KHLUG

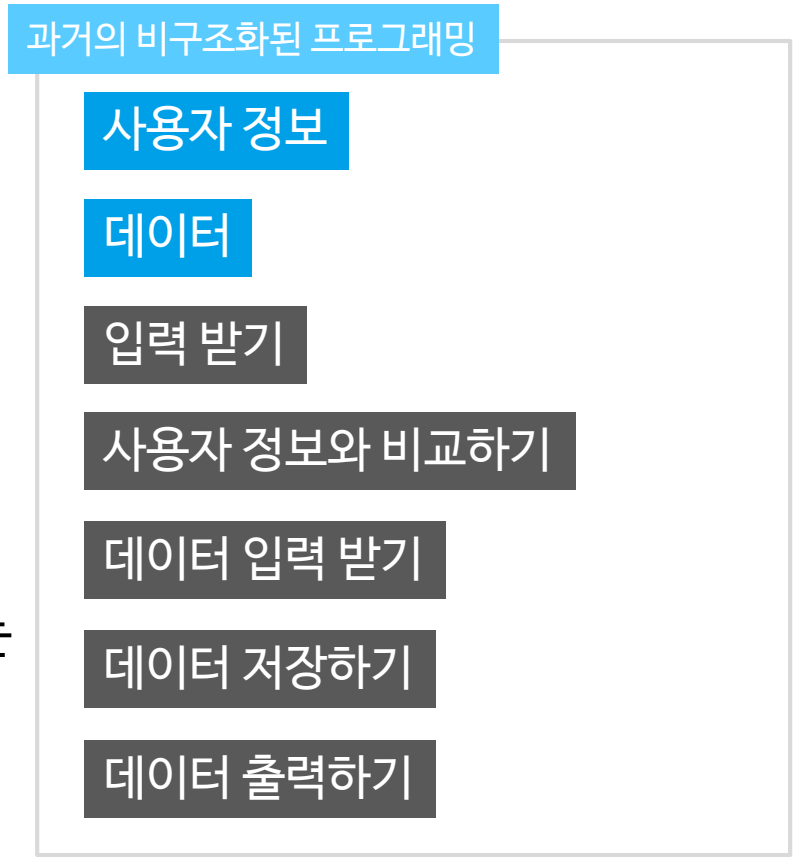
경희대학교 IT 동아리

9. 클래스

프로그래밍 기초 교육

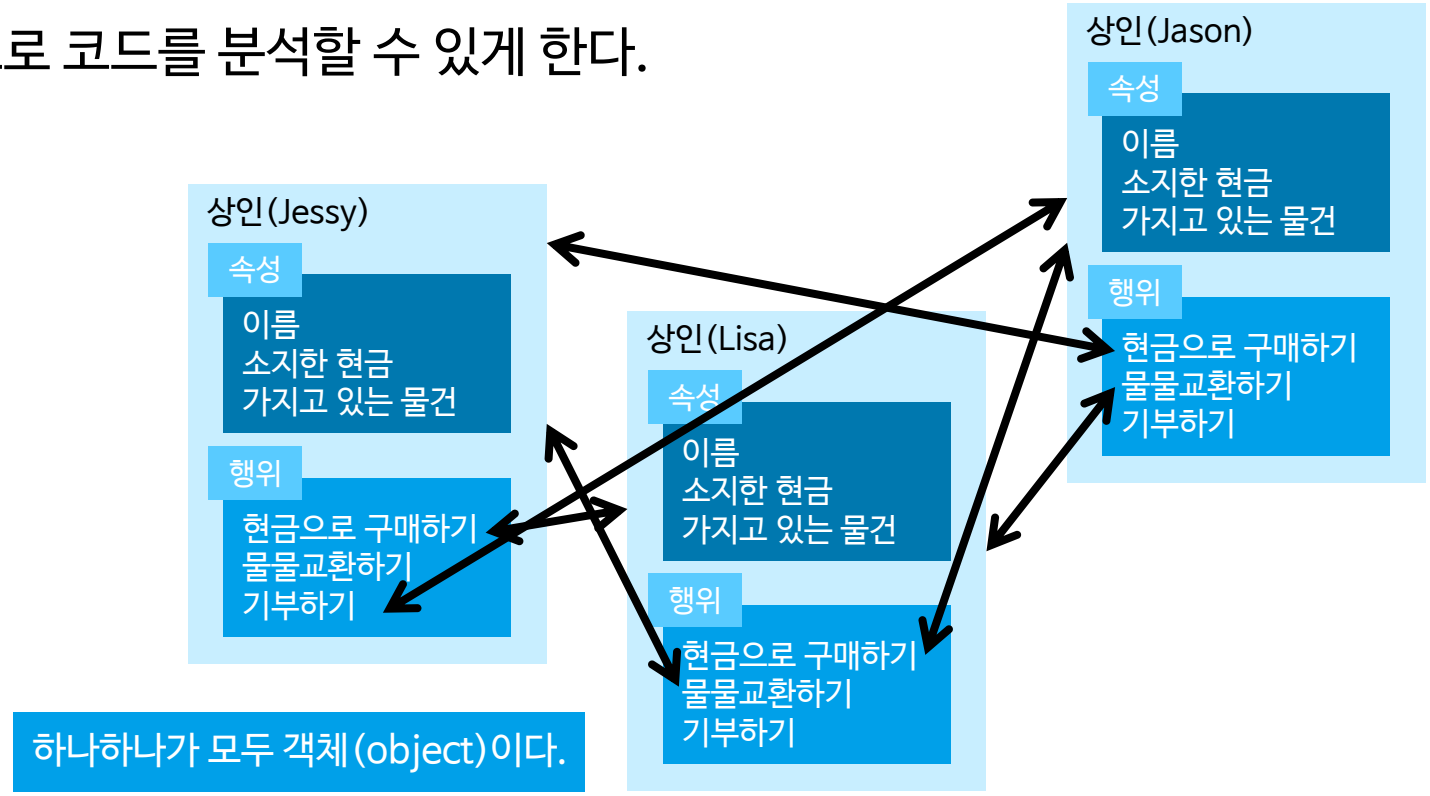
구조화된 프로그래밍

- 갈수록 프로그래밍으로 해결해야 할 문제가 복잡해지고, 더 적은 시간에 더 많은 개발자를 가지고 프로그램을 만들어야 한다.
- 기존에 있던 프로그램을 재활용하여 개발 시간을 단축시켜야 한다.
- 하나의 프로그램에 매달리는 개발자가 많아지면서 서로 소통이 될 수 있도록 공통된 인터페이스가 필요하다.
- 더 많은 사용자에 대응해야 한다.
- 프로그램 자체의 단순한 동작(절차)보다 어떤 종류의 데이터가 어떻게 다루어지는지가 더 중요해졌다.



객체 지향 프로그래밍

- 프로그램은 단순히 명령어들의 모음이 아니라 여러 개의 독립된 단위, "객체"들의 모임으로, 객체들은 서로 메시지를 주고 받고 데이터를 처리한다.
- 소프트웨어 개발과 보수를 간편하게 하며, 직관적으로 코드를 분석할 수 있게 한다.



클래스

- 어떤 종류의 객체를 속성과 행위를 나누어 정의한 것이다.
- 일종의 사용자 정의 자료형이다.
- 속성은 클래스의 멤버 변수(member data)가 되고, 행위는 클래스의 멤버 함수(member function)이 된다.
- 이렇게 묶어 놓은 것을 가지고 클래스의 특징: 캡슐화(Capsulation)이라고 한다.

main() 위에 작성한다.

분자(numerator)와 분모(denominator)를 가지고 있는 분수(Fraction)를 표현할 클래스를 선언한다.

이 클래스에는 분자와 분모를 지정할 store()과 분수 모양대로 출력할 print()라는 멤버 함수가 있다.

