06.08

1. 실습

1.1 async & await

1.1.1 async

await은 반드시 async로 되어있는 함수 안에서만 사용해야 합니다.

async 함수가 명시적으로 프로미스를 반환하지 않더라도 async 함수는 암묵적으로 반환값을 resolve하는 프로미스를 반환합니다.

```
async function foo(n){
  return n;
}
```

1.1.2 await

프로미스가 settled상태가 될 때까지 대기하다가 settled 상태가 되면 프로미스가 resolve한 처리 결과를 반환합니다. await 키워드는 반드시 프로미스 앞에서 사용해야 합니다.

```
async function foo(n){
  const result = await promise(n);
  return result;
}
```

프로미스가 settled 상태가 되면 프로미스가 resolve한 처리 결과가 res변수에 할당되는 구조입니다.

1.1.3 사용 시 유의할 점

꼭 순차적으로 작동되어야 하는 것인지 고려해야 합니다.

만약, 이전 비동기의 반환값을 가지고 처리를 해야한다는 로직이면 await을 각각 작성해주지만, 아무 연관성이 없는 비동기들은 await을 사용해줄 필요가 없습니다.

비효율 적인 코드

```
async function foo() {
  const a = await new Promise(resolve => setTimeout(() => resolve(1), 3000));
  const b = await new Promise(resolve => setTimeout(() => resolve(1), 3000));
  const c = await new Promise(resolve => setTimeout(() => resolve(1), 3000));

  console.log([a,b,c]);
}
```

개선안 → 이런식으로 나중에 코드 리뷰 해나가시면 됩니다.

```
async function foo() {
  const res = await Promise.all([
    new Promise(resolve => setTimeout(() => resolve(1), 3000)),
    new Promise(resolve => setTimeout(() => resolve(1), 3000)),
    new Promise(resolve => setTimeout(() => resolve(1), 3000))
  ]);
  console.log([a,b,c]);
}
```

서로 연관성 있는 비동기 로직

```
async function foo() {
  const a = await new Promise(resolve => setTimeout(() => resolve(1), 3000));
  const b = await new Promise(resolve => setTimeout(() => resolve(a+1), 3000));
  const c = await new Promise(resolve => setTimeout(() => resolve(b+1), 3000));

  console.log([a,b,c]);
}
```

1.2 try catch

1.2.1 template

```
try {
    //실행 할 코드(에러가 발생할 가능성이 있는 코드)
} catch (error) {
    //try 코드 블록에서 에러가 발생하면 이 코드 블록의 코드가 실행됩니다.
} finally {
    //에러 발생과 상관없이 반드시 한 번 실행됩니다.
}
```

1.3 Promise 복습

1.3.1 기본편

```
function promise() {
    return new Promise((resolve, reject) => {
        reject(1);
    });
}

promise()
.then((result) => {
  return result+1;
})
.catch(() => {
  //error 처리
})
.finally(() => {
})
```

```
function promise() {
    return new Promise((resolve, reject) => {
        reject(1);
    });
}

promise()
.then((result) => {
  return result+1;
})
.then((result) => {
  return result+1;
})
.then((result) => {
  return result+1;
})
.catch(() => {
  //error 처리
})
.finally(() => {
})
```

1.3.2

```
function promise() {
    return new Promise((resolve, reject) => {
        reject(1);
    });
}

promise()
    .then((result) => {
        console.log("1then", result);
```

```
        return result + 1;
    })
    .catch((error) => {
        console.log("1catch", error);
    })
    .then((result) => {
        console.log("2then", result);
        return result + 1;
    })
    .catch((error) => {
        console.log("2catch", error);
    });
```

1.3.3

```
function promise() {
    return new Promise((resolve, reject) => {
        resolve(1);
    });
}
promise()
    .then((result) => {
        console.log("1then", result);
        return Promise.resolve(result + 1);
    })
    .catch(() => {
        console.log("1catch", error);
    })
    .then((result) => {
        console.log("2then", result);
        return result + 1;
    })
    .catch(() => {
        //error 처리
    });
```

