자료구조 실습 과제 #1 (20171622 박건후)

1. Ploymorphism(다형성)은 어떻게 사용되었는가?

```
public abstract class Employee
{
    private Date joinDate;

    // 생략

    public void setJoinDate(int month, int day, int year){
        joinDate= new Date( month, day, year );
    }

    public Date getJoinDate(){
        return joinDate;
    }

    @Override
    public String toString(){
      return String.format( "%s %s\n%s: %s\n%s: %s\n%s: %s",
         getFirstName(), getLastName(),
         "social security number", getSocialSecurityNumber(),
         "birth date", getBirthDate(), "join date", getJoinDate() );
    } // end method toString
} // end abstract class Employee
```

다형성이란 객체지향개념에 있어서는 여러가지 형태를 가질 수 있는 능력을 뜻한다. 하나의 객체를 여러 가지 타입으로 선언할 수 있음을 뜻하기도 한다. Java에서 다형성은 상속과 인터페이스를 통해 이루어진다. 이번 프로젝트에서 사용된 Employee 추상클래스의 상속을 일례로 이 프로젝트에서 다형성이 사용되었다 말할 수 있다.

```
public class HourlyEmployee extends Employee
{
   // 생략
   @Override
   public double earnings()
   {
      if ( getHours() <= 40 ) // no overtime
         return getWage() * getHours();
      else
         return 40 * getWage() + ( getHours() - 40 ) * getWage() * 1.5;
   } // end method earnings

   // return String representation of HourlyEmployee object
   @Override
   public String toString()
   {
      return String.format( "hourly employee: %s\n%s: $%,.2f; %s: %,.2f",
         super.toString(), "hourly wage", getWage(),
         "hours worked", getHours() );
   } // end method toString
```

```
} // end class HourlyEmployee
```

위의 HourlyEmployee 클래스는 Employee라는 추상클래스를 상속받았다. 동시에 toString이라는 메소드를 오버라이드하고 있다. toString 이라는 메소드는 java의 Object 클래스에서 생성된 것이다.

```
//java.lang.Object.toString
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

Java 객체를 생성할 때 기본적으로 Object를 상속받는다. 따라서 클래스에 String toString() 이라는 것을 정의하면, 오버라이드함을 의미한다. 위의 코드 중 HourlyEmployee의 double earnings 이란 메소드는 Employee의 추상메소드인 double earnings 를 오버라이드 한 것이다. 추상메소드로 선언하는 이유는 자손클래스에게 오버라이딩을 반드시 할 수 있게끔 강제하기 때문이다.

---

2. 다형성을 사용하지 않으면 어떻게 해결 할 수 있는가?

다형성을 사용하지 않으면 프로젝트의 규모가 커진다. 프로젝트의 규모가 커진다는 것은, 같은 논리의 코드를 각 클래스마다 반복한다는 것을 의미한다. 클래스마다 property상 공통되는 부분이 있기에 상속이라는 것으로 객체지향의 다형성을 이용하는 것이다. 만약 다형성을 사용하지 않는다면, HourlyEmployee의 코드는 아래와 같을 것이다.

```
public class HourlyEmployee {
   private String firstName;
   private String lastName;
   private String socialSecurityNumber;
   private Date birthDate;
   private Date joinDate;


   private double wage; // wage per hour
   private double hours; // hours worked for week

   // eight-argument constructor
   public HourlyEmployee( String first, String last, String ssn,
      int month, int day, int year,
      double hourlyWage, double hoursWorked )
   {
      super( first, last, ssn, month, day, year );
      setWage( hourlyWage );
      setHours( hoursWorked );
   } // end eight-argument HourlyEmployee constructor

   // set wage
   public void setWage( double hourlyWage )
   {
      wage = hourlyWage < 0.0 ? 0.0 : hourlyWage;
   } // end method setWage

   // return wage
   public double getWage()
   {
```

```
      return wage;
   } // end method getWage

   // set hours worked
   public void setHours( double hoursWorked )
   {
      hours = ( ( hoursWorked >= 0.0 ) && ( hoursWorked <= 168.0 ) ) ?
         hoursWorked : 0.0;
   } // end method setHours

