C++_클래스2, 구조체, enum

class, struct, enum

정적 멤버 함수

▪ 정적 멤버 함수

static이 붙은 함수를 정적 멤버 함수라 한다.

정적 멤버함수는 인스턴스를 생성하지 않고, 클래스 이름으로 직접 접근한다.

수학 연산은 상태를 저장할 필요가 없으므로 정적 함수가 적합하다.

또한, static이 붙은 변수를 정적 멤버 변수라 한다.

```
class Card {
private:
    //모든 객체가 공유하는 정적 변수
    static int serialNum;
    string name;    //고객 이름
    int cardNumber; //개별 카드 번호
```

수학 관련 라이브러리

- **<cmath> 헤더파일**

  C++에서 제공되는 수학 관련 함수는 <cmath> 헤더에 정의 되어있으며, C언어의 <math.h>와 호환된다.

```
#include <iostream>
#include <cmath>
#define _USE_MATH_DEFINES
using namespace std;

int main()
{
    cout << "절대값: " << abs(-3) << endl;
    cout << "최대값: " << max(10, 20) << endl;
    cout << "최소값: " << min(10, 20) << endl;
    cout << "거듭제곱: " << pow(2, 3) << endl;

    return 0;
}
```

정적 멤버 함수

- **MyMath 클래스**

```
class MyMath {
public:
    //절대값 계산
    static int abs(int x) {
        return (x < 0) ? -x : x;
    }

    //최대값 계산
    static int max(int x, int y) {
        return (x > y) ? x : y;
    }

    //최소값 계산
    static int min(int x, int y) {
        return (x < y) ? x : y;
    }
};
```

정적 멤버 함수

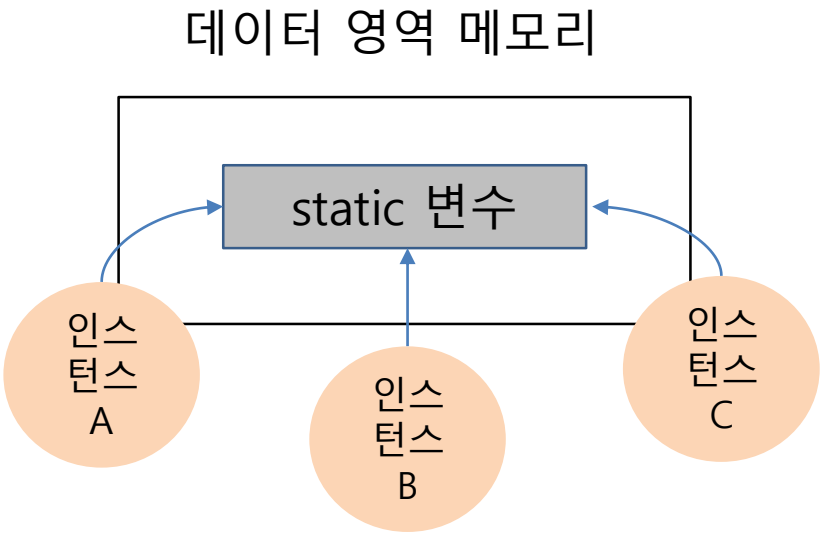
- **MyMath 클래스 테스트**

```
int main()
{
    //객체(인스턴스)를 생성하지 않음
    /*Math math1;
    cout << math1.abs(-3) << endl;*/

    //클래스 이름으로 직접 접근(범위 연산자)
    cout << "-3의 절대값: " << Math::abs(-3) << endl;
    cout << "10과 20중 큰수: " << Math::max(10, 20) << endl;
    cout << "10과 20중 작은수: " << Math::min(10, 20) << endl;

    return 0;
}
```

```
-3의 절대값: 3
10과 20중 큰수: 20
10과 20중 작은수: 10
```

정적 멤버 변수(static)

- **정적 멤버 변수의 정의와 사용 방법**
  - 정적 멤버변수란 static 이 붙은 멤버 변수이다. 클래스 외부에서 초기화 필요
  - 공유 자원 관리: 모든 객체가 동일한 기준값을 참조 해야할때(예: 카드번호, ID 생성기)



static 예약어

```
static int serialNum=1000;
```

정적 멤버 변수(static)

