1. Installeer de Geth client
2. Kopieer de genesis.json file naar de map waar je Geth hebt geïnstalleerd (meestal C:/Program Files/Geth
3. Vanaf je command-line initialiseer je de genesis.json
    a. geth init genesis.json
4. Start Geth
    a. geth --networkid 42 console
5. Voeg nieuw account toe
    a. personal.newAccount()
    b. onthoud je wachtwoord!
6. Connect met een van de peers
    a. admin.addPeer()
    b. als parameter geef je de enode op die hieronder staat. Pas het ip-adres (dikgedrukt aan het eind) aan naar wat er doorgegeven is.
    c. "enode://18ba098fa40117d0980d5cfa52611d2de8798c89253b2104d715d312c42aa2fc5393d45f6f0ab9b5733bfd06aae45be8a92617634e1abe1d6da8abd47304665c@**192.1.1.5**:30303"
    d. Als het goed is zou hij nu blocks moeten gaan synchroniseren. Dit kan enkele tellen duren
7. Start de Geth UI
    a. Ethereum Wallet.exe
    b. In principe moet nu je account tevoorschijn komen in de GUI

Smart contracts

Een smart contract is eigenlijk gewoon en stukje software dat op de blockchain draait. In dit voorbeeld gaan we onze eigen token maken. Een token kun je zien als een cryptocurrency. Veel van de bekende crypto's zijn namelijk gewoon 'tokens'. Om te beginnen maar je een nieuw bestand aan met Notepad++. Deze kun je uiteindelijk gewoon opslaan als txt. Het contracts moet namelijk toch nog gecompileerd worden. Je kunt ook het contract schrijven in de editor die in de Geth wallet zit. Dit is misschien makkelijker qua code controle. Voor deze editor ga je in de wallet naar 'Contracts' en dan naar 'Deploy new contract'.

1. We beginnen met de basis van het contract.

```
pragma solidity ^0.4.18;

contract DuoCoin {

}
```

2. Hierna gaan we een paar constanten definieren direct aan het begin van het contract (na de eerste {. We beginnen met de totale voorraad aan coins, de naam van de coin en het symbool van de coin. Deze laatste is een 3 of 4 lettercode van de coin (bijvoorbeeld BTC voor Bitcoin). Ook voegen de een constante toe voor het aantal decimalen dat deze coin kan hebben. Je mag deze gegevens naar eigen wens aanpassen!

```
uint256 supply = 1000000000000;

string public constant name = "DUO Coin";

string public constant symbol = "DUO";

uint8 public constant decimals = 8;
```

3. Vervolgens gaan we een *modifier* schrijven. Dit stukje code wordt uigevoerd voor of na een methode die deze *modifier* mee krijgt. In dit geval zorgen we er met deze modifier voor dat de methode die deze *modifier* mee krijgt alleen kan worden aangeroepen door de eigenaar van het contract. De modifier ziet er als volgt uit. Het _ geeft aan waar de methode komt. In dit voorbeeld zal de methode die de *modifier* mee krijgt pas komen *na* de *modifier.*

```
modifier onlyOwner() {
        if (msg.sender != owner) {
                revert();
        }
        _;
}
```

Daarnaast zullen we nog een variabele moeten toevoegen aan het contract waar we deze 'owner' gaan opslaan. Deze kun je ook bovenaan het contract zetten, bijvoorbeeld onder de constanten die je eerder hebt gedefinieerd.

```
address public owner;
```

4. De owner die je net als variabele hebt gedefinieerd, moet natuurlijk ook ergens worden gevuld. Dit doe je meteen als het contract wordt geïnstantieerd. Hiervoor gebruiken we de constructor. Onder de modifier gaan we de constructor definieren. Dit doen we als volgt.

```
function DuoCoin() public {
        owner = msg.sender;
        balances[owner] = supply;
}
```

   Hiermee wordt de eigenaar van het contract opgeslagen, én wordt de gehele voorraad van coins toegewezen aan de eigenaar.

5. We gaan nu twee methodes toevoegen die gebruikt worden om de balans van een adres op te vragen en om de totale voorraad op te vragen van deze coin. Om de balans van een adres op te vragen schrijven we een methode die een parameter mee krijgt. Ook definieren we wat de methode gaat terug geven, in dit geval een uint256.

```
function balanceOf(address _owner) public constant returns (uint256 balance) {
        return balances[_owner];
}
function totalSupply() public constant returns (uint256 total) {
        return supply;
}
```

6. Om coins te kunnen uitwisselen moet je een transfer methode hebben. Uiteraard heb je nu twee parameters nodig. Het adres van de ontvanger, en de hoeveelheid die je wil versturen.

```
function transfer(address _to, uint256 _amount) public returns (bool success) {
        if (balances[msg.sender] >= _amount
                && _amount > 0
                && balances[_to] + _amount > balances[_to]) {
                balances[msg.sender] -= _amount;
                balances[_to] += _amount;
                emit Transfer(msg.sender, _to, _amount);
                return true;
        } else {
                return false;
        }
}
```