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    void store(int numer, int denom);
    void print();
};
```

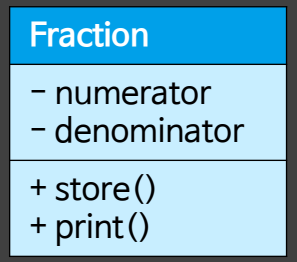
멤버 변수

멤버 변수

멤버 함수

멤버 함수

멤버 함수는 메서드(method)라고도 한다.



클래스는 오른쪽과 같은 다이어그램으로 표현할 수 있다.
위쪽이 멤버 변수, 아래 쪽이 멤버 함수이다.
private인 건 -, public인 건 +를 붙인다.

클래스

- 어떤 종류의 객체를 속성과 행위를 나누어 정의한 것이다.
- 일종의 사용자 정의 자료형이다.
- 속성은 클래스의 멤버 변수(member data)가 되고, 행위는 클래스의 멤버 함수(member function)이 된다.
- 이렇게 묶어 놓은 것을 가지고 클래스의 특징: 캡슐화(Capsulation)이라고 한다.

main() 위에 작성한다.

분자(numerator)와 분모(denominator)를 가지고 있는 분수(Fraction)를 표현할 클래스를 선언한다.

이 클래스에는 분자와 분모를 지정할 store()과 분수 모양대로 출력할 print()라는 멤버 함수가 있다.

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    void store(int numer, int denom);
    void print();
};
```

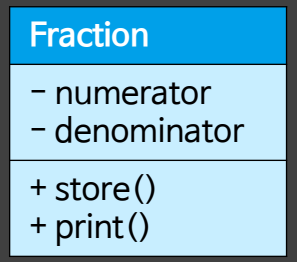
멤버 변수

멤버 변수

멤버 함수

멤버 함수

멤버 함수는 메서드(method)라고도 한다.



클래스는 오른쪽과 같은 다이어그램으로 표현할 수 있다.
위쪽이 멤버 변수, 아래 쪽이 멤버 함수이다.
private인 건 -, public인 건 +를 붙인다.

접근자

- 클래스 외부에서 알 필요가 없는 정보에는 접근을 제한해야 한다. 사용자는 동작하는 것만 보면 된다. 이를 클래스의 특징: 정보 은닉 (Data Hiding)이라고 한다.
- private 멤버는 외부에서 접근할 수 없다.
- public 멤버는 외부에서 접근할 수 있다.

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    void store(int numer, int denom);
    void print();
};
```

일반적으로 멤버 변수는 private: 아래에 두어 외부에서 데이터에 마음대로 접근해 열람하거나 수정할 수 없도록 한다.

외부에서 멤버 함수를 통해 데이터를 조작해야 하므로 멤버 함수는 보통 public: 아래에 두어 외부에서도 접근할 수 있도록 한다.

Java에서는 private 멤버 변수에 접근하기 위해 getter 함수: getNumerator(), getDenominator()와 setter 함수: setNumerator(), setDenominator()를 만들 것을 권장하고, IDE에서 만들어준다.

멤버 함수의 구현

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    void store(int numer, int denom);
    void print();
};
```

여기에선 선언만 하고 구현은 클래스 밖에서 한다.

클래스가 정의된 것 밖에서 함수를 구현하기 때문에 클래스 이름을 써줘야 한다.

```
void Fraction::store
    (int numer, int denom) {
    numerator = numer;
    denominator = denom;
    return;
}
```

같은 클래스의 멤버를 사용하려면 그냥 그 이름을 그대로 적는다.

같은 클래스의 멤버 함수이므로 private인 멤버 변수에 자유롭게 접근할 수 있다.

```
void Fraction::print() {
    cout << numerator << "/"
       << denominator;
    return;
}
```

인스턴스

- 클래스는 그냥 자료의 종류(추상적인 모양)일 뿐이다.
- `int a` 하듯이 실체화해야 하며 이를 인스턴스화 한다고 하고, 실체화된 객체를 인스턴스(instance)라고 한다.

