

# 내부 Class

# 1. 내부 Class

- ❖ 별도로 클래스를 정의하기에는 크기가 작은 경우나 하나의 클래스 안에서 구분되어야 하는 정보를 포함해야 하는 경우 클래스 내부에 다른 클래스를 선언해서 사용하는 것이 가능한데 GUI의 이벤트 처리를 할 때 많이 사용
- ❖ 종류
  - ✓ inner class: instance variable
  - ✓ nested (static inner) class: class variable
  - ✓ local class : local variable
  - ✓ anonymous inner class

## 2. 일반 inner class

- ❖ 클래스의 내부에 클래스를 선언해서 내부 클래스의 객체를 생성해서 사용
- ❖ 형식

```
class Outer{  
    변수  
    메소드  
    class Inner{  
        변수  
        메소드  
    }  
}
```

- ❖ 클래스 생성 구조

- Outer.class  
=> Outer\$Inner.class

- ❖ Outer의 메소드에서 Inner의 변수나 메소드를 호출 시에는 Instance화를 해서 사용
- ❖ Inner의 메소드에서 Outer의 변수나 메소드를 직접 호출하는 것은 가능
- ❖ 외부에서 내부 클래스의 Instance화

바깥클래스 인스턴스명 = new 바깥클래스();

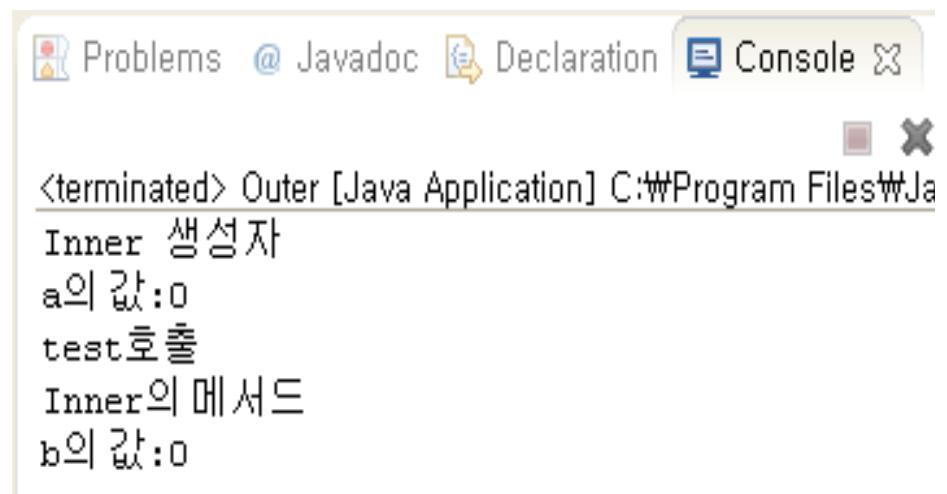
바깥클래스.안쪽클래스 인스턴스명=바깥클래스의 인스턴스명.new 안쪽 클래스의 생성자();

Outer ou=new Outer();

Outer.Inner in=ou.new Inner();

# 실습(Outer.java)

```
public class Outer {  
    private int a;  
  
    public Outer() {  
        System.out.println("Outer의 생성자");  
    }  
  
    public void test() {  
        System.out.println("test호출");  
    }  
}
```



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following text:

```
<terminated> Outer [Java Application] C:\Program Files\Ja  
Inner 생성자  
a의 값:0  
test호출  
Inner의 메서드  
b의 값:0
```

# 실습(Outer.java)

```
public class Inner {  
    private int b;  
    public Inner() {  
        System.out.println("Inner 생성자");  
        // 사용 가능 : 이유는 Inner클래스 안에서 Outer의 멤버는 바로 사용 가능  
        System.out.println("a:" + a);  
        test();  
    }  
    public void method() {  
        System.out.println("Inner의 메소드");  
        System.out.println("b:" + b);  
    }  
}  
public static void main(String args[]) {  
    Outer ou = new Outer(); // 결과 : Outer의 생성자  
    // ou.method(); //Inner의 메소드를 바로 접근 불가.  
    Outer.Inner in = ou.new Inner(); // 결과 : Inner 생성자  
    in.method(); // 결과 : Inner의 메소드  
    // 결과 : test호출0  
}
```

