304-307p

상속을 위한 조건 : IS-A 관계의 성립
(ISAInheritance.cpp 1/4)

◆ 전화기 -> 무선전화기 와 같이 내포된 성질이 같을 때 **상속**이 적절.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

class Computer
{
private:
        char owner[50];
public:
        Computer(char* name)
        {
                strcpy(owner, name);
        }
        void Calculate()
        {
                cout << "요청 내용을 계산합니다." << endl;
        }
};
```

모든 컴퓨터의 공통된 특성을 클래스로 표현.

304-307p

상속을 위한 조건 : IS-A 관계의 성립
(ISAInheritance.cpp 2/4)

```
class NotebookComp : public Computer {
private:
        int battary;
public:
        NotebookComp(char* name, int initChag)
                : Computer(name), battary(initChag)
        {  }
        void Charging() { battary += 5; }
        void UseBattary() { battary -= 1; }
        void MovingCal() {
                if (GetBattaryInfo() < 1)
                {
                        cout << "충전이 필요합니다." << endl;
                        return;
                }

                cout << "이동하면서 ";
                Calculate();
                UseBattary();
        }
        int GetBattaryInfo() { return battary; }
};
```

컴퓨터의 일종이면서 노트북의 특성을 추가로 가진 클래스.

상속을 위한 조건 : IS-A 관계의 성립
(ISAInheritance.cpp 3/4)

304-307p

```
class TabletNotebook : public NotebookComp {
private:
        char regstPenModel[50];
public:
        TabletNotebook(char* name, int initChag, char* pen)
                : NotebookComp(name, initChag) {
                strcpy(regstPenModel, pen);
        }
        void Write(char* penInfo) {
                if (GetBattaryInfo() < 1) {
                        cout << "충전이 필요합니다." << endl;
                        return;
                }
                if (strcmp(regstPenModel, penInfo) != 0) {
                        cout << "등록된 펜이 아닙니다.";
                        return;
                }
                cout << "필기 내용을 처리합니다." << endl;
                UseBattary();
        }
};
```

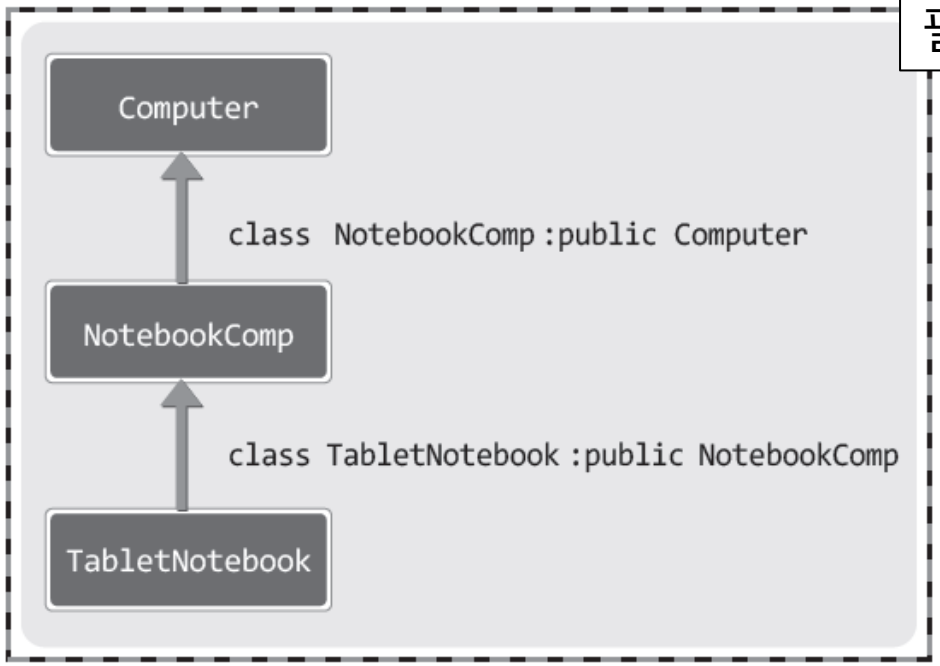
노트북의 일종이면서 태블릿의 특성을 추가로 가진 클래스.

304-307p

상속을 위한 조건 : IS-A 관계의 성립
(ISAInheritance.cpp 4/4)

