

능동적 사고 방식의

java

강사 박주병

Park Ju Byeong

Park Ju Byeong



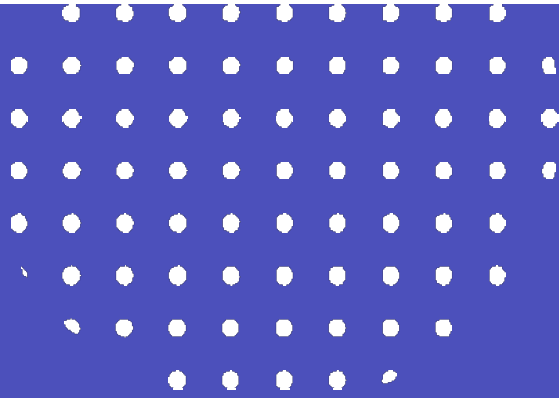
Part10 객체지향2 .

01 패키지와 import

02 제어자

03 접근제어자

04 실습 문제



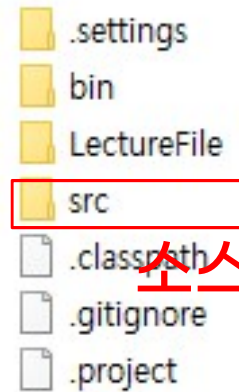
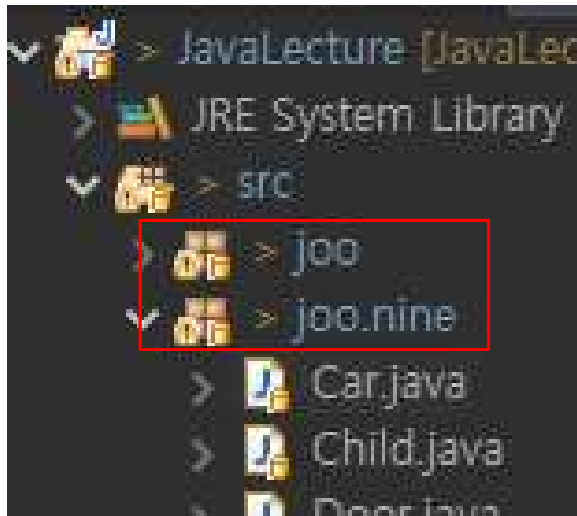
01 —

패키지와 import



패키지

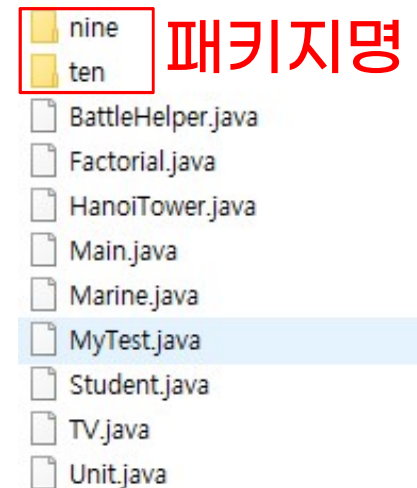
- 클래스를 분류하는 폴더



소스코드 폴더



패키지명



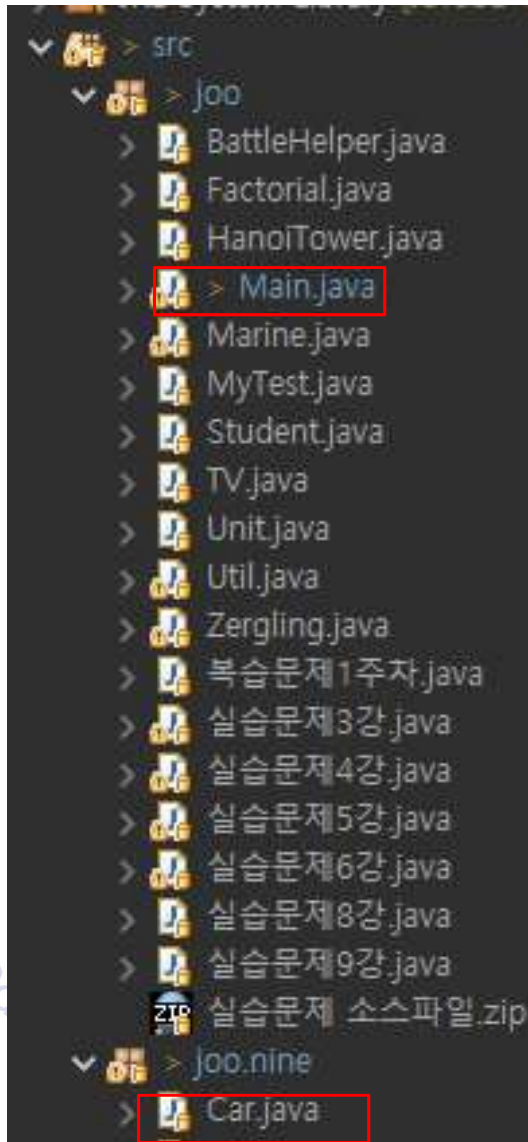
패키지명

패키지는 실제로 폴더로 생성되어 관리된다.

