
객체-지향 프로그래밍1

(object-oriented programming)

Programming Languages

2018 1학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

목차

- 객체 지향 프로그래밍 개념
- 상속
- 다형성

객체 지향 프로그래밍 개념

□ 추상 데이터 타입

- 장점
 - ✓ 캡슐화, 정보 은닉, 분리 컴파일
 - > 구조화된 프로그래밍이 가능, 효과적인 소프트웨어 생산, 유지 보수에 유리
- 문제점
 - ✓ 재사용성이 여전히 좋지 않은
 - ✓ 실세계에서 자주 등장하는 상/하위 관계 표현이 미흡
(두 추상 데이터 타입은 평면적으로 대등한 관계를 가지고 독립적으로 존재)
- “재사용성이 보다 좋고, 실제 세계를 더 잘 표현할 수 있는 개념이 필요”

객체 지향 프로그래밍 개념

□ 객체 지향 개념

- 추상 데이터 타입과 비슷함(캡슐화)
 - ✓ 추상 데이터 타입 : 클래스(class)
 - ✓ 각 클래스에 속하는 각 개체 : 객체(object) , 해당 클래스의 사례(instance)라고 함
 - ✓ 객체에 대한 연산(부프로그램)들 : 메소드(method)
 - 메소드는 인터페이스를 나타내는 헤더 선언과 구현으로 나뉨
- 추상 데이터 타입과 다름
 - ✓ 상속(inheritance)
 - ✓ 다형성(polymorphism)
 - ✓ 기타

객체 지향 프로그래밍 개념

□ 객체 지향 개념과 타입

– 클래스와 타입

- ✓ 클래스도 실세계의 대상을 표현하고, 공간의 효율적 사용과 타입 검사를 지원할 수 있음
- ✓ 객체를 담는 변수(이하, '객체 변수')는 그를 사례로 가지는 클래스의 타입을 갖게 됨

▪ 예) Person p = <Person : 이름= 홍길동, 주소 : 서울시 성동구 행당동, 연산 = {printData()... }>

- ✓ 추상데이터 타입과 마찬가지로

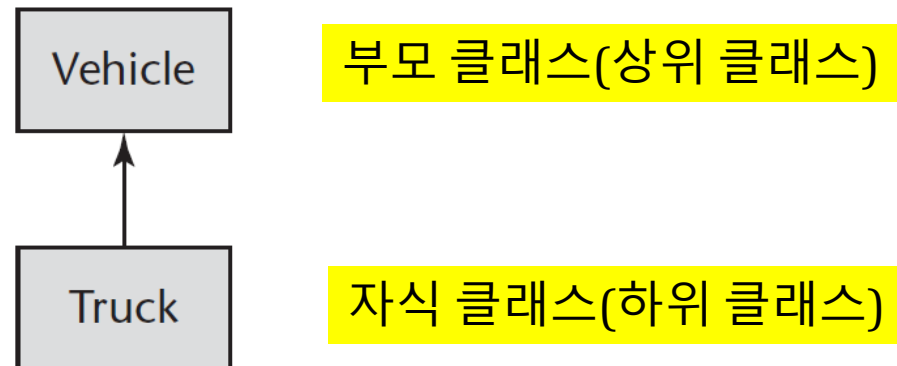
상속(inheritance)

□ 상속

- 두 개의 클래스가 가지는 부모 자식 관계
- 부모 클래스는 상위 클래스(super class), 자식 클래스는 하위 클래스(sub class)라 함
- 하위 클래스에 속하는 객체는 상위클래스에도 속하는 것으로 간주
- 예) 차와 트럭, 새와 물새
 - ✓ 집합의 포함 관계와도 비슷
 - ✓ 실세계에 많이 내재됨

Figure 12.1

A simple example of inheritance



상속(inheritance)

□ 상속의 재사용성

- 하위 클래스의 정의 = 상위 클래스의 정의 + α
 - 상위 클래스의 모든 데이터 선언과 메소드를 이어 받아야 하며, 추가적으로 데이터 선언과 메소드를 더 가질 수 있음
- 새로운 클래스를 만들때 유리
 - 기존의 적당한 클래스로 부터 상속받아 개발 시간이 단축
 - 상위 클래스의 내용을 완전히 모르더라도 상속받을 수 있음
 - 개발을 레고 조립처럼 컴포넌트화 하기에 유리

상속(inheritance)

□ 상속과 상/하위 타입

– 상/하위 타입

- 하위 타입 객체는 상위 타입의 객체가 수행하는 모든 연산을 지원, 상위 타입 객체를 기대하는 모든 위치에서 사용 가능
- 예) Student는 Person 의 하위 타입으로 설정 가능