```
int main(void)
{
        NotebookComp nc((char*)"이수종", 5);
        TabletNotebook tn((char*)"정수영", 5, (char*)"ISE-241-242");
        nc.MovingCal();
        tn.Write((char*)"ISE-241-242");
        return 0;
}
```

[실행결과]
이동하면서 요청 내용을 계산합니다.
필기 내용을 처리합니다.



UML 표기법:
(화살표 방향에 유의.)

HAS-A 관계
(HASComposite.cpp 1/3)

- 상속보다는 복합관계(멤버 변수로 선언하여 사용)가 적합.
- HAS-A 를 상속으로 구현하면 아래의 제약을 갖게 됨
  - 다른 성질도 표현해야 할 경우 다중상속의 복잡함이 생김.
  - 기초클래스의 성질을 갖지 않은 경우(소유 관계이므로 갖지 않은 경우도 가능함)를 표현하기가 어려움.

```
#include <iostream>
#include <cstring>
using namespace std;

class Gun {
private:
        int bullet;          // 장전된 총알의 수
public:
        Gun(int bnum) : bullet(bnum)
        { }
        void Shut() {
                cout << "BBANG!" << endl;
                bullet--;
        }
};
```

HAS-A 관계
(HASComposite.cpp 2/3)

```
class Police {
private:
        int handcuffs;    // 소유한 수갑의 수
        Gun* pistol;      // 소유하고 있는 권총
public:
        Police(int bnum, int bcuff) : handcuffs(bcuff) {
                if (bnum > 0)
                        pistol = new Gun(bnum);
                else
                        pistol = NULL;
        }
        void PutHandcuff() {
                cout << "SNAP!" << endl;
                handcuffs--;
        }
        void Shut() {
                if (pistol == NULL)
                        cout << "Hut BBANG!" << endl;
                else
                        pistol->Shut();
        }
        ~Police() {
                if (pistol != NULL)
                        delete pistol;
        }
};
```

307-311p

HAS-A 관계
(HASComposite.cpp 3/3)

```
int main(void)
{
        Police pman1(5, 3);
        pman1.Shut();
        pman1.PutHandcuff();

        Police pman2(0, 3);     // 권총 소유하지 않은 경찰
        pman2.Shut();
        pman2.PutHandcuff();
        return 0;
}
```

[실행결과]
BBANG!
SNAP!
Hut BBANG!
SNAP!

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 1/9)

314-319p

week9_oop.cpp 파일 다운로드 받아 수정하기.

- 클래스의 분류
  - Control 클래스 : 프로그램의 기능 담당
  - Entity 클래스 : 데이터 표현 담당
- Control 클래스를 추가하자! (주요기능들+전역변수들을 하나로 모으기)
  - 계좌개설 기능
  - 입금 기능
  - 출금 기능
  - 계좌정보 전체 출력 기능
  - 객체 저장을 위한 배열과 변수들
- 멤버함수의 선언만 클래스 내부에 남기고 정의는 클래스 외부로.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>

using namespace std;
const int NAME_LEN = 20;

enum { MAKE = 1, DEPOSIT, WITHDRAW, INQUIRE, EXIT };
```

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 2/9)

```
class Account
{
private:
        int accID;
        int balance;
        char* cusName;

public:
        Account(int ID, int money, char* name);
        Account(const Account& ref);

        int GetAccID() const;
        void Deposit(int money);
        int Withdraw(int money);
        void ShowAccInfo() const;
        ~Account();
};
```

멤버함수의 선언만 클래스 내부에 남기기

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 3/9)

```
Account::Account(int ID, int money, char* name)
        : accID(ID), balance(money)
{
        cusName = new char[strlen(name) + 1];
        strcpy(cusName, name);
}

Account::Account(const Account& ref)
        : accID(ref.accID), balance(ref.balance)
{
        cusName = new char[strlen(ref.cusName) + 1];
        strcpy(cusName, ref.cusName);
}

int Account::GetAccID() const { return accID; }
```

멤버함수의 정의는 클래스 외부로 이동.

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 4/9)

```
void Account::Deposit(int money) {
        balance += money;
}

int Account::Withdraw(int money) {
        if (balance < money)
                return 0;
        balance -= money;
        return money;
}

void Account::ShowAccInfo() const {
        cout << "계좌ID: " << accID << endl;
        cout << "이  름: " << cusName << endl;
        cout << "잔  액: " << balance << endl;
}

Account::~Account() {
        delete[]cusName;
}
```

멤버함수의 정의는 클래스 외부로 이동.

