pragma solidity ^0.4.0;

contract owned {

**function** **owned**() { owner = msg.sender; }

address owner;

*// This contract only defines a modifier but does not use*

*// it - it will be used in derived contracts.*

*// The function body is inserted where the special symbol*

*// "\_;" in the definition of a modifier appears.*

*// This means that if the owner calls this function, the*

*// function is executed and otherwise, an exception is*

*// thrown.*

modifier onlyOwner {

**if** (msg.sender != owner)

**throw**;

\_;

}

}

contract mortal **is** owned {

*// This contract inherits the "onlyOwner"-modifier from*

*// "owned" and applies it to the "close"-function, which*

*// causes that calls to "close" only have an effect if*

*// they are made by the stored owner.*

**function** **close**() **onlyOwner** {

selfdestruct(owner);

}

}

contract priced {

*// Modifiers can receive arguments:*

modifier costs(uint price) {

**if** (msg.value >= price) {

\_;

}

}

}

contract Register **is** priced, owned {

mapping (address => bool) registeredAddresses;

uint price;

**function** **Register**(uint initialPrice) { price = initialPrice; }

*// It is important to also provide the*

*// "payable" keyword here, otherwise the function will*

*// automatically reject all Ether sent to it.*

**function** **register**() **payable** **costs**(price) {

registeredAddresses[msg.sender] = true;

}

**function** **changePrice**(uint \_price) **onlyOwner** {

price = \_price;

}

}