CST4125: Blockchain Development
Week: 17
Title: OOP & Private Networks

Dr Ian Mitchell

Bedfordshire,
UK

January 2023

Contact and Office Hours

Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: smerf.net

Contact and Office Hours

Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: smerf.net

Office Hours

- During term time only
- When: Autumn Term: Mondays 1100-1300hrs
- Please read notifications or emails
- There are occassions that these could be arranged online, e.g., due to industrial action or inclement weather

Deadlines

				Feedback	
Description	Submission	Weight	Deadline	Formative	Summative
1. Hyperledger	MyLearning	50%	18th December 2022	LW11-12	12/01/2023
2. Ethereum	MyLearning	50%	2nd April 2023	LW23-24	24/04/2023
Resits	MyLearning	50-100%	1st July 2023	None	None
Deferals	MyLearning	50-100%	1st July 2023	None	None

Lecture Objectives

- Object-Oriented
- EOA
- Contract
- Payable
- Interfaces
- Creating Contracts
- Private Networks
- Truffle and Ganache
- Metamask

Ethereum SC

- Solidity is used to write SC
- Object-Oriented
- Code executed in EVM
- Similar to C++ class
- Composed of state and behaviour
- Contracts have addresses
- Senders have addresses, EOA
- Inheritance and aggregate classes
- Encapsulation

Recap

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.0;
contract A{
  uint x;
  function setX(uint _x) public { x=_x;}
  function getX() public returns (uint){ return x;}
  }
contract B{
  function useNew() public returns (uint) {
    A a = new A();
    a.setX(100);
    return a.getX();
  }
}
```


Object-Oriented Programming

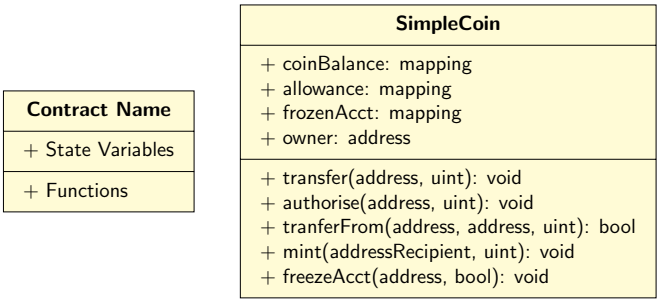
- Contract ≡ class
- Implementation of a contract is an object
- Behaviour
- Structure
- UML: Class diagrams become Contract Diagrams
- Contracts can inherit
- Revision for UML, see [3]

Constructor

- What address is used, when creating a new instance?
- Constructors, are default
- Declare constructors
- Only one contructor, no overloading
- Constructor has the same name as the contract
- Constructor cannot return any values
- Can set state variables to default values

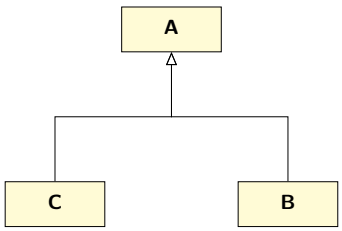
Contracts Diagram

taken from [1]



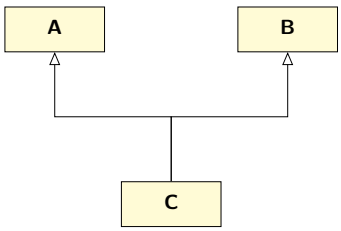
Contracts and Single Inheritance

```
contract A{}

contract B is A{}

contract C is A{}
```



Contracts and Multple Inheritance

```
contract A{}

contract C is A,B{}
```



Polymorphism

Functional Polymorphism

- Function name is same
- Different datatype for Function parameters
- Different number of function parameters
- Return types are not considered
- Method overloading

Contract Polymorphism

- Method overriding
- Function Named X in Class A
- Function Named X in Class B
- B inherits A
- In the instance of B, which function gets executed?
- Example

Functional Polymorphism

```
1  //SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3  contract example{
4    function double(int8 x) public pure returns(int16 y)
       {
5      return x + x;
6    }
7    function double(int16 x) public pure  returns(int32
       y){
8      return x + x;
9    }
10 }
```


Override

