객체프로그래밍 정리노트

202204103 ict융합공학부 이승훈
202204023 ict융합공학부 김보민

[프렌즈 선언]

1. 외부함수 초대 (전역함수)

```
// 외부 함수 equals()를 Rect 클래스에 프렌드로 선언
class Rect { // Rect 클래스 선언
...
friend bool equals(Rect r, Rect s);
};
```

2. 다른 클래스의 멤버 함수 초대 (다른 클래스의 특정 멤버 함수)

```
// RectManager 클래스의 equals() 멤버 함수를 Rect 클래스에 프렌드로 선언
class Rect {
.............
friend bool RectManager::equals(Rect r, Rect s);
};
```

3. 다른 클래스의 전체 함수 초대 (다른 클랴스의 모든 멤버 함수)

```
// RectManager 클래스의 모든 멤버 함수를 Rect 클래스에 프렌드로 선언
class Rect {
.............
friend RectManager;
};
```

[9주차 정리노트]

//보민 , ->승훈

[예제7-2]

승훈 풀이)

ㄴ 문제에 필요한 class들 선언하기

ㄴ 클래스의 equal멤버를 클래스로 선언하는 코드 작성하기

ㄴ 메인 함수 작성하여 출력하기

```
#include <iostream>
using namespace std;
// 처음 코드를 작성할 때 Rect 클래스가 선언되기 전에 먼저 참조되는 컴파일 오류를 막기 위해서 오류 방지 코드를 작성해줘야 해
class Rect {
	int width, height;
public:
	Rect(int width, int height) {
		this->width = width;this->height = height;
	}
	friend bool equals(Rect r, Rect s);
	-> 프렌드 함수를 적용해줌으로써 외부 함수가 해당 클래스의 private 멤버에 접근할 수 있게 되는게 중요하다고 생각해
	// 그래서 외부에서 프렌드 함수를 정의할 수 있게 되는 거군
};

bool equals(Rect r, Rect s) {
	if (r.width == s.width && r.height == s.height) return true;
	else return false;
} -> 프렌드 함수 equals가 두 개의 Rect 객체를 비교해주고 있는 코드라고 볼 수 있어
int main() {
	Rect a(3, 4), b(4, 5);
	if (equals(a, b))cout << "equal" << endl;
	else cout << "not equal" << endl;
}
```

[수정]

코드 작성 맨 위에 class Rect;를 추가하여 컴파일 오류 방지

[예제7-3]

보민 풀이)

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

class RectManager {

public:
	bool equals(Rect r, Rect s);
	void copy(Rect& dest, Rect& src);
};

class Rect {
	int width,  height;
public:
	Rect(int width, int height) { this->width = width;this->height = height; }
	friend bool RectManager::equals(Rect r, Rect s);

}; // 프렌드 함수를 선언할 때 Rect 함수가 이미 정의되어 있어야 하고, RectManager 클래스도 이미 선언되어 있기 때문에 Rect 클래스의 equals 함수가 RectManager의 멤버 함수에 접근할 수 있게 되는 거 맞지 ?
-> 맞아, 근데 여기서 더 중요한 점은 RectManager::equals 함수가 Rect 클래스의 private 멤버에 접근할 수 있지만 Rect 클래스가 RectManager 의 private 멤버 함수에 접근할 수 없다는 걸 알아둬야 해.
// 좀 더 자세히 설명해줘
-> 클래스 간의 프렌드 관계는 단방향이기 때문이야. 프렌드 함수의 사용은 접근이 제한되고 클래스 간의 결합을 관리하는 것에 도움이 돼.
bool RectManager::equals(Rect r, Rect s) {
	if (r.width == s.width && r.height == s.height)return true;
	else return false;
}
int main() {
	Rect a(3, 4), b(3, 4);
	RectManager man;

	man.copy(b, a);
	if (man.equals(a, b))cout << "equal" << endl;
	else cout << "not equal" << endl;
}
```

[예제7-4]
승훈 풀이)

ㄴ클래스 선언하기
ㄴ필요한 함수들 선언하기
ㄴ함수 구현 코드 작성(연산자 사용)
ㄴ메인 함수 작성하여 출력하기

