



Portfolio

Presentation

# calculator & graph

계산기 & 그래프



# 목차

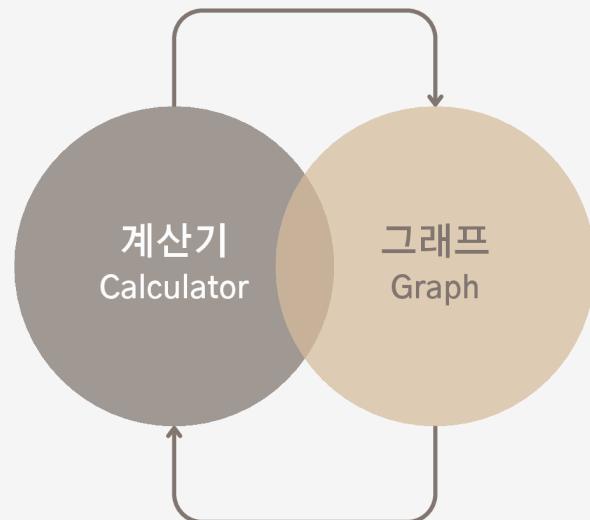
1. 프로젝트 개요	<u>3 P</u>
2. 프로젝트 목적	<u>5 P</u>
3. 디자인	<u>10 P</u>
4. 코드 분석	<u>13 P</u>
5. 애플리케이션	<u>31 P</u>
6. 리뷰	<u>32 P</u>

# 1. 프로젝트 개요

CALCULATOR  
& GRAPH



## 프로젝트 소개



사칙연산 등 기본적인 연산 작동과  
결과값에 세 자리 단위마다 쉼표(,) 표기,  
소수점 표기 포맷(자릿수) 등을 설정 할 수 있는 **계산기**를 구현하고,

데카르트 좌표계를 통해 **그래프**를 좌표 평면 위에 나타낸다.  
계수를 임의의 입력값으로 받아,  
일차함수, 이차함수, 삼차함수, 원의 형태로 변환하여 표시하고  
이동, 확대 / 축소, 초기화, 원점이동 등의 기능을 구현한다.

# 1. 프로젝트 개요

CALCULATOR  
& GRAPH



## 프로젝트 일정



### 기획 단계의 어려움



지난 포트폴리오에서는 기획 단계에서 가장 어려움을 겪었다. 그러나 이번 프로젝트에서는 필요한 기능을 신속하게 이해하고 진행하는 데 주력하였다.

### 마감기한 준수 목표



지난 작업에서 마감기한을 완벽하게 지키지 못한 점이 아쉬웠다. 이에 따라 이번 프로젝트에서는 마감기한 준수를 가장 중요한 목표로 설정하였다.

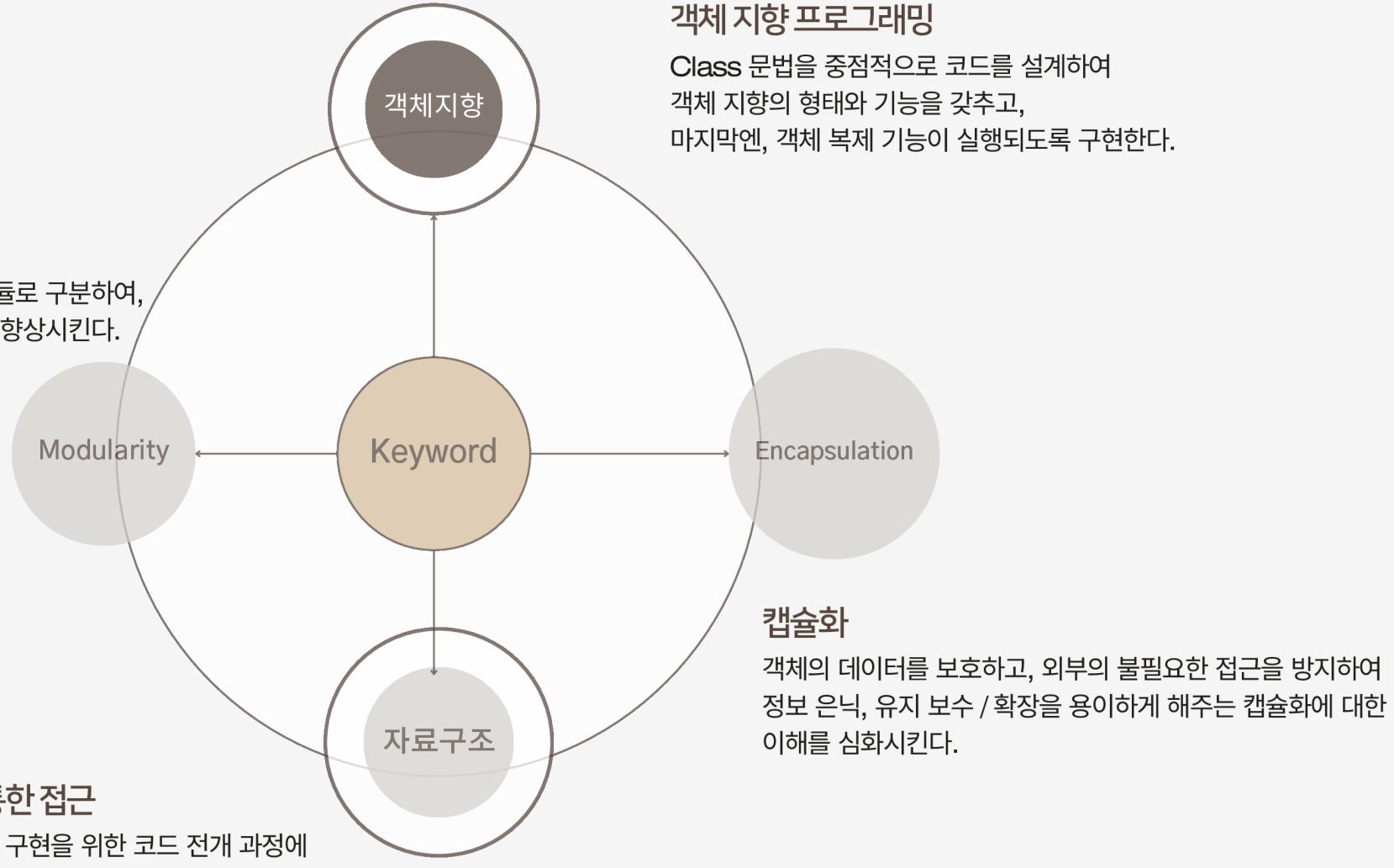
## 2. 프로젝트 목적

CALCULATOR  
& GRAPH



### 모듈화

데이터 처리, 시각화, 제어를 독립적인 모듈로 구분하여, 코드의 유지 보수성, 효율성, 유연성을 향상시킨다.

