Java 프로그래밍 길잡이

✓ 원리를 알면 IT가 맛있다

Java Programming for Beginners

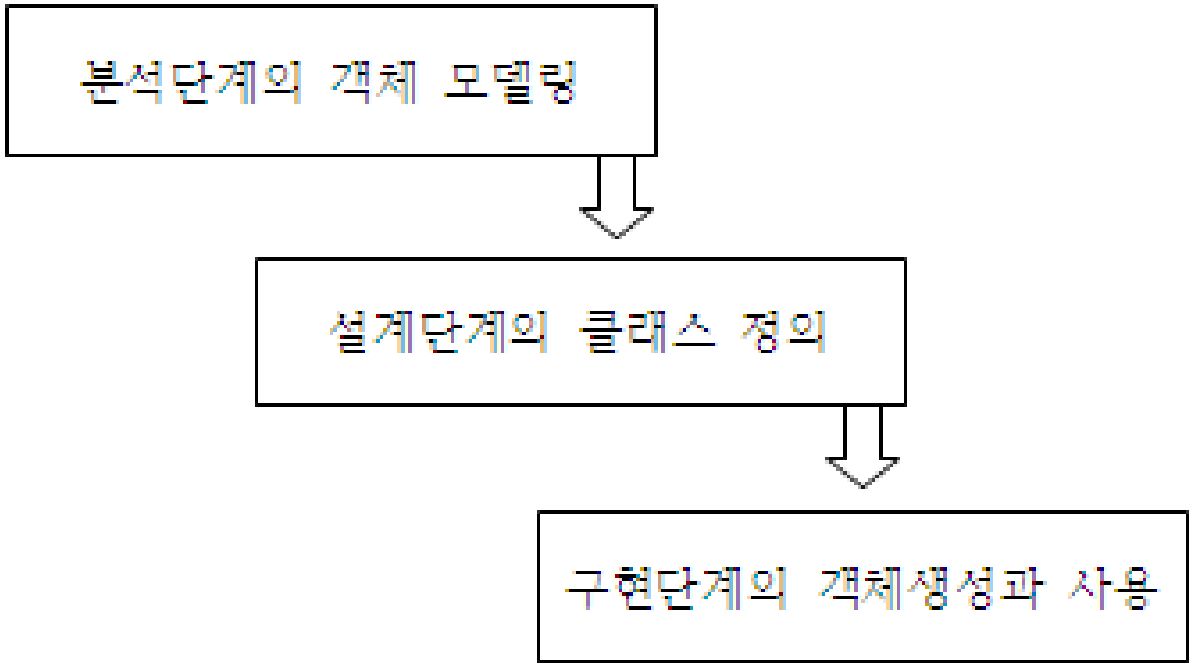
chapter 05.

객체와 클래스

❑ 학습목표

- 자바클래스의 핵심인 클래스와 객체에 관하여 학습한다.
- 클래스의 구성요소인 인스턴스 변수,메소드, 생성자에 대해서 학습한다.
- 오버로딩 메소드, 오버로딩 생성자에 관하여 학습한다.
- this 키워드 사용법에 관하여 학습한다.
- package와 import문에 관하여 학습한다.
- static 키워드와 final 키워드에 관하여 학습한다.
- Varargs에 대해 학습한다.

❑ 1) 자바의 객체 지향 개발 3단계



❍ 객체생성

정의된 클래스를 사용해서 실객체를 메모리에 올리는 작업을 의미한다. 이 작업을 '객체생성','인스턴스화', '메모리 로딩'이라고 부른다. 모든 클래스는 반드시 객체생성을 통해서 메모리에 올려놓고 사용할 수 있다. Heap 메모리에 생성되고 new 키워드를 사용한다.

❑ 2] 클래스

문법:

```
public class  클래스명 {
    // 인스턴스 변수
    // 메소드
    // 생성자
 }
```

* 클래스의 구성요소 3가지

- 인스턴스 변수 (instance variable) : 멤버변수
  => 클래스를 특성을 표현하기 위한 속성값을 저장할 때 사용.
- 메소드 (method) : 멤버 메소드
  => 인스턴스 변수의 값을 관리하는 역할.(저장 및 수정 , 조회)
- 생성자 (constructor)
  => 인스턴스 변수에 값을 처음 저장하는 역할. (초기화 역할)

*지정자 (modifier)

특정 목적을 위해서 사용하는 키워드로서 클래스,변수,메소드에 사용할 수 있고 생략도 가능하다.

