KH정보교육원

클래스와 객체2

목차

Chap04.
필드 (Field)

▶ 필드 (Field)

✓ 필드(Field)

클래스 어디에서 만들어지냐에 따라 다르다
클래스 안에 바로 만들어지는 애들, 생성자,메소드 모두가 사용하는 변수 ->필드
생성자와 메소드 내부에 선언 -> 로컬변수(지역 변수)

객체의 데이터가 저장되는 곳

변수와 비슷하지만 생성자나 메소드 외부에 선언되어 클래스 전체에서 사용됨

생성자와 메소드 내부에 선언된 변수는 로컬 변수(지역변수)라 부름

✓ 필드 종류

클래스변수 : 클래스 영역에 static키워드(예약어)를 가짐.

멤버변수(인스턴스 변수) : 클래스 영역에 선언.

▶ 필드 (Field)

✓ 필드 특징

객체가 생성될 때 만들어지고, 객체가 소멸할 때 같이 소멸됨 → 인스턴스 변수에 대한 특징

필드는 기본타입, 또는 참조타입 변수로 생성 가능

선언 시 초기값을 넣을 수 있음

클래스 내부에서 필드 사용 시 이름만 불러서 사용

클래스 외부에서 필드 사용 시 객체명.필드명 으로 사용

✓ 필드 표현식

[접근제한자] [예약어] class 클래스명 {

[접근제한자] [예약어] 자료형 변수명 [= 초기값];

}

대괄호 있는 애들 생략 가능

▶ 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)

정적 멤버-
클래스에 고정된 멤버로서 객체를 생성하지 않고 사용할 수 있는 필드와 메소드
같은 클래스의 객체들이 공유할 수 있는 공간이 하나 생성된다
정적 멤버는 객체에 소속된 멤버가 아니라 클래스에 소속된 멤버이기 때문에 클래스 멤버라고도 한다.

✓ 필드 예약어 – static (정적필드)

객체 만들지 않아도 해당 클래스 이름으로
static에 공간이 만들어진다

같은 타입의 여러 객체가 공유할 목적의 필드에 사용

프로그램 실행 시에 정적 메모리 영역에 자동 할당

같은 타입의 모든 객체가 꼭 필요한 공통적인 속성을 가진 변수에 사용

✓ static 표현식

```
public class VariableExam {
        public static int num;
}
```

▶ 필드 (Field)

✓ 필드 예약어 – final (상수)

필드가 가진 초기값 수정 불가능

초기값은 필드를 선언할 때나, 생성자 에서만 설정 가능

하나의 값만 계속 저장해야 하는 변수에 사용하는 예약어

✓ final 표현식

```
public class VariableExam {
        public final int NUM = 100;
}
```

▶ 필드 (Field)

별도 공간에 따로 만들어짐

✓ 필드 예약어 - static final

별도 공간에 따로 만들어진 절대로 변경이 되면 안되는 값
모두 같이 쓸 수 있는 고정된 값

정적필드(static)면서 상수(final)로 고정된 값

한번 초기값이 저장되면 변경할 수 없음

관례적으로 상수이름은 모두 대문자로 지정

여러 단어가 연결되는 경우 _ 로 구분

✓ static final 표현식

```
public class VariableExam {
        public static final int NUM = 100;
}
```

▶ 필드 초기화

✓ 초기화 방법

1. JVM 기본값 초기화 선언해놓고 값 아무것도 안 넣어놓을 때
2. 명시적 초기화 직접 지정해서
3. 초기화 블록을 이용한 초기화
4. 생성자를 이용한 초기화

✓ JVM 기본값 초기화

별도의 초기값을 지정해 주지 않은 경우 JVM에서 자동으로 초기값 설정

논리형 : false

정수형 : 0

실수형 : 0.0

문자형 : (공백)

문자열형 : null

▶ 필드 초기화

✓ 명시적 초기화

필드 선언 시 초기값 지정

✓ 명시적 초기화 예시

```
public class VariableExam {
    public boolean var1 = true;       // 논리형
    public int var2 = 10;             // 정수형
    public double var3 = 3.5;         // 실수형
    public char var4 = 'A’;           // 문자형
    public String var5 = "Hello";     // 문자열형
}
```

