Chap04.
필드 (Field)

▶ 필드 (Field)

✓ 변수의 종류

필드에서는 변수를 선언하게 되며 아래와 같이 구분한다.

1. 클래스 변수 : static 키워드를 가지고 필드에 선언하는 변수, 메모리의 static영역 사용
2. 멤버 변수(인스턴스 변수) : static 키워드 없이 필드에 선언하는 변수, 메모리의 heap 영역 사용
3. 지역 변수 : 메서드, 생성자, 초기화 블록 내부에서 선언하는 변수

변수구분	소멸시기
클래스변수	프로그램 종료시
멤버변수(인스턴스변수)	객체소멸시(GC 소관)
지역변수	메소드 종료시

▶ 필드 (Field)

✓ 필드 표현식

```
[접근제한자] [예약어] class 클래스명 {

        [접근제한자] [예약어] 자료형 변수명 [= 초기값];

}
```

✓ 필드 예시

```
public class Academy {
        public int temp1;
        protected int temp2;
        int temp3;              //접근제한자 생략 시 default
        private int temp4;      //캡슐화 원칙으로 private 사용
}
```

▶ 필드 (Field)

✓ 필드 접근제한자

구분		해당 클래스 내부	같은 패키지 내	후손 클래스 내	전체
+	public	O	O	O	O
#	protected	O	O	O	
~	(default)	O	O		
-	**private**	O			

▶ 필드 (Field)

✓ 필드 예약어 - static

같은 타입의 여러 객체가 공유할 목적의 필드에 사용하며,

프로그램 start시에 정적 메모리 영역에 자동 할당되는 멤버에 적용

✓ static 표현식

```
public class Academy {
        private static int temp1;
}
```

▶ 필드 (Field)

✓ 필드 예약어 - final

하나의 값만 계속 저장해야 하는 변수에 사용하는 예약어

✓ final 표현식

```
public class Academy {
        private final int TEMP1 = 100;
        private int temp4;
}
```

▶ 필드 (Field) - 클래스 초기화 블럭

✓ 클래스 초기화 블럭

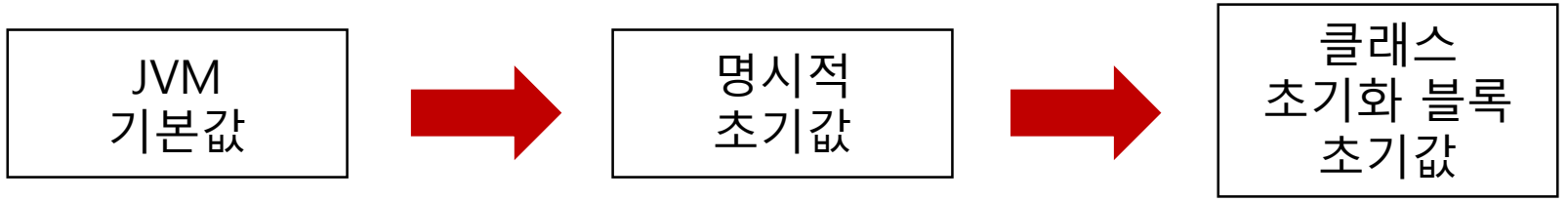
- **인스턴스 블럭 ({ })** - 인스턴스 변수를 초기화 시키는 블럭으로 객체 생성시 마다 초기화
- **static(클래스) 블럭 (static{ })** - static 필드를 초기화 시키는 블럭으로 프로그램 시작 시 한 번만 초기화

✓ 클래스 초기화 블럭 표현식

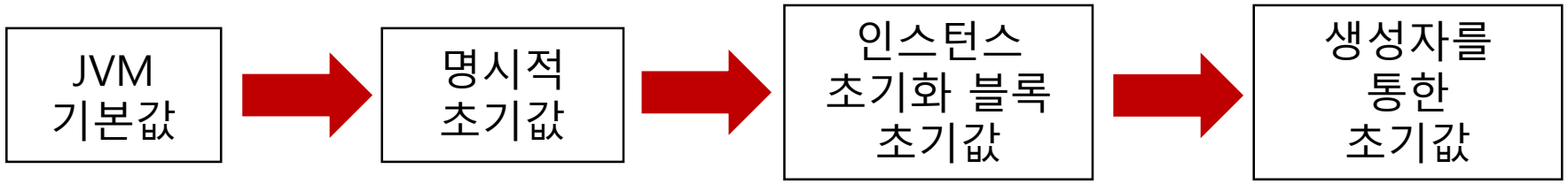
```
[접근제한자] [예약어] class 클래스명 {
        [접근제한자] static 자료형 필드1;
        [접근제한자] 자료형 필드2;

        static{  필드1 = 초기값;  }
        {  필드2 = 초기값;  }
}
```

