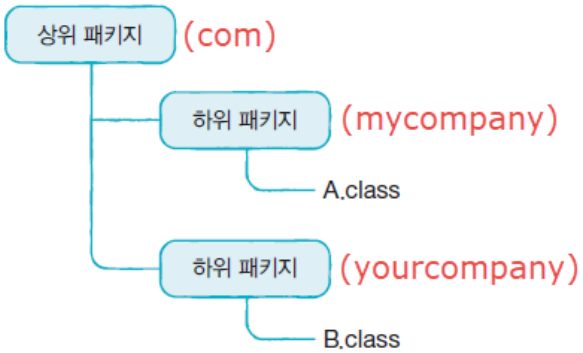
■ 패키지(package)

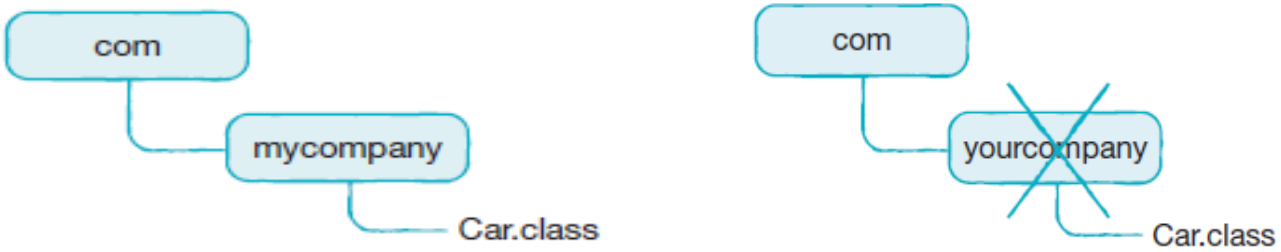
- 물리적인 형태는 파일 시스템의 폴더
- 클래스의 일부분으로 클래스를 유일하게 만들어주는 식별자 역할
- 클래스명이 동일하더라도 패키지가 다르면 다른 클래스로 인식
- 클래스의 전체이름은 패키지+클래스로 표현
  - 상위패키지.하위패키지.클래스
  - java.util.Random, java.util.Scanner
  - com.mycompany.A
  - com.yourcompany.B



■ 패키지 선언

● 클래스 작성 시 해당 클래스가 어떤 패키지에 속할 것인지를 선언

```
package com.mycompany;


public class Car { … }
```



● 이름 규칙

- 숫자로 시작 불가
- _ (underscore) 및 $ 를 제외한 특수문자 사용 불가
- java로 시작하는 패키지는 표준 API에서만 사용하므로 사용 불가
- 모두 소문자로 작성하는 것이 관례

■ import

- 사용하고자 하는 클래스가 다른 패키지에 소속되어 있는 경우
- 해당 패키지에서 가져와 사용할 것임을 명시

```
import 상위패키지.하위패키지.클래스이름;
import 상위패키지.하위패키지.*;
```

```
package com.mycompany;

import com.hankook.Tire;
[ 또는 import com.hankook.*; ]

public class Car {
 Tire tire = new Tire();
}
```

- 하위 패키지는 별도로 import

```
import com.hankook.*;
import com.hankook.project.*;
```

- 동일한 이름으로 클래스를 만든 경우는 impor와 상관없이 패키지 전체 기술

  ex) 현재 클래스명이 Random 인데 java.util.Random 클래스를 사용

  java.util.Random ran = new java.util.Random();

  ran.nextInt();

■ 접근제어자 (Access Modifier)