▶ 필드 초기화

KH정보교육원

✓ 초기화 블록

영역

초기화를 위한 별도의 영역을 만들어서 초기화를 시켜준다

- **인스턴스 블록 ({ })**

  인스턴스 변수를 초기화 시키는 블록으로 객체 생성시 마다 초기화

  객체가 생성될 때 초기화를 시켜줌

- **static(클래스) 블록 (static{ })** 정적 필드를 위한 블록

  static 필드를 초기화 시키는 블록으로 프로그램 시작 시 한 번만 초기화

✓ 초기화 블럭 표현식

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

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

▶ 필드 초기화

KH정보교육원

✓ 인스턴스 초기화 블록 예시

명시적

```
public class VariableExam {
    public boolean var1 = true;        // 논리형
    public int var2 = 10;              // 정수형
    public double var3 = 3.5;          // 실수형
    public char var4 = 'A';            // 문자형
    public String var5 = "Hello";      // 문자열형
    {
        var1 = false;
        var2 = 20;
        var3 = 5.6;
        var4 = 'B';
        var5 = "안녕하세요";
    }
}
```

블록 초기화

명시적, 블록 두개 쓰면
블록으로 적용된다

▶ 필드 초기화

KH정보교육원

✓ 클래스 초기화 블록 예시

```
public class VariableExam {
    public boolean var1 = true;       // 논리형
    public static int var2 = 10;      // 정수형
    public double var3 = 3.5;         // 실수형
    public char var4 = 'A';           // 문자형
    public String var5 = "Hello";     // 문자열형
    static {
        var2 = 20;
    }
    {
        var1 = false;
        var3 = 5.6;
        var4 = 'B';
        var5 = "안녕하세요";
    }
}
```

▶ 필드 초기화

KH정보교육원

✓ 생성자를 이용한 초기화

객체가 만들어 질 때 자동으로 실행됨

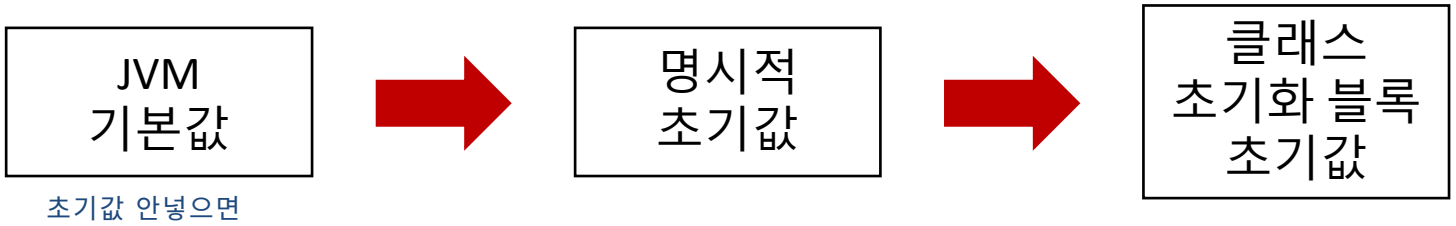
생성자 내부에 초기화 코드를 작성

✓ 생성자를 이용한 초기화 예시

```
public class VariableExam {
    public boolean var1;        // 논리형
    public int var2;            // 정수형
    public double var3;         // 실수형
    public char var4;           // 문자형
    public String var5;         // 문자열형
    public Variable() {
        var1 = true;
        var2 = 30;
        var3 = 10.5;
        var4 = 'C';
        var5 = "반갑습니다.";
    }
}
```

