Hardhat DAOs

```solidity
import "@openzeppelin/contracts/access/Ownable.sol";

contract Box is Ownable {
  uint256 private value;

  // Emitted when the stored value changes
  event ValueChanged(uint256 newValue);

  // Stores a new value in the contract
  function store(uint256 newValue) public onlyOwner {
    value = newValue;
    emit ValueChanged(newValue);
  }

  // Reads the last stored value
  function retrieve() public view returns (uint256) {
    return value;
  }
}
```

```solidity
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";

contract GovernanceToken is ERC20Votes {
  uint256 public s_maxSupply = 1000000000000000000000000;

  constructor() ERC20("GovernanceToken", "GT") ERC20Permit("GovernanceToken") {
    _mint(msg.sender, s_maxSupply);
  }

  // The functions below are overrides required by Solidity.

  function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal override(ERC20Votes) {
    super._afterTokenTransfer(from, to, amount);
  }

  function _mint(address to, uint256 amount) internal override(ERC20Votes) {
    super._mint(to, amount);
  }

  function _burn(address account, uint256 amount) internal override(ERC20Votes) {
    super._burn(account, amount);
  }
}
```

```
    function _execute(
        uint256 proposalId,
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        bytes32 descriptionHash
    ) internal override(Governor, GovernorTimelockControl) {
        super._execute(proposalId, targets, values, calldatas, descriptionHash);
    }

    function _cancel(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        bytes32 descriptionHash
    ) internal override(Governor, GovernorTimelockControl) returns (uint256) {
        return super._cancel(targets, values, calldatas, descriptionHash);
    }

    function _executor() internal view override(Governor, GovernorTimelockControl) returns (address) {
        return super._executor();
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(Governor, GovernorTimelockControl)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```

```sol
import "@openzeppelin/contracts/governance/TimelockController.sol";

contract TimeLock is TimelockController {
  // minDelay is how long you have to wait before executing
  // proposers is the list of addresses that can propose
  // executors is the list of addresses that can execute
  constructor(
    uint256 minDelay,
    address[] memory proposers,
    address[] memory executors
  ) TimelockController(minDelay, proposers, executors) {}
}
```

```sol
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Wrapper.sol";

contract MyToken is ERC20, ERC20Permit, ERC20Votes, ERC20Wrapper {
  constructor(IERC20 wrappedToken)
    ERC20("MyToken", "MTK")
    ERC20Permit("MyToken")
    ERC20Wrapper(wrappedToken)
  {}

  // The functions below are overrides required by Solidity.

  function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal override(ERC20, ERC20Votes) {
    super._afterTokenTransfer(from, to, amount);
  }

  function _mint(address to, uint256 amount) internal override(ERC20, ERC20Votes) {
    super._mint(to, amount);
  }

  function _burn(address account, uint256 amount) internal override(ERC20, ERC20Votes) {
    super._burn(account, amount);
  }
}
```