03_상속

❖ 상속

- 클래스 사이의 데이터와 연산을 공유하기 위한 메커니즘
- 기존 클래스로부터 상속을 통해 정의되는 클래스는 기존 클래스의 데이터와 연산 기능을 가지며 나아가 새로운 데이터와 연산을 추가할 수 있음

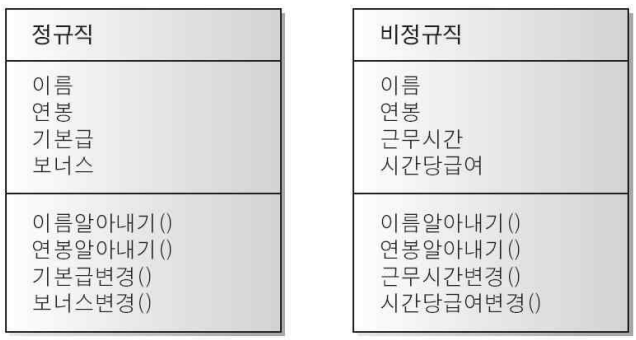
❖ 상위 클래스와 하위 클래스

- 상속을 통해 정의되는 클래스를 하위 클래스(subclass) 또는 파생 클래스(derived class)
- 상속해 주는 클래스를 상위 클래스(superclass), 기반 클래스(base class) 또는 부모 클래스(parent class)

1

03_상속

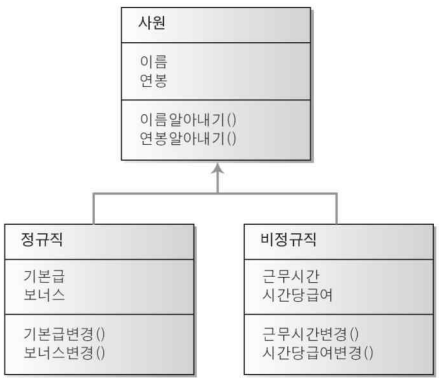
- 예를 통한 이해
  - 정규직과 비정규직을 구분



2

03_상속

- 공통된 사항들을 묶어 사원이라는 클래스를 정의하고, 정규직과 비정규직은 사원 클래스를 상속하여 정의
- 사원 클래스를 상속받는 정규직과 비정규직 클래스



- 정규직과 비정규직 클래스는 사원 클래스의 이름, 연봉, 이름알아내기(), 연봉알아내기()를 상속받아 자신의 데이터와 연산인 것처럼 사용할 수 있음

3

03_상속

❖ C++ 상속

- 상속을 이용하여 파생 클래스를 정의하는 형식

```
class 파생클래스 이름 : 접근모드 기반 클래스 이름 {
            :
};
```

- 접근 모드로 사용 가능한 경우 : public, private, protected
- 접근 모드 : public
  - ➢ 기반 크래스의 모든 멤버에 대한 접근 권한이 파생 클래스에서도 그대로 유지됨

4

03_상속

- 기반 클래스를 상속하는 파생 클래스 정의 예

```
class employee {
private:
   int annualsalary;
     ⋮
public:
   char *getName( ) { … }
     ⋮
};

class permanent : public employee {
private:
   int bonus;
     ⋮
public:
   void changeBasicSalary( ) { … }
     ⋮
};
```

5

03_상속

- 기반 클래스를 상속하는 파생 클래스 정의-오류 발생

```
class employee {
private:
   int annualsalary;
     ⋮
};
class permanent : public employee {
     ⋮
   void changeBasicSalary( ) {
      annualsalary = … ;   // 오류가 발생한다.
   }
};
```

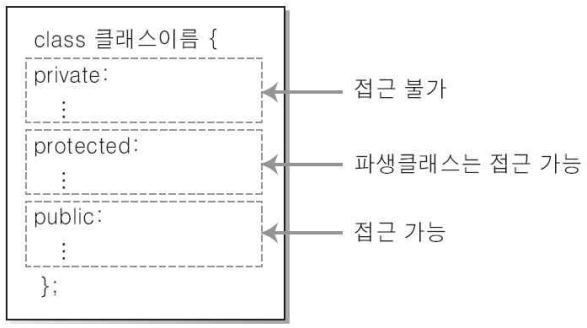
- 파생 클래스인 permanent에서 employee 클래스의 private 멤버인 annualsalary를 접근하려는 코드는 오류가 발생