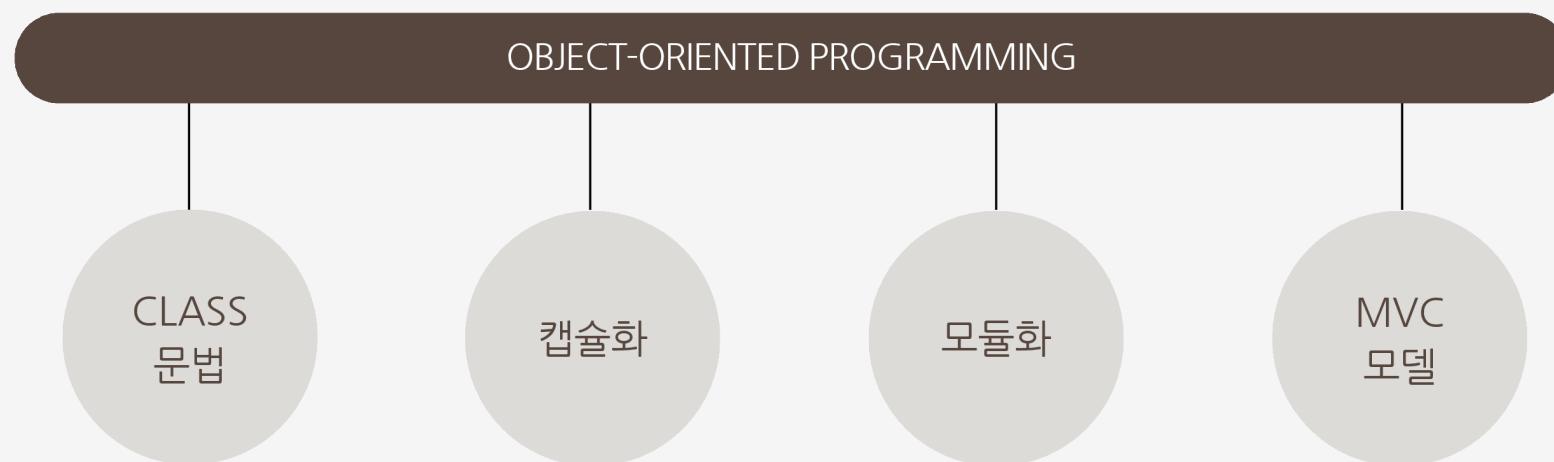


## 2. 프로젝트 목적

CALCULATOR  
& GRAPH



### 객체 지향 프로그래밍



- ✓ 완벽한 객체 지향 프로그래밍을 구현할 수는 없으나,  
프로젝트 진행을 통해 해당 개념에 대한 이해를 높이고 심화시키기  
위해 노력하였다.
- ✓ 설계 단계부터 구조 개선에 대해 고민하였으며,  
재사용성, 유지보수성, 확장성 등 객체 지향 프로그래밍의  
장점을 고려하였다.

## 2. 프로젝트 목적

CALCULATOR  
& GRAPH



### 자료구조의 이해

선형 자료구조 (Linear Structure)

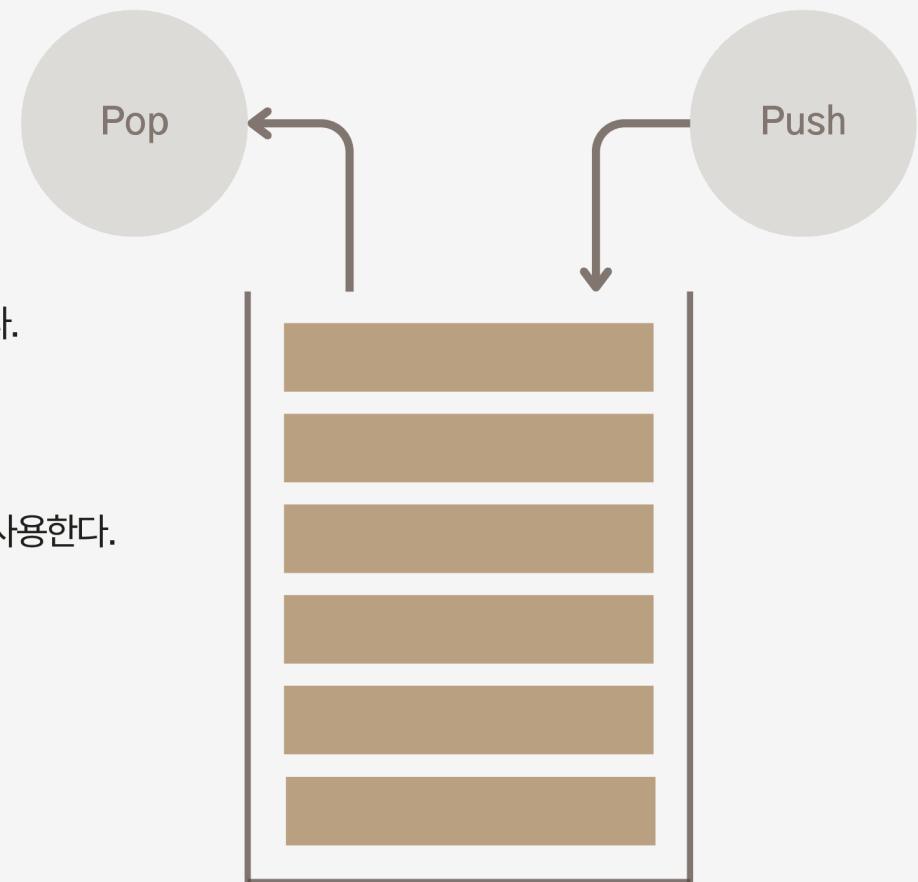
#### 1. Stack

스택은 한쪽 끝에서만 자료를 넣고 빼는 작업이 이루어지는 자료구조이다.

**LIFO (Last In First Out),**

가장 나중에 삽입된 데이터가 먼저 꺼내지는 방식으로 동작한다.

예를 들어, 자료들의 나열 순서를 역순으로 바꾸고 싶은 상황 등에 주로 사용한다.