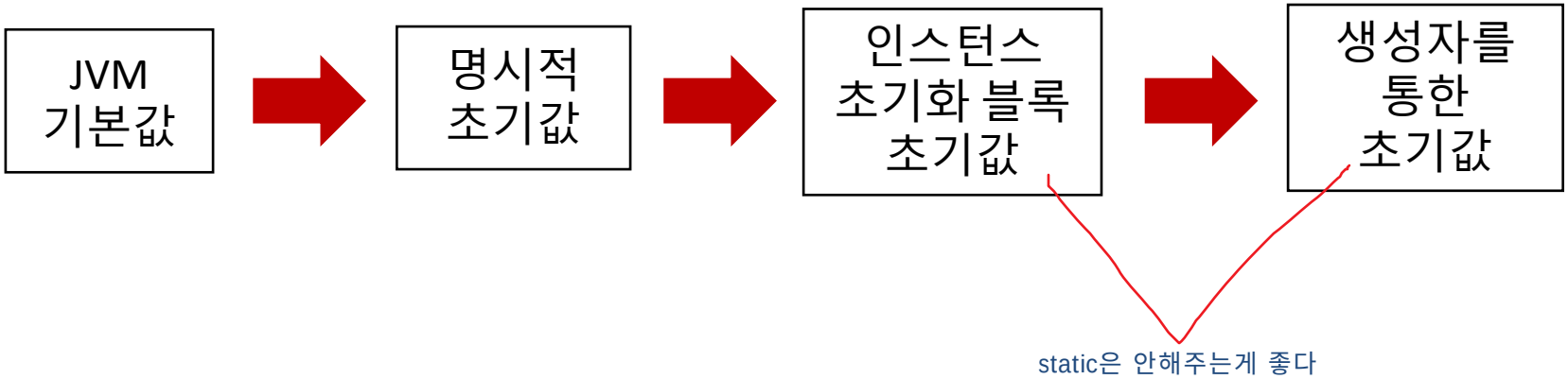
▶ 필드 (Field) – 초기화 순서

✓ 클래스 변수 static



초기값 안넣으면

✓ 인스턴스 변수



static은 안해주는게 좋다

가장 마지막에 실행 되는게 보인다

Chap05.
메소드(Method)

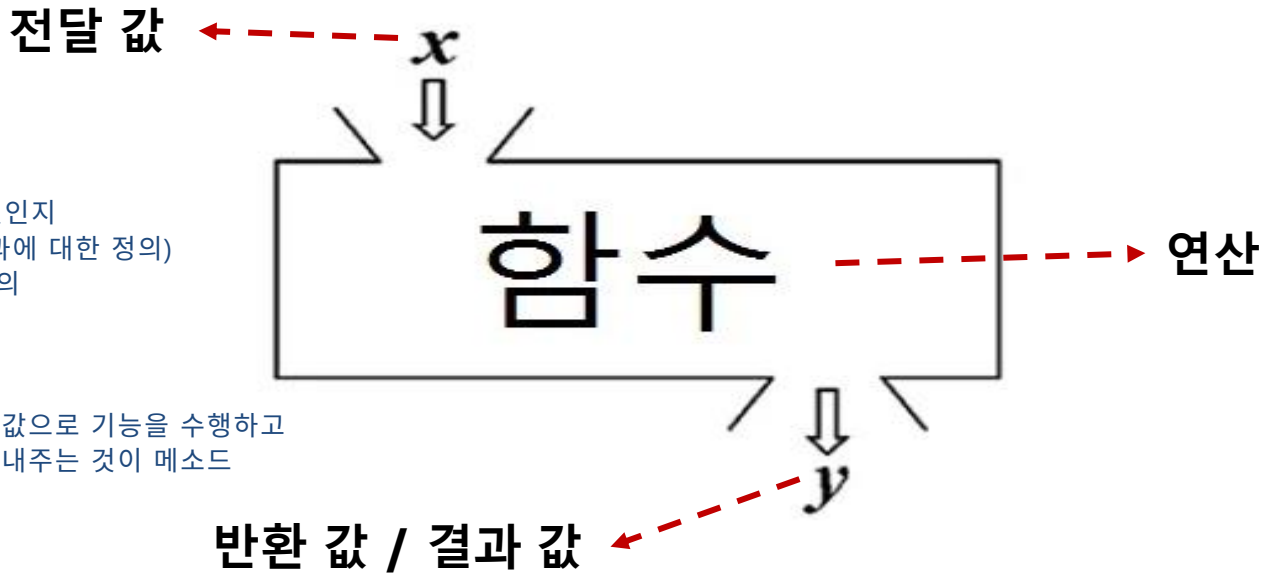
특정 기능을 가진 도구

▶ 메소드 (Method)