필드 (Field) – 초기화 순서

✓ 클래스 변수



✓ 인스턴스 변수



▶ 실습문제 2

다음 코드의 콘솔결과를 예상해보고, 실제 출력값을 확인하세요.

com.kh.practice2.model.vo.Sample

```
private int a = 10;
public static int b;

{
    System.out.println(a);
    a = 100;
    System.out.println(a);
}

static {
    System.out.println(b);
    b = 99;
    System.out.println(b);
}
```

com.kh.practice2.controller.Run

```
public static void main(String args[]){

    Sample s1 = new Sample();
    Sample s2 = new Sample();
}
```

Chap05.
생성자(Constructor)

▶ 생성자 (Constructor)

✓ 생성자란 ?

객체가 new 연산자를 통해 Heap 메모리 영역에 할당될 때

객체 안에서 만들어지는 필드 초기화

생성자는 일종의 메소드로 전달된 초기값을 받아서 객체의 필드에 기록

✓ 생성자 규칙

생성자의 선언은 메소드 선언과 유사하나 반환 값이 없으며

생성자명을 클래스명과 똑같이 지정해주어야 함

▶ 생성자 (Constructor)

✓ 생성자 표현식

[접근제한자] [예약어] class 클래스명 {

[접근제한자] 클래스명() { }

[접근제한자] 클래스명(매개변수) { (this.)필드명 = 매개변수; }

}

```
public class Academy {
        private int studentNo;
        private String name;

        public Academy() {} // 기본 생성자
        public Academy(int studentNo, String name) { //매개변수 있는 생성자
                this.studentNo = studentNo;
                this.name = name;
        }
}
```

▶ 생성자 (Constructor)

✓ 기본 생성자

작성하지 않은 경우, 클래스 사용 시 **JVM이 자동으로 기본 생성자 생성**

✓ 매개변수 생성자

- 객체 생성 시 전달받은 값으로 객체를 초기화 하기 위해 사용
- **매개변수 생성자 작성 시 JVM이 기본 생성자를 자동으로 생성해주지 않음**
- 상속에서 사용 시 반드시 기본 생성자를 작성
- 오버로딩을 이용하여 작성

▶ 오버로딩

✓ 오버로딩이란 ?

한 클래스 내에 동일한 이름의 메소드를 여러 개 작성하는 기법

✓ 오버로딩 조건

- 같은 메소드 이름
- 다른 매개변수의 개수 또는 다른 매개변수 타입

▶ this

✓ this란 ?

모든 인스턴스의 메소드에 숨겨진 채 존재하는 레퍼런스로, 할당된 객체를 가리킴

함수 실행 시 전달되는 객체의 주소를 자동으로 받음

✓ this 사용 예시

```
public class Academy{
        private String name;
        public Academy() { }
        public Academy(String name) { this.name = name;  }
}
```

* 위와 같이 매개변수를 가지는 생성자에서 **매개변수 명이 필드명과 같은 경우**
매개변수의 변수명이 우선이므로 this 객체를 이용하여 대입되는 변수가 필드라는 것을 구분해줌

▶ this()

✓ this()란 ?

