

# JAVA WEEK 3

Min-su Kim

Korea univ.  
KOSMOS

October 13, 2018

# Overview

## ① More Keywords

1. final
2. static

## ② Package

1. What is Package?
2. Package Example
3. import

## ③ Classes

1. Math Class
2. String Class

## ④ More about OOP

1. Information Hiding
2. Encapsulation

# final

- final로 선언된 변수는 처음 한 번만 초기화 가능
- 상수와 같은 기능이라고 생각하면 됨
- C 언어의 const와 비슷한 것으로 생각
- Constructor를 통해서도 초기화 가능

## final Example

```
1  class Program
2  {
3      ▶ Run | 🐞 Debug
4      public static void main(String[] args)
5      {
6          final int n;
7          n = 10;
8          n = 11;
9      }
```

- 한 번만 초기화 가능 하므로 두 번째 대입 연산자에서 compile error

## final Example

```
1  class Box
2  {
3      public final int max;
4
5      public Box (int max)
6      {
7          this.max = max;
8      }
9  }
```

- Constructor를 통해서 초기화 가능

# static

- 클래스 내의 변수나 메소드 앞에 붙임.
- 인스턴스와 별개로, 클래스 내의 변수나 메소드를 사용할 수 있게 해줌
- static으로 선언하면, 따로 인스턴스를 생성하지 않고도 사용 가능.
- 즉, 메모리가 미리 할당되어 있음.

## static Example1

- 클래스를 설계할 때, 객체로부터 생성된 모든 인스턴스에 공통적으로 사용되는 것에는 static을 붙임
- 이렇게 되면 메모리의 같은 곳을 참조하기 때문에, 메모리를 아낄 수 있고 값을 변경시켜야 할 때에도 한 번에 변경 가능
- ex) 공장에서 만든 제품의 번호,

## static Example2

- static이 붙은 변수나 메소드는 인스턴스의 생성 없이도 사용 가능
- static이 아닌 변수들은 인스턴스마다 서로 다른 값을 유지하지만, static의 경우, 한 클래스로부터 생성된 인스턴스에서는 같은 값을 유지하게 됨.



## static Example2

```
1  public class test
2  {
    ▶ Run | 🐞 Debug
3      public static void main(String[] args)
4      {
5          System.out.println(Counter.n);
6
7          Counter c = new Counter();
8          System.out.println(c.n);
9      }
10 }
```

- 이 Counter 클래스를 설계

## static Example2

```
1  public class test
2  {
    ▶ Run | 🐞 Debug
3      public static void main(String[] args)
4      {
5          Counter c = new Counter();
6          System.out.println("n = " + c.n);
7          System.out.println("n = " + Counter.n);
8      }
9  }
```

- 두 번째 println처럼, static으로 선언된 변수의 경우 인스턴스를 직접적으로 사용하지 않고도 n을 사용 가능
- 1
- n = 1
- n = 1

## static Example3

- static으로 선언된 메소드에선 static이 아닌(instance라고 부름) 변수나 메소드를 사용할 수 없음

## static Example4

```
1  public class test
2  {
    ▶ Run | 🐞 Debug
3      public static void main(String[] args)
4      {
5          int    a = 1;
6          int    b = 3;
7          int    sum;
8
9          sum = add(1, 3);
10     }
11
12     int add(int a, int b)
13     {
14         return a + b;
15     }
16 }
```

## static Example4

- add는 static으로 선언되지 않음. 반면 main은 static으로 선언됨. So, error
- why? → 이전 예시(example3)에서처럼 static으로 선언된 경우, 인스턴스를 생성하지 않고도 그 변수나 메소드를 참조 가능 but, 참조하는 도중에 static이 아닌 변수나 메소드를 사용할 수 있기 때문.
- 반대로, static이 아닌 메소드에서는 static 변수나 메소드를 호출 가능. static이 아닌 것들의 메모리가 할당 되었을 때에는 static에도 당연히 메모리가 할당 되었을 것이기 때문.
- static int add로 선언했을 경우, error 없음.