# 3. nested(static inner) class

- ❖ 내부 클래스를 클래스 변수처럼 사용
- ❖ 내부 클래스 안에 static 멤버를 포함시켜야 하거나 외부 클래스의 static 메소드 내에서 클래스를 사용하는 경우에 사용
- ❖ 일반 내부 클래스에는 static 멤버를 포함시킬 수 없음

```
class Outer{  
    변수  
    메소드  
    static class Inner {  
        변수  
        메소드  
    }  
}
```

- Outer.class  
=> Outer\$Inner.class

- ❖ 클래스의 특성이 static
- ❖ 외부 클래스의 객체없이 내부 클래스 객체 생성 가능
- ❖ 내부 클래스에서는 외부 클래스에 선언된 멤버 중에서 static 멤버만 사용이 가능

# 실습(StaticInnner.java)

```
public class StaticInnner {  
    int a;  
    public static class Inner  
    {  
        static int g_n;  
        static {  
            g_n=1;  
        }  
        public void disp()  
        {  
            System.out.println("g_n 의 값:" + g_n);  
        }  
    }  
    public static void main(String args[])  
    {  
        //외부 클래스의 객체 없이도 내부 클래스의 객체 생성 가능  
        Inner obj = new Inner();  
        obj.disp();  
    }  
}
```



# 4. Local inner class

- ❖ 클래스를 메소드 내부에 선언하여 지역변수의 형태로 사용

```
class Outer{  
    변수  
    메소드 {  
        class Local {  
            변수  
            메소드  
        }  
    }  
}  
- Outer.class  
=> Outer$1Local.class
```

- ❖ Local 내부 클래스는 포함하고 있는 메소드의 지역 변수를 사용할 수 없으며 final 변수와 외부 클래스의 멤버 변수나 메소드 사용 가능
- ❖ 메소드의 호출이 끝나면 자동으로 소멸

# 실습(LocallInner.java)

```
public class LocallInner {  
    int a = 100;//멤버 변수  
    public void innerTest(int k){  
        int b = 200;// 지역변수  
        final int c = k; //상수  
        class Inner{  
            // Local 내부 클래스는 외부클래스의 멤버 변수와  
            // 상수들만 접근이 가능하다.  
            public void getData(){  
                System.out.println("member field a : "+a);  
                System.out.println("int b : "+b);  
                System.out.println("final int c : "+c);//상수 사용  
            }  
        }  
        Inner i = new Inner();//메소드내에서 Local 내부 클래스 생성  
        i.getData();//생성된 reference를 통해 메소드 호출  
    }  
}
```



# 실습(Locallnner.java)

```
public static void main(String[] args) {  
    Locallnner outer = new Locallnner();  
    outer.innerTest(1000); //외부 클래스의 멤버 메소드 호출  
}  
}
```

# 5. Anonymous Inner Class

- ❖ 클래스를 이름 없이 객체를 바로 생성해서 사용
- ❖ 객체를 생성하는 시점에 필요한 메소드를 완성해서 사용
- ❖ 특정 순간에만 사용하는 클래스가 필요한 경우에 사용
- ❖ GUI(윈도우 프로그래밍, 안드로이드) 프로그래밍의 이벤트 Listener 들이 이 방법을 많이 사용
- ❖ 인터페이스를 구현하는 경우 메소드가 1개이면 람다 사용 가능 – Android Studio 에서는 자동 변경

```
class Outer{  
    변수  
    메소드 {  
        new 클래스(){  
            메소드  
            {}  
        }.메소드;  
    }    }  
}
```

– Outer.class  
=> Outer\$1.class

# 실습(AnonymousInnner.java)

```
public class AnonymousInnner{  
    interface TestInter{  
        int data = 10000;  
        public void printData();  
    }  
    public void test(){  
        new TestInter(){  
            public void printData()//미완성된 것을 완성하여  
            {  
                //JVM이 확인 가능하도록 해준다.  
                System.out.println("data : "+data);  
            }  
        }.printData();  
    }  
    public static void main(String[] args){  
        new AnonymousInnner().test();  
    }  
}
```

Problems @ Javadoc Declaration

<terminated> AnonymousInnner [Java Application]  
data : 10000

# 6. 내부 인터페이스

- ❖ 클래스 내부에 인터페이스 선언 가능
- ❖ 클래스 내부에 인터페이스가 선언되면 static으로 간주해서 외부 객체 생성없이 사용이 가능 – 안드로이드의 이벤트 처리 인터페이스가 이 구조로 많이 만들어져 있음