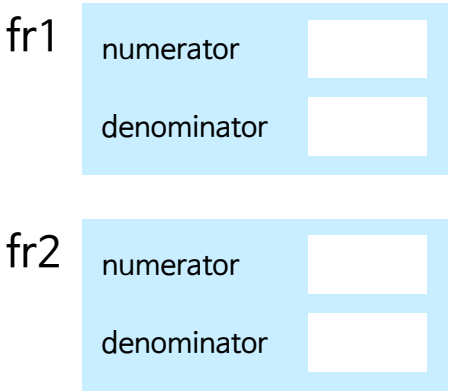
넓은 의미에서 '객체'는 클래스, 변수 등 모든 것을 말하지만
좁은 의미에서 '객체'는 '인스턴스'를 뜻하기도 한다.

클래스가 붕어빵 틀이라면, 클래스로부터
나온 인스턴스는 붕어빵이라고 할 수 있다.

```
int main () {
    Fraction fr1;
    Fraction fr2;

    return 0;
}
```

fr1이라는 Fraction의 인스턴스
fr2라는 Fraction의 인스턴스



멤버 함수 호출

- (객체식별자).(멤버식별자)를 통해 인스턴스의 멤버를 사용할 수 있다.

```
void Fraction::store
    (int numer, int denom) {
    numerator = numer;
    denominator = denom;
    return;
}

void Fraction::print() {
    cout << numerator << "/"
       << denominator;
    return;
}
```

```
int main () {
    Fraction fr;
    fr.store(4,19);
    fr.print();

    /*
        헷갈리지 않기 위해
        아래와 같이 클래스 이름을
        붙일 수도 있다.
        fr.Fraction::store(4,19);
        fr.Fraction::print();
    */
    return 0;
}
```

numerator	4
denominator	19

매개변수로 객체 전달

- 객체 역시 매개변수로 전달될 수 있다.

```
class Fraction {
    // 중략
    void addTo(const Fraction& fr2);
};
```

더하려는 인스턴스는 매개변수로 넘어오는 과정에서 굳이 복제되지 않도록 참조(reference)로 받아오고, 그 내용이 수정되면 안 되므로 const를 붙인다.

```
void Fraction::addTo(const Fraction& fr2)
{
    int numer = (numerator *
        fr2.denominator) +
        (fr2.numerator * denominator);
    int denom = denominator *
        fr2.denominator;

    numerator = numer;
    denominator = denom;

    return;
}
```

```
int main () {
    Fraction fr1;
    fr1.store(4,5);
    Fraction fr2;
    fr2.store(4,9);

    fr1.addTo(fr2);
    fr1.print();

    return 0;
}
```

사용자 정의 헤더

- 클래스 마다 따로 파일을 분리하면 해당 클래스를 보다 쉽게 수정할 수 있다.

main.cpp

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

int main () {
    int numer, denom;
    cout << "분자: ";
    cin >> numer;
    cout << "분모: ";
    cin >> denom;
    Fraction fr;
    fr.store(numer, denom);
    fr.print();
    cout << endl;
    return 0;
}
```

사용자 정의 헤더는 "(큰따옴표)로 묶는다.

"fraction.h"가 인클루드 된 main.cpp에서도 〈iostream〉을 사용할 수 있지만, "fraction.h"가 분리될 때를 대비해 〈iostream〉을 또 인클루드 한다.

cpp 파일과 같은 위치에 만든다.

fraction.h

```
#include <iostream>
using namespace std;

class Fraction {
private:
    int numerator;
    int denominator;
public:
    void store(int numer, int denom);
    void print();
};

void Fraction::store (int numer, int denom) {
    numerator = numer;
    denominator = denom;
    return;
}

void Fraction::print() {
    cout << numerator << "/" << denominator;
    return;
}
```

여기에서 cout을 사용하려면 역시 〈iostream〉을 넣어줘야 한다.

클래스 사용해보기

방금 만들어보았던 Fraction 클래스에 Getter 함수와 Setter 함수를 구현한다.

- Getter: 멤버 변수의 값을 받아내기 위한 함수,
  멤버 변수의 값을 반환한다.
- Setter: 멤버 변수의 값을 지정하기 위한 함수,
  매개변수로 받은 값을 멤버 변수에 넣는다.

힌트

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    // Getter
    int getNumerator();
    int getDenominator();
    // Setter
    void setNumerator(int);
    void setDenominator(int);
}
```

클래스 사용해보기

사용자에게 분자와 분모를 입력 받아 분수를 만들어 출력하는 프로그램을 작성하시오.

