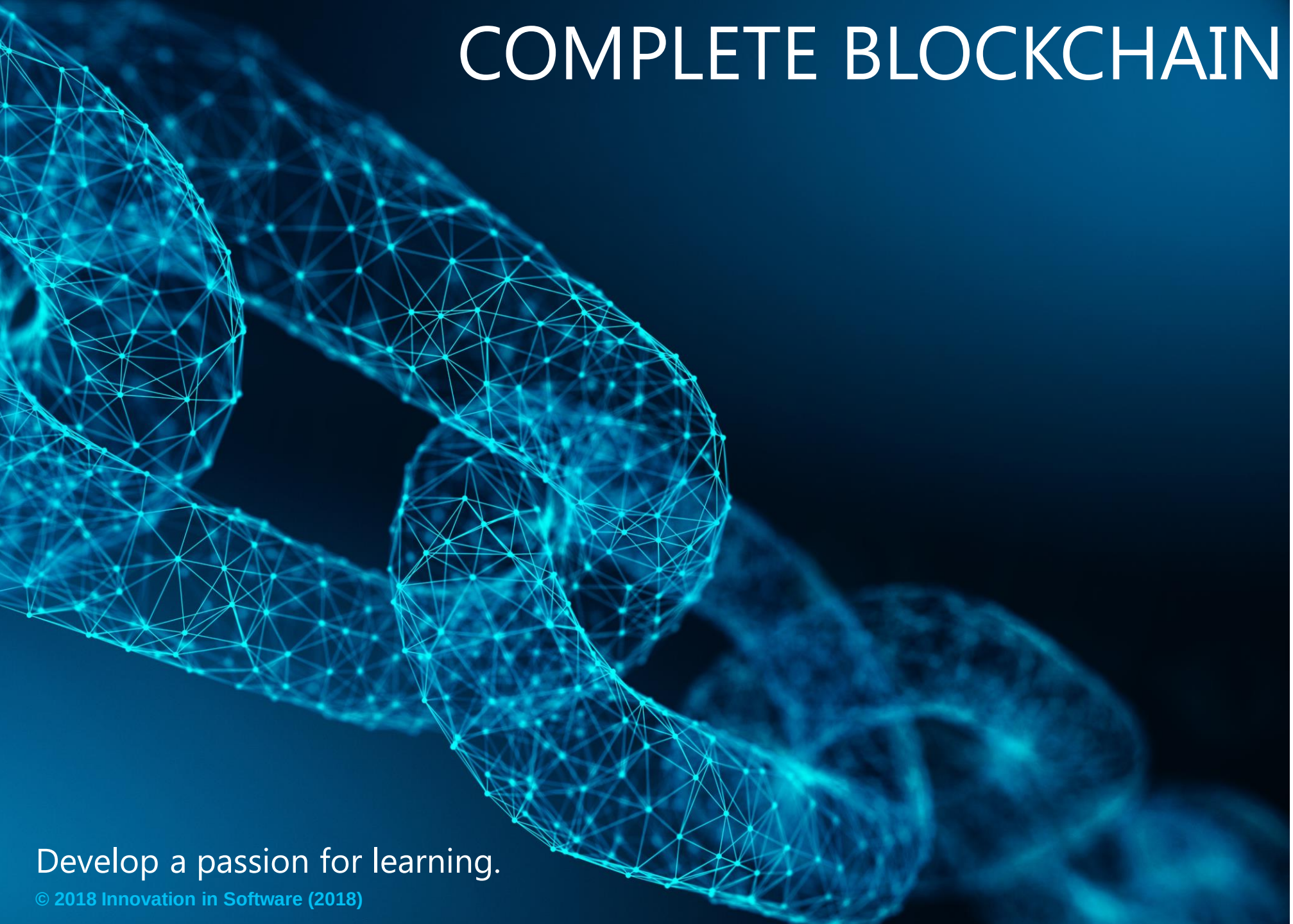
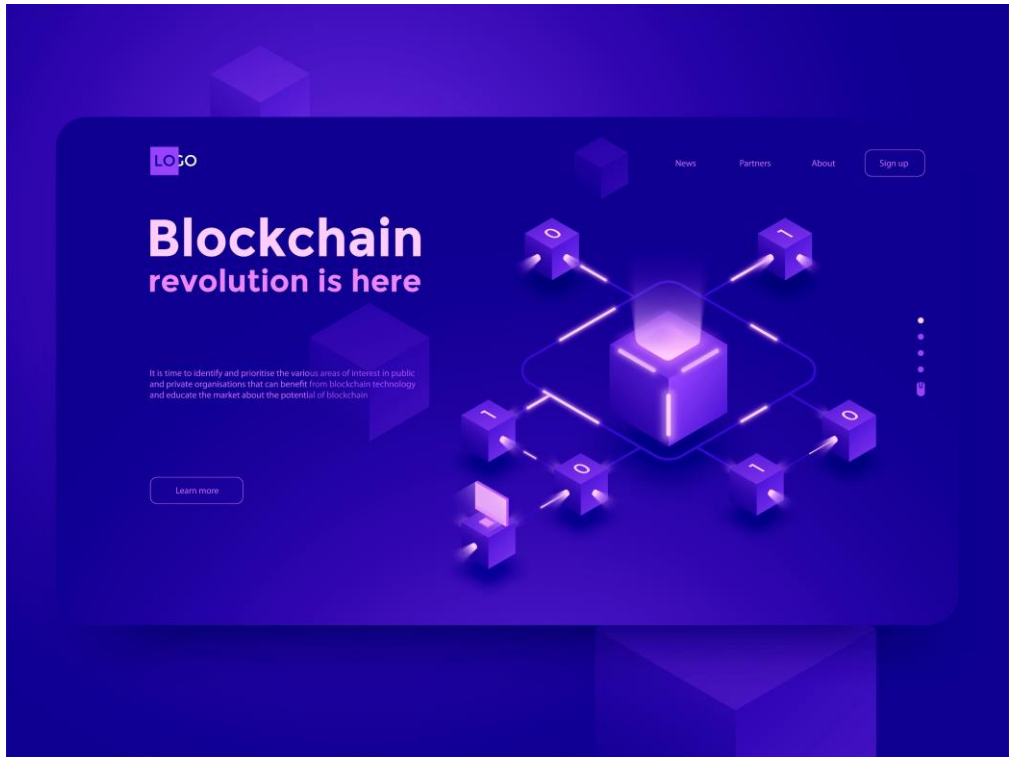
COMPLETE BLOCKCHAIN



Develop a passion for learning.

