제14장

상속

학습 목표

- 상속의 기초
  - 파생 클래스
  - 파생 클래스의 생성자
  - protected: 제한자
  - 멤버 함수의 재정의
  - 상속되지 않는 함수들
- 상속을 이용한 프로그래밍
  - 할당 연산자와 복사 생성자
  - 파생 클래스의 소멸자
  - 다중 상속

상속의 개요

- 객체 지향 프로그래밍
  - 강력한 프로그래밍 기술인 상속이라는 추상화의 관점을 제공

- 일반화된 형태의 클래스가 정의
  - 특화된 버전의 클래스가 일반 클래스의 특성을 상속.
  - 적절한 사용을 위해 함수에 추가하거나 함수를 수정.

- 상속!
  - 새로운 클래스가 다른 클래스로부터 상속된다.
  - 기반 클래스: 다른 클래스가 파생되는 “일반화” 클래스
  - 파생 클래스: 새로운 파생된 클래스
    - 자동적으로 기반 클래스의 멤버변수와 멤버함수를 상속받음
    - 추가적인 멤버 함수와 변수를 추가할 수 있음

파생 클래스

- 예제: 클래스 "Employees"
  - 정규직 직원 또는 시간제 직원
  - 각각은 직원의 “부분 집합”.
    - 직원은 주 또는 월 단위의 고정 임금을 받을 수 있음
  - 포괄적인 종류의 “직원”이 “필요”하지는 않다.
    - 누구도 단순히 “직원”이 아니기 때문이다.
  - 하지만! 직원의 일반적인 개념이 도움이 된다!
    - 모두 이름과 주민등록번호를 가지고 있으므로, “기반”과 관련된 함수들은 직원 모두에게 같이 적용
  - 그래서 “일반” 클래스는 직원에 관한 이 모든 “것들”을 가지고 있다.

Employee 클래스

- “employee” 클래스의 많은 멤버들은 모든 종류의 직원에 적용
  - accessor 함수
  - mutator 함수
  - 대부분의 데이터 항목: SSN / Name / Pay
- 그러나 이 클래스형 “객체”는 존재하지 않음

- printCheck() 함수
  - 파생 클래스에서 항상 재정의
  - 다른 종류의 직원은 다른 종류의 수표를 가질 수 있음
    - “구별되지 않는” 직원에게는 실제로 의미가 없다.
  - Employee 클래스에서 printCheck() 함수는 다음과 같이 동작.
    - 오류 메시지가 “printCheck이 구별되지 않는 직원에 대해 호출되었다!! 중지…” 라고 말한다.

디스플레이 14.3 파생 클래스 HourlyEmployee의 인터페이스 (1 of 2)

디스플레이 14.3 **파생 클래스 HourlyEmployee의 인터페이스**

```
1
2   //이것은 헤더 파일 hourlyemployee.h이다.
3   //이것은 클래스 HourlyEmployee를 위한 인터페이스이다.
4   #ifndef HOURLYEMPLOYEE_H
5   #define HOURLYEMPLOYEE_H

6   #include <string>
7   #include "employee.h"

8   using std::string;

9   namespace SavitchEmployees
10  {
```

#ifndef 구조

employee.h를 포함

디스플레이 14.3 파생 클래스 HourlyEmployee의 인터페이스 (2 of 2)

자동으로 Employee의
모든 멤버 변수와 함수를 상속

```
11      class HourlyEmployee : public Employee
12      {
13      public:
14          HourlyEmployee( );
15          HourlyEmployee(string theName, string theSsn,
16                                 double theWageRate, double theHours);
17          void setRate(double newWageRate);
18          double getRate( ) const;
19          void setHours(double hoursWorked);
20          double getHours( ) const;
21          void printCheck( );
22      private:
23          double wageRate;
24          double hours;
25      };

26  }//SavitchEmployees

27  #endif //HOURLYEMPLOYEE_H
```

새로운 또는 "재정의된" 멤버들만을 나열

만약 함수의 정의를 변경하고 싶으면
멤버 함수를 정의하면 된다.