– 클래스 상속의 재사용성은 상/하위 타입과 유사

- 하위클래스의 데이터 구조와 메소드가 상위 클래스의 데이터 구조와 메소드를 모두 가지고 있음

-> 상/하위 타입

- 상위 클래스 타입의 변수가 하위 클래스 객체를 가질 수 있음
 - 예) Person p = < Student : 이름= 홍길동, 주소 : 서울시 성동구 행당동, 학번 =20810, 학점 = 4.3 , 연산 = {printData()... }>

상속(inheritance)

□ 오버라이드(override)

- 상위 클래스가 제공하는 기존의 메소드의 구현을 그대로 상속하지 않고 자신에게 맞게 변경해서 상속 받는 것
 - 클래스 Person의 printData()는 주민번호와 이름과 주소만을 출력,
 - 클래스 Student는 이를 오버라이드하여 printData()에서 학번, 학점까지 더 출력하도록 할 수 있음

C++ 에서의 객체 지향 프로그래밍(상속) 예제

□ 자식(파생) 클래스의 구문 형태

자식 클래스(하위 클래스)

부모 클래스(상위 클래스)

```
class derived_class_name : derivation_mode base_class_name  
{ data member and member function declarations } ;
```

유도 모드(derivation_mode) : public or private

□ 생성자 호출

- 부모 생성자에 초기화 데이터가 제공되어야 할 때, 그 데이터는 서브 클래스 객체 생성자에 대한 호출에 주어짐

```
subclass(subclass parameters) : parent_class(superclass parameters) {  
    ...  
}
```

C++ 에서의 객체 지향 프로그래밍(상속) 예제

□ 유도 모드에 따른 상속

- 부모 클래스의 공용(public)과 보호(protected) 멤버는 공용(public) 유도 클래스(자식 클래스)에서 각각 공용과 보호
- 전용-유도 클래스에서 부모 클래스의 공용과 보호 멤버는 모두 전용

```
class base_class {  
    private:  
        int a;  
        float x;  
    protected:  
        int b;  
        float y;  
    public:  
        int c;  
        float z;  
};  
  
class subclass_1 : public base_class {...};  
class subclass_2 : private base_class {...};
```

subclass1에서 b, y는 protected, c와 z는 public
subclass2에서 b, y, c, z는 private

C++ 에서의 객체 지향 프로그래밍(상속) 예제

□ 유도 모드에 따른 상속

- 전용(private) 클래스 유도에서, 부모 클래스의 어느 멤버도 묵시적으로 파생 클래스의 사례에 가시적이지 않음
- 가시적이어야 하는 멤버는 파생 클래스에서 재반출(reexport) 되어야 함

- 예)

```
class subclass_3 : private base_class {  
    base_class :: c;  
    ...  
}
```

- ✓ subclass_3의 사례(instance)는 c를 접근할 수 있음. c에 관한한 마치 유도가 공용인 것 같음

C++ 에서의 객체 지향 프로그래밍(상속) 예제

❑ C++의 linked-list 클래스

```
class single_linked_list {  
    private:  
        class node {  
            public:  
                node *link;  
                int contents;  
        };  
        node *head;  
    public:  
        single_linked_list() {head = 0};  
        void insert_at_head(int);  
        void insert_at_tail(int);  
        int remove_at_head();  
        int empty();  
};
```

C++ 에서의 객체 지향 프로그래밍(상속) 예제

□ C++의 linked-list 클래스를 상속하는 클래스

```
class stack : public single_linked_list {
public:
    stack() {}
    void push(int value) {
        insert_at_head(value);
    }
    int pop() {
        return remove_at_head();
    }
};

class queue : public single_linked_list {
public:
    queue() {}
    void enqueue(int value) {
        insert_at_tail(value);
    }
    int dequeue() {
        return remove_at_head();
    }
};
```

이 경우, public으로 상속했기 때문에 다른 공용 멤버를 접근할 수 있음
-> 스택 혹은 큐 무결성을 파괴할 수 있음(다른 함수 호출 가능)

C++ 에서의 객체 지향 프로그래밍(상속) 예제

❑ C++의 linked-list 클래스를 상속하는 클래스

```
class stack_2 : private single_linked_list {
public:
    stack_2() {}
    void push(int value) {
        single_linked_list :: insert_at_head(value);
    }
    int pop() {
        return single_linked_list :: remove_at_head();
    }
    single_linked_list:: empty();
};

class queue_2 : private single_linked_list {
public:
    queue_2() {}
    void enqueue(int value) {
        single_linked_list :: insert_at_tail(value);
    }
    int dequeue() {
        single_linked_list :: remove_at_head();
    }
    single_linked_list:: empty();
};
```

이 경우, 위에서 “::”로 메소드 접근을 허용하지만,
다른 부모의 메소드는 접근이 불가함

Thank you for your attention !!

Q and A