Main클래스와 엔트리포인트

- Main클래스는 일반적으로 main메서드 하나만을 가지며 엔트리포인트를 가지는 클래스 역할을 한다.

```
public class Main {  
    public static void main(String[] args) {  
        프로그램의 시작 지점이 되는 특별한 메서드  
    }  
}
```

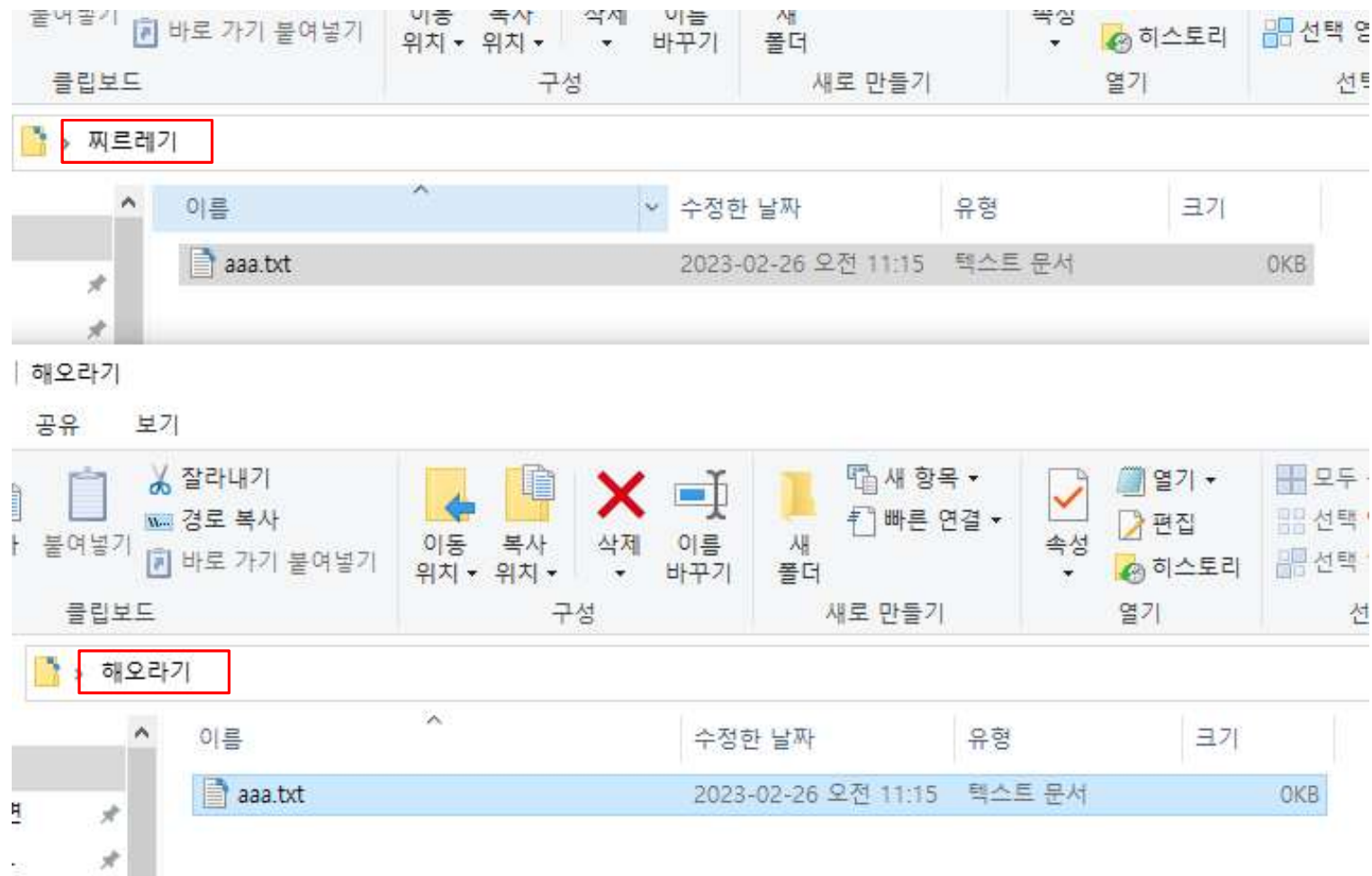


```
1 package joo;  
2  
3 import joo.nine.Car;  
4  
5  
6  
7 public class Main {  
8  
9  
10  
11     public static void main(String[] args) {  
12         |  
13         Car test = new Car();  
14     }  
15 }
```

