민 선 생 코 딩 학 원 훈 련 반

수업노트 Class2

배우는 내용

1. 오버로딩
2. 생성자
3. private / public

오버로딩

- 같은 이름이지만, 인자값이 다른 함수를 여러 개 만들 수 있음 = 함수 오버로딩
- 인자값 보고, 어떤 함수로 호출 될 지 판단 함

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

void kfc()
{
    cout << "ONE";
}

void kfc(int t)
{
    cout << t;
}

int main()
{
    kfc();
    kfc(15);

    return 0;
}
```

인자값이 없으므로
첫번째 함수 호출

인자값이 한 개 있으므로
두번째 함수 호출

클래스에서도 오버로딩이 가능 함

- 메서드이름이 같을 때
  인자값에 따라서 호출되는 메서드가 달라 짐

- 실행결과
  0
  15

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

class BBQ
{
public:
    int t;

    void init()
    {
        t = 0;
    }

    void init(int gg)
    {
        t = gg;
    }

    void print()
    {
        cout << t << endl;
    }
};

int main()
{
    BBQ x;

    x.init();
    x.print();
    x.init(15);
    x.print();

    return 0;
}
```

생성자란?

- 인스턴스를 생성할 때 **자동으로 호출되는 특수한 메서드**
- 생성자 이름은 반드시 **클래스와 같은 이름으로** 해야한다
  void도 붙이지 않는다.

<생성자>
인스턴스 선언시
자동호출된다

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

class BBQ
{
public:

    int x;
    int y;

    BBQ()
    {
        cout << "START";
        x = 10;
        y = 15;
    }
};

int main()
{
    BBQ t;

    return 0;
}
```

생성자를 언제 사용할까?

- 생성자는 클래스 안에 있는 맴버변수들을 초기화할때 쓰는 용도로 주로 사용 됨

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

class BBQ
{
public:

    int a, b, c;

    BBQ(int m1, int m2, int m3)
    {
        a = m1;
        b = m2;
        c = m3;
    }
};

int main()
{
    BBQ t(3, 6, 1);

    return 0;
}
```

생성자 오버로딩

- 생성자도 오버로딩이 가능하다.

- 실행결과
  # ###

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

class MC
{
public:
    MC()
    {
        cout << "# ";
    }

    MC(int t)
    {
        for (int i = 0; i < t; i++)
        {
            cout << "#";
        }
    }
};

int main()
{
    MC a;
    MC b(3);

    return 0;
}
```

주의
b(3)은 함수b를 호출하는 것이 아니다.
인스턴스 b 생성 후, 생성자에 3을 보내는 것이다

private / public은 공개 여부 설정

- 맴버들을 클래스 밖에서 공개할지 말지를 결정한다.
- 인스턴스가 맴버들을
  사용할 수 있을지, 없을 지 결정한다
  (private / public 으로 접근 여부를 결정)

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

class MC
{
    int a;
    int b;
    int c;

    void kfc()
    {

    }

public:

    int g;
    void bbq()
    {

    }
};

int main()
{


    return 0;
}
```

public: 외에 있는
맴버들은
모두 private로
비공개로 자동 설정

public:
이후에 나오는 맴버들은
모두 공개로 설정

클래스 내부에서 private / public

- private 맴버들과 public 맴버들은
  클래스 안에서 자유롭게 쓸 수 있다.

- 클래스 안에서 공개여부는 신경쓰지 않는다.

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

class MC
{
    int a;
    int b;
    int c;

    void kfc()
    {

    }

public:

    int g;
    void bbq()
    {
        a = 10;
        b = 20;
        c = 15;

        kfc();
    }
};

int main()
{
    MC t;
    t.bbq();

    return 0;
}
```

private / public 예제

- 객체 t는 private 맴버들을 사용 못함
- 객체 t는 public 맴버들을 사용할 수 있음

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

class MC
{
    int a;
    int b;
    int c;

    void kfc()
    {

    }

public:

    int g;
    void bbq()
    {

    }
};

int main()
{
    MC t;
    t.g = 15; //OK
    t.a = 20; //Runtime Error

    t.bbq(); //OK
    t.kfc(); //Runtime Error

    return 0;
}
```

공개 여부를 설정하는 이유1

- class에 있는 모든 맴버들을 public 으로 설정하면, 편리하게 접근 가능하긴하다

  하지만, class를 만들 때 private와 public으로 구분하는 이유가 있다.

- Class를 사용할 때, **잘못된 사용으로 인한 버그를 막을 수 있다** (=안전성 확보)

class **사용자가 건드리지 않았으면 하는 맴버는 private 로 설정**하고,

class 사용자가 마음대로 쓸 수 있도록 하는 맴버는 **public**으로 설정한다

공개 여부를 설정하는 이유2

- Class를 쓰는 대표적인 이유는 재사용성이다.
- 재사용중에 공개되어있는 class 맴버변수들은 건드렸을 때 버그가 발생한다면 안전성이 떨어지기 때문에, 재사용성이 떨어진다고 볼 수 있다.
- **비공개 / 공개 할 맴버변수, 메서드를 구분해서 Class를 제작하는 습관을 길러야 한다**