COMPLETE BLOCKCHAIN



In this module you will create a complete blockchain:

- Key pairs
- Transactions
- Mining
- Account balances
- Security
- Persistence
- And much more

HANDS ON EXPERIENCE

This project will take around three hours. Pair programming is recommended for this lab. This is not one person typing and another watching; but two active participants implementing and researching, when necessary, the solution.

Feel free to take breaks during this lab as necessary.

CLASSES

You will create several classes:

- TXOutput: output transactions
- TXInput: input transactions
- Block: blockchain block and related functionality
- BlockChain: wrapper for blockchain and related functions
- Proof Of Work: proof of work algorithm
- Transaction: amalgamation of input and output transactions
- Utilities: Helper methods

RECOMMENDATIONS

- Paired programming
- Use blockchain terminology
- Self documenting code
- Use exiting code when possible
- Build often

UTILITIES CLASS

Create two methods in the Utilities class.

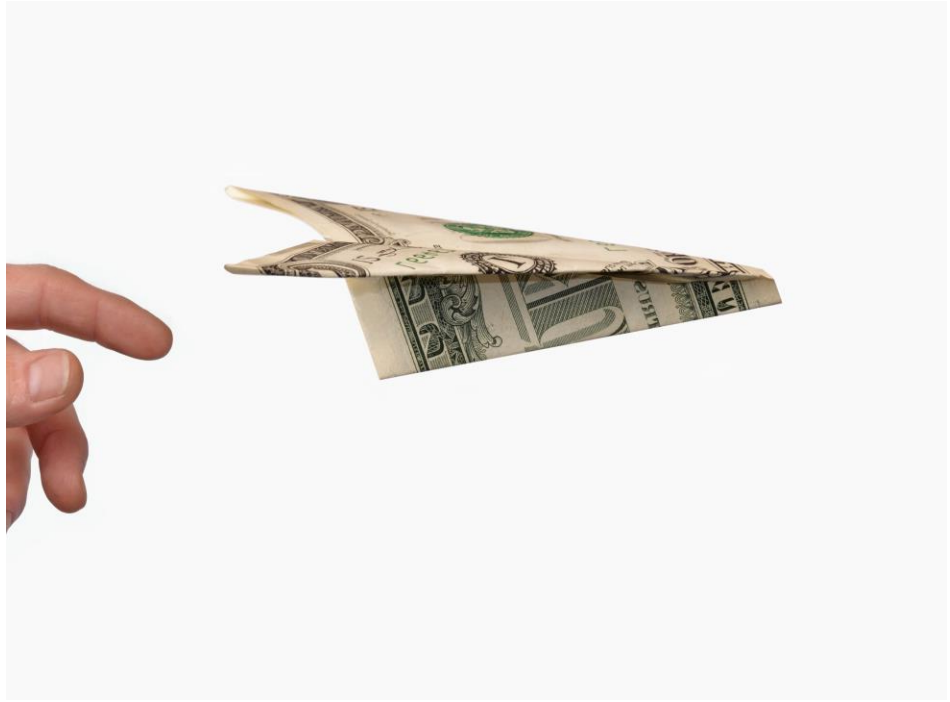
- **Hash2HexString.** Hash2HexString converts a hash to a displayable string. This method has one parameter, which is a byte array and returns a string.
- **SetHash**. SetHash converts a string to a hash and returns that hash as a displayable string. This method accepts a string as a parameter and returns a string.

TXOUTPUT CLASS

This class is for output transactions (i.e., sending money to someone).

- There are three fields:
  - coin value for transaction
  - person receiving the transaction
  - block transaction id
- Create a constructor that initializes the value and person fields. Transaction id is implemented as a count of total transactions. This is the row height.
- Create separate accessor methods to return the coin value, person, and transaction id.
- CanUnlockOutputWith. This method confirms the identity of the person. There is one string parameter and returns a Boolean. The string parameter is the expected identity. Returns true if user identifier matches the person of this transaction.

TXINPUT CLASS



This class is for input transactions. This is spent money from your perspective.

- There are two fields:

  transaction id of output id

  the person doing the transaction

- Create a constructor that initializes the two fields with the input parameters.
- CanUnlockOutputWith. This method confirms the identity of the person. There is one string parameter and returns a Boolean. The string parameter is the expected identity. Returns true if user identifier matches the person of this transaction.
- Create two accessor methods that return separately the transaction id and the person identity.

TRANSACTION CLASS

The Transaction class is a wrapper for the inputs and outputs of a transaction (debits and credits).

- There are three fields:

  transaction id

  array of input transactions

  array of output transactions

- Create a constructor that initializes the array of input and output transactions to constructor parameters.
- SetId. Initializes the transaction id to a count of transactions. Increments the count.
- NewCoinbaseTX. This method creates a coin base transaction. There are two parameters: the destination and arbitrary message. Returns a transaction.

  1. Initialize a TXInput array with one transaction. This is a empty transaction. Why?

  TXInput[] txInput= {new TXInput("", -1, message)};

  2. Initialize a TXOutput array with one transaction. This this should have a value of 100.
  3. Create a new transaction
  4. Return transaction

BLOCK CLASS

The Block class:

Here are the fields:

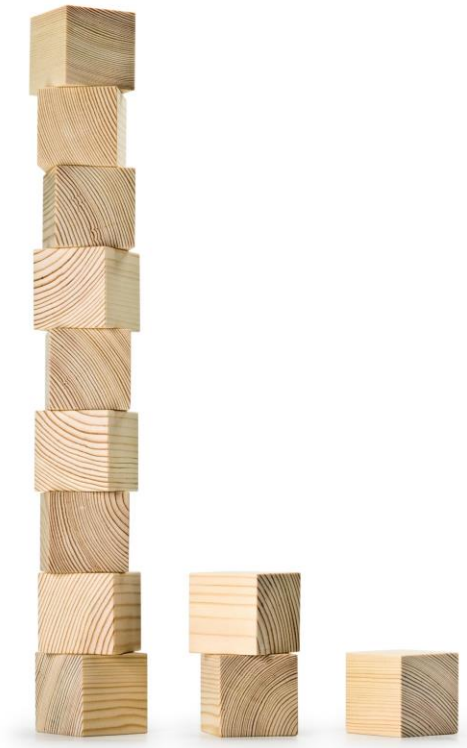
- Array of transactions
- Previous block hash
- Current block hash
- Block height
- Time stamp
- Static count

BLOCK CLASS - 2

Create a constructor. Initialize the various fields of the block class. Initialize the previousHash and transactions with constructor parameters. Calculate the remaining fields:

1. Initialize the time stamp to the current date
2. Calculate blockHeight by incrementing a count (static sequence data member)
3. Calculate the current hash for the block using Utilities.SetHash
4. Create an instance of the Proof of Work object. Pass in the current block as the parameter and a difficulty of 2.
5. Mine the current block using the Proof of Work object.

