4. *Running smart contract methods with commands in console*

A. Prerequisites:

1. According to the previous tutorial, deploy your adder.sol smart contract and create your first smart contract instance.
2. Reach the step just before Tutorial 3: Section F-3 "*Calling the functions/methods with the instance of the smart contract*" in the previous tutorial.

B. Call the methods in the deployed smart contract by commands in console

1. Find and copy the ABI to your text editor:
   - **a)** In the Solidity Compiler's Side Panel
   - **b)** At the bottom, clicking ABI to copy the ABI and paste it to a text editor.
2. Usage of Application Binary Interface (ABI)
   - **a)** In the console of the peer that your IDE is connecting via Web3 Provider, let us define a JavaScript variable, say `abi`, as follows:

```
var abi = [ { "inputs": [], "stateMutability":
"nonpayable", "type": "constructor" }, { "inputs":
[ { "internalType": "int256", "name": "add", "type":
"int256" } ], "name": "addToTotal", "outputs":
[ { "internalType": "int256", "name": "", "type":
"int256" } ], "stateMutability": "nonpayable", "type":
"function" }, { "inputs": [ { "internalType": "int256",
"name": "add", "type": "int256" } ], "name":
"addToTotalConst", "outputs": [ { "internalType": "int256",
"name": "", "type": "int256" } ], "stateMutability":
"view", "type": "function" }, { "inputs": [], "name":
"getTotal", "outputs": [ { "internalType": "int256",
"name": "", "type": "int256" } ], "stateMutability":
"view", "type": "function" } ]
```

   It is worth noting that the value of `abi` is wrapped by a pair of square brackets.

   Please try to copy and use your own ABI, even with the same code, if you have any different configurations from mine in your IDE, the ABI will not work with your deployed smart contract.
3. Usage of the address of the deployed smart contract instance.
   - **a)** Copy the new deployed smart contract address.
   - **b)** In the same console where the `abi` variable is defined, we can define another JavaScript variable, say `address`, the value of the define variable should be initialized as follows:
     `var address= "new deployed smart contract address"`
     Note that, we must have a pair of quotes wrapping outside the hexadecimal string with prefix 0x.
4. Call the method in smart contract by commands.

a) Run the following command in the console where the variables `api` and `address` are defined.

```
var myAdder=eth.contract(abi).at(address)
```

It defines another variable like a pointer (or reference) pointing (or referring) to the created smart contract instance.

b) Run the following command to see what the variable `myadder` really is:

```
myAdder
```

c) Now, run the command

```
myAdder.getTotal.call()
```

to check the current value of the total variable in the smart contract instance. This command will not ask for your permission, as it will not consume any computational resources in the network.

d) Run the command

```
myAdder.addToTotal.sendTransaction(100)
```

if you want to increase the value of the variable total by `100`. The console may return you the following error:

```
Error: invalid address
      at inputAddressFormatter (web3.js:3940:11(45))
      at inputTransactionFormatter
(web3.js:3756:41(14))
      at web3.js:5043:37(8)
      at map (native)
      at web3.js:5042:12(12)
      at web3.js:5068:34(18)
      at send (web3.js:5093:39(13))
      at web3.js:4155:41(50)
      at bound  (native)
      at <eval>:1:1(4)
```

This is because you have not specified which account that you want to use to pay for this transaction. The reason why this transaction needs to be paid is because it changes the variables in the smart contract and computational resources in the network may be consumed.

To deal with this problem, we can ether use the following command:

```
myAdder.addToTotal.sendTransaction(100, {from:
eth.accounts[0]})
```

or set the default account for transaction fee payment purposes by

```
eth.defaultAccount=eth.accounts[0]
```

and rerun `myAdder.addToTotal.sendTransaction(100)` again.

We assume the second solution is adopted.

e) Run the command

```
myAdder.addToTotal.sendTransaction(100)
```

again, and this time you may receive the following error message

Error: authentication needed: password or unlock
at web3.js:6347:37(47)
at web3.js:5081:62(37)
at web3.js:4137:41(57)
at <eval>:1:48(10)

You should know how to solve this issue now.
Use command `personal.unlockAccount(eth.accounts[0])` to unlock the account being used for payment. Now, running this again will get through and a transaction ID within a pair of quotes will be returned to you after the above command has been executed.

f) If there is no miner running, executing `eth.pendingTransactions` should list you there is a transaction to be mined. Start a miner until the transaction is mined. Then, run command `myAdder.getTotal.call()` again, the console should show you the updated value of the variable total in the smart contract.

C. Verification of the private network

1. Assume the second peer is also running and updated.
2. Repeat the commands that you used in 2-b, 3-a, 3-b, 4-a, 4-b, and 4-c in the console of the second peer, the same result should be shown as in the first peer.

D. Extension

1. Make sure the second peer is running and being connected with the first peer and also make sure the **explore** software introduced in the second tutorial is running. Then, use a web browser to browse http://localhost:8000 will show you the recently mined blocks. All the transactions should be found in the blocks and translation ID should be the same as you have in the console.

E. If you have any problems when you try to reproduce the experiment, please contact me for help. Email: zwang@ece.ubc.ca