## static Example5

- static 변수는 한 클래스로부터 생성된 인스턴스에서는 값을 공유

## static Example5

```
public class test
{
```


▶ Run |  Debug

```
    public static void main(String[] args)
    {
        Counter c1 = new Counter();
        Counter c2 = new Counter();
        Counter c3 = new Counter();
    }
}
```

- 1
- 2
- 3
- 같은 클래스로부터 생성되었으므로, static 변수는 같은 값을 유지

# Quiz1

```
1  public class Test
2  {
3      static int a = 0;
4      int b = 0;
5
6      public static void addOne()
7      {
8          a += 1;
9          b += 1;
10     }
11 }
```



- test.addOne(); 코드 실행 시, a와 b의 값의 변화는?



# Quiz1 Solution

- Compile error!
- static 메소드에서 인스턴스 변수인 b를 참조.
- public void addOne()으로 선언 시, 인스턴스 생성 후 사용 가능.

# Package

- 클래스를 저장하는 디렉토리(폴더).
- 클래스들을 담는 틀.
- 클래스보다 큰 개념.
- 클래스들을 정리할 수 있게 해줌.
- ex) Scanner class는 java.util package에 있음

# Package Usage

- package (패키지명);
- 위처럼 소스코드 맨 위에 선언하면, 이 클래스들이 (패키지명)의 패키지에 속한다는 것을 뜻함.

# Package Example

```
1 package house;
2
3 public class Myhouse {
4     double width;
5     double height;
6     int people;
7
8     public Myhouse()
9     {
10         this.width = 30.0;
11         this.height = 100.0;
12         this.people = 4;
13     }
14
15     public void houseInfo()
16     {
17         System.out.println(this.width);
18         System.out.println(this.height);
19         System.out.println(this.people);
20     }
21 }
```

# Package Example

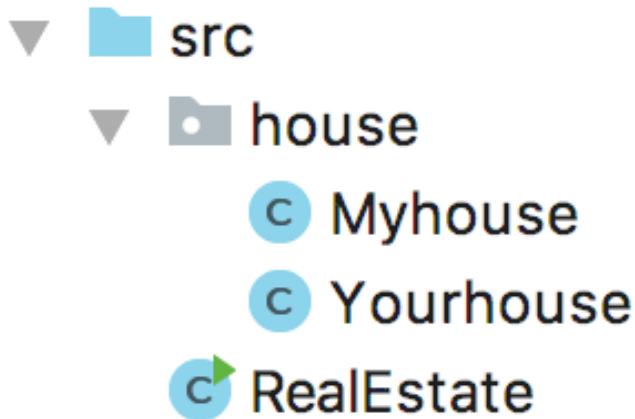
```
1 package house;
2
3 public class Yourhouse {
4     double width;
5     double height;
6     int people;
7
8     public Yourhouse()
9     {
10         this.width = 100.0;
11         this.height = 300.0;
12         this.people = 32;
13     }
14
15     public void houseInfo()
16     {
17         System.out.println(this.width);
18         System.out.println(this.height);
19         System.out.println(this.people);
20     }
21 }
22
```

## Package Example

```
1  import house.Myhouse;
2  import house.Yourhouse;
3
4  public class RealEstate {
5      public static void main(String[] args)
6      {
7          Myhouse    my = new Myhouse();
8          Yourhouse  ur = new Yourhouse();
9
10         my.houseInfo();
11         ur.houseInfo();
12     }
13 }
```

- 이런식으로, house라는 패키지 내의 클래스들을 불러서, 밖에서 사용 가능.

## Package Example



- 디렉토리 구조는 이렇게 됨.
- 앞의 c는 class라는 뜻
- .java파일(소스코드)도 같은 방식으로 존재

# Package

- 모든 클래스는 패키지에 속함.
- 패키지를 명시적으로 선언해주지 않으면 default package에 속함.
- 앞의 RealEstate class는 default package에 속함.
- eclipse에서는 잘 나올꺼예요



# Import

- 특정 패키지에 있는 클래스를 사용하기 위해 선언
- `import 패키지명.클래스명;`
- `import 패키지명.*;`
- \*은 해당 패키지 내의 모든 클래스를 불러오겠다는 의미.
- ex) `import java.util.Scanner;`

# Import Example

```
1  import house.Myhouse;
2  import house.Yourhouse;
3
4  public class RealEstate {
5      public static void main(String[] args)
6      {
7          Myhouse    my = new Myhouse();
8          Yourhouse  ur = new Yourhouse();
9
10         my.houseInfo();
11         ur.houseInfo();
12     }
13 }
```

- 패키지 내의 클래스 Myhouse, Yourhouse를 사용하기 위해 import

# Import Example

```
1  import house.*;
2
3  public class RealEstate {
4      public static void main(String[] args)
5      {
6          Myhouse    my = new Myhouse();
7          Yourhouse  ur = new Yourhouse();
8
9          my.houseInfo();
10         ur.houseInfo();
11     }
12 }
```

- 이전 예시와 조금 다르게, \*을 사용.
- But, 이전 예시와 같은 효과.

# Import

- java.lang 패키지에 있는 클래스들은 따로 import하지 않아도 그냥 사용 가능
- ex) java.lang.System, java.lang.Math, java.lang.String, ...

# Math

- 수학에 관한 변수와 메소드들이 정의되어 있는 클래스.
- 다른 패키지 내에서 불러오므로, Math 클래스 내의 사용할 수 있는 변수나 메소드는 모두 static으로 선언 되어있음.

# Math Example

```
public class Mathematics
{
    ▶ Run | 🐞 Debug
    public static void main(String[] args)
    {
        System.out.println(Math.PI);
        System.out.println(Math.sin(Math.PI));
        System.out.println(Math.max(20, 500));
    }
}
```

- 3.141592653589793
- 1.2246467991473532E-16
- 500

# String

- 예시를 통해 사용

# String

```
1  public class Strings
2  {
    ▶ Run | 🐞 Debug
3  public static void main(String[] args)
4  {
5      String str1 = "KOSMOS C";
6      String str2 = "KOSMOS JAVA";
7      System.out.println(str1.length() + str2.length());
8      System.out.println(str1.concat(str2));
9      System.out.println(str1.equals(str2));
10     System.out.println(str1.contains("KOSMOS"));
11     System.out.println(str2.contains("JAVA"));
12 }
13 }
```

- 19
- KOSMOS CKOSMOS JAVA
- false
- true
- true



# Information Hiding

- 객체 내의 정보를 외부로부터 보이지 않게 하는 방법.
- public, protected, private keyword를 사용.
- 이 keyword들을 접근 제어 지시자라고 부름.

# Information Hiding

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	O	X	X	X
default	O	O	X	X
protected	O	O	O	X
public	O	O	O	O

- public은 모든 곳에서 접근 가능
- protected는 같은 패키지 내의 클래스와 해당 클래스를 상속받은 외부 패키지의 클래스에서 접근 가능
- 아무것도 쓰지 않으면(default) 해당 패키지 내에서만 접근 가능
- private는 해당 클래스 내에서만 접근 가능

# Information Hiding Example

```
public class Minsu
{
    private int age = 20;
    private int studentNumber = 2017320168;
}
```

- 내 나이랑 학번은 비밀
- 또한, 다른 사람의 개인 정보를 외부에서 편하게 건드릴 수 있으면 안됨.
- 게임에서, 외부에서 변수를 조작하지 못하도록 private로 선언할 수 있음
- ex) 게임에서, 내가 가지고 있는 돈의 양

# Information Hiding

- 값을 얻어오거나, 자신이 직접 조작하려고 할 때는 접근할 수 있어야 함.
- 이 때, getter, setter 메소드를 만들어서 사용.

# Information Hiding Example

```
1  public class Minsu
2  {
3      private int    age = 20;
4      private int    studentNumber = 2017320168;
5
6      public int getAge()
7      {
8          return this.age;
9      }
10     public int getSN()
11     {
12         return this.studentNumber;
13     }
14     public void setAge(int age)
15     {
16         this.age = age;
17     }
18     public void setSN(int studentNumber)
19     {
20         this.studentNumber = studentNumber;
21     }
22 }
```

- 이런 식으로, 간접적으로 private 변수에 접근

# Encapsulation

- 외부에서 내부의 구조를 알 수 없음.
- 즉, 외부에서 어떤 객체의 정보를 임의로 조작할 수 없음.
- 한 객체 내에서 처리가 되게 만들기 때문에, 내부의 구조가 변해도 다른 객체에 주는 영향을 최소화 가능.
- 오류 발생이 적고 유지 보수에 용이.
- 지금까지 배운 객체지향의 개념을 통해 실현 가능(객체의 개념, 접근 제어 지시자, ...)

# D END

-끝-