Echter, we hebben nog nergens iets gedefinieerd om de addressen in op te slaan. Hiervoor maken we nog een variable aan. Een zogenaamde mapping.

```
mapping(address => uint256) balances;
```

De methode begint met een aantal checks. Namelijk, je wil weten of de verzendende partij genoeg coins heeft. Daarna controleer je of de te verzenden hoeveelheid wel groter is dan 0. Daarna komt een interessante. Je gaat dan kijken of de ontvanger na het optellen van het te verzenden bedrag meer heeft dan daarvoor. Dit heeft een hele goeie reden. Omdat je werkt met een unsignerd integer kun je als je tot de max van een uint256, oftewel 2^256-1. Zodra je over dit getal heen gaat kom je weer bij 0. Dit wil je natuurlijk niet als je geld naar iemand overmaakt, want op die manier kun je z'n geld laten verdwijnen.

Daarna ga je de balansen van de betrokken partijen bijwerken. Aftrekken van de verzender, bijtellen bij de ontvanger, en daarna doe je een emit. Dit is een 'Event' en deze wordt gebruikt om data te loggen op de Blockchain. Ook kunnen externe applicaties die interacteren met de blockchain reageren op dit soort events.

Deze event moet nog gespecificeerd worden bij de variabelen. Bij de variabelen voeg je de volgende regel toe.

```
event Transfer(address indexed _from, address indexed _to, uint256 _value);
```

7. Je kunt ook iemand een bedrag toekennen die ze van jou mogen uitgeven. De coins blijven dus op je eigen account staan, maar iemand anders mag ze voor je uitgeven. Hiervoor moeten we een variabele aanmaken waarin dit wordt bijgehouden.

   ```
   mapping(address => mapping(address => uint256)) allowed;
   ```

   Deze mapping bevat een mapping. Als je goed kijkt is het wel te volgen. Hierin kun je opslaan hoeveel iemand van jou mag uitgeven. Hiervoor hebben we natuurlijk een methode nodig om dit vast te leggen.

   ```
   function approve(address _spender, uint256 _amount) public returns (bool
   success) {
   
       allowed[msg.sender][_spender] = _amount;
   
       emit Approval(msg.sender, _spender, _amount);
   
       return true;
   
   }
   ```

   En we willen ook graag op kunnen vragen hoeveel deze persoon nog mag uitgeven.

   ```
   function allowance(address _owner, address _spender) public constant returns
   (uint256 remaining) {
   
       return allowed[_owner][_spender];
   
   }
   ```

   Als je goed opgelet hebt is het je vast opgevallen dat in de *approve* functie ook een event wordt gegooid. Hiervoor moeten we weer een variabele toevoegen voor dit event. Voeg de volgende regel toe.

```
event Approval(address indexed _owner, address indexed _spender, uint256 _value);
```

8. Nu komt de laatste methode. We *transferFrom* methode. Deze wordt gebruikt om geld uit te geven dat je hebt toegewezen gekregen door middel van de voorgaande methodes. Deze ziet er aardig uitgebreid uit. Kijk goed hoe de methode in elkaar zit en wat waar voor dient. Het lijkt ingewikkeld, maar door er even goed naar te kijken kom je er vast uit. Zo niet, vraag dan even om uitleg.

```
function transferFrom(address _from, address _to, uint256 _amount) public returns (bool success) {
        if (balances[_from] >= _amount
                && allowed[_from][msg.sender] >= _amount
                && _amount > 0
                && balances[_to] + _amount > balances[_to]) {
                balances[_from] -= _amount;
                allowed[_from][msg.sender] -= _amount;
                balances[_to] += _amount;
                emit Transfer(_from, _to, _amount);
                return true;
        } else {
                return false;
        }
}
```

9. Om het contract op de blockchain te zetten selecteer je rechts uit het dropdown menu je contract. Vervolgens druk je op 'Deploy'. Mocht je nog geen Ether hebben kun je even gaan minen. Dit doe je door naar de console te switchen, en daar typ je: *miner.start(1)*

   Zorg dat je de 1 als parameter mee geeft, zodat je maar met 1 core gaat minen. Anders krijgt je laptop het erg snel erg warm.

   Zodra het contract is opgepikt door een miner moet deze verschijnen in je wallet met de coins die je jezelf hebt toegewezen. Nu kun je deze coins naar anderen gaan sturen. Daarvoor heb je alleen hun wallet address nodig.

   Anderen kunnen je contract ook toevoegen aan hun wallet. Dit zou automatisch moeten gaan als je iemand coins stuurt. Gebeurd dit niet kun je deze toevoegen in het tabje 'contracts' en dan 'watch token'.

10. Nu kun je zelf nog van alles gaan toevoegen. Enkele ideeën:
    a. Een methode waar je een coin naar stuurt, elke 10^e die dit doet krijgt de pot.
    b. Een methode die 'at random' je coin doorstuurt naar een van de adressen in de mapping

    c. Een methode die pas uitbetaald aan een adres wanneer minimaal 5 mensen coins doneren
    d. Etc
11. Toon je creativiteit. Als je een leuke methode hebt geschreven laat dit dan weten!