314-319p

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 5/9)

```
class AccountHandler
{
private:
        Account* accArr[100];
        int accNum;

public:
        AccountHandler();
        void ShowMenu(void) const;
        void MakeAccount(void);
        void DepositMoney(void);
        void WithdrawMoney(void);
        void ShowAllAccInfo(void) const;
        ~AccountHandler();
};
```

Control 클래스 생성

전역 변수들을 포함시키기.

기능 포함하기.

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 6/9)

314-319p

전역 함수들을 클래스 내부로.

```
void AccountHandler::ShowMenu(void) const {
        ……
}
void AccountHandler::MakeAccount(void) {
        ……
}
void AccountHandler::DepositMoney(void) {
        ……
}
void AccountHandler::WithdrawMoney(void) {
        ……
}
void AccountHandler::ShowAllAccInfo(void) const {
        ……
}
```

314-319p

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 9/9)

```
AccountHandler::AccountHandler() : accNum(0)
{  }

AccountHandler::~AccountHandler()
{
        for (int i = 0; i < accNum; i++)
                delete accArr[i];
}
```

Constructor,
Destructor 추가.

OOP 프로젝트 05단계
(BankingSystemVer05.cpp 10/10)

314-319p

main 함수 변경.

```
int main(void) {
	AccountHandler manager;
	int choice;
	while (1) {
		manager.ShowMenu();
		cout << "선택: ";
		cin >> choice;
		cout << endl;

		switch (choice) {
		case MAKE:	manager.MakeAccount(); break;
		case DEPOSIT:	manager.DepositMoney(); break;
		case WITHDRAW: manager.WithdrawMoney(); break;
		case INQUIRE:	manager.ShowAllAccInfo(); break;
		case EXIT:	return 0;
		default:	cout << "Illegal selection.." << endl;
		}
	}
	return 0;
}
```

AccountMnager 클래스의 destructor에서 메모리 해제하므로 여기서는 지움.

8. 상속과 다형성

9주차

326-330p

객체 포인터 : 유도 클래스 객체 주소도 OK

```
#include <iostream>
using namespace std;
class Person {
public:
        void Sleep() { cout << "Sleep" << endl; }
};
class Student : public Person {
public:
        void Study() { cout << "Study" << endl; }
};
class PartTimeStudent : public Student {
public:
        void Work() { cout << "Work" << endl; }
};
int main(void) {
        Person* ptr1 = new Student();
        Person* ptr2 = new PartTimeStudent();
        Student* ptr3 = new PartTimeStudent();
        ptr1->Sleep();
        ptr2->Sleep();
        ptr3->Study();
        delete ptr1; delete ptr2; delete ptr3;
        return 0;
}
```

[결과]
Sleep
Sleep
Study

클래스 객체도 선언 아닌 할당 가능.
포인터 타입에 맞게.

포인터 타입 : 유도클래스 객체도 대입 가능.

포인터 타입과 같은 기초 클래스의 함수들만 호출가능.
유도 클래스 함수는 호출 불가.

객체 포인터 : 유도 클래스 객체 주소도

326-330p

```
#include <iostream>
using namespace std;
class Shape {
protected:
        int height;
public:
        Shape(int _height) { height = _height; }
        int GetArea() const { return 0; }
};
class Rect : public Shape {
public:
        Rect(int _height) : Shape(_height) {}
        int GetArea() const { return height * height; }
};
class Circle : public Shape {
public:
        Circle(int _height) : Shape(_height) {}
        int GetArea() const { return height/2 * height/2 * 3.14; }
};
int main(void) {
        Shape *arr[2] = { new Rect(10), new Circle(10) };
        for (int i = 0;i < 2;i++)
                printf( "%d의 면적 : %d₩n", i, arr[i]->GetArea());
        delete arr[0]; delete arr[1];
        return 0;
}
```

