

클래스

OOP

❖ OOP(Object Oriented Programming)

- ✓ 객체 지향: 실제 사물의 속성(데이터) 과 기능(작업)을 분석한 후 관련 있는 속성 과 기능을 묶어서 구현하고 이를 활용하는 기법
- ✓ 객체 지향(Object Oriented)의 특징
 - 캡슐화(encapsulation) – 변수와 Method를 하나로 묶는 것으로 클래스를 만드는 것
 - ◆ 정보은닉 (information hiding) – 불필요한 정보는 숨기는 것
 - 상속성(inheritance) – 하위 클래스가 상위 클래스의 모든 멤버를 물려받는 것
 - 다형성(Polymorphism) – 동일한 메시지에 대하여 다르게 반응하는 것
- ✓ 객체 지향 프로그래밍(Object Oriented Programming)의 장점
 - 코드의 재 사용성: 새로운 코드를 작성할 때 클래스 단위로 작성해서 코드의 재 사용성을 증가시킴
 - 코드 관리 용이: 코드 간의 관계를 이용해서 적은 노력으로 쉽게 코드를 변경할 수 있으며 불필요한 코드의 중복을 제거할 수 있음
 - 신뢰성이 높은 프로그래밍을 가능하게 함: 불필요한 접근을 막을 수 있음

OOP

❖ Class 와 Instance

✓ 클래스(Class)

- 사물의 특성을 소프트웨어적으로 추상화하여 모델링 한 것
- 유사한 인스턴스를 만들기 위한 설계도
- 동일한 목적을 가진 Method(함수 또는 프로시저)와 변수(필드, 속성)의 집합
- 사용자가 만들어 사용할 수 있는 사용자 정의 자료형

✓ 객체(Object)

- 물리적으로 존재하거나 추상적으로 생각할 수 있는 모든 것
- 클래스를 기반으로 해서 만들어진 객체를 인스턴스(Instance)라고 함

✓ classpath에 설정되지 않은 클래스는 처음 호출 될 때 별도의 메모리를 할당받아(Method 영역 또는 static 영역) 생성됨

✓ 클래스는 한 번 메모리를 할당 받으면 프로그램이 종료될 때까지 수정할 수 없고 소멸되지 않고 재할당 받지 않음 - static(정적)

✓ Instance

- new 연산자를 이용해 생성
- new 연산자는 인스턴스를 생성할 수 있는 생성자를 호출해서 메모리를 할당한 후 할당 받은 메모리 참조를 리턴하는 연산자
- 인스턴스의 내용은 필요에 따라 수정할 수 있고 원하는 시점에 인스턴스를 삭제 할 수 있음 - dynamic(동적)

OOP

❖ Class 와 Instance

✓ 클래스의 구성 요소

- 속성(Property): 필드, 특성(attribute), field, state
 - ◆ static 변수
 - ◆ static 이 아닌 멤버 속성(인스턴스 변수): 인스턴스 속성
- 초기화 블록
- 생성자(Constructor): 인스턴스의 초기화를 담당
- 기능(function): method, 함수, behavior(행위)
 - ◆ static Method
 - ◆ static 이 아닌 멤버 Method(인스턴스 Method)
- 내포 클래스(Nested Class): 클래스 안에 존재하는 클래스

OOP

❖ 클래스 생성

[접근 지정자] + [클래스 특성] + class + class명 + [extends super] + [implements interface]

- ✓ 접근 지정자 - public, default(package), protected, private 등이 있음
 - protected와 private은 클래스 안에 클래스(내포 클래스)를 만들 때 만 사용 가능
- ✓ 클래스 특성으로는 abstract(반드시 상속 받아서 사용)와 final(상속할 수 없음) 그리고 static(내포 클래스에만 가능)
- ✓ extends - 상속받을 때 사용 [자바-단일 상속], 자바의 클래스는 반드시 다른 클래스로부터 상속을 받아야 하는데 Object 클래스로부터 상속받는 경우는 생략이 가능
- ✓ implements - interface를 구현하기 위해 사용
- ✓ 클래스 이름의 첫 문자는 대문자 로 시작하는 것이 관례

OOP

❖ Source File Name

- ✓ 이름: 클래스 이름 중 하나와 일치 - 대소문자 구분
- ✓ 하나의 소스 파일에 존재하는 클래스 개수: 보통 한 개지만 여러 개 가능
- ✓ 하나의 파일에 여러 개의 클래스가 존재하는 경우
 - 모두 default(접근 지정자 생략) 클래스: 접근 지정자가 생략된 클래스들로만 구성된 경우에는 파일명은 아무 클래스 이름이나 사용 가능하지만 main()이 있는 클래스가 있다면 main()이 있는 클래스의 이름이 파일명이 되어야 함
 - public 클래스는 하나의 파일에 하나 이하로만 존재해야 하며 이때 소스 파일명은 public 클래스 이름
- ✓ 컴파일: 바이트 코드<class file>은 클래스 별로 생성

OOP

❖ Instance

- ✓ 참조형 변수: heap이나 static 영역의 참조를 저장하기 위한 변수
- ✓ 인스턴스는 heap 영역의 메모리를 할당 받아서 생성
- ✓ 인스턴스는 생성될 때 클래스에 선언된 static 아닌 멤버 속성(인스턴스 변수) 과 클래스 코드의 참조만 할당 받아서 생성
- ✓ 인스턴스를 생성할 때는 Constructor를 호출하지만 목적이나 코드의 효율성 문제 때문에 Constructor가 아닌 Method를 이용해서 인스턴스를 생성하는 경우도 있음

● 형식1>

클래스명 인스턴스참조_변수명;

인스턴스참조_변수명 = new 클래스_Constructor명();

예> Car carEx;

carEx = new Car();

● 형식2>

클래스명 인스턴스참조_변수명 = new 클래스_Constructor명();

예> Car carEx = new Car();

OOP

❖ 멤버 접근

- ✓ 인스턴스나 클래스를 이용해서 멤버에 접근 할 때는 점[.]을 이용하여 접근
 - `carEx.speed = 10; //멤버 필드의 접근`
 - `carEx.speedUp(); //멤버 Method의 접근`
- ✓ 클래스 정의 안에서 자신의 멤버에 접근 할 때는 멤버의 이름만으로 접근이 가능
- ✓ 인스턴스나 클래스를 이용해서 호출하지 않는 Method
 - `main()` Method
 - ◆ 자바 프로그램을 실행할 때 클래스 이름을 기재하면 자바 가상머신에서 호출하는 Method(entry point)
 - ◆ 형식

```
public static void main(String args[])
{ 내용; }
```
 - `main()` Method의 호출
 - ◆ `c:\W>java main Method를 소유한 클래스이름`

OOP

❖ 변수

✓ 지역 변수(Local Variable)

- Method 안에서 선언된 변수 또는 Method의 매개변수로 선언된 변수로 접근 지정자를 지정하지 않음
- 자료형 앞에는 final 만 추가할 수 있으며 final을 추가하면 변수를 상수화시켜 변경할 수 없도록 함
- final 변수는 관례상 모두 대문자로 이름을 설정
- 지역 변수는 자동으로 초기화되지 않으므로 선언만 한 상태에서는 사용할 수 없음

✓ 클래스 변수(Class Variable)

- 클래스 안에서 static 키워드를 사용해서 생성
- 인스턴스 사이의 데이터 공유를 위해서 생성
- 인스턴스 생성없이 클래스이름 만으로 호출이 가능하며 자바에서는 인스턴스를 이용해서도 호출 가능
- static 영역에 생성되며 초기화 하지 않아도 0 또는 false, null로 초기화

✓ 인스턴스 변수(Instance Variable – Member Variable)

- 클래스 안에 선언되어 각각의 인스턴스가 별도로 생성해서 사용하는 변수
- 초기화 하지 않아도 0 또는 false, null로 초기화

OOP

❖ 속성

✓ 멤버 변수(인스턴스 변수, field, attribute, property) 선언

● 형식

[accessLevel<public,private,protected, default>] + [transient] + [volatile] + [static] + [final] + type + name

● transient는 직렬화(Serialize)할 때 제외

● volatile은 스레드 사용 시 변수의 값을 복사해서 사용하지 않도록 함

● 필드를 선언할 때 초기 값을 할당할 수 있지만 필드는 Constructor나 인스턴스 이니셜라이저를 이용해서 초기화 하는 것을 권장

● 멤버 필드나 멤버 Method의 접근 지정자: 접근을 제한하는 것

◆ private: 클래스 내부의 인스턴스 Method에서만 사용이 가능

◆ default(package): 접근 지정자를 생략하는 것으로 동일한 패키지 내에서는 public 다른 패키지에서는 private

◆ protected: 자신의 클래스 내부와 상속받은 클래스 내부의 인스턴스 Method에서 사용이 가능한데 동일한 패키지 내에서는 public으로 동작

◆ public: 자신의 클래스 내부 및 상속받은 클래스 내부의 인스턴스 Method에서 사용이 가능하고 인스턴스 참조 변수를 이용해서 직접 접근이 가능