## 2. 프로젝트 목적

CALCULATOR  
& GRAPH



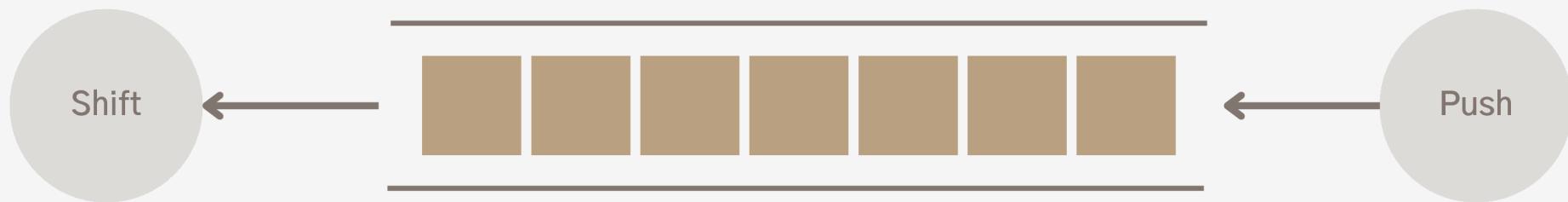
### 자료구조의 이해

선형 자료구조 (Linear Structure)

#### 2. Queue

한쪽에서만 데이터의 삽입 삭제가 이루어졌던 스택과 달리,  
큐는 양쪽 끝에서 데이터의 삽입과 삭제가 각각 이루어진다.

**FIFO (First In First Out)**, 가장 먼저 추가된 데이터가 가장 먼저 꺼내지는 방식으로 동작한다.  
대기열과 관련된 프로세스, 순서대로 일을 처리하는 상황에 주로 사용한다.



## 2. 프로젝트 목적

CALCULATOR  
& GRAPH



### 자료구조의 이해

선형 자료구조 (Linear Structure)

#### 3. Deque



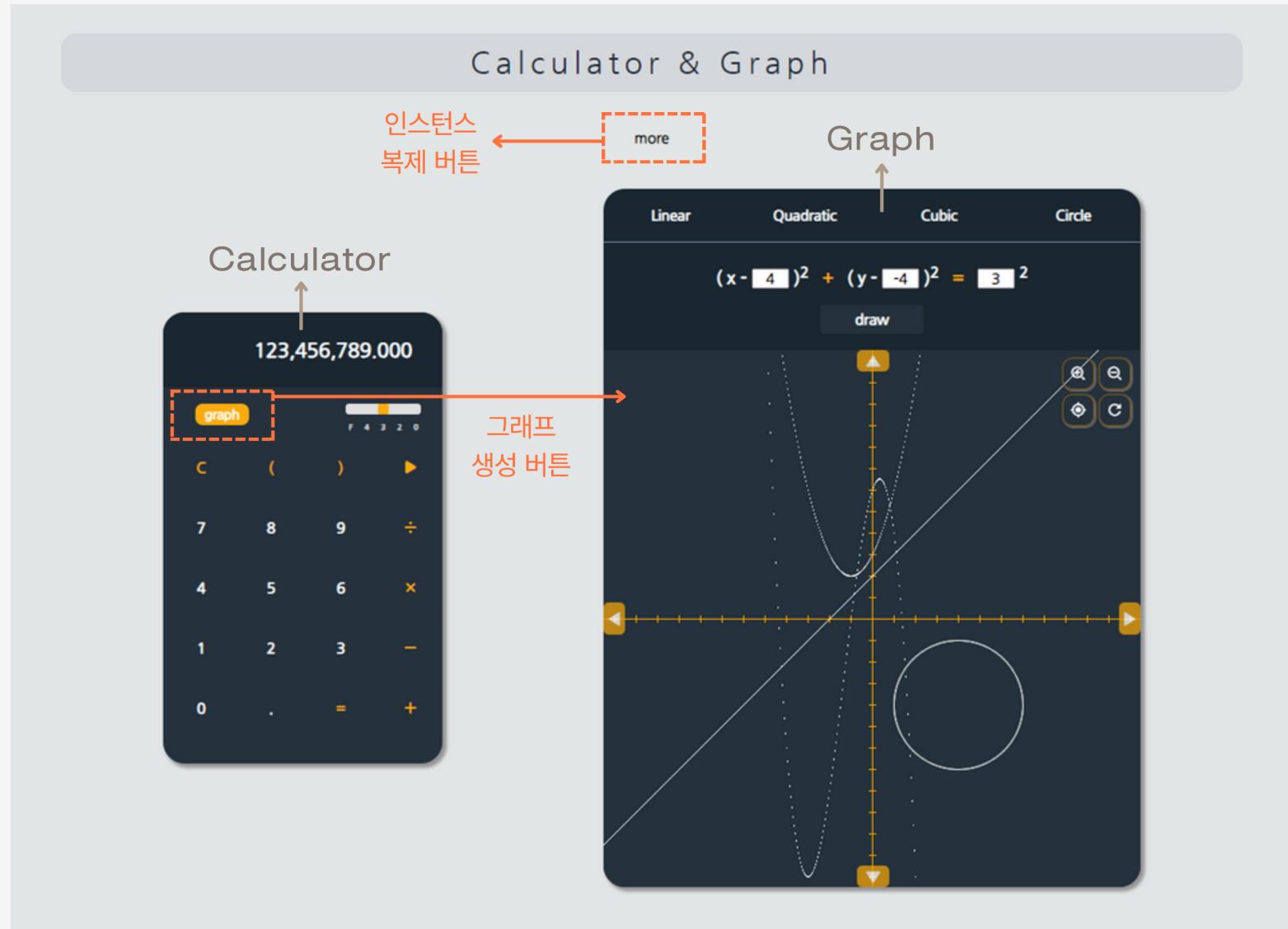
회문(Palindrome) 판별 등에 유용하게 사용될 수 있다.