330p

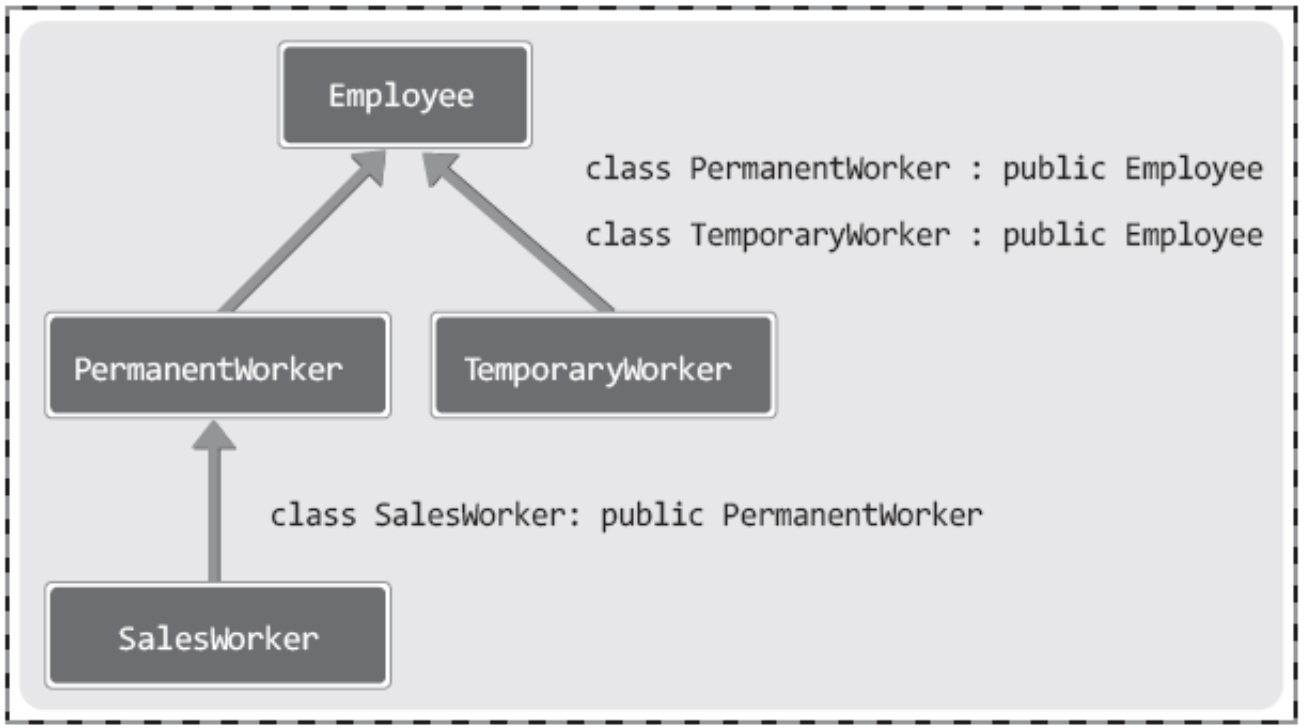
오렌지미디어 급여관리 확장성문제 1차 해결

- 고용인 Employee
- 정규직 PermanentWorker
- 영업직 SalesWorker
- 임시직 TemporaryWorker

"정규직, 영업직, 임시직 모두 고용의 한 형태이다(고용인이다)."

"영업직은 정규직의 일종이다."

Control 클래스에서 모든 Entity 클래스의 객체를 Employee 클래스의 객체로 간주(처리)할 수 있는 기반을 마련!



331-333p

오렌지미디어 급여관리 확장성문제 1차 해결
(EmployeeManager2.cpp 1/5)

- Employee 클래스 추가하여 PermanentWorker 클래스를 여기서 유도 받기
- EmployeeHandler 클래스에서는 Employee 객체를 관리하기.
  - Employee에서 유도 받은 클래스의 객체들을 다 관리할 수 있음.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

class Employee {
private:
	char name[100];
public:
	Employee(char* name) {
		strcpy(this->name, name);
	}
	void ShowYourName() const {
		cout << "name: " << name << endl;
	}
};
```

331-333p

오렌지미디어 급여관리 확장성문제 1차 해결
(EmployeeManager2.cpp 2/5)

```
class PermanentWorker : public Employee
{
private:
	int salary;
public:
	PermanentWorker(char* name, int money)
		: Employee(name), salary(money)
	{  }
	int GetPay() const
	{
		return salary;
	}
	void ShowSalaryInfo() const
	{
		ShowYourName();
		cout << "salary: " << GetPay() << endl << endl;
	}
};
```

331-333p

오렌지미디어 급여관리 확장성문제 1차 해결
(EmployeeManager2.cpp 3/5)