가. 일반 지정자 (modifier)

- static
- final
- abstract

나. 접근 지정자 (access modifier)

- public
- protected
- (friendly , default)
- private

- 클래스명

- 객체를 잘 표현할 수 있는 의미 있는 명사형으로 지정한다.
- 반드시 첫 글자는 대문자로 지정한다. 소문자로 지정해도 문법적으로는 문제가 없지만 관습적으로 첫 글자는 대문자로 지정해서 사용한다.
- 파일 저장시에는 반드시 '클래스명.java ' 형식으로 저장해야 된다.

- 예> 학생객체 → Student.java
  사원객체 -> Employee.java

- 인스턴스 변수 (instance variable)

문법:

```
public class 클래스명{
   [접근지정자]  데이터형    변수명;
}
```

- 클래스가 처리할 데이터를 저장하는 용도이다.
- 클래스를 사용하기 위해서는 반드시 객체생성(인스턴스화)을 해야 하는 데, 이때 생성되는 변수이다. (객체 생성시 매번 생성된다.)
- 변수명은 일반 변수명 규칙을 따른다.(의미 있는 명사형,소문자)
- Heap 메모리에 저장된다.

- 변수 (variable) 종류

가. 로컬변수 (local)

- 메소드 안에서 선언된 변수이다.
- 메소드가 호출될 때 생성되고 메소드가 종료될 때 제거된다.
  stack 메모리에 저장된다. 임시적으로 사용된다.
- 접근 지정자를 사용하지 못한다.
- 기본적으로 블록({}) scope를 따른다.
- 반드시 사용전 에 초기화해야 된다.

나. 인스턴스 변수 (instance)

- 메소드 밖에서 선언된 변수이다.
- 객체 생성할 때 생성되고 객체가 제거될 때 삭제된다.
  heap 메모리에 저장된다. 클래스의 인스턴스 정보(속성)를 저장하는 용도이다.
- 사용전에 초기화 하지 않으면 기본값으로 자동 설정된다.
- 기본적으로 클래스 블록({}) scope를 따른다.

다. 클래스 변수 (static)

- 메소드 밖에서 선언된 변수이다.
  인스턴스 변수와 차이점은 변수 선언시 static 키워드를 사용한다.

예> int num; // 인스턴스 변수
static int age; // 클래스 변수

- 프로그램이 실행될 때 자동으로 생성되고, 프로그램이 종료 될때 삭제된다.
- 프로그램은 한번만 실행되기 때문에 클래스 변수도 단 한번만 생성된다.
  객체 생성 전에 생성이 된다. 따라서 '클래스명.변수' 형식으로 접근한다.
- 데이터 누적용으로 주로 사용된다.
- Method Area 영역에 저장된다. (static 키워드 및 상수 저장 영역)
- 초기화 하지 않으면 자동으로 기본값으로 자동 설정된다.

- 메서드 (method)

문법:

```
public class 클래스명{
    [지정자]  리턴타입   메소드명([변수,변수2]){
        …
            [return [값];]
    }
}
```

- 일반적으로 인스턴스 변수에 저장된 데이터를 수정 및 조회하는 역할이다.
- 메서드를 사용하기 위해서는 인스턴스 변수와 마찬가지로 반드시 클래스를 객체 생성 해야 된다. 객체 생성시마다 메서드가 생성되지 않고 공유해서 사용한다. 이유는 메서드 블록({})내의 코드가 모두 동일하기 때문이다.
- 메서드는 반드시 객체 생성 후에 명시적으로 호출해야 수행된다.
- 호출해서 수행된 메서드는 모든 작업이 끝나면 호출한 곳으로 돌아간다. (return 키워드 때문이다.)