# 객체 간의 데이터 공유

# 데이터 공유

- ❖ 동일한 클래스로부터 생성된 객체는 클래스에 static 필드를 만들어서 사용
- ❖ 동일한 클래스가 아니고 서로 아무런 관계가 없는 클래스인 경우에는 싱글톤 클래스를 만들어서 데이터를 공유 - 하나의 public 클래스를 만들고 public static 변수를 만들어서 사용하는 방법도 있지만 권장하지 않음

# static 변수 – StaticShare.java

```
public class StaticShare {  
    //클래스로부터 만들어진 모든 객체가 공유  
    public static String share;  
    //클래스로부터 만들어진 객체가 각각 소유  
    public int each;  
}
```

# static 변수 – Main.java

```
public class Main {  
  
    public static void main(String[] args) {  
        StaticShare obj1 = new StaticShare();  
        StaticShare obj2 = new StaticShare();  
        obj1.share = "안녕하세요";  
        obj1.each = 100;  
  
        //share는 static 이므로 obj1이 변경한 내용이 출력  
        System.out.println(obj2.share);  
        //each는 static이 아니므로 자신의 값 출력  
        System.out.println(obj2.each);  
    }  
}
```

# 생성에 관련된 패턴

## ❖ 싱글톤 패턴

- ✓ 클래스가 하나의 객체만 생성할 수 있는 디자인 패턴
- ✓ 시스템 전체에 공통적으로 적용되는 설정정보를 보관하는 관리자 클래스나 전역 변수를 소유한 클래스를 만들 때 주로 이용
- ✓ 적용 방법
  - 생성자를 private으로 만들어서 외부에서 인스턴스 생성할 수 없도록 함
  - 자신의 타입으로 된 static 변수를 선언
  - static 변수가 null이면 생성해서 리턴하고 그렇지 않으면 그냥 리턴하는 static 메서드를 생성
- ✓ jdk의 클래스 중에서 객체 생성 시 생성자를 호출하지 않고 static 메서드를 이용해서 객체를 리턴받는 클래스는 singleton 패턴이 적용된 클래스 인 경우가 있음

# 실습(Singleton.java)

```
class OnlyOne {  
    static OnlyOne obj = null;  
    private OnlyOne(){  
    }  
    public static OnlyOne getInstance(){  
        if(obj==null)  
            obj = new OnlyOne();  
        return obj;  
    }  
}  
  
public class Singleton{  
    public static void main(String[] args) {  
        OnlyOne obj1 = OnlyOne.getInstance();  
        OnlyOne obj2 = OnlyOne.getInstance();  
  
        System.out.println("obj1의 해시코드:" + obj1.hashCode());  
        System.out.println("obj2의 해시코드:" + obj2.hashCode());  
    }  
}
```

# UML



# 1.UML

## UML

□ 분석, 설계를 비주얼화, 문서화하기 위한 그래픽 언어

□ Unified

○ 이전의 OO 방법들의 통합

□ Modeling

○ 객체지향 분석 설계를 위한 비주얼 모델링

□ Language

○ 모형화된 지식(의미)을 표현



# 1.UML

## ❖ UML

- ✓ 시스템에 대한 지식을 찾고 표현하기 위한 언어
- ✓ 시스템을 개발하기 위한 탐구 도구
- ✓ 비주얼 모델링 도구
- ✓ 근거가 잘 정리된 가이드 라인
- ✓ 분석, 설계 작업의 마일스톤
- ✓ 실용적 표준
- ✓ 비주얼 프로그래밍 언어나 데이터베이스 표현도구 아님
- ✓ 개발 프로세스나 품질 보증 방안이 아님

# 1.UML

## 클래스 나타내기

### □ 박스 위에 클래스 이름

- 추상 클래스는 이탈릭체
- 인터페이스 클래스는 <>interface<> 추가

### □ 속성

- 객체가 가지는 모든 필드를 포함

### □ 오퍼레이션/메소드

- 아주 흔한 메소드(get/set)는 생략
- 상속된 메소드도 포함할 필요 없음

#### Rectangle

```
- width: int  
- height: int  
/ area: double  
+ Rectangle(width: int, height: int)  
+ distance(r: Rectangle): double
```

