Sample Solidity & Java Contracts

We have prepared some examples of various Solidity contracts to help you learn Jthereum faster.

Greeter Contract

This is a very basic contract that performs only 3 steps:

- It sets greeting on contract creation and allows to change it by the owner (creator of the contract) later.
- It returns greeting to everyone who calls `sayHello` method
- It returns `Hello Daddy` to the creator

We will look closely into this example to see how Jthereum works and what to expect when executing a contract like this one.

If you wanted to write this contract in Solidity it would look as following:

```
pragma solidity ^0.4.22;

contract Greeter {

    string greeting;

    address owner;

    modifier onlyOwner {
        require(isOwner(), "Only owner can do that!");
        _;
    }

    constructor(string _greeting) public {
        greeting = _greeting;

        owner = msg.sender;
    }

    function sayHello() public view returns(string) {
        if (isOwner()) {
            return "Hey daddy!";
        } else {
            return greeting;
        }
    }

    function setGreeting(string _newGreeting) public onlyOwner {
        greeting = _newGreeting;
    }

    function isOwner() view private returns(bool) {
        return msg.sender == owner;
    }
}
```

And this is how it looks like in Java

```
package com.u7.jthereum.exampleContracts;

import com.u7.jthereum.annotations.*;
import com.u7.jthereum.types.*;

import static com.u7.jthereum.Jthereum.*;
import static com.u7.jthereum.SolidityStaticImports.*;

public class Greeter {
 String greeting = "Hi";
 Address owner;

 @View
 private void onlyOwner() {
  require(isOwner(), "Only owner can do that!");
 }

 public Greeter() {
  owner = msg.sender;
 }

 @View
 public String sayHello() {
                if (isOwner()) {
                    return "Hey daddy!";
                } else {
                    return greeting;
                }
 }

 public void setGreeting(String _newGreeting) {
  onlyOwner();

  greeting = _newGreeting;
 }

 @View
 private boolean isOwner() {
  return msg.sender.equals(owner);
 }

        // Compile, deploy, and call.
 public static void main(final String[] args) {
  compileAndDeploy();

  final Greeter a = createProxy(Greeter.class);

  p("message: " + a.sayHello());

  p("Setting new greeting");
  a.setGreeting("Hello from Java!");

  p("message: " + a.sayHello());
 }
}
```

What happens when we build and run the code? Well, Jthereum converts your Java code into the Solidity code. Take a look at it:

```
Compiling com.u7.jthereum.exampleContracts.Greeter

Solidity Source:

// Generated by Jthereum BETA version!
pragma solidity ^0.5.9;
contract Greeter
{
 string private greeting = "Hi";
 address private owner;
 function onlyOwner() private view
 {
  require(isOwner(), "Only owner can do that!");
 }
 constructor() public
 {
  owner = msg.sender;
 }
 function sayHello() public view returns (string memory)
 {
  if (isOwner())
  {
   return "Hey daddy!";
  } else
  {
   return greeting;
  }
 }
 function setGreeting(string memory _newGreeting) public
 {
  onlyOwner();
  greeting = _newGreeting;
 }
 function isOwner() private view returns (bool)
 {
  return (msg.sender == owner);
 }

}
```

Then your Solidity code is being compiled and deployed on the Blockchain. In this case we specified the blockchain to be “test”. Take a look at the output:

```
Argument values for call to Jthereum.compileAndDeploy():

  Argument               Source
 ==========             ========
 Class contractClass = com.u7.jthereum.exampleContracts.Greeter.class;   // call stac
 String blockchainName = "test";        // global property: DEFAULT_BLOCKCHAIN
 String web3jURL = "http://test.jthereum.com:8545";     // global property: TEST_Web3
 BigInteger privateKey = pk for address: 0x572ba8f2a93abab1555981ce8f6bb3320a5b230b;
 BigInteger gasPrice = 1;        // default gas price for 'test' blockchain
 BigInteger gasLimit = 7,999,999;       // default gas limit for 'test' blockchain


Deploying contract to the blockchain

Tx hash: 0xe3a539dbe871ea41e88313b12c81a7185f5307a12f571e601f68bfd93fb22b50
Waiting for transaction to be mined.
Your contract was deployed in block #2,823,106
The new contract address is at: 0x206c46ac16a5447e69539d1a1841663bb8dcf693
```