### 3. 디자인

CALCULATOR  
& GRAPH



전체 디자인



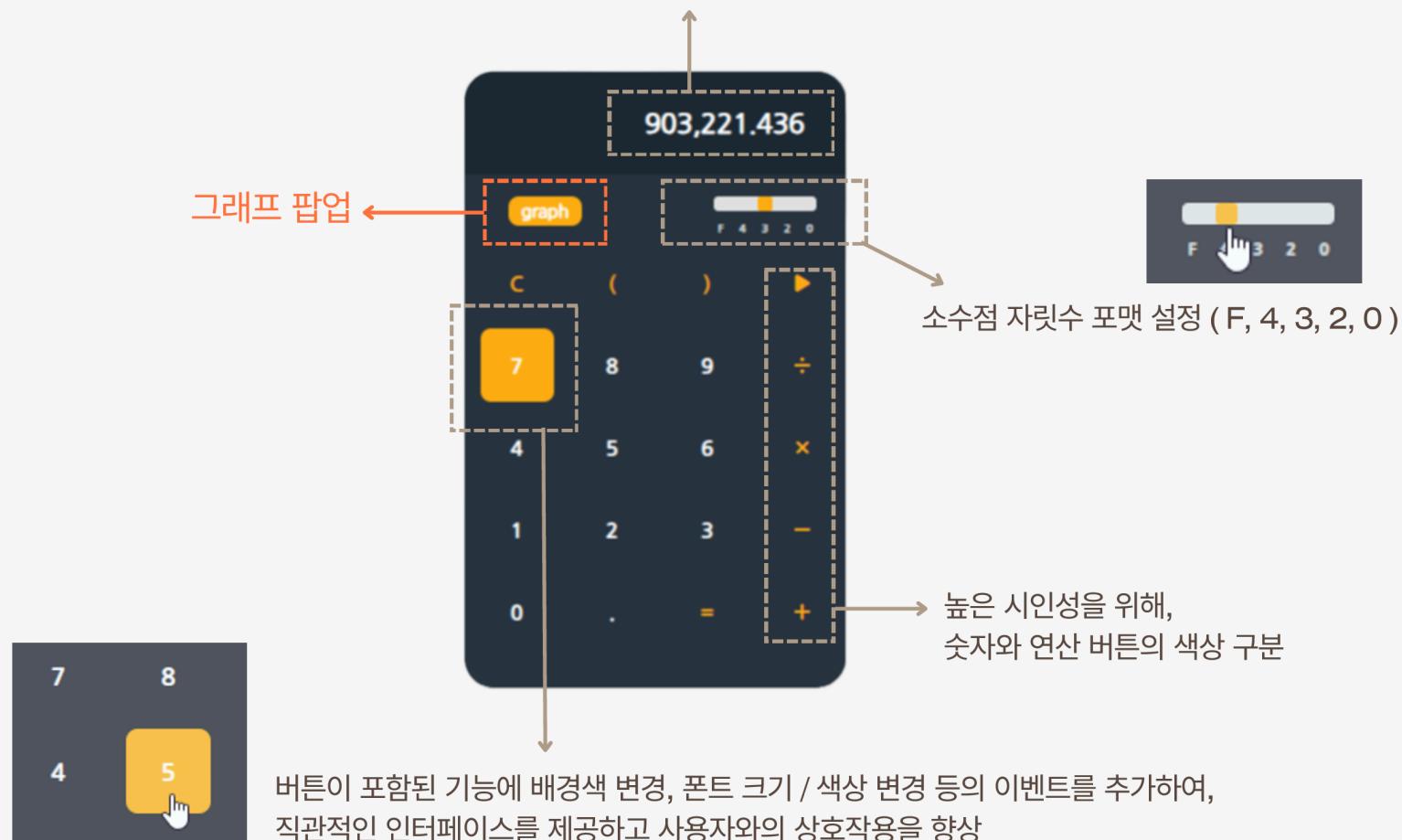
### 3. 디자인

CALCULATOR  
& GRAPH



#### Calculator

가독성을 높이기 위해 세자리 단위마다 콤마( , )를 표시

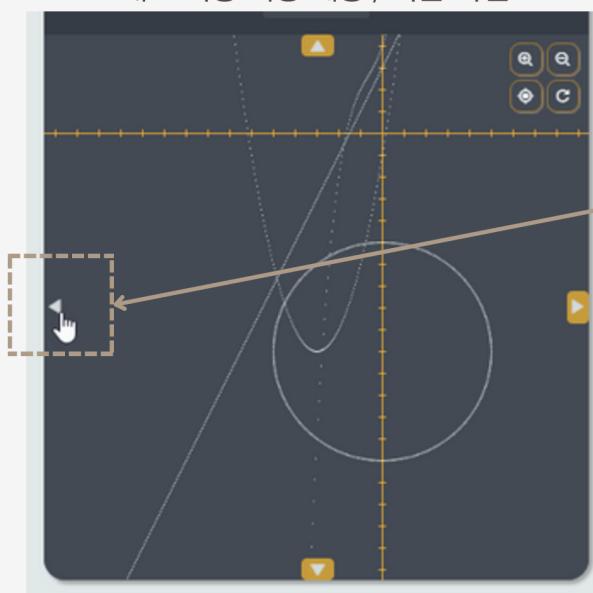


### 3. 디자인

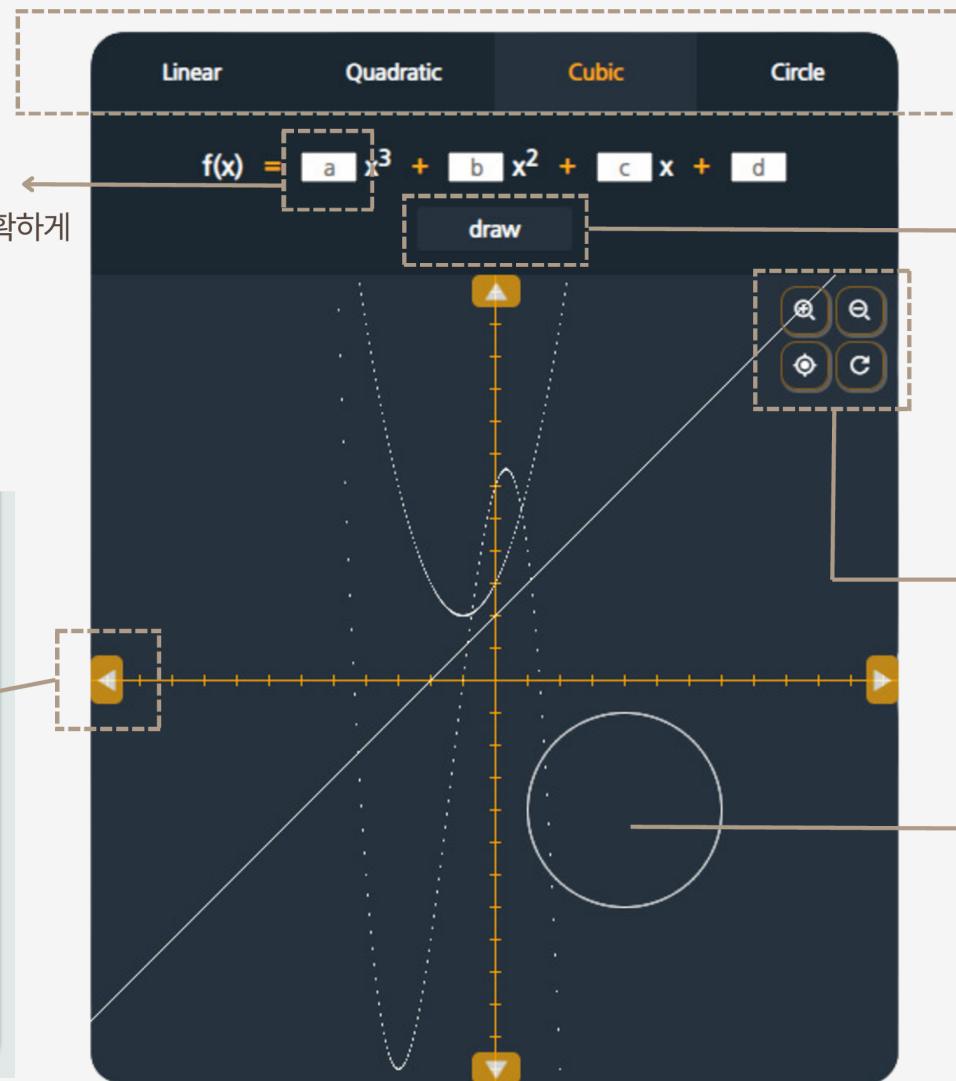
CALCULATOR  
& GRAPH



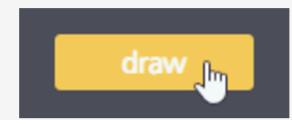
#### Graph



입력 필드에 기본 표시 텍스트를  
삽입하여, 입력해야할 내용을 명확하게  
안내하고 이해를 도움



메뉴탭의 형태와  
색전환 버튼 이벤트를 통해  
직관적 인터페이스 제공



그래프 출력 버튼



확대 / 축소 기능  
원점이동 / 그래프 초기화 기능

배경색과 대비되는  
그래프색 / x, y축의 색상을  
따로 설정하여 시인성 확보

## 4. 코드 분석

CALCULATOR  
& GRAPH



### 전체 코드구조 프리뷰

```

class Calculation {          계산기&그래프 구현 및 기능을 위한 class
>     constructor(id) {
>         initCal(targetDom) {
>             시각화 영역 ( displayData(getData) {
>                 displayResult(resultData, getPointIndex) {
>                     자료처리 영역 — inputData() {
>                         operateCanvas() {
>                             제어 영역 — control(targetDom) {
>                         }
>
class Duplicater {          위의 class를 통해 인스턴스를 자동으로 복제하기 위한 class
>     constructor(id) {
>         duplicate() {
>
const main = () => {      프로그램 실행 함수
>     const calGraph = new Duplicater('calGraph');
>     calGraph.duplicate();

```

## 4. 코드 분석

CALCULATOR  
& GRAPH



### 세부 코드구조

```

class Calculation {    Class 문법을 활용하여 객체 지향 구조를 설계
  constructor(id) {
    this.id = id;
    this.calString = '';
  }
  initCal(targetDom) {    계산기 생성/초기화 method
    const calculatorHTML =
      <section id="${this.id}" class="mainBox">
        <div id="${this.id}Result" class="resultArea">0</div>
        <div class="upperBox">
          <div class="canvasBtnBox">
            <button id="${this.id}canvasBtn" class="canvasBtn">graph</button>
          </div>
          <div class="decimalPoint">
            <input id="${this.id}inputPoint" class="inputPoint" type="range">
            <div class="inputValue"></div>
          </div>
        </div>
      </section>    캡슐화를 통해 객체 지향 추구
    ;
    const calculatorCSS =
    const canvasHTML =
    const canvasCSS =
      targetDom.innerHTML += calculatorHTML + canvasHTML;
      targetDom.innerHTML += calculatorCSS + canvasCSS;
  }
}

```

- □ ×

클래스 내부에서 컴포넌트를 설정하고 DOM에 HTML과 CSS를 삽입,  
