Лабораторная работа 4

Основы Solidity. Смарт-контракты для бизнеса

Теоретический блок

Наследование.

- Solidity поддерживает множественное наследование.
- Когда контракт наследуется от других контрактов, в блокчейне создается только один контракт, и код из всех базовых контрактов компилируется в созданный контракт.
- Для наследования используется ключевое слово `**is**`.
- Унаследованные смарт-контракты могут иметь доступ ко всем не приватным (non-private) членам наследуемого класса, включая intenal функции и переменные состояния.
- Ключевое слово **super** используется для доступа к контракту на один уровень выше в иерархии наследования.
- Функции могут быть **переопределены** другой функцией с тем же названием функции и тем же количеством / типами входных параметров.
- Если конструктор принимает аргумент, он должен быть предоставлен в заголовке при объявлении наследования (см. в примерах ниже)
- Документация: https://solidity.readthedocs.io/en/v0.5.12/contracts.html#inheritance

Пример наследования:

```
contract Owned {
    constructor() public { owner = msg.sender; }
    address payable owner;
}

contract Mortal is Owned {
    function kill() public {
        if (msg.sender == owner) selfdestruct(owner);
    }
}
```

Пример множественного наследования:

```
contract Named is Owned, Mortal {
    constructor(bytes32 name) public {
      // some code
    }
    function kill() public {
        if (msg.sender == owner) { Mortal.kill();}
    }
}

contract PriceFeed is Owned, Mortal, Named("GoldFeed") {
    function updateInfo(uint newInfo) public {
        if (msg.sender == owner) info = newInfo;
    }

    function get() public view returns(uint r) { return info; }
}
```

Абстрактные смарт-контракты.

- Контракты являются абстрактными, когда хотя бы у одной из функций отсутствует реализация.
- Не могут быть скомпилированы.
- Могут использовать в качестве базовых контрактов.
- Также контракт является абстрактным, если он наследуется от абстрактного контракта и не реализует все функции путем переопределения.
- Документация: https://solidity.readthedocs.io/en/v0.5.12/contracts.html#abstract-contracts

Пример абстрактного смарт-контракта:

```
contract Feline {
    function utterance() public returns (bytes32);
}
```

Интерфейсы.

- похожи на абстрактные контракты, но они не могут иметь реализованных функций
- не могут наследовать другие контракты или интерфейсы
- не могут иметь конструктор.
- не могут иметь переменных состояния.
- все функции должны быть external
- обозначаются ключевым словом interface
- Документация: https://solidity.readthedocs.io/en/v0.5.12/contracts.html#interfaces

Пример интерфейса:

```
interface Token {
    enum TokenType { Fungible, NonFungible }
    struct Coin { string obverse; string reverse; }
    function transfer(address recipient, uint amount) external;
}
```

Библиотеки

- Библиотеки в Solidity подобны контрактам, но они деплоятся один раз на конкретный адрес, и их код повторно используется с помощью **delegatecall** (С помощью delegatecall можно вызвать функцию другого смарт-контракта в контексте вызывающего смарт-контракта. То есть данные читаются из памяти текущего контракта.). Это означает, что, если функции библиотеки вызваны, то их код выполняется в контексте вызывающего контракта, то есть this указывает на вызывающий контракт, и доступно хранилище из вызывающего контракта.
- не могут наследовать или наследоваться
- не могут получать эфир
- Директива **using A for B;** используется, чтобы прикрепить функции библиотеки (из библиотеки A) к любому типу (B). Эти функции получат объект, который они вызывают, в качестве их первого параметра.
- internal-функции библиотек видны всем контрактам
- Документация: https://solidity.readthedocs.io/en/v0.5.12/contracts.html#libraries

Практический блок

1. **Написать смарт-контракт согласно требованиям ниже и скомпилировать его в IDE Remix.**

Требования к смарт-контракту:

1) Бизнес-требования:
    a. Написать смарт-контракт, который будет осуществлять запись на прием к врачам поликлиники на определенную дату (отдельно передавать день, месяц, год) и время за определенное количество эфиров и переводить их на адрес поликлиники.
    b. Также должна быть возможность отмены записи без возврата комиссии.
    c. **!** Запись на приём и отмена возможны только за день до самого обращения
    ПОДСКАЗКА (предлагаемый вариант решения): т.е. у нас должна быть переменная состояния, которая бы каждый день обновлялась и хранила текущую дату.
    Дату можно хранить в виде структуры (struct) с полями день, месяц, год.
    Нужна функция для обновления даты раз в сутки администратором
    P.S. можно придумать другие варианты хранения данных (это даже приветствуется)
    d. Сотрудник больницы – адрес, который инициализируется во время загрузки смарт-контракта
    e. У сотрудника больницы должна быть возможность получить подтверждение по *адресу, времени и ФИО* (или, например, по *идентификационному номеру*) *врача* о том, что данный аккаунт имеет запись на приём.
    f. Добавление врачей, к которым можно записать на приём, осуществляет только владелец смарт-контракта
    ПОДСКАЗКА (предлагаемый вариант решения): список врачей хранить в маппинге, где ключом будет ИД врача, а значением (true, false). True – если такой врач существует в системе, false – такого врача не существует.
    g. Смарт-контракт должен содержать следующие элементы:
        1. **конструктор** – для инициализации адреса поликлиники, размера комиссии за запись на приём, сотрудника поликлиники
        2. **основные функции** (запись на приём, отмена записи, проверка записи для сотрудников, добавление врачей для владельца и ещё функция обновления текущей даты для владельца)
        3. **fallback-функцию**
        4. где необходимо – ключевое слово **payable**
        5. **модификаторы** (для ограничения доступа только для сотрудников у функции проверки существования записи)
        6. **события** (перевод комиссии, запись прошла успешно, запись отменена)
        7. **видимость функций** и переменных состояния (public, private, external, internal)

8. функцию **kill**, которая деактивирует смарт-контракт (доступна только для владельца)

2) Версия Solidity: 0.5.11
3) Создать смарт-контракт **Ownable** и унаследовать от него основной смарт-контракт. Смарт-контракт Ownable должен содержать:
   a. модификатор **onlyOwner**
   b. функцию, которая возвращает адрес владельца смарт-контракта
   c. конструктор, в котором инициализируется владелец смарт-контракта
   d. функцию с названием **transferOwnership (address newOwner**), позволяющую изменить владельца смарт-котракта. Эта функция должна быть доступна только текущему владельцу

Дополнительные требования (ВЫПОЛНЯТЬ НЕ ОБЯЗАТЕЛЬНО):

1) Организовать систему ролей (администраторы, сотрудники больницы) Для этого использовать библиотеку https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Roles.sol
2) Только владелец может добавлять администраторов
3) Вместо одного сотрудника больницы в контракте их может быть их несколько (то есть роль «сотрудник» могут иметь несколько аккаунтов)
4) Администраторы могут добавлять сотрудников больницы и врачей, к которым можно записаться на приём

2. Задеплоить смарт-контракт в тестовую сеть Ropsten используя Metamask и IDE Remix.
   В отчёте предоставить **адрес** задеплоенного смарт-контракта.
   Отдельным файлом приложить **код** смарт-контракта (файл с расширением .sol)

Контрольные вопросы:

1) Видны ли internal-функции библиотек всем контрактам?
2) Могут ли библиотеки наследовать или наследоваться?
3) Каким ключевым словом обозначаются абстрактные классы?
4) Что такое интерфейсы? Какая их особенность?
5) Существует специальный вариант вызова сообщения с помощью delegatecall. Какова его особенность?

Задание

1) Изучить теоретический материал

2) Выполнить практическую часть лабораторной работы

3) Ответить на контрольные вопросы

4) Оформить **отчёт**

5) Оформить **файл смарт-контракта с кодом** (файл имеет расширение **.sol**) **!!!!**

Защита лабораторной работы 4: **отчёт** и **файл с кодом смарт-контракта** должны быть отправлены на портал. То есть на портал должно быть отправлено 2 файла!

Общие требования к отчёту

Отчёт должен содержать следующие элементы:

1) номер и название лабораторной работы,

2) ФИО студента, группу и курс

3) выполненные задания практического блока: скриншоты с etherscan.io, metamask и Remix и **!!!!!адрес смарт-контракта**

4) краткие ответы на контрольные вопросы.

В случае обнаружения плагиата (в том числе и в ответах на контрольные вопросы) отчёт не будет принят преподавателем!