- **Card.h**

```
#ifndef CARD_H
#define CARD_H

#include <iostream>
using namespace std;

class Card {
    static int serialNum; //카드번호 생성을 위한 기준번호
    string name;      //고객 이름
    int cardNumber;   //카드 번호

public:
    Card();  //기본 생성자
    Card(string name);

    string getName();
    int getCardNumber();
};
#endif
```

정적 멤버 변수(static)

- **Card.cpp**

```
#include "Card.h"

//정적 변수는 클래스 외부에서 한번만 정의함
int Card::serialNum = 1000;
/*
Card::Card() {
    serialNum++;
    cardNumber = serialNum;
}
Card::Card(string name) {
    serialNum++;
    cardNumber = serialNum;
    this->name = name;
}
*/

//생성자 초기화 목록 사용
Card::Card() : cardNumber(++serialNum) {}
Card::Card(string name) : name(name), cardNumber(++serialNum) {}

string Card::getName() { return name; }
int Card::getCardNumber() { return cardNumber; }
```

정적 멤버 변수(static)

- **CardMain.cpp**

```
#include "Card.h"

int main()
{
    Card card1("신유진");
    Card card2("이정우");
    Card card3("김선화");

    cout << "카드 번호: " << card1.getCardNumber() << endl;
    cout << "카드 번호: " << card2.getCardNumber() << endl;
    cout << "카드 번호: " << card3.getCardNumber() << endl;

    return 0;
}
```

정적 멤버 변수(static)

- **CardMain.cpp**

```
const int SIZE = 3;

/*Card cardList[SIZE] = {
    Card("신유진"),
    Card("우상영"),
    Card("김선화")
};*/

//사용자 입력
Card cardList[SIZE]; //기본 생성자로 객체 배열 생성

for (int i = 0; i < SIZE; i++) {
    string name; //고객 이름
    cout << i + 1 << "번째 고객 이름 입력: ";
    getline(cin, name);
    cardList[i] = Card(name);
}

for (int i = 0; i < SIZE; i++) {
    cout << "카드번호: " << cardList[i].getCardNumber() << endl;
}
```

객체의 동적 생성 및 반환

- 객체의 동적 메모리 할당
  - **프로그램 실행 중에 필요한 메모리의 크기를 결정**
  - 시스템은 **힙(heap)**이라는 공간을 관리하고 있는데, 프로그램에서 요청하는 공간을 할당하여 시작 주소를 알려준다.
  - 할당된 시작 주소는 반드시 어딘가에 저장되어야 하고 이때 포인터가 사용됨
  - 할당시 **new** , 해제시 **delete** 사용

    ▪ **동적 객체 생성**

    ```
    Car* car1 = new Car()
    ```

    ▪ **동적 객체 반환**

    ```
    delete car1;
    ```

객체의 동적 생성 및 반환

- **Car.h**

```
#ifndef CAR_H
#define CAR_H

#include <iostream>
using namespace std;

class Car {
private:
    string model; //모델 이름
    int year;  //연식

public:
    //생성자 초기화는 선언부에만 가능(기본생성자 포함)
    Car(string model = "", int year = 0);

    void setModel(string model);
    void setYear(int year);

    void carInfo();
};
#endif
```

객체의 동적 생성 및 반환

- **Car.cpp**

```
#include <iostream>
#include "Car.h"

//생성자 - 초기화 목록
Car::Car(string model, int year) :
            model(model), year(year) {}

void Car::setModel(string model) {
    this->model = model;
}

void Car::setYear(int year) {
    this->year = year;
}

void Car::carInfo() {
    cout << "모델명: " << model << endl;
    cout << "연식: " << year << endl;
}
```