명확한 변수명을 통해 코드의 가독성 및 이해도 향상

## 4. 코드 분석

CALCULATOR  
& GRAPH



```



데이터 처리를 위해,  
DEQUE 자료구조를 사용.  
(계산기의 가장 기본적인 기능인  
연산식을 수행하거나 수정할 때,  
데이터의 양쪽 끝에서  
쉽게 접근이 가능하다고 판단)


```

## 4. 코드 분석

CALCULATOR  
& GRAPH



```
let makeDeliverValue = () => {
  let deliverValue = '';
  bundle.countLength();
  for (let i = 0; i < bundle.length; i++) {
    let tempValue = '';
    tempValue = bundle.shiftItem();
    deliverValue += tempValue;
    bundle.pushItem(tempValue);
  }
  return deliverValue;
}
```

DEQUE를 통해 만들어진 데이터 자체를  
시각화 영역에 직접 전달하지 않고,  
새로운 변수(데이터)로 생성(복제)하여 전달

```
let pointIndex = 0;

document.getElementById(`#${this.id}inputPoint`).addEventListener('change', () => {
  pointIndex = document.getElementById(this.id + 'inputPoint').value;
})
```

소수점 포맷(자릿수) 설정 기능을 위한 INDEX 전달값 생성

## 4. 코드 분석

CALCULATOR  
& GRAPH



```

document.getElementById(this.id).addEventListener('click', (e) => {
    let event = e ? e : event;
    if (e.target.dataset.value != null) {
        const btn = e.target.dataset.value;
        switch (btn) {
            case 'C':
                bundle.storage = [];
                this.displayData('0');
                break;

            case '=':
                this.displayData('');
                this.displayResult(makeDeliverValue(), pointIndex);
                bundle.storage = [];
                break;

            case '>':
                bundle.popItem();
                this.displayData(makeDeliverValue());
                break;

            default:
                bundle.pushItem(e.target.dataset.value);
                this.displayData(makeDeliverValue());
                break;
        }
    }
})

```

switch문을 활용한 계산기 버튼 클릭 이벤트

makeDeliverValue() 함수를 통해  
DEQUE의 복제된 데이터를 시각화 영역에  
인자로 전달

## 4. 코드 분석

CALCULATOR  
& GRAPH

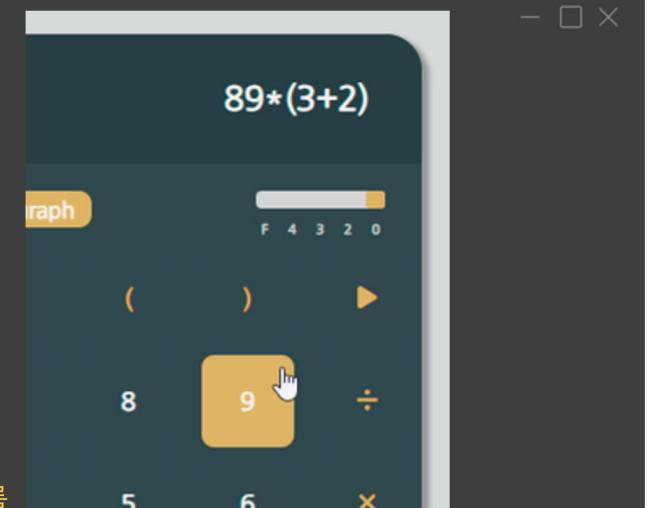


```

displayData(getData) { 시각화 영역① 입력값 표시 method
    this.calString = '';
    if (Array.isArray(getData)) {
        getData.forEach((v) => {
            this.calString += v;
        })
    } else {
        this.calString = getData;
    }
    const displayValue = this.calString;
    document.getElementById(this.id + "Result").innerHTML = displayValue;
}