BLOCK CLASS - 3



- Create assessor methods for the current hash, previous hash, and blockId. Create a mutator for the sequence field.
- NewBlock. Accepts a string and array of transactions as parameters. Returns a Block. NewBlock is a static method. In the method create a new instance of a Block class. Return the new block.
- HashTransactions. Concatenate the transactions in the block and hash the results. Extra credit: create a Merkle tree from the transactions. Integrate the Merkle root into the class.
- DisplayTransactions: First, iterate and display the output transactions. Second, iterate and display input transactions.
- GetData. Return the concatenations of the previous hash+hash of the transactions+timestamp.

PROOFOFWORK CLASS

The Proof of Work class is used to mine blockchain blocks.

- Here are the fields:
  - difficulty
  - target block to mine
  - target for mining, such as “000”
  - Nonce
  - Maximum nonce value
- Create a constructor. Initialize target block and difficulty with constructor parameters. Assign mask as a series of zeros based on difficulty.
- **PrepareData.** PrepareData method that returns the block header (Block.GetData)+nonce. Nonce is incremented before returning from function.
- **Run.** Mine the block stored in the class.

BLOCKCHAIN CLASS

The Blockchain class manages an array of Blocks.

There are two constructors:

- The first constructor has one parameter – a list of blocks. Initialize Blockchain list with this parameter.
- The second constructor has one parameter, which is list of transactions. Create the genesis block with the transactions. Call the method CreateGenesisBlock.
- **AddBlock**. Adds a new block to the blockchain. The only parameter is an array of transactions. Returns nothing. Create a new block using Block.NewBlock with the transactions. Add the block to the list of blocks. Save the block to a Json file: BlockChain.SaveChain.
- **CreateGenesisBlock.** Create the first block of the blockchain. This method has a single parameter, which is an array of transactions.
    1. Create an array of transactions initialized with a single transaction, which is a coinbase transaction.
    2. Create a new block.
    3. Add the block to the blockchain.

BLOCKCHAIN CLASS - 2

- **ContainsInInputTx**. This method finds unused transactions. The only parameter is a TXOutput. Search for the output transaction in the list of input transactions.
  - Iterate the blocks of the blockchain.
  - Iterate the transactions of the block.
  - If the output transaction is assigned to a input transaction, return true. Otherwise, return false.
- **FindUTXO** . This method is finds unspent transactions for a particular person. The method has a single parameter, which is the person. Returns the UTXO for that person.
  1. Create an empty list of TXOutput transactions. This is the list of UTXO.
  2. Iterate the blocks of the blockchain.
  3. Iterate the transactions of each block.
  4. Iterate the array of TXOutput for each transaction.
  5. If ContainsInInputTx returns false, add to the UTXO list.

BLOCKCHAIN CLASS - 3

- NewOutputTransaction. This function sends money from one person to another. You are spending UTXO. There are three parameters: who is sending the money, who is receiving the money, and the amount being sent.
  - Create an empty list of TXInputs and TXOutputs.
  - Create an empty array of TXOutput, which will hold the UTXO. Initialize with FindUTXO.
  - Iterate the list of UTXO transactions. Keep a running total of the transaction value.
  - Create a new TXInput transaction for each UTXO transaction. Add to the TXInputs list. Break when the total is greater than the amount being spent.
  - Create a new TXOutput transaction for the total amount.
  - If there is *change*, create a new TXOutput transaction for that amount. Send to yourself.
  - Create a new Transaction from the list of TXInputs and TXOutputs. Set the Transaction id.
  - Return the Transaction.

BLOCKCHAIN CLASS - 4

- **GetBalance**. This method returns the balance for a person. There is one parameter, which identifies the person. Iterate and total the UTXO. Return the total.
- **SaveChain**. Here is the code to save the blockchain.

```
public void SaveChain( ) throws IOException {
try (Writer writer = new FileWriter("Output.json")) {
    Gson gson = new GsonBuilder().create();
    gson.toJson(blocks, writer)
}
```

BLOCKCHAIN CLASS - 5

ReadChain. Here is the code to read the blockchain.

```
public static Blockchain ReadChain( ) throws IOException {
Gson gson = new Gson();
ArrayList<Block> blocks=null;
try  {
        BufferedReader br = new BufferedReader(
        new FileReader("Output.json"));
        java.lang.reflect.Type blockType =
        new TypeToken<ArrayList<Block>>() {}.getType();
        blocks = gson.fromJson(br,blockType);
}       catch (IOException e) {
                e.printStackTrace();
}
Blockchain blockChain=new Blockchain(blocks);
Block.SetSequence(blocks.size());
TXOutput.count=blockChain.GetOutputCount();
return blockChain;
}
```

TEST BLOCKCHAIN

This is sample code to test your private blockchain.

```
public static void main(String [] args) throws Exception {
//CreateChain();
ReadBlockChain();
}
public static void CreateChain() throws Exception {

Transaction transaction1=Transaction.NewCoinbaseTX("Donis", "");
Blockchain blockChain=new Blockchain(new Transaction []
{transaction1});

System.out.println("Block 1\n\n");

Transaction transaction2=blockChain.NewUTXOTransaction(
                    "Donis", "Bob", 50);
blockChain.AddBlock(new Transaction[]{transaction2});
```

TEST BLOCKCHAIN - 2

This slide is a continuation of the previous slide.