   // return hours worked
   public double getHours()
   {
      return hours;
   } // end method getHours

   // calculate earnings; override abstract method earnings in Employee
   @Override
   public double earnings()
   {
      if ( getHours() <= 40 ) // no overtime
         return getWage() * getHours();
      else
         return 40 * getWage() + ( getHours() - 40 ) * getWage() * 1.5;
   } // end method earnings

   // return String representation of HourlyEmployee object

   public String toString()
   {
      return String.format( "hourly employee: %s\n%s: $%,.2f; %s: %,.2f",
         super.toString(), "hourly wage", getWage(),
         "hours worked", getHours() );
   } // end method toString

    public void setJoinDate(int month, int day, int year){
        joinDate= new Date( month, day, year );
    }

    public Date getJoinDate(){
        return joinDate;
    }

   // set first name
   public void setFirstName( String first )
   {
      firstName = first;
   } // end method setFirstName

   // return first name
   public String getFirstName()
   {
      return firstName;
   } // end method getFirstName

   // set last name
   public void setLastName( String last )
   {
```

```
      lastName = last;
   } // end method setLastName

   // return last name
   public String getLastName()
   {
      return lastName;
   } // end method getLastName

   // set social security number
   public void setSocialSecurityNumber( String ssn )
   {
      socialSecurityNumber = ssn;  // should validate
   } // end method setSocialSecurityNumber

   // return social security number
   public String getSocialSecurityNumber()
   {
      return socialSecurityNumber;
   } // end method getSocialSecurityNumber

   // set birth date
   public void setBirthDate( int month, int day, int year )
   {
      birthDate = new Date( month, day, year );
   } // end method setBirthDate

   // return birth date
   public Date getBirthDate()
   {
      return birthDate;
   } // end method getBirthDate

} // end abstract class Employee
```

위와 같이 다형성을 적용하지 않는다면, HourlyEmployee 클래스와 비슷한 종류의 클래스인 SalariedEmployee 클래스에서도 setBirthDate, getSocialSecurityNumber 와 같은 메소드를 또 다시 기술해야한다. 똑같은 코드를 다른 클래스에서 또 적어내기에는 복잡하며, 그만큼 귀찮은 일도 없다.

또한 객체 간의 비교에 있어서 중요한 변수인 employees 배열을 사용하지 못한다. Employee 객체를 상속받지 못해 HourlyEmployee, SalariedEmployee 가 서로 다른 클래스기 때문에 한 배열에 존재할 수 없다. 따라서 아래와 같은 코드는 사용하지 못한다.

```
for ( Employee currentEmployee : employees ) {
        if ( currentEmployee instanceof BasePlusCommissionEmployee ) {
            // Casting 이 불가능하다.
            BasePlusCommissionEmployee employee =
              ( BasePlusCommissionEmployee ) currentEmployee;
        }

}
```

그러므로 클래스로 객체를 만든 다음, 아래와 같이 하나하나 호출해줘야한다.

```
SalariedEmployee salariedEmployee = new SalariedEmployee( "John", "Smith", "111-11-1111", 6, 15, 1944, 800.00 );
HourlyEmployee hourlyEmployee = new HourlyEmployee( "Karen", "Price", "222-22-2222", 12, 29, 1960, 16.75, 40 );

