Lab 2: ERC20 Token Contract

Metamask

For a later lab, let’s get metamask installed so we can interact with Ropsten, rather than relying on a Java VM for our contracts.

https://metamask.io/download.html

Note, works for Chrome and Firefox only.

ERC20 Token Contract

An ERC20 token contract has to confirm to the following interface:

```
interface ERC20Interface {
   function totalSupply() external view returns (uint256);
   function balanceOf(address account) external view returns (uint256);
   function allowance(address owner, address spender) external view returns (uint256);
   function transfer(address recipient, uint256 amount) external returns (bool);
   function approve(address spender, uint256 amount) external returns (bool);
   function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

6 functions, 2 events (which get written to logs)
https://ethereum.stackexchange.com/questions/11228/what-is-an-event

totalSupply is the totalSupply of the token.

balanceOf returns the balance of an account

allowance returns the amount of tokens approved by the owner that can be transferred to the spender's account

transfer transfers the balance from token owner's account to recipient account

approve allows the token owner to approve a \`spender\` to transferFrom(...) *amount* of tokens from the token owner's account

transferFrom transfers amount of tokens from the *sender* account to the *recipient* account

If a contract conforms to the interface, then it's a valid ERC20 contract. There are security audited examples out there, but let's build the simplest (but insecure!) one we can that conforms to the standard.

Okay, so let’s implement these methods in a contract.

```
 pragma solidity ^0.6.6;

 interface ERC20Interface {
   function totalSupply() external view returns (uint256);
   function balanceOf(address account) external view returns (uint256);
   function allowance(address owner, address spender) external view returns (uint256);
   function transfer(address recipient, uint256 amount) external returns (bool);
   function approve(address spender, uint256 amount) external returns (bool);
   function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract myToken is ERC20Interface {

        string public symbol;
        string public name;
        uint8 public decimals;
        uint public _totalSupply;
        address public tokenOwner;

        mapping(address => uint) private _balances;
        mapping(address => mapping(address => uint256)) private _allowances;

        constructor() public {
        tokenOwner = msg.sender;
        symbol="TOK";
        name="Fixed supply token";
        decimals=18;
        _totalSupply = 1000000 * 10**uint(decimals);
        _balances[tokenOwner] = _totalSupply;
        emit Transfer(address(0)[1], tokenOwner, _totalSupply);
        }

        function totalSupply() public view override returns (uint256) {
        return _totalSupply - _balances[address(0)];
        }

        function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
        }

        function allowance(address owner, address spender) public view virtual override returns
(uint256) {
        return _allowances[owner][spender];
        }

        function transfer(address recipient, uint256 amount) public virtual override returns (bool)
{
        address sender = msg.sender;

        _balances[sender] = _balances[sender] - amount;
        _balances[recipient] = _balances[recipient] + amount;
        emit Transfer(sender, recipient, amount);
        return true;
        }

        function approve(address spender, uint256 amount) public virtual override returns (bool) {
```

[1] https://stackoverflow.com/questions/48219716/what-is-address0-in-solidity

```
        address sender = msg.sender;

        _allowances[sender][spender] = amount;
        emit Approval(sender, spender, amount);
        return true;
        }

        function transferFrom(address sender, address recipient, uint256 amount) public virtual
override returns (bool) {

        _balances[sender] = _balances[sender] - amount;
        _balances[recipient] = _balances[recipient] + amount;
        emit Transfer(sender, recipient, amount);

        _allowances[sender][recipient] = amount;
        emit Approval(sender, recipient, amount);
        return true;
        }
}
```

Exercises

1. Get it compiling and testable!
2. Change the token name and the totalSupply to customise the contract to your token.
3. There are obvious fatal flaws in the implementation that would make it very risky to deploy as an actual ERC20! Find some!
4. There's some repeated code in the transferFrom method that is repeated from the *transfer* and *approve* methods. Refactor that repeated code into additional methods (_transfer and _approve) and remove the repeated code.
5. Add a check to ensure the addresses passed into _transfer and _approve aren't from the address(0) address.

```
require(sender != address(0), "ERC20: transfer from the zero address");
require(recipient != address(0), "ERC20: transfer to the zero address");
```

6. Look at changing the unsafe maths over to the safe maths of the https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol contract.

   Syntax of
   ```
   _balances[sender] = _balances[sender] - amount;
   ```

   would change to

   ```
   _balances[sender] = _balances[sender].sub(amount, "transfer amount exceeds balance");
   ```

```
library SafeMath {

        function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
        }

        function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
        }


        function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
        }

        function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
                return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
        }


        function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
        }


        function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
        }


        function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
        }


        function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b != 0, errorMessage);
        return a % b;
        }
}
```

To use this library, drop the code into remix and add this line to your contract:

```
contract myToken is ERC20Interface {
        using SafeMath for uint256;
```