```
1  contract Parent{
2      uint public baseCost;
3      event displayCost(uint x);
4      function setBaseCost(uint x) public {
5          baseCost=x;
6      }
7      function cost(uint delivery) public  virtual returns (uint) {
8          uint total=baseCost+delivery;
9          emit displayCost(total);
10         return total;
11     }
12 }
13 contract Child is Parent {
14     function cost(uint delivery) public override returns (uint) {
15         uint intl=100;
16         uint total= baseCost+delivery+intl;
17         emit displayCost(total);
18         return total;
19     }
20 }
21 contract example {
22     Parent UK = new Parent();
23     Parent India = new Child();
24     function testUK() public{
25         UK.setBaseCost(100);
26         UK.cost(10);
27     }
28     function testIndia() public {
29         India.setBaseCost(100);
30         India.cost(10);
31     }
32 }
```


Abstract Contracts

- Partial function declaration
- No instance of an abstract contract
- Must be inherited by a child
- Help in defining structure of a contract
- Any contract inheriting this form must implement functions not defined
- Consistent
- Code reuse
- Abstract keyword
- In older versions
- Contract becomes abstract if there are no implementations of the functions

Abstract Contract

Definition [1]

contract is considered *abstract* if it contains at least one declared but unimplemented function. An abstract contract is used as a base class for other contracts, but it can't be instantiated. A contract whose functions have all been implemented is considered a *concrete* contract.

Abstract Contract

adapted from soliditylang.org

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.6.0 <0.9.0;
3
4  abstract contract Feline {
5      function utterance() public pure virtual returns (
       bytes32);
6  }
7
8  contract Cat is Feline {
9      function utterance() public pure override returns
       (bytes32) { return "miaow"; }
10 }
11
12 contract Lion is Feline {
13     function utterance() public pure override returns
       (bytes32) {return "roar";}
14 }
```

Interface

- Similar to Java
- Defines all methods for a contract to implement
- Structure and consistency
- Interface does not implement any methods
- Can define complex types, not state variables
- Can't define variables
- Can't define constructors
- Can't define modifiers
- Code reuse
- Can inherit other interfaces, not other Contracts

Address Functions

Address

- `address.balance`
- `address.transfer(amount)`
- `address.send(amount)`
- `address.call(payload)`
- `address.callcode(payload)`
- `address.delegatecall()`

`send`

`transfer`

`call`

Fallback

Definition

Something to which one can resort or retreat

- Using a smart contract's functions
- Make a call to a function that does not exist?
- When the fallback function is invoked

Fallback

Definition

Something to which one can resort or retreat

- Using a smart contract's functions
- Make a call to a function that does not exist?
- When the fallback function is invoked

Solidity Definition

A fallback function is invoked when the function call does not match any of the function names

Fallback Rules

```
1 //SPDX-License-Identifier:
      MIT
2 pragma solidity ^0.8.0;
3 contract fallbackExample{
4   function () { uint a
      =10;}
5 }
```

- Only one fallback function per contract
- No parameters
- No function name
- No return values
- Gas required to execute a fallback function
- Limit Gas consumption to 2,300Wei
- Test fallback function does not exceed gas consumption
- Security lapses
- Can have a payable fallback function

Application Development Lifecycle Management

- Design, Develop, Deploy & Test
- Requirements capture
- Architects documents these
- Composed of sprints
- Eventually, coded into contracts, data and functions
- Deployed
- Tested
- Need for tools

Truffle

- Application Lifecycle Management (ALM)
- Need supporting tools
- Decentralisation is different
- Truffle accommodates some of these differences
- Help in the DevOps process
- Truffle is a utility for blockchain development that:
  - Works with Ethereum and other blockchain development environments
  - Compile smart contracts
  - Deploy smart contracts
  - Test smart contracts
- Overall: makes production more efficient

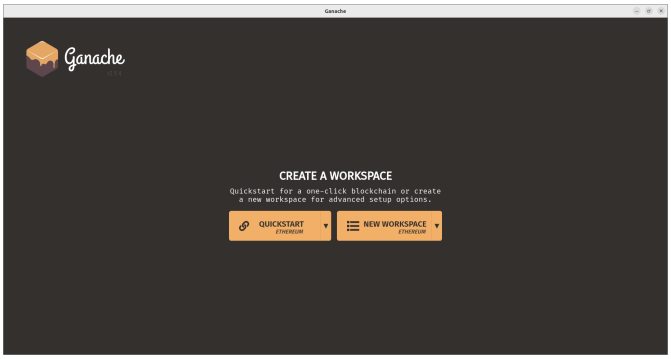
Installing Truffle

- VM has Truffle pre-installed
- Truffle
- Debian:
  - `sudo apt update`
  - `sudo apt install nodejs`
  - `sudo apt install npm`
  - `sudo npm install -g truffle`
  - `truffle --version`