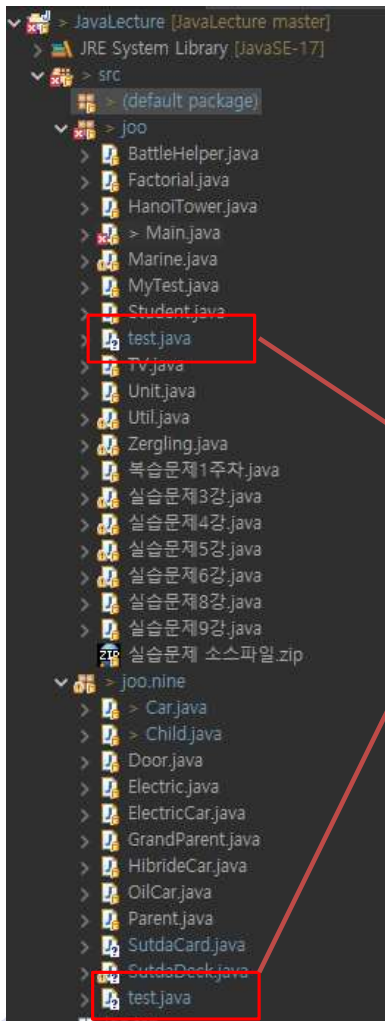
현재 파일의 패키지명
(하나의 패키지명만 가질수 있다.)

다른 패키지의 클래스를 사용하기 위해서
import를 해야 한다.

왜 쓰는걸까?



폴더가 다르면 파일명이 같아도 된다.

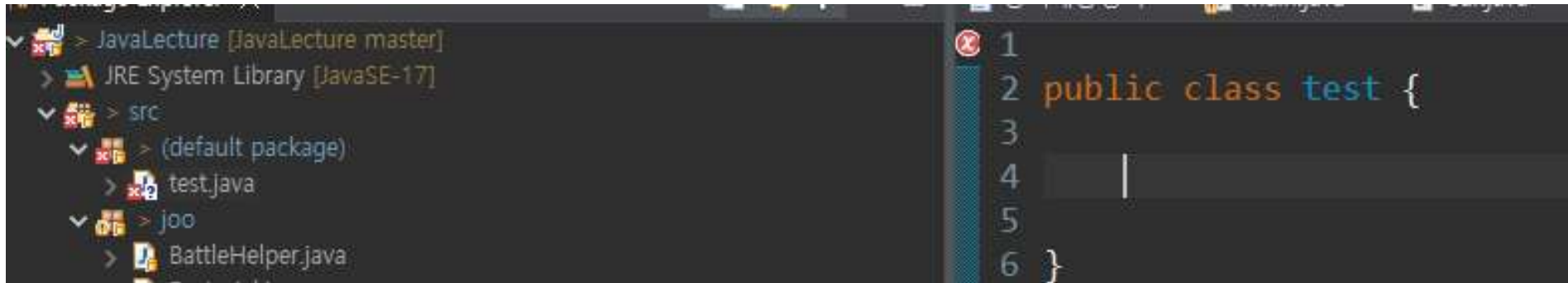


패키지가 다르면 클래스명이 같아도 된다.

패키지 규칙

- 모든 클래스는 반드시 하나의 패키지 안에 속해야 한다.(자바 버전에 따라 다를수 있음)
- 대소문자 모두 사용 가능하지만 클래스명과의 구분을 위해 소문자만 쓰는것이 관례이다.
- 문법적 규칙은 아니지만 일반적으로 도메인 형식을 거꾸로 만드는것이 관례이다.
ex(com.naver.blog) -> 네이버 블로그 개발과 관련된 클래스 패키지

지금까지 패키지를 만들지 않아도 씻는데??



패키지를 만들지 않으면 자동으로 디폴트 패키지를 만든다

* **java** 버전에 따라 패키지가 없어도 되나
자바9에서부터 모듈이 추가됨에 따라 무조건 패키지를 만들어야 된다.

import

- 다른 패키지의 클래스를 가져올때 사용한다.

```
2
3 import java.util.Scanner;
4
5 public class Main {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10
11         Scanner sc = new Scanner(System.in);
12     }
```

import

```
public class Main {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        java.util.Scanner sc = new java.util.Scanner(System.in);  
    }  
}
```

import 하지 않아도 패키지명을 다 적어주어 사용가능하다.

import는 성능에 영향을 주지 않는다.



```
4
5 import joo.nine.Car;
6 import joo.nine.Child;
7
8
9 public class Main {
10
11
12
13     public static void main(String[] args) {
14
15         Car test = new Car();
16         Child child = new Child(null, 0);
17     }
18 }
```