● 클래스 및 클래스의 구성 멤버에 대한 접근을 제한하는 역할

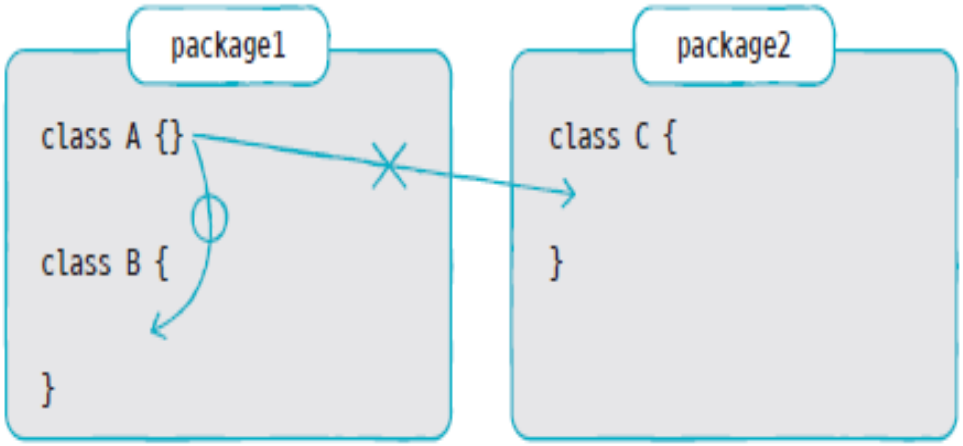
- 다른 패키지에서 클래스를 사용하지 못하도록 (클래스 제한)
- 클래스로부터 객체를 생성하지 못하도록 (생성자 제한)
- 특정 필드와 메소드를 숨김 처리 (필드와 메소드 제한)

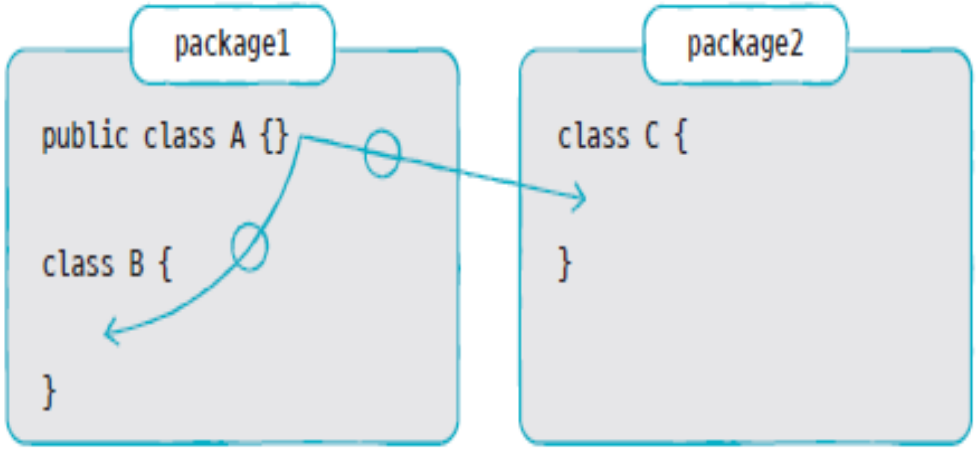
지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

```
public > protected > default > private
```

■ 접근 제한

● 같은 패키지 내에서만 사용할 것인지 다른 패키지에서 사용할 것인지 결정





■ 접근제어자 사용 (public – 어디서든 사용 가능)

```
package ch11.vo;
public class Student {
    public int grade;
    public int classNum;
    public String name;
}
```

```
package ch11.process;
public class DataProcess {
    void input(int grade, int classNum, String name) {
        Student student = new Student();
        student.grade = grade;
        student.classNum = classNum;
        student.name = name;

        // DataBase Insert
    }
}
```

■ 접근제어자 사용 (default – 동일 패키지에서만 사용 가능)

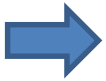
```
package ch11;
public class Student {
    int grade;
    int classNum;
    String name;
}
```

```
package ch11;
public class DataProcess {
    void input(int grade, int classNum, String name) {
        Student student = new Student();
        student.grade = grade;
        student.classNum = classNum;
        student.name = name;

        // DataBase Insert
    }
}
```