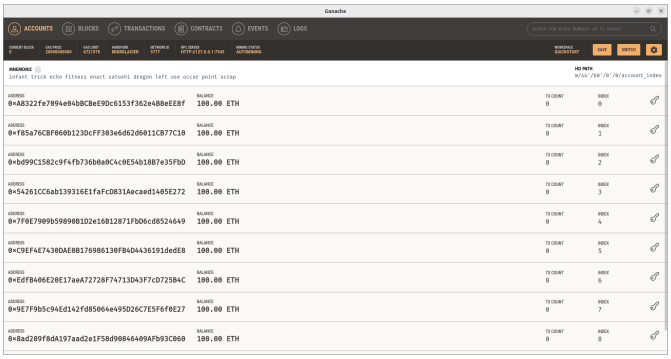
Ganache

- Another Tool
- Decentralisation
- Accounts, Transactions and Blocks
- User End
- GUI and CLI
- Installation (Debian) & Implementation:
  - Visit Ganache
  - Download and move to a new folder in Home directory, Ganache
  - Debian systems not supporting FUSE, `sudo apt install libfuse2`
  - Double click `AppImage` file

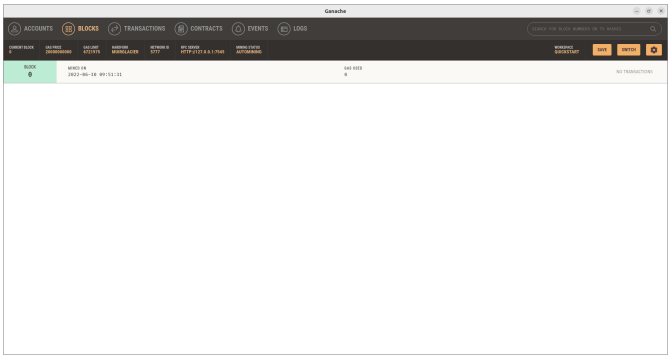
Ganache
Quickstart



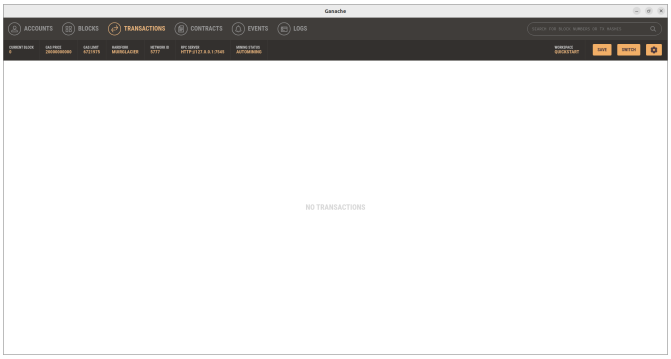
Ganache
Accounts



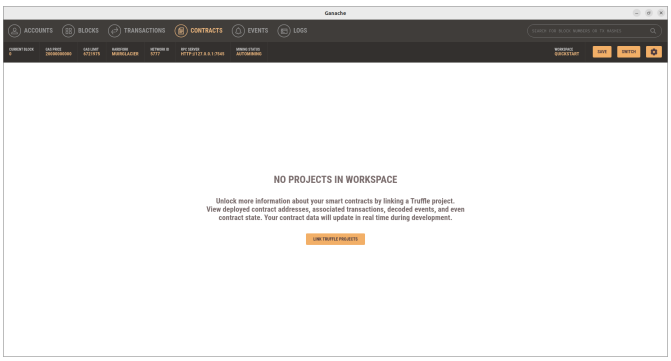
Ganache
Blocks



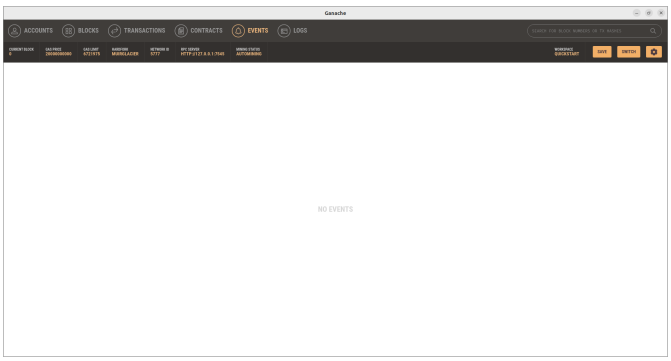
Ganache
Transactions