#### Student

```
-name:String  
-id:int  
totalStudents:int  
  
#getID():int  
+getName():String  
~getEmailAddress():String  
+ getTotalStudents():int
```

# 1.UML

## 클래스 속성

### □ 속성(필드, 인스턴스 변수)

- visibility name: type[count] = default value

- visibility: + public  
# protected  
- private  
~ package(디폴트)  
/ derived

- Underline static variable

- 파생된 속성: 저장되지 않고 다른 속성값으로부터 계산됨

### Rectangle

```
- width: int
- height: int
/ area: double
+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double
```

### Student

```
-name:String
-id:int

```

```
#getId():int
+getName():String
~getEmail Address():String

```

# 1.UML

## 클래스 오퍼레이션/메소드

### □ 오퍼레이션/메소드

- visibility name(**parameters**) : *return\_type*
- visibility: + public  
# protected  
- private  
~ package(디폴트)
- Underline static method
- 파리메타 타입 (name: type)
- 생성자나 리턴 타입이 void인 경우는  
**return\_type** 생략

| Rectangle                            |
|--------------------------------------|
| - width: int                         |
| - height: int                        |
| / area: double                       |
| + Rectangle(width: int, height: int) |
| + distance(r: Rectangle): double     |

| Student                   |
|---------------------------|
| -name:String              |
| -id:int                   |
| <u>total Students:int</u> |
| #getID():int              |
| +getName():String         |
| -getEmailAddress():String |
| +getTotalStudents():int   |

# 1.UML

## 클래스 사이의 관계

### □ 일반화(generalization): 상속(isa) 관계

- 클래스 사이의 상속
- 인터페이스 구현

### □ 연관(association): 사용(usage) 관계(3 종류)

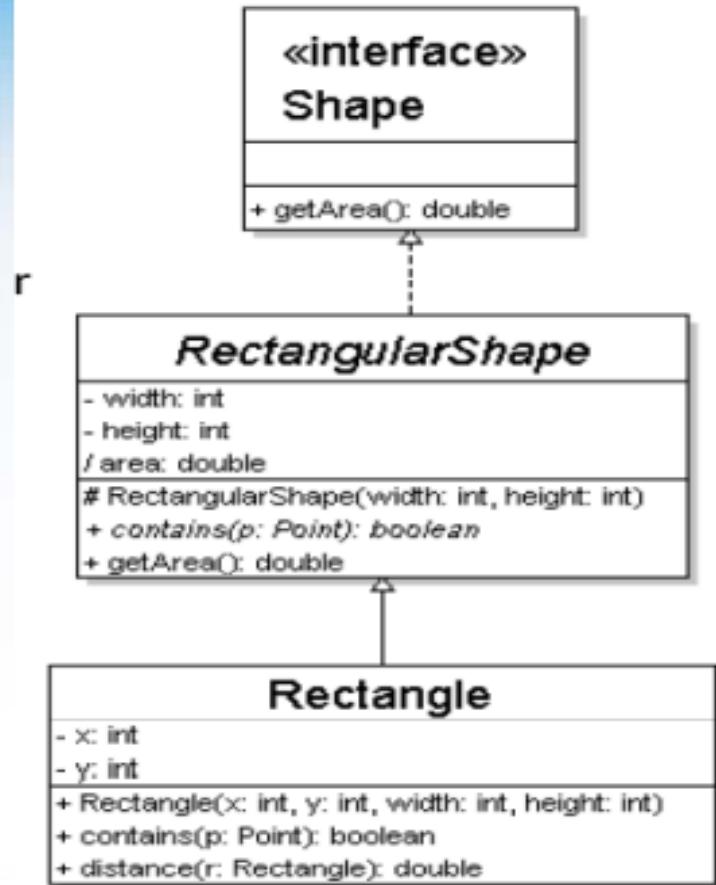
- 의존
- 집합(aggregation): 어떤 클래스가 다른 클래스의 모임으로 구성
- 합성(composition): 포함된 클래스가 컨테이너 클래스가 없이는 존재할 수 없는 집합관계의 변형

# 1.UML

## 일반화 관계

### □ 일반화(상속)

- 부모를 향한 화살표로 표시되는 하향 계층 관계
- 선/화살표는 부모 클래스의 종류에 따라 다름
  - ❖ 클래스:  
    실선/검은 헤드 화살표
  - ❖ 추상 클래스:  
    실선/흰 헤드 화살표
  - ❖ 인터페이스:  
    점선/흰 헤드 화살표



