클래스의 상속

(Class's inheritance)

들어가기 전에

- 오늘의 소스코드를 받아봅시다.
  - 중요한 코드는 꼭 다른 곳으로 복사해두세요.
  - phycom2015-2 디렉토리에서,
    - git stash
    - git pull
  - 디렉토리가 사라졌다면,
    - git clone https://github.com/geonmo/phycom2015-2.git

지금까지의 내용

- 변수란 무엇인가?
  - 변수와 상수
- 제어문이란?
  - 조건문과 반복문
- 함수란?
  - 한 로직을 처리하기 위한 프로그램 코드의 묶음
- 자료구조란?
  - 데이터를 저장/관리하는 가장 효율적인 방법들
- 포인터란?
  - 변수 "자체"를 가리키는 변수
- 클래스란?
  - 멤버변수 + 멤버함수 => 객체를 만들어내는 사용자지정 데이터형

Class의 상속

- 기존에 존재하는 클래스를 이용하여 자식 클래스를 만드는 방법
  - 포함 관계의 클래스를 만들 때 아주 좋음
    - 예) 학생 – 초등학생, 중학생, 고등학생, 대학생
      - 공통 멤버변수 : 이름, 나이
      - 공통 멤버함수 : 공부
      - 별도 멤버함수 : (대학생 전용)drinkAlchol()
    - 예2) 근로자 – 정직원, 임시직원, 시급 아르바이트
      - 공통 멤버변수 : (string)담당업무
      - 공통 멤버함수 : doWork(),
      - 별도 멤버변수 : (아르바이트 전용) int worktime
      - 별도 멤버함수 : (임시직원용)extendContract(),
(각자) calculateSalary()

사용방법

- Class의 선언부분
  - 클래스 선언시 부모클래스 명시
    - public 키워드 사용시 public까지 상속
    - protected면 public->protected로 상속
    - private라면 public,protected->private로 상속

```
class ChildClass : public ParentClass
```

- Class의 정의부분
  - 부모 클래스의 생성자로
    일부의 변수값 초기화 가능

```
ChildClass::ChildClass(string name, int pay_per_hour) : ParentClass(name)
```

살펴보기

- 프로그램 소개
  - 어떤 회사 사장님이 직원들에게 줄 월급을 계산하기 위한 프로그램 작성
- Step0
  - Employee 클래스로 직원을 클래스(분류화)함
    - 직원의 이름, 일한 날짜, 임금 날짜를 멤버변수
    - doWork, goVacation 등으로 실제 일을 시키거나 (유상)휴가를 보냄
  - 프로그램 이름 : SalaryManager

살펴보기

- Step0 [Employee.h]

```
#include<iostream>
#include<string>

using namespace std;
class Employee {

private:
  string name_;
  string job_title_;
  int day_of_service_;
  int salary_day_;
public:
  Employee(string name, string title);
  void doWork();
  void goVacation();
  void receiveSalary();
  int calculateSalary();
};
```

살펴보기

- Step0 [프로그램 실행]

```
직원을 고용합니다. : 이름(박철수) 업무(경리)
경리 업무를 진행합니다.
경리 업무를 종료합니다.
경리 업무를 진행합니다.
경리 업무를 종료합니다.
경리 업무를 진행합니다.
경리 업무를 종료합니다.
경리 업무를 진행합니다.
경리 업무를 종료합니다.
휴가를 갑니다.
업무로 복귀합니다.
박철수은/는 봉급으로 400000원을 받았습니다.
```

살펴보기

- Step1
  - Employee 클래스로부터 PartTime클래스를 상속하여 아르바이트 직원 또한 관리할 수 있도록 작성
    - 아르바이트생들을 일급이 아니라 시급을 받기 때문에 날짜가 아니라 시간 정보를 가지고 있어야 하고 (유급)휴가가 없기 때문에 이를 제외하도록 작성

살펴보기

- Employee vs PartTime

멤버	상속여부
Name	직원의 이름은 아르바이트 여부와 관계 없음으로 그대로 상속
Job_title	직원의 업무 또한 아르바이트 여부와 상관 없음으로 그대로 상속해도 되나 아르바이트생은 "잡무"만 처리하므로 생성자 변경
Day_of_service	상속X. 새로운 변수 time of services생성
doWork	상속 후 오버라이딩(덮어쓰기)해야함. 기능이 달라짐
goVacation	오버라이딩
receiveSalary	기능 변화 없음. 그대로 상속.
calculateSalary	오버라이딩