- 메서드 역할에 따른 분류

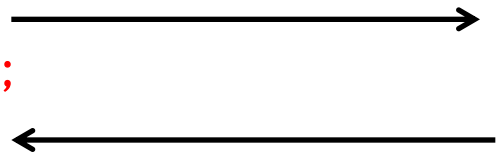
가. getter 메소드

특정 결과값을 얻고자 작성한 메서드를 의미한다. 메서드 호출시 주의할 점은 반드시 메서드명과 인자리스트(순서,개수,타입)가 동일해야 된다.

메서드명은 get변수명(첫 글자는 대문자)형식으로 지정한다.

예>

```
//caller 메소드
String name = getName();
```



```
// worker 메소드
public String getName(){
  return "홍길동";
}
```

```
//caller 메소드
int age= getAge();
```

```
// worker 메소드
public int getAge(){
  return  20;
}
```

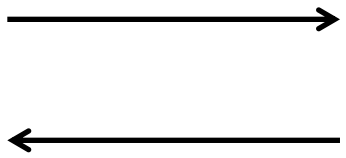
나. setter 메소드

특정값을 설정(넘겨 줄때)하고자 할 때 사용된다.

메서드명은 set변수명(첫글자는 대문자)형식으로 지정한다.

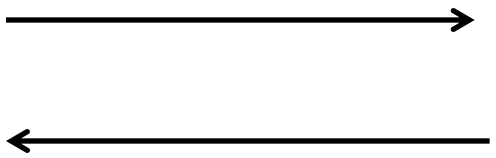
예>

```
//caller 메소드
setName("홍길동");
```



```
// worker 메소드
String name;
public void setName(String n){
    name = n;
    return ; //생략가능
}
```

```
//caller 메소드
setAge(20);
```



```
// worker 메소드
Int age;
public void setAge(int  a){
    age = a;
}
```

- 생성자 (constructor)

문법:

```
public class 클래스명{
    [접근 지정자] 클래스명([인자,인자2]){
        ...
    }
}
```

- 인스턴스 변수에 데이터를 초기화하는 역할이다. 즉, 인스턴스변수에 데이터를 맨 처음 저장하는 역할이다.
- setter 메서드를 사용하여 초기화 할 수도 있지만, 자바에서는 특별하게 초기화하는 역할을 담당하는 생성자를 제공한다. 따라서 setter 메서드는 인스턴스 변수에 저장된 데이터를 수정하는 역할을 담당하게 된다.

❑ 3] 클래스 구성요소

- 메서드와 여러 가지로 비슷한 특징을 갖는다. 메서드처럼 , 호출해야 수행이 된다. 수행 후에는 호출한 곳으로 돌아간다.
- 중요한 차이점은 리턴 타입이 없으며, 반드시 클래스명으로 지정해야 된다.
- 또한 자동으로 '기본 생성자(default constructor)'가 생성된다.
- 기본 생성자의 형식은 다음과 같다.

```
public 클래스명(){}
```

만약, 개발자가 명시적으로 생성자를 작성하면 기본 생성자는 더 이상 자동으로 제공되지 않는다. 이유는 개발자가 명시적으로 작성한 생성자를 이용해서 초기화해야 된다는 것을 의미한다.

필요에 의해서 생성자를 여러 개 작성할 수도 있다.(오버로딩 생성자)

❍ 객체 생성

문법:

클래스명 참조 변수명 = new 클래스명([인자,인자2]);