```

매개변수로 전달받은 데이터를  
데이터 type에 따라 변환하여 화면에 표시



## 4. 코드 분석

CALCULATOR  
& GRAPH

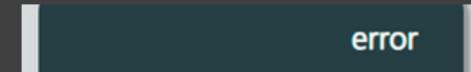
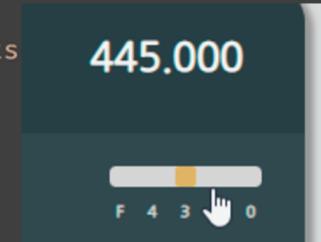


```

displayResult(resultData, getPointIndex) { 시각화 영역② 결과값 표시 method
    const maxPointArray = [16, 4, 3, 2, 0];
    const minPointArray = [0, 4, 3, 2, 0];
    let maxPointValue = maxPointArray[getPointIndex - 1]; 두번째 매개변수로 전달받은 소수점 INDEX 값을 통해
    let minPointValue = minPointArray[getPointIndex - 1]; 설정된 포맷에 따라 소수점 자릿수를 표시
    let pointOption = { minimumFractionDigits: minPointValue, maximumFractionDigits: 4 }

    try {
        this.calString = resultData;
        const resultValue = eval(this.calString).toFixed(maxPointValue); /
        const resultValuePoint = parseFloat(resultValue).toLocaleString('ko-KR', pointOption);
        document.getElementById(this.id + "Result").innerHTML = resultValuePoint;
        this.calString = resultValuePoint; 첫번째 매개변수로 전달받은 데이터를
    } catch (err) { 연산처리하고, 해당 결과값을 화면에 표시
        document.getElementById(this.id + "Result").innerHTML = "error"; 연산 불가한 경우 예외처리
    }
}

```



## 4. 코드 분석

CALCULATOR  
& GRAPH



- □ ×

```
operateCanvas() {  
    class Canvas {  
        constructor(id) {  
            this.id = id;  
            this.canvas = null;  
            this.pen = null;  
            this.moveWidth = 0;  
            this.moveHeight = 0;  
            this.scaleFactor = 20;  
            this.scalePrint = 1 / this.scaleFactor;  
        }  
    }  
}
```

Graph 기능 구현을 위한 method  
중첩 class 활용

## 4. 코드 분석

CALCULATOR  
& GRAPH



```
initCan() {    그래프 영역(캔버스) 생성/초기화 method
    document.getElementById(this.id + 'canvasBtn').addEventListener('click', () => {
        const mainCanvasBox = document.getElementById(this.id + 'mainCanvasBox');
        let opa = mainCanvasBox.style.opacity;
        let topa = mainCanvasBox.style.top;
        let topdis = mainCanvasBox.style.display;

        if (topdis == 'block') {
            mainCanvasBox.style.top = '-1000px';
            mainCanvasBox.style.display = 'none';
        } else {
            mainCanvasBox.style.top = '0px';
            mainCanvasBox.style.display = 'block'; }

        setTimeout(() => {
            if (topdis == 'block') {
                mainCanvasBox.style.opacity = '0';
            } else {
                mainCanvasBox.style.opacity = '1';
            }
        }, 300)
    });
}
```

## 4. 코드 분석

CALCULATOR  
& GRAPH



```
menuSwitch() { 그레프 메뉴탭 선택(변경) 이벤트 method
    document.getElementById(this.id + 'functionBox').addEventListener('click', (e) => {
        const funcName = e.target.id;

        document.querySelectorAll(`.${this.id}functionInput`).forEach(input => {
            input.style.display = "none";
        })
        switch (funcName) {
            case `${this.id}f_name`:
                document.getElementById(this.id + 'f_functionInput').style.display = "block";
                break;

            case `${this.id}s_name`:
                document.getElementById(this.id + 's_functionInput').style.display = "block";
                break;

            case `${this.id}t_name`:
                document.getElementById(this.id + 't_functionInput').style.display = "block";
                break;

            case `${this.id}c_name`:
                document.getElementById(this.id + 'c_functionInput').style.display = "block";
                break;
        }
    });
}
```



## 4. 코드 분석

CALCULATOR  
& GRAPH



```
printDot(x, y) {    캔버스 기본 좌표 출력 함수
    this.pen.fillStyle = 'orange';
    this.pen.fillRect(x, y, 1, 1);
}

modiDot(x, y) {    데카르트 좌표계에 따라 수정된 좌표 출력 함수
    this.pen.fillStyle = 'white';
    this.pen.fillRect(((this.canvas.width / 2) + this.moveWidth) + x * (this.scaleFactor),
                    ((this.canvas.height / 2) + this.moveHeight) - y * (this.scaleFactor), 1, 1);
}

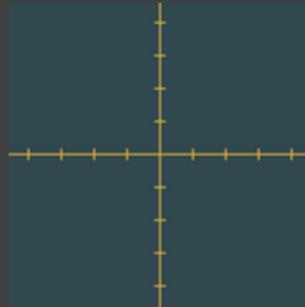
printCross() {    데카르트 좌표계의 x축, y축 출력 함수
    for (let x = 0; x < this.canvas.width; x++) {
        this.printDot(x, ((this.canvas.height) / 2) + this.moveHeight);
    }
    for (let y = 0; y < this.canvas.height; y++) {
        this.printDot(((this.canvas.width) / 2) + this.moveWidth, y);
    }
}
```

## 4. 코드 분석

CALCULATOR  
& GRAPH



```
printUnit() {    // 좌표계 x축, y축 눈금선 출력 함수
    for (let x = this.canvas.width / 2 + this.moveWidth; x <= this.canvas.width; x += this.scaleFactor) {
        for (let y = this.canvas.height / 2 - 3; y <= this.canvas.height / 2 + 3; y++) {
            this.printDot(x, y + this.moveHeight);
        }
    }
    for (let x = this.canvas.width / 2 + this.moveWidth; x >= 0; x -= this.scaleFactor) {
        for (let y = this.canvas.height / 2 - 3; y <= this.canvas.height / 2 + 3; y++) {
            this.printDot(x, y + this.moveHeight);
        }
    }
    for (let y = this.canvas.height / 2 + this.moveHeight; y <= this.canvas.height; y += this.scaleFactor) {
        for (let x = this.canvas.width / 2 - 3; x <= this.canvas.width / 2 + 3; x++) {
            this.printDot(x + this.moveWidth, y);
        }
    }
    for (let y = this.canvas.height / 2 + this.moveHeight; y >= 0; y -= this.scaleFactor) {
        for (let x = this.canvas.width / 2 - 3; x <= this.canvas.width / 2 + 3; x++) {
            this.printDot(x + this.moveWidth, y);
        }
    }
}
```



## 4. 코드 분석

CALCULATOR  
& GRAPH



```

firstGraph() {    일차함수 그래프 출력 함수
  let f_a = Number(eval(document.getElementById(this.id + 'f_a').value));
  let f_b = Number(eval(document.getElementById(this.id + 'f_b').value));

  for (let x = -this.canvas.width / 2; x < this.canvas.width / 2; x = x + this.scalePrint) {
    const y = (f_a * x) + f_b;
    this.modiDot(x, y);
  }
}

이차함수, 삼차함수 생략
...