■ 접근제어자 사용 (private – 동일 클래스 내에서만 사용 가능)

```
public class Student {
    private int grade;
    private int classNum;
    private String name;
}
```

```
public class DataProcess {
    void input(int grade, int classNum, String name) {
        Student student = new Student();
        student.grade = grade;
        student.classNum = classNum;
        student.name = name;

        // DataBase Insert
    }
}
```



사용불가

■ 접근제어자 사용 (변수 private / 메소드 public)

```
package ch11.process;

public class Student {
    private int grade;
    private int classNum;
    private String name;

    public int getGrade() {
        return grade;
    }
    public void setGrade(int grade) {
        this.grade = grade;
    }
    public int getClassNum() {
        return classNum;
    }
    public void setClassNum(int classNum) {
        this.classNum = classNum;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

■ 접근제어자 사용 (변수 **private** / 메소드 **public**)

```
package ch11.process;

import ch11.vo.Student;

public class DataProcess {
    void input(int grade, int classNum, String name) {
        Student student = new Student();
        student.setGrade(grade);
        student.setClassNum(classNum);
        student.setName(name);

        // DataBase Insert
    }
}
```

■ 캡슐화 (Encapsulation – package, hidden)

```
package ch11.process;

public class Student {
    private int grade;
    private int classNum;
    private String name;

    public int getGrade() {
        return grade;
    }
    public void setGrade(int grade) {
        if(grade < 1 || grade > 4) {
            System.out.println("오류");
        } else {
            this.grade = grade;
        }
    }
    public int getClassNum() {
        return classNum;
    }
    public void setClassNum(int classNum) {
        this.classNum = classNum;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

■ default 클래스

- default 클래스
  - 동일한 패키지 내에 정의된 클래스에 의해서만 인스턴스 생성이 가능

```
package apple;
class AAA    // default 클래스 선언
{
     . . . .
}
```

```
package peal;
class BBB    // default 클래스 선언
{
     public void make()
     {
          apple.AAA inst=new apple.AAA();
          . . . .
     }
     . . . .
}
```

인스턴스 생성 불가! AAA와 BBB의 패키지가 다르므로!

파일을 대표할 정도로 외부에 의미가 있는 클래스 파일을 public으로 선언한다.

■ public 클래스

- public 클래스
  - 동일한 패키지 내에 정의된 클래스에 의해서만 인스턴스 생성이 가능
  - 하나의 소스파일에 하나의 클래스만 public으로 선언 가능
  - public 클래스 이름과 소스파일 이름은 일치해야 한다.

```
package apple;
public class AAA     // public 클래스 선언
{
     . . . .
}
```

```
package peal;
public class BBB     // public 클래스 선언
{
     public void make()
     {
          apple.AAA inst=new apple.AAA();
          . . . .
     }
     . . . .
}
```

AAA는 public 클래스이므로 어디서든 인스턴스 생성 가능!

■ 클래스와 생성자의 접근제어자

```
public class AAA
{
    AAA(){…}
    . . . .
}
```

클래스는 public으로 선언되어서 파일을 대표하는 상황! 그럼에도 불구하고 생성자가 default로 선언되어서 동일 패키지 내에서만 인스턴스 생성을 허용하는 상황!

```
class BBB
{
    public BBB(){…}
    . . . .
}
```

생성자가 public임에도 클래스가 default로 선언되어서 동일 패키니 내에서만 인스턴스 생성이 허용되는 상황!

■ 디폴트 생성자

디폴트 생성자의 접근제어 지시자는 클래스의 선언형태에 따라서 결정된다.

```
public class AAA
{
    public AAA() {…}
    . . . .
}
```

public 클래스에 디폴트로 삽입되는 생성자

```
class BBB
{
    BBB() {…}
    . . . .
}
```

default 클래스에 디폴트로 삽입되는 생성자