# 1.UML

## 연관 관계

연관(association): 어떤 클래스의 인스턴스가 작업을 수행하기 위하여 다른 클래스를 알아야 하는 함

### 1. 다중도(multiplicity)

- \* ⇌ 0, 1, or more
- 1 ⇌ 정확히 1개
- 2..4 ⇌ 2개 내지 4개
- 3..\* ⇌ 3개 이상

### 2. 이름 – 객체들의 관계 이름

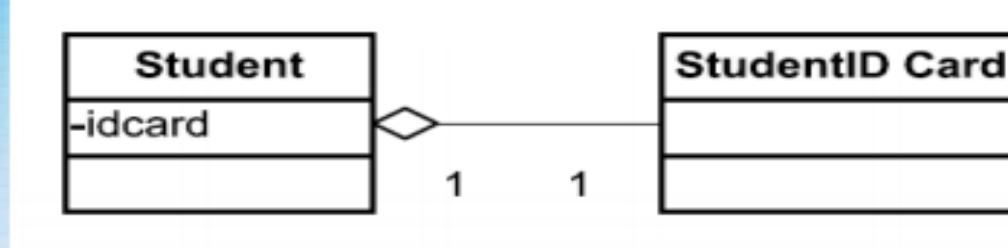
### 3. 방향성(navigability) – 질의의 방향, 객체 사이의 선으로 표시하며 양쪽 방향인 경우는 화살표시 없음

# 1.UML

## 연관 관계의 다중도

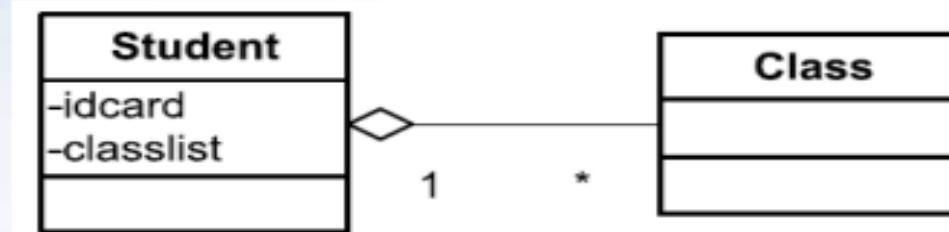
### □ 1 대 1

- 학생 1명이 학생증(id card) 한 개만을 가진다.



