한 번에 끝내는
블록체인 개발 A to Z

Chapter 3

Lottery 컨트랙트 v1 개발

Chapter 3

Lottery 컨트랙트 v1 개발

Lottery 컨트랙트 테스트하기 - hardhat test

Hardhat test

- Hardhat은 mocha 프레임워크를 기반으로 chai 라이브러리 사용함
- Hardhat은 web3 대신 ethers 라이브러리 사용
- Hardhat 프로젝트 초기화시 테스트용 패키지 설치 선택 가능
- 이더리움 테스트용 패키지로 컨트랙트 이벤트 및 revert 체크 모듈, 확장된 chai expect 함수 제공

Hardhat의 테스트용 패키지

- @nomicfoundation/hardhat-network-helpers
  - Hardhat 블록체인 네트워크와의 손쉬운 연결 지원
  - loadFixture: 여러 개의 컨트랙트를 반복해서 재배포할 때 사용하면 효율적
    - beforeEach()에서 컨트랙트를 재배포하던 기존 형식을 효율적으로 개선한 것
    - 재배포하는 대신 state를 reset 시켜줌 (faster than 실제 재배포)
- @nomicfoundation/hardhat-chai-matchers
  - 이더리움 테스트용으로 확장된 chai 라이브러리 지원
  - event 파라미터 체크, revert 체크, bignumber 비교시 유용

```
const { loadFixture } = require("@nomicfoundation/hardhat-network-helpers");
require("@nomicfoundation/hardhat-chai-matchers");
```

Hardhat의 테스트용 패키지

- chai 라이브러리의 expect
  - Hardhat에선 expect() 함수를 사용하려면 chai 모듈을 import 해줘야함
  - @nomicfoundation/hardhat-chai-matchers도 같이 import 해준다면, expect의 기능을 이더리움 테스트용으로 확장하여 사용 가능
- Hardhat의 ethers
  - Hardhat에서 제공하는 ethers 라이브러리 import

```
const { expect } = require("chai");
const { ethers } = require("hardhat");
```

컨트랙트 연결하기

- 테스트 케이스 실행 전, before()에서 Lottery 컨트랙트 배포 후 전역변수에 저장
- Hardhat에서 기본적으로 제공해주는 signers 리스트를 전역 변수에 저장
- signers = await ethers.getSigners()
- 여러 번 배포할 경우를 대비해, 배포 함수를 선언하고 Hardhat에서 제공하는 loadFixture() 이용해 배포

```
describe("Lottery", () => {
    async function deployLottery() {
        const Signers = await ethers.getSigners();

        const LotteryContract = await ethers.getContractFactory("Lottery");
        const Lottery = await LotteryContract.deploy();

        return { Lottery, Signers };
    }

    let lottery;
    let signers;

    before(async () => {
        const { Lottery, Signers } = await loadFixture(deployLottery);
        lottery = Lottery;
        signers = Signers;
    });
```

배포 후 생성자 값 체크

- Lottery owner가 signers[0]의 주소로 잘 세팅 되어있는지 체크

```
describe("Constructor", () => {
    it("Owner should be set to signers[0]", async () => {
        const owner = await lottery.owner();
        expect(owner).to.equal(signers[0].address);
    });
});
```

enter() 테스트

- 먼저 require() 구문이 잘 동작하는지 체크
- 0.01 ETH 미만의 금액 전송 시 revert가 잘 되는지 확인하기 위해 Hardhat에서 제공하는 이더리움 테스트용으로 확장된 expect 기능 이용
  - await expect(테스트할 함수).to.be.revertedWith(컨트랙트에서 사용한 revert 메시지)
  - lottery.connect(signers[1]).enter(): ethers에서 트랜잭션의 from account를 변경하기 위해선 .connect(signer)로 사용

```
function enter() public payable {
    require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
    players.push(payable(msg.sender));
}
```

```
describe("Enter", () => {
    it("Should revert if a player enters less than 0.01 ether", async () => {
        const enterAmt = ethers.utils.parseEther("0.009");
        console.log(`enterAmt: ${enterAmt}`);

        await expect(lottery.connect(signers[1]).enter({ value: enterAmt })).to.be.revertedWith("msg.value should be greater than or equal to 0.01 ether");
    });
```

enter() 테스트

- 5명의 player가 enter 하는 상황 테스트
- 한 명씩 enter 할 때마다 컨트랙트의 ETH balance가 의도한대로 느는지, players 배열에 의도한대로 account가 저장되는지 체크
- Hardhat에서 제공하는 이더리움 테스트용으로 확장된 expect를 이용함으로써 bignumber 타입을 string 타입으로 치환하여 비교할 필요 x
- value equal 비교에선 expect().to.equal() 사용
- object(array 포함) equal 비교에선 expect().to.deep.equal() 사용

```
it("Enter 5 players and check values", async () => {
    const enterAmt = ethers.utils.parseEther("0.01");
    console.log(`enterAmt: ${enterAmt}`);

    // player1 enter
    await lottery.connect(signers[1]).enter({ value: enterAmt });
    expect(await lottery.getBalance()).to.equal(enterAmt);
    expect(await lottery.getPlayers()).to.deep.equal([signers[1].address]);

    // player2 enter
    await lottery.connect(signers[2]).enter({ value: enterAmt });
    expect(await lottery.getBalance()).to.equal(enterAmt.mul(2));
    expect(await lottery.getPlayers()).to.deep.equal([signers[1].address, signers[2].address]);
```