1.3.4

```
function promise() {
    return new Promise((resolve, reject) => {
        resolve(1);
    });
}
promise()
    .then((result) => {
        console.log("1then", result);
        return Promise.reject(result + 1);
    })
    .catch((error) => {
        console.log("1catch", error);
    })
    .then((result) => {
        console.log("2then", result);
        return result + 1;
    })
```

```
    .catch((error) => {
        console.log("2catch", error);
    });
```

1.4 try catch로 바꿔보기

1.4.4 `1.3.4` 코드 바꿔보기

```
function promise() {
    return new Promise((resolve, reject) => {
        resolve(1);
    });
}

async function example() {
    try {
        const result = await promise();
        console.log("1then", result);
        const then2 = await Promise.reject(result + 1);
        try {
            console.log("2then", then2);
        } catch (error) {
            console.log("2catch", error);
        }
    } catch (error) {
        console.log("1catch", error);
    }
}
example();
```

▼ 한 번 실행해보세요!

```
function promise() {
    return new Promise((resolve, reject) => {
        resolve(1);
    });
}

async function example() {
    try {
        const result = await promise();
        console.log("1then", result);
        try {
            const then2 = await Promise.reject(result + 1);
            console.log("2then", then2);
        } catch (error) {
            console.log("2catch", error);
        }
        console.log("에러가 나도 괜찮아요");
    } catch (error) {
        console.log("1catch", error);
    }
}
example();
```

💡 await async는 동기처럼 보이지만, 여전히 비동기라는 것에 유의해주세요!

1.4.5

💡 await async에서 promise 형태로 바꾸는 연습과, promise를 await async로 바꾸는 연습을 자주 해주세요!
await async가 동기처럼 보이다보니, 실행 컨텍스트 측면에서 헷갈리실 수 있습니다.

아래 함수 example가 실제로 실행컨텍스트 상에서 언제 끝나는지 생각해보세요!

```
async function example() {
    console.log('1');
    const result = await promise();
    console.log(result);
}
```

1.5 실습 풀이

▼ async & await 개념

```
//async, await으로 변환하는 방법

// 1. Promise 대신 async로 비동기 처리해 'elice'를 반환하도록 fetchUser 함수를 수정하세요.
async function fetchUser() {
  return 'elice';
}

const user = fetchUser();
user.then(console.log);

// 2. delay 함수를 이용해 getCoffee와 getTea 함수를 작성하세요.
function delay(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function getCoffee() {
  await delay(1000);
  return 'coffee';
}

async function getTea() {
  await delay(1000);
  return 'tea';
}

// 3. 위 두 개의 함수를 사용해서 coffee와 tea를 한번에 반환합니다.
```

```
async function getDrinks() {
  const coffee = await getCoffee();
  const tea = await getTea();
  return `${coffee} and ${tea}`;
}

getDrinks().then(console.log);
```

2. 실습

2.1 실습 풀이

▼ 철인 3종 경기

```
const btn = document.getElementById('btn');
const mes = document.getElementById('message');

async function triathlon() {
  let swimEnd = await exercise(0, 'swim');
  let bicycleEnd = await exercise(swimEnd, 'bicycle');
  let runEnd = await exercise(bicycleEnd, 'run');

  return runEnd;
}

const delay = ms => new Promise(resolve => setTimeout(resolve, ms));

async function exercise(startTime, name) {
  const exerciseInput = document.getElementById(name);
  // 1. 완주에 걸리는 시간을 구하세요.
  const finishTime = parseInt(exerciseInput.value);
  // 2. setTimeout 대신 delay 함수를 사용해 비동기처리를 하세요.
  await delay(finishTime);
  mes.innerText += `${name} finished at ${finishTime + startTime}\n`;
  // 3. 완주한 후의 시간을 반환하세요.

  return startTime + finishTime;
}

btn.addEventListener('click', () => {
  triathlon().then(param => {
    mes.innerText += `total time : ${param}`;
  });
});
```