생성자, 같은 클래스의 다른 생성자를 호출할 때 사용, **반드시 첫 번째 줄에 선언해야 함**

✓ this() 사용 예시

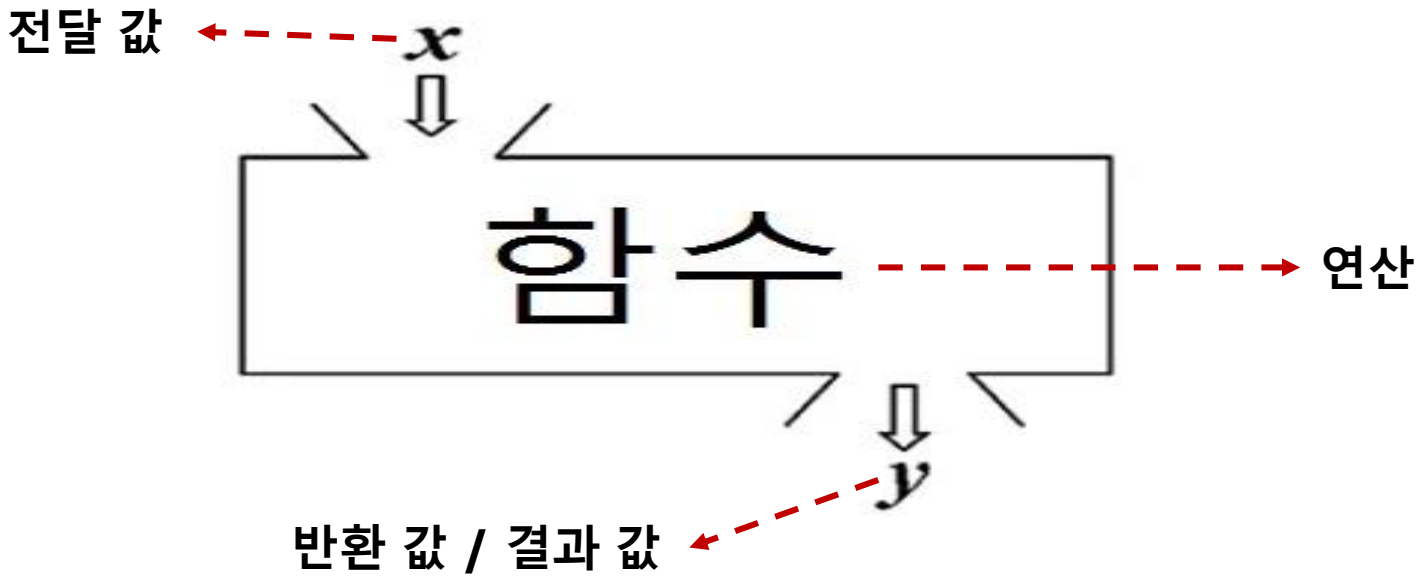
```
public class Academy{
        private int age;
        private String name;
        public Academy() { this(20, "김철수"); }
        public Academy(int age, String name) {
                this.age = age;      this.name = name;
        }
}
```

Chap06.
메소드(Method)

▶ 메소드 (Method)

✓ 메소드란 ?

수학의 함수와 비슷하며 호출을 통해 사용, 전달 값이 없는 상태로 호출을 하거나
어떤 값을 전달하여 호출을 하며, 함수 내에 작성된 연산 수행
수행 후 반환 값 / 결과 값은 있거나 없을 수 있음



▶ 메소드 (Method)

✓ 메소드 표현식

```
[접근제한자] [예약어] 반환형 메소드명( [매개변수] ) {
        // 기능 정의
}
```

```
public void information() {
        System.out.println(studentNo);
}
```

▶ 메소드 (Method)

✓ 메소드 접근제한자

구분		클래스	패키지	자손 클래스	전체
+	public	O	O	O	O
#	protected	O	O	O	
~	(default)	O	O		
-	private	O			

▶ 메소드 (Method)

✓ 메소드 예약어

구분	전체
static	static 영역에 할당하여 객체 생성 없이 사용
final	종단의 의미, 상속 시 오버라이딩 불가능
abstract	미완성된, 상속하여 오버라이딩으로 완성시켜 사용해야 함
synchronized	동기화 처리, 공유 자원에 한 개의 스레드만 접근 가능함
static final (final static)	static과 final의 의미를 둘 다 가짐

▶ 메소드 (Method)

✓ 메소드 반환형

구분	전체
void	반환형이 없음을 의미, 반환 값이 없을 경우 반드시 작성
기본 자료형	연산 수행 후 반환 값이 기본 자료형일 경우 사용
배열	연산 수행후 반환 값이 배열인 경우 배열의 주소값이 반환
클래스	연산 수행후 반환 값이 해당 클래스 타입의 객체일 경우 해당 객체의 주소값이 반환 (클래스 == 타입)

▶ 메소드 (Method)

✓ 메소드 매개변수

구분	전체
( )	매개변수가 없는 것을 의미
기본 자료형	기본형 매개변수 사용 시 값을 복사하여 전달 매개변수 값을 변경하여도 본래 값은 변경되지 않음
배열	배열, 클래스 등 참조형을 매개변수로 전달 시에는 데이터의 주소 값을 전달하기 때문에 매개변수를 수정하면 본래의 데이터가 수정됨(얕은 복사)
클래스	
가변인자	매개변수의 개수를 유동적으로 설정하는 방법으로 가변 매개변수 외 다른 매개변수가 있으면 가변 매개변수를 마지막에 설정 * 방법 : (자료형 ... 변수명)

* 매개변수의 수에 제한이 없다.