수학의 함수와 비슷하며 호출을 통해 사용

전달 값이 없는 상태로 호출하거나 어떤 값을 전달하여 호출

함수 내에 작성된 연산 수행 후 반환 값/결과 값은 있거나 없을 수 있음



1. 어떤 결과를 낼 것인지
-> 반환 자료형 (결과에 대한 정의)
2. 입력값에 대한 정의
3. 기능
4. 이름

값을 입력받아서 그 값으로 기능을 수행하고
그에 대한 결과값을 내주는 것이 메소드

▶ 메소드 (Method)

✓ 객체의 메소드

하나의 객체 안에 기능을 정의하기 위해서 사용하는 것이 메소드

객체의 기능을 수행하기 위한 코드 블록

객체의 메소드를 호출하면 메소드 블록 안에 있는 코드 들이 순서대로 실행 됨

메소드 명은 관례적으로 소문자로 작성

메소드를 호출할 때 매개변수와 동일한 타입과 개수의 값을 넘겨줘야 정상 실행됨

▶ 메소드 (Method)

✓ 메소드 표현식

보통 public

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

1 2 3 4

```
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 따로 객체를 만들지 않더라도
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 반환 값이 있다면 그 반환값은 무슨 형일지 정의를 해줘야한다
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)

KH정보교육원

✓ 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)

입력값이 있음

✓ 메소드 표현식 – 매개변수가 있고 리턴 값이 없을 때

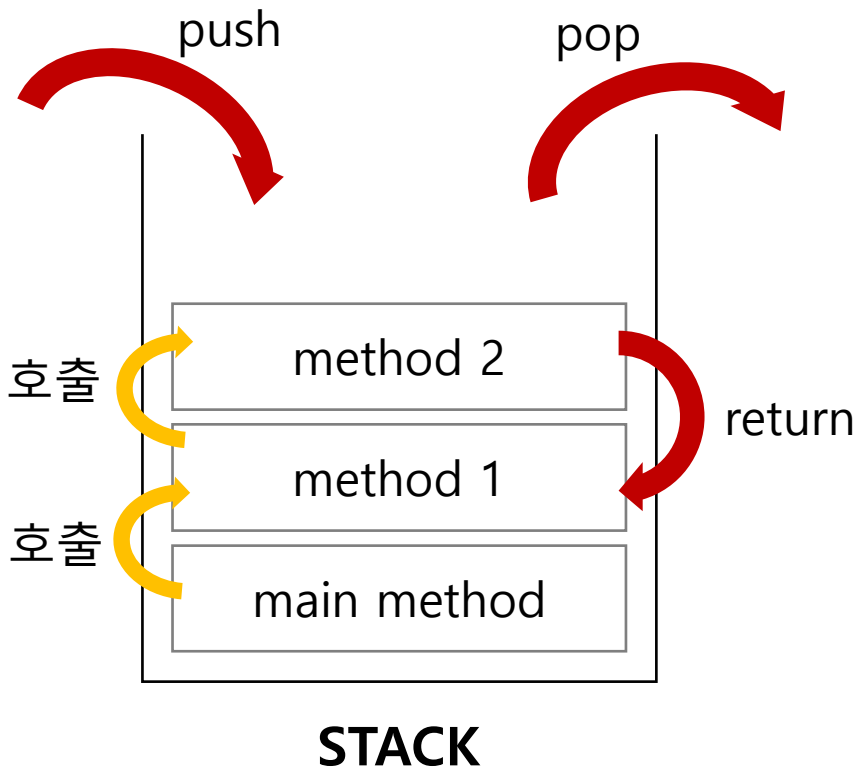
```
[접근제한자] [예약어] void 메소드명(자료형 변수명) {
        // 기능 정의
}
```

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

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)

▶ 오버로딩

동일한 이름의 메소드를 여러 개 정의해서 사용하는 것
하나의 함수가 마치 여러 일을 하는 것처럼 정의할 수 있기 때문에 다형성(polymorphism)이 구현 됨

✓ 오버로딩 조건

- 메소드의 이름이 같아야 함
- 매개 변수의 타입이나 개수를 다르게 정의해야 함
- 매개변수명은 무관함
- 리턴 타입은 무관함

▶ 오버로딩

KH정보교육원

✓ 매개변수의 형식이 다르므로 함수 오버로딩 성립

