Chapter 9. 클래스 : 기능

학습목표

- 클래스는 크게 속성과 기능으로 구성됩니다. 8장에서 속성에 대해 학습하였고, 이 장에서는 클래스의 가장 중요한 역할을 담당하는 기능(생성자와 메소드)에 대해 기술합니다.
- 객체를 생성할 때 초기화를 담당하는 생성자에 관해 학습합니다.
- 생성자 오버로딩에 관해 학습합니다.
- 클래스의 기능을 나타내는 메소드에 관해 학습합니다.
- main 메소드에 관해 학습합니다.
- 메소드 오버로딩에 관해 학습합니다.
- 메소드 값 전달 기법에 관해 학습합니다.

목차

Section 1. 생성자

Section 2. 생성자 오버로딩

Section 3. 예약어 this

Section 4. 메소드

Section 5. 메소드 오버로딩

Section 6. 메소드에 값 전달 기법

Section 1.

생성자

1.1 생성자의 역할

- 객체를 생성할 때 초기화하는 역할을 수행

그림 9-1 생성자의 역할

1.2 생성자의 구성과 사용

- **생성자는 메소드와 비슷하지만, 주로 객체의 초기화 과정을 수행**
  - 생성자는 객체가 생성될 때 자동으로 수행
  - 주로 객체의 초기화를 위해 사용
  - 생성자의 이름은 클래스의 이름과 동일

[형식] 생성자

```
[public / private] 클래스 이름([매개 변수],[매개 변수]....) {

  초기화 문장들
}
```

- 생성자는 반환 값이 없다
- 접근 한정자는 멤버 변수의 접근 한정자와 같은 의미(private 한정자는 내부적으로만 사용되는 생성자)
- 클래스에 생성자가 지정되지 않는 경우에는 묵시적 생성자가 자동 생성된다(묵시적 생성자는 매개 변수가 없는 생성자를 의미한다)

1.2 생성자의 구성과 사용

● 묵시적 생성자가 없는 경우

```
01: class Cons1 {
02:     public int num;
03: }
04: public class ConsTest1 {
05:     public static void main(String args[]) {
06:         Cons1 cons = new Cons1();
07:     }
08: }
```

생성자 없이 클래스 생성

클래스로부터 객체 생성

1.2 생성자의 구성과 사용

● 묵시적 생성자를 지정하는 경우

```
01: class Cons2 {
02:     public int num;
03:     public Cons2() {
04:         System.out.println("묵시적 생성자");
05:     }
06: }
07: public class ConsTest2 {
08:     public static void main(String args[]) {
09:         Cons2 cons = new Cons2();
10:     }
11: }
```

매개 변수가 없는 묵시적 생성자 선언.

Cons 클래스로부터 객체 생성. new 명령어에 의해 묵시적 생성자가 수행되어 "묵시적 생성자"가 출력된다.

1.2 생성자의 구성과 사용

● 명시적 생성자가 있는 경우

```
01: class Cons3 {
02:     public int num;
03:     public Cons3(String s) {
04:         System.out.println(s + " 명시적 생성자");
05:     }
06: }
07: public class ConsTest3 {
08:     public static void main(String args[]) {
09:         Cons3 cons1 = new Cons3("1번째");
10:         //Cons3 cons2 = new Cons3();
11:     }
12: }
```

매개 변수가 있는 명시적 생성자 선언

매개 변수를 지정하여 객체 생성.
"1번째 명시적 생성자" 출력

오류 발생. 해당되는 생성자가 없다. 클래스에 어떠한 생성자도 없는 경우에는 묵시적 생성자가 없어도 객체가 생성되지만, 명시적 생성자가 하나라도 있으면, 묵시적 생성자를 사용하기 위해서는 반드시 정의해야 한다(생성자 오버로딩–다음 절에서 설명).

1.2 생성자의 구성과 사용

예제 9.1 Box4Test1.java

```
01: class Box4 {
02:     int width;
03:     int height;
04:     int depth;
05:     public Box4(int w, int h, int d) <-- 3개의 매개 변수를 가진 생성자 선언
06:     {
07:         width = w;
08:         height = h;          <-- 속성에 값을 설정하는 초기화 과정
09:         depth = d;
10:     }
11: }
12: public class Box4Test1 {
13:     public static void main(String args[]) {
14:         Box4 mybox1 = new Box4(10,20,30); <-- 명시적 생성자를 사용하여 객체를 생성
15:         //Box4 mybox2 = new Box4(); <-- 오류 발생 적합한 생성자가 없다.
16:         int vol = mybox1.width * mybox1.height * mybox1.depth; <-- 박스의 부피를 구한다.
17:         System.out.println("박스의 부피 : " + vol);
18:     }
19: }
```

