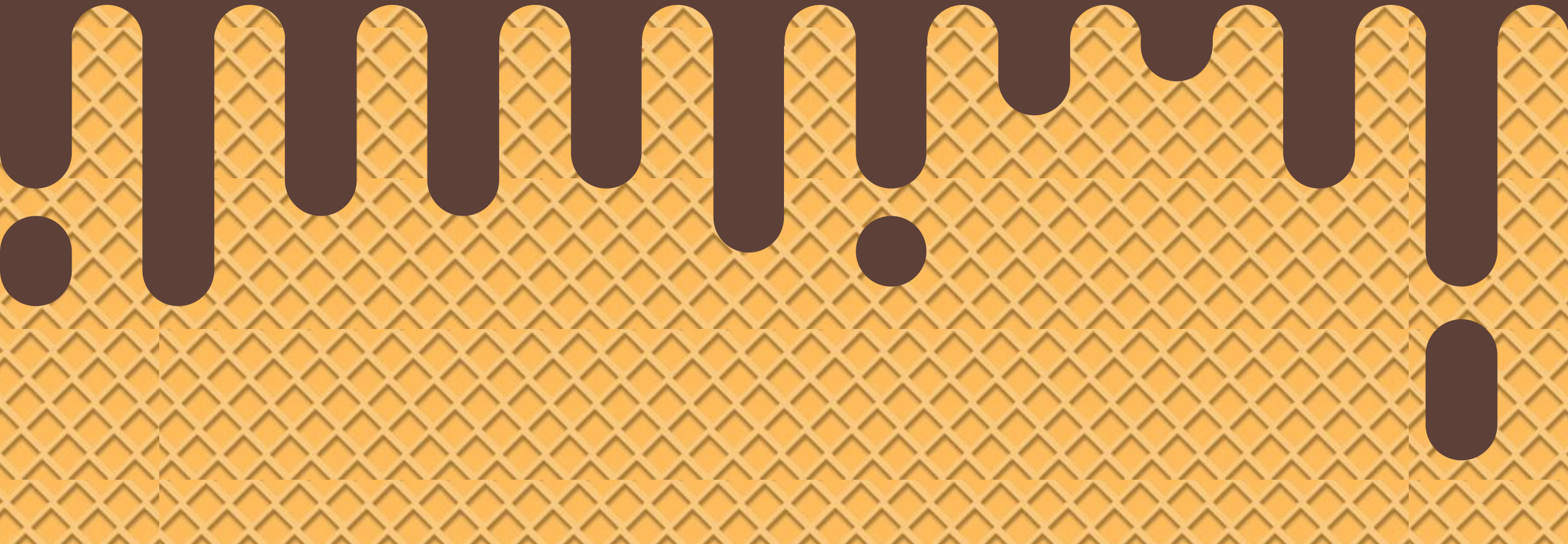
SKKRYPTO

SKKRYPTO DEV

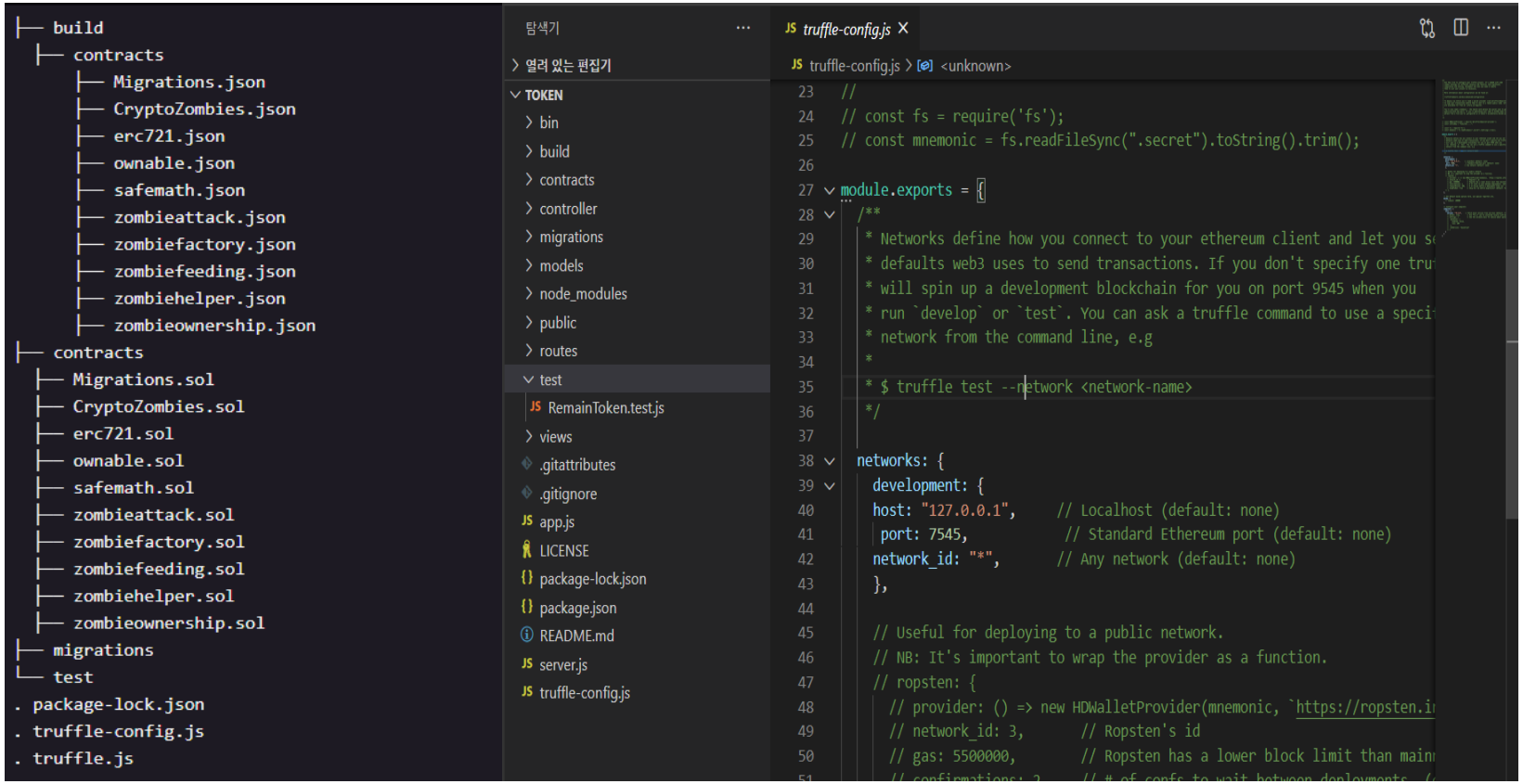
Testing Smart Contracts with Truffle

5기 배준호



Solidity Tutorial

CH1. Getting Setup

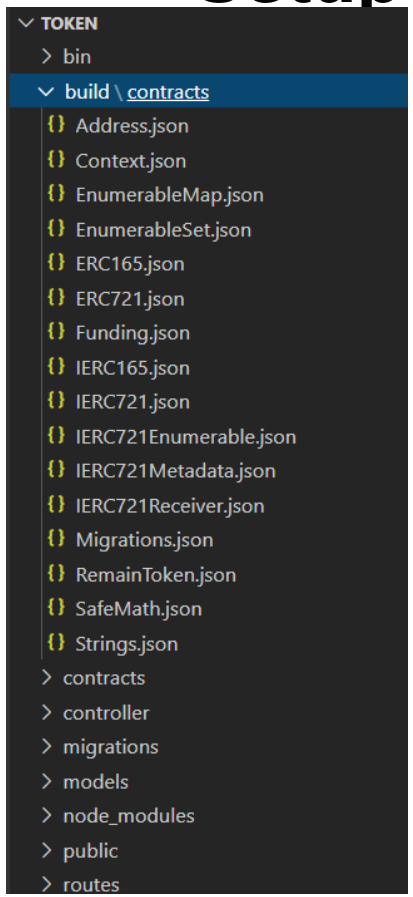


1. touch test/CryptoZombies.js

2. Truffle, Mocha, Chai

Solidity Tutorial

CH3. Getting Setup




```
1  const CryptoZombies = artifacts.require("CrptoZombies");
2
3  contract("CryptoZombies", (accounts)=> {
4      it("should be able to recieve Ethers",() => {
5
6      })
7  })
```