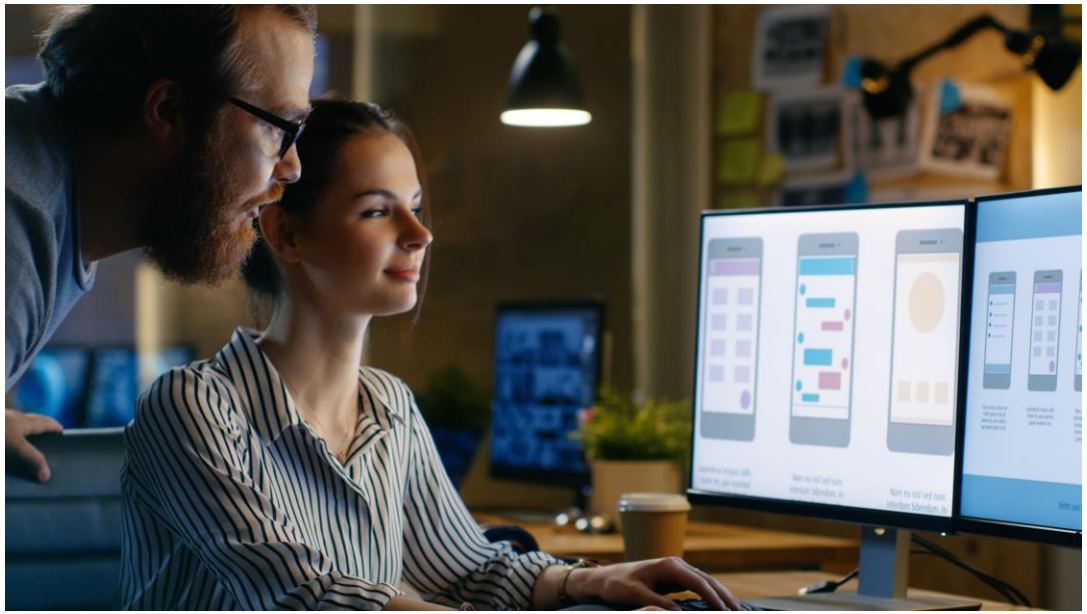
```
Transaction transaction3=blockChain.NewUTXOTransaction("Donis", "Fred", 25);
blockChain.AddBlock(new Transaction[]{transaction3});

Transaction transaction4=blockChain.NewUTXOTransaction("Bob", "Fred", 12);
blockChain.AddBlock(new Transaction[]{transaction4});

blockChain.SaveChain();
blockChain.DisplayChain();

int balance=blockChain.GetBalance("Donis");
System.out.println("The balance is Donis "+balance);

balance=blockChain.GetBalance("Bob");
System.out.println("The balance is Bob "+balance);

balance=blockChain.GetBalance("Fred");
System.out.println("The balance is Fred "+balance);
}
```

TEST BLOCKCHAIN - 3

This is a continuation of the previous slide.

```
Transaction transaction3=blockChain.NewUTXOTransaction("Donis", "Fred", 25);
blockChain.AddBlock(new Transaction[]{transaction3});


Transaction transaction4=blockChain.NewUTXOTransaction("Bob", "Fred", 12);
blockChain.AddBlock(new Transaction[]{transaction4});

blockChain.SaveChain();
blockChain.DisplayChain();

int balance=blockChain.GetBalance("Donis");
System.out.println("The balance is Donis "+balance);

balance=blockChain.GetBalance("Bob");
System.out.println("The balance is Bob "+balance);

balance=blockChain.GetBalance("Fred");
System.out.println("The balance is Fred "+balance);

}
```