- 기반 클래스의 모든 멤버는 파생 클래스로 상속 가능
- 기반 클래스의 private 멤버에 대해서는 파생 클래스에서 접근 불가, public 멤버에 대해서만 접근 가능

6

03_상속

- **protected**
  - **파생 클래스에서는 접근이 가능하지만 외부에서는 접근이 불가능한 멤버를 지정**



[그림 9-11] private, protected, public 멤버

7

03_상속

- **기반 클래스를 상속하는 파생 클래스 정의-오류 수정**

```
class employee {
protected:
   int annualsalary;
     ⋮
};
class permanent : public employee {
     ⋮
   void changeBasicSalary( ) {
      annualsalary = … ;   // 접근 가능!!
   }
};
```

8

03_상속

❖ 프로그램

▪ C++로 구현한 employee 클래스

```
class employee {
private:
  char *name;
protected:
  int annualsalary;
public:
  employee(char *na, int as) {
    name = new char[strlen(na)+1];
    strcpy(name, na);
    annualsalary = as;
  }
 ~employee() {
    delete []name;
  }
  char* getName() {
    return name;
  }
  int getAnnualSalary() {
    return annualsalary;
  }
};
```

*private
-. name : 파생 클래스에서 직접 접근하지 않음

*protected
-. annualsalary: 외부에서 접근하지 않으나 파생 클래스에서 직접 접근함

*public
getName(), getAnnualSalary : 외부에서 직접 접근

9

03_상속

▪ C++로 구현한 permanent 클래스

```
class permanent : public employee {
private:
  int basicsalary;
  int bonus;
public:
  permanent(char *na, int bs, int bon, int as) : employee(na, as) {
    bonus = bon;
    basicsalary = bs;
  }
  void changeBasicSalary(int bs) {
    basicsalary = bs;
    annualsalary = basicsalary*12 + basicsalary*bonus*0.01;
  }
  void changeBonus(int bon) {
    bonus = bon;
    annualsalary = basicsalary*12 + basicsalary*bonus*0.01;
  }
};
```

*멤버데이터
-. basicsalary, bonus

*멤버함수
-. changeBasicSalary(), changeBonus()

*annualsalary
protected 멤버로서, 각 멤버함수에서 annualsalary에 대해 직접 접근 가능

10

03_상속

- C++로 구현한 temporary 클래스

```
class temporary : public employee {
private:
  int workhours;
  int hoursalary;
public:
  temporary(char *na, int hs, int wh, int as) : employee(na, as) {
    workhours = wh;
    hoursalary = hs;
  }
  void changeWorkHours(int wh) {
    workhours = wh;
    annualsalary = workhours * hoursalary / 10000
  }
  void changeHourSalary(int hs) {
    hoursalary = hs;
    annualsalary = workhours * hoursalary / 10000;
  }
};
```

*멤버데이터
-. workhours, hoursalary

*멤버함수
-. changeWorkHours(), changeHourSalary()

11

03_상속

- permanent 클래스의 객체인 emp1을 생성

```
permanent emp1( “aaa” , 150, 400, 2400);
```

- emp1 객체의 name과 annualsalary를 알아내는 문장

```
cout << emp1.getName() << “  “ << emp1.getAnnualSalary();
```

- emp1 객체의 기본급을 200으로 변경

```
emp1.changeBasicSalary(200);
```

- temporary 클래스의 객체인 emp2를 생성하고, 이름과 연봉을 출력하고, 근무시간을 변경

```
temporary emp2("bbb", 5000, 2000, 1000);
cout << emp2.getName() << " " << emp2.getAnnualSalary();
emp2.changeWorkHours(2500);
```

12

03_상속

- 다중 상속의 예

```
class A {
    :
};
class B {
    :
};
class C : public A, public B {
    :
};
```

- C++의 경우, 파생 클래스가 두 개 이상의 기반 클래스를 동시에 상속 가능
  ➔ 다중 상속(multiple inheritance)

13

03_상속

❖ Java의 상속

- 상속을 이용하여 파생 클래스를 정의하는 형식

```
class 파생클래스 이름 extends 기반 클래스 이름 {
        :
};
```

- 기반 클래스를 상속하는 파생 클래스 정의

```
class employee {
  protected int annualsalary;
    :
  public String getName ( ) { … }
    :
}

class permanent extends employee {
  private int bonus;
    :
  public void changeBasicSalary( ) { … } {
    :
}
```

14

03_상속

- 기반 클래스를 상속하는 파생 클래스 정의-오류 발생

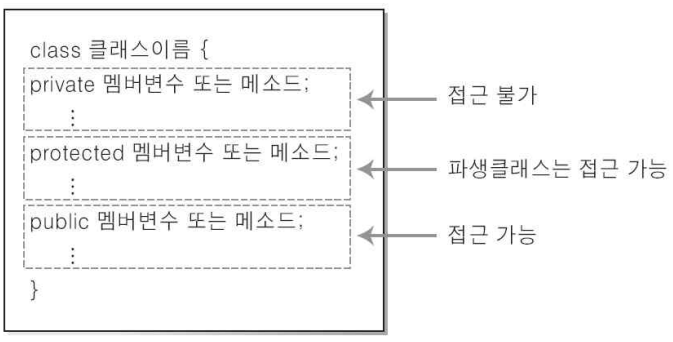
```
class employee {
  private int annualsalary;
    ⋮
}

class permanent extends employee {
    ⋮
  public void changeBasicSalary( ) { … } {
    annualsalary = … ;  // 오류 발생!
  }
}
```

- 파생 클래스인 permanent에서 getName()에는 접근할 수 있으나, private 멤버인 annualsalary를 접근하려는 코드는 오류가 발생
- 기반 클래스의 모든 멤버는 파생 클래스로 상속 가능
- 기반 클래스의 private 멤버에 대해서는 파생 클래스에서 접근 불가, public 멤버에 대해서만 접근 가능

15

03_상속

- protected
  - 파생 클래스에서는 접근이 가능하지만 외부에서는 접근이 불가능한 멤버를 지정



[그림 9-12] private, protected, public 멤버

16

03_상속

- 기반 클래스를 상속하는 파생 클래스 정의-오류 수정

```
class employee {
  protected  int annualsalary;
    ⋮
}

class permanent extends employee {
    ⋮
  public void changeBasicSalary( ) { … } {
    annualsalary = … ;  // 접근 가능!!
  }
}
```

17

03_상속

❖ 프로그램

- 자바로 구현: employee 클래스

```
class employee {
  private String name;
  protected int annualsalary;

  public employee(String name, int annualsalary) {
    this.name = name;
    this.annualsalary = annualsalary;
  }
  public String getName() {
    return name;
  }
  public int getAnnualSalary() {
    return annualsalary;
  }
}
```

- C++ 소스와 유사
- 소멸자가 없다는 점이 큰 차이점

18

03_상속

- 자바로 구현: permanent 클래스

```
class permanent extends employee {
  private int basicsalary;
  private int bonus;

  public permanent(String name, int basicsalary, int bonus, int annualsalary) {
    super(name, annualsalary); //기반 클래스 employee의 생성자인 employee()를 호출
    this.bonus = bonus;
    this.basicsalary = basicsalary;
  }
  public void changeBasicSalary(int basicsalary) {
    this.basicsalary = basicsalary;
    annualsalary = (int) (basicsalary * 12 + basicsalary*bonus*0.01);
  }
}
```

19

03_상속

- 자바로 구현: temporary 클래스

```
class temporary extends employee {
  private int workhours;
  private int hoursalary;

  public temporary(String name, int hoursalary, int workhours, int annualsalary){
    super(name, annualsalary);
    this.workhours = workhours;
    this.hoursalary = hoursalary;
  }
  public void changeWorkHours(int workhours) {
    this.workhours = workhours;
    annualsalary = workhours * hoursalary / 10000;
  }
  public void changeHourSalary(int hoursalary) {
    this.hoursalary = hoursalary;
    annualsalary = workhours * hoursalary / 10000;
  }
}
```

20

03_상속

- permanent 클래스의 객체인 emp1을 생성

```
permanent emp1 = new permanent( “aaa” , 150, 400, 2400);
```

- emp1 객체의 name과 annualsalary를 알아내는 문장

```
System.out.println(emp1.getName() +  “ ” + emp1.getAnnualSalary();
```

- emp1 객체의 기본급을 200으로 변경

```
emp1.changeBasicSalary(200);
```

- temporary 클래스의 객체인 emp2를 생성하고, 이름과 연봉을 출력하고, 근무시간을 변경

```
temporary emp2 = new temporary("bbb", 5000, 2000, 1000);
System.out.println(emp2.getName() +  “ ” + emp2.getAnnualSalary();
emp2.changeWorkHours(2500);
```

21

04_동적 바인딩

❖ 동적 바인딩

- 상속 관계가 있는 객체에 대한 멤버 함수 호출에 대해 대응되는 멤버 함수가 동적으로 결정되는 것을 의미

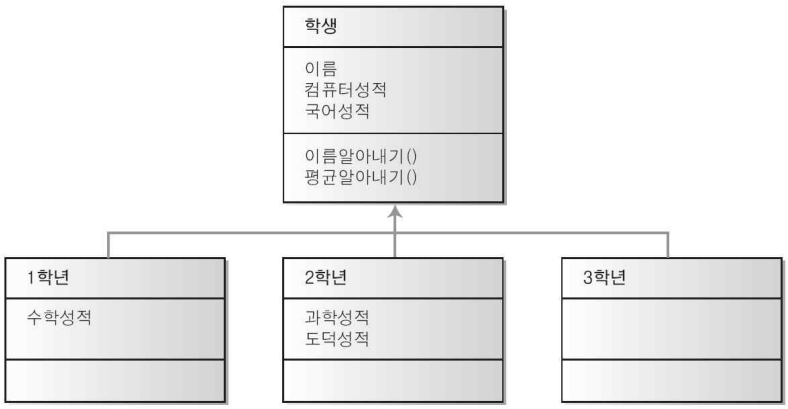
- 예를 통한 이해
  - 학생 클래스

➢1학년과 2학년은 추가로 이수할 과목 추가됨

22

04_동적 바인딩

- **학생 클래스를 상속하는 파생 클래스**
  - 학년 클래스를 상속하는 1학년, 2학년, 3학년 파생 클래스 생성
  - 1학년 클래스 : 수학 성적 추가
  - 2학년 클래스 : 과학 성적과 도덕 성적 추가

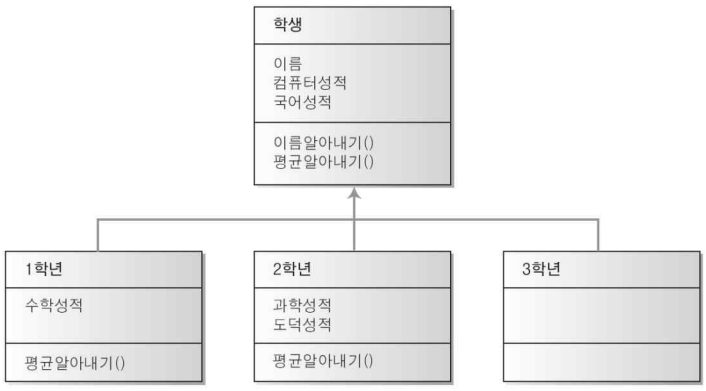


[그림 9-14] 학생 클래스를 상속하는 파생 클래스

23

04_동적 바인딩

- **1학년과 2학년 클래스에 평균알아내기() 함수 정의**



- 1학년 또는 2학년 클래스의 객체에 대해 평균 알아내기 함수를 호출하면 파생 클래스에서 새롭게 정의된 함수가 동작하고, 3학년 클래스의 객체에 대해 평균 알아내기 함수를 호출하면 기반 클래스에서 정의된 함수가 동작
- 상속 관계가 있는 객체에 대해 멤버 함수를 호출했을 때 동작하게 될 멤버 함수가 동적으로 결정되는 개념 ➔ **동적 바인딩(dynamic binding)**

24