How each contract work:

1. Associate Profit Splitter

The objective of this smart contract is to split the value deposited in the contract among the three wallets selected during contract deployment time.

* Source Code: “AssociateProfitSplitter.sol”
* How it works:
  + At deployment time, we need to provide 3 wallet addresses. In our case I am picking the following 3 addresses from Ganache:

Graphical user interface, table

Description automatically generated with medium confidence

The initial balance for these 3 wallets is 104 ETH. We will use the ending balance to demonstrate that a deposit had been made into the contract and moved to the accounts as the smart contract is programmed for.

Graphical user interface, text, application

Description automatically generated

* Proof of Contract Deployment:

Graphical user interface, text, application, email

Description automatically generated

Contract deployed at: 0xAa7B045Dc459889Eb46b6B8c9d8987847904EdFb

Graphical user interface, application, website

Description automatically generated

Now we will proceed to the distribution. The first step is to insert a value of ether that we want to distribute. In this case I am using 24 ETH.

Graphical user interface, application, Teams

Description automatically generated

Then in the deployed contrac, click “deposit”:

Graphical user interface, application, website

Description automatically generated

Authorize the payment of the transaction in the wallet:

Graphical user interface, application

Description automatically generated

Once the transaction is confirmed, means that the execution of the contract was completed.

If we look the balance of those 3 accounts used for deploying the smart contract, we would see that they increased by 8 ETH each as it was programmed in the smart contract to distribute evenly.

Graphical user interface, application

Description automatically generated

1. Tiered Profit-Splitter

The objective of this smart contract is to split the value deposited in the contract among the three wallets selected during contract deployment time, using a 65-25-10 ratio to split the ETH deposited in the contract. In case there is a reminder, it should go to the holder of the highest value wallet, which in this case is the 65% receipient.

* Source Code: “TieredProfitSplitter.sol”
* How it works:
  + Like In the previous case, at deployment time, we need to provide 3 wallet addresses. In our case I am picking the following 3 addresses from Ganache (for simplicity of calculations, I am using the same 3 addresses as in the previous case):

Graphical user interface, table

Description automatically generated

* + Like In the previous case, at deployment time, we need to provide 3 wallet addresses. In our case I am picking the following 3 addresses from Ganache (for simplicity of calculations, I am using the same 3 addresses as in the previous case):

Graphical user interface, text

Description automatically generated

The initial balance for these 3 wallets is 112 ETH. We will use the ending balance to demonstrate that a deposit had been made into the contract and moved to the accounts as the smart contract is programmed for.

* Proof of Contract Deployment:

Graphical user interface, text, application, email

Description automatically generated

Contract deployed at: 0xC82074Fc81413675aeF73430753503Ef43916198

Graphical user interface, application

Description automatically generated

Now we will proceed to the distribution. The first step is to insert a value of ether that we want to distribute. In this case I am using 100 ETH.

Graphical user interface, application, Teams

Description automatically generated

Then in the deployed contract, click “deposit”:

Graphical user interface, application

Description automatically generated

Authorize the payment of the transaction in the wallet:

Graphical user interface, application

Description automatically generated

Once the transaction is confirmed, means that the execution of the contract was completed.

If we look the balance of those 3 accounts used for deploying the smart contract, we would see that they increased by:

60 ETH the first wallet

25 ETH the second wallet

15 ETH the third wallet

Graphical user interface, application

Description automatically generated

Wallet ending in f86e: initial balance of 112 ETH + 60 ETH = 172 ETH

Wallet ending in 1Ca8: initial balance of 112 ETH + 25 ETH = 137 ETH

Wallet ending in 75AC: initial balance of 112 ETH + 15 ETH = 127 ETH

1. Deferred- Equity Plan

The objective of this smart contract is to distribute 1000 shares every year to the beneficiary of the smart contract

* Source Code: “DeferredEquityPlan.sol”
* How it works:
  + In this case, at deployment time we need to provide only 1 wallet address, the one of the person that will be beneficiary of this smart contract. We will use address 0xAE5d9aDDdcF93D5947EEC2De6F42684049d411DC, although this smart contract do not produce any change in the balance of the wallet, only transfers a number of shares to the account.

Graphical user interface, application

Description automatically generated

Graphical user interface, text

Description automatically generated

* Proof of Contract Deployment:

Graphical user interface, text, application, email

Description automatically generated

Contract deployed at: 0xAcCa360C20c72A4Ad66c3F1663f894a3EB8AaE83

Graphical user interface, website

Description automatically generated

Now we will proceed to the distribution. We need to make sure that the “value” field in remix is set to 0 and then click on distribute.

Graphical user interface, application

Description automatically generated

* Distribution Executed:

Graphical user interface, text, application, email

Description automatically generated

As this is a yearly distribution, the contract has a feature that does not allow for it to be executed more than once a year. So, when we execute it again, we obtain an error:

Text

Description automatically generated

We included a feature that allows us to bypass this limitation so we can test the functionality. In order to do so, we created the “fastforward” functionality which simulates moving in time 400 days in advance so we can re-run the distribution.

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Then we can distribute again:

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated