Java

Class

- class의 관계와 Inheritance의 개념 및 중요성을 배운다.
- Inheritance에 이어 polymorphism 을 배운다.
- final Keyword의 활용도를 알아본다.
- 추상화?
- interface가 무엇인지 알아보고 interface 정의법과 사용법을 배운다.
- Enumeration에 대해 알아보고 필요성과 생성 법, 활용법을 배운다.
- Inner class의 개념과 장점 등을 배운다.

Class Inheritance

class의 관계

※ 클래스 관계의 종류

구분	클래스 관계
[has a]	특정 객체 내에서 다른 객체를 가지고 있는 것을 의미한다.
[is a]	특정 객체가 다른 객체에게 자신의 능력을 포함시켜주는 상속 관계를 의미한다.

Inheritance의 개념과 중요성

class Inheritance

Inheritance? 부모가 보유하고 있는 재산 중 일부를 자식이 물려받는 것을 의미. Java에서는 이런 class들간의 multiple Inheritance를 지원하지 않으므로 Object의 명확성을 높였다.

※ 상속관계의 용어

용어	설명
Base Class(기본 클래스) Super Class(슈퍼 클래스) Parent Class(부모 클래스)	왼쪽의 용어가 모두 같은 뜻을 의미하며 이는 상속을 주기 위해 준비된 특정 클래스를 의미한다.
Derivation Class(유도 클래스) Sub Class(하위 클래스) Child Class(자식 클래스)	왼쪽의 용어가 모두 같은 뜻을 의미하여 이것은 특정 클래스로부터 상속을 받아 새롭게 정의되는 클래스를 의미한다.

Class Inheritance

Class Inheritance Definition

- Java에서 Inheritance라는 것은 특정 class가 가지는 일부 속성과 기능을 다른 새로운 class에게 제공하기 위해 맺는 class들간의 관계를 말한다. 이는 super class를 새로운 sub class에서 [extends]라는 Keyword를 사용하여 서로 관계를 맺은 상태이다.

```
class [sub class name] extends [super class name] {
   …;
}
```

Class Inheritance

❖ Class Inheritance의 중요성

Class Inheritance는 Object의 재사용이라는 장점뿐만 아니라 code의 간결성을 제공해주는 Object oriented language의 장점과 중요한 특징이 된다. 잘 정의된 super class가 있다면 sub class의 작성이 간편해지고 개발 시간이 단축된다.

```
01  class CellPhone{
02
03          String model; // 모델명
04          String number; // 전화번호
05          int chord; // 화음
06
07          public void
setNumber(String n){
08                      number = n;
09          }
10          public String getModel(){
11                      return model;
12          }
13          public int getChord(){
14                      return chord;
15          }
```

```
16          public String getNumber(){
17                      return number;
18          }
19  }
```

Class Inheritance

```
01  class D_caPhone extends CellPhone{
02
03        String pixel; //화소 수
04        public D_caPhone (String model,String num, int chord,
String pixel){
05                  this.model = model;
06                  number = num;
07                  this.chord = chord;
08                  this.pixel = pixel;
09        }
10  }
```

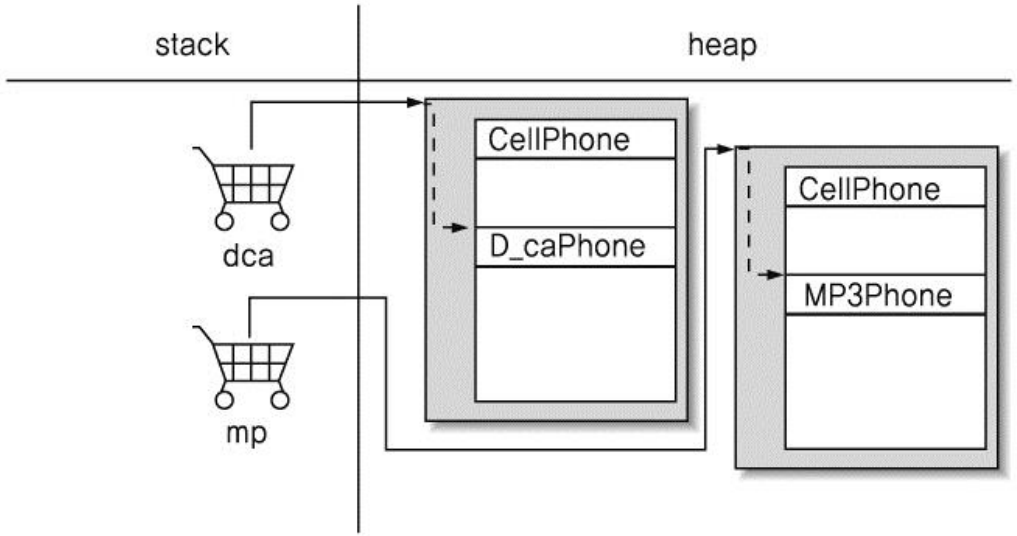
```
01  class MP3Phone extends CellPhone{
02
03        int size; // 저장 용량
04        public MP3Phone (String model,String num, int chord, int
size){
05                  this.model = model;
06                  number = num;
07                  this.chord = chord;
08                  this.size = size;
09        }
10  }
```

Class Inheritance

```
01  class CellPhoneTest{
02
03          public static void main(String[] args){
04                  D_caPhone dca = new D_caPhone(
05                          "IN-7600","011-9XXX-9XXXX",60,"400만");
06                  MP3Phone mp = new MP3Phone(
07                          "KN-600","011-9XXX-9XXXX",60,256);
08
09                  System.out.println(dca.getModel()+","+
10                          dca.getChord()+","+dca.getNumber());
11          }
12  }
```

Class Inheritance

- Inheritance관계에 있어 정보의 공유와 상속을 받는 sub class의 작업이 매우 유익함을 알 수 있다. 그림을 보고 Inheritance 관계의 Object생성시 구조를 이해하자!



Class Inheritance

Overriding

- Overriding은 [method redefinition]라고도 불리며 이는 서로 Inheritance관계로 이루어진 Object들간의 관계에서 비롯된다.

  super class가 가지는 method를 sub class에서 똑 같은 것을 새롭게 만들게 되면 더 이상 super class의 이름이 같은 method를 호출할 수 없게 된다. 이를 Overriding이라 하고 또는 member 은폐라고도 한다.

※ 오버라이딩과 오버로딩의 차이

오버라이딩(재정의)	구분	오버로딩(다중정의)
상속관계	적용	특정 클래스
super 클래스의 메서드보다 sub 클래스의 메서드 접근제한이 동일하거나 더 넓어야 한다. 예를 들어, protected라면 protected/public이다.	접근제한	상관없다.
기본적으로 같아야 한다.	리턴형	상관없다.
super 클래스의 메서드명과 sub 클래스의 메서드명이 같아야 한다.	메서드명	반드시 같아야 한다.
반드시 같아야 한다.	인자(타입, 개수)	반드시 달라야 한다.

Class Inheritance

- Overriding Ex

```
01 class Parent{
02
03        String msg = "Parent클래스";
04        public String getMsg(){
05                  return msg;
06        }
07 }
08
09 class Child extends Parent{
10
11        String msg = "Child클래스";
12        public String getMsg(){ //메서드 Overriding
13                  return msg;
14        }
15 }
16
```

```
17 public class OverridingEx {
18        public static void main(String[]
args){
19
20              Child cd = new Child();
21              System.out.println("cd :
"+cd.getMsg());
22
23              Parent pt = new Child();
24              System.out.println("pt :
"+pt.getMsg());
25        }
26 }
```

Class Inheritance

❖ super와 super()

- Super는 this와 함께 Object를 참조할 수 있는 reference변수이다. this는 특정 Object 내에서 자기 자신의 Object를 참조할 수 있는 유일한 reference변수이다. super라는 reference변수는 현재 Object의 바로 상위인 super class를 참조할 수 있는 것이다.

- super()는 바로 super class의 Constructor를 의미한다. Arguement가 있다면 argument의 type와 일치하는 Constructor를 의미한다.

Class Inheritance

- super() Ex

```
01 class Parent{
02
03     public Parent(int var){
04         System.out.println("Parent 클래스");
05     }
06 }
07 class SuperEx extends Parent {
08     public SuperEx() {
09         super(1);
10         System.out.println("SuperEx 클래스");
11     }
12
13     public static void main(String[] args) {
14         SuperEx se = new SuperEx();
15     }
16 }
```

final Keyword

- Final은 Keyword이며 이것은 더 이상의 확장이 불가능함을 알리는 종단(Constant)과 같은 것을 의미한다.
  - 변수에 final을 적용 시 Constant를 의미한다.

    ※ 변수에 final을 적용할 때의 사용 예

구성	사용 예
final [자료형] [변수명] ;	final int VAR = 100;

  - Method에 final을 적용 시 Overriding으로의 확장이 불가능하다.

    ※ 메서드에 final을 적용할 때의 사용 예

구성	사용 예
[접근제한] final [반환형] [메서드명]() { }	public final void method() { }

  - class에 final을 적용 시 더 이상의 상속 확장이 불가능하다.

    ※ 클래스에 final을 적용할 때의 사용 예

구성	사용 예
[접근제한] final class [클래스명] { }	public final class MeEX{ }

abstract Keyword

- abstract화의 이해와 선언법
  - abstract화라는 것은 구체적인 개념으로부터 공통된 부분들만 추려내어 일반화 할 수 있도록 하는 것을 의미. 일반적으로 사용할 수 있는 단계가 아닌 아직 미완성적 개념인 것이다. Java에서 얘기하는 abstract화 작업을 하기 위해서는 먼저 abstract method를 이해해야 한다.

※ 추상 메서드의 구성과 예문

구성	사용 예
[접근제한] abstract void [메서드명] ( );	public abstract void absTest( );

위의 표는 abstract method의 구성이다. method를 정의하면서 brace({}) 를 생략하여 실상 method가 하는 일(body)이 없이 semicolon(;)으로 문장의 끝을 나타내었다. 그리고 abstract라는 keyword를 통해 현 method가 abstract method임을 명시하였다.

abstract Keyword

- abstract method를 하나라도 가지게 되는 class가 바로 abstract class가 된다. 그리고 이런 abstract class 또한 다음과 같이 abstract class임을 명시해야 한다.

※ 추상 클래스의 구성과 예문

구성	사용 예
[접근제한] abstract class [클래스명] { }	public abstract class AbsEx{ }

```
01  abstract class AbsEx1{
02          int a = 100; //변수
03          final String str = "abstract test"; //Constant
04          public String getStr(){ //일반 method
05                  return str;
06          }
07
08          // abstract method는 몸체(body)가 없다.
09          abstract public int getA();
10  }
```

abstract Keyword

- Abstarct class의 상속 관계
  - Abstract class들간에도 Inheritance가 가능하다. 일반 class들간의 Inheritance와 유사하지만 abstract class들간의 Inheritance에서는 Inheritance받은 abstract method들을 꼭 re-definition할 필요는 없다. Inheritance만 받아두고 있다가 언제가 일반 class와 Inheritance관계가 이루어 질 때가 있다. 이때 re-deifinition 하지 못했던 Inheritance 받은 abstract method들을 모두 일반 class내에서 re-definition해도 되기 때문이다. AbsEx1을 Inheritance받는 abstract class를 작성해 보자!

```
01  abstract class AbsEx2 extends AbsEx1{
02          public  int  getA(){  //Super  class 의 abstract  methodre-
definition
03                  return a;
04          }
05          public abstract String getStr();
06  }
```

abstract Keyword

```
01 class AbsEx extends AbsEx2{
02
03      public String getStr(){ //AbsEx2의 추상 메서드 재 정의
04              return str; //str은 AbsEx1의 멤버이다
05      }
06      public static void main(String[] args){
07              AbsEx ae = new AbsEx();
08              System.out.println("ae.getA():"+ae.getA());
09              System.out.println("ae.getStr():"+ae.getStr());
10      }
11 }
```

interface

- interface는 음식점의 메뉴판과 같다.
  메뉴판을 보고 고객이 원하는 음식을 요청하게 되는데 메뉴판 자체가 음식을 주지는 않는다. 음식은 주방이라는 곳에서 나오므로 메뉴판은 고객이 호출할 수 있는 Service의 목록이라 할 수 있다.

```
[Access Modifier] interface [interface name] {
   Constant;
   abstract method;
}
```

- 위의 interface의 구조를 보고 알 수 있듯이 interface 내에는 Constant 또는 abstract method들만 정의가 가능하다. 그리고 사용하기 위해서는 일반 class에서 implements해야 한다. 일반 class에서 "implements" 라는 Keyword로 특정 interface를 구현하겠다고 명시하는 것이다. 그렇게 되면 명시한 interface가 가지는 abstract method들은 구현 받은 class에서 하나도 빠짐없이 Overriding해야 한다. 다음의 예제는 interface의 기본 구성과 구현을 다룬 예제이다.

interface

```
01 interface InterTest {
02       static final int A = 100;
03       abstract int getA(); //abstract keyword는 생략 가능!
04 }
05
06 class  InterTestEx implements InterTest
07 {
08       public int getA(){
09                return A;
10       }
11
12       public static void main(String[] args)
13       {
14                InterTestEx it1 = new InterTestEx();
15                System.out.println("getA():"+it1.getA());
16       }
17 }
```

interface

- Interface간의 Inheritance
  - interface 내에는 Constant 또는 동작부분을 구현하지 않은 abstract method들이 정의된다. interface를 구현(implements)하겠다고 명시한 일반 class에서 원하는 형태로 실제 구현력을 가지게 된다. 그러므로 실제 구현력이 없는 interface들 간의 Inheritance에서는 multiple Inheritance가 제공된다

    interface는 메뉴판과 같이 음식점에서 어떤 음식을 만들 수 있는지를 알려주는 중계자 역할만 할 뿐이다. 음식을 만들어 가져오는 것은 그 메뉴판을 포함(구현)하고 있는 음식점이 반드시 제공해야 할 의무가 있는 것이다.

```
[Access Modifier] interface [interface명] extends 부모interface명1,부모interface명2,…,부모interface명n {
  Constant;
  Abstract Method;
}
```

  - interface가 다른 interface로부터 Inheritance를 받았다고 하지만 Overriding을 할 수는 없다. interface는 body를 가지는 일반 method를 포함할 수 없다. 그러므로 상속을 받은 자식 interface를 구현(implements)하는 일반 class에서 부모 interface와 자식 interface의 abstract method들을 모두 Overriding해야 한다.

interface

```
01  interface Inter1{
02          public int getA();
03  }
04 ////////////////////////////////////////
05  interface Inter2{
06          public int getA();
07  }
08 ////////////////////////////////////////
09  interface Inter3 extends Inter1, Inter2{
10          public int getData();
11  }
12 ////////////////////////////////////////
13  class InterEx2 implements Inter3{
14          int a = 100;
15          public int getA(){
16                  return a;
17          }
18          public int getData(){
19                  return a+10;
20          }
```

```
21          public static void main(String[] args){
22                  InterEx2 it = new InterEx2();
23                  Inter1 it1 = it;
24                  Inter2 it2 = it;
25                  Inter3 it3 = it;
26                  System.out.println(it1.getA());
27                  System.out.println(it2.getA());
28                  System.out.println(it3.getData());
29          }
30  }
```

Enumerated types

❖ enum의 정의와 구성

- C language에서 정수를 자동적으로 증가하여 상수들로 만들어 쓰는 enum과 비슷하며 Java에서 얘기하는 enumerated types은 Constant를 하나의 Object로 인식하고 여러 개의 Constant Object들을 한 곳에 모아둔 하나의 묶음(Object)이라 할 수 있다.

- enum의 구성

```
[Access Modifier] enum [enum name]{
   Constant1, Constant2, …, Constant n
}
```

Enumerated types

```
01  public class EnumEx1 {
02    public enum Lesson {
03                    JAVA, XML, EJB
04    }
05
06    public static void main(String[] args) {
07        Lesson le = Lesson.JAVA;
08
09        System.out.println("Lesson : " + le);
10        System.out.println("XML : " + Lesson.XML);
11    }
12  }
```

Enumerated types

❖ enumerated types의 실체

- Enumerated types은 내부에서 순차적으로 정의되는 값들을 JAVA, XML, EJB라는 이름으로 그냥 일반적인 Constant라는 개념만으로 저장되는 것이 아니다. java.lang이라는 package에 Enum이라는 abstract class를 Inheritance 받는 Inner class가 정의되는 것이다. 그럼 앞의 예제에서 Lesson이라는 enumerated types를 가지고 예를 들어보면 다음과 같다.

```
public static final class EnumEx1$Lesson extends Enum
```

Enumerated types

❖ Enumerated types의 활용

- enumerated types(열거형)가 Constant를 가지고 java.lang package의 Enum Object를 생성하여 모아둔 것이라는 것을 알았다. 다음 예제를 보자.

```
01 class EnumEx2 {
02    public enum Item{
03        Add, Del, Search, Cancel
04    }
05
06    public static void main(String[] args) {
07        Item a1 = Item.Search;
08        if (a1 instanceof Object){ //열거형이 객체인지 아닌지 비교
09                System.out.println(a1.toString()+"^^");
10                System.out.println("OK! instanceof Object");
11                System.out.println("저장된 실제 정수값 : "+a1.ordinal());
12        }
13
```

Enumerated types

```
14          Item[] items = Item.values();
15                  System.out.println("items.length : "+items.length);
16
17                  for(Item n : Item.values())
18                          System.out.println(n+":"+n.ordinal());
19
20          }
21  }
```

- enumerated types내에 정의한 Add, Del, Search, Cancel들이 enumerated types 자신의 형태로 객체가 생성되어 Array로 관리됨을 알 수 있다. 그리고 그 Object 하나 하나가 기억하고 있는 실제 Constant 값은 ordinal() method를 통해 확인이 가능하다.

Inner class

- Inner class의 이해와 특징
  - Inner class? class내에 또 다른class가 정의되는것을 의미.
    Inner class가 필요한 이유는 지금까지 작업해 왔던 class들과는 다르게 독립적이지는 않지만 하나의 member처럼 사용할 수 있는 특징이 있다.
  - Inner class를 정의 시 주의사항과 장점
    - Inner class는 외부 class의 모든 member들을 마치 자신의 member처럼 사용할 수 있다.
    - Static Inner class는 제외하고는 다른 Inner class는 항상 외부 class를 통해야 생성이 가능하다.
- Inner class의 종류와 사용 방법

※ 내부 클래스의 종류

종류	설명
Member	멤버 변수나 멤버 메서드들과 같이 클래스가 정의된 경우에 사용한다.
Local	특정한 메서드 내에 클래스가 정의된 경우에 사용한다.
Static	static 변수(클래스 변수)와 같이 클래스가 static으로 선언된 경우에 사용한다.
Anonymous	참조할 수 있는 이름이 없는 경우에 사용한다.

Inner class

- Member Inner class
  Object를 생성해야만 사용할 수 있는 Member들과 같은 위치에 정의되는 class를 말한다. 즉 Inner class를 생성하려면 외부 class의 Object를 생성한 후에 생성할 수 있다.

- Member Inner class의 구성

```
class Outer {
   ...
  class Inner {

  }
  ...
}
```

Inner class

- **Member Inner class Ex**

```
01  class MemberInner{
02
03          int a = 10;
04          private int b = 100;
05          static int c = 200;
06
07          class Inner {  //Inner class definition.
08                  public void printData(){
09                          System.out.println("int a : "+a);
10                          System.out.println("private int b : "+b);
11                          System.out.println("static int c : "+c);
12                  }
13          }
14          public static void main(String[] args){
15
16          //          MemberInner outer = new MemberInner();
17          //          MemberInner.Inner inner = outer.new Inner();
18                      MemberInner.Inner inner = new MemberInner().new Inner();
19                      inner.printData();
20          }
21  }
```

Inner class

- Local Inner class

  Local Inner class는 method 안에서 정의되는 class를 말한다. method 안에서 선언되는 local variable와 같은 것이다. method가 호출될 때 생성할 수 있으며 method의 수행력이 끝나면 local variable와 같이 자동 소멸된다.