2. CyptoZombies = artifacts.require("CryptoZombies")
=> CryptoZombies라는 build artifacts를 가져온다. Contract abstractions 반환

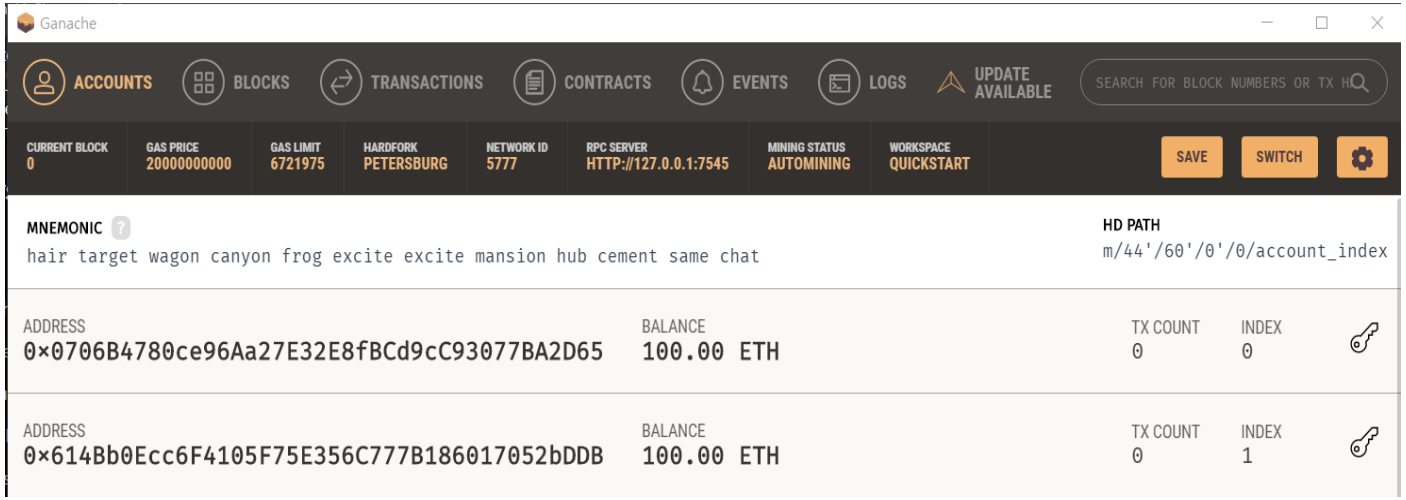
1. build/contracts : 컨트랙트가 컴파일 되면 컴파일러가 json 형식의 build artifacts를 만듬
-> 테스트 할 때 이 build artifacts를 가져온다.

3. contract() 메소드를 통해 group test 진행 가능
   - describe라는 Mocha의 메서드를 확장. Accounts 제공
   - "contract('이름', '콜백')"
4. it("테스트 설명", "콜백")

Solidity Tutorial

CH4. The First Test

```
1  const CryptoZombies = artifacts.require("CryptoZombies");
2  contract("CryptoZombies", (accounts) => {
3      //1. initialize `alice` and `bob`
4      let [alice, bob] = accounts;
5      it("should be able to create a new zombie", async () => { //2 & 3. Replace t
6      })
7  })
8
```



1. 가나슈로부터 account를 받아와 alice와 bob으로 할당

2. Async 키워드 추가하여 콜백 실행

Solidity Tutorial

CH4. The First Test

test/CryptoZombies.js

```
1  const CryptoZombies = artifacts.require("CryptoZombies");
2  const zombieNames = ["Zombie 1", "Zombie 2"];
3  contract("CryptoZombies", (accounts) => {
4      let [alice, bob] = accounts;
5      it("should be able to create a new zombie", async () => {
6          const contractInstance = await CryptoZombies.new();
7      })
8  })
9
```

1. 일반적으로 테스트는
   1) Setup
   2) Act
   3) Assert
   의 단계를 거침

2. Set up
현재 컨트랙트를 불러오기는 했지만 abstract일뿐, 실제 테스트를 위해서는 객체를 만들어줘야함!

3. Await
Block chain과 소통하기 때문에 await 키워드 추가

Solidity Tutorial

CH5. The First Test

test/CryptoZombies.js

```
1  const CryptoZombies = artifacts.require("CryptoZombies");
2  const zombieNames = ["Zombie 1", "Zombie 2"];
3  contract("CryptoZombies", (accounts) => {
4      let [alice, bob] = accounts;
5
6
7      it("should be able to create a new zombie", async () => {
8          const contractInstance = await CryptoZombies.new();
9          const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
10         assert.equal(result.receipt.status, true);
11         assert.equal(result.logs[0].args.name,zombieNames[0]);
12     })
13
14
15 })
16
```

1. result
- 새로 생성한 컨트랙트의 인스턴스로부터 컨트랙트의 메서드 실행
- Await 키워드 잊지 않기
- {}안에 from : alice가 msg.sender
- Truffle에서 log를 생성 해줘서 resul에 담김!
- Txhash 등등 다양한 값을 가져올 수 있음!
- 데이터 저장으로 사용가능! (컨트랙트에서 접근은 불가능)

2. Assert
- Assert라는 built-in 메서드를 통해 검사

Solidity Tutorial

CH6. Keeping the Fun

test/CryptoZombies.js | test/helpers/utils.js

```
1  const CryptoZombies = artifacts.require("CryptoZombies");
2  const utils = require("./helpers/utils");
3  const zombieNames = ["Zombie 1", "Zombie 2"];
4  contract("CryptoZombies", (accounts) => {
5      let [alice, bob] = accounts;
6      let contractInstance;
7      beforeEach(async () => {
8          contractInstance = await CryptoZombies.new();
9      });
10     it("should be able to create a new zombie", async () => {
11         const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
12         assert.equal(result.receipt.status, true);
13         assert.equal(result.logs[0].args.name,zombieNames[0]);
14     })
15     it("should not allow two zombies", async () => {
16     })
17 })
18
```

```
afterEach(async () => {
    await contractInstance.kill();
});
```

```
function kill() public onlyOwner {
    selfdestruct(owner());
}
```

1. beforeEach :
- 테스트 코드를 작성할 때 매번 컨트랙트의 인스턴스를 선언해야 한다면 매우 번거로움
- 테스트 전에 실행되는 beforeEach에서 컨트랙트 인스턴스를 새로 선언.

2. Selfdestruct()
- contract.new()를 사용하면 트러플은 매번 이 커트랙트를 배포함.
- 만약 계속해서 컨트랙트가 배포된다면 비효율적임.
- Selfdestruct()를 통해서 스마트 컨트랙트가 스스로를 지울 수 있게 할 수 있다.
- afterEach를 사용해 매번 테스트가 끝날때마다 스스로를 없엘 수 있음.

Solidity Tutorial

CH7. Keeping the Fun

```
1  const CryptoZombies = artifacts.require("CryptoZombies");
2  const utils = require("./helpers/utils");
3  const zombieNames = ["Zombie 1", "Zombie 2"];
4  contract("CryptoZombies", (accounts) => {
5      let [alice, bob] = accounts;
6      let contractInstance;
7      beforeEach(async () => {
8          contractInstance = await CryptoZombies.new();
9      });
10     it("should be able to create a new zombie", async () => {
11         const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
12         assert.equal(result.receipt.status, true);
13         assert.equal(result.logs[0].args.name,zombieNames[0]);
14     })
15     it("should not allow two zombies", async () => {
16         await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
17         await utils.shouldThrow(contractInstance.createRandomZombie(zombieNames[1], {from: alice}));
18     })
19 })
20
```

```
async function shouldThrow(promise) {
  try {
    await promise;
    assert(true);
  }
  catch (err) {
    return;
  }
assert(false, "The contract did not throw.");

}

module.exports = {
  shouldThrow,
};
```

- 우리의 테스트 코드는 “스마트 컨트랙트가 에러를 내면 “ 통과하게 할 것이기 때문에, 로직을 shouldThrow와 같이 작성
- 코드를 간결하고 이쁘게 관리하기 위해 다른 파일로 빼냄

Solidity Tutorial

CH8. Zombie Transfer

```
xcontext("with the single-step transfer scenario", async () => {
    it("should transfer a zombie", async () => {
      // start here.
    })
})
xcontext("with the two-step transfer scenario", async () => {
    it("should approve and then transfer a zombie when the approved address calls transferForm", async () => {
      // TODO: Test the two-step scenario.  The approved address calls transferFrom
    })
    it("should approve and then transfer a zombie when the owner calls transferForm", async () => {
       // TODO: Test the two-step scenario.  The owner calls transferFrom
     })
```

- Context 메서드 : 시나리오를 위한 그룹 테스트시 사용

- It or context 앞에 x를 붙이면 트러플이 얘네를 패스하고 테스트 진행

```
with the single-step transfer scenario
  - should transfer a zombie
with the two-step transfer scenario
  - should approve and then transfer a zombie when the owner calls transferForm
  - should approve and then transfer a zombie when the approved address calls trar
```

Solidity Tutorial

CH9. ERC 721 Transfer –Single Step Scenario

```
xcontext("with the single-step transfer scenario", async () => {
    it("should transfer a zombie", async () => {
      // start here.
    })
})
xcontext("with the two-step transfer scenario", async () => {
    it("should approve and then transfer a zombie when the approved address calls transferForm", async () => {
      // TODO: Test the two-step scenario.  The approved address calls transferFrom
    })
    it("should approve and then transfer a zombie when the owner calls transferForm", async () => {
       // TODO: Test the two-step scenario.  The owner calls transferFrom
     })
```

- Context 메서드 : 시나리오를 위한 그룹 테스트시 사용

- It or context 앞에 x를 붙이면 트러플이 얘네를 패스하고 테스트 진행

```
with the single-step transfer scenario
  - should transfer a zombie
with the two-step transfer scenario
  - should approve and then transfer a zombie when the owner calls transferForm
  - should approve and then transfer a zombie when the approved address calls trar
```

Solidity Tutorial

CH9. ERC 721 Transfer –Single Step Scenario

```
context("with the single-step transfer scenario", async () => {
    it("should transfer a zombie", async () => {
        const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
        const zombieId = result.logs[0].args.zombieId.toNumber();
        await contractInstance.transferFrom(alice, bob, zombieId, {from: alice});
        const newOwner = await contractInstance.ownerOf(zombieId);
        assert.equal(newOwner, bob);
    })
})
xcontext("with the two-step transfer scenario", async () => {
```

- Context 앞에 x 지우기!
- 지금까지 쓴 내용 그대로.

Solidity Tutorial

CH10. ERC 721 Transfer –Two Step Scenario

```
context("with the single-step transfer scenario", async () => {
    it("should transfer a zombie", async () => {
        const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
        const zombieId = result.logs[0].args.zombieId.toNumber();
        await contractInstance.transferFrom(alice, bob, zombieId, {from: alice});
        const newOwner = await contractInstance.ownerOf(zombieId);
        assert.equal(newOwner, bob);
    })
})
xcontext("with the two-step transfer scenario", async () => {
```

- Context 앞에 x 지우기!
- 지금까지 쓴 내용 그대로.

CH10. ERC 721 Transfer –Two Step Scenario

```
context("with the two-step transfer scenario", async () => {
    it("should approve and then transfer a zombie when the approved address calls transferForm", async () => {
        const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
        const zombieId = result.logs[0].args.zombieId.toNumber();
        await contractInstance.approve(bob, zombieId, {from: alice});
        await contractInstance.transferFrom(alice, bob, zombieId, {from: bob});
        const newOwner = await contractInstance.ownerOf(zombieId);
        assert.equal(newOwner,bob);
    })
```

- Two-step 중 첫번 째 case! : alice가 bob에게 approve하고, bob이 alice의 좀비를 가져오는 시나리오
- 좀비의 주인이 bob으로 잘 바뀌었는지 확인.

CH11. ERC 721 Transfer –Two Step Scenario

```
    it("should approve and then transfer a zombie when the owner calls transferForm", async () => {
        const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
        const zombieId = result.logs[0].args.zombieId.toNumber();
        await contractInstance.approve(bob, zombieId, {from: alice});
        await contractInstance.transferFrom(alice, bob, zombieId, {from: alice});
        const newOwner = await contractInstance.ownerOf(zombieId);
        assert.equal(newOwner,bob);
    })
```

- Two-step 중 두번 째 case! : alice가 bob에게 approve하고, alice가 alice의 좀비를 bob에게 보내는 시나리오

- 차이는 transfer from의 msg.sender가 누군지만 다를뿐!

- 좀비의 주인이 bob으로 잘 바뀌었는지 확인.

Solidity Tutorial

CH12. Zombie Attack

```
it("zombies should be able to attack another zombie", async () => {
    let result;
    result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
    const firstZombieId = result.logs[0].args.zombieId.toNumber();
    result = await contractInstance.createRandomZombie(zombieNames[1], {from: bob});
    const secondZombieId = result.logs[0].args.zombieId.toNumber();
    //TODO: increase the time
    await time.increase(time.duration.days(1));
    await contractInstance.attack(firstZombieId, secondZombieId, {from: alice});
    assert.equal(result.receipt.status, true);
})
```

- Zombie attack은 위의 코드대로 실행하면 cooldown으로 인해 attack이 실패
- Time Travel 필요!

Solidity Tutorial

CH12. Zombie Attack (Time Travel)

```
async function increase(duration) {

    //first, let's increase time
    await web3.currentProvider.sendAsync({
        jsonrpc: "2.0",
        method: "evm_increaseTime",
        params: [duration], // there are 86400 seconds in a day
        id: new Date().getTime()
    }, () => {});

    //next, let's mine a new block
    web3.currentProvider.send({
        jsonrpc: '2.0',
        method: 'evm_mine',
        params: [],
        id: new Date().getTime()
    })

}
```

- Web3 메서드를 이용해 블록체인과 소통
- Evm_increaseTime 요청을 보내고, evm_mine을 시작하면 다음 블록에 찍힌 타임스탬프를 하루 늦게 찍는 것이 가능.

```
const duration = {

    seconds: function (val) {
        return val;
    },
    minutes: function (val) {
        return val * this.seconds(60);
    },
    hours: function (val) {
        return val * this.minutes(60);
    },
    days: function (val) {
        return val * this.hours(24);
    },
}

module.exports = {
    increase,
    duration,
};
```

- 시간 단위를 초로 변환

Solidity Tutorial

CH13. Chai

Once you've finished this lesson, feel free check out their guides to further your knowledge.

That said, let's take a look at the three kinds of assertion styles bundled into `Chai`:

- *expect*: lets you chain natural language assertions as follows:

```
let lessonTitle = "Testing Smart Contracts with Truffle";
expect(lessonTitle).to.be.a("string");
```

- *should*: allows for similar assertions as `expect` interface, but the chain starts with a `should` property:

```
let lessonTitle = "Testing Smart Contracts with Truffle";
lessonTitle.should.be.a("string");
```

- *assert*: provides a notation similar to that packaged with node.js and includes several additional tests and it's browser compatible:

```
let lessonTitle = "Testing Smart Contracts with Truffle";
assert.typeOf(lessonTitle, "string");
```

```
    //TODO: replace with expect
    expect(result.receipt.status).to.equal(true);
})
```

- Npm install chai
- var expect = require('chai').expect;

Solidity Tutorial

CH14. Chai

Once you've finished this lesson, feel free check out their guides to further your knowledge.

That said, let's take a look at the three kinds of assertion styles bundled into `Chai`:

- *expect*: lets you chain natural language assertions as follows:

```
let lessonTitle = "Testing Smart Contracts with Truffle";
expect(lessonTitle).to.be.a("string");
```

- *should*: allows for similar assertions as `expect` interface, but the chain starts with a `should` property:

```
let lessonTitle = "Testing Smart Contracts with Truffle";
lessonTitle.should.be.a("string");
```

- *assert*: provides a notation similar to that packaged with node.js and includes several additional tests and it's browser compatible:

```
let lessonTitle = "Testing Smart Contracts with Truffle";
assert.typeOf(lessonTitle, "string");
```

```
    //TODO: replace with expect
    expect(result.receipt.status).to.equal(true);
})
```

- Npm install chai
- var expect = require('chai').expect;