13 - 신윤호, 신은총

정리노트#6 (11/13)

① 상속관계에서의 함수 재정의
→ 기본 클래스의 멤버 함수로 원하는 작업을 할 수 없으면 파생 클래스에서 기본클래스 멤버 함수와 동일한 이름, 원형으로 해당 함수를 재정의 사용 할 수있다.

② 정적 바인딩
→ 컴파일시 어떤 함수를 호출 할 것지 결정되는 것
+ 범위 지정 연산자로 기본 클래스 멤버함수 / 파생 클래스 재정의 함수 구분해 호출 가능

③ 함수 재정의
→ 파생 클래스의 객체는, 기본클래스의 함수나 파생클래스에서 재정의된 함수를 갖고 있음.

④ 오버라이딩 (오버라이딩 범위 지정 연산자 ::)
→ 파생 클래스의 재정의된 가상함수만이 호출될 기회를 갖고 있다.
→ 오버라이딩 : 가상함수를 만드는 이유는, 파생 클래스가 자신의 목적에 맞게 재정의 하라.

⑤ 동적 바인딩 발생 상황 4가지
- 기본 클래스 내 멤버 함수가 가상 함수를 호출
- 파생 클래스 내 멤버 함수가 가상 함수를 호출
- main() 같은 외부 함수에서 기본클래스 포인터로 가상함수 호출
- 다른 클래스에서 가상함수 호출

⑥ 가상 소멸자
- 기본 클래스의 소멸자를 만들때, 가상 함수로 작성 할 것을 권장

⑦ 추상 클래스
- 객체 생성 위한 클래스가 아니라, 상속을 위하여 기본 클래스로 활용하는것이 목적

* 예제 9-7(실습) 코드 분석 / 실습

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

class Calculator {
   void input() {
      cout << "정수 2 개를 입력하세요>> ";
      cin >> a >> b;
   }
protected:
   int a, b;
   virtual int calc(int a, int b) = 0; // 두 정수의 합 리턴
public:
   void run() {
      input();
      cout << "계산된 값은 " << calc(a, b) << endl;
   }
};

int main() {
   Adder adder;
   Subtractor subtractor;
   adder.run();
   subtractor.run();
}
```

순수 가상함수, `Calculator 클래스의 인스턴스를 직접 생성할 수 없음을 의미`

→ Calculator 클래스 상속 받아 calc() 함수 구현

```
class Adder: public Calculator {
protected:
    int calc(int a, int b) {
        return a+b;
    }
};

class Subtractor : public Calculator {
protected:
    int calc(int a, int b) {
        return a-b;
    }
};
```