객체의 동적 생성 및 반환

- **CarMain.cpp**

```
#include "Car.h"

int main()
{
    //기본 생성자로 동적 객체 생성
    Car* car1 = new Car();
    //매개변수가 있는 생성자로 동적 객체 생성
    Car* car2 = new Car("EV6", 2024);

    //차 정보 입력
    car1->setModel("Sonata");
    car1->setYear(2021);

    //차 정보 출력
    car1->carInfo();
    car2->carInfo();

    delete car1; //메모리 반납
    delete car2;

}
```

```
모델명 : Sonata
연식 : 2021
모델명 : EV6
연식 : 2024
```

객체 배열의 동적 생성 및 반환

- **동적 객체 배열 생성**

```
Car* cars = new Car[3]
```

- **동적 객체 배열 반환**

```
delete[ ] cars;
```

객체의 동적 생성 및 반환

- **동적 객체 배열**

```
#include "Car.h"

int main()
{
    const int SIZE = 3;

    //동적 객체 배열 생성 - 기본 생성자
    /*Car* cars = new Car[3];

    cars[0].setModel("Sonata");
    cars[0].setYear(2017);
    cars[1].setModel("Morning");
    cars[1].setYear(2020);
    cars[2].setModel("Ionic6");
    cars[2].setYear(2024);*/
```

객체의 동적 생성 및 반환

- **동적 객체 배열**

```
    Car* cars = new Car[SIZE]{ //생성자
        Car("Sonata", 2017),
        Car("Morning", 2020),
        Car("Ionic6", 2024)
    };

    for (int i = 0; i < SIZE; i++) {
        cars[i].carInfo();
    }

    delete[] cars;
}
```

```
모델명 : Sonata
연식 : 2017
모델명 : Morning
연식 : 2020
모델명 : Ionic6
연식 : 2024
```

C++ 구조체

❖ C++의 구조체(structure)란?

C++에서는 struct도 기본 접근 지정자가 public일 뿐, class와 거의 동일하게 동작합니다.

- 구조체 정의

```
struct 구조체이름{
      멤버 변수;
      생성자
      멤버 함수
};
```

- 객체 생성

```
구조체이름 객체(인스턴스);
```

C++ 구조체

- 학생 구조체 정의 - Student.h

```
#ifndef STUDENT_H
#define STUDENT_H

#include <iostream>
using namespace std;

struct Student {
    string name;
    int grade;
    string address;

    Student(string name, int grade);

    void showInfo();
};
#endif
```

C++ 구조체

- Student.cpp

```cpp
#include "Student.h"

Student::Student(string name, int grade) :
    name(name), grade(grade) {}

void Student::showInfo() {
    cout << "이름: " << name << endl;
    cout << "학년: " << grade << endl;
}
```

C++ 구조체

- StudentMain.cpp

```
#include "Student.h"

int main() {

    Student st1("정우주", 1);
    Student st2("박은하", 3);

    st1.showInfo();
    st2.showInfo();

    return 0;
}
```

열거형 자료형 enum

❖ **enum 자료형**

- enumeration(열거하다)의 영문 약자 키워드로 , 사용자가 직접 정의하여 사용할 수 있는 자료형이다.
- 열거형은 정수형 상수에 이름을 붙여서 코드를 이해하기 쉽게 해줌
- 열거형은 상수 인덱스 번호 0번에서 시작함

```
const int VALUE_A = 1;
const int VALUE_B = 2;
const int VALUE_C = 3;
```

```
enum VALUE{
    VALUE_A = 1,
    VALUE_B
    VALUE_C
}
```

※ 상수의 개수가 많아지면 선언하기에 복잡해짐

열거형 자료형 enum

❖ **enum 자료형**

```
enum VALUE {
    //기본 인덱스는 0부터 시작함
    VALUE_A = 1,
    VALUE_B,
    VALUE_C
};
```

```
int main()
{
    //상수 선언
    /*const int VALUE_A = 1;
    const int VALUE_B = 2;
    const int VALUE_C = 3;

    cout << VALUE_A << endl;
    cout << VALUE_B << endl;
    cout << VALUE_C << endl;*/

    // enum 자료형 사용
    enum VALUE value;
    value = VALUE_C;

    cout << value << endl;

    return 0;
}
```