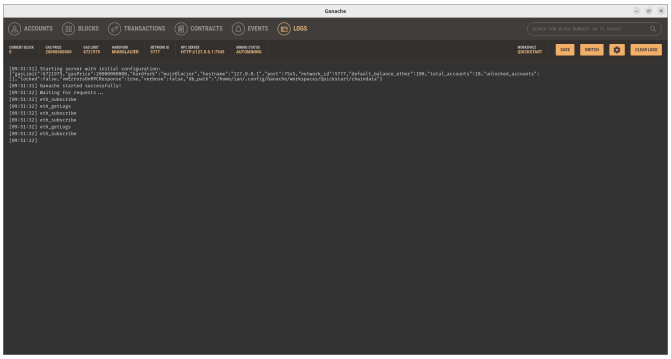
Ganache
Contracts



Ganache
Events



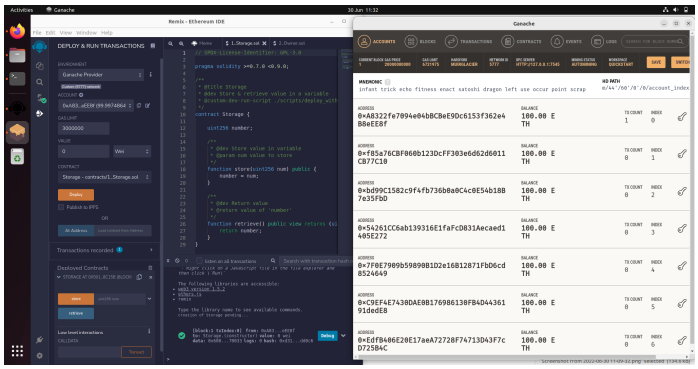
Ganache
Logs



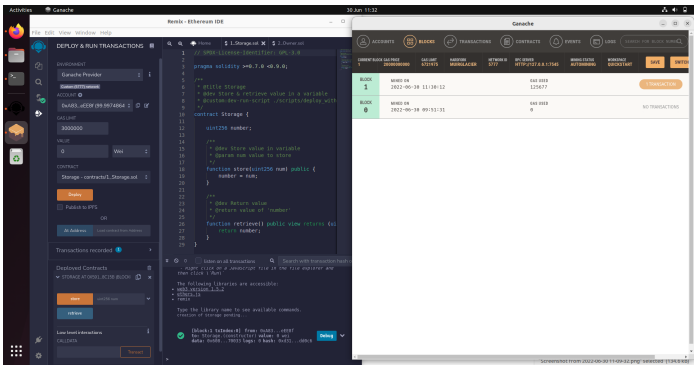
Remix Desktop

- Offline resource
- Link up to Ganache
- Have all functionality of remix online
- Debian Installation:
  - Remix-ide
  - Download AppImage (note about FUSE on previous slides)
  - Create a remix sub-directory in the home directory
  - Move the AppImage file to the remix directory
  - Change the permissions of the file to allow execution (`chmod +x fileName.AppImage`)
  - Double click (and be patient and wait)