3. 실습

3.1 실습 풀이
▼ 유저 정보 요청하여 rendering 하기

```
//randomuser.me 라는 API는 무작위로 생성된 사용자의 프로필 이미지를 포함해 디테일한 정보까지 어디서든 요청 가능합니다.

document.getElementById('myBtn').addEventListener('click', getData);
function getData() {
  console.log('test');
  fetch('https://randomuser.me/api/?results=100')
    .then(data => data.json())
    .then(response => {
      let output = '<h2><center>사용자 정보 받기</center></h2>';
      const userList = response.results;

      userList.forEach(lists => {
        output += `
<div class="container">
<div class="card mt-4 bg-light">
<ul class="list-group">
<li class="list-group-item list-group-item-primary"><h2>Name: ${lists.name.first}</h2></li>
<li class="list-group-item"><img src="${lists.picture.large}"></li>
<li class="list-group-item">Phone Number: ${lists.cell}</li>
<li class="list-group-item">DOB: ${lists.dob.date}</li>
<li class="list-group-item">Age: ${lists.registered.age}</li>
<li class="list-group-item">Email ID: ${lists.email}</li>
<li class="list-group-item">City: ${lists.location.city}</li>
<li class="list-group-item">Country: ${lists.location.country}</li>
<li class="list-group-item">PostCode: ${lists.location.postcode}</li>
</ul>
</div>
</div>`;
      });

      document.getElementById('output').innerHTML = output;
    });
}
```

4. 실습

4.1 실습 풀이
▼ fetch 사용하여 API호출하기 3

```
// 자유롭게 코드를 작성하여, 예시 화면이 구현되도록 해 보세요.
const buttonSubmit = document.getElementById('buttonSubmit');
const imageElement = document.getElementById('dogImage');
buttonSubmit.addEventListener('click', getImage);

function getImage() {
  fetch('https://dog.ceo/api/breeds/image/random')
    .then(response => response.json())
    .then(data => {
      const imageUrl = data.message;
      imageElement.src = imageUrl;
```

```
    });
}
```

4.2 api 2개 연속으로 사용해보기

4.2.1

```
const buttonSubmit = document.getElementById('buttonSubmit');
const imageElement = document.getElementById('dogImage');
buttonSubmit.addEventListener('click', getImage);

function getImage() {
  fetch('https://randomuser.me/api/?results=100')
    .then(data => data.json())
    .then(response => {
      console.log(1);
    });

  fetch('https://dog.ceo/api/breeds/image/random')
    .then(response => response.json())
    .then(data => {
      console.log(2);
    });
}
```

4.2.2

```
const buttonSubmit = document.getElementById('buttonSubmit');
const imageElement = document.getElementById('dogImage');
buttonSubmit.addEventListener('click', getImage);

function getImage() {
  fetch('https://randomuser.me/api/?results=100')
    .then(data => data.json())
    .then(response => {
      console.log(1);
      fetch('https://dog.ceo/api/breeds/image/random')
        .then(response => response.json())
        .then(data => {
          console.log(2);
        });
    });
}
```

4.2.3

```
const buttonSubmit = document.getElementById('buttonSubmit');
const imageElement = document.getElementById('dogImage');
buttonSubmit.addEventListener('click', getImage);

async function getImage() {
  await fetch('https://randomuser.me/api/?results=100')
    .then(data => data.json())
    .then(response => {
      console.log(1);
    });
  await fetch('https://dog.ceo/api/breeds/image/random')
    .then(response => response.json())
    .then(data => {
      console.log(2);
    });
}
```

5. 실습

5.1 Error 객체

Error 생성자 함수는 에러를 상세히 설명하는 에러 메세지를 인수로 전달할 수 있습니다.

```
const error = new Error("invalid");
```

앞으로 많이 보일 에러 객체들

SyntaxError	자바스크립트 문법에 맞지 않은 문을 해석할 때 발생
ReferenceError	참조할 수 없는 식별자를 참조했을 때 발생
TypeError	피연산자 또는 인수의 데이터 타입이 유효하지 않을 때 발생
RangeError	숫자값의 허용 범위를 벗어났을 때 발생

5.2 throw 문

Error 생성자 함수로 객체를 만든다고 해도 에러가 발생하는 것은 아닙니다.

```
try{
  throw new Error('error');
} catch(error){
  console.lor(error);
}
```

5.3 실습 풀이

5.3.1 실습 문제 구조

constants.js

애플리케이션에서 변하지 않는 상수값을 따로 지정해놓은 파일입니다.