열거형 자료형 enum

❖ switch ~ case 문에서 사용하기

```
enum 열거형 이름{
    값1 = 초기값,
    값2,
    값3
}
```

→ enum 열거형 이름 **변수 이름**

```
//열거형 상수 정의
enum MEDAL {
    GOLD = 1,
    SILVER,
    BRONZE
};
```

열거형 자료형 enum

❖ switch ~ case 문에서 사용하기

```
//enum MEDAL medal; //선언
//medal = SILVER;   //사용
//int medal = SILVER; //사용 가능

int medal;
cout << "메달 선택(1 ~ 3 입력): ";
cin >> medal;

switch (medal)
{
case GOLD:
    cout << "금메달" << endl;
    break;
case SILVER:
    cout << "은메달" << endl;
    break;
case BRONZE:
    cout << "동메달" << endl;
    break;
default:
    cout << "메달이 없습니다. 다시 입력하세요" << endl;
    break;
}
```

```
메달 선택(1 ~3 입력): 1
금메달
```

벡터(vector)

❖ 벡터(vector)

- vector는 내부에 배열을 가지고 원소를 저장, 삭제, 검색하는 가변 길이 배열을 구현한 클래스이다.
- <vector> 헤더 파일을 include 해야 함

vector 객체 생성

vector <자료형> 객체 이름

삽입 : push_back()

수정 : vi[0] = 3

벡터(vector) --〉 int형

❖ **벡터(vector)**

```
#include <iostream>
#include <vector>  //vector 컨테이너 사용
#include <string>
using namespace std;

int main()
{
    //여러 개의 정수를 저장할 벡터 생성
    vector<int> vec;

    //정수 추가
    vec.push_back(1);
    vec.push_back(2);
    vec.push_back(3);
```

벡터(vector) --〉 int형

❖ 벡터(vector)

```
//리스트의 크기
cout << vec.size() << endl;

//요소 검색
cout << vec[0] << endl;

//2번 인덱스 값 수정
//vec[2] = 10;
vec.at(2) = 10;

//전체 조회
for (int i = 0; i < vec.size(); i++)
{
    cout << vec[i] << " ";
}
```

```
3
1
1 2 10
```

벡터(vector) --〉 string형

❖ 벡터(vector)

```
//여러 개의 문자열을 저장할 벡터 생성
vector<string> list;
string name;

//저장
list.push_back("jerry");
list.push_back("luna");
list.push_back("han");
list.push_back("elsa");

//리스트의 크기
cout << list.size() << endl;

for (int i = 0; i < list.size(); i++)
{
    cout << list[i] << " ";
}
```