Now argument values for Jthereum Proxy:

```
Argument values for call to Jthereum.createProxy():

  Argument               Source
 ==========             ========
 Class contractClass = com.u7.jthereum.exampleContracts.Greeter.class;   // directly
 String blockchainName = "test";        // most recent deployment for class Greeter
 String web3jURL = "http://test.jthereum.com:8545";     // global property: TEST_Web3
 String address = "0x206c46ac16a5447e69539d1a1841663bb8dcf693";    // most recent dep
 BigInteger privateKey = pk for address: 0x572ba8f2a93abab1555981ce8f6bb3320a5b230b;
```

Finally, here is our output result:

```
Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = com.u7.jthereum.exampleContracts.Greeter.class;   // inherited
 String blockchainName = "test";        // inherited from Proxy Object configuration
 String web3jURL = "http://test.jthereum.com:8545";     // inherited from Proxy Objec
 String address = "0x206c46ac16a5447e69539d1a1841663bb8dcf693";    // inherited from
 BigInteger privateKey = pk for address: 0x572ba8f2a93abab1555981ce8f6bb3320a5b230b;


Calling sayHello:
message: Hey daddy!
Setting new greeting

Argument values for call to ContractProxyContextInfo.callContractTransaction():

  Argument               Source
 ==========             ========
 Class contractClass = com.u7.jthereum.exampleContracts.Greeter.class;   // inherited
 String blockchainName = "test";        // inherited from Proxy Object configuration
 String web3jURL = "http://test.jthereum.com:8545";     // inherited from Proxy Objec
 String address = "0x206c46ac16a5447e69539d1a1841663bb8dcf693";    // inherited from
 BigInteger privateKey = pk for address: 0x572ba8f2a93abab1555981ce8f6bb3320a5b230b;
 BigInteger gasPrice = 1;        // default gas price for 'test' blockchain
 BigInteger gasLimit = 7,999,999;       // default gas limit for 'test' blockchain


Calling setGreeting:

Tx hash: 0x85087866572b0ca1aa35245b87a95f4c4789e2b35552fbf63db323a717061bf2
Waiting for transaction to be mined.
Gas Used: 22,808
Total WEI used: 22,808
Total ETH used: 0.000000
(Gas Price: 1 wei)
Transaction succeeded with status: 0x1


Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = com.u7.jthereum.exampleContracts.Greeter.class;   // inherited
 String blockchainName = "test";        // inherited from Proxy Object configuration
 String web3jURL = "http://test.jthereum.com:8545";     // inherited from Proxy Objec
 String address = "0x206c46ac16a5447e69539d1a1841663bb8dcf693";    // inherited from
 BigInteger privateKey = pk for address: 0x572ba8f2a93abab1555981ce8f6bb3320a5b230b;


Calling sayHello:
message: Hello from Java!

Process finished with exit code 0
```

Basic Token Contract

On the Ethereum blockchain everyone can create a token. So why don't we create one? This contract creates a token with various different options:

- It sets the intial supply of tokens on creation

- The contract creator gets initial tokens
- Tokens can be transferred to any account
- It sets a protection from overflow
- It sets an option for everyone to check balances

If you wanted to write this contract in Solidity it would look as following:

```
pragma solidity ^0.4.22;

contract BasicToken {
    uint public initialSupply;

    mapping(address=>uint) balances;

    constructor(uint _initialSupply) public {
        initialSupply = _initialSupply;
        balances[msg.sender] = _initialSupply;
    }

    function transfer(address _recipient, uint _amount) public {
        require(balances[msg.sender] >= _amount, "Not enough funds");
        require(_recipient != msg.sender, "No need to send tokens to yourself");
        require(balances[_recipient] + _amount > balances[_recipient]); //overflow ch
        balances[msg.sender] -= _amount;
        balances[_recipient] += _amount;
    }

    function balanceOf(address _owner) public view returns (uint) {
        return balances[_owner];
    }
}
```