MACHINE_PREPARE_TIME
GRIND_TIME
BREW_TIME
BEANS_PER_BREW
COFFEE_PER_BREW

api.js

통신(비동기 로직)을 수행하는 함수를 모아놓은 파일입니다.

prepareMachine	커피 머신을 준비하는 기능을 합니다.	MACHINE_PREPARE_TIME 시간이 걸리는 함수입니다.
grindBean	커피콩을 가는 기능을 합니다.	GRIND_TIME 시간이 걸리는 함수입니다.
brewPowder	파우더를 우리는 기능을 합니다.	BREW_TIME 시간이 걸리는 함수입니다.

store.js

커피머신의 정보를 저장하고있는 파일입니다.

initialState	초기 커피머신의 상태를 나타내는 값입니다.
createAction	액션의 포맷에 맞게끔 액션을 반환하는 함수입니다. 액션의 형태는 {type, payload}의 형태로 이루어져있습니다.
updateState	커피머신의 현재 상태에서 diffState의 상태로 변화시켜줍니다.
initializeState	커피머신 객체는 초기값으로 initialBeans, maxMachine의 값을 셋팅할 수 있으며, 그 값으로 state값을 갱신시켜주는 역할을 하는 함수입니다.
update	action명에 따라서 switch문으로 하는 역할을 나눠줍니다. 전반적인 커피머신의 로직을 담당하는 함수입니다. update함수는 action함수의 결과 값으로 state(action을 취하고 나서 커피메이커의 상태)를 반환합니다. 이 함수에 들어가있는 action에 따른 기능 명세는 따로 주황색 표로 표시해두었습니다.
'prepareMachine'	기계를 준비하는 로직입니다. 가지고 있는 기계의 개수를 -1 시켜주는 로직입니다.

'grindBean'	커피콩을 가는 액션을 취하는 로직입니다. //지시사항에 따라 작성해줘야 하는 부분입니다.
'getCoffee'	커피를 주문하는 로직입니다. //지시사항에 따라 작성해줘야 하는 부분입니다.
'addBeans'	커피콩을 추가하는 로직입니다. //지시사항에 따라 작성해줘야 하는 부분입니다.
'brewPowder'	커피파우더를 이용해서 우리는 로직입니다. //지시사항에 따라 작성해줘야 하는 부분입니다.

CoffeeMajer.js

closure	외부에서 사용할 수 있는 기능을 한꺼번에 모아준 것입니다. 파일에 가장 마지막 부분에 export default closure; 라는 구문을 봐주시면 됩니다!
coffeeMaker	커피메이커 객체를 생성할 수 있는 생성자 함수입니다. coffeeMaker의 인자에 대해서는 주황색 표로 표시해두었습니다.
InitialBeans	초기 커피콩의 개수를 셋팅할 수 있습니다. 이 인자를 가지고 store.js의 initializeState 함수를 이용해서 초기 커피애플리케이션의 값을 셋팅합니다.
maxMachine	초기 최대 커피머신 개수를 셋팅할 수 있습니다. 이 인자를 가지고 store.js의 initializeState 함수를 이용해서 초기 커피애플리케이션의 값을 셋팅합니다.
onUpdate	App.js의 코드를 참고해보면, onUpdate는 만약 커피애플리케이션의 재고 값이 바뀌면 해당 화면을 다시 렌더링 시켜주는 기능을 하고 있습니다.
handleUpdate	action 값과 data값을 인자로 받고, store.js의 update를 사용하여 coffee 애플리케이션의 상태값을 갱신시켜주고, 바뀐값에 따라서 onUpdate를 시켜주어 화면을 rendering 해줍니다.
grindBean	grindBean의 액션을 진행합니다. grindBean의 api를 진행합니다.
brewPowder	brewPowder의 액션을 진행합니다. brewPowder의 api를 진행합니다.
prepareMachine	prepareMachine의 액션을 진행합니다. prepareMachine의 api를 진행합니다.
getCoffee	getCoffee 액션을 진행합니다.
addBean	addBean 액션을 진행합니다.
getState	현재 커피앱 애플리케이션의 상태를 반환합니다.