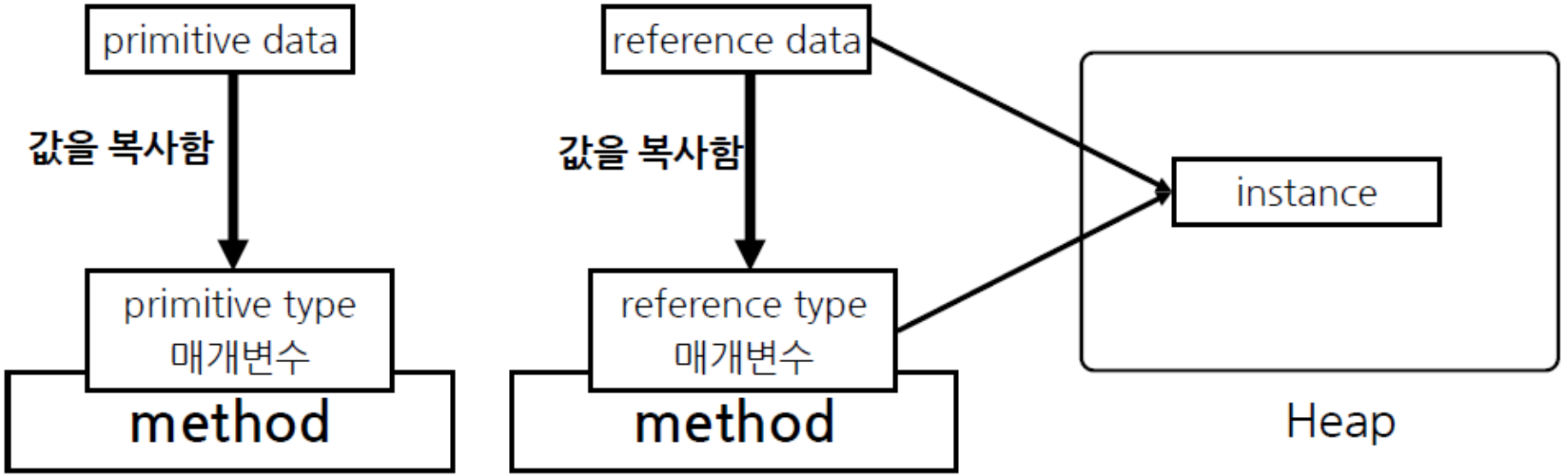
- 정의한 클래스를 이용하여 실객체를 메모리에 올려놓는 방법이다.
- 메모리에 올라간 클래스의 구현체(실객체)를 '인스턴스(instance)'라고 한다.
- Heap 메모리에 클래스의 멤버(인스턴스변수/메소드)가 생성된다. 인스턴스 변수는 new할 때마다 매번 생성되며, 메서드는 공유해서 사용한다.
- 참조 변수명에는 heap 메모리에 저장된 클래스의 인스턴스 위치값이 저장되어 있어서, 이 참조변수를 이용하여 클래스의 멤버를 접근할 수 있다.
- 예> 참조변수명.멤버
- 같은 클래스 안의 변수와 메소드는 .(dot)없이 바로 접근 가능하다.

Call by Value

- 자바의 매개변수는 값 복사를 통해서만 전달된다.

즉, 메소드 호출시 매개변수로 전달되는 값은 기본 데이터인 경우에는 변수에 저장된 실제값이 전달되고,

잠조 데이터형인 경우에는 변수에 저장된 객체의 주소값이 전달된다.



❑ 5] 오버로딩 메소드/ 오버로딩 생성자

기본적으로 같은 클래스 내에서는 이름이 중복되면 식별이 불가능하기 때문에 이름이 중복되면 에러가 발생된다.

하지만 메소드와 생성자는 이름과 인자로 식별이 가능하기 때문에 동일한 이름의 메소드와 생성자가 여러 개 지정될 수 있다. 이것을 '오버로딩 메소드/오버로딩 생성자'라고 한다. 재사용성을 위해서 지원된다.

규칙:

반드시 인자리스트(순서, 타입 , 개수)가 달라야 된다.

예>

```
public  Student(){}
pubic   Student(String n){}
public  Student(String n, int a){}
public  Student( int a, String n){}
```

```
public int a(){}
public int a(int n){}
public int a(String n){}
public int a(int a, String n){}

public void a(){}
```

❑ 6] this 키워드

❍ this 는 객체생성후에 메모리에 올라간 자기자신의 인스턴스를 가리킨다.
다른 클래스에서 는 참조변수명을 사용하여 생성된 인스턴스를 참조할 수 있지만 자기자신이 참조할 때는 this 키워드를 사용한다.

* this 키워드를 명시적으로 사용하는 2가지 형태

가. 인스턴스 변수명과 로컬변수명이 동일한 경우

예>
```
String name;
public void setName( String name){
    this.name  = name;
}
```

나. 오버로딩 생성자에서 (생성자에서 다른 생성자를 호출할 때)

```
예>   public Student( String n, int a , String addr){
          this.name = n;
          this.age = a;
          this.address = addr;
      }
      public Student( String n, int a ){
          this( n, a , "서울");
       }
      public Student( String n ){
          this( n , 20 );
      }
```

반드시 생성자 첫 라인에서 호출해야 된다. 중복코드를 제거하여 재사용이 가능하다.