클래스명

```
4
5 import joo.nine.*;
6
7
8 public class Main {
9
10
11
12     public static void main(String[] args) {
13
14         Car test = new Car();
15         Child child = new Child(null, 0);
16
17         System.out.print("");
18     }
19 }
```

모든 클래스

java.lang 패키지

- String, System, Math 등등 자주 쓰이는 기본 클래스들이 들어있다.

```
1 package joo;  
2 |  
3 public class Main {  
4  
5  
6  
7     public static void main(String[] args) {  
8  
9         String name = "이름";  
10        System.out.print("");  
11        /*  
12
```

```
5  
6 package java.lang;  
7
```

String, System은 import 하지 않아도 써지는데??

java.lang 패키지는 매우 빈번하게 쓰이므로 자바에서
자동으로 **import** 해준다.

static import

```
System.out.print("");
```

클래스

멤버변수(객체)

PrintStream의 메서드

```
* @since 1.0
*/
public final class System {
    /* Register the natives via the static
    *
    * The VM will invoke the initPhase1 me
    * of this class separate from <clinit>
    */
    public static final PrintStream out = null;
}
```

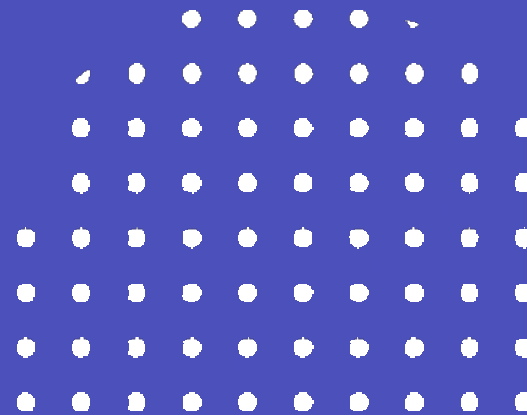
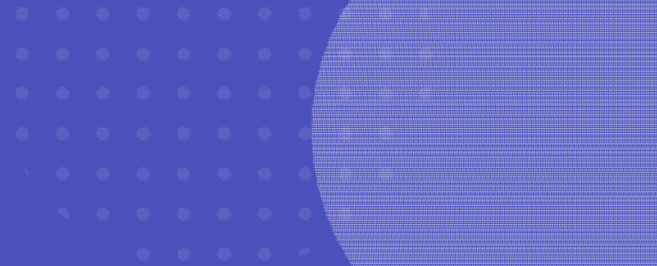
static import

```
• import static java.lang.Math.random;  
import static java.lang.System.out;  
  
public class Main {  
  
• public static void main(String[] args) {  
    String name = "이름";  
    out.print("");  
    random();|
```

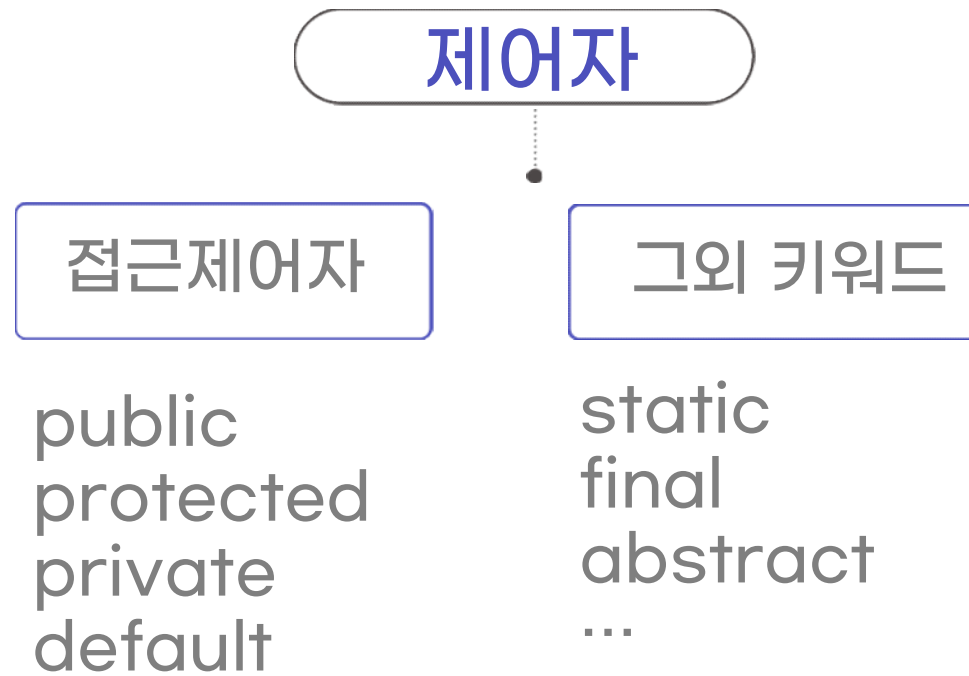
오히려 가독성을 떨어뜨리고 성능의 이점도 없어 잘 사용하지 않는다.

— 02

제어자



제어자



```

7 public class Main {
8
9
0
1 public static final String name="이름";
2
3 public static void main(String[] args) {
4
5     String name = "이름";
6
7     out.print("");

```

```

final static public String name="이름";
static public void main(String[] args) {

```

제어자들끼리 순서는 상관 없으나 일반적으로
접근제어자를 가장 먼저 작성한다.

final (변수에 사용)

- 변하지 않는이라는 의미
- 변수를 상수(변하지 않는값)으로 지정한다.

```
public class Main {  
    public static void main(String[] args) {  
        final double PI = 3.14;  
  
        PI = 10; // 한번 초기화 되면 수정할수 없는 상수  
    }  
}
```

```
final class Card{  
    final int number;  
    Card(int number)  
    {  
        this.number = number;  
    }  
}
```

선언과 동시에 혹은 생성자에서 딱
한번 초기화 가능

상수인데 생성자에서 초기화가 가능하게 풀어준건 왜일까요?

final 변수 초기화

- 변수 선언즉시 혹은 생성자에서 최초 1번만 초기화 가능하다.

```
class People
{
    final String jumin="951225-1234567";
}

public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        People p = new People();
        p.jumin = "901225-1234567";
    }
}
```

객체마다 주민번호가 달라야 하는데 그렇다고 final을 안할순 없는데…?

```
class People
{
    final String jumin;
```

```
    People(String jumin)
```

```
{
```

```
        this.jumin = jumin;
```

```
}
```

```
}
```

명시적 초기화 대신 생성자에서
초기화 하면된다.

```
public class Main {
```

```
    public static void main(String[] args) {
        // TODO Auto-generated method stub
```

```
        People p = new People("901225-1234567");
```

```
        People p1 = new People("650512-2234567");
```

```
    }
```

객체마다 다르게 초기화
가능하며 한번 정해지면 바꿀수
없다.

final (메서드에 사용)

- 해당 메서드는 오버라이딩 할수없다.

```
7  class a{
8
9  public final void test()
10 {
11
12 }
13 }
14
15 class b extends a
16 {
17 public final void test()
18 {
19 |
20 }
21 }
```

→ 오버라이딩 불가능

final (클래스에 사용)

- 상속될수 없다.

```
final class a{  
    public final void test()  
    {  
    }  
}
```

```
class b extends a  
{  
    |  
}
```

→ a클래스는 final 이므로 상속 불가능

abstract(중요)

- 추상의, 미완성의
- 객체 생성을 못하게 막는다.
- abstract가 적용된 클래스를 추상클래스라고 부른다.

```
4
5 abstract class Person
6 {
7
8 }
9
10 public class Main {
11
12     public static void main(String[] args) {
13
14         Person p = new Person();
15
16     }
17
18 }
```

추상클래스

객체 생성 불가능

abstract(중요)

- 메서드의 내용이 없을때 사용한다.
- abstract가 적용된 메서드를 추상메서드 라고 부른다.
- 추상메서드는 가진 클래스는 반드시 추상클래스가 되어야 한다.
(메서드 내용이 없기에 객체화 할수 없기때문이다.)
- 추상메서드를 가진 클래스를 상속받을시 오버라이딩하여 내부를 채워 넣어야 한다.아니면 자식또한 추상클래스가 되어야 한다.

```
1
2
3
4
5 abstract class Person
6 {
7     abstract void go();
8
9 }
10
```

추상메서드

메서드의 내부가 없다.

객체를 만들지도 못하는 추상 클래스
내용이 비어 있는 추상 메서드

왜 필요 할까?

사람, 책, 노트북 등등은 실제로 존재하는것인가?



```
Car car = new Car();
```

현실 세계와는 맞지 않는 코드이다.
상속의 용도로만 제한을 뒤편해야 한다.

추상메서드는 왜 필요할까?

```
abstract class Animal
```

```
{
```

```
    abstract void go();
```



```
}
```

상속받는 자식마다 내용이 다르다.

```
class Tiger extends Animal
```

```
{
```

추상메서드를 반드시
오버라이딩 해야 한다.

```
}
```

```
abstract class Animal
```

```
{
```

```
    abstract void go();
```

```
}
```

```
class Tiger extends Animal
```

```
{
```

```
    void go()
```


```
{
```

```
        System.out.println("기어간다");
```

```
}
```

```
}
```

자식마다 다르다면 그냥 부모에서 안만들면 되잖아?

```
public class Animal {  
      
}
```

```
public class Tiger extends Animal{  
    void go()  
    {  
        System.out.println("달려간다.");  
    }  
}
```

```
public class Bird extends Animal{  
    void move()  
    {  
        System.out.println("날아간다");  
    }  
}
```

Tiger는 go()를 사용하고 Bird를 움직일때는 move()를 사용해야해
다른 동물들은...???

```
class Tiger extends Animal
{
    void go()
    {
        System.out.println("기어간다");
    }
}

class Bird extends Animal
{
    void go() {
        System.out.println("날아간다");
    }
}
```

일관된 메서드 시그니처를 강요할수 있다.
(더 자세한건 다형성에서 다룬다)

```
class Card
{
    final int number =3;

    Card()
    {

    }

    abstract void Shuffle();
}
```



```
abstract class Card
{
    final int number =3;

    Card()
    {

    }

    abstract void Shuffle();
}
```

추상 메서드가 있다면 해당 클래스는 인스턴스화 할 수 없다.

추상메서드를 가진 클래스는
추상클래스가 될 수밖에 없다.

1-1 실습문제(normal)

Phone, Galaxy, iPhone 클래스를 만들자.

- Galaxy, iPhone 클래스는 Phone 클래스를 상속 받는다.
- Phone 클래스는 객체화 될 필요가 있는가?
- 모든 클래스는 String phoneNumber 멤버변수를 가진다.

1-1 문제풀이(normal)

1-2 실습문제(normal)

People 클래스를 만들고 멤버변수마다 적절한 제어자를 사용하자.

- 멤버변수

String name

int age

String juminNumber

String gender

- 한번 정해지면 변경 될수 없는 값들은 무엇일까?

- 멤버변수를 초기화 할수 있는 생성자를 만들자.

1-2 문제풀이(normal)

1-3 실습문제(normal)

앞서 만든 People 타입의 객체배열을 만들어 아래와 같이 출력해보자

- Object클래스의 toString()을 오버라이딩 해서 사용해보자.
- Object클래스의 toString()의 원형은 public String toString()이다

(@Override 어노테이션을 이용하여 제대로 오버라이드 된건지
체크해볼수있다.)

```
People[] list = new People[5];
```

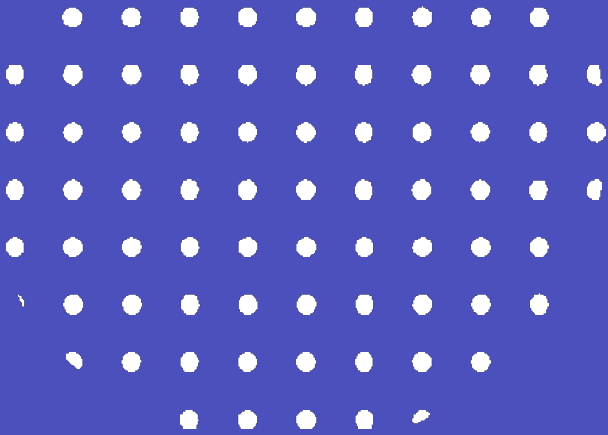
```
list[0]= new People("홍길동1",25,"000825-3456789" , "남자");
list[1]= new People("홍길동2",30,"950825-1456789" , "남자");
list[2]= new People("홍길동3",25,"000825-4456789" , "여자");
list[3]= new People("홍길동4",25,"000825-2456789" , "여자");
list[4]= new People("홍길동5",25,"000825-2456789" , "여자");
```

```
for(People p : list)
{
    System.out.println(p.toString());
}
```

<terminated> Main (12) [Java Application] C:\Users\zest1\p2\pool\plugins\org.eclipse.justj.openjdk

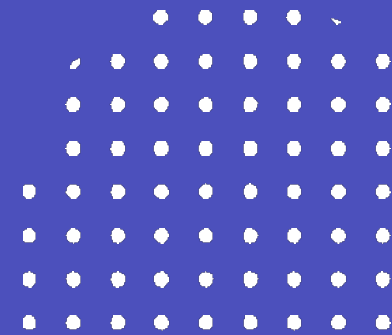
이름:홍길동1	나이:25	주민번호:000825-3456789	성별:남자
이름:홍길동2	나이:30	주민번호:950825-1456789	성별:남자
이름:홍길동3	나이:25	주민번호:000825-4456789	성별:여자
이름:홍길동4	나이:25	주민번호:000825-2456789	성별:여자
이름:홍길동5	나이:25	주민번호:000825-2456789	성별:여자

1-3 문제풀이(normal)



03

접근 제어자

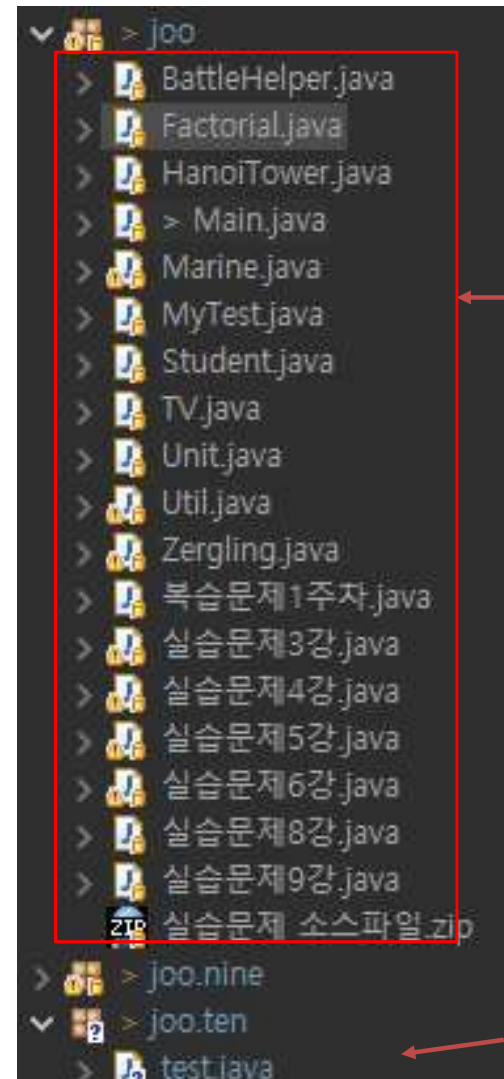


자바의 영역

```
6
7 class a
8 {
9
10
11     void test()
12     {
13
14     }
15
16 }
17
```

← 클래스 내부

← 메서드 내부



← 같은 패키지 영역

← 외부 패키지

접근제어자

- public : 제한없이 어디서든 접근가능
- protected : 같은 패키지내에서, 혹은 다른 패키지더라도
자손클래스에서 접근가능
- default : 같은 패키지내에서 접근가능
- private : 같은 클래스 내에서만 접근 가능

```
public int a;
```

```
protected int a;
```

```
int a;
```

```
private int a;
```

대상	접근제어자
클래스	public, default
메서드	전부 사용
멤버변수	전부 사용

멤버변수의 접근제어(private)

- 클래스 외부에서는 사용할수 없다.

```
class test
{
    private String name;

    void a()
    {
        name = "클래스내부 사용가능";
    }
}

public class Main {
    public static void main(String[] args) {

        test t= new test();
        t.name = "외부에서 사용불가능";
    }
}
```

내부에서는 사용가능

private 이라서 클래스 외부에서는 사용할수 없다.

멤버변수의 접근제어(default)

- 같은 패키지 내에서 사용가능

```
class test
{
    String name;

    void a()
    {
        name = "클래스내부 사용가능";
    }
}

public class Main {

    public static void main(String[] args) {

        test t= new test();
        t.name = "외부에서 사용가능";
    }
}
```

클래스의 밖이지만 같은 패키지 이므로 사용가능



```
import joo.ten.test;

public class Main {

    public static void main(String[] args) {

        test t= new test();

        t.name = "디폴트는 다른패키지에서 사용불가";
    }
}
```

다른 패키지에서는 사용 불가능

멤버변수의 접근제어(protected)

- 같은 패키지 내에서 사용가능
- 다른 패키지더라도 자식이라면 사용가능

```
package test;

public class Tiger extends Animal{

    protected String name;

    void go()
    {
        System.out.println("달려간다.");
    }
}
```

```
package test;

public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Tiger tiger = new Tiger();

        tiger.name = "호순이";
    }
}
```

Main클래스는 같은 패키지 이므로 사용가능

```
package com.test;

import test.Tiger;

public class ChildTiger extends Tiger{
    ChildTiger()
    {
        name = "새끼호랑이";
    }
}
```

→ 다른 패키지이지만 상속관계이다

→ 디폴트 멤버변수를 사용가능

```
package com.test;

import test.Tiger;

public class ChildTiger{

    Tiger t1 = new Tiger();
    ChildTiger()
    {
        t1.name = "protected라서 다른 패키지에서 자식이 아니면 사용불가능";
    }
}
```


멤버변수의 접근제어(public)

- 어디서든 사용가능

```
4 import joo.ten.test;
5
6
7 public class Main {
8
9     public static void main(String[] args) {
10
11         test t= new test();
12
13         t.name = "public은 어디서든 사용가능";
14     }
```

멤버메서드의 접근제어

- 멤버변수에서의 접근제어와 똑같은 특징을 가진다.

```
class test
{
    private void add(int a, int b)
    {

    }
}
```

메서드는 멤버변수와 동일하게 적용된다.