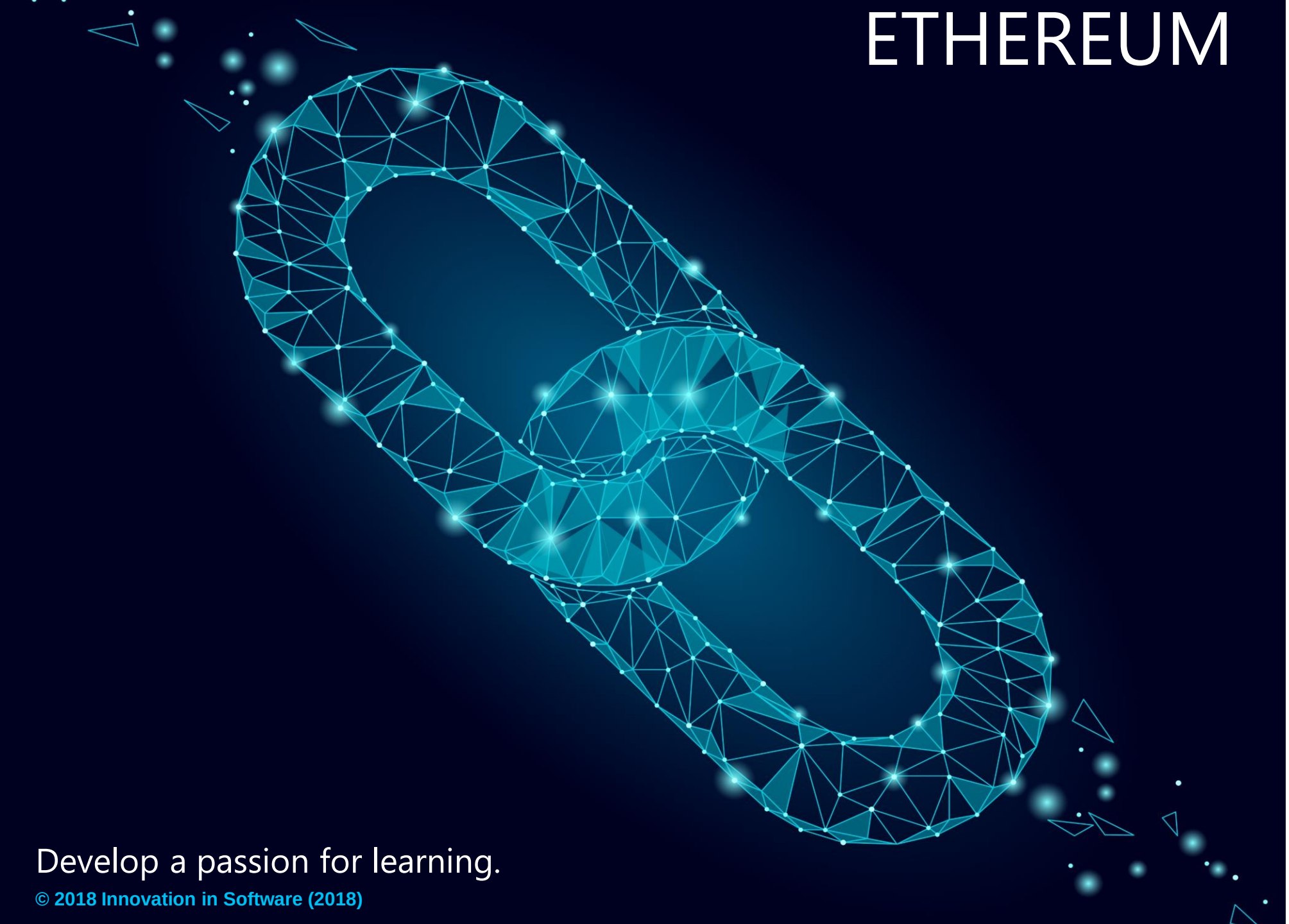
TEST BLOCKCHAIN - 4



This is a continuation of the previous slide.

```
Transaction transaction3=blockChain.NewUTXOTransaction("Donis", "Fred", 25);
blockChain.AddBlock(new Transaction[]{transaction3});

Transaction transaction4=blockChain.NewUTXOTransaction("Bob", "Fred", 12);
blockChain.AddBlock(new Transaction[]{transaction4});

blockChain.SaveChain();
blockChain.DisplayChain();

int balance=blockChain.GetBalance("Donis");
System.out.println("The balance is Donis "+balance);

balance=blockChain.GetBalance("Bob");
System.out.println("The balance is Bob "+balance);

balance=blockChain.GetBalance("Fred");
System.out.println("The balance is Fred "+balance);

}
```

ETHEREUM



Develop a passion for learning.

WHAT IS ETHEREUM

Ethereum is a platform for Decentralized Applications (dapps) while hosting a programmable blockchain. A smart contract is a Ethereum transaction that targets a particular programmable component available on the nodes of the Ethereum blockchain. Each full node of the Ethereum blockchain supports a Ethereum Virtual Machine (EVM), which is a sandbox for running smart contracts.

SMART CONTRACT

Smart contracts are code components that reside on the nodes of the blockchain:

- Executable functions
- State variables

When a transaction is sent to the component, the code is executed.

Smart contracts can be written in high-level languages such as Python. There are also contract specific compilers. Solidity is an example of a smart contract specific compiler. Once complied to bytecode, the code is uploaded to the blockchain and assigned a unique address.

MINING

Ethereum has mining also, similar to a standard blockchain, where miners compete to complete a complex algorithm for a reward in ethers. The winning miner can add the block to the blockchain with PoW. The block is then communicated to the other nodes on the Ethereum network.

Ethereum uses Ethash algorithm instead of Hashcash for PoW. The Ethash is beyond the scope of this class.

MINING

Ethereum currency is ether. The smallest domination is an Wei, where 1 ether is 10^{18} Wei.
Gas is the currency consumed for running an operation on the Ethereum network.

When an EOA sends a transaction, it includes a transaction value (gas). This is the fee the EOA is willing to pay to execute the operation to the miner.

DAPP

A Decentralized Application (Dapp) is an application that leverages smart contracts. Frequently, dapps are web applications that present a user interface to customers leveraging smart contracts for business logic. This is the underlying concept of Web 3.0.

Truffle, is a Dapp framework, and simplifies the creation of Dapp applications.

SMART CONTRACT

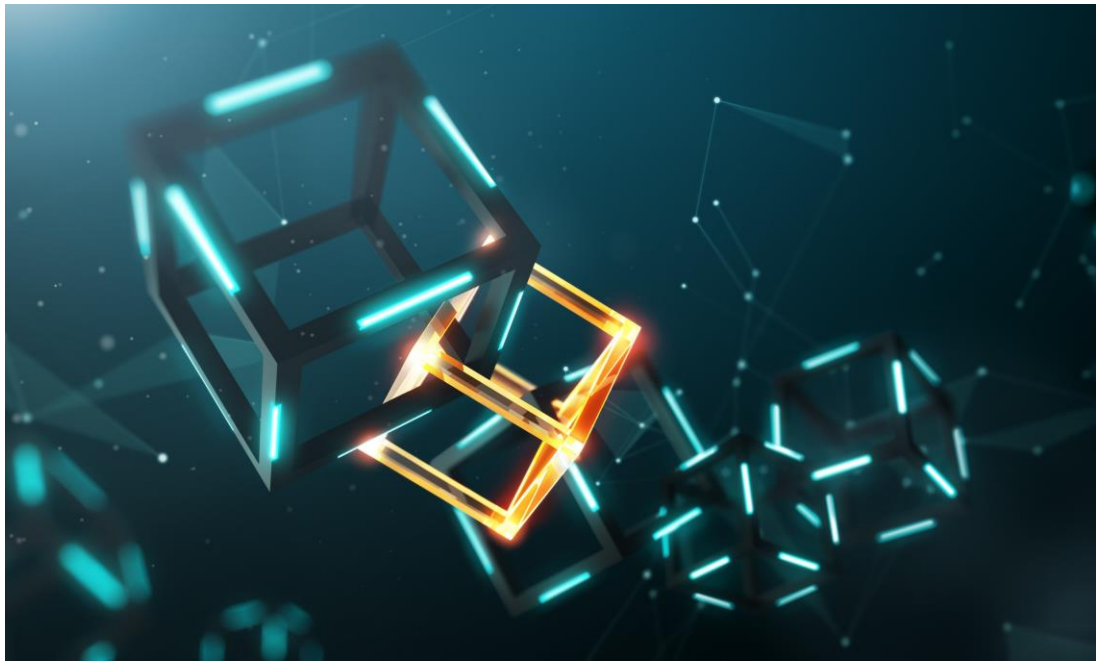
Ethereum clients are freely distributable clients for creating an Ethereum node. Ethereum clients can:

- Create accounts
- Define contracts
- Send transactions
- Mine the Ethereum network

Ethereum clients are developed in a variety of languages.

- Go: Geth
- Python: Pyethapp
- JavaScript: EthereumJS
- Java: EthereumJ

PRIVATE BLOCKCHAIN



You can create a private instance of a blockchain using Ethereum.