실행 결과

```
박스의 부피 : 6000
```

Section 2.
생성자 오버로딩

● 하나의 클래스에 여러 개의 생성자가 있다 : 생성자 오버로딩

- 생성자 매개 변수의 타입과 개수가 달라야 한다

예제 9.2 Box5Test1.java

```
01: class Box5 {
02:     int width;
03:     int height;
04:     int depth;
05:     public Box5()
06:     {
07:         width = 1;
08:         height = 1;
09:         depth = 1;
10:     }
11:     public Box5(int w)
12:     {
13:         width = w;
14:         height = 1;
15:         depth = 1;
16:     }
17:     public Box5(int w, int h)
18:     {
19:         width = w;
20:         height = h;
21:         depth = 1;
22:     }
23:     public Box5(int w, int h, int d)
24:     {
25:         width = w;
26:         height = h;
27:         depth = d;
28:     }
29: }
```

생성자 오버로딩. 매개 변수의 형과 개수가 다르다.

2 생성자 오버로딩

```
30: public class Box5Test1 {
31:     public static void main(String args[]) {
32:         Box5 mybox1 = new Box5(); <-------- 매개 변수가 없는 생성자 호출
33:         int vol = mybox1.width * mybox1.height * mybox1.depth;
34:         System.out.println("박스의 부피(매개 변수 없음) : " + vol);
35:         mybox1 = new Box5(10); <-------- 매개 변수가 하나 있는 생성자 호출
36:         vol = mybox1.width * mybox1.height * mybox1.depth;
37:         System.out.println("박스의 부피(매개 변수 1개) : " + vol);
38:         mybox1 = new Box5(10,20); <-------- 매개 변수가 두 개 있는 생성자 호출
39:         vol = mybox1.width * mybox1.height * mybox1.depth;
40:         System.out.println("박스의 부피(매개 변수 2개) : " + vol);
41:         mybox1 = new Box5(10,20,30); <-------- 매개 변수가 세 개 있는 생성자 호출
42:         vol = mybox1.width * mybox1.height * mybox1.depth;
43:         System.out.println("박스의 부피(매개 변수 3개) : " + vol);
44:     }
45: }
```

실행 결과

```
박스의 부피(매개 변수 없음) : 1
박스의 부피(매개 변수 1개) : 10
박스의 부피(매개 변수 2개) : 200
박스의 부피(매개 변수 3개) : 6000
```

1.2 생성자의 구성과 사용

예제 9.3 Box6Test1.java

```
01: class Box6 {
02:     int width;
03:     int height;
04:     int depth;
05:     double dwidth;
06:     double dheight;
07:     double ddepth;
08:     public Box6(int w, int h, int d)
09:     {
10:         width = w;
11:         height = h;
12:         depth = d;
13:     }
14:     public Box6(double w, double h, double d)
15:     {
16:         dwidth = w;
17:         dheight = h;
18:         ddepth = d;
19:     }
20: }
```

3개의 정수 매개 변수를 가진 생성자

3개의 실수 매개 변수를 가진 생성자

2 생성자 오버로딩

```
21: public class Box6Test1 {
22:     public static void main(String args[]) {
23:         Box6 mybox1 = new Box6(10,20,30);
24:         int vol = mybox1.width * mybox1.height * mybox1.depth;
25:         System.out.println("박스의 부피(정수 매개 변수) : " + vol);
26:         mybox1 = new Box6(10.5, 20.5, 30.5);
27:         double dvol = mybox1.dwidth * mybox1.dheight * mybox1.ddepth;
28:         System.out.println("박스의 부피(실수 매개 변수) : " + dvol);
29:         mybox1 = new Box6(10,20,30.5);
30:         dvol = mybox1.dwidth * mybox1.dheight * mybox1.ddepth;
31:         System.out.println("박스의 부피(정수와 실수 혼합) : " + dvol);
32:     }
33: }
```

3개의 정수 매개 변수를 가진 생성자 수행

3개의 실수 매개 변수를 가진 생성자 수행

혼합되어 있는 경우는 자동으로 확대 형 변환이 수행되어 3개의 실수 매개 변수를 가진 생성자가 수행

실행 결과

```
박스의 부피(정수 매개 변수) : 6000
박스의 부피(실수 매개 변수) : 6565.125
박스의 부피(정수와 실수 혼합) : 6100.0
```

Section 3.