❍ package 문

윈도우의 파일을 관리하기 위해서 폴더를 사용하는 것처럼, 클래스 파일들을 효율적으로 관리하기 위해서 package을 사용할 수 있다.

문법:

```
package  패키지명.패키지명2;
```

- 하나의 클래스에서 한번만 사용 가능하다.
- 클래스 선언보다 먼저 사용한다.
- 패키지명은 계층구조를 가질 수 있다.
- 중복되면 안되기 때문에 일반적으로 도메인명으로 지정한다.
- 패키지가 다르면 접근 불가능하다. (import 문 해결)
- 일반 컴파일이 아닌 패키지 컴파일을 해야 된다.
- 시스템이 사용한 패키지명으로 지정 불가능하다. (java 로 시작 안됨)
- 시스템이 제공하는 API는 모두 package로 제공된다.

❍ import 문

기본적으로 package가 서로 다르면 접근이 불가능하다. 이것을 가능하게 하는 방법이 import 문을 사용하는 것이다.

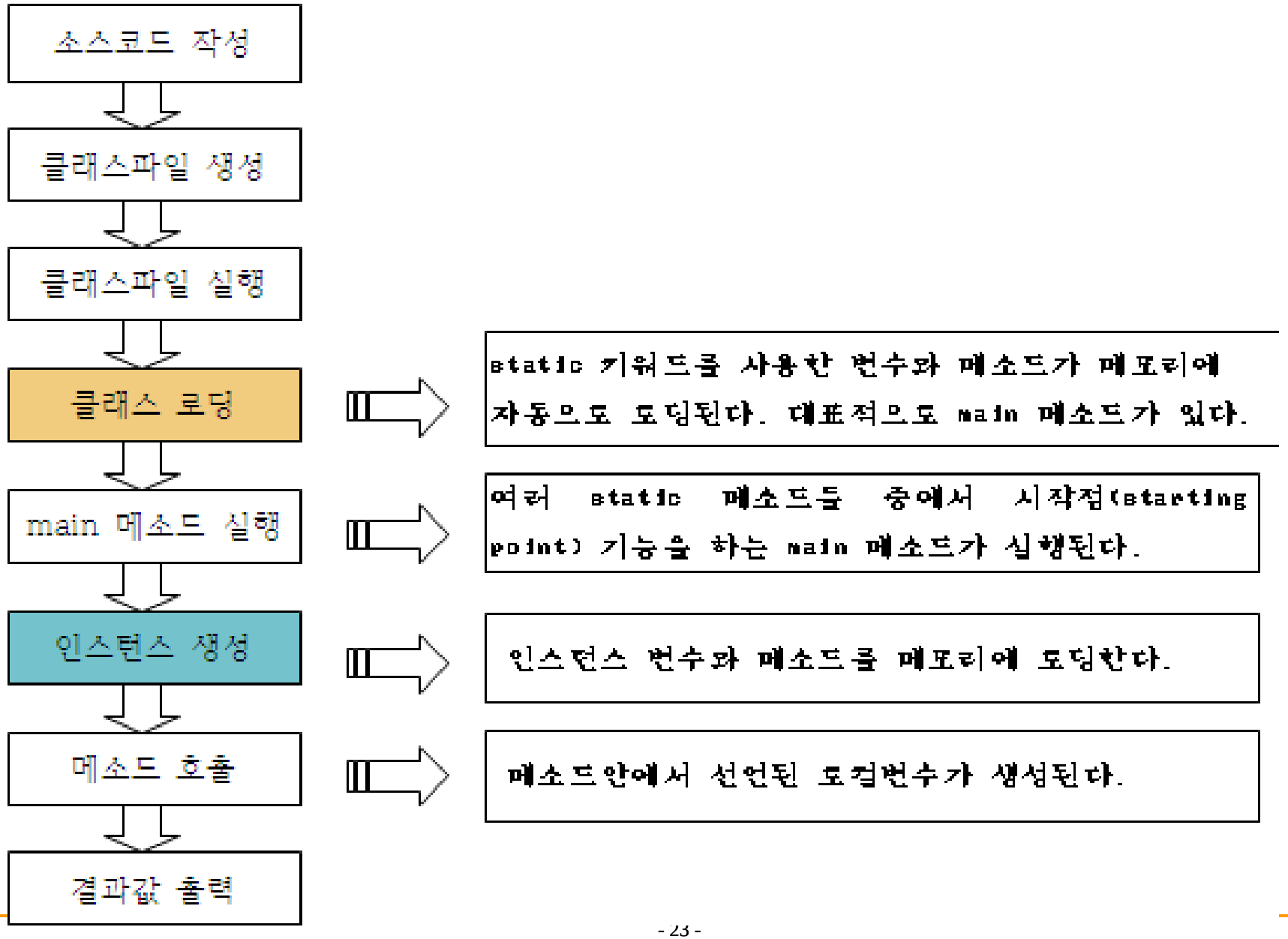
문법:

```
import 패키지명.패키지명2.클래스명;
```

- 하나의 클래스에서 여러 번 사용 가능하다.
- 클래스 선언보다 먼저 사용하고 package문 보다 나중에 사용한다.
- 클래스명 대신에 * 를 사용할 수 있지만 권장 안함
- 시스템이 제공하는 API중에서 java.lang 패키지는 자동으로 import 된다.

역으로 시스템이 제공하는 API중에서 java.lang 패키지를 제외하고는 모두 import 해서 사용해야 된다.

❑ 8) static 키워드


소스코드 작성
클래스파일 생성
클래스파일 실행
클래스 로딩
main 메소드 실행
인스턴스 생성
메소드 호출
결과값 출력
static 키워드를 사용한 변수와 메소드가 메모리에 자동으로 로딩된다. 대표적으로 main 메소드가 있다.
여러 static 메소드들 중에서 시작점(starting point) 기능을 하는 main 메소드가 실행된다.
인스턴스 변수와 메소드를 메모리에 로딩한다.
메소드안에서 선언된 로컬변수가 생성된다.

❑ 8) static 키워드

특징:

- 클래스, 변수, 메소드 지정자로 사용 가능 하다.
- 단 한번 생성된다.
- 프로그램이 실행될 때 생성되기 때문에 객체생성과 무관하다.

즉, 객체생성 없이도 사용 가능하다. (클래스명으로 접근)
프로그램이 종료될 때 제거된다.

- static 메소드는 Overriding 불가, static 블록 이용한 초기화 가능.

용도:

- 클래스 용도 : inner 클래스에서만 사용 가능
- 메소드 용도 : 객체 생성없이 메소드를 사용하기 위해서. (편리성)
- 예> int num = Integer.parseInt("123")

주의할 점은 static 메서드내에서 인스턴스 변수를 사용하지 못한다.

- 변수 용도: 데이터 누적용
  프로그램 내에서 특정 데이터값을 계속 유지할 목적이라면
  static변수가 적합하다.

❑ 8) static import

static으로 정의된 변수와 메소드는 매번 클래스 이름으로 접근하지 않고도 사용할 수 있도록 하는 방법이다.

예> import static java.lang.Math.PI;
import static java.lang.Integer.parseInt;

```
System.out.println( Math.PI );
System.out.println( PI );

String s  = parseInt( “123”);
```

❑ 9) final 키워드

특징:

– '마지막' 의미

용도:

– 클래스 용도: 상속 불가
– 메소드 용도: overriding 불가
– 변수 용도: 값 변경 불가 (상수)

- 상수

문법:

public static final 데이터 형 상수 = 값;

예> public static final int NUM = 100;

❑ 10] Varargs [가변 인자]

- 기본적으로 메소드를 호출할 때에는 반드시 메소드명과 인자리스트가 일치해야 된다.