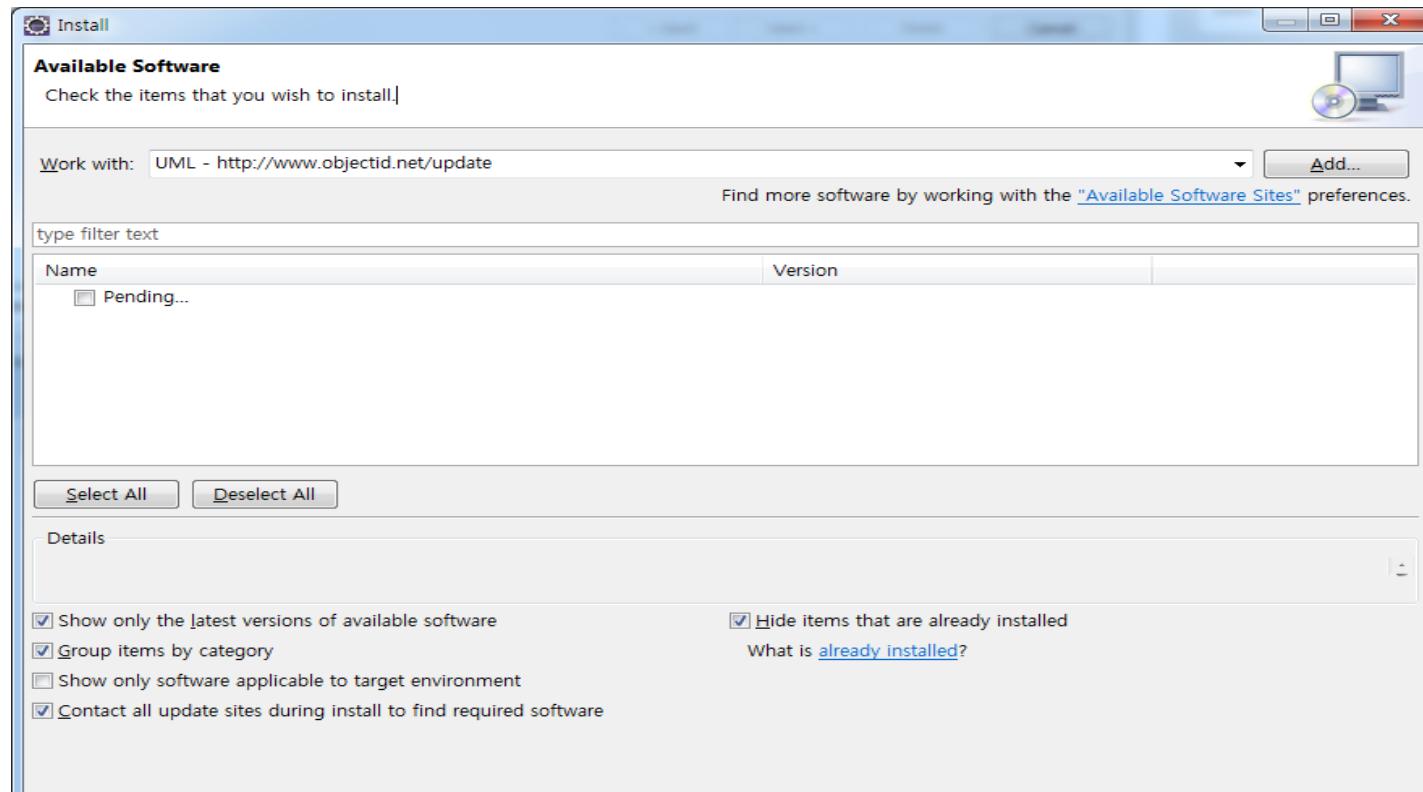
### □ 1 대 다

- 학생 1명이 여러 클래스를 수강할 수 있다.



# 2. Eclipse Class Diagram

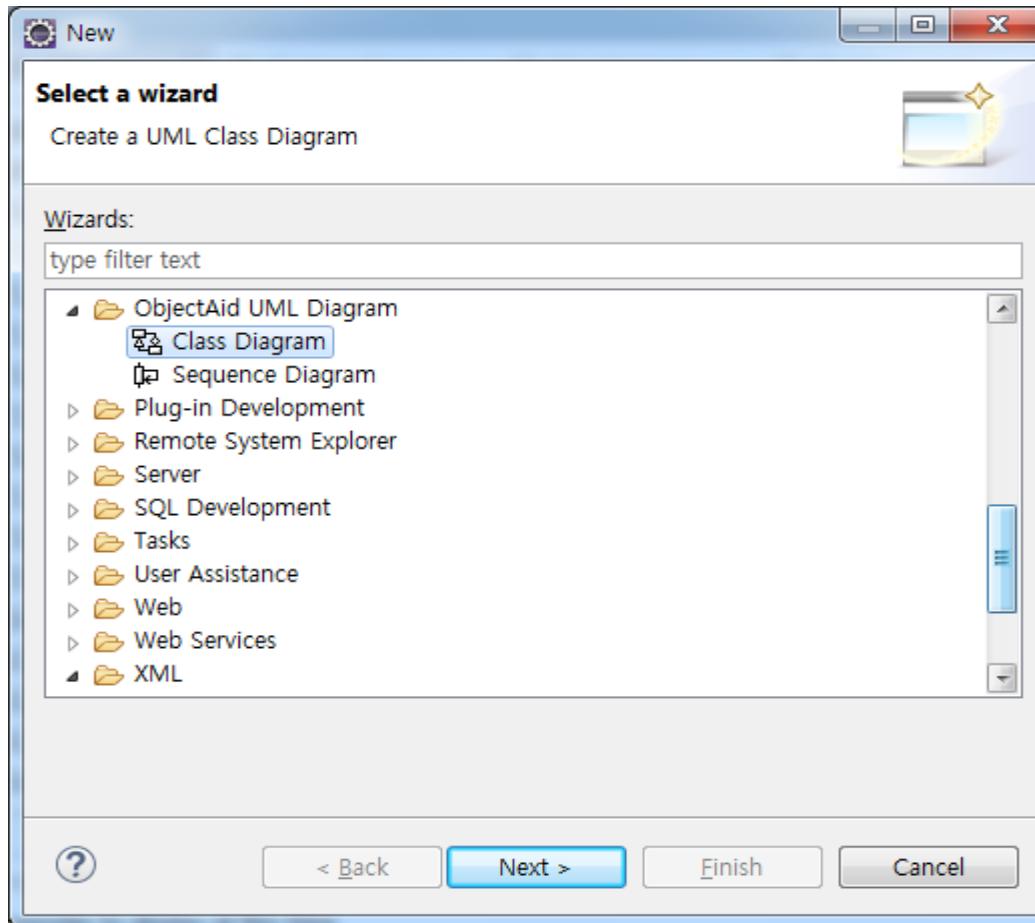
- ❖ Plug in 설치 : [help] – [Install New Software]
  - ✓ <http://www.objectaid.com/update/current>