pickWinner() 테스트

- pickWinner()는 컨트랙트 owner만 호출 가능
- owner가 아닌 계정이 콜했을 때 정상적으로 revert 되는지 확인

```
describe("PickWinner", () => {
    it("Should revert if pickWinner is called by not owner", async () => {
        // owner: signers[0]
        await expect(lottery.connect(signers[1]).pickWinner()).to.be.revertedWith("You're not the owner");
    });
```

pickWinner() 테스트

- pickWinner 전, 후 상황 확인 및 테스트
- pickWinner() 호출 전, 5명의 player의 ETH 잔액 체크
  - pickWinner 후, winner의 ETH balance가 0.05 ETH 늘었는지 체크하기 위함
  - ethers.provider.getBalance() = web3.eth.getBalance()
- pickWinner() 호출

```
it("PickWinner", async () => {
    console.log(">>> before pickWinner");

    // check players' ETH balances before pickWinner
    const account1ETHBal_bef = await ethers.provider.getBalance(signers[1].address);
    console.log(`account1's ETH balance: ${account1ETHBal_bef}`);
    const account2ETHBal_bef = await ethers.provider.getBalance(signers[2].address);
    console.log(`account2's ETH balance: ${account2ETHBal_bef}`);
    const account3ETHBal_bef = await ethers.provider.getBalance(signers[3].address);
    console.log(`account3's ETH balance: ${account3ETHBal_bef}`);
    const account4ETHBal_bef = await ethers.provider.getBalance(signers[4].address);
    console.log(`account4's ETH balance: ${account4ETHBal_bef}`);
    const account5ETHBal_bef = await ethers.provider.getBalance(signers[5].address);
    console.log(`account5's ETH balance: ${account5ETHBal_bef}`);

    // pickWinner
    console.log(">>> pickWinner");
    await lottery.pickWinner();
```

pickWinner() 테스트

- pickWinner() 호출 후
- lotteryId가 정상적으로 1 증가했는지 체크
- lotteryHistory에 0회차 winner가 잘 저장됐는지 체크
- 5명의 player의 ETH balance 체크
- pickWinner() 호출 전후로 0.05 ETH 늘어난 계정 체크

```
console.log(">>> after pickWinner");

const lotteryId = await lottery.lotteryId();
console.log(`lotteryId: ${lotteryId}`);
expect(lotteryId).to.equal(1);

const winner = await lottery.lotteryHistory(lotteryId - 1);
console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

// check players' ETH balances after pickWinner
const account1ETHBal_aft = await ethers.provider.getBalance(signers[1].address);
console.log(`account1's ETH balance: ${account1ETHBal_aft}`);
const account2ETHBal_aft = await ethers.provider.getBalance(signers[2].address);
console.log(`account2's ETH balance: ${account2ETHBal_aft}`);
const account3ETHBal_aft = await ethers.provider.getBalance(signers[3].address);
console.log(`account3's ETH balance: ${account3ETHBal_aft}`);
const account4ETHBal_aft = await ethers.provider.getBalance(signers[4].address);
console.log(`account4's ETH balance: ${account4ETHBal_aft}`);
const account5ETHBal_aft = await ethers.provider.getBalance(signers[5].address);
console.log(`account5's ETH balance: ${account5ETHBal_aft}`);

// check balance difference
console.log(`account1 balance difference: ${account1ETHBal_aft.sub(account1ETHBal_bef)}`);
console.log(`account2 balance difference: ${account2ETHBal_aft.sub(account2ETHBal_bef)}`);
console.log(`account3 balance difference: ${account3ETHBal_aft.sub(account3ETHBal_bef)}`);
console.log(`account4 balance difference: ${account4ETHBal_aft.sub(account4ETHBal_bef)}`);
console.log(`account5 balance difference: ${account5ETHBal_aft.sub(account5ETHBal_bef)}`);
```

