Create a directory for our chaincode application.

mkdir -p $GOPATH/src/sacc && cd $GOPATH/src/sacc

Create the source file which will contain our chaincode code.

touch sacc**.**go

Add the necessary dependencies for our chaincode. We will import the chaincode shim package and the peer protobuf package.

package main  
**import** (  
 "fmt"  
 "github.com/hyperledger/fabric/core/chaincode/shim"  
 "github.com/hyperledger/fabric/protos/peer"  
)

Add a struct which will act as a receiver for the Chaincode shim function

type BillAsset struct {  
}

Implement the **Init** function

**//** Init **is** called during chaincode instantiation to initialize any data**.**  
func (t **\***BillAsset) Init(stub shim**.**ChaincodeStubInterface) peer**.**Response {  
  
}

This function was implemented in class by everyone. And I shared the final source code file as well. Therefore, I will leave out the details.

Add the **Invoke** function signature

**//** Invoke **is** called per transaction on the chaincode**.** Each transaction **is**  
**//** either a 'get' **or** a 'set' on the asset created by Init function**.** The 'set'  
**//** method may create a new asset by specifying a new key**-**value pair**.**  
func (t **\***BillAsset) Invoke(stub shim**.**ChaincodeStubInterface) peer**.**Response {  
  
}

This too was implemented in class. Along with the smartcontract functions “updateState” and “getState”

Finally, we need to add the **main** function, which will call the **shim.Start** function

**//** main function starts up the chaincode **in** the container during instantiate  
func main() {  
 **if** err :**=** shim**.**Start(new(BillAsset)); err **!=** nil {  
 fmt**.**Printf("Error starting BillAsset chaincode: %s", err)  
 }  
}

Now lets build our chaincode program

go get **-**u **--**tags nopkcs11 github**.**com**/**hyperledger**/**fabric**/**core**/**chaincode**/**shim  
go build **--**tags nopkcs11

To now test it, lets navigate to the **chaincode-docker-devmode** directory of the **fabric-samples** clone.

Update docker-compose-simple.yaml to mount the chaincode application we wrote to the chaincode container

Open 3 terminals. In the first terminal, we will start the network. In the second terminal, we will build and start the chaincode. And in the third terminal, we will use the chaincode.

.

So in the first terminal, where you must be in “**fabcar-samples/chaincode-docker-devmode**” directory, issue the following command:

docker**-**compose **-**f docker**-**compose**-**simple**.**yaml up

In the second terminal, issue the following commands:

docker exec **-**it chaincode bash

cd ../chaincodedev/cfd02  
go build

CORE\_PEER\_ADDRESS**=**peer:7052 CORE\_CHAINCODE\_ID\_NAME**=**mycc:0 **./**cfd02

Now, in the third terminal, we will issue commands to run the chaincode.

docker exec **-**it cli bash

peer chaincode install **-**p chaincodedev**/**chaincode**/**sacc **-**n mycc **-**v 0  
peer chaincode instantiate **-**n mycc **-**v 0 **-**c '{"Args":["a","10"]}' **-**C myc

Now, lets issue a command to change the value of “a” from “10” to “20”

peer chaincode invoke **-**n mycc **-**c '{"Args":["updateState", "a", "20"]}' **-**C myc

Finally, lets query “**a”**

peer chaincode query **-**n mycc **-**c '{"Args":["getState","naba"]}' **-**C myc

peer chaincode query -n mycc