벡터(vector) --〉 string형

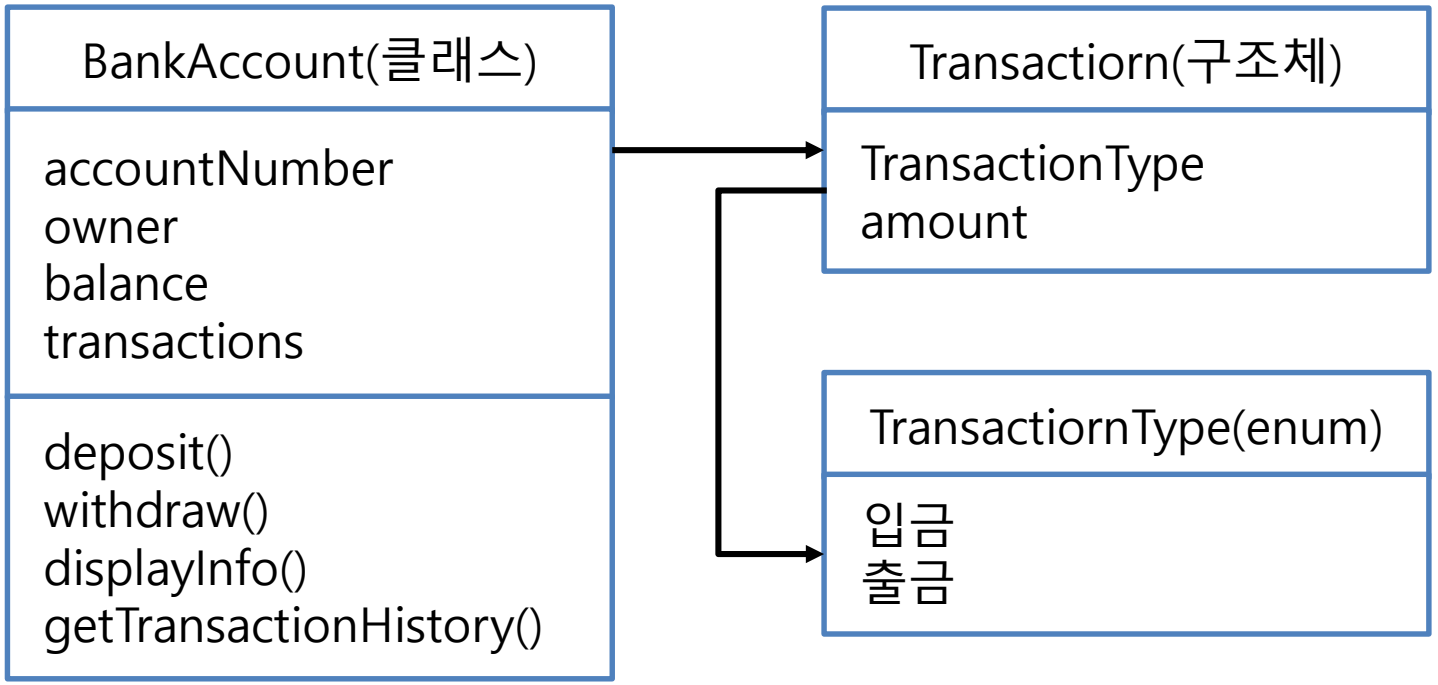
❖ 벡터(vector)

```
//최대값 계산
name = list.at(0);  //최대값으로 설정
for (int i = 0; i < list.size(); i++)
{
    if (list[i] > name)
        name = list[i];
}
cout << "사전에서 가장 뒤에 나오는 이름은 " << name << endl;
```

```
4
jerry luna hangang elsa
==========================
사전에서 가장 뒤에 나오는 이름은 luna
```

은행 거래 프로젝트

❖ 은행 거래 내역



은행 거래 프로젝트

❖ 은행 거래 내역

```
10000원이 입금되었습니다. 현재 잔액: 10000원
30000원이 입금되었습니다. 현재 잔액: 30000원
20000원이 출금되었습니다. 현재 잔액: 10000원
잔액이 부족합니다. 다시 입력하세요.

[계좌 정보]
    계좌 번호: 1001
    예금주: 이우주
    현재 잔고: 10000
---------------------------------
[이우주] 계좌 거래 내역(최근 1건)
 |입금| 10000원
=================================

[계좌 정보]
    계좌 번호: 1002
    예금주: 정은하
    현재 잔고: 10000
---------------------------------
[정은하] 계좌 거래 내역(최근 2건)
 |입금| 30000원
 |출금| 20000원
=================================

[계좌 정보]
    계좌 번호: 1003
    예금주: 한강
    현재 잔고: 100000
---------------------------------
[한강] 계좌 거래 내역(최근 0건)
거래 내역이 없습니다.
```

은행 거래 프로젝트

❖ BankAccount.h

```
#ifndef BANKACCOUNT_H
#define BANKACCOUNT_H

#include <string>
#include <vector>
using namespace std;

enum TransactionType { // 거래 유형 - 열거형
    입금,
    출금
};

struct Transaction { // 거래 구조체
    TransactionType type; //거래 유형
    int amount;           //거래 금액

    Transaction(TransactionType type, int amount);
};
```