And this is how it looks like in Java

```
package com.u7.jthereum.exampleContracts;

import com.u7.jthereum.annotations.*;
import com.u7.jthereum.internal.*;
import com.u7.jthereum.types.*;
import com.u7.jthereum.wellKnownInterfaces.*;

import static com.u7.util.gg.cf;
import static com.u7.jthereum.Jthereum.*;
import static com.u7.jthereum.SolidityStaticImports.*;

public class BasicToken implements ERC20<BasicToken>
{
 public final String name = "Basic Token";
 public final String symbol = "BAS";

 public final Uint8 decimals = Uint8.valueOf(8);

 private final Uint256 initialSupply = Uint256.valueOf(100_000_000_000_000_000L);

 // All of the balances
 final Mapping<Address, Uint256> balancesByAddress = new Mapping<>();

 public BasicToken()
 {
  // set up initial supply
  balancesByAddress.put(msg.sender, initialSupply);
```

```
  }

  @Override
  public boolean transfer(final Address _recipient, final Uint256 _amount)
  {
   require(balancesByAddress.get(msg.sender).greaterThanOrEqual(_amount), "Not enough
   require(!_recipient.equals(msg.sender), "No need to send tokens to yourself");

   // Compute the new balances
   final Uint256 newSenderBalance = balancesByAddress.get(msg.sender).subtract(_amount
   final Uint256 newRecipientBalance = balancesByAddress.get(_recipient).add(_amount);

   require(newRecipientBalance.greaterThanOrEqual(balancesByAddress.get(_recipient)));

   balancesByAddress.put(msg.sender, newSenderBalance);
   balancesByAddress.put(_recipient, newRecipientBalance);

   // Log the transaction
   emitEvent(new Transfer(msg.sender, _recipient, _amount));

   return true;
  }

  @Override
  @View
  public Uint256 balanceOf(final Address who)
  {
   return balancesByAddress.get(who);
  }

  public static void main(final String[] args)
  {
   compileAndDeploy();

   final BasicToken t = (BasicToken)createProxy();

   // burn some tokens
   t.transfer(Address.ZERO, new Uint256(1000));

   String myAddress = getMyAddress();
   p("My Balance: " + cf(t.balanceOf(new Address(myAddress))));
  }
 }
```

What happens when we build and run the code? Well, Jthereum converts your Java code into the Solidity code. Take a look at it:

```
Compiling BasicToken
Compiling com.u7.jthereum.wellKnownInterfaces.ERC20

Solidity Source:

// Generated by Jthereum BETA version!
pragma solidity ^0.5.9;
contract BasicToken
{
 string public name = "Basic Token";
 string public symbol = "BAS";
 uint8 public decimals = 8;
 uint256 private initialSupply = 100_000_000_000_000_000;
 mapping (address => uint256) private balancesByAddress;
 constructor() public
 {
  balancesByAddress[msg.sender] = initialSupply;
 }
 function transfer(address _recipient, uint256 _amount) public returns (bool)
 {
  require((balancesByAddress[msg.sender] >= _amount), "Not enough funds");
  require(!(_recipient == msg.sender), "No need to send tokens to yourself");
  uint256 newSenderBalance = (balancesByAddress[msg.sender] - _amount);
  uint256 newRecipientBalance = (balancesByAddress[_recipient] + _amount);
  require((newRecipientBalance >= balancesByAddress[_recipient]));
  balancesByAddress[msg.sender] = newSenderBalance;
  balancesByAddress[_recipient] = newRecipientBalance;
  emit Transfer(msg.sender, _recipient, _amount);
  return true;
 }
 function balanceOf(address who) public view returns (uint256)
 {
  return balancesByAddress[who];
 }


 // Imported Event declarations
 event Transfer(address indexed from, address indexed to, uint256 value);

 event Approval(address indexed owner, address indexed spender, uint256 value);


}
```

Then your Solidity code is being compiled and deployed on the Blockchain. In this case we specified the blockchain to be “ropsten”. Take a look at the output:

```
Argument values for call to Jthereum.compileAndDeploy():

  Argument               Source
 ==========             ========
 String blockchainName = "ropsten";       // directly supplied
 String web3jURL = "http://ropsten.jthereum.com:8545";     // global property: ROPSTE
 Class contractClass = BasicToken.class;       // call stack (when called from 'main(
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
 BigInteger gasPrice = 15,000,000,000;       // default gas price for 'ropsten' block
 BigInteger gasLimit = 4,000,000;       // default gas limit for 'ropsten' blockchain


Deploying contract to the blockchain

Tx hash: 0xfe52b5307420cf75c5c48b644eaea3474e5f98075377b0398052d5b5695891a9
Waiting for transaction to be mined.
Your contract was deployed in block #5,999,310
The new contract address is at: 0xf4c6a308757eb2e4cbfe137011c09dd2c7cbed39

Preparing to validate contract source code on Etherscan for blockchain 'ropsten'
Validating contract source code on Etherscan for blockchain 'ropsten'
Version hash: 5a6ea5b1
Compiler Version: v0.5.10+commit.5a6ea5b1
Result: {"status":"1","message":"OK","result":"ebfrwmpbjakx1u1u7i9gvru3tstf5zs3zzgzmi

Contract source code validation has been submitted.  You can view your contract on Et
https://ropsten.etherscan.io/address/0xf4c6a308757eb2e4cbfe137011c09dd2c7cbed39
```