살펴보기

- Employee and PartTime

```
#include<iostream>
#include<string>

using namespace std;
class Employee {

private:
  string name_;
  string job_title_;
  int day_of_service_;  // unable variable for part-time
job
  int salary_day_;      // unable variable for part-time
job
public:
  Employee(string name);
  Employee(string name, string title);
  virtual void doWork();
  virtual void goVacation();    // part-timer can not go
vacation.
  void receiveSalary();
  virtual int calculateSalary();

};
```

```
#include"Employee.h"
#include<iostream>
using namespace std;
class PartTime : public Employee{

private :
  int time_of_service_;
  int pay_per_hour_;
  int salary_time_;
public :
  PartTime(string name, int pay_per_hour);
  void doWork();
  void doWork(int hour);
  void goVacation(){ cout<<"아르바이트생은 유
상휴가가 존재하지 않습니다."<<endl; }
  int calculateSalary();
};
```

살펴보기

- 멤버변수 상속

```
PartTime::PartTime(string name, int pay_per_hour) : Employee(name) {

}
```

  - 부모클래스의 생성자를 먼저 실행하고 이후에 자식클래스의 생성자를 실행한다고 생각하면 됨.

살펴보기

- 가상함수(Virtual function)
  - 부모클래스로부터 상속될 클래스가 형변환에 영향을 받지 않고 반드시 자식 클래스의 함수로 실행되게 설정
  - 예) 1.Employee의 doWork를 가상함수로 설정
    2. doWork를 상속하여 오버라이딩한 PartTime의 doWork를 작성
    3. PartTimeClass em2를 Employee클래스로 형변환
    4. 실제로 사용해보면 Employee 클래스의 doWork가 아니라 PartTimeClass의 doWork가 작동
  - 같은 상황에서 doWork를 가상함수로 설정하지 않으면 부모클래스의 doWork가 작동

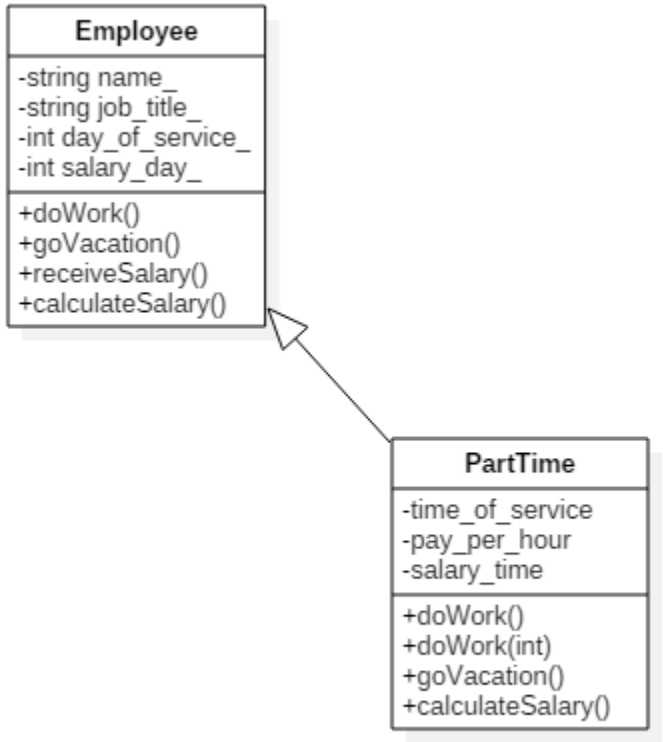
살펴보기

- 즉,
  - 상속 후 그대로 사용할 멤버함수가 아니라면 반드시 virtual키워드를 통해 가상함수로 설정할 것!

```
virtual void doWork();
virtual void goVacation();
```

살펴보기

- 클래스 다이어그램



살펴보기

- Step2
  - 부모클래스로부터 상속받은 자식클래스는 부모클래스의 배열 혹은 컨테이너에 삽입이 가능!
  - 이를 구현한 것이 Step2

살펴보기

- SalaryManager.cpp만 변경됨

```
// Test suite3.
  vector<Employee*> em_list;
  Employee em1("박철수","경리");
  PartTime em2("한영희",6000);
  em_list.push_back(&em1);
  em_list.push_back(&em2);
  em_list[0]->doWork();
  em_list[0]->doWork();
  em_list[0]->doWork();
  em_list[0]->goVacation();
  em_list[0]->receiveSalary();

  PartTime* em3 = dynamic_cast<PartTime*>(em_list[1]);
  em3->doWork();
  em3->doWork();
  em3->doWork();
  em3->doWork(4);
  em3->goVacation();
  em3->receiveSalary();

  return 0;
```

살펴보기

- dynamic_cast<>
  - 부모클래스와 자식클래스간의 상호변환이 가능한지를 판단 후 형변환을 시도
  - 부모클래스에 반드시 virtual 함수가 1개 이상 있어야 제대로 동작
  - 원래대로라면
    부모클래스->자식클래스로 변환이 되지 않으나 실제 em_list[1]이 PartTime클래스이기 때문에 형변환이 가능함