▼ CoffeeMaker 만들기

```
//CoffeeMaker.js
import { COFFEE_PER_BREW, BEANS_PER_BREW } from './constants';
import { createAction, update, initializeState } from './store';
import * as API from './api';

function CoffeeMaker(initialBeans, maxMachine, onUpdate) {
  let state = initializeState(initialBeans, maxMachine);

  const closure = {
    grindBean,
    brewPowder,
    prepareMachine,
    getCoffee,
    getState,
    addBean,
```

```
  };

  function handleUpdate(action, data) {
    state = update(state, createAction(action, data));
    onUpdate(closure);
    return closure;
  }

  function grindBean() {
    handleUpdate('grindBean');
    return API.grindBean().then(() => closure);
  }

  function brewPowder() {
    if (state.beanPowder < BEANS_PER_BREW)
      return Promise.reject(new Error('커피 가루가 부족합니다.'));

    handleUpdate('brewPowder');
    return API.brewPowder().then(() => closure);
  }

  function prepareMachine() {
    if (state.currentMachine <= 0) {
      // currentMachine이 0이하 일 경우 "남은 머신이 없습니다." 라는 에러 메시지와 함께 promise를 reject 합니다.
      return Promise.reject(new Error('남은 머신이 없습니다.'));
    }

    if (state.beans < BEANS_PER_BREW) {
      return Promise.reject('커피 원두가 부족합니다.');
    }

    handleUpdate('prepareMachine');
    return API.prepareMachine().then(() => closure);
  }

  function getCoffee() {
    handleUpdate('getCoffee');
    return COFFEE_PER_BREW;
  }

  function getState() {
    return state;
  }

  function addBean(amount) {
    handleUpdate('addBeans', amount);
  }

  return closure;
}

export default CoffeeMaker;
```

```
//store.js
import { BEANS_PER_BREW, COFFEE_PER_BREW } from './constants';

const initialState = {
  beans: 0,
  beanPowder: 0,
  coffee: 0,
  currentMachine: 0,
```

```
};

export const createAction = (type, payload) => ({ type, payload });

const updateState = (state, diffState) => ({ ...state, ...diffState });

export function update(state, action) {
  switch (action.type) {
    // prepareMachine 액션에 대해 새로운 상태를 반환합니다.
    case 'prepareMachine': {
      return updateState(state, {
        currentMachine: state.currentMachine - 1,
      });
    }
    // grindBean, addBeans, brewPowder, getCoffee 액션에 대한 행위를 위의 예시를 참고하여 작성해보세요.
    case 'grindBean': {
      return updateState(state, {
        beans: state.beans - BEANS_PER_BREW,
        beanPowder: state.beanPowder + BEANS_PER_BREW,
      });
    }

    case 'addBeans': {
      return updateState(state, {
        beans: state.beans + action.payload,
      });
    }

    case 'brewPowder': {
      return updateState(state, {
        coffee: state.coffee + COFFEE_PER_BREW,
        beanPowder: state.beanPowder - BEANS_PER_BREW,
      });
    }
    case 'getCoffee': {
      return updateState(state, {
        currentMachine: state.currentMachine + 1,
      });
    }

    default:
      return state;
  }
}

export function initializeState(initialBeans, maxMachine) {
  return {
    ...initialState,
    beans: initialBeans,
    currentMachine: maxMachine,
  };
}
```