예약어 this

3 예약어 this

- 생성자나 메소드에서 this가 사용되면, this는 자신을 가동시킨 객체를 의미

- 예

```
01: class Box {
02:     int width;
03:     int height;
04:     int depth;
05:     public Box(int w, int h, int d) {
06:         width=w;
07:         height=h;
08:         depth=d;
09:     }
10: }
```

매개 변수로 지정된 w, h, d의 의미를 알기 어렵다. 변수명을 width, height, depth로 하는 것이 더 명확하다.

3 예약어 this

- 예

```
01: class Box {
02:     int width;
03:     int height;
04:     int depth;
05:     public Box(int width, int height, int depth) {
06:         width=width;
07:         height=height;
08:         depth=depth;
09:     }
10: }
11: public class BoxTest {
12:       ..................................................
13:               Box mybox = new Box(10,20,30);
14:               System.out.println(mybox.width);
15:       ..................................................
16:  }
```

매개 변수로 width, height, depth를 사용하여 의미는 명확해졌다.

이 경우 width=width;의 의미는 자신의 변수에 자신의 값을 저장하게 된다. 즉 생성자 메소드 내의 변수로만 취급된다.

세 개의 값으로 객체를 생성

100이 아닌 00이 출력된다.

3 예약어 this

- 예

```
01: class Box {
02:     int width;
03:     int height;
04:     int depth;
05:     public Box(int width, int height, int depth) {
06:         this.width=width;
07:         this.height=height;
08:         this.depth=depth;
09:     }
10: }
11: public class BoxTest {
12:         ..................................................
13:                 Box mybox = new Box(10,20,30);
14:                 System.out.println(mybox.width);
15:         ..................................................
16: }
```

이 경우 this.width=width;는 생성자를 호출한 객체의 width(객체의 속성)에 자신의 width 값(생성자 지역 변수)을 배정

100이 출력된다.

- **this의 또 다른 용도 : 생성자 내에서 단독으로 사용**
  - 다른 생성자를 호출한다
  - 생성자 내에서 사용될 경우에는 반드시 첫 번째 라인에 위치해야 한다

3 예약어 this

예제 9.4 Box7Test1.java

```
01: class Box7 {
02:     int width;
03:     int height;
04:     int depth;
05:     public Box7()
06:     {
07:         this(1,1,1);
08:         System.out.println("매개 변수 없는 생성자 수행");
09:     }
10:     public Box7(int width)
11:     {
12:        this(width,1,1);
13:         System.out.println("매개 변수(1개) 생성자 수행");
14:     }
15:    public Box7(int width, int height)
```

this를 이용하여 3개의 매개 변수를 가진 생성자 호출

생성자 내에서 출력문 수행

this를 이용하여 3개의 매개 변수를 가진 생성자 호출

3 예약어 this

```
16:     {
17:         this(width,height,1);
18:         System.out.println("매개 변수(2개) 생성자 수행");
19:     }
20:     public Box7(int width, int height, int depth)
21:     {
22:         System.out.println("매개 변수(3개) 생성자 수행");
23:         this.width = width;
24:         this.height = height;
25:         this.depth = depth;
26:     }
27: }
```

this를 이용하여 3개의 매개 변수를 가진 생성자 호출

객체의 속성에 매개 변수의 값을 배정

3 예약어 this

```
28: public class Box7Test1 {
29:     public static void main(String args[]) {
30:         Box7 mybox1 = new Box7();
31:         int vol = mybox1.width * mybox1.height * mybox1.depth;
32:         System.out.println("박스의 부피(매개 변수 없음) : " + vol);
33:         mybox1 = new Box7(10);
34:         vol = mybox1.width * mybox1.height * mybox1.depth;
35:         System.out.println("박스의 부피(매개 변수 1개) : " + vol);
36:         mybox1 = new Box7(10,20);
37:         vol = mybox1.width * mybox1.height * mybox1.depth;
38:         System.out.println("박스의 부피(매개 변수 2개) : " + vol);
39:         mybox1 = new Box7(10,20,30);
40:         vol = mybox1.width * mybox1.height * mybox1.depth;
41:         System.out.println("박스의 부피(매개 변수 3개) : " + vol);
42:     }
43: }
```

실행 결과

```
매개 변수(3개) 생성자 수행
매개 변수 없는 생성자 수행
박스의 부피(매개 변수 없음) : 1
매개 변수(3개) 생성자 수행
매개 변수(1개) 생성자 수행
박스의 부피(매개 변수 1개) : 10
매개 변수(3개) 생성자 수행
매개 변수(2개) 생성자 수행
박스의 부피(매개 변수 2개) : 200
매개 변수(3개) 생성자 수행
박스의 부피(매개 