Same:

- Code base
- Client application
- Same SDK
- Protocol

Different:

- Genesis block
- Network identifier

MINERS RETIRED?

Miners cannot retire – at least not yet. Someone needs to mine to gather ether to pay gas to process transactions.

Note. Ethereum has a bug for private blockchains, where two miners are required to make sure that all transactions are processed.

ETHEREUM CLIENTS

Ethereum client	Language
Go-Ethereum	Go
Parity	Rust
Cpp-Ethereum	C++
Pyethapp	Python
Ethereumjs-lib	Javascript
Ethereum	Java
Ruby-Ethereum	Ruby
ethereumH	Haskell

There a variety of Ethereum clients that are also platform agnostic. The "con" is that there is not a universal Ethereum client installer. The "pro" is that the flexibility of clients promotes innovation.

Go-Ethereum is the most popular Ethereum client and used in this module.

GETH

GETH

Geth is a command-line application for running an Ethereum client. Geth allows you to:

- Mining
- Send ether
- Create and execute contracts
- Create accounts
- Manage a blockchain

Some advantages of Geth:

- Quicker mining
- Multi-threaded
- Command-line tool

Download Geth at location:

https://geth.ethereum.org/downloads

CREATE PRIVATE BLOCKCHAIN

```
{
  "config": {
    "chainId": 1907,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  },
  "difficulty": "10",
  "gasLimit": "2100000",
  "alloc": {}
}
```

The first step to creating a private blockchain with Ethereum / Geth is defining the genesis block. This can be done with a JSON configuration file.

GENESIS CONFIGURATION FILE

- chainId:1 refers to the main Etherum network. For a private blockchain, enter a unique number.
- alloc: prepopulate accounts with ether when blockchain created
- homesteadBlock: you are using the Homestead version of Ethereum, which is the second major release. "0" confirms the blockchain is using Homestead instead of Frontier.
- EIP is Ethereum Improvement Proposal and defines the standards for Ethereum.
  - EIP155Block: prevents replay attacks
  - EIP158Block: treats empty accounts as non-existent
- Difficulty sets the difficulty of mining where 0x400 means there is a 1/1024 opportunity to succeed at mining the block.
- Gaslimit: maximum number of computations allowed on any block.

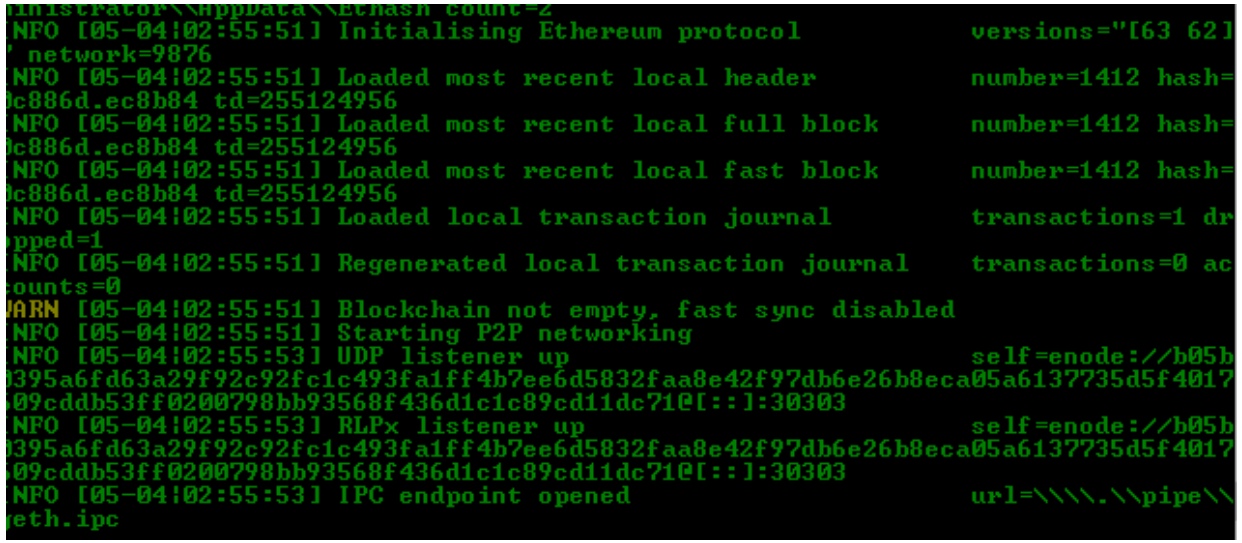
CREATE A BLOCKCHAIN

- Of course, make sure Geth client installed and in path environment variable
- Create a subdirectory for the blockchain and account credentials
- Create json file for genesis block.
- From the console window, here is the command:

geth init genesis.json --datadir test

- This command indicates where to create the blockchain and store related data.

OPEN AN ETHEREUM NODE



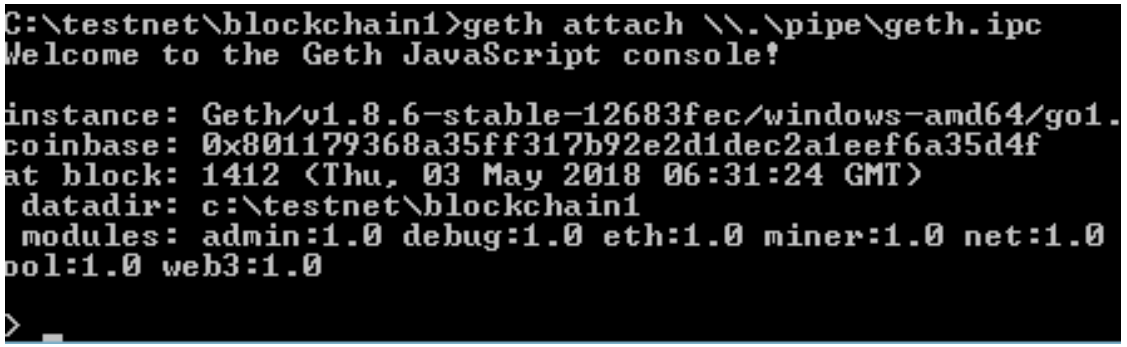
Here is the command to start an active Ethereum node.

```
geth --networkid 123 --datadir test
```

This command sets the network id to 123. You can communicate directly with the blockchain via a pipe (\\.\pipe\geth.ipc).

See opposite.