Now argument values for Jthereum Proxy:

```
Argument values for call to Jthereum.createProxy():

  Argument               Source
 ==========             ========
 Class contractClass = BasicToken.class;       // call stack (when called from 'main(
 String blockchainName = "ropsten";       // most recent deployment for class BasicTo
 String web3jURL = "http://ropsten.jthereum.com:8545";     // global property: ROPSTE
 String address = "0xf4c6a308757eb2e4cbfe137011c09dd2c7cbed39";    // most recent dep
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
```

Finally, here is our output result:

```
Argument values for call to ContractProxyContextInfo.callContractTransaction():

  Argument               Source
 ==========             ========
 Class contractClass = BasicToken.class;       // inherited from Proxy Object configu
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xf4c6a308757eb2e4cbfe137011c09dd2c7cbed39";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
 BigInteger gasPrice = 15,000,000,000;       // default gas price for 'ropsten' block
 BigInteger gasLimit = 4,000,000;       // default gas limit for 'ropsten' blockchain


Calling transfer:

Tx hash: 0x8412a91b1f22f9a89391e84045f2aec5517b1a08b38a8fa705a0063add7f7b33
Waiting for transaction to be mined.
Gas Used: 49,936
Total WEI used: 749,040,000,000,000
Total ETH used: 0.000749
(Gas Price: 15,000,000,000 wei)
Transaction succeeded with status: 0x1


Argument values for call to Jthereum.getMyAddress():

  Argument               Source
 ==========             ========
 Class contractClass = BasicToken.class;       // call stack (when called from 'main(
 String blockchainName = "ropsten";       // most recent deployment for class BasicTo
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;



Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = BasicToken.class;       // inherited from Proxy Object configu
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xf4c6a308757eb2e4cbfe137011c09dd2c7cbed39";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling balanceOf:
My Balance: 99,999,999,999,999,000
```

Basic Info Getter Contract

On the Ethereum blockchain you can interact with different contracts. It creates a huge opportunity for various use cases. In this example we use different methods to retrieve information of a contract.

- It returns an address of the contract
- It returns an address of the contract's owner
- It returns an address of the sender
- It returns a balance of the contract
- It returns a balance of the contract's owner (ONLY if you are the owner)
- It returns a balance of the sender

If you wanted to write this contract in Solidity it would look as following:

```
pragma solidity ^0.4.22;

contract Checker {
    address owner;

    constructor() public {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    function getBalanceOfContract() public view returns(uint) {
        return address(this).balance;
    }

    function getBalanceOfOwner() public view onlyOwner returns(uint){
        return owner.balance;
    }

    function getBalanceOfSender() public view returns(uint) {
        return msg.sender.balance;
    }

    function getAddressOfContract() public view returns(address) {
        return this;
    }

    function getAddressOfOwner() public view returns(address) {
        return owner;
    }

    function getAddressOfSender() public view returns(address) {
        return msg.sender;
    }
}
```

And this is how it looks like in Java

```
package com.u7.jthereum.exampleContracts;

import com.u7.jthereum.annotations.*;
import com.u7.jthereum.types.*;

import java.math.*;

import static com.u7.util.gg.cf;
import static com.u7.jthereum.Jthereum.*;
import static com.u7.jthereum.SolidityStaticImports.*;

public class BasicInfoGetter
{
 Address owner;

 public BasicInfoGetter()
 {
  owner = msg.sender;
 }
```

```
// Allow the contract to receive payments
@FallbackFunction
@Payable
public void fallbackFunction()
{
}

@View
public Uint getBalanceOfContract()
{
 return address(this).balance;
}

@View
public Uint getBalanceOfOwner()
{
 onlyOwner();

 return owner.balance;
}

@View
public Uint getBalanceOfSender()
{
 return msg.sender.balance;
}

@View
public Address getAddressOfContract()
{
 return new Address(this);
}

@View
public Address getAddressOfOwner()
{
 return owner;
}

@View
public Address getAddressOfSender()
{
 return msg.sender;
}

@View
private void onlyOwner()
{
 require(isOwner(), "Only owner can do that!");
}

@View
private boolean isOwner()
{
 return msg.sender.equals(owner);
}



public static void main(final String[] args)
{
 final String address = compileAndDeploy("ropsten");

 // Send a few Wei to the contract
 sendEth(address, BigInteger.valueOf(5));
```

```
  final BasicInfoGetter a = createProxy(BasicInfoGetter.class);

  p("Address of Contract: " + a.getAddressOfContract());
  p("Address of Owner: " + a.getAddressOfOwner());
  p("Address of Sender: " + a.getAddressOfSender());
  p("Balance of Contract: " + cf(a.getBalanceOfContract()));
  p("Balance of Owner: " + cf(a.getBalanceOfOwner()));
  p("Balance of Sender: " + cf(a.getBalanceOfSender()));
 }
}
```