OOP

❖ 속성

✓ 실습

- 데이터 클래스 – Student1.java

```
public class Student1 {  
    public String name;  
    public int kor;  
    public int eng;  
    public int mat;  
}
```

OOP

❖ 속성

✓ 실습

- 실행 클래스 - InstanceVariable.java

```
public class InstanceVariable{
    public static void main(String[] args) {
        Student1 obj1 = new Student1();
        // 인스턴스 변수는 초기화 하지 않으면 자동으로 0의 값을 가지며
        // 반드시 인스턴스 이름으로 호출해야 합니다.
        obj1.name = "아이린";
        obj1.eng = 40;
        obj1.mat = 50;
        System.out.println("name:" + obj1.name);
        System.out.println("kor:" + obj1.kor);
        System.out.println("mat:" + obj1.mat);
        System.out.println("eng:" + obj1.eng);

        Student1 obj2 = new Student1();
        // 인스턴스 변수는 각 인스턴스 마다 별개로 소유합니다.
        obj2.name = "배수지";

        System.out.println("obj1.name:" + obj1.name);
        System.out.println("obj2.name:" + obj2.name);
    }
}
```

name:아이린
kor:0
mat:50
eng:40
obj1.name:아이린
obj2.name:배수지

OOP

❖ 속성

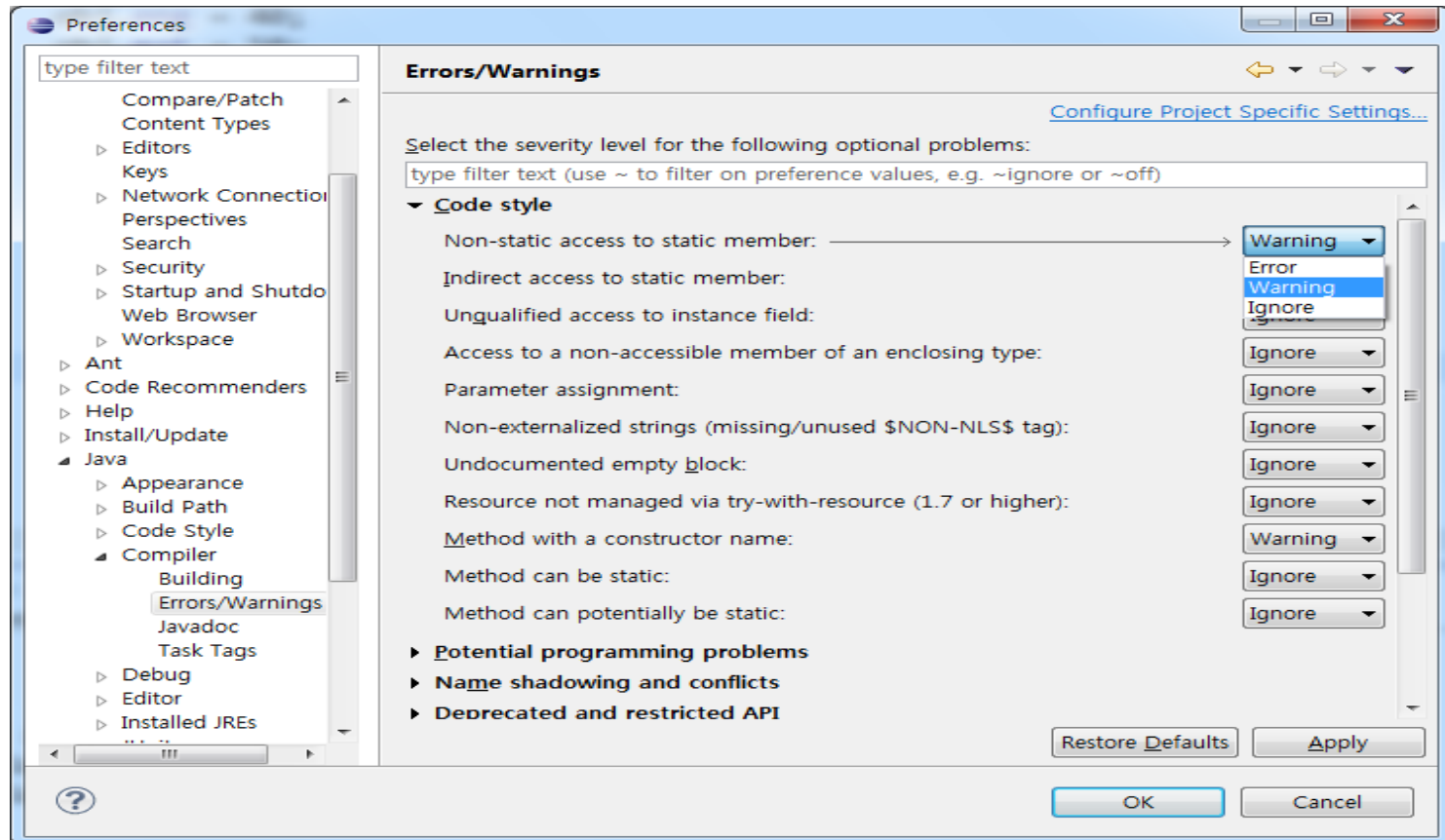
✓ static 필드

- 클래스(static)영역에 생성되는 변수
- 클래스 내부에서 static이라는 예약어와 함께 선언된 변수는 변수가 선언된 클래스를 이용해서 생성된 인스턴스로 접근할 수 있고 클래스로 접근할 수 있음 - 프로그래밍 언어마다 다름
- 하나의 클래스로 여러 개의 인스턴스를 생성하더라도 클래스 영역에 하나만 생성
- 인스턴스들이 공유하는 데이터를 만들기 위해서 사용
- 인스턴스를 이용해서 static 변수에 접근하면 eclipse에서는 경고(대다수의 프로그래밍 언어에서는 에러)
- 위의 경고를 없애고 싶으면 [Window] - [Preferences] 메뉴를 이용해서 환경 설정 화면을 열어서 [Java] - [Compiler] - [Errors/Warnings] 부분을 수정
- 인스턴스는 자신의 클래스 참조를 기억하고 있어서 자신의 멤버를 찾을 때 자기 메모리 영역에서 찾아보고 없으면 클래스 참조를 가지고 클래스 영역에서 찾을

OOP

❖ 속성

✓ static 필드



OOP

❖ 속성

✓ 실습

- 데이터 클래스 – Student2.java

```
public class Student2 {  
    public String name;  
    public int kor;  
    public int eng;  
    public int mat;
```

```
//모든 인스턴스가 공유하기 위한 속성  
static String teacher = "김구";
```

```
}
```

Student의 선생님= 김구

obj1의 선생님= 안중근

obj2의 선생님= 안중근

이름 변경 후

Student의 선생님= 남자현

obj1의 선생님= 남자현

obj2의 선생님= 남자현

OOP

❖ 속성

✓ 실습

● 실행 클래스 – StaticVariable.java

```
public class StaticVariable {  
    public static void main(String[] args) {  
        Student2 obj1 = new Student2();  
        Student2 obj2 = new Student2();  
        //static 으로 선언된 변수는 클래스 이름으로 접근 가능  
        System.out.println("Student의 선생님 = " + Student2.teacher);  
        Student2.teacher = "안중근";  
        //static으로 선언된 변수는 인스턴스 참조 변수로도 접근이 가능  
  
        //인스턴스에게도 영향을 미침  
        System.out.println("obj1의 선생님 = " + obj1.teacher);  
        System.out.println("obj2의 선생님 = " + obj2.teacher);  
        //인스턴스가 변경해도 클래스나 다른 인스턴스가 공유하므로 전부  
        변경된 결과가 출력됩니다.  
        obj1.teacher="남자현";  
        System.out.println("이름 변경 후");  
        System.out.println("Student의 선생님 = " + Student2.teacher);  
        System.out.println("obj1의 선생님 = " + obj1.teacher);  
        System.out.println("obj2의 선생님 = " + obj2.teacher);  
    }  
}
```

OOP

❖ static initializer(초기화)

- ✓ 클래스를 선언하는 구조 안에서 `static { }`을 이용해서 초기화
- ✓ 클래스가 메모리에 load될 때 한번만 수행(처음 클래스 이름이 사용 될 때 - 인스턴스 생성이 이루어지기 이전)
- ✓ static 필드와 지역 변수만 사용 가능
- ✓ 여러 번 사용이 가능한데 작성한 순서대로 수행
- ✓ 일반적으로 static 필드의 초기화에 이용하지만 다른 초기화 문장을 작성하는 것도 가능

OOP

❖ static initializer(초기화)

✓ 실습 - Student3.java

```
public class Student3 {  
    private static String school;  
    private static String teacher;  
    static{  
        school = "삼학국민학교";  
        System.out.println("클래스가 메모리에 로드되었습니다.");  
        System.out.println("학교 이름은 " + school + "입니다");  
        System.out.println("저의 첫번째 선생님 성함은 " + teacher + "선  
        생님 입니다");  
    }  
  
    static{  
        teacher = "오양록";  
        System.out.println("저의 첫번째 선생님 성함은 " + teacher + "선  
        생님 입니다");  
    }  
}
```

OOP

❖ static initializer(초기화)

✓ 실행 - StaticInit.java

```
public class StaticInit {  
    public static void main(String [] args){  
        Student3 obj1;  
        System.out.println("static 초기화는 클래스의 Constructor를 처음  
호출하면 수행됩니다.");  
        obj1= new Student3();  
        Student3 obj2 = new Student3();  
    }  
}
```

static 초기화는 클래스의 Constructor를 처음 호출하면 수행됩니다.
클래스가 메모리에 로드되었습니다.
학교 이름은 삼학국민학교입니다
저의 첫번째 선생님 성함은 null선생님 입니다
저의 첫번째 선생님 성함은 오양록선생님 입니다

OOP

❖ final 필드

- ✓ final 필드는 필드의 값을 변경하지 못하도록 할 때 사용: readonly
- ✓ 변수를 상수화 시킬 때 이용
- ✓ 이름은 모두 대문자로 하는 것을 권장
- ✓ 지역변수가 아니면 static 과 함께 사용하는 경우가 대부분이며 선언하자마자 초기화
- ✓ 선언만하고 초기화를 하지 않으면 이후에는 값을 대입하지 못하므로 선언과 동시에 초기화하거나 Constructor에서 초기화해야 함
- ✓ 선언할 때 초기화하지 않고 Constructor에서도 초기화하지 않으면 이후에는 값을 변경할 수 없으므로 final 변수는 기본값으로만 사용할 수 있음

OOP

❖ Method

- ✓ Method: 전달받은 인수를 처리하여 결과를 돌려주는 작은 프로그램
- ✓ 함수(function)라고도 하는데 함수는 전역 공간에 생성되고 Method는 클래스 안에 만들어진 것(소속이 있음)으로 구분
- ✓ Method의 선언
 - 접근지정자 + [특성] + 결과형 + Method명(매개 변수 나열){ Method 내용}
 - 접근 지정자: public, private, protected, default(생략, package)
 - 특성
 - ◆ abstract – 추상 Method 선언에 사용 – 반드시 오버라이딩 해서 사용
 - ◆ final – 종단 Method 선언에 사용 – 오버라이딩 할 수 없음
 - ◆ static – 클래스 Method 선언
 - ◆ synchronized – 스레드 임계 영역 처리에 사용
 - ◆ native: c 언어와 같은 native 언어의 Method를 사용하고자 하는 경우 선언
 - 결과형
 - ◆ Method를 수행 하고 결과로 반환되는 값의 자료형
 - ◆ void – 반환 값 없음을 의미
 - return: Method 수행 후 결과를 반환할 때 사용하는 예약어로 이 키워드를 만나면 Method 영역에서 벗어나서 호출하는 곳으로 제어권을 이동
 - 매개변수는 없으면 생략 가능

OOP

❖ Method

✓ Method 선언 - 생성, 구현

접근지정자 + 특성 + 결과형 + Method명(매개 변수 나열){

Method의 내용;

//결과형이 void 이면 return 생략 가능

//결과형이 void 인 경우 return을 사용하려면 값을 리턴하면 안됨

return 값;

}

✓ Method의 호출 - 실행

- 리시버.Method(매개변수);
- 리시버는 클래스 이름이나 인스턴스 이름
- 자신의 클래스 내부에서는 리시버 없이 호출 가능

✓ Method의 원형

- Method의 원형이라고 할 때는 **Method이름(매개변수의 자료형, ...)** 만이며 리턴 타입이나 매개변수의 이름은 포함되지 않음

OOP

❖ Method

- ✓ Method 사용 이유
 - 반복적으로 사용하는 코드를 하나의 이름으로 묶어 둔 후 이름 호출만으로 코드를 수행하기 위해서 - 재 사용성
 - 하나의 Method는 한정된 메모리만 사용이 가능하기 때문에 복잡한 프로그램을 구조화해야 할 필요가 있기 때문에
- ✓ 매개변수(argument, parameter): Method를 수행할 때 넘겨주는 데이터로 매개변수가 없으면 항상 동일한 작업을 수행하지만 매개변수가 있으면 매개변수에 따라 유동적인 작업을 수행할 수 있으며 Method를 외부에서 바라볼 때는 argument라는 표현을 사용하고 내부에서는 parameter라고 표현하는 경우가 많음
- ✓ 리턴 값: 리턴 값이 없으면 void로 리턴 타입을 만드는데 리턴 타입이 void인 Method는 작업을 수행하고 남겨주는 데이터가 없으므로 작업의 흐름이 종료되는 것이고 리턴 값이 있으면 그 데이터를 이용해서 다른 작업을 호출할 수 도 있고 작업에 이용할 수 있음

OOP

❖ Method

✓ 실습 - ParamClass.java

```
public class ParamClass {  
    // 매개변수가 없는 Method  
    public void noParam() {  
        for (int i = 0; i < 3; i = i + 1) {  
            System.out.println("한소희");  
        }  
    }  
    // 매개변수가 1개인 Method  
    public void oneParam(int cnt) {  
        for (int i = 0; i < cnt; i = i + 1) {  
            System.out.println("한소희");  
        }  
    }  
    // 매개변수가 2개인 Method  
    public void twoParam(int cnt, String name) {  
        for (int i = 0; i < cnt; i = i + 1) {  
            System.out.println(name);  
        }  
    }  
}
```

OOP

❖ Method

✓ 실행 – ParamMain.java

```
public class ParamMain {
```

```
    public static void main(String[] args) {  
        ParamClass obj = new ParamClass();  
        obj.noParam();  
        System.out.println("=====");  
        obj.noParam();  
  
        System.out.println("=====");  
        obj.oneParam(3);  
        System.out.println("=====");  
        obj.oneParam(2);  
  
        System.out.println("=====");  
        obj.twoParam(2, "신예은");  
        System.out.println("=====");  
        obj.twoParam(3, "한소희");  
    }
```

```
}
```

한소희

한소희

한소희

=====
한소희

한소희

한소희

한소희

=====
한소희

한소희

한소희

한소희

=====
한소희

한소희

한소희

=====
신예은

신예은

=====
한소희

한소희

한소희

한소희

한소희

한소희

OOP

❖ Method

✓ static Method

- 클래스 이름으로 호출가능 한 Method
- static 필드와 지역변수만 사용이 가능
- 인스턴스 변수를 참조하게 되면 에러가 발생
- 인스턴스를 생성하는 Method 또는 인스턴스 변수를 참조하지 않는 Method를 만들 때 사용

OOP

❖ Method

- ✓ static Method 실습 – StaticTestMain.java

```
class StaticTest{
    static private String message = "STATIC";

    public static String getString()
    {
        return message;
    }
}

public class StaticTestMain{
    public static void main(String[] args)
    {
        //static Method는 인스턴스 생성없이 호출하는 것이 가능
        System.out.println("message : "+StaticTest.getString());
    }
}
```

message : STATIC

OOP

❖ Method

✓ 실습 – Student4.java

```
public class Student4 {  
    public String name;  
    public int kor;  
    public int eng;  
    public int mat;  
}
```

신예은의 평균은 75.0입니다.



OOP

❖ Method

✓ 실습 - Student4.java

```
//매개변수를 4개 받아서 set 하는 Method - void set(String, int, int, int)
public void set(String n, int n1, int n2, int n3) {
    name = n;
    kor = n1;
    eng = n2;
    mat = n3;
}
//내부에서만 사용할 Method로 kor, mat, eng 의 평균을 구한 후 실수로 리턴
하는 Method
private double calc() {
    double avg;
    avg = (kor + mat + eng) / 3.0;
    return avg;
}
//평균을 호출해서 결과를 콘솔에 출력하는 Method
public void disp() {
    System.out.println(name + "의 평균은 " + calc() + "입니다.");
}
}
```

OOP

❖ Method

- ✓ 실행 클래스 – InstanceMethod.java

```
public class InstanceMethod {  
    public static void main(String[] args) {  
        Student4 obj = new Student4();  
        obj.set(80, 60, 85);  
        obj.disp();  
    }  
}
```

OOP

❖ Method

- ✓ Method 오버로딩(Overloading – 중복 정의)
 - 하나의 클래스에 동일한 이름의 Method를 2개 이상 정의하는 것
 - Method들의 매개변수 개수 또는 타입은 달라야 함
 - 유사한 역할을 하는 Method의 경우 Method 이름의 일관성을 위해서 존재
- ✓ Method 인수 전달 방식
 - 값에 의한 호출 (Call by Value)
 - ◆ 매개변수의 데이터 형이 기본형(데이터 1개)인 경우 (int, char, ...)
 - ◆ Method의 실행이 전달되는 변수에 영향을 주지 않음
 - 레퍼런스에 의한 호출(Call by Reference)
 - ◆ 매개변수의 데이터 형이 기본형이 아닌 자료형(데이터가 0개 이상)인 경우 (String, ...)
 - ◆ Method의 실행이 전달되는 변수에 영향을 줄 수 있습니다.

OOP

❖ Method

✓ 실습 – Student5.java

```
public class Student5 {  
    public String name;  
    private int kor;  
    private int eng;  
    private int mat;
```

```
//매개변수를 4개 받아서 set 하는 Method – void set(String, int, int, int)  
public void set(String n, int n1, int n2, int n3) {  
    name = n;  
    kor = n1;  
    eng = n2;  
    mat = n3;  
}
```

```
//Method 오버로딩  
public void set(int n1, int n2, int n3) {  
    kor = n1;  
    eng = n2;  
    mat = n3;  
}
```

OOP

❖ Method

✓ 실습 - Student5.java

//내부에서만 사용할 Method로 kor, mat, eng 의 평균을 구한 후 실수로 리턴하는 Method

```
private double calc() {  
    double avg;  
    avg = (kor + mat + eng) / 3.0;  
    return avg;  
}
```

//평균을 호출해서 결과를 콘솔에 출력하는 Method

```
public void disp() {  
    System.out.println(name + "의 평균은 " + calc() + "입니다.");  
}
```

```
}
```

OOP

❖ Method

✓ 실습 – MethodOverloading.java

```
public class MethodOverloading {  
    public static void main(String[] args) {  
        Student5 obj1 = new Student5();  
        Student5 obj2 = new Student5();  
        obj1.set("조이", 80, 60, 85);  
        obj2.name = "제니";  
        obj2.set(70, 90, 87);  
        obj1.disp();  
        obj2.disp();  
    }  
}
```

조이의 평균은 75.0입니다.

제니의 평균은 82.33333333333333입니다.

OOP

❖ Method

✓ 실습 – Calculate.java

```
class Calculate{  
    int result = 0;  
  
    //매개변수가 정수 – call by value  
    void inc1(int n) {  
        n = n + 1;  
    }  
  
    //매개변수가 Calculate 클래스 – call by reference  
    void inc2(Calculate test){  
        test.result = test.result + 1;  
    }  
}
```

OOP

❖ Method

✓ 실습 – ParamPassing.java

```
public class ParameterPassing {  
    public static void main(String[] args) {  
        int a = 10;  
        Calculate cal = new Calculate();  
        cal.result=10;  
        //파라미터로 기본형 데이터를 전송 했으므로 변경되지 않습니다.  
        cal.inc1(a);  
        System.out.println("a= " + a);  
        //참조형 데이터를 전송 했으므로 데이터가 변경 되었을 수 있습니다.  
  
        cal.inc2(cal);  
        System.out.println("cal.result= " + cal.result);  
    }  
}
```

a= 10
cal.result= 11

OOP

❖ Method

✓ return

- Method의 수행을 종료하고 호출하는 곳으로 데이터와 함께 제어권을 넘기는 명령어
- 넘겨주는 데이터가 없을 때는 return type을 void라고 작성 - 없으면 에러
- return을 할 때 데이터가 없으면 이 Method는 호출하는 것으로 작업의 흐름이 끝나는 것이고 데이터를 return하면 그 데이터를 가지고 다른 작업을 계속 수행할 수 있음



OOP

❖ Method

✓ 실습 – ReturnClassMain.java

```
class ReturnClass {  
    public void noReturnAdd(int first, int second) {  
        System.out.println("합계:" + (first + second));  
    }  
  
    public int returnAdd(int first, int second) {  
        return (first + second);  
    }  
}  
  
public class ReturnClassMain{  
    public static void main(String [] args) {  
        ReturnClass obj = new ReturnClass();  
        obj.noReturnAdd(100, 200);  
  
        int result = obj.returnAdd(100, 200);  
        result = obj.returnAdd(result, 300);  
        System.out.println("결과:" + result);  
    }  
}
```

합계:300
결과:600

OOP

❖ Method

✓ this

- 클래스 내부의 인스턴스 Method에서 인스턴스를 가리키는 포인터로 숨겨져 있음
- 인스턴스 생성 시 자동으로 만들어지는 변수이고 모든 인스턴스 Method의 숨겨진 매개 변수
- 인스턴스 Method에서 인스턴스 필드를 사용하면 코드 상에서 보이지지는 않지만 this가 삽입되서 코드화

OOP

❖ Method

✓ this

- 클래스에 Method 선언

```
public double calc() {  
    double avg;  
    avg = (kor + mat + eng) / 3.0;  
    return avg;  
}
```

- 실제 내부에 선언이 될 때는 아래처럼 선언

```
public double calc(Student this) {  
    double avg;  
    avg = (this.kor + this.mat + this.eng) / 3.0;  
    return avg;  
}
```

- 호출

인스턴스.calc() -> 클래스참조.calc(Obj1)의 형태로 호출됨

OOP

❖ Method

✓ this

- Method 내에서 만든 Local Variable 과 외부에서 만든 static 이나 instance variable의 이름이 같은 경우 이름만 사용하면 Local Variable이 호출되는데 명시적으로 static 이나 instance variable을 호출하고자 하는 경우에 this.변수명으로 호출할 수 있고 this.Method명()으로 Method를 호출
- this.을 붙이게 되면 Method 내부에서는 찾지 않음 – Kotlin 이나 Swift 처럼 람다를 많이 사용하는 언어에서 중요

OOP

❖ Method

✓ this 실습 - ThisStudent

```
public class ThisStudent {  
    public String name;  
  
    public void disp(){  
        String name = "배주현";  
        System.out.println("지역변수-집에서 사용하는 이름:" + name);  
        System.out.println("멤버필드-외부에서 사용하는 이름:" +  
this.name);  
    }  
}
```

OOP

❖ Method

✓ this 실습 - ThisMain

```
public class ThisMain {  
    public static void main(String[] args) {  
        ThisStudent obj = new ThisStudent();  
        obj.name = "아이린";  
        obj.disp();  
    }  
}
```

지역 변수-집에서 사용하는 이름:배주현
멤버 필드-외부에서 사용하는 이름:아이린

OOP

❖ Method

✓ 접근자 Method

- 객체 지향 언어에서는 멤버 필드에 직접 접근하는 것을 권장하지 않음
- 멤버 필드는 `private`으로 지정해서 인스턴스를 이용해서 직접 접근이 안되도록 하고 멤버 필드의 값을 리턴해주는 `getter Method`와 멤버 필드의 값을 설정해주는 `setter Method`를 `public`으로 만들어서 멤버 필드를 사용하는 것을 권장
- `getter`
 - ◆ 멤버 필드의 값을 리턴해주는 Method
 - ◆ 리턴 타입은 멤버 필드의 타입과 동일하게 하고 이름은 `get필드명` 으로 하고 필드명의 첫 글자는 대문자로 하며 매개변수는 없고 Method의 내용은 멤버 필드의 값을 리턴하기만 함
 - ◆ 필드의 타입이 `boolean` 인 경우 `get` 대신에 `is`를 사용하기도 함
- `setter`
 - ◆ 멤버 필드의 값을 설정해주는 Method
 - ◆ 리턴 타입은 `void`로 하고 이름은 `set필드명` 으로 하고 필드명의 첫 글자는 대문자로 하고 매개변수는 필드의 타입과 동일한 타입 1개로 하며 내용은 매개변수의 값을 멤버 필드에 대입

OOP

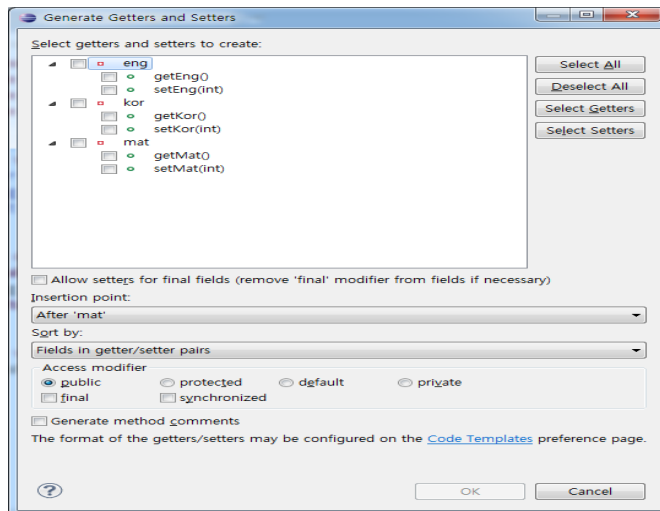
❖ Method

✓ 접근자 Method

- 직접 생성해도 되지만 이클립스에서 제공해주는 기능을 이용해도 되며 자바에서는 lombok이라는 라이브러리를 이용해서 어노테이션으로 생성하기도 함

[Source] - [Generate Getters and Setters...]

- 접근자 Method를 수정하는 경우가 있는데 getter를 수정하는 경우는 지연 생성 등을 위해서 코드 내용을 수정하거나 멤버 필드의 자료형이 boolean인 경우 자료형을 명시적으로 나타내기 위해서 이름을 is변수명 으로 변경하는 경우가 있음
- setter를 수정하는 경우는 멤버 필드의 값이 조건을 만족하면 다른 작업을 수행하도록 하는 경우로 이것을 key-value observing 이라고 함



OOP

❖ Method

- ✓ 접근자 Method – Student6.java

```
public class Student6 {  
    private String name;  
    private int kor;  
    private int eng;  
    private int mat;
```

OOP

❖ Method

✓ 접근자 Method – Student6.java

```
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}  
public int getKor() {  
    return kor;  
}  
public void setKor(int kor) {  
    this.kor = kor;  
}  
public int getEng() {  
    return eng;  
}  
public void setEng(int eng) {  
    this.eng = eng;  
}
```

OOP

❖ Method

- ✓ 접근자 Method – Student6.java

```
public int getMat() {  
    return mat;  
}  
public void setMat(int mat) {  
    this.mat = mat;  
}
```

OOP

❖ Method

✓ 접근자 Method – Student6.java

```
//매개변수를 4개 받아서 set 하는 Method – void set(String, int, int, int)
public void set(String n, int n1, int n2, int n3) {
    name = n;
    kor = n1;
    eng = n2;
    mat = n3;
}
//내부에서만 사용할 Method로 kor, mat, eng 의 평균을 구한 후 실수로 리턴
하는 Method
private double calc() {
    double avg;
    avg = (kor + mat + eng) / 3.0;
    return avg;
}
//평균을 호출해서 결과를 콘솔에 출력하는 Method
public void disp() {
    System.out.println(name + "의 평균은 " + calc() + "입니다.");
}
}
```

OOP

❖ Method

- ✓ 접근자 Method – AccessorMethod.java

```
public class AccessorMethod {  
    public static void main(String[] args) {  
        Student6 obj = new Student6();  
        obj.setName("지수");  
        obj.setKor(90);  
        obj.setEng(87);  
        obj.setMat(54);  
        obj.disp();  
    }  
}
```

지수의 평균은 77.0입니다.

OOP

❖ Method

✓ 재귀 호출 – recursion, recursive call

- Method가 자기 자신을 다시 호출하는 것
- 자바에서는 속도 문제 때문에 권장하지는 않으나 특정한 Method의 경우 재귀 호출을 이용하게 되면 코드를 단순화 시킬 수 있어서 사용
- 피보나치 수열이나 하노이의 탑이 대표적인 경우
- 피보나치 수열
 - ◆ $F(n) = F(n-1) + F(n-2)$
 - ◆ 단 $n \geq 3$ 이고 $n=1$ 또는 $n=2$ 이면 1

OOP

❖ Method

- ✓ 재귀 호출 – recursion, recursive call

```
import java.util.*;
public class FiboMain {
    public static int fibo(int n) {
        if (n == 1 || n == 2)
            return 1;
        else
            return fibo(n - 1) + fibo(n - 2);
    }
}
```

알고 싶은 피보나치 수열의 값은 몇번째? 11
입력한 11번째 피보나치의 값은 89입니다

OOP

❖ Method

- ✓ 재귀 호출 – recursion, recursive call

```
public static void main(String[] args) {  
    int n = -1;  
    Scanner in = new Scanner(System.in);  
    while (true) {  
        System.out.print("알고 싶은 피보나치 수열의 값은 몇  
        번째?");  
  
        n = in.nextInt();  
        if (n < 1)  
            System.out.println("1 이상의 정수를 입력하  
            세요");  
        else  
            break;  
    }  
    in.close();  
    System.out.println("입력한 " + n + "번째 피보나치의 값은 " +  
    fibo(n) + "입니다");  
}
```

OOP

❖ Method

- ✓ varargs(Variable Argument) – 5.0 이 후 버전에서 추가된 인자의 개수를 지정하지 않아도 되는 Method
 - 매개변수에 **자료형 ... 배열명**으로 기재하면 Method를 호출할 때 매개변수의 개수에 상관없이 대입이 가능
 - 주의할 점은 varargs는 반드시 매개변수의 마지막에 사용해야 하는데 매개변수의 개수를 지정하지 않아도 되는 매개변수가 앞쪽에 사용되면 언제 이 매개변수가 끝나는지 알 수 없어서 다음 매개변수를 판별해 낼 수 없기 때문
 - 배열을 매개변수로 이용해도 비슷한 기능을 수행

OOP

❖ Method

- ✓ varargs(Variable Argument) – VarArgsTest

```
public class VarArgsTest {  
    public void varArg(int... n) {  
        for (int i = 0; i < n.length; i++)  
            System.out.println("n[" + i + "]: " + n[i]);  
        System.out.println("-----");  
    }  
    public void varArray(int ar[]) {  
        for (int i = 0; i < ar.length; i++)  
            System.out.println("ar[" + i + "]: " + ar[i]);  
        System.out.println("-----");  
    }  
    public static void main(String[] args) {  
        VarArgsTest vt = new VarArgsTest();  
        vt.varArg(100, 200);  
        vt.varArg(150, 200, 250, 300);  
        int ar[] = { 100, 200 };  
        vt.varArray(ar);  
        int ar1[] = { 150, 200, 250, 300 };  
        vt.varArray(ar1);  
    }  
}
```

OOP

❖ Method

- ✓ varargs(Variable Argument) – VarArgsTest

n[0]:100

n[1]:200

n[0]:150

n[1]:200

n[2]:250

n[3]:300

ar[0]:100

ar[1]:200

ar[0]:150

ar[1]:200

ar[2]:250

ar[3]:300

OOP

❖ Constructor(생성자)

- ✓ 인스턴스를 생성하기 위해 호출하는 Method
- ✓ Constructor 호출 방법: **new Constructor(매개변수)**
- ✓ Constructor의 기본 형식

```
-[public/private/protected] + 클래스이름(매개변수)
{
    //일반 Method와 유사
    //초기화 문장
}
```
- ✓ 일반 Method 와 의 차이점
 - 리턴 형이 없음
 - 이름은 클래스 이름과 동일
 - 리시버를 이용해서 호출할 수 없으며 new 연산자 만을 이용해서 호출
- ✓ 용도: 인스턴스의 초기화 작업을 처리하기 위한 목적으로 생성
- ✓ 정의하지 않는 경우: 클래스를 정의하면 매개변수가 없는 Constructor가 자동으로 생성되며 별도로 Constructor를 정의하면 기본적으로 제공되는 Constructor는 소멸

OOP

❖ Constructor(생성자)

- ✓ Constructor도 오버로딩 가능

- ✓ super()

- 상위 클래스의 Constructor 호출 – 가장 먼저 와야 함

```
예> class AnimationThread extends Thread {  
    public AnimationThread() {  
        super("AnimationThread");  
        ...  
    }  
}
```

- ✓ this(): 자신의 다른 Constructor를 호출하는 것으로 Constructor가 오버로딩 된 경우 사용 가능하며 super()를 호출한다면 그 다음에 위치해야 하며 사용하는 이유는 코드의 중복을 피하기 위한 목적

OOP

❖ Constructor(생성자)

✓ Student7.java

```
public class Student7 {  
    private String name;  
    private int kor;  
    private int eng;  
    private int mat;  
    //매개변수가 없는 Constructor를 Default Constructor라고 합니다.  
    public Student7(){  
        System.out.println("디폴트 Constructor");  
    }  
    //Constructor 오버로딩 - 매개변수가 있는 Constructor를 정의  
    public Student7(String n, int n1, int n2, int n3)  
    {  
        //매개변수가 없는 Constructor 호출  
        //다른 문장보다 앞에 기재되어야하며 Constructor에서만 사용  
        //super()가 있는 경우에는 super() 다음에 와야 합니다.  
        this();  
        name = n;  
        kor = n1;  
        eng = n2;  
        mat = n3;  
    }  
}
```

OOP

❖ Constructor(생성자)

✓ Student7.java

//접근자 Method

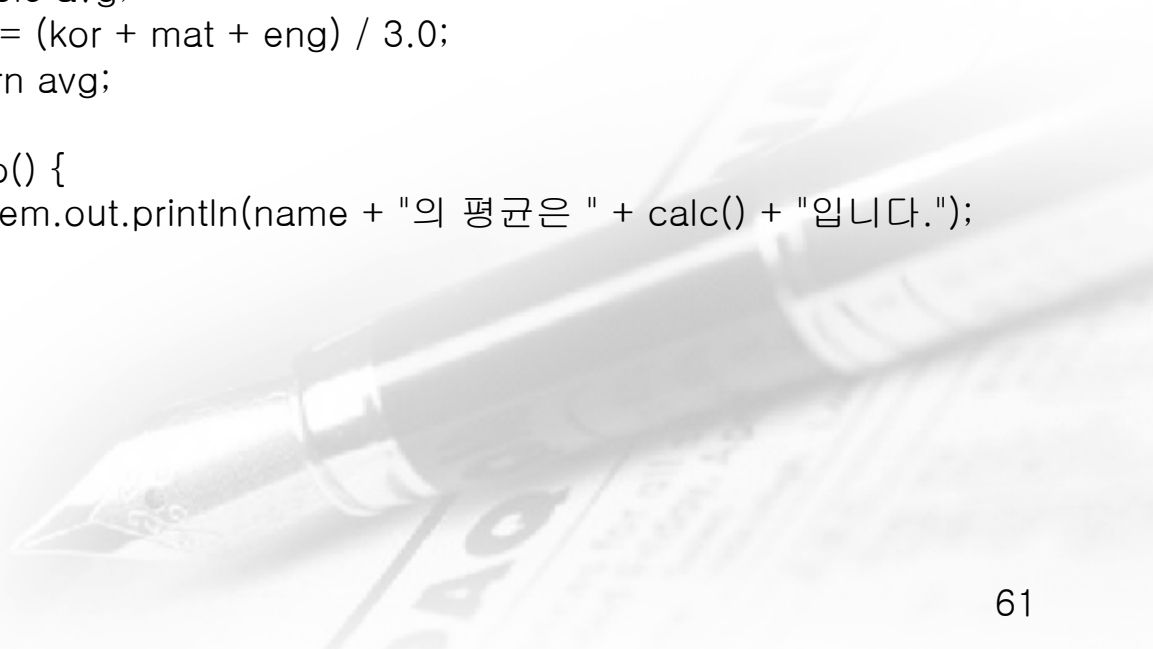
```
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}  
public int getKor() {  
    return kor;  
}  
public void setKor(int kor) {  
    this.kor = kor;  
}  
public int getEng() {  
    return eng;  
}  
public void setEng(int eng) {  
    this.eng = eng;  
}
```

OOP

❖ Constructor(생성자)

✓ Student7.java

```
public int getMat() {  
    return mat;  
}  
public void setMat(int mat) {  
    this.mat = mat;  
}  
  
public double calc() {  
    double avg;  
    avg = (kor + mat + eng) / 3.0;  
    return avg;  
}  
public void disp() {  
    System.out.println(name + "의 평균은 " + calc() + "입니다.");  
}  
}
```



OOP

❖ Constructor(생성자)

✓ ConstructorTest.java

```
public class ConstructorTest {  
    public static void main(String[] args) {  
        Student7 Obj1 = new Student7("", 70, 60, 80);  
        Student7 Obj2 = new Student7();  
        Obj1.disp();  
        Obj2.disp();  
    }  
}
```

디폴트 Constructor
디폴트 Constructor
의 평균은 70.0입니다.
null의 평균은 0.0입니다.

OOP

❖ Constructor(생성자)

- ✓ Static variable 과 Constructor를 적절히 이용하면 일련번호 부여 가능

- Student8.java

```
public class Student8 {  
    private int number;  
    static int sequence = 0;  
    //일련번호 생성을 위한 Constructor  
    public Student8(){  
        number = ++sequence;  
    }  
  
    //접근자 Method  
    public int getNumber() {  
        return number;  
    }  
    public void setNumber(int number) {  
        this.number = number;  
    }  
  
    public void disp() {  
        System.out.println("번호는 " + number + " 입니다.");  
    }  
}
```

OOP

❖ Constructor(생성자)

- ✓ Static variable 과 Constructor를 적절히 이용하면 일련번호 부여 가능

- SequenceMain.java

```
public class SequenceMain {  
  
    public static void main(String[] args) {  
        Student8 Obj1 = new Student8();  
        Student8 Obj2 = new Student8();  
        Obj1.disp();  
        Obj2.disp();  
    }  
}
```

번호는 1 입니다.
번호는 2 입니다.

OOP

❖ Constructor(생성자)

✓ 인스턴스 배열

- 인스턴스 배열은 선언과 동시에 Constructor를 호출하거나 크기를 결정(메모리 할당)을 한 후 Constructor를 각각 호출하는 형태로 생성
- 인스턴스 배열을 선언하고 크기를 결정한다고 해서 인스턴스가 생성되서 대입되는 것은 아니며 처음 배열을 생성하기만 하면 전부 null이 대입되서 배열의 요소를 가지고 자신의 멤버를 호출하면 NullPointerException 예외가 발생
- 배열을 생성할 때 사용한 new는 저장할 수 있는 크기를 설정한 것이고 각각의 인덱스에 접근해서 실제 인스턴스 생성을 해 주어야 함

OOP

❖ Constructor(생성자)

✓ 인스턴스 배열

● ObjectArray.java

```
public class ObjectArray {  
    public static void main(String[] args) {  
        //배열에 저장할 수 있는 크기만 설정  
        Student8 ar[] = new Student8[2];  
  
        // 아래 2개의 문장을 호출하지 않으면 예외 발생  
        //ar[0] = new Student8();  
        //ar[1] = new Student8();  
  
        //배열을 생성할 때 Constructor를 호출해서 생성  
        //Student8 ar[] = { new Student8(), new Student8() };  
  
        ar[0].disp();  
        ar[1].disp();  
    }  
}
```

Exception in thread "main" java.lang.NullPointerException
at chap06.ObjectArray.main(ObjectArray.java:16)

OOP

❖ Constructor(생성자)

✓ Instant Field Initializer

- 인스턴스 생성 시 호출되는 코드를 만드는 방법으로는 Constructor 이외에 **인스턴스 필드 이니셜라이저** 라는 방법이 제공
- 클래스 정의 안에 { }를 사용해서 코드를 작성
- 이렇게 작성한 코드는 내부적으로 <init>이라는 Method로 저장

```
class Student{  
    private int kor;  
    {  
        kor = 100;  
    }  
}
```

● 변수의 초기화 순서

- ◆ static 변수는 처음 선언할 때는 기본값으로 만들어지고 그 후 주어진 값으로 초기화가 이루어지고 static 초기화 블록이 있으면 그 내용을 수행해서 초기화
- ◆ Instance 변수는 처음 선언할 때 기본값으로 만들어지고 그 후 명시적으로 초기화 한 값으로 초기화가 이루어지고 인스턴스 초기화 블록을 수행한 후 Constructor의 내용을 수행

OOP

❖ Constructor(생성자)

- ✓ 복사 Constructor: 동일한 타입의 인스턴스를 매개변수로 받아서 대입 받은 인스턴스가 소유한 멤버의 값을 복사해서 인스턴스를 생성하는 Constructor

```
class Student{  
    private int kor;  
    public Student(Student obj){  
        this.kor = obj.kor;  
    }  
}
```

OOP

❖ 메모리 정리

- ✓ Java는 GarbageCollection이 heap 메모리를 정리
- ✓ GarbageCollection의 호출 시점은 우리가 알 수 없음
- ✓ 강제로 호출해주는 System.gc()가 있지만 바로 호출되지 않을 수 있음
- ✓ 우선순위가 낮아서 다른 작업이 수행 중이면 바로 호출되지 않음
- ✓ Java에서 메모리 정리를 하고 싶은 인스턴스가 있으면 null을 대입해서 참조를 제거하면 참조가 없는 heap의 공간을 정리
- ✓ null을 대입하면 인스턴스의 finalize()가 호출되고 차후에 GarbageCollection에 의해 heap에서 제거

OOP

❖ 메모리 정리

```
public class GC {  
    //메모리 정리가 발생할 때 호출되는 Method  
    public void finalize(){  
        System.out.println("메모리 정리");  
    }  
    public static void main(String[] args) {  
        GC obj = new GC();  
        obj = null;  
        System.gc();  
    }  
}
```