▶ 메소드 (Method)

✓ 메소드 표현식 – 매개변수가 없고 리턴 값이 있을 때

[접근제한자] [예약어] 반환형 메소드명() {

```
        // 기능 정의
```

}

```
public int information() {
        return studentNo;
}
```

▶ 메소드 (Method)

✓ 메소드 표현식 – 매개변수가 없고 리턴 값이 없을 때

[접근제한자] [예약어] void 메소드명() {

```
        // 기능 정의
}
```

```
public void information() {
        System.out.println(studentNo);
}
```

▶ 메소드 (Method)

✓ 메소드 표현식 – 매개변수가 있고 리턴 값이 있을 때

```
[접근제한자] [예약어] 반환형 메소드명(자료형 변수명) {
        // 기능 정의
}
```

```
public String information(String studentName) {
        return studentNo + " " + studentName;
}
```

▶ 메소드 (Method)

✓ 메소드 표현식 – 매개변수가 있고 리턴 값이 없을 때

[접근제한자] [예약어] void 메소드명(자료형 변수명) {

```
        // 기능 정의
}
```

```
public void information(String studentName) {
        System.out.println(studentNo + " " + studentName);
}
```

▶ getter와 setter 메소드

✓ setter 메소드

필드에 변경할 값을 전달 받아서 필드 값을 변경하는 메소드

✓ 표현식

[접근제한자] [예약어] void set필드명(자료형 변수명) {

(this.)필드명 = 자료형 변수명;

}

```
public void setStudentNo(int studentNo) {
        this.studentNo = studentNo;
}
```

▶ getter와 setter 메소드

✓ getter 메소드

필드에 기록된 값을 읽어서 요청한 쪽으로 읽은 값을 넘기는 메소드

✓ 표현식

[접근제한자] [예약어] 반환형 get필드명() {

return 필드명;

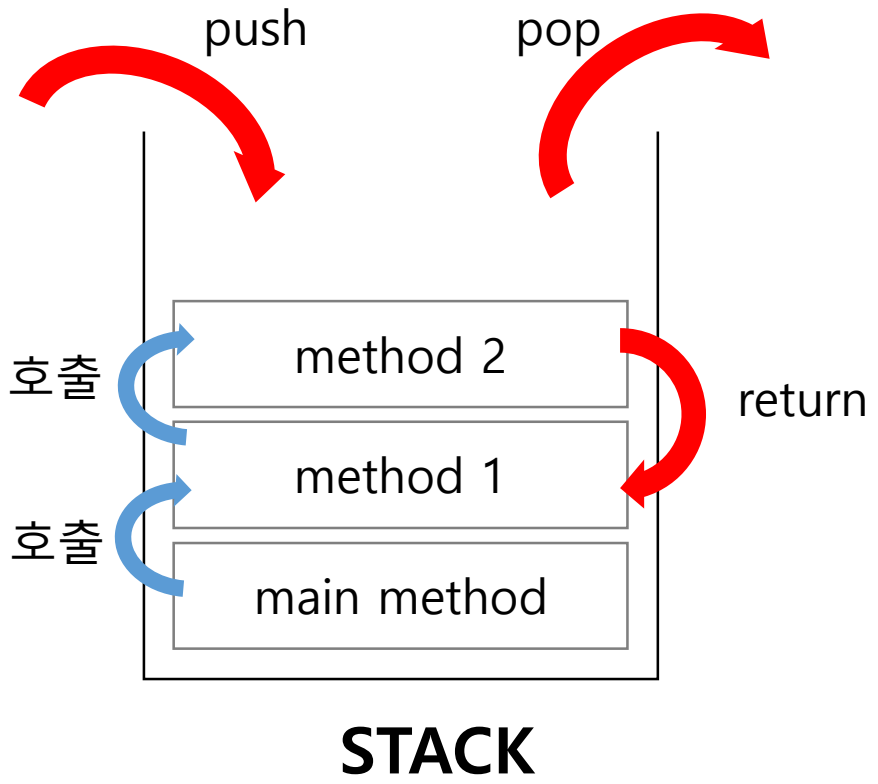
}

```
public int getStudentNo() {
        return studentNo;
}
```

return

해당 메소드를 종료하고 자신을 호출한 메소드로 돌아가는 예약어

반환 값이 있다면 반환 값을 가지고 자신을 호출한 메소드로 돌아감



* STACK의 자료구조 : LIFO(Last-Input-First-Out)