살펴보기

- Step3
  - Employee클래스를 Permanent클래스와 Temporary클래스로 분리함
  - Permanent와 Temporary클래스는 유급휴가와 무급휴가의 차이만 만듬

살펴보기

- FullTime.h and Permanent&Temporary

```
using namespace std;
class FullTime :public Employee{

private:
  int day_of_service_;
  int salary_day_;
public:
  FullTime(string name);
  FullTime(string name, string title);
  virtual void doWork();
  virtual void goVacation()=0;     // Difference
between permanent and temporary
  virtual int calculateSalary();
  int getDayOfService(){return day_of_service_; }
  int getSalaryDay(){ return salary_day_ ; }
  void setDayOfService(int day_of_service)
{ day_of_service_ = day_of_service; }
  void setSalaryDay(int salary_day ) { salary_day_ =
salary_day; }
};
```

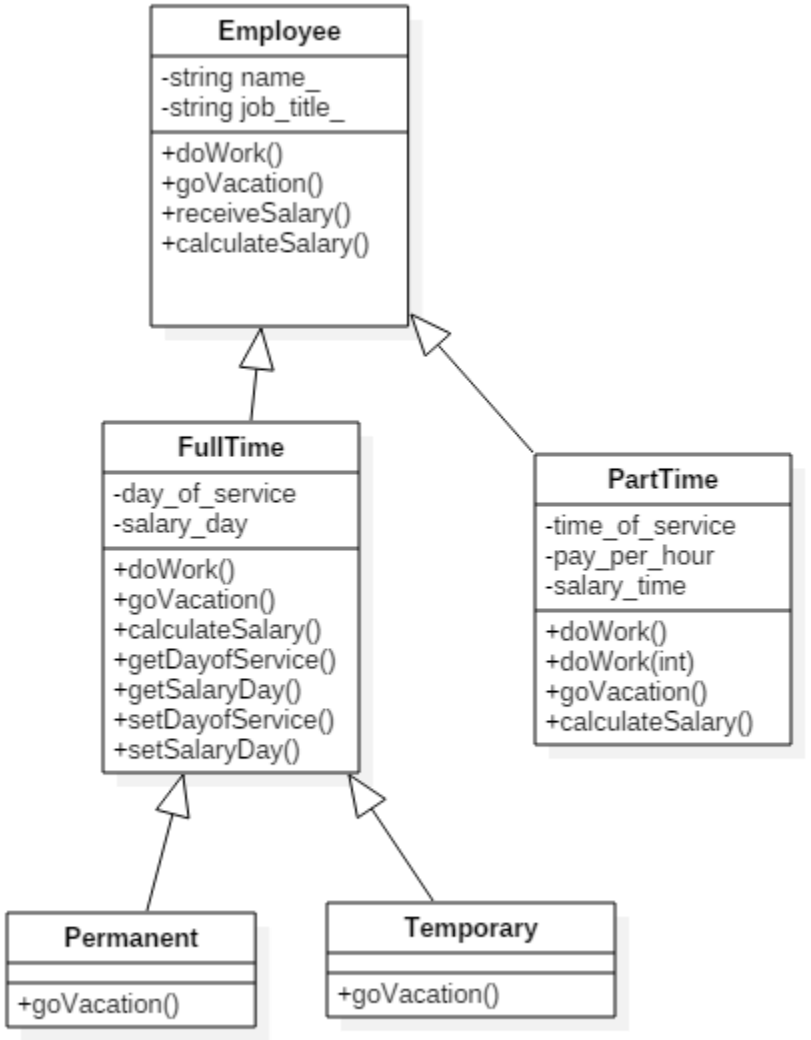
```
using namespace std;
class Permanent : public FullTime{

private:

public:
  Permanent(string name);
  Permanent(string name, string title);
  virtual void goVacation();     // Difference
between permanent and temporary
};
```

살펴보기

- 클래스 다이어그램
  - Emplyee의 날짜관련 변수들이 FullTime으로 이동
  - Permanent, Temporary 클래스의 경우 대부분의 역할을 FullTime 클래스가 담당하고 차이점만 별도로 분리되어 가짐
  - 즉, 프로그램 수정시 FullTime만 변경하면 될 가능성이 높아짐



살펴보기

- 순수가상함수
  - 부모클래스의 객체를 만들면 안될 때 객체 생성을 막기 위해 사용
  - 무조건 상속 후 오버라이딩이 되어야 할 함수의 정의를 =0 으로 설정하면 됨

```
virtual void goVacation()=0;
```

  - 위 goVacation 함수의 경우 FullTime의 함수로 적절하지 않기 때문에(FullTime 자체는 추상적인 개념) 이를 순수가상함수로 설정