Fraction 클래스에 저장된 분수의 값을 기약분수로 만드는 멤버 함수 normalize()를 구현하시오.
분자와 분모의 최대공약수로 각각을 나누면 된다.
예를 들면, 6/8은 6과 8의 최대공약수가 2이므로 분자와 분모를 2로 나누어 3/4가 된다.

```
int gcd(int a, int b) {
    while (1) {
        if (a > b) {
            if (a % b ==0) return b;
            else a = a % b;
        } else {
            if (b % a ==0) return a;
            else b = b % a;
        }
    }
}
```

힌트

두 정수의 최대공약수는 유클리드 호제법을 이용해 쉽게 구할 수 있다.
a = b * q + r이면 gcd(a,b)==gcd(b,r)가 참이다.
따라서 a와 b가 주어지고, a가 더 크다고 했을 때
gcd(a, b)==gcd(b,a%b)이고,
gcd(b,a%b) 또한 같은 방식으로 작은 수와
큰 수 % 작은 수의 최대공약수를 구하면 된다.
이 때, 큰 수가 작은 수로 나누어 떨어지면(약수가 되면)
작은 수는 최대공약수가 된다.
(약수가 원본보다 클 수 없기 때문에 최대이다.)

재귀함수를 이용하여 아래와 같이 작성할 수도 있다.
왜 이렇게 쓰일 수 있는지 생각해보자.

```
int gcd(int p, int q) {
    if (q == 0) return p;
    return gcd(q, p % q);
}
```

this 포인터

- 클래스 안쪽에서 자신(인스턴스)를 가리키는 포인터이다.

```
void Fraction::store
    (int numer, int denom) {
    this->numerator = numer;
    this->denominator = denom;
    return;
}
```

포인터이기 때문에 .이 아니라 ->로 접근한다.
(*this).numerator 처럼 사용할 수도 있다.

평소에는 묵시적으로(implicitly) 사용되고 있다.
(굳이 this를 명시하지 않아도 본인의 멤버를 사용한다.)

```
void Fraction::store
    (int numerator, int denominator) {
    this->numerator = numerator;
    this->denominator = denominator;
    return;
}
```

매개변수 등 지역 변수가 멤버 변수와 같은 식별자를 가지고 있는 경우를 대비해 명시적으로 사용하는 것을 권장한다.

인라인 함수

- 인라인(inline) 함수가 되면
  컴파일 과정에서 함수가 호출된 곳에
  코드를 그대로 삽입하여 실행 시에는
  호출이 일어나지 않도록 한다.

```
inline int isOdd(int i) {
    return (i % 2);
}
```

호출을 하지 않기 때문에 빨라진다.

하지만 너무 긴 코드를 인라인으로 만드는 것은 권장하지 않는다. (3~8줄 내외를 추천)

```
int main() {
    int sum = 0;
    for (int i = 0; i <= 1000; i++) {
        if (isOdd(i))
            sum += i;
    }
    cout << "1 ~ 1000 까지의 홀수 합"
        << i << endl;
}
```

```
int main() {
    int sum = 0;
    for (int i = 0; i <= 1000; i++) {
        if (i % 2)
            sum += i;
    }
    cout << "1 ~ 1000 까지의 홀수 합"
        << i << endl;
}
```

인라인 멤버 함수

명시적인 방법

```
class Fraction {
// 생략
    inline void print();
};
```

구현할 때는 명시하지 않아도 된다.

```
void Fraction::print() {
    cout << numerator << "/"
        << denominator;
    return;
}
```

묵시적인 방법

```
class Fraction {
// 생략
```

선언에서 구현할 때는 inline을 적지 않아도 된다.

```
    void print() {
        cout << numerator << "/"
            << denominator;
        return;
    }
};
```

생성자

- 생성자(constructor)는 인스턴스가 만들어질 때 무조건 호출되는 함수이다.
- 모든 클래스에는 하나 이상의 생성자가 반드시 존재해야 한다.

