ownable.sol

```
pragma solidity ^0.4.25;

/**
* @title Ownable
* @dev The Ownable contract has an owner address, and provides basic
authorization control
* functions, this simplifies the implementation of "user permissions".
*/
contract Ownable {
  address private _owner;

  event OwnershipTransferred(
    address indexed previousOwner,
    address indexed newOwner
  );

  /**
  * @dev The Ownable constructor sets the original `owner` of the contract to
the sender
  * account.
  */
  constructor() internal {
    _owner = msg.sender;
    emit OwnershipTransferred(address(0), _owner);
  }

  /**
  * @return the address of the owner.
  */
  function owner() public view returns(address) {
    return _owner;
  }

  /**
  * @dev Throws if called by any account other than the owner.
  */
  modifier onlyOwner() {
    require(isOwner());
    _;
  }

  /**
```

```
  * @return true if `msg.sender` is the owner of the contract.
  */
  function isOwner() public view returns(bool) {
    return msg.sender == _owner;
  }

  /**
  * @dev Allows the current owner to relinquish control of the contract.
  * @notice Renouncing to ownership will leave the contract without an owner.
  * It will not be possible to call the functions with the `onlyOwner`
  * modifier anymore.
  */
  function renounceOwnership() public onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
  }

  /**
  * @dev Allows the current owner to transfer control of the contract to a
newOwner.
  * @param newOwner The address to transfer ownership to.
  */
  function transferOwnership(address newOwner) public onlyOwner {
    _transferOwnership(newOwner);
  }

  /**
  * @dev Transfers control of the contract to a newOwner.
  * @param newOwner The address to transfer ownership to.
  */
  function _transferOwnership(address newOwner) internal {
    require(newOwner != address(0));
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
  }
}
```