//SalariedEmployee 의 salary 지불 코드
paySalaryToSalariedEmployee();
//HourlyEmployee 의 salary 지불 코드
paySalaryToHourlyEmployee();
```

분명 paySalaryToSalariedEmployee, paySalaryToHourlyEmployee는 같은 논리인 코드다. 다형성을 이용하면, 불필요하게 사용되는 코드를 줄일 수 있으며, 프로젝트의 규모가 커지지 않는다.

정리하자면, 위와 같은 프로젝트는 다형성을 사용하지 않고서도 구성할 수 있을 것이다. 하지만 다음과 같은 문제가 따를 것이다.

- 첫 번째로, 다른 개발자와 협업함에 있어서 비효율적이다. 추상클래스, 인터페이스의 목적은 설계 측면에서 약속을 하는 것이다. 각 객체마다 공통된 특성이 있다. 생일 같은 경우 SalariedEmployee도 있고, HourlyEmployee도 있다. 그런데 SalariedEmployee에서는 birthDate라 쓰고, HourlyEmployee에서는 bd라고 기술하면 협업하는데 있어 큰 충돌(conflict)가 있을 것이다.
- 두 번째로, 동일한 로직의 코드를 여러 클래스에 기술해야함으로써 비효율적이다. 또한 동일한 로직의 함수를 여러번 작성해야하므로 비효율적이다. paySalaryToHourlyEmployee(), paySalaryToHourlyEmployee() 는 동일한 로직의 코드지만 클래스마다 출력처리를 해야하기에 긴 코드가 작성된다.

---

3. 추가한 함수, 클래스 변경 사항

Class Employee

- Employee 클래스에 Date joinDate를 추가
- setJoinDate 메소드와 getJoinDate 메소드 구현

```
    public void setJoinDate(int month, int day, int year){
        joinDate= new Date( month, day, year );
    }

    public Date getJoinDate(){
        return joinDate;
    }
```

String date2String

- Date 객체의 month, day를 SimpleDateFormat 형식으로 바꾸기 위한 메소드
- ex) 2020/1/23 -> 20200123 으로 바꿔주기 위함

```
public static String date2String(int value) {
     if (value < 10) {
      return "0" + String.valueOf(value);
     } else {
      return String.valueOf(value);
     }
}
```

boolean isTenYears

- 근속한 지 10년이 넘으면 총 급여+총 급여의 10% 를 위한 메소드
- diffday는 급여날과 입사날의 차이. day 형태 (ex. 3일)
- diffday가 3650일(10년)을 넘으면 근속년수 10년이 넘었다고 판단하면서 true를 반환

```
public static boolean isTenYears(Employee employee, int currentMonth) {
     Calendar cal = Calendar.getInstance();

     int currentYear = cal.get(cal.YEAR);

     int employeeJoinYear = employee.getJoinDate().getYear();
     int employeeJoinMonth = employee.getJoinDate().getMonth();
     int employeeJoinDay = employee.getJoinDate().getDay();
     String employeeJoinDate = String.valueOf(employeeJoinYear) +
date2String(employeeJoinMonth) + date2String(employeeJoinDay);

     String currentDate = String.valueOf(currentYear) +
date2String(currentMonth) + "01";
     String strFormat = "yyyyMMdd";
     SimpleDateFormat sdf = new SimpleDateFormat(strFormat);
     try {
      Date startDate = sdf.parse(employeeJoinDate);
      Date endDate = sdf.parse(currentDate);

      long diffDay = (endDate.getTime() - startDate.getTime()) / (24 * 60 * 60 *
1000);
      System.out.println("근속년수 : " + diffDay + "일");

      if (diffDay > 3650) {
       return true;
      }


     } catch (ParseException e) {
      e.printStackTrace();
     }

     return false;
    }
```

4. 실행결과

- 근속일수 10년 이상, 생일월이 급여달인 경우

```
salaried employee: 박 건후
social security number: 555-55-5555
birth date: 2/9/1998
join date: 3/2/1990
weekly salary: 800.00
birthday bonus : +100.00
근속년수 : 10928일
(multiply 1.1 to salary / 10years) earned $ 990.00
```

- 근속일수 10년 이상 조건만 달성한 경우

```
base-salaried commission employee: Sue Jones
social security number: 888-88-8888
birth date: 3/2/1998
join date: 3/2/2000
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
근속년수 : 7275일
```

- 생일 조건만 달성한 경우

```
base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
birth date: 3/2/1965
join date: 3/2/2019
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
birthday bonus : +$100.00
근속년수 : 365일
earned $ 630.00
```

- 어떠한 조건도 달성하지 못했을 경우

```
salaried employee: John Smith
social security number: 111-11-1111
birth date: 6/15/1944
join date: 3/2/2019
weekly salary: $800.00
근속년수 : 336일
earned $ 800.00
```