```
class EmployeeHandler {
private:
        Employee* empList[50];
        int empNum;
public:
        EmployeeHandler() : empNum(0) { }
        void AddEmployee(Employee* emp) {
                empList[empNum++] = emp;
        }
        void ShowAllSalaryInfo() const {
                /*
                for(int i=0; i<empNum; i++)
                        empList[i]->ShowSalaryInfo();
                */
        }
        ……
```

저장의 대상이 PermanentWorker 객체에서 Employee 객체로 바뀜.

Employee 클래스에는 없는 멤버 함수이므로 오류 발생!

오렌지미디어 급여관리 확장성문제 1차 해결
(EmployeeManager2.cpp 4/5)

331-333p

```
        .....
        void ShowTotalSalary() const {
                int sum = 0;
                /*
                for(int i=0; i<empNum; i++)
                        sum+=empList[i]->GetPay();
                */
                cout << "salary sum: " << sum << endl;
        }
        ~EmployeeHandler()
        {
                for (int i = 0; i < empNum; i++)
                        delete empList[i];
        }
};
```

Employee 클래스에는 없는 멤버 함수이므로 오류 발생!

331-333p

오렌지미디어 급여관리 확장성문제 1차 해결
(EmployeeManager2.cpp 4/5)

```
int main(void)
{
	// 직원관리를 목적으로 설계된 컨트롤 클래스의 객체생성
	EmployeeHandler handler;

	// 직원 등록
	handler.AddEmployee(new PermanentWorker((char*)"KIM", 1000));
	handler.AddEmployee(new PermanentWorker((char*)"LEE", 1500));
	handler.AddEmployee(new PermanentWorker((char*)"JUN", 2000));

	// 이번 달에 지불해야 할 급여의 정보
	handler.ShowAllSalaryInfo();

	// 이번 달에 지불해야 할 급여의 총합
	handler.ShowTotalSalary();
	return 0;
}
```

334-337p

오렌지미디어 급여관리 확장성문제 2차 해결

- 영업직과 계약직도 관리할 수 있게 Employee에서 유도 받은 클래스 2개 추가
  - TemporaryWorker 클래스
  - SalesWorker 클래스
- 관리 대상 클래스가 다양화 되었어도 EmployeeHandler 클래스는 변할 필요가 없음!

334-337p

오렌지미디어 급여관리 확장성문제 2차 해결
(EmployeeManager3.cpp 이전 소스에 추가 1/3)

```
class TemporaryWorker : public Employee {
private:
        int workTime;
        int payPerHour;
public:
        TemporaryWorker(char* name, int pay)
                : Employee(name), workTime(0), payPerHour(pay) {  }
        void AddWorkTime(int time) {
                workTime += time;
        }
        int GetPay() const {
                return workTime * payPerHour;
        }
        void ShowSalaryInfo() const {
                ShowYourName();
                cout << "salary: " << GetPay() << endl << endl;
        }
};
```

함수 overriding. (overloading과 다름!)
- 기초 클래스와 같은 함수를 재정의.
- 클래스 포인터에 의해 호출하면 유도된 조상 클래스를 따라 가까이 구현된 함수가 호출됨.

334-339p

오렌지미디어 급여관리 확장성문제 2차 해결
(EmployeeManager3.cpp 이전 소스에 추가 2/3)

```
class SalesWorker : public PermanentWorker {
private:
    int salesResult;    // 월 판매실적
    double bonusRatio;    // 상여금 비율
public:
    SalesWorker(char* name, int money, double ratio)
        : PermanentWorker(name, money), salesResult(0), bonusRatio(ratio)
    { }
    void AddSalesResult(int value) {
        salesResult += value;
    }
    int GetPay() const {
        return PermanentWorker::GetPay() + (int)(salesResult * bonusRatio);
    }
    void ShowSalaryInfo() const {
        ShowYourName();
        cout << "salary: " << GetPay() << endl << endl;
    }
};
```

PermanentWorker의 GetPay 함수 호출

함수 overriding. (overloading과 다름!)

기초 클래스 구현과 동일하지만 다시 구현한 이유 : 기초 클래스가 아닌 자신의 GetPay() 함수를 사용하기 위해서

포인터 타입에 따른 함수 호출
(FunctionOverride.cpp 1/2)

◆ 유도 클래스가 기초 클래스의 함수를 overriding 했다면
- 포인터 타입에 따라 함수가 호출됨.