```

```

circleGraph() {    원 그래프 출력 함수
  let c_a = Number(eval(document.getElementById(this.id + 'c_a').value));
  let c_b = Number(eval(document.getElementById(this.id + 'c_b').value));
  let c_r = Number(eval(document.getElementById(this.id + 'c_r').value));

  for (let i = 0; i <= 360; i++) {
    let radian = i * Math.PI / 180;
    let x = c_r * Math.cos(radian);
    let y = c_r * Math.sin(radian);
    this.modiDot(x + c_a, y + c_b);
  }
}

```

## 4. 코드 분석

CALCULATOR  
& GRAPH



```
updateCanvas() {    그래프 초기화하고 기본요소 출력
    this.pen.clearRect(0, 0, this.canvas.width, this.canvas.height);
    this.printCross();
    this.printUnit();
}

zoom() {    확대/축소 기능 - 버튼 이벤트
    document.getElementById(this.id + 'zoomInBtn').addEventListener('click', () => {
        if (this.scaleFactor <= 75) {
            this.scaleFactor += 5;
            this.updateCanvas();
            this.drawGraphs();
        }
    })
    document.getElementById(this.id + 'zoomOutBtn').addEventListener('click', () => {
        if (this.scaleFactor >= 20) {
            this.scaleFactor -= 5;
            this.updateCanvas();
            this.drawGraphs();
        }
    })
}
```



캔버스(그래프)를 확대/축소하여 출력

## 4. 코드 분석

CALCULATOR  
& GRAPH



```

- □ ×

zeroPoint() { 원점이동 기능 - 버튼 이벤트
    document.getElementById(this.id + 'zeroBtn').addEventListener('click', () => {
        this.moveHeight = 0;
        this.moveWidth = 0;
        this.scaleFactor = 20;
        this.updateCanvas();
        this.drawGraphs();
    })
}

canvasClear() { 캔버스 초기화 기능 - 버튼 이벤트
    document.getElementById(this.id + 'clearBtn').addEventListener('click', () => {
        this.pen.clearRect(0, 0, this.canvas.width, this.canvas.height);
        document.querySelectorAll(`#${this.id}functionInput`).forEach(input => {
            input.value = '';
        })
        this.printCross();
        this.printUnit();
    })
}

```

캔버스 중심을  
(0, 0) 좌표로 이동

그раф를  
초기화하여  
빈 캔버스를 출력

## 4. 코드 분석

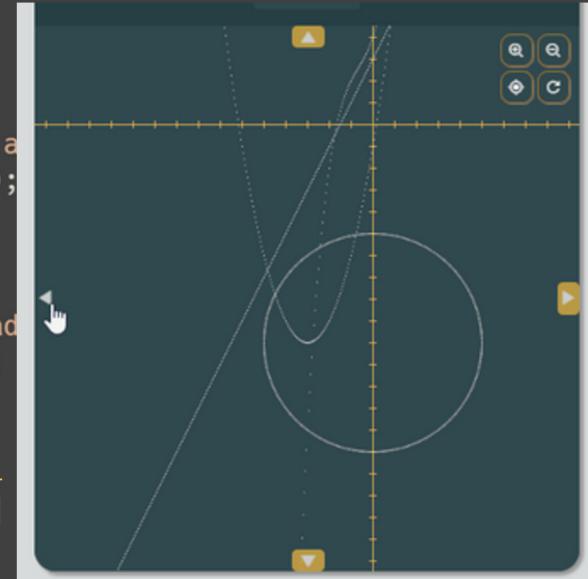
CALCULATOR  
& GRAPH



```

canvasMove( ) { } 좌우상하 이동 기능 - 버튼 이벤트
    document.getElementById(this.id + 'moveBtnTop').addEventListener('click', () => {
        this.moveHeight += 20 * (this.scaleFactor / 20);
        this.updateCanvas();
        this.drawGraphs();
    });
    document.getElementById(this.id + 'moveBtnLeft').addEventListener('click', () => {
        this.moveWidth += 20 * (this.scaleFactor / 20);
        this.updateCanvas();
        this.drawGraphs();
        console.log(this.moveWidth)
    });
    document.getElementById(this.id + 'moveBtnBottom').a
        this.moveHeight -= 20 * (this.scaleFactor / 20);
        this.updateCanvas();
        this.drawGraphs();
    });
    document.getElementById(this.id + 'moveBtnRight').ad
        this.moveWidth -= 20 * (this.scaleFactor / 20);
        this.updateCanvas();
        this.drawGraphs();
    });
}
  
```

캔버스를 원하는 방향으로  
이동하여 보여줄 수 있도록 출력



## 4. 코드 분석

CALCULATOR  
& GRAPH



```

controlCanvas() {
    this.initCan();
    this.menuSwitch();
    this.printCross();
    this.printUnit();
    this.printGraph();
    this.zeroPoint();
    this.zoom();
    this.canvasClear();
    this.canvasMove();
}
const calcCanvas = new Canvas(this.id);
calcCanvas.controlCanvas();

control(targetDom) {
    this.initCal(targetDom);
    this.inputData();
    this.operateCanvas();
}
}

```

제어를 위한 method  
(controller)

데이터 처리, 시각화, 컨트롤러 영역을  
명확하게 구분하여, 각 영역의 역할에 집중하고  
관리/유지보수를 용이하게 함.

특정 부분을 수정하거나,  
새로운 기능을 추가(확장) 할 때,  
다른 부분에 미치는 영향을 최소화 할 수 있다.

## 4. 코드 분석

CALCULATOR  
& GRAPH



```

class Duplicater {    객체 복제 기능을 위한 class 생성
  constructor(id) {
    this.id = id;
  }
  duplicate() {
    let createCount = 1;
    const createCalc = {
      Calculator1: new Calculation('Calculator1')
    };
    createCalc.Calculator1.control(document.getElementById('calcInputArea'));
    document.getElementById('moreBtn').addEventListener("click", () => {
      createCount++;
      const newCalculator = new Calculation(`Calculator${createCount}`);
      createCalc[`Calculator${createCount}`] = newCalculator;
      const newCalcArea = document.createElement('div');
      newCalcArea.id = `calcInputArea${createCount}`;
      newCalcArea.className = 'totalInputArea';
      document.getElementById('mainInputArea').appendChild(newCalcArea);
      newCalculator.control(document.getElementById(`calcInputArea${createCount}`));
    });
  }
}

const main = (() => {
  const calGraph = new Duplicater('calGraph');
  calGraph.duplicate();
})();

```

버튼 클릭 시, 변수명을 생성하고  
해당 변수명의 인스턴스 자동 생성.