- Local Inner class의 구성

```
class Outer {
   ...
  public void methodA() { // member mthod
    class Inner {

    }
  }
  ...
}
```

Inner class

- Local Inner class Ex

```
01  class LocalInner {
02
03          int a = 100; //member variable
04          public void innerTest(int k){
05                      int b = 200; // local variable
06                      final int c = k; //Constant
07                      class Inner{
08                        // Local Inner class는 외부class의 member variable와
09                        // Constant들만 접근이 가능하다.
10                        public void getData(){
11                                  System.out.println("int a : "+a);
12                       //         System.out.println("int b : "+b);
13                         // local Inner class는 local variable를 사용할 수 없다.
14                                  System.out.println("final int c : "+c);//Constant사용
15                        }
16                      }
17                      Inner i=new Inner();//method 내에서 Local  Inner class생성
18                      i.getData(); //생성된 reference를 통해  method 호출
19          }
```

Inner class

```
20          public static void main(String[] args) {
21                      LocalInner outer = new LocalInner();
22                      outer.innerTest(1000);
23          }
24  }
```

Inner class

- static Inner class

  static Inner class로 어쩔 수 없이 정의하는 경우가 있는데 그것은 바로 Inner class 안에 static variable를 가지고 있다면 어쩔 수 없이 해당 Inner class는 static으로 선언하여야 한다.

- Static Inner class의 구성

```
class Outer {
   ...
  static class Inner {

}
  ...
}
```

Inner class

- Static Inner class Ex

```
01 class StaticInner {
02
03         int a = 10;
04         private int b = 100;
05         static int c = 200;
06
07         static class Inner{
08           // 어쩔 수 없이 Inner class를 static으로 선언해야 할 경우가 있다.
09           // 그건 바로 Inner class의 member들 중 하나라도
10           // static member가 있을 때이다.
11                     static int d = 1000;
12                     public void printData(){
13 //                           System.out.println("int a : "+a);  //error
14 //                           System.out.println("private int b : "+b);  //error
15                             System.out.println("static int c : "+c);
16                     }
17         }
```

Inner class

```
18        public static void main(String[] args) {
19                //또 다른 독립된 Object에서 static Inner class 생성시
20                StaticInner.Inner inner = new StaticInner.Inner();
21                inner.printData();
22
23                // StaticInner라는 외부 class내에서 생성시
24                // Inner inner = new Inner();
25        }
26  }
```

- Static variable나 method들은 Object를 생성하지 않고도 접근이 가능하다고 했다. Static Inner class는 외부 class를 생성하지 않고도 [외부_class명.내부_class_Constructor()]로 생성이 가능함.

Inner class

- Anonymous Inner class

  Anonymous? 이름이 없는 것을 의미, Java의 program적으로 해석하면 정의된 class의 이름이 없다는 것이 된다.

- Anonymous Inner class의 구성

```
class Outer {
  ...
  Inner inner = new Inner(){
    ...;
  };
public void methodA() { // member method
new Inner() {
    ...;
};
}
...
}
```

Inner class

- Anonymous Inner class Ex

```
01 abstract class TestAbst{
02   int data = 10000;
03   public abstract void printData(); // abstract method
04 }
05 class  AnonyInner1{
06
07   TestAbst inn = new TestAbst (){
08       public void printData(){ // 미완성된 것을 완성한다.
09
10          System.out.println("data : "+data);
11       }
12   };
13
14   public static void main(String[] args){
15       AnonyInner1 ai = new AnonyInner1();
16       ai.inn.printData();
17   }
18 }
```

오늘 꿈을 오늘 이룹시다 화이팅!! ~.^