매개변수가 없는 생성자를 '기본 생성자(default constructor)'라고 한다.

클래스를 만들 때 생성자를 만들지 않으면 컴파일러가 내용이 없는 기본 생성자를 은글슬쩍 만들어준다.

main()

```
Fraction fr;
fr.print();
```

생성자가 호출된다.

```
Hello, Fraction!
0/1
```

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    Fraction();

    void store(int numer, int denom);
    void print();
};

Fraction::Fraction() {
    clog << "Hello, Fraction" << endl;
    this->numerator = 0;
    this->denominator = 1;
}
```

생성자는 클래스와 이름이 같으며, 반환 형을 가지지 않는다.

생성자에서 멤버 변수의 기본값을 지정해주기도 한다.

생성자

- 생성자(constructor)에 매개변수를 넣어 멤버 변수의 기본값을 초기화하는 데 활용할 수 있다.

main()

```
Fraction fr1;
fr1.print(); cout << endl;

Fraction fr2(4);
fr2.print(); cout << endl;

Fraction fr3(4, 19);
fr3.print(); cout << endl;
```

```
0/1
4/1
4/19
```

다른 생성자를 만들려면 반드시 기본 생성자를 정의해줘야 한다.

```
Fraction();
Fraction(int numer);
Fraction(int numer, int denom);
```

```
Fraction::Fraction() {
    this->numerator = 0;
    this->denominator = 1;
}
```

분자 분모 둘 다 없으면 그냥 0/1로 초기화한다.

```
Fraction::Fraction(int numer) {
    this->numerator = numer;
    this->denominator = 1;
}
```

분자만 들어오면 (분자)/1로 초기화한다.

```
Fraction::Fraction(int numer,int denom) {
    this->numerator = numer;
    this->denominator = denom;
}
```

둘 다 들어오면 (분자)/(분모)로 초기화한다.

기본 생성자

- 기본 생성자에 모든 매개변수에 대한 기본값을 줄 수도 있다.

```
Fraction(int numer, int denom);
```

얘가 기본 생성자이기 때문에 Fraction()을 만들지 않아도 된다.

```
Fraction::Fraction
    (int numer = 0, int denom = 1) {
    this->numerator = numer;
    this->denominator = denom;
}
```

매개변수는 앞에서부터 채워진다.

main()

```
Fraction fr1;
fr1.print(); cout << endl;

Fraction fr2(4);
fr2.print(); cout << endl;

Fraction fr3(4, 19);
fr3.print(); cout << endl;
```

매개변수를 안 넣었을 때 호출되는 생성자가 기본 생성자이다.

매개변수가 하나 밖에 없다면 numer에 그 값이 들어가고, denom은 기본값인 1이 된다.

매개변수가 두 개면 numer과 denom에 각각 그 값들이 들어간다.

```
0/1
4/1
4/19
```

초기화 리스트

- 매개변수가 멤버 변수를 초기화한다면, 초기화 리스트를 이용할 수 있다.

```
Fraction(int numer, int denom);
```

초기화 리스트 없이

```
Fraction::Fraction
    (int numer = 0, int denom = 1) {
    this->numerator = numer;
    this->denominator = denom;
}
```

초기화 리스트 있이

```
Fraction::Fraction
    (int numer = 0, int denom = 1)
    : numerator   (numer),
      denominator (denom) {

    // 따로 초기화 구문을
    // 써주지 않아도 된다.
}
```

매개변수 목록 뒤에 초기화 리스트를 써준다.

복사 생성자

- 생성자 중에 기존에 있는 인스턴스를 매개변수로 받아 자신에게 복제하는 생성자를 복사(copy) 생성자라고 한다.

main()

```
Fraction fr1(4, 19);
cout << "fr1: ";
fr1.print();
cout << endl;

Fraction fr2(fr1);
cout << "fr2: ";
fr2.print();
cout << endl;
```

```
fr1: 4/19
fr2: 4/19
```

원본 인스턴스는 매개변수로 넘어오는 과정에서 복제되지 않도록 참조(reference)로 받아오고, 그 내용이 수정되면 안 되므로 const를 붙인다.

