상속
(Chapter 14 Inheritance)

숭실대학교
김강희 교수
(khkim@ssu.ac.kr)

개요

- 객체 지향 프로그래밍
  - 강력한 프로그래밍 테크닉
  - '상속'이라고 부르는 추상화 방법을 제공함
- 클래스를 정의할 때
  - general class (base class, parent class, super class)의 속성들을 상속받는 specialized class (derived class, child class, sub class) 을 작성한다.
    - specialized class 는 general class 의 멤버 변수들과 멤버 함수들을 갖는다.
  - specialized class 안에서 general class 의 기능을 변경하거나, 새 기능을 추가한다.

Tetris.h → CTetris.h

```
18  class Tetris {
19  protected:
20      // static members
21      static Matrix ***setOfBlockObjects;
22      static int numTypes;
23      static int numDegrees;
24      static int wallDepth;
25
26      // dynamic members
27      int rows; // rows of screen = dy + wallDepth
28      int cols; // columns of screen = dx + 2*wallDepth
29      int type;
30      int degree;
31      int top;
32      int left;
33
34      TetrisState state;
35      Matrix *iScreen;
36      Matrix *oScreen;
37      Matrix *currBlk;
38
39  public:
40      static void init(int **setOfBlockArrays, int nTypes, int nDegrees);
41      static void deinit(void);
42      Tetris(int cy, int cx);
43      ~Tetris();
44
45      // accessors
46      static int get_wallDepth(void)  {  return wallDepth; }
47      static int get_numTypes(void)  {  return numTypes; }
48      Matrix *get_oScreen(void) const {  return oScreen; }
49
50      // mutators
51      TetrisState accept(char key);
52  };
```

```
8   class CTetris : public Tetris {
9   private:
10      // static members
11      static Matrix ***setOfColorBlockObjects;
12
13      // dynamic members
14      Matrix *oCScreen; // outputColorScreen
15
16      int *allocColorArrayScreen(int dy, int dx, int dw);
17      void deallocColorArrayScreen(int *array);
18  protected:
19  public:
20      static void init(int **setOfColorBlockArrays, int nTypes, int
21      static void deinit(void);
22      CTetris(int cy, int cx);
23      ~CTetris();
24
25      // accessors
26      Matrix *get_oCScreen(void) const {  return oCScreen; }
27
28      // mutators
29      TetrisState accept(char key);
30  };
```

protected:

// overriding

public:

// overriding

public:

// overriding

자식 클래스에서의 생성자 함수

- 초기화 섹션 안에서 부모 클래스의 생성자 함수를 반드시 호출할 것
  - CTetris(cy,cx): Tetris(cy,cx) { ... }

```
41  CTetris::CTetris(int cy, int cx) : Tetris(cy, cx) {
42      iCScreen = new Matrix(iScreen);
43      oCScreen = new Matrix(oScreen);
44      currCBlk = NULL;
45  }
```

  - 원칙적으로, 부모 클래스의 생성자 함수들은 상속되지 않으나, 위와 같이 호출은 가능함
  - 호출하지 않으면, 부모 클래스의 디폴트 생성자가 자동 호출됨!

부모 클래스의 private 멤버들

- 부모 클래스의 private 멤버들(변수들, 함수들)은 자식 클래스에게 상속됨
- 그러나, 자식 클래스에서는 그 멤버들을 직접 접근할 수 없음
- private 멤버들은 그것이 정의된 클래스 안에서만 직접 접근할 수 있음
- 그럼에도 불구하고, private 멤버들이 상속되는 이유는, helper 역할을 수행하기 때문임

protected 키워드

- protected 멤버들은
  - 클래스 외부(에 있는 어떤 함수들) 입장에서는 private 멤버로 취급됨
  - 자식 클래스 입장에서는 멤버 이름으로 직접 접근할 수 있음
  - 정보 은닉의 원칙을 일부 위반했다고 보는 관점이 있음

자식 클래스의 멤버 함수들

- 부모 클래스의 멤버 함수들을 재정의(redefine/override)하거나
  - 이 경우에는 부모의 멤버 함수는 더 이상 사용되지 않으나, 자식 클래스(의 동일한 멤버 함수) 안에서 호출될 수 있음
  - 예: Tetris::accept() → CTetris::accept() { _state = Tetris::accept(); }
- 새로운 멤버 함수들을 추가할 수 있음

Redefining vs. Overloading

- 두 가지는 비슷해 보이나, 사실 다르다.
- 자식 클래스에서의 함수 재정의
  - 부모 멤버 함수와 동일한 파라미터 리스트를 갖는다.
  - 함수 body 를 새로 작성하는 것이다.
- 함수 오버로딩:
  - 각 함수는 서로 다른 파라미터 리스트를 갖는다. 따라서 서로 다른 함수 시그니처를 갖는다.
    - 함수 시그니처는 함수 이름, 함수 인자들의 타입의 리스트로 구성된다.
    - 함수 시그니처에 리턴 타입, const, & 는 포함되지 않는다.

상속되지 않는 함수들

- 부모의 생성자 함수들
- 부모의 소멸자 함수
  - 자식의 소멸자 호출되면, 부모의 소멸자도 자동 호출됨 (명시적으로 호출할 필요 없음)
  - 자식의 소멸자는 자식이 추가 정의한 포인터들에 관해서만 소멸 작업을 진행하면 됨
- 부모의 복사 생성자
  - 자식 입장에서 정의하지 않으면, 컴파일러가 디폴트로 생성함 (shallow copy)
  - 그러나, 자식 클래스 안에서 호출할 수 있음

```
Derived::Derived(const Derived& Object) : Base(Object), ... {...}
```

- 할당(=) 연산자 함수
  - 자식 입장에서 정의하지 않으면, 컴파일러가 디폴트로 생성함 (shallow copy)
  - 그러나, 자식 클래스 안에서 호출할 수 있음

```
Derived& Derived::operator=(const Derived & rightSide)
{  Base::operator=(rightSide);    ... }
```

소멸자와 생성자 호출 순서

- 클래스 A 를 상속받아 B 를 정의하고, B 를 상속받아 C 를 정의한다면,
  - A → B → C
- 객체 C 가 소멸될 때
  - C 의 소멸자 → B 의 소멸자 → A 의 소멸자 순으로 호출된다.
- 객체 C 가 생성될 때
  - A 의 생성자 → B 의 생성자 → C 의 생성자 순으로 호출된다.

"Is a" vs. "Has a"

- 상속
  - CTetris “is a” Tetris.
- 멤버 변수들
  - CTetris “has a” oCScreen (Matrix object).

public/protected/private 상속

class CTetris : public/protected/private Tetris { ... }

- public 키워드
  - 부모의 public 멤버들을 자식의 public 멤버들로 삼는다.
- protected 키워드
  - 부모의 public 멤버들을 자식의 protected 멤버들로 삼는다.
- private 키워드
  - 부모의 모든 멤버들을 자식의 private 멤버들로 삼는다.

다중 상속

- 자식 클래스는 두 개 이상의 부모 클래스로부터 상속받을 수 있다.
  - class derivedMulti : public base1, base2 {…}
- 모호성의 문제가 발생할 가능성이 매우 높다.
  - 두 부모 클래스가 같은 이름의 변수들을 사용한다면?
  - 두 부모 클래스가 같은 시그니처의 함수들을 사용한다면?
- 그래서, 다중 상속을 권장하지 않는다.