getRandomNumber() 테스트

- getRandomNumber() 사용하는 pickWinner() 호출 후
- getRandomNumber()의 랜덤값과 getRandomNumber()의 PRNG를 구현하여 계산한 랜덤값이 같은지 체크
- ethers.BigNumber.from() = web3.utils.toBN()
- ethers.utils.solidityKeccak256([type1, …], [value1, …]) = web3.utils.keccak256(web3.utils.encodePacked({value: value1, type: type1}, …))
- 계산해서 예측한 winner와 실제 winner가 같은지 체크

```
it("Calculate winner - getRandomNumber", async () => {
    const lotteryId = await lottery.lotteryId();
    console.log(`lotteryId: ${lotteryId}`);

    const winner = await lottery.lotteryHistory(lotteryId - 1);
    console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

    const randomNum = await lottery.getRandomNumber();
    console.log(`randomNumber: ${randomNum}`);

    const blockNumber = await ethers.provider.getBlockNumber();
    console.log(`block number: ${blockNumber}`);

    const currentBlock = await ethers.provider.getBlock(blockNumber);
    console.log(`current block:`, currentBlock);

    const calculatedRandomNum = ethers.BigNumber.from(ethers.utils.solidityKeccak256(["address", "uint256"], [await lottery.owner(), currentBlock.timestamp]));
    console.log(`calculated random number: ${calculatedRandomNum}`);
    expect(randomNum).to.equal(calculatedRandomNum);

    const calculatedWinnerIndex = ethers.BigNumber.from(ethers.utils.solidityKeccak256(["address", "uint256"], [await lottery.owner(), currentBlock.timestamp])).
    mod(5);
    console.log(`calculated winner index: ${calculatedWinnerIndex}`);
    expect(winner).to.equal(signers[calculatedWinnerIndex.add(1)].address);
});
```

getRandomNumberV2() 테스트

- getRandomNumberV2() 사용하는 pickWinner() 호출 후
- 계산해서 예측한 winner와 실제 winner가 같은지 체크
- 여기선 getRandomNumberV2()의 랜덤값과 getRandomNumberV2()의 PRNG를 구현하여 계산한 랜덤값이 같은지 체크 x
    - V2의 경우, pickWinner() 호출시 V2에서 사용하는 players 배열이 리셋됨 → pickWinner()가 호출된 후 호출하는 V2의 랜덤값은 pickWinner() 실행시 사용된 랜덤값과 다른 값

```
it("Calculate winner - getRandomNumberV2", async () => {
    // V2에서만 getRandomNumberV2() 호출후 해당 랜덤값을 web3를 이용해서 구한 랜덤값과 비교 x -> 이미 pickWinner가 불리고 나면 players 배열이 리셋되기 때문에 사후에 구할 수 없음
    const lotteryId = await lottery.lotteryId();
    console.log(`lotteryId: ${lotteryId}`);

    const winner = await lottery.lotteryHistory(lotteryId - 1);
    console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

    const blockNumber = await ethers.provider.getBlockNumber();
    console.log(`block number: ${blockNumber}`);

    const currentBlock = await ethers.provider.getBlock(blockNumber);
    console.log(`current block:`, currentBlock);

    const calculatedRandomNum = ethers.BigNumber.from(ethers.utils.solidityKeccak256(["address", "uint256"], [await lottery.owner(), currentBlock.timestamp]));
    console.log(`calculated random number: ${calculatedRandomNum}`);

    const calculatedWinnerIndex = ethers.BigNumber.from(ethers.utils.solidityKeccak256(["uint256", "uint256", "address[]"], [currentBlock.difficulty,
    currentBlock.timestamp, [signers[1].address, signers[2].address, signers[3].address, signers[4].address, signers[5].address]])).mod(5);
    console.log(`calculated winner index: ${calculatedWinnerIndex}`);
    expect(winner).to.equal(signers[calculatedWinnerIndex.add(1)].address);
});
```

getRandomNumberV3() 테스트

- getRandomNumberV3() 사용하는 pickWinner() 호출 후
- getRandomNumberV3()의 랜덤값과 getRandomNumberV3()의 PRNG를 구현하여 계산한 랜덤값이 같은지 체크
- 계산해서 예측한 winner와 실제 winner가 같은지 체크

```
it("Calculate winner - getRandomNumberV3", async () => {
    const lotteryId = await lottery.lotteryId();
    console.log(`lotteryId: ${lotteryId}`);

    const winner = await lottery.lotteryHistory(lotteryId - 1);
    console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

    const randomNum = await lottery.getRandomNumberV3();
    console.log(`randomNumber: ${randomNum}`);

    const blockNumber = await ethers.provider.getBlockNumber();
    console.log(`block number: ${blockNumber}`);

    const currentBlock = await ethers.provider.getBlock(blockNumber);
    console.log(`current block:`, currentBlock);

    const calculatedRandomNum = ethers.BigNumber.from(ethers.utils.solidityKeccak256(["bytes32", "uint256"], [currentBlock.parentHash, currentBlock.timestamp]));
    console.log(`calculated random number: ${calculatedRandomNum}`);
    expect(randomNum).to.equal(calculatedRandomNum);

    const calculatedWinnerIndex = ethers.BigNumber.from(ethers.utils.solidityKeccak256(["bytes32", "uint256"], [currentBlock.parentHash, currentBlock.
    timestamp])).mod(5);
    console.log(`calculated winner index: ${calculatedWinnerIndex}`);
    expect(winner).to.equal(signers[calculatedWinnerIndex.add(1)].address);
});
```