상속 용어

- 가족 관계를 시뮬레이션하는 것과 같다.
  - 부모 클래스: 기반 클래스
  - 자식 클래스: 파생 클래스
  - 조상 클래스: 부모의 부모의 … 부모 클래스
  - 자손 클래스: 조상 클래스의 반대

파생 클래스의 생성자

- 기반 클래스 생성자는 파생 클래스에 상속되지 않음!!!
  - 그러나 파생 클래스 생성자에서 호출 가능
- 기반 클래스 생성자는 모든 기반 클래스 멤버 변수를 초기화.
  - 파생 클래스 생성자는 간단히 기반 클래스 생성자를 호출한다.
    - 파생 클래스 생성자가 하는 "첫 번째" 일

```
HourlyEmployee::HourlyEmployee(string nname, string nnum, double nw, double nh)
                                : Employee(nname, nnum), wageRate(nw), hours(nh)
{
        //Deliberately empty
}
```

```
HourlyEmployee::HourlyEmployee() : Employee(),  wageRate(0), hours(0)
{
        //Deliberately empty
}
```

생성자: 기반 클래스 호출이 없을 때

- 파생 클래스는 기반 클래스 생성자 중의 하나를 항상 호출해야.
- 호출하지 않는다면:
  - 디폴트 기반 클래스 생성자가 자동으로 호출된다.
- 동등한 생성자 정의:

```
HourlyEmployee::HourlyEmployee() : wageRate(0), hours(0)
{ }
```

함정: 기반 클래스에서 private 멤버의 사용

- 파생 클래스는 기반 클래스의 private 멤버를 “상속한다”.
  - 하지만, 직접 접근할 수 없고, 파생클래스 멤버 함수를 통해서도 접근 불가
    - private 멤버는 그들이 정의된 멤버 함수에서만 “이름”으로 접근
  - 멤버 변수는 accessor 또는 mutator 멤버 함수를 통하여 간접적으로 접근
  - 멤버 함수는 불가능
    - private 멤버 함수는 단순히 “협조자” 함수이므로 정의된 곳에서만 사용되어야

protected: 제한자

- 클래스 멤버의 새로운 분류
- 파생 클래스에서 “이름으로” 접근 허용.
  - 다른 클래스에서는 “이름으로” 접근 불가

- 정의된 클래스에서 → private처럼 동작한다.
- 파생 클래스에서의 “protected”.
  - 추가적인 파생을 허용한다

  - 많은 사람들이 이것은 정보 은닉을 “위반”한다고 판단

멤버 함수의 재정의

- 파생 클래스의 인터페이스:
  - 새로운 멤버 함수의 정의를 가지고 있음
  - 또한 상속된 멤버 함수가 변경되는 선언을 포함
  - 상속된 멤버 함수를 선언하지 않음
    - 자동으로 상속되고 변경되지 않음
- 파생 클래스의 구현은:
  - 새로운 멤버 함수를 정의하고,
    상속된 함수를 재정의

재정의 대 오버로딩

- 매우 다르다!
- 파생 클래스의 재정의:
  - 같은 매개변수 목록
  - 근본적으로 같은 함수를 “다시 작성”한다.
- 오버로딩:
  - 다른 매개변수 목록
  - 다른 매개변수를 가지는 “새로운” 함수를 정의한다.
  - 오버로딩된 함수는 다른 시그니처를 가져야 한다.

  - signature(시그니처)
    - 함수의 이름, 매개변수 목록에서 일련의 형(순서, 수, 형을 포함)
    - 반환 형, const 키워드, & 는 시그니처로 취급 하지 않음

재정의된 기반 클래스 함수에 대한 접근

- 파생 클래스에서 재정의되었을 때
  기반 클래스의 정의를 잃은 것은 아님
- 다음과 같이 사용할 수 있다:

```
Employee          JaneE;
HourlyEmployee SallyH;
JaneE.printCheck();  → calls Employee's
                                    printCheck function
SallyH.printCheck();  → calls HourlyEmployee
                                    printCheck function
SallyH.Employee::printCheck();  → Calls Employee's
                                    printCheck function!
```

- 전형적이지는 않지만, 가끔 유용.

상속되지 않는 함수들

- 기반 클래스의 모든 “일반적”함수들은 파생 클래스에 상속.
- 예외:
  - 생성자 (앞에서 언급)
  - 소멸자
  - 복사 생성자
    - 그러나 정의되지 않으면, “디폴트” 복사 생성자를 만든다.
    - 포인터에 대해서는 정의할 필요가 있다는 것을 상기하자!
  - 할당 연산자
    - 정의되지 않으면 → 디폴트

할당 연산자와 복사 생성자

- 상기: 오버로딩된 할당 연산자와 복사 생성자는 상속되지 않는다.
  - 그러나 파생 클래스 정의에서 사용할 수 있다.
  - 정형적으로 사용되어야 한다!
  - 파생 클래스 생성자가 기반 클래스 생성자를 호출하는 방법과 비슷하다.