메모리 정리

package

❖ package

- ✓ package: 서로 관련된 클래스와 인터페이스의 클래스 파일들을 하나의 디렉토리처럼 묶어 놓은 것
 - 하나의 응용프로그램은 여러 개의 패키지로 구성
 - 하나의 패키지에 여러 개의 클래스 파일을 저장할 수 있음
 - 패키지는 jar 파일로 압축해서 사용할 수 있음
 - JDK에서 제공하는 표준 패키지는 rt.jar에 압축되어 있음
- ✓ import: 클래스의 사용을 간편하게 하기 위해서 줄여 쓸 수 있도록 하기 위한 개념
- ✓ 패키지의 모든 클래스를 줄여서 사용하고자 하는 경우
 - `import 패키지명.*;`
- ✓ 특정 클래스만 줄여쓰기를 하고자 하는 경우
 - `import 패키지명.클래스명;`
 - 여러 패키지에 동일한 클래스 이름이 있을 경우에는 특정 클래스만 사용한다고 기재하면 우선 순위가 높음
- ✓ static import: 클래스의 static 멤버나 final 변수를 클래스 이름 없이 사용하고자 할 때 사용하는 import
 - `import static [패키지경로.클래스명.*];`
 - `import static [패키지경로.클래스명.final변수명];`

package

❖ package

✓ 다른 패키지 이름 사용하기

- import를 이용하지 않는 경우

- ◆ 소스 내에서 매번 전체 패키지 이름과 클래스 이름을 써주어야 함

- import 키워드 이용하는 경우

- ◆ 소스의 시작 부분에 사용하려는 패키지 명시: 소스에는 클래스 명만 명시하면 됨

- ◆ 특정 클래스의 경로명만 포함: `import java.util.Scanner;`

- ◆ 패키지 내의 모든 클래스를 포함시키는 경우

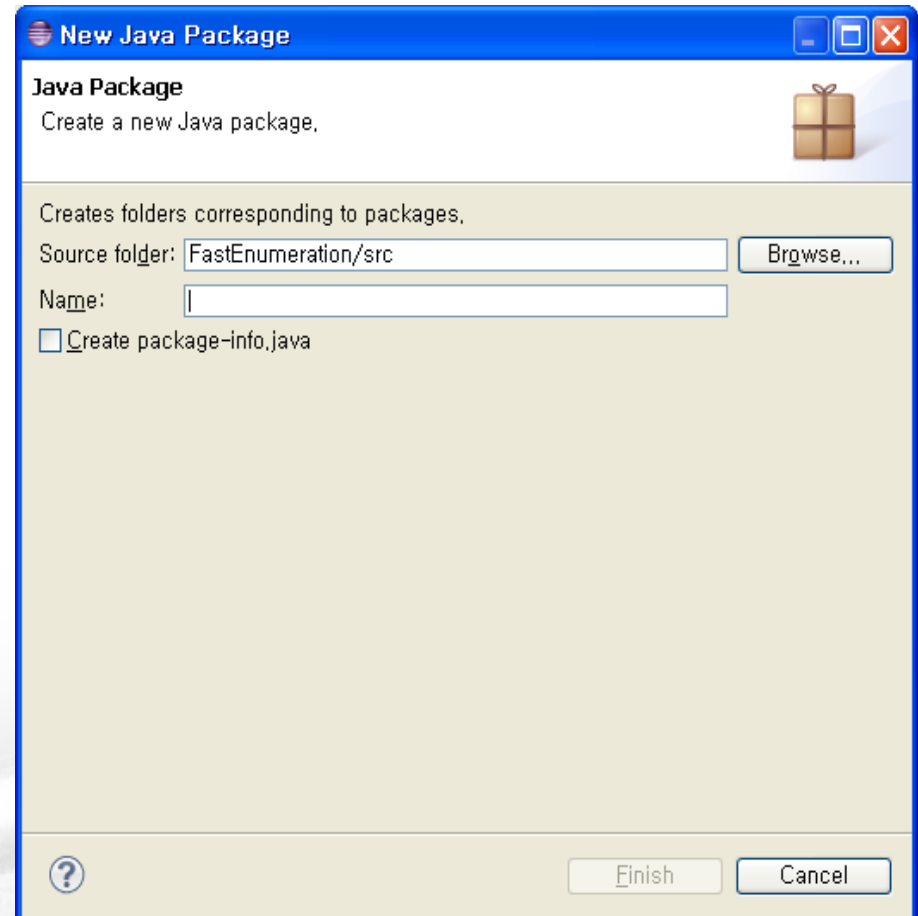
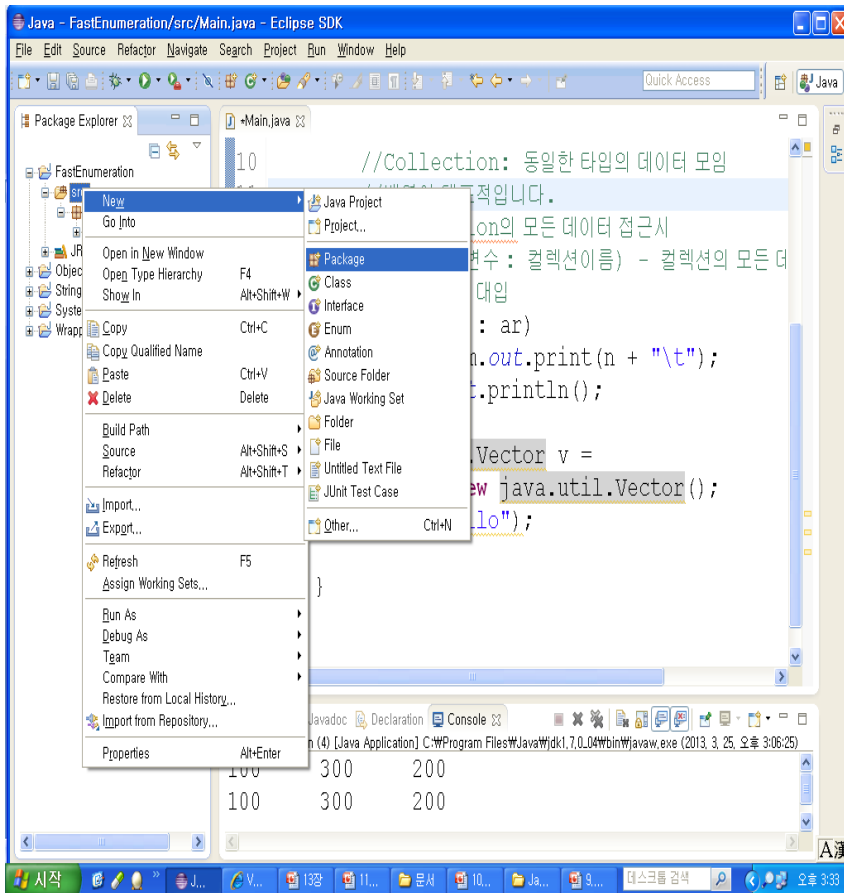
- `import java.util.*;`

- *는 현재 패키지 내의 클래스만을 의미하며 하위 패키지의 클래스까지 포함하지 않음

package

❖ package

✓ 패키지 생성



package

❖ package

✓ Static Import

```
import static java.lang.System.out;
import static java.lang.Thread.MAX_PRIORITY;;
public class StaticImport {
    public static void main(String[] args)
    {
        out.println("Hello");
        out.println(MAX_PRIORITY);
    }
}
```

Hello
10



package

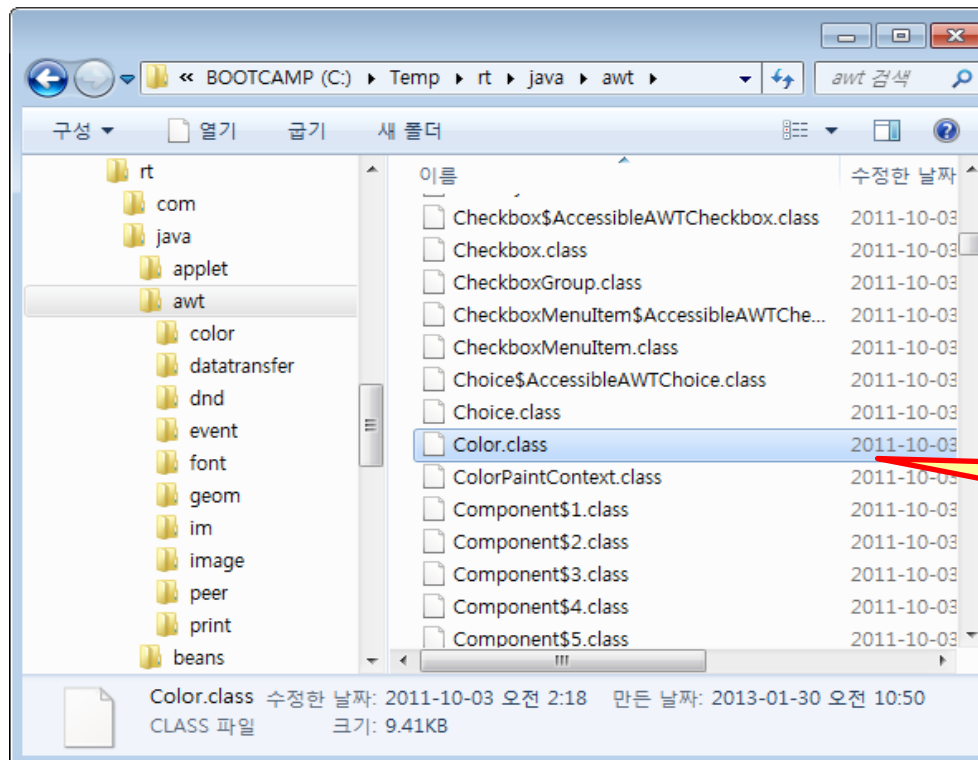
❖ 패키지의 특징

- ✓ 패키지 계층 구조
 - 클래스나 인터페이스가 너무 많아지면 관리의 어려움
 - 관련된 클래스 파일을 하나의 패키지로 계층화하여 관리 용이
- ✓ 패키지 별 접근 제한
 - default로 선언된 클래스나 멤버는 동일 패키지 내의 클래스들이 자유롭게 접근하도록 허용
- ✓ 동일한 이름의 클래스와 인터페이스의 사용 가능
 - 서로 다른 패키지에 이름이 같은 클래스와 인터페이스 존재 가능
- ✓ 패키지1.패키지2 과 패키지1 은 상하위 관계가 아님
 - 디렉토리 구조로 보면 패키지1 디렉토리가 상위 디렉토리이지만 별도의 패키지로 간주
 - 패키지1을 import 했다고 해서 패키지2가 import 되지는 않음

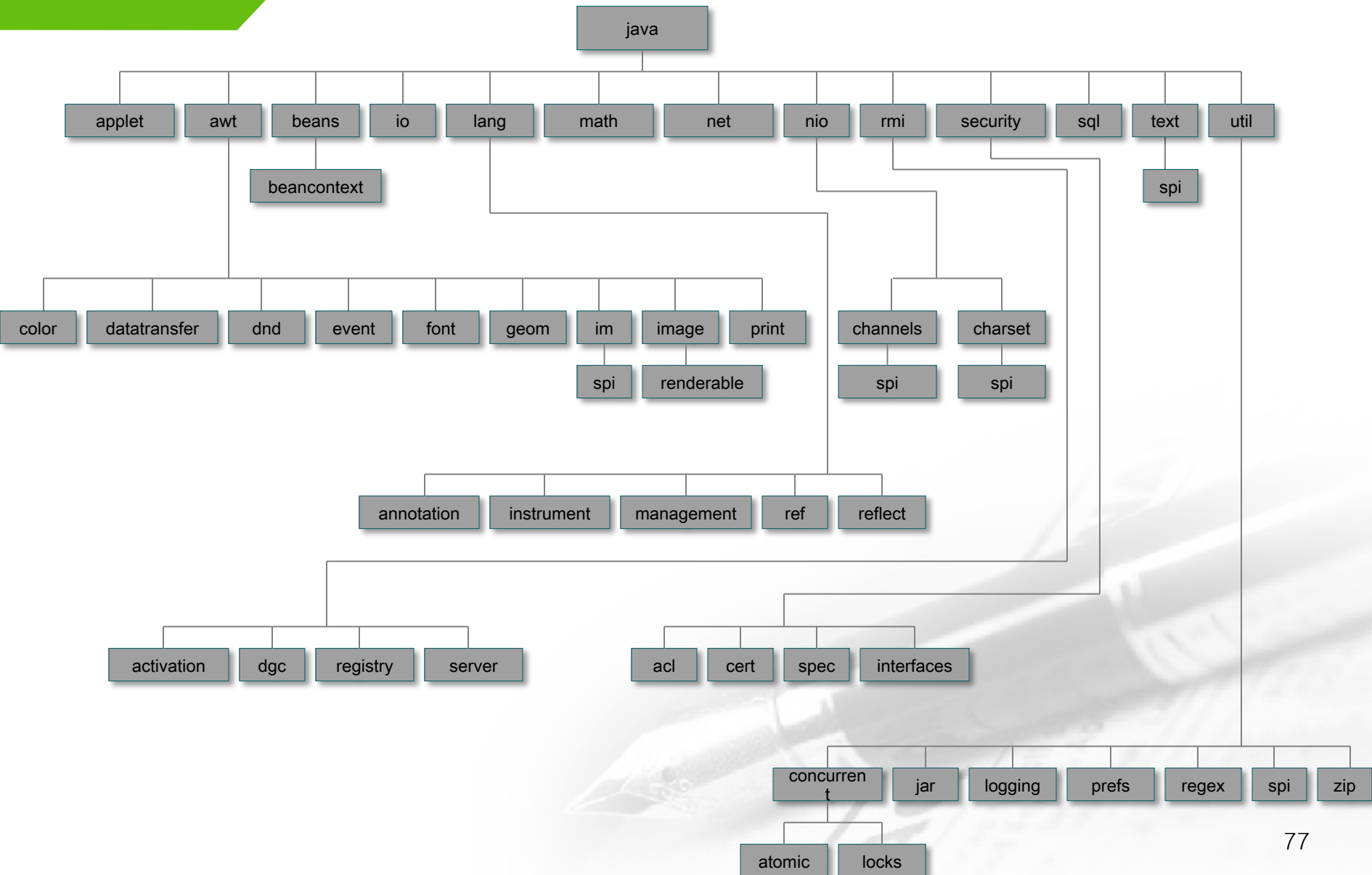
package

❖ JDK 패키지

- ✓ 자바에서는 관련된 클래스들을 표준 패키지로 묶어 사용자에게 제공
- ✓ 자바에서 제공하는 패키지는 C언어의 표준 라이브러리와 유사
- ✓ JDK의 표준 패키지는 rt.jar에 담겨 있음
 - C:\WProgram Files\Java\jdk1.7.0_04\jre\lib\Wrt.jar



package



package

❖ 자바 프로그램이 클래스를 찾는 방법

- ✓ 컴파일 타임 검색 - 컴파일러가 소스 코드에서 Method 호출과 같은 구문을 만나면 이 Method를 호출할 수 있는지 확인하기 위해서 호출하는 리시버의 타입 선언 부분으로 컴파일 위치가 옮겨지고 컴파일러는 자바 플랫폼 패키지(rt.jar)을 검색한 후 확장 타입을 위해 확장 패키지를 검색해서 클래스를 찾는데 만일 jvm에 -sourcepath 커맨드라인 옵션이 설정되어 있으면 컴파일러는 옵션에 지정된 경로에 있는 소스 파일을 검색해서 로드
- ✓ 런타임 검색 - JVM이 실행 중에 특정한 타입을 만나게 되면 클래스 로더라고 하는 특별한 코드를 통해 연관된 클래스 파일을 로드하는데 이 때는 classpath가 설정되어 있으면 classpath에서 검색하고 이 환경변수가 설정되어 있지 않으면 현재 디렉토리에서 검색하며 eclipse는 프로젝트 안에 bin 폴더를 생성해서 class 파일을 위치시키고 이 디렉토리를 classpath로 사용
- ✓ 강제로 클래스를 로드할 수 있는데 Class.forName 이라는 Method로 클래스를 강제로 로드
- ✓ 클래스 파일들을 jar 파일로 압축해서 포함시키는 것이 가능
- ✓ 클래스나 패키지 프로젝트를 선택하고 마우스 오른쪽을 클릭 후 [export] 하면 생성 가능

package & import

- ❖ java로 시작하는 패키지는 Java가 처음 만들어질 때부터 있었던 패키지이고 javax로 시작하는 패키지는 나중에 추가된 패키지
- ❖ java.lang
 - ✓ 스트링, 수학 함수, 입출력 등 자바 프로그래밍에 필요한 기본적인 클래스와 인터페이스
 - ✓ 자동으로 import 됨 - import 문 필요 없음
- ❖ java.util
 - ✓ 자바 유틸리티 패키지: 날짜, 시간, 벡터, 해쉬 테이블 등과 같은 다양한 유틸리티 클래스와 인터페이스 제공
- ❖ java.io
 - ✓ 키보드, 모니터, 프린터, 디스크 등에 입출력을 할 수 있는 클래스와 인터페이스 제공
- ❖ java.awt
 - ✓ 자바 GUI 프로그래밍을 위한 클래스와 인터페이스 제공
- ❖ javax.swing
 - ✓ 자바 GUI 프로그래밍을 위한 스윙 패키지

package & import

- ❖ java.sql, javax.sql
 - ✓ 데이터베이스 관련 패키지
- ❖ java.net
 - ✓ 자바 네트워크 프로그래밍 관련 패키지
- ❖ java.security
 - ✓ 보안을 위한 암호화 알고리즘 및 인증에 관련된 클래스
- ❖ java.text
 - ✓ 출력 포맷과 관련된 클래스

연습문제

- ❖ 은행 계좌 정보를 저장하기 위한 클래스를 생성
 - ✓ 프로퍼티
 - ❑ 계좌번호, 사용자이름, 잔액 정보를 저장
 - ❑ 계좌번호와 사용자이름은 문자열이고 잔액은 실수
 - ❑ 인스턴스가 직접 접근할 수 없도록 생성
 - ✓ Constructor
 - ❑ 계좌번호를 매개변수로 받아서 계좌번호를 초기화 하고 이름은 noname 잔액은 0으로 초기화하는 Constructor를 생성
 - ❑ 계좌번호, 사용자이름, 잔액을 매개변수로 받아서 초기화하는 Constructor를 생성
 - ✓ 접근자 Method 생성
 - ✓ 계좌번호를 매개변수로 받아서 계좌번호와 잔액 정보를 출력하는 Method를 생성
 - ✓ 계좌번호와 금액(실수)을 매개변수로 받아서 계좌번호에 해당하는 데이터의 잔액을 금액과의 연산으로 수정하는 Method를 생성