```
Fraction(const Fraction& original);
```

```
Fraction::Fraction
    (const Fraction& original) {
    this->numerator = original.numerator;
    this->denominator
        = original.denominator;
}
```

복사 생성자는 이럴 때 호출된다.
- 기존 인스턴스를 받아 새롭게 복사된 인스턴스가 만들어질 때
- 인스턴스가 함수의 매개변수로 전달될 때 (pass by value)
  그래서 함수의 매개변수로 전달할 땐 원본을 활용할 수 있도록 &를 붙여서 전달하는 게 좋다. (pass by reference)
- 함수에서 인스턴스가 반환될 때

익명 객체

- 함수에서 객체를 반환할 때 따로 인스턴스를 만들지 않고 바로 만들어서 반환할 수 있다.
- 인스턴스 정의를 하지 않았으므로 식별자(이름)가 없으므로 '익명(anonymous) 객체'라고 한다.

```
Fraction inputFraction();

int main() {
    Fraction fr = inputFraction();
    fr.print();
    cout << endl;
    return 0;
}

Fraction inputFraction() {
    int numer, denom;
    cout << "분자: ";
    cin >> numer;
    cout << "분모: ";
    cin >> denom;

    return Fraction(numer, denom);
}
```

인스턴스를 따로 만들지 않고
클래스 이름을 사용해 바로 반환해준다.

소멸자

- 생성자와는 반대로 소멸자(destructor)는 인스턴스가 죽어 없어질(die) 때 호출된다.

main()

```
Fraction* pFr = new Fraction();
delete pFr;
```

정적으로 할당된 객체는 프로그램이 끝날 때 같이 사라지므로 프로그램 마지막에 호출된다. 동적할당을 했다면 delete될 때 호출된다.

```
Hello, Fraction
Bye, Fraction
```

생성자에서 출력되는 "Hello, Fraction"

소멸자에서 출력되는 "Bye, Fraction"

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    Fraction(int numer, int denom);
    ~Fraction();

    void store(int numer, int denom);
    void print();
};

Fraction::~Fraction() {
    cout << "Bye, Fraction!" << endl;
    // 동적할당한 게 있으면 해제해주고
    // 파일 연 게 있으면 닫아주고
    // 뒷정리를..
}
```

멤버 함수의 종류

- 멤버 함수는 크게 세 종류로 구분할 수 있다.

관리자 함수(Manager Function)

- 생성, 복사, 소멸하는 함수
  ex) 생성자, 소멸자

변형자 함수(Mutator Function)

- 값을 수정하는 함수
  ex) store(int numer, int denom)

접근자 함수(Accessor Function)

- 값에 접근해 출력하거나 넘겨주는 함수
  ex) print()

프렌드 함수

- 클래스의 외부에 만들어진 함수는 private 멤버에 접근할 수 없다.
- 그런데 클래스에서 "얘는 내 친구야" 라고 해주면 private에도 접근할 수 있게 된다.

main()

```
Fraction fr1(4, 5);
Fraction fr2(4, 6);

Fraction fr = add(fr1, fr2);

cout << "fr1: "; fr1.print(); cout << endl;
cout << "fr2: "; fr2.print(); cout << endl;
cout << "fr: "; fr.print(); cout << endl;
```

```
fr1: 4/5
fr2: 4/6
fr: 44/30
```

```
class Fraction {
private:
    // 중략
public:
    // 중략
    friend Fraction add
        (const Fraction& fr1,
        const Fraction& fr2);
}

Fraction add (const Fraction& fr1,
    const Fraction& fr2) {
    int numer = (fr1.numerator *
        fr2.denominator) +
        (fr2.numerator * fr1.denominator);
    int denom = fr1.denominator *
        fr2.denominator;

    return Fraction(numer, denom);
}
```

add()는 Fraction의 멤버 함수가 아니지만 private 멤버 변수인 numerator과 denominator에 접근하고 있다.