```
#include <iostream>
using namespace std;
class Power {
        int kick;
        int punch;
public:
        Power(int kick = 0, int punch = 0) {
                this->kick = kick;this->punch = punch;
        }
        void show();
        Power operator+(Power op2);
// 이 코드를 설명해보면 operator+ 함수는 +연산자를 오버로딩 해주고 있어
-> 오버로딩을 하는 이유는 정확히 뭘까 ?
// 연산자의 오버로딩은 클래스나 기본 데이터 타입인 연산자의 동작을 재정의 해주고 확장하기 위해 쓰인다고 볼 수 있어.

};
void Power::show() {
        cout << "kick=" << kick << ',' << "punch=" << punch << endl;
//오버로딩 한 걸 가지고 kick과 punch 값을 더하여 Power이라는 새로운 객체를 생성하고 반환하도록 구현한 코드라고 볼 수 있지.
}
Power Power::operator+(Power op2) {
        Power tmp;
        tmp.kick = this->kick + op2.kick;
        tmp.punch = this->punch + op2.punch;
        return tmp;
}

int main() {
        Power a(3, 5), b(4, 6), c;
        c = a + b;
        a.show();
        b.show();
        c.show();
```

```
}
```

[예제7-5]

보민 풀이)

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
	Power(int kick = 0, int punch = 0) {
		this->kick = kick;this->punch = punch;
	}
	void show();
	bool operator==(Power op2);
	-> 이도 예제 7-4와 같은 방법으로 ==연산자가 쓰였네
};

void Power::show() {
	cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}

bool Power::operator==(Power op2) {
	if (kick == op2.kick && punch == op2.punch)return true;
	else return false;
}
int main() {
	Power a(3, 5), b(3, 5);
	a.show();
	b.show();
	if (a == b)cout << "두 파워가 같다." << endl;
	else cout << "두 파워가 같지 않다." << endl;
}
```

[예제 7-11]
승훈 풀이)

->이거 위에 연산자를 프렌드로 작성하는 방법은 모르겠어
// 아까 pulic 에 연산자를 정의했던 것처럼 friend를 사용해주고 Power클래스 안에 연산자를 지정해주면 돼.
-> 아 그러면 아까처럼 아래에서도 연산자 함수를 구체적으로 구현해서 코드 작성을 하면 되겠네

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
	Power(int kick = 0, int punch = 0) {
		this->kick = kick; this->punch = punch;
	}
	void show();
	friend Power operator+(int op1, Power op2);
};
void Power::show() {
	cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
Power operator+(int op1, Power op2) {
	Power tmp;
	// 이처럼 임시 객체를 선언한 이유는 int와 Power 객체 즉, 두 개의 다른 데이터 타입을 더하기 위해서 쓰이는 것을 볼 수 있어.
	-> 다른 데이터 타입을 더하는 연산자를 오버로딩 하기 위해서 두 개를 어떻게 더할지 정해주는 거구나,
	tmp.kick = op1 + op2.kick;
	tmp.punch = op1 + op2.punch;
	return tmp;
}
int main() {
	Power a(3, 5), b;
	a.show();
	b.show();
	b = 2 + a;
	a.show();
	b.show();
}
```

[예제 7-14]
보민 풀이)

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
	Power(int kick = 0, int punch = 0) {
		this->kick = kick; this->punch = punch;
	}
	void show();
	Power& operator << (int n);
};
void Power::show() {
	cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
Power& Power::operator <<(int n) {
	// 이 코드는 Power 객체에 값을 추가하기 위해서 사용했어
	-> 그럼 이건 n 정수값을 전달하고 이는 kick 과 Power에 더해지겠구나
	kick += n;
	punch += n;
	return *this;
}

int main() {
	Power a(1, 2);
	a << 3 << 5 << 6;
	//이런 식으로 코드를 작성하게 되면 순차적으로 더하는 것이라는 걸 알아둬야 해.

	a.show();
}
```