객체지향프로그래밍
6주차 실습 활동지

이름 : 김태석
학번 : 201721083
학과 : 미디어학과
학년 : 2학년

1.

```
<terminated> PayrollSystemTest [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Con
Employees processed individually:

John earned: $800.00

Karen earned: $670.00

Sue earned: $600.00

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The field SalariedEmployee.weeklySalary is not visible

        at test.Manager.getEarnings(Manager.java:14)
        at test.PayrollSystemTest.main(PayrollSystemTest.java:29)
```

- 부모클래스인 SalariedEmployee에서 weeklySalary가 private로 선언되어있기때문에 에러가 뜨게 된다.

2.

```
<terminated> PayrollSystemTest [Java Application] /Library/Java/JavaVirt
Employees processed individually:

John earned: $800.00

Karen earned: $670.00

Sue earned: $600.00

Bob earned: $2,500.00

Employees processed polymorphically:

John earned $800.00

Karen earned $670.00

Sue earned $600.00

Bob earned $2,500.00

Bob earned: $2,500.00

Bob earned: $2,500.00
```

- 메소드의 이름이 바뀌어서 상속이 불가능했다. 따라서 payrollsystemtest의 마지막 부분이 제대로 실행 될 수 없었다.

```
Employees processed individually:

John earned: $800.00

Karen earned: $670.00

Sue earned: $600.00

Bob earned: $2,500.00

Employees processed polymorphically:

John earned $800.00

Karen earned $670.00

Sue earned $600.00

Bob earned $2,500.00

Bob earned: $2,500.00

Bob earned: $2,500.00
```

- @override를 지워서 Manager 의 메소드인 getearnigs 메소드 자체에는 오류가 없었졌다. 이는 상속받는것이 아닌 Manager클래스에 별개의 메소드 하나를 의미하게되었기 때문이다. 때문에 결과는 그대로이다.
- @override 는 해당 메소드가 상위 클래스의 메소드를 오버라이딩 할 목적으로 정의됐음을 알리는 역할을 한다.

4.

```
<terminated> PayrollSystemTest [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents,
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The method setBonus(double) is undefined for the type Employee

        at test.PayrollSystemTest.main(PayrollSystemTest.java:67)
```

- Employee 클래스에 setBonus가 정의되어있지 않기에 문제가 발생하였다.

5.

Manager, Employee 클래스를 제외한 각 클래스에 setBonus 메소드를 추가해준다.

```
@Override
public void setBonus(double d) {

}

} // end class CommissionEmployee
```

-CommissionEmployee 클래스-

-

```
@Override
public void setBonus(double d) {

}
} // end class HourlyEmployee
```

-HourlyEmployee 클래스-

```
@Override
public void setBonus(double d) {

}

} // end class SalariedEmployee
```

-SalariedEmployee클래스-

그 후 Employee 클래스에

```
    public abstract void setBonus(double d);

} // end abstract class Employee
```

이와 같이 코드를 첨가해주면 메인함수에 주어진 코드를 그대로 넣어도 오류없이 보너스가 추가된 급여를 출력해낼 수 있다.

6. 기존 프로그램과 결과가 동일하게 나온다.

7. 변경하기전엔 프로그램 오류가 나지만, 변경 후엔 오류없이 실행이 된다. 그 이유는 추상클래스는 직접적인 객체 인스턴스를 생성 할 수 없기 때문이다.

8.

```
<terminated> PayrollSystemTest [Java Application] /Library/Java/JavaVirtu
Hello Java1
Hello Java4
Hello Java1
Hello Java3
Hello Java1
Hello Java2
Hello Java1
Hello Java4
Hello Java5
Employees processed individually:

John earned: $800.00

Karen earned: $670.00

Sue earned: $600.00

Bob earned: $2,500.00

Employees processed polymorphically:

John earned $800.00

Karen earned $670.00

Sue earned $600.00

Bob earned $2,600.00

Bob earned: $2,600.00

Bob earned: $3,500.00
```

Employee 클래스에는 Hello Java1을 넣어주었고
CommissionEmployee 클래스에는 Hello Java2
HourlyEmployee 클래스에는 Hello Java3
SalariedEmployee 클래스에는 Hello Java4
Manager 클래스에는 Hello Java5 를 추가해 주었다.
결과를 통해 알 수 있는 점은 생성자들이 일반적으로 상속의 가장 윗단계에 있는 순으로 실행된다는 점이다.

exercise

(가) Account class의 instance variable :

```
private final String name;
private final String ssn;
```

(나) Account class의 constructor :

```
public Account(String name, String ssn) {
    this.name = name;
    this.ssn = ssn;
}
```

(다) Account class의 concrete methods :

```
public String getName() {
    return name;
}
public String getssn() {
    return ssn;
}
```

(라) Account class의 abstract methods :

```
public abstract double getBalance();
public abstract void addInterest();
```

(마) Account class를 고려하여 CheckingAccount와 SavingAccount를 적절하게 수정하시오

-CheckingAccount :

```
public class CheckingAccount extends SavingAccount{
    private double balance;
    private double interest;
    private double interestRate;

    public CheckingAccount(String name, String ssn,double balance,
            double interest, double interestRate,int duration) {
        super(name,ssn,balance,interest,interestRate,duration);
    }
    @Override
    public double getBalance() {
        return balance;
    }
    public void deosit(double amt) {
        balance = balance+amt;
    }
    public void withdraw(double amt) {
        balance = balance - amt;
    }
    @Override
    public void addInterest() {
        balance+=balance+interestRate/12.0;
    }
}
```

-SavingAccount :

```
public class SavingAccount extends Account{
    private double balance;
    private double interest;
    private double interestRate;
    public int duration;
    public SavingAccount(String name, String ssn,
            double balance, double interest, double interestRate,int duration) {
        super(name,ssn);
        this.duration = duration;
    }
    @Override
    public double getBalance() {
        return balance;
    }
    @Override
    public void addInterest() {
        balance+=balance*interestRate*duration/12;
    }
}
```