```
#include <iostream>
using namespace std;

class First {
public:
        void MyFunc() {
                cout << "FirstFunc" << endl;
        }
};

class Second : public First {
public:
        void MyFunc() {
                cout << "SecondFunc" << endl;
        }
};
```

[결과]
FirstFunc
SecondFunc
ThirdFunc

339-346p

포인터 타입에 따른 함수 호출
(FunctionOverride.cpp 2/2)

```
class Third : public Second {
public:
        void MyFunc() {
                cout << "ThirdFunc" << endl;
        }
};

int main(void) {
        Third* tptr = new Third();
        Second* sptr = tptr;
        First* fptr = sptr;

        fptr->MyFunc();
        sptr->MyFunc();
        tptr->MyFunc();
        delete tptr;
        return 0;
}
```

[결과]
FirstFunc
SecondFunc
ThirdFunc

같은 객체에 대해 같은 함수를 호출했지만 포인터 타입에 따라 함수가 선택되었음.

기초클래스 포인터로는 기초클래스 함수만 사용할 수 있다면 기초 클래스포인터 한가지 타입으로만 관리할 수 없으므로 의미가 없음.
-> 가상 함수 사용!

346-348p

가상함수(Virtual Function)
(FunctionVirtualOverride.cpp)

- 기초 클래스에서 virtual로 선언된 함수는 유도 클래스에서 override해도 virtual 함수 성격을 가짐.
- virtual 함수는 호출 시 포인터 타입 대신 실제 객체 타입에 따른 호출을 하게 됨.
  - 기초 클래스 포인터만 사용하여 유도 클래스 객체들을 일괄 관리할 수 있게 됨.

```
#include <iostream>
using namespace std;
class First {
public:
        virtual void MyFunc() {
                cout << "FirstFunc" << endl;
        }
};
class Second : public First {
public:
        virtual void MyFunc() {
                cout << "SecondFunc" << endl;
        }
};
```

[결과]
ThirdFunc
ThirdFunc
ThirdFunc

346-348p

가상함수(Virtual Function)
(FunctionVirtualOverride.cpp)

```
class Third : public Second {
public:
        virtual void MyFunc() {
                cout << "ThirdFunc" << endl;
        }
};

int main(void) {
        Third* tptr = new Third();
        Second* sptr = tptr;
        First* fptr = sptr;

        fptr->MyFunc();
        sptr->MyFunc();
        tptr->MyFunc();
        delete tptr;
        return 0;
}
```

[결과]
ThirdFunc
ThirdFunc
ThirdFunc

객체가 Third 클래스 객체이고 MyFunc가 virtual함수이므로 어느 포인터를 사용해도 Third 클래스의 함수가 호출됨.

348-352p

오렌지미디어 급여관리 확장성문제 완전 해결
(EmployeeManager4.cpp 이전 소스에 추가 1/2)

- 영업직과 계약직도 관리할 수 있게 Employee에서 유도 받은 클래스 2개 추가
  - TemporaryWorker 클래스
  - SalesWorker 클래스
- 관리 대상 클래스가 다양화 되었어도 EmployeeHandler 클래스는 변할 필요가 없음!

```
class Employee {
private:
        char name[100];
public:
        Employee(char* name)
        {
                strcpy(this->name, name);
        }
        void ShowYourName() const
        {
                cout << "name: " << name << endl;
        }
        virtual int GetPay() const {
                return 0;
        }
        virtual void ShowSalaryInfo() const
        { }
};
```

기초 클래스에 (유도 클래스에서 필요함한) virtual 함수 추가.

[결과]

```
name: KIM
salary: 1000

name: LEE
salary: 1500

name: Jung
salary: 3500

name: Hong
salary: 1700

salary sum: 7700
```

348-352p

오렌지미디어 급여관리 확장성문제 완전 해결
(EmployeeManager4.cpp 이전 소스에 추가 2/2)

```
class EmployeeHandler {
private:
        Employee* empList[50];
        int empNum;
public:
        EmployeeHandler() : empNum(0) { }
        void AddEmployee(Employee* emp) {
                empList[empNum++] = emp;
        }
        void ShowAllSalaryInfo() const {
                for(int i=0; i<empNum; i++)
                        empList[i]->ShowSalaryInfo();
        }
        void ShowTotalSalary() const {
                int sum = 0;
                for(int i=0; i<empNum; i++)
                        sum+=empList[i]->GetPay();
                cout << "salary sum: " << sum << endl;
        }
        ~EmployeeHandler() {
                for (int i = 0; i < empNum; i++)
                        delete empList[i];
        }
};
```

virtual 함수 호출.

[결과]

```
name: KIM
salary: 1000

name: LEE
salary: 1500

name: Jung
salary: 3500

name: Hong
salary: 1700

salary sum: 7700
```