프로그램 시작 함수



# 5. 애플리케이션

CALCULATOR  
& GRAPH



## 데스크톱 애플리케이션



ELECTRON을 사용하여 웹 기술 기반의  
크로스 플랫폼 데스크톱 애플리케이션으로 확장

```
const { app, BrowserWindow } = require('electron');
const path = require('path');

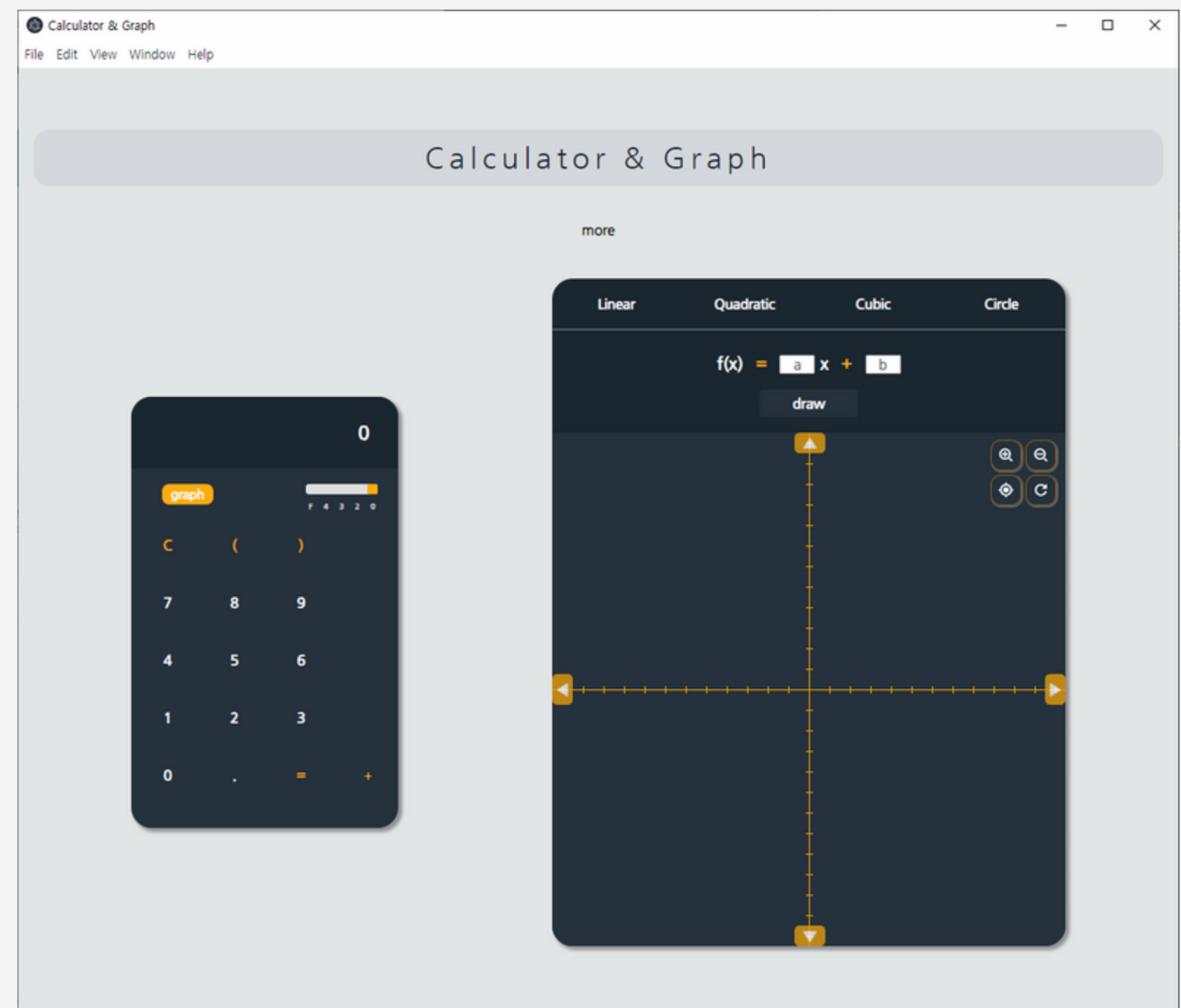
const createWindow = () => {
  const win = new BrowserWindow({
    width: 640,
    height: 480,
    webPreferences:
      { preload: path.join(__dirname, 'preload.js') }
  });

  win.loadFile('index.html');

  app.whenReady().then(() => {
    createWindow();

    app.on('activate', () => {
      if (BrowserWindow.getAllWindows().length === 0)
        createWindow();
    });
  });

  app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') app.quit();
  });
};
```



# 6. 리뷰

CALCULATOR  
& GRAPH



## 프로젝트 목표

이번 프로젝트를 통해 객체 지향 프로그래밍의 여러 개념을 이해하고, 이를 실제로 적용하는 경험을 할 수 있었습니다.

- 먼저, **클래스 문법**을 사용함으로써 객체를 간결하고 직관적으로 정의할 수 있었고, 덕분에 코드의 가독성이 이전 프로젝트에 비해 눈에 띄게 향상된 것을 느꼈습니다.
- 자료 처리, 시각화, 제어 영역을 명확하게 구분하여 설계하는 데 큰 도움이 되었으며, 특히 코드를 수정하거나 확장할 때 클래스 문법의 장점을 실감할 수 있었습니다.
- 또한, **자료 구조**에 대한 기본 원리를 이해하고, 데이터를 효율적으로 다루는 방법에 대해 생각해볼 기회를 가졌습니다.
- 마지막으로, **캡슐화와 모듈화** 덕분에 컴포넌트 간의 구분이 명확해져 코드 구조가 한층 정리된 느낌이 들었고, **MVC 모델**에 대한 기본적인 이해도 높일 수 있었습니다.

## 느낀점

이번 프로젝트는 완벽하다고 말할 수는 없지만,

첫 번째 프로젝트에 비해 진행 과정과 결과 부분에서 큰 만족감을 느꼈습니다.

- **진행 과정** : 모든 단계에서 여전히 어려움이 존재하지만, 특히 기획을 포함한 각 단계를 보다 체계적으로 진행시킬 수 있었습니다.
- **최종 결과** : 문제를 해결하는 과정들에서 많은 경험을 쌓을 수 있었고, 지난 프로젝트보다 코드에 저의 의도를 더 많이 반영시킬 수 있었습니다.
- **개념 적용** : 다양한 개념들을 직접 사용하고 최대한 적용하려는 노력을 통해, 이들 개념이 어떻게 반영되는지를 확인하고 심화시킬 수 있었습니다.



Thank you

감사합니다.