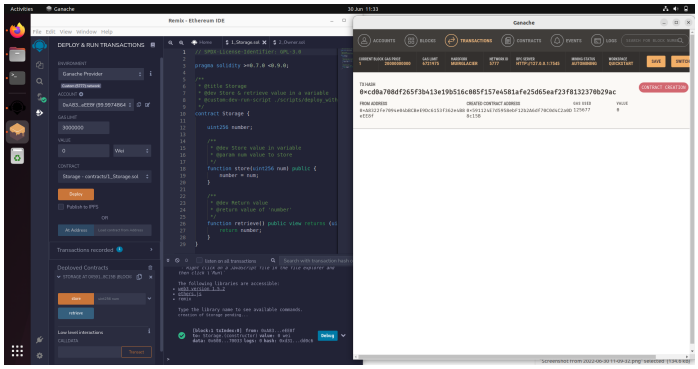
Ganache and Remix
Initialise & Setup



Ganache and Remix
Blocks



Ganache and Remix
Transactions



Send Ether

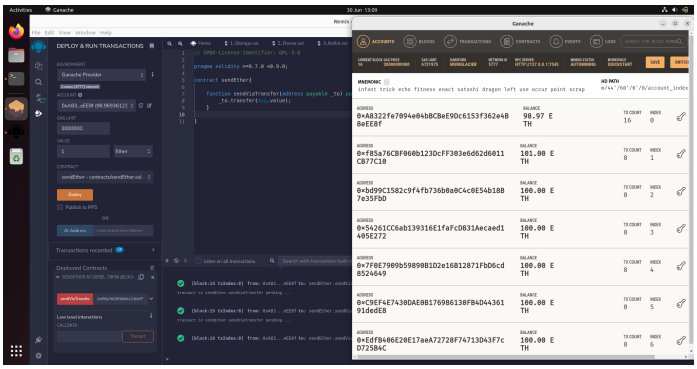
- Contract with 1 address: EOA
- Send ether from EOA to a nominated address
- Include a value of 1 ether to transfer

Send Ether

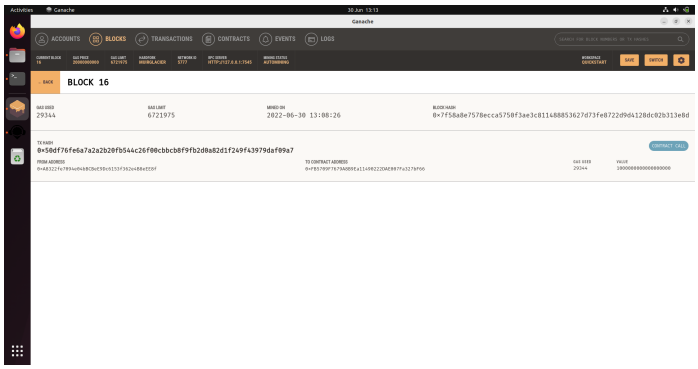
```
// SPDX-License-Identifier: GPL
    -3.0

pragma solidity >=0.7.0 <0.9.0;

contract sendEther{

    function sendViaTransfer(
    address payable _to) public
    payable{
        _to.transfer(msg.value)
    ;
    }

}
```

- Contract with 1 address: EOA
- Send ether from EOA to a nominated address
- Include a value of 1 ether to transfer

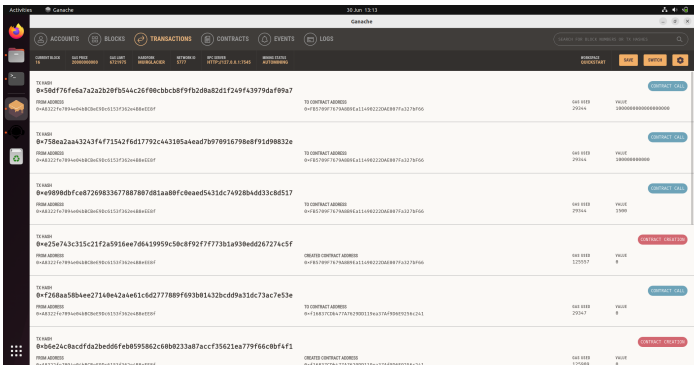
Send Ether
Example



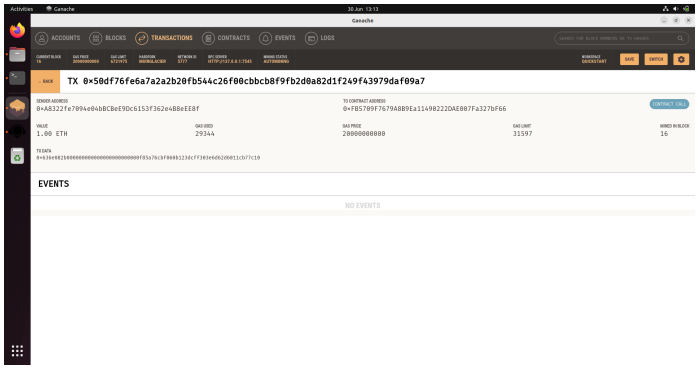
Send Ether
Blocks



Send Ether
Transactions



Send Ether
Transaction Details



Summary

- Private Networks
- Ganache
- Truffle
- Remix-ide (Desktop)
- Contract as a class (OOP)
- Fallback
- Transfer

Reading

Read Chapter 6 & 7 in [2]

References I

[1] R. Infante. *Building Ethereum Dapps*. Manning, 2019. ISBN: 9781617295157.

[2] Ritesh Modi. *Solidity Programming Essentials*. Packt, 2018. ISBN: 978-1-78883-138-3.

[3] Martina Seidl et al. *UML@ classroom: An introduction to object-oriented modeling*. Springer, 2015.