**Implementing the lottery state - Enum**

**Introduction to the Concept of Enum**

In Solidity, enum stands for Enumerable. It is a user-defined data type that restricts a variable to have only one of the predefined values listed within the enum declaration. These predefined values are internally treated as unsigned integers, starting from 0 up to the count of elements minus one. Enums are useful for improving code readability and reducing potential errors by limiting the range of acceptable values for a variable. Read more about enums [here](https://docs.soliditylang.org/en/v0.8.26/types.html#enums).

**How can we use enums in our Project?**

Let's think about all the possible states of our Raffle. We deploy the contract and the raffle is started, the participants buy a ticket to register. Let's call this state OPEN. After this, we have a period when we need to wait at least 3 blocks for Chainlink VRF to send us the random number this means that we have at least 36 seconds (12 seconds/block) of time when our Raffle is processing the winner. Let's call this state CALCULATING.

Let's code all these!

Paste the following code between the errors definition section and the state variables section:

// Type declarations

enum RaffleState {

OPEN, // 0

CALCULATING // 1

}

// Put this one in `Raffle related variables`

RaffleState private s\_raffleState;

Amazing, let's default our raffle state to open inside the constructor.

Add the following inside your constructor:

s\_raffleState = RaffleState.OPEN;

Amazing! But what's the reason we did all this? Security! The thing we love the most!

Chainlink VRF has an [interesting page](https://docs.chain.link/vrf/v2-5/security) where they provide Security Considerations you should always implement when interacting with their service. One of these is called Don't accept bids/bets/inputs after you have made a randomness request, in our case this translates to Don't let people buy tickets while we calculate the final winner. I strongly encourage you to give that whole page a read, it will save you a lot of headaches.

Let's implement this in the code:

function enterRaffle() external payable {

if(msg.value < i\_entranceFee) revert Raffle\_\_NotEnoughEthSent();

if(s\_raffleState != RaffleState.OPEN) revert Raffle\_\_RaffleNotOpen(); // If not open you don't enter.

s\_players.push(payable(msg.sender));

emit EnteredRaffle(msg.sender);

}

Make sure to also define the new Raffle\_\_RaffleNotOpen() error.

Great, now let's also change the state of the Raffle when we commence the process of picking the winner.

function pickWinner() external {

// check to see if enough time has passed

if (block.timestamp - s\_lastTimeStamp < i\_interval) revert();

s\_raffleState = RaffleState.CALCULATING;

}

The last thing we need to do is to reopen the Raffle after we pick the winner inside fulfillRandomWords function.

function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords) internal override {

uint256 indexOfWinner = randomWords[0] % s\_players.length;

address payable winner = s\_players[indexOfWinner];

s\_recentWinner = winner;

s\_raffleState = RaffleState.OPEN;

(bool success,) = winner.call{value:address(this).balance}("");

if (!success) {

revert Raffle\_\_TransferFailed();

}

}

I know you thought about it: But why are we opening the Raffle again? We've selected a winner but the s\_players array is still full! And you are right!

We will take care of this in the next lesson!