```
public void func(int num) { ... }
public void func(char ch) { ... }
```

✓ 매개변수의 개수가 다르므로 함수 오버로딩 성립

```
public void func(int num) { }
public void func(int num, int num2) { }
```

✓ 매개변수의 형식과 개수가 동일하고
반환 자료형만 다르므로 함수 오버로딩 성립 안됨

```
public void func(int num) { }
public int func(int num) { }
```

▶ 오버로딩

✓ 오버로딩 예시 – MethodExam.java

```
public class MethodExam {
    public void func() {
        System.out.println("1번째 메소드");
    }
    public void func(int num) {
        System.out.println("2번째 메소드");
    }
    public void func(char ch) {
        System.out.println("3번째 메소드");
    }
    public void func(int num1, int num2) {
        System.out.println("4번째 메소드");
    }
}
```

▶ 오버로딩

KH정보교육원

✓ 오버로딩 예시 – Run.java

```
MethodExam me = new MethodExam();
me.func();
me.func(10);
me.func('A');
me.func(10,20);
```

▶ this

모든 인스턴스 메소드에 숨겨진 채 존재하는 레퍼런스로 할당된 객체를 가리킴

함수 실행 시 전달되는 객체의 주소를 자동으로 받음

✓ this 사용 예시

```
public class MethodExam {
    public String name="홍길동";

    public void printName() {
        String name = "이길동";
        System.out.println("name : "+name);
        System.out.println("this.name : "+this.name);
    }
}
```

* 위와 같이 메서드 내 **변수 명이 필드명과 같은 경우**
메서드 내 변수가 우선이므로 this 객체를 이용하여 필드 사용 가능

▶ getter와 setter 메소드

✓ setter 메소드

입력 받은 값을 검증한 후 필드에 저장하도록 해주는 메소드

메소드명은 set+필드이름으로 지정(첫글자 대문자)

✓ 표현식

[접근제한자] [예약어] void set필드명(자료형 변수명) {

(this.)필드명 = 변수명;

}

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

▶ getter와 setter 메소드

✓ getter 메소드

필드의 값을 가공한 후 외부에서 읽을 수 있도록 반환해 주는 메소드

메소드명은 get+필드이름으로 지정(첫글자 대문자)

필드타입이 boolean일 경우 is+필드이름으로 지정(첫글자 대문자)

✓ 표현식

[접근제한자] [예약어] 반환형 get필드명() {

return 필드명;

}

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

Chap06.
생성자(Constructor)

▶ 생성자 (Constructor)

객체가 생성되면 자동으로 호출되는 메소드

클래스명과 이름이 같음

일반적으로 객체가 제대로 동작 할 수 있게 **초기화 (기본값 입력)** 하는 용도로 많이 사용됨

✓ 생성자 규칙

생성자의 선언은 메소드 선언과 유사하나 반환 값이 없으며

생성자명을 클래스명과 똑같이 지정해주어야 함

▶ 생성자 (Constructor)

✓ 기본(Default) 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

▶ 생성자 (Constructor)

✓ 생성자 표현식

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

✓ 생성자 예시1 - ConstructorExam.java

```java
public class ConstructorExam {
    public String name;
    public int age;

    public ConstructorExam(){
        System.out.println("생성자1 호출");
    }
}
```

▶ 생성자 (Constructor)

✓ 생성자 예시2 - ConstructorExam.java

```
public class ConstructorExam {
    public String name;
    public int age;

    public ConstructorExam(String name, int age) {
        System.out.println("생성자2 호출");
    }
}
```

✓ 생성자 예시 – Run.java

```
ConstructorExam ce = new ConstructorExam();       // 예시 1
ConstructorExam ce = new ConstructorExam("홍길동",20);// 예시2
```

▶ 생성자 (Constructor)

KH정보교육원

✓ 생성자 오버로딩 예시 – ConstructorExam.java

```
public class ConstructorExam {
    public String name;
    public int age;

    public ConstructorExam(){
        System.out.println("생성자1 호출");
    }
    public ConstructorExam(String name, int age) {
        System.out.println("생성자2 호출");
    }
}
```

✓ 생성자 오버로딩 예시 – Run.java

```
ConstructorExam ce1 = new ConstructorExam();
ConstructorExam ce2 = new ConstructorExam("홍길동",20);
```

▶ this() 생성자 호출

생성자에서 다른 생성자를 호출하는 경우

생성자의 첫 줄에서만 혀용

✓ this() 표현식

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

▶ this() 생성자 호출

✓ this() 사용 예시 - ConstructorExam.java

```
public class ConstructorExam {
    public String name;
    public int age;

    public ConstructorExam(){
        this("이름없음",0);
        System.out.println("생성자1 호출");
    }
    public ConstructorExam(String name, int age) {
        System.out.println("생성자2 호출");
    }
}
```

✓ this() 사용 예시 - Run.java

```
ConstructorExam ce1 = new ConstructorExam();
```

Chap07.
package와 import

소스파일

✓ 소스파일 구성 순서

1. package문
2. import문
3. 클래스 선언

✓ 소스파일 구성 예시

```
package kh.academy; //package문

import java.util.Date; //import문

public class ImportTest { //클래스 선언
        public static void main(String[] args) {

        }
}
```

▶ package

서로 관련된 클래스 혹은 인터페이스의 묶음으로 폴더와 비슷

패키지는 서브 패키지를 가질 수 있으며 '.'으로 구분

예) Scanner 클래스의 full name은 패키지명이 포함된 java.util.Scanner 이다.

✓ 패키지의 선언

소스파일 첫 번째 문장에 단 한번 선언하며 하나의 소스파일에

둘 이상의 클래스가 포함된 경우, 모두 같은 패키지에 속함

모든 클래스는 하나의 패키지에 속하며, 패키지가 선언되지 않은

클래스는 자동적으로 이름없는 패키지(default)에 속하게 됨

예) package java.util;

▶ package

✓ 패키지 명명규칙

숫자로 시작할 수 없고 $, _ 를 제외한 특수문자 사용 불가

java로 시작하는 패키지는 자바 표준 API 에서만 사용되므로 사용 불가

관례적으로 모두 소문자로 작성

일반적으로 회사에서는 도메인이름으로 많이 작성함

관례적으로 도메인 이름은 역순으로 지정 후 마지막에 프로젝트 이름을 붙여줌

예) package kr.or.iei.project1

▶ package

✓ 같은 패키지 내 클래스 사용

같은 패키지에 속한 클래스들은 조건 없이 다른 클래스를 사용 가능

✓ 다른 패키지 내 클래스 사용

1. 패키지와 클래스명을 모두 적어 사용하는 방법

   형식 : 패키지명.클래스명 필드명

2. import문 사용하는 방법

▶ import

사용할 클래스가 속한 패키지를 지정하는 데 사용

import문을 사용하면 클래스를 사용할 때 패키지 명 생략 가능

java.lang패키지의 클래스는 import를 하지 않고도 사용 가능

* java.lang 패키지 내의 클래스 → String, Object, System....

✓ import문의 선언

import문은 패키지문과 클래스 선언의 사이에 선언하며 컴파일 시에 처리되므로 프로그램 성능에 영향을 주지 않음

지정된 패키지에 포함된 클래스는 import 가능하지만 서브 패키지에 속한 모든 클래스까지는 불가능

예)
```
import java.util.Date;
import java.util.*;          //  java.util 패키지 내의 모든 클래스 (단, 서브클래스는 X)
import java.*;               //  불가능
```

import

✓ import문 주의사항

이름이 같은 클래스가 속한 두 패키지를 import 할 때는 클래스 앞에 패키지 명을 붙여 구분해 주어야 함

예)
```
package kh.academy;

   import java.sql.Date;

   public class ImportTest {

      public static void main(String[] args) {
         java.util.Date today = new java.util.Date();
      }

   }
```