상속된 클래스에서의 할당 연산자

- 상속되지 않는 기반클래스의 할당 연산자를 명시적으로 호출
  - 예: “Base”로부터 파생된 “Derived”가 주어질 때

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

  - 단계1: 기반 클래스에서 할당 연산자를 호출한다.
    - 이것은 모든 상속된 멤버 변수를 돌본다.
  - 단계2: 파생 클래스에서 새로운 변수를 설정한다.

상속된 클래스에서의 복사 생성자

- 상속되지 않는 기반클래스의 복사 생성자를 명시적으로 호출

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

  - : 이후가 기반 복사 생성자의 호출이다.
    - 생성된 파생 클래스 객체의 상속된 멤버 변수를 설정
    - 객체는 Derived 형이다: 또한 Base 형이기도 하므로, 인자가 적법함

파생 클래스의 소멸자

- 기반 클래스 소멸자가 올바르게 동작하면
  - 파생 클래스 소멸자를 작성하는 것은 쉽다.
- 파생 클래스 소멸자가 호출되면:
  - 자동으로 기반 클래스 소멸자를 호출한다!
  - 그래서 명시적인 호출이 필요 없다.
- 그래서 파생 클래스 소멸자는 파생 클래스 변수에만 관련.
  - 파생클래스의 모든 데이터
  - 기반 클래스 소멸자는 자동으로 상속된 데이터를 다룸
- 소멸자 호출 순서
  - 클래스 B는 클래스 A로부터 파생되고, 클래스 C는 클래스 B로부터 파생

    A ← B ← C
  - 클래스 C 소멸자가 첫 번째로 호출되고, 클래스 B 소멸자가 호출되고, 마지막으로 클래스 A 소멸자가 호출 (생성자가 호출되는 순서의 역순)

“is a" 대 “has a" 관계

- 상속 관계는 “is a" 클래스 관계여야 함.
  - 예, HourlyEmployee는 Employee”이다”.(“is a”)
  - Convertible은 Automobile ”이다”.(“is a”)

- 클래스는 멤버 데이터로 다른 클래스의 객체를 가진다면 “has a" 클래스 관계여야 함
  - 예, 어떤 클래스가 데이터로 다른 클래스의 객체를 “가진다”. (“has a”)

protected와 private 상속

- 새로운 상속 “형태“: 둘 다 드물게 사용.
- protected 상속:

```
class SalariedEmployee : protected Employee
{…}
```

- private 상속:

```
class SalariedEmployee : private Employee
{…}
```

기반 클래스에서 접근 지시자	상속 유형 (파생 클래스 정의에서 클래스 이름 뒤에 오는 지시자)		
	public	protected	private
public	public	protected	private (멤버함수와 friend 정의 내에서만 이름으로 접근 가능)
protected	protected	protected	private(멤버함수와 friend 정의 내에서만 이름으로 접근 가능)
private	파생클래스에서 이름으로 직접 접근 불가		

다중 상속

- 파생 클래스는 하나 이상의 기반 클래스를 가질 수 있다!
  - 모든 기반 클래스를 쉼표로 구분한다:

    class derivedMulti : public base1, base2

    {…}
- 모호성이 끝이 없다! 위험한 일!
  - 몇몇 사람들은 결코 사용하면 안 된다고 믿는다.
  - 확실히 경험이 많은 프로그래머가 사용해야 한다!

요약

- 상속은 코드 재사용을 제공한다.
  - 한 클래스가 다른 클래스로부터 “파생”되고 자질을 추가한다.
- 파생 클래스 객체는 기반 클래스의 멤버를 상속하며 멤버를 추가할 수 있다.
  - 기반 클래스의 private 멤버 변수는 파생 클래스에서 “이름으로” 접근 불가
  - private 멤버 함수는 상속되지 않는다.
  - 상속된 멤버 함수를 재정의할 수 있다.
- 기반 클래스의 protected 멤버:
  - 파생 클래스 멤버 함수에서 “이름으로” 접근할 수 있다.
- 오버로딩된 할당 연산자는 상속될 수 없다.
  - 그러나 파생 클래스에서 호출될 수 있다.
- 생성자는 상속되지 않는다.
  - 파생 클래스의 생성자로부터 호출된다.