What happens when we build and run the code? Well, Jthereum converts your Java code into the Solidity code. Take a look at it:

```
Compiling BasicInfoGetter

Solidity Source:

// Generated by Jthereum BETA version!
pragma solidity ^0.5.9;
contract BasicInfoGetter
{
 address private owner;
 constructor() public
 {
  owner = msg.sender;
 }
 function () external payable
 {
 }
 function getBalanceOfContract() public view returns (uint)
 {
  return address(this).balance;
 }
 function getBalanceOfOwner() public view returns (uint)
 {
  onlyOwner();
  return owner.balance;
 }
 function getBalanceOfSender() public view returns (uint)
 {
  return msg.sender.balance;
 }
 function getAddressOfContract() public view returns (address)
 {
  return address(this);
 }
 function getAddressOfOwner() public view returns (address)
 {
  return owner;
 }
 function getAddressOfSender() public view returns (address)
 {
  return msg.sender;
 }
 function onlyOwner() private view
 {
  require(isOwner(), "Only owner can do that!");
 }
 function isOwner() private view returns (bool)
 {
  return (msg.sender == owner);
 }

}
```

Then your Solidity code is being compiled and deployed on the Blockchain. In this case we specified the blockchain to be “ropsten”. Take a look at the output:

```
Argument values for call to Jthereum.compileAndDeploy():

  Argument               Source
 ==========             ========
 String blockchainName = "ropsten";       // directly supplied
 String web3jURL = "http://ropsten.jthereum.com:8545";     // global property: ROPSTE
 Class contractClass = BasicInfoGetter.class;      // call stack (when called from `m
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
 BigInteger gasPrice = 15,000,000,000;       // default gas price for 'ropsten' block
 BigInteger gasLimit = 4,000,000;       // default gas limit for 'ropsten' blockchain


Deploying contract to the blockchain

Tx hash: 0x51a2f9ec5b1f13c31db7d4390972d827569b990e050af8820efbf9998f401780
Waiting for transaction to be mined.
Your contract was deployed in block #5,999,333
The new contract address is at: 0xfd324349df87ffa88f245b90e67f3ed6928358e3

Preparing to validate contract source code on Etherscan for blockchain 'ropsten'
Validating contract source code on Etherscan for blockchain 'ropsten'
Version hash: 5a6ea5b1
Compiler Version: v0.5.10+commit.5a6ea5b1
Result: {"status":"1","message":"OK","result":"uvx5ynhdqvu9vahhy1e2zpti3zp1bewq9f6wvu

Contract source code validation has been submitted.  You can view your contract on Et
https://ropsten.etherscan.io/address/0xfd324349df87ffa88f245b90e67f3ed6928358e3

Argument values for call to Jthereum.sendEth():

  Argument               Source
 ==========             ========
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // directly suppli
 Class contractClass = BasicInfoGetter.class;      // call stack (when called from `m
 String blockchainName = "ropsten";       // most recent deployment for class BasicIn
 String web3jURL = "http://ropsten.jthereum.com:8545";     // global property: ROPSTE
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
 BigInteger gasPrice = 15,000,000,000;       // default gas price for 'ropsten' block
 BigInteger gasLimit = 4,000,000;       // default gas limit for 'ropsten' blockchain


Tx hash: 0x3f2f8eb0b166429a3327a57f5cd695c01d83fda63ca10074f0c9074fed37d066
Waiting for transaction to be mined.
Gas Used: 21,040
Total WEI used: 315,600,000,000,000
Total ETH used: 0.000316
(Gas Price: 15,000,000,000 wei)
Total WEI sent as payment: 5
Total ETH sent as payment: 0.000000
Transaction succeeded with status: 0x1
```