# 2. Eclipse Class Diagram

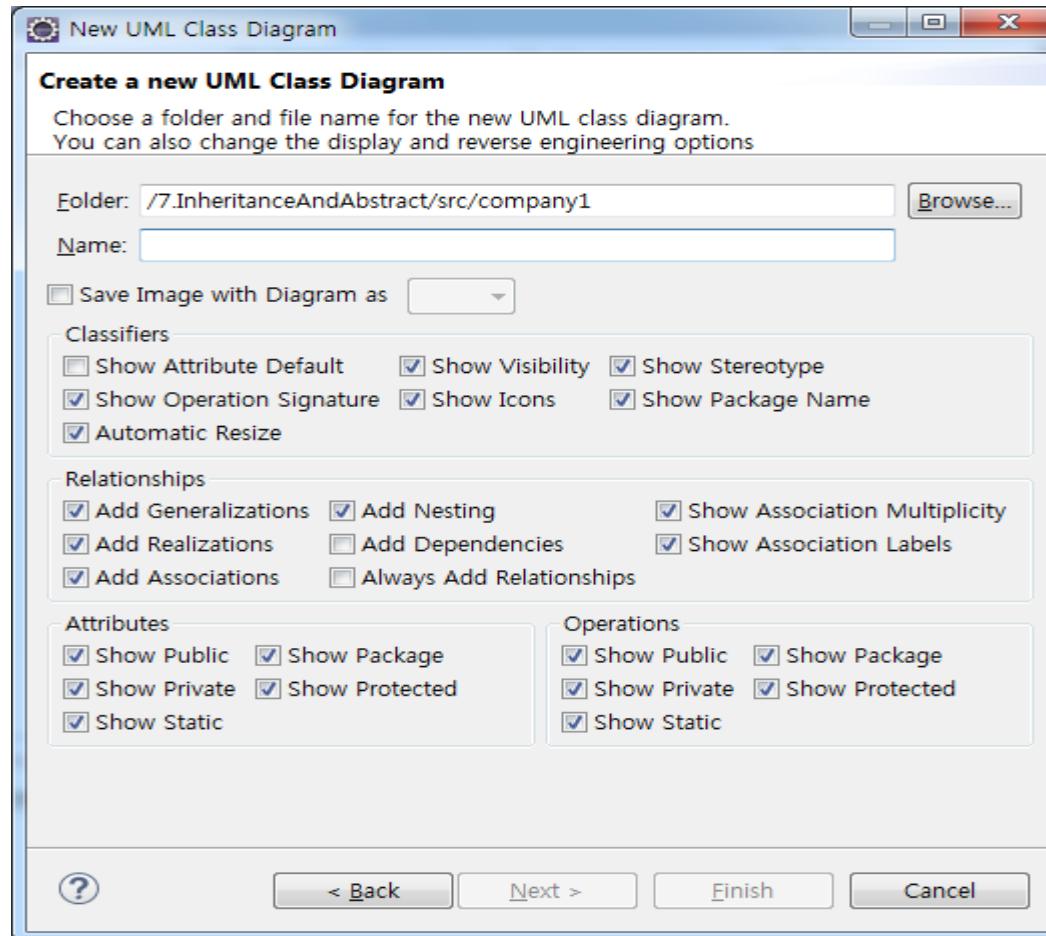
- ❖ Diagram 만들기

- ✓ File > New > Others > ObjectAid UML Diagram > Class Diagram



# 2. Eclipse Class Diagram

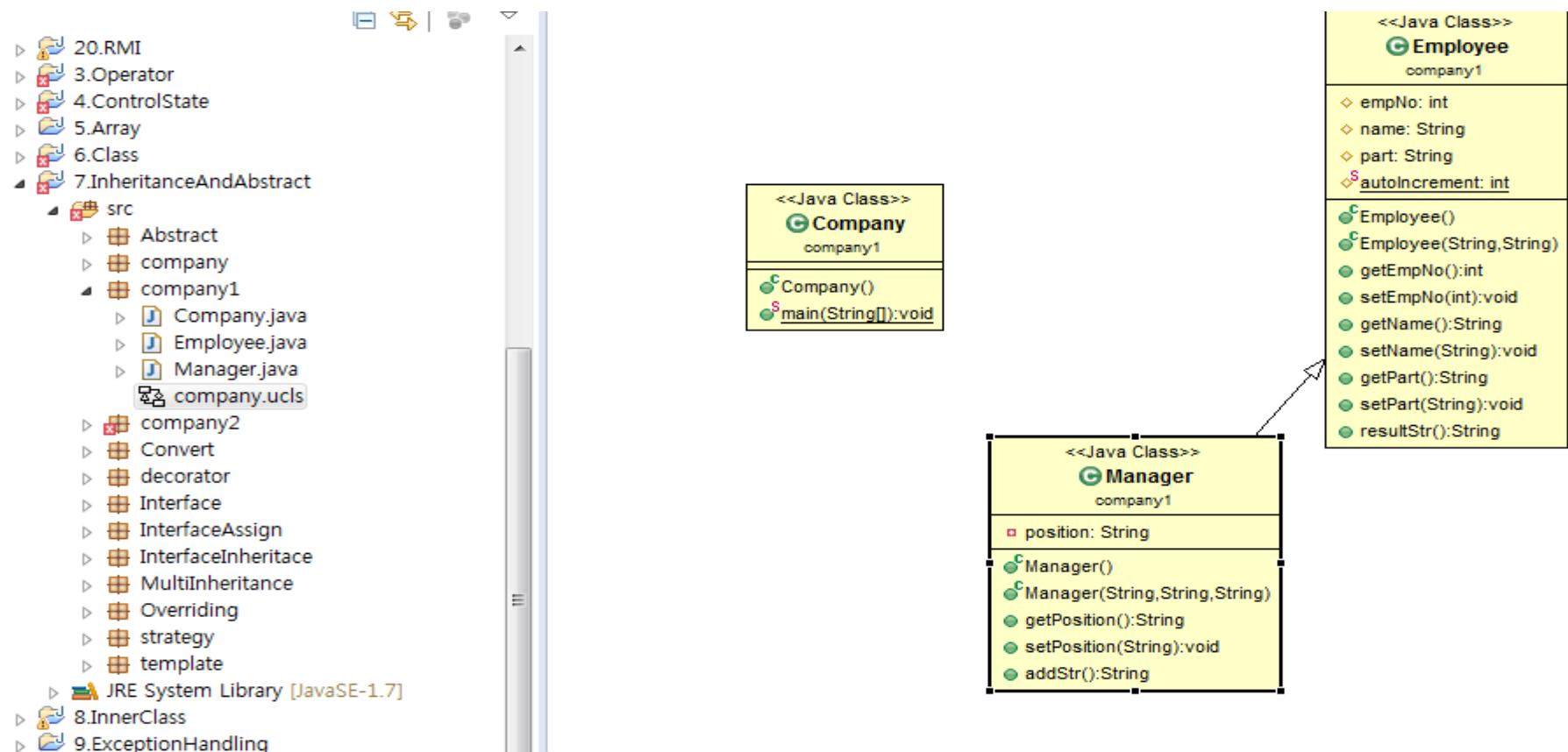
- ❖ Diagram 만들기
  - ✓ 파일명과 이미지 저장 여부 설정



# 2. Eclipse Class Diagram

## ❖ Diagram 만들기

- ✓ 다이어그램에 포함시킬 인터페이스와 클래스 파일을 Drag and Drop



# 2. Eclipse Class Diagram

- ❖ Diagram 만들기

- ✓ 이미지로 저장하고자 하는 경우는 save as를 이용해서 이미지 파일의 확장자로 저장