클래스에는 왜 private 과 protected 가 없을까?

```
private class test  
{  
  
}
```

아무도 해당 클래스를 사용 할 수 없다.

protected class

같은 패키지내에서 사용가능한거면 class 에도 적용 되어야 하는거 아닌가?

```
{  
    void test()  
    {  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        test t= new test();  
        t.name = "public은 어디서든 사용가능";  
    }  
}
```

그 역할은 이미 default가 하고 있다.
protected는 사실상 상속 관계에서
유의미한 접근제어이다.

따라서 클래스 내부의 변수나
메서드들에만 사용해도 그 역할을
다하는것이다.

퀴즈

- 부모의 생성자가 private 이라면??

```
class a
{
    private a()
    {
    }
}

class b extends a
{
    b()
    {
    }
}
```

b의 생성자가 없으니 b의 디폴트 생성자가 만들어진다.

super()가 자동으로 삽입되어야 하는데 부모의 생성자가 private 으로 접근할수 없다. ERROR!

생성자가 private 이라면 클래스에 final을 적어줌으로써 상속이 불가능하다고 알려주는것이 좋다.

2-1 실습문제(normal)

Time클래스를 만들어보자

- 아래표를 보고 멤버변수와 멤버메서드를 만들자.

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Time t = new Time();  
        t.hour = 20;  
        t.minute = 13;  
        t.second = 10;  
        System.out.println(t.toString());  
  
    }  
}
```

클래스명	Time	
멤버변수	int hour;	시
	int minute;	분
	int second;	초
메서드	String toString()	멤버변수의 값을 문자열로



Problems Javadoc Search
<terminated> Main (12) [Java Appl
20시 13분 10초

2-1 문제풀이(normal)

Park Ju Byeong

Park Ju Byeong

2-2 실습문제(normal)

앞서 만든 Time 클래스를 개선해보자

- 현재 멤버변수 hour minute second 등등은 접근제어가 디폴트 이기에 클래스 외부에서 맘대로 값을 넣을수 있다.
 - 하지만 hour : 0~23 minute : 0~59 second : 0~59로 값 입력을 제한해야 한다.
 - 외부에서 멤버변수에 직접 접근을 할수 있다면 값을 필터링 할수 없다.
- 따라서 private으로 막고 메서드를 통해 값을 초기화 하도록 해서 필터링을 해야 한다.

```
Time t = new Time();
t.setHour(30); // 30을 넘겨줘도 메서드 내부에서 적절하지 않는 값이면 멤버변수에 값을 넣지 않는다.
System.out.println(t.toString());

t.setHour(20);
System.out.println(t.toString());

t.setMinute(-50); // 적절하지 않는 값이므로 메서드 내부에서 멤버변수에 값을 넣지 않는다.
System.out.println(t.toString());

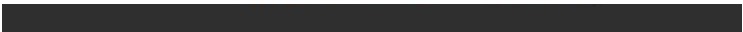
t.setMinute(50);
System.out.println(t.toString());
```

```
<terminated> main (12) [
0시 0분 0초
20시 0분 0초
20시 0분 0초
20시 50분 0초
```

2-2 문제풀이(normal)

Pa.

yeong



2-3 실습문제(hard)

싱글톤 패턴 만들기

serverConnection 클래스를 만들자. 해당 클래스는 프로그램이 서버와 통신하기 위한 기능을 만들것이다.

일반적으로 이런 클래스는 여러 개의 객체가 생성되지 못하게 막아 한 개의 객체를 돌려쓰며 서버의 자원을 절약한다. 이러한 구조를 체계화 해놓은 것이 디자인패턴중 싱글톤패턴이다.

getInstance() 메서드를 만들어 객체는 항상 1개만 유지되도록 하자.

Static과 private를 활용하자.

- 생성자를 직접 사용못하게 막아야 한다.
- getInstance() 메서드를 통해서만 객체를 가져갈수 있도록 해야 한다.
- Static을 활용해보자

```
serverConnection con = serverConnection.getInstance();

serverConnection con1 = serverConnection.getInstance();
serverConnection con2= serverConnection.getInstance();
serverConnection con3 = serverConnection.getInstance();

System.out.println(con);
System.out.println(con1);
System.out.println(con2);
System.out.println(con3);
```

```
<terminated> main [Java Application] C:\Users\zest1\p2\pool\plugins
joo.ten.serverConnection@27d415d9
joo.ten.serverConnection@27d415d9
joo.ten.serverConnection@27d415d9
joo.ten.serverConnection@27d415d9
```

2-3 문제풀이(hard)



THANK YOU



강사 박주병