Now argument values for Jthereum Proxy:

```
Argument values for call to Jthereum.createProxy():

  Argument               Source
 ==========             ========
 Class contractClass = BasicInfoGetter.class;      // directly supplied
 String blockchainName = "ropsten";       // most recent deployment for class BasicIn
 String web3jURL = "http://ropsten.jthereum.com:8545";     // global property: ROPSTE
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // most recent dep
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
```

Finally, here is our output result:

```
Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = BasicInfoGetter.class;      // inherited from Proxy Object con
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling getAddressOfContract (Not a Transaction!):
Address of Contract: 0xfd324349df87ffa88f245b90e67f3ed6928358e3

Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = BasicInfoGetter.class;      // inherited from Proxy Object con
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling getAddressOfOwner (Not a Transaction!):
Address of Owner: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa

Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = BasicInfoGetter.class;      // inherited from Proxy Object con
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling getAddressOfSender (Not a Transaction!):
Address of Sender: 0x0000000000000000000000000000000000000000

Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = BasicInfoGetter.class;      // inherited from Proxy Object con
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
```

```
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling getBalanceOfContract (Not a Transaction!):
Balance of Contract: 5

Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = BasicInfoGetter.class;      // inherited from Proxy Object con
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling getBalanceOfOwner (Not a Transaction!):
Balance of Owner: 3,963,877,391,197,344,453,575,983,046,348,115,674,221,700,746,820,7

Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = BasicInfoGetter.class;      // inherited from Proxy Object con
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xfd324349df87ffa88f245b90e67f3ed6928358e3";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling getBalanceOfSender (Not a Transaction!):
Balance of Sender: 115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,63

Process finished with exit code 0
```

Incrementer Contract

This example contract demonstrates a simple non-constant (transactional) function you can perform on the Ethereum blockchain.

If you wanted to write this contract in Solidity it would look as following:

```
contract Incrementer {

    address creator;
    uint iteration;

    function Incrementer() public
    {
        creator = msg.sender;
        iteration = 0;
    }

    function increment()
    {
        iteration = iteration + 1;
    }

    function getIteration() constant returns (uint)
    {
        return iteration;
    }

    /**********
     Standard kill() function to recover funds
    **********/

    function kill()
    {
        if (msg.sender == creator)
            suicide(creator);  // kills this contract and sends remaining funds back
    }

}
```

And this is how it looks like in Java

```
package com.u7.jthereum.exampleContracts;

import com.u7.jthereum.annotations.*;
import com.u7.jthereum.types.*;

import static com.u7.jthereum.Jthereum.*;
import static com.u7.jthereum.SolidityStaticImports.*;

public class Incrementer
{
 AddressPayable creator;
 Uint iteration;

 public Incrementer()
 {
  creator = msg.sender;
  iteration = Uint.ZERO;
 }

 public void increment()
 {
  iteration.increment();
 }

 @View
 @Constant
 public Uint getIteration()
 {
  return iteration;
 }

 /**********
  Standard kill() function to recover funds
  **********/

 public void kill()
 {
  if(msg.sender.equals(creator))
  {
   selfdestruct(creator);  // kills this contract and sends remaining funds back to c
  }
 }

 public static void main(final String[] args)
 {
  compileAndDeploy();

  final Incrementer a = createProxy(Incrementer.class);

  a.increment();
  a.increment();

  p("Iteration: " + a.getIteration().asBigInteger());
 }
}
```

What happens when we build and run the code? Well, Jthereum converts your Java code into the Solidity code. Take a look at it:

```
Compiling Incrementer

Solidity Source:

// Generated by Jthereum BETA version!
pragma solidity ^0.5.9;
contract Incrementer
{
 address payable private creator;
 uint private iteration;
 constructor() public
 {
  creator = msg.sender;
  iteration = 0;
 }
 function increment() public
 {
  (iteration ++);
 }
 function getIteration() public view returns (uint)
 {
  return iteration;
 }
 function kill() public
 {
  if ((msg.sender == creator))
  {
   selfdestruct(creator);
  }
 }

}
```