GETH INTERACTIVE SESSION



Here is the command to start an interactive Javascript command-line session with a Ethereum block chain.

geth attach \\.\pipe\geth.ipc

Notice the “>” prompt.

To close session, enter *exit*.

CREATE USER ACCOUNTS

You can create new user accounts in either the interactive session or command line.

Interactive session:

- eth.accounts
- eth.getBalance
- personal.newAccounts
- personal.unlockAccount

Command line:

- geth --datadir node1 account new

MINING

Start mining

- miner.start

Stop mining

- miner.stop

SOLIDITY

```
// math.sol

pragma solidity ^0.4.16;
contract Math {
 uint256 counter = 5; //state variable we assigned earlier
 function add() public {  //increases counter by 1
  counter++;
 }

 function subtract() public { //decreases counter by 1
  counter--;
 }
}
```

Program smart contracts using Solidity or a variety of languages, such as Python.

Here is the link for downloading and installing the Solidity compiler (solc.exe).

Make sure the binary is included in system path.

https://bit.ly/2I8HYtF

Either Node.js or Microsoft Visual Studio must also be installed to compile Solidity applications.

SOLIDITY - COMPILER

```
var simpleContract=eth.contract(


[{"constant":false,"inputs":[{"name":"_a","type":"uint256"},{"name":
"_b","type":"uint256"}],"name":"multiply","outputs":[{"name":"","ty
pe":"uint256"}],"payable":false,"stateMutability":"nonpayable","ty
pe":"function"},{"constant":false,"inputs":[{"name":"_a","type":"uin
t256"},{"name":"_b","type":"uint256"}],"name":"arithmetics","outp
uts":[{"name":"o_sum","type":"uint256"},{"name":"o_product","typ
e":"uint256"}],"payable":false,"stateMutability":"nonpayable","typ
e":"function"}]


)
```

For example, compile smart contract with:

Solc.exe math.sol

This will generate an abi and bin file. Running the contract in the interactive windows requires modifying these files into scripts. Here is the modification for the abi file first.

Modifications are highlight in red.

SOLIDITY - COMPILER

```
personal.unlockAccount(eth.accounts[0])
var simple = simpleContract.new(
{ from: eth.accounts[0],
data:

"0x6080604052348015610010576000 80fd5b5061013b806100206000396000f3
006080604052600436106100 4c576000357c0100000000000000000000000000
0000000000000000000000000000000000900463ffffffff168063165c4a1614610051
5780638c12d8f01461009c575b600080fd5b34801561005d57600080fd5b50610
0866004803603810190808035906020019092919080359060200190929190505
0506100ee565b604051808281526020019150506040518091039 0f35b34801561
00a857600080fd5b506100d160048036038101908080359060200190929190 80
359060200190929190505050506100fb565b604051808381526020018281526020
019250505060405180910390f35b600081830290509291505056 5b600080828
401915082840290509250929050505600a165627a7a72305820bfbe58c1896a0d5
352164ca7143aa80e5b1c6f7ee804c78731c7fa94d7b781330029",
gas: 500000
}
)
```

Here are the modifications for the bin file.

RUN CONTRACT

- Load both the abi and bin scripts to execute an Ethereum smart contract.

> loadScript("contract/math.abi")

> loadScript("contract/math.bin")

- You cannot execute a smart contract until the node has been mined (i.e., miner.start)

- Here is a method invocation.

    Math.subtract()

Lab 8- SMART CONTRACT

GETH AND SOLIDITY

It this lab, you will create and execute a smart contract.

For the lab, make sure these components are installed.

- Geth

https://geth.ethereum.org/download

- Solidity

https://bit.ly/2I8HYtF

- Visual Studio or (but not both)

https://www.visualstudio.com/downloads/

- Node.js

https://nodejs.org/en/download/

CREATE THE BLOCKCHAIN

```
{
    "config": {
        "chainId": 987,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty": "0x400",
    "gasLimit": "0x8000000",
    "alloc": {}
}
```

Create a new directory called Blockchain1.

In that directory, create a json file with any text editor that defines the genesis block of our new blockchain. Name the file mygenesis.json.

geth init *dirpath**filename.json* --datadir *dirpath*

Create an active blockchain.

geth --networkid 123 --datadir \dirpath

COMPILE SMART CONTRACT

```
pragma solidity ^0.4.13;

contract Simple {
  function arithmetics(uint _a, uint _b) returns (uint o_sum, uint o_product) {
    o_sum = _a + _b;
    o_product = _a * _b;
  }

  function multiply(uint _a, uint _b) returns (uint) {
    return _a * _b;
  }
}
```

The adjacent smart contract performs mathematical calculations.

With a text editor, create the .sol file for the Solidity source code.

Compile the source code using solc.exe.

Confirm that the smart contract was created successfully.

CONVERT FILES INTO SCRIPTS

As shown earlier in this module, convert abi and bin files into scripts. For example, this is the abi file.

```
var simpleContract=eth.contract(


[{"constant":false,"inputs":[{"name":"_a","type":"uint256"},{"name":"_b","type":
"uint256"}],"name":"multiply","outputs":[{"name":"","type":"uint256"}],"payabl
e":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"in
puts":[{"name":"_a","type":"uint256"},{"name":"_b","type":"uint256"}],"name":"
arithmetics","outputs":[{"name":"o_sum","type":"uint256"},{"name":"o_produc
t","type":"uint256"}],"payable":false,"stateMutability":"nonpayable","type":"fu
nction"}]


)
```

CREATE USER AND MINE FOR COINS

Create a user account. The account will not own any coins initially.
Start mining to provide the user some coins.

Open the geth interactive JavaScript session.

- geth attach \\.\pipe\geth.ipc

Create a user account.

- personal.newAccount

Check balance of new account

- eth.getBalance("userid")

Start mining – after 15 seconds check account balance

Stop mining

EXECUTE SMART CONTRACTS

Load ABI and BIN scripts with the loadScript command. If the return of loadScript is false, there is a problem. If loadScript returns true, proceed to the next step.

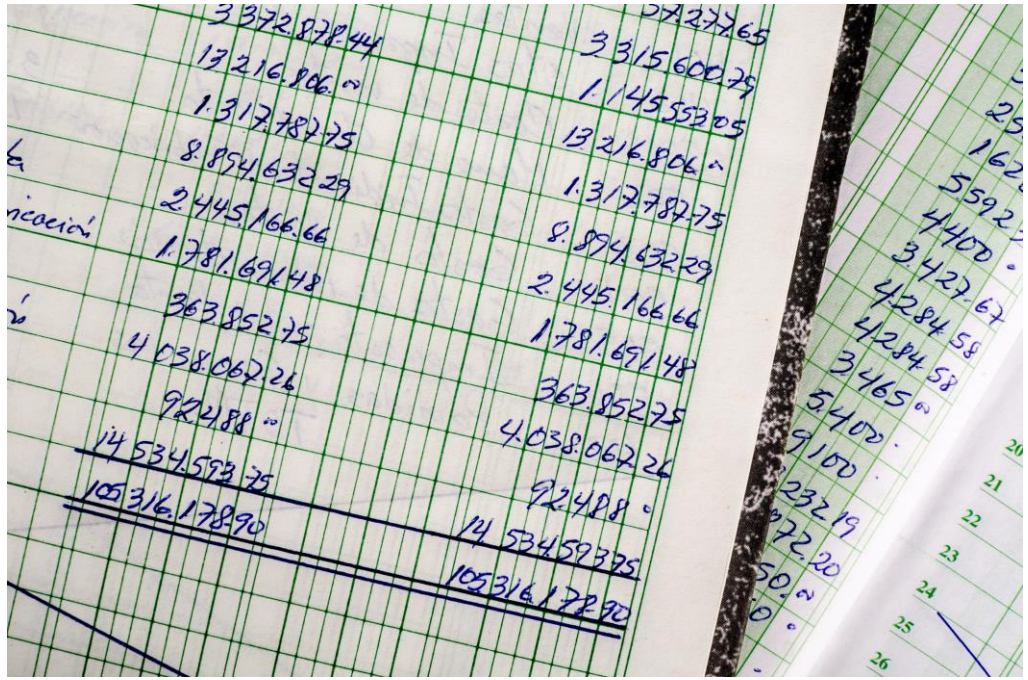
Mine again to deploy the smart contract to the blockchain.

Test simple.Multiple().

HyperLedger Fabric

Develop a passion for learning.

MINING



HyperLedger is several things: a platform, a permissioned blockchain, a blockchain ecosystem.

HyperLedger has a high degree of resilience, security, flexibility, and scalability. Part of the benefit of HyperLedger is the modularization of its architecture.

HISTORY

The HyperLedger project was initiated by the Linux Foundation and started in December 2016. The goal was to create a hardened blockchain solution for enterprise organization with additional support for cross industry global transactions with the focus on:

- Technological
- Financial
- Supply chain companies

MEMBERS

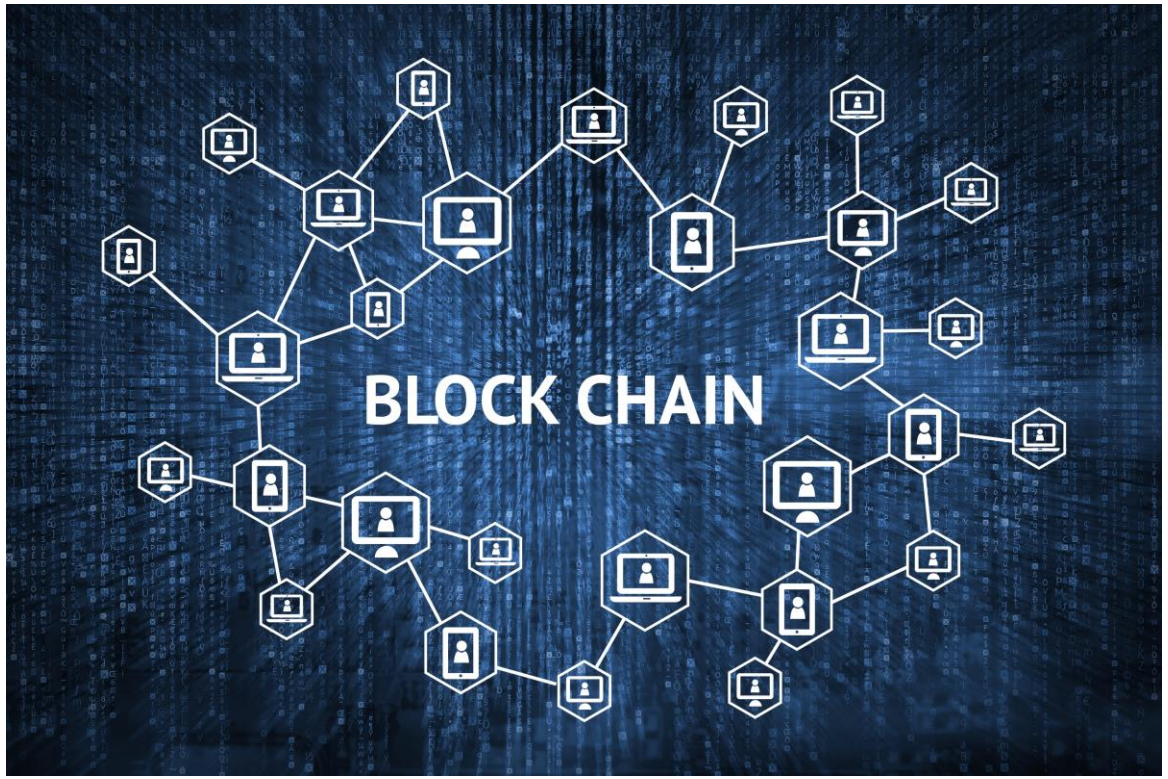
Some of the initial members of the HyperLedger project include:

- IBM
- Cisco
- VMWare
- Oracle
- J. P. Morgan
- Wells Fargo
- Deutsche Bank
- And more

HYPERLEDGER BURROW

It is a blockchain Ethereum client that includes the Ethereum Virtual Machine.

HYPERLEDGER FABRIC



This is probably the most known HyperLedger solution and is a permissioned blockchain with a pluggable architecture. HyperLedger Fabric supports smart contracts – called chaincode in this environment.

HyperLedger Fabric does not include a cryptocurrency.

IBM contributed HyperLedger Fabric to the HyperLedger to the project.

HYPERLEDGER INDY

HyperLedger Indy is a security tool and various components in support of digital identities.