은행 거래 프로젝트

❖ BankAccount 클래스

```
class BankAccount { // 은행 계좌 클래스
private:
    int accountNumber;  //계좌 번호
    string owner;       //예금주
    int balance;        //잔고
    vector<Transaction> transactions; //거래

public:
    //생성자 - 목록
    BankAccount(int accountNumber, string owner, int balance = 0);

    void deposit(int amount);          // 입금
    void withdraw(int amount);         // 출금
    void displayInfo();                // 계좌 정보 출력
    void getTransactionHistory();      // 거래 내역 출력
private:
    void addTransaction(TransactionType type, int amount); // 거래 추가
};
#endif
```

은행 거래 프로젝트

❖ BankAccount.cpp

```cpp
#include <iostream>
#include "BankAccount.h"

//Transaction의 생성자
Transaction::Transaction(TransactionType type, int amount) :
    type(type), amount(amount) {}

//BankAccount 생성자 목록
BankAccount::BankAccount(int accountNumber, string owner, int balance)
    : accountNumber(accountNumber), owner(owner), balance(balance) {
}

void BankAccount::deposit(int amount) { //예금 기능
    if (amount < 0) {
        cout << "유효한 금액을 입력하세요.\n";
    }
    else {
        balance += amount;
        cout << amount << "원이 입금되었습니다. 현재 잔액: " << balance << "원\n";
        //addTransaction(TransactionType::입금, amount);
        addTransaction(입금, amount);
    }
}
```

은행 거래 프로젝트

❖ **BankAccount.cpp**

```
void BankAccount::withdraw(int amount) { //출금 기능
    if (amount < 0) {
        cout << "유효한 금액을 입력하세요.\n";
    }
    else if (amount > balance) {
        cout << "잔액이 부족합니다. 다시 입력하세요.\n";
    }
    else {
        balance -= amount;
        cout << amount << "원이 출금되었습니다. 현재 잔액: " << balance << "원\n";
        //addTransaction(TransactionType::출금, amount);
        addTransaction(출금, amount);
    }
}
```

은행 거래 프로젝트

❖ **BankAccount.cpp**

```
//거래 추가
void BankAccount::addTransaction(TransactionType type, int amount) {
    Transaction transaction(type, amount);  //거래 1건 생성
    transactions.push_back(transaction);    //거래를 벡터에 저장
}

//거래 내역 출력
void BankAccount::getTransactionHistory() {
    cout << "[" << owner << "] 계좌 거래 내역(최근 " << transactions.size() << "건)\n";
    if (transactions.empty()) {
        cout << "거래 내역이 없습니다.\n";
        return;
    }
```

은행 거래 프로젝트

❖ BankAccount.cpp

```
    //거래 내역 출력
    for (Transaction& transaction : transactions) {
        cout << " |" << (transaction.type == 입금 ? "입금" : "출금");
        cout << "| " << transaction.amount << "원\n";
    }
    cout << "==================================\n";
}

void BankAccount::displayInfo() { //계좌 정보 출력
    cout << "\n[계좌 정보]\n";
    cout << "    계좌 번호: " << accountNumber << endl;
    cout << "    예금주: " << owner << endl;
    cout << "    현재 잔고: " << balance << endl;
    cout << "----------------------------------\n";
}
```

은행 거래 프로젝트

❖ 은행 거래 메인

```
#include <iostream>
#include <vector>
#include "BankAccount.h"
using namespace std;

int main() {
    //벡터 자료구조 인스턴스 생성
    vector<BankAccount> accounts;

    //은행 계좌 인스턴스 생성 및 저장
    accounts.push_back(BankAccount(1001, "이우주"));
    accounts.push_back(BankAccount(1002, "정은하"));
    accounts.push_back(BankAccount(1003, "한강", 100000));
```

은행 거래 프로젝트

❖ 은행 거래 메인

```
    // 입금
    accounts[0].deposit(10000);
    accounts[1].deposit(30000);

    // 출금
    accounts[1].withdraw(20000);
    accounts[1].withdraw(50000); // 잔액 부족

    // 계좌 정보 및 거래 내역 출력
    for (BankAccount& account : accounts) {
        account.displayInfo();
        account.getTransactionHistory();
    }

    return 0;
}
```