3))
            instance := create2(0, 0x09, 0x37, salt)
        }
        require(instance != address(0), "ERC1167: create2 failed");
    }

```

```
58        /**
59         * @dev Computes the address of a clone deployed using
          {Clones-cloneDeterministic}.
60         */
61        function predictDeterministicAddress(
62            address implementation,
63            bytes32 salt,
64            address deployer
65        ) internal pure returns (address predicted) {
66            /// @solidity memory-safe-assembly
67            assembly {
68                let ptr := mload(0x40)
69                mstore(add(ptr, 0x38), deployer)
70                mstore(add(ptr, 0x24), 0x5af43d82803e903d91602b57fd5bf3ff)
71                mstore(add(ptr, 0x14), implementation)
72                mstore(ptr, 0x3d602d80600a3d3981f3363d3d373d3d3d363d73)
73                mstore(add(ptr, 0x58), salt)
74                mstore(add(ptr, 0x78), keccak256(add(ptr, 0x0c), 0x37))
75                predicted := keccak256(add(ptr, 0x43), 0x55)
76            }
77        }
78
79        /**
80         * @dev Computes the address of a clone deployed using
          {Clones-cloneDeterministic}.
81         */
82        function predictDeterministicAddress(address implementation, bytes32 salt)
83            internal
84            view
85            returns (address predicted)
86        {
87            return predictDeterministicAddress(implementation, salt, address(this));
88        }
89    }
90
```