하지만 Varargs을 사용하면 메소드명과 인자타입만 일치하면 호출이 가능하다.
인자 갯수는 자동으로 배열로 관리된다.
오버로딩 메소드를 사용하지 않아도 된다.(중복제거효과)

문법:

```
[지정자] 리턴타입  메소드명(  데이터형  ...  변수명){

    }
```

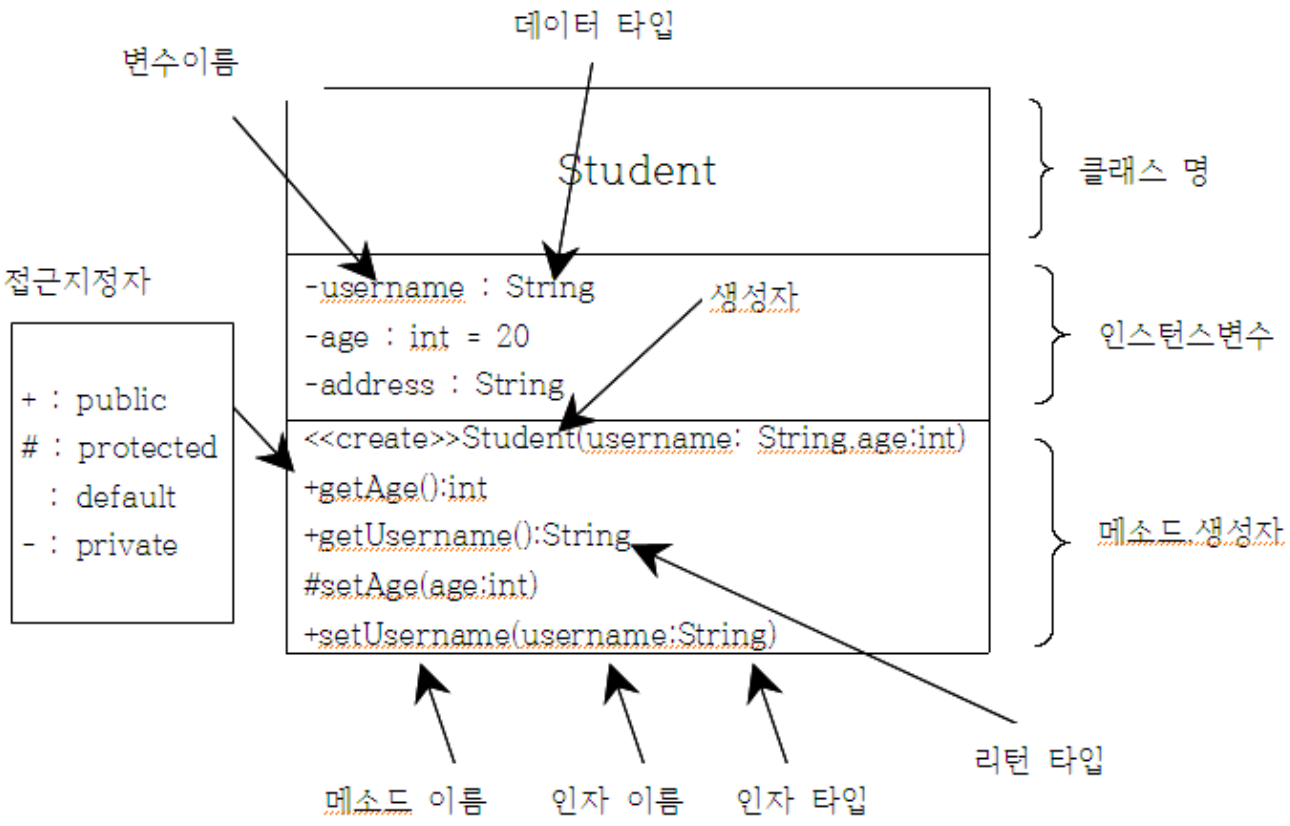
예>

```
public  void   a( int  ... n ){ }

    a(10);
    a(10,20);
    a(10,20,30);
```

❍클래스 다이어그램 (p182 참조)

- 클래스 내부의 정적인 내용이나 클래스 관계를 표기할 수 있으며, 클래스의 구성요소인 인스턴스 변수와 메소드, 생성자를 도식화할 수 있다. 또한 클래스간의 상속관계, 의존관계등을 표기할 수 있다.



Thank you