Then your Solidity code is being compiled and deployed on the Blockchain. In this case we specified the blockchain to be “ropsten”. Take a look at the output:

```
Argument values for call to Jthereum.compileAndDeploy():

  Argument               Source
 ==========             ========
 String blockchainName = "ropsten";       // directly supplied
 String web3jURL = "http://ropsten.jthereum.com:8545";     // global property: ROPSTE
 Class contractClass = Incrementer.class;      // call stack (when called from 'main(
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
 BigInteger gasPrice = 15,000,000,000;       // default gas price for 'ropsten' block
 BigInteger gasLimit = 4,000,000;       // default gas limit for 'ropsten' blockchain


Deploying contract to the blockchain

Tx hash: 0x6ba7dfe9f8361392aec11484faad17ab38c29eaf9d2c40cc9a24dae202a7bcfe
Waiting for transaction to be mined.
Your contract was deployed in block #5,999,354
The new contract address is at: 0xe1b34b1f593fba1a79d686f60e4d4b45b7402c95

Preparing to validate contract source code on Etherscan for blockchain 'ropsten'
Validating contract source code on Etherscan for blockchain 'ropsten'
Version hash: 5a6ea5b1
Compiler Version: v0.5.10+commit.5a6ea5b1
Result: {"status":"1","message":"OK","result":"uapsz3mfksjkjwfbtbctckgcgul7p88cnb598n

Contract source code validation has been submitted.  You can view your contract on Et
https://ropsten.etherscan.io/address/0xe1b34b1f593fba1a79d686f60e4d4b45b7402c95
```

Now argument values for Jthereum Proxy:

```
Argument values for call to Jthereum.createProxy():

  Argument               Source
 ==========             ========
 Class contractClass = Incrementer.class;      // directly supplied
 String blockchainName = "ropsten";       // most recent deployment for class Increme
 String web3jURL = "http://ropsten.jthereum.com:8545";     // global property: ROPSTE
 String address = "0xe1b34b1f593fba1a79d686f60e4d4b45b7402c95";    // most recent dep
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
```

Finally, here is our output result:

```
Argument values for call to ContractProxyContextInfo.callContractTransaction():

  Argument               Source
 ==========             ========
 Class contractClass = Incrementer.class;      // inherited from Proxy Object configu
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xe1b34b1f593fba1a79d686f60e4d4b45b7402c95";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
 BigInteger gasPrice = 15,000,000,000;       // default gas price for 'ropsten' block
 BigInteger gasLimit = 4,000,000;       // default gas limit for 'ropsten' blockchain


Calling increment:

Tx hash: 0x0d0a3f33e4ce3e34fcc3c00d5fc8b39f285757134660d557f4a3c2c2649b88fb
Waiting for transaction to be mined.
Gas Used: 41,631
Total WEI used: 624,465,000,000,000
Total ETH used: 0.000624
(Gas Price: 15,000,000,000 wei)
Transaction succeeded with status: 0x1


Argument values for call to ContractProxyContextInfo.callContractTransaction():

  Argument               Source
 ==========             ========
 Class contractClass = Incrementer.class;      // inherited from Proxy Object configu
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xe1b34b1f593fba1a79d686f60e4d4b45b7402c95";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;
 BigInteger gasPrice = 15,000,000,000;       // default gas price for 'ropsten' block
 BigInteger gasLimit = 4,000,000;       // default gas limit for 'ropsten' blockchain


Calling increment:

Tx hash: 0xf303f3ab84f2c0d84a5d8a808a535faf3bc56c83032b6e922cd15785733b3c42
Waiting for transaction to be mined.
Gas Used: 26,631
Total WEI used: 399,465,000,000,000
Total ETH used: 0.000399
(Gas Price: 15,000,000,000 wei)
Transaction succeeded with status: 0x1


Argument values for call to ContractProxyContextInfo.callContractViewOrPure():

  Argument               Source
 ==========             ========
 Class contractClass = Incrementer.class;      // inherited from Proxy Object configu
 String blockchainName = "ropsten";       // inherited from Proxy Object configuratio
 String web3jURL = "http://ropsten.jthereum.com:8545";     // inherited from Proxy Ob
 String address = "0xe1b34b1f593fba1a79d686f60e4d4b45b7402c95";    // inherited from
 BigInteger privateKey = pk for address: 0xb9e071fbc62fa28e9895b6031818ce20765fb6aa;


Calling getIteration (Not a Transaction!):
Iteration: 2

Process finished with exit code 0
```