구조체와 클래스

- 구조체(Structure)와 클래스는 구조도 사용법도 똑같다.
- 접근자가 없을 때 구조체는 public, 클래스는 private이 기본값이 된다.

둘은 같은 것이다.

```
struct Fraction {
private:
    int numerator;
    int denominator;
public:
    void store(int, int);
    void print();
};
```

```
class Fraction {
private:
    int numerator;
    int denominator;
public:
    void store(int, int);
    void print();
};
```

```
struct Fraction {
    int numerator;
    int denominator;
    void store(int, int);
    void print();
};
```

얘는 모두 public이다.

```
class Fraction {
    int numerator;
    int denominator;
    void store(int, int);
    void print();
};
```

얘는 모두 private이다.

둘은 다른 것이다.

프렌드 함수 사용해보기

분자와 분모를 모두 받는 생성자가 있는 Fraction 클래스에 두 개의 매개변수를 넣어 인스턴스화 시킬 때, 분모에 0이 들어가면 에러 메시지와 100번 에러를 반환하며 종료되는 프로그램을 작성하시오.
정상적으로 값이 들어오면 기약분수로 만들고 그 값을 출력한다. 단, 기약분수로 만들 때 분수의 값이 음수인 경우 분자에만 – 기호를 사용한다.

오래 전에 만들었던 Fraction의 멤버 함수 addTo()를 참고하여 다른 사칙연산에 함수 subtractTo(), multiplyBy(), divideBy()를 추가하고 main()에서 각 함수의 결과를 출력하시오.

$$\frac{b}{a}+\frac{d}{c}=\frac{bc+ad}{ac} \qquad \frac{b}{a}-\frac{d}{c}=\frac{bc-ad}{ac} \qquad \frac{b}{a}\times\frac{d}{c}=\frac{bd}{ac} \qquad \frac{b}{a}\div\frac{d}{c}=\frac{bc}{ad}$$

프렌드 함수 사용해보기

Fraction의 프렌드 함수 add()를 다른 사칙연산에 대한 프렌드 함수 subtract(), multiply(), divide()를 추가하고 main()에서 각 함수의 결과를 출력하시오.
이 때, 생성자는 분자, 분모가 들어가는 한 개만 만들도록 하고 초기화 리스트를 사용한다.

$$\frac{b}{a}+\frac{d}{c}=\frac{bc+ad}{ac} \qquad \frac{b}{a}-\frac{d}{c}=\frac{bc-ad}{ac} \qquad \frac{b}{a}\times\frac{d}{c}=\frac{bd}{ac} \qquad \frac{b}{a}\div\frac{d}{c}=\frac{bc}{ad}$$

랩탑 컴퓨터를 나타내는 Laptop 클래스를 구현하고 main()에서 테스트하시오.

Laptop

- osName: char[20]
- displayWidth: double
- memorySize: long

// 운영체제 이름
// 화면 너비 (인치)
// 저장 공간 용량 (MB)

+ Laptop() ≪create≫
+ Laptop(char[20], double, long) ≪create≫
+ ~Laptop() ≪destroy≫
+ getOsName()
+ getDisplayWidth()
+ getMemorySize()
+ setOsName()
+ setDisplayWidth()
+ setMemorySize()
+ print()

// 기본 생성자, 사용자에게 값을 입력 받도록 함, "Laptop (this의 값) assembled" 출력
// 3개의 매개변수가 멤버 변수의 값이 되는 생성자, "Laptop (this의 값) assembled" 출력
// 소멸자, "Laptop (this의 값) destroyed" 출력
// Getter 함수
// Getter 함수
// Getter 함수
// Setter 함수
// Setter 함수
// Setter 함수
// 멤버 변수를 이쁘게 정리해서 출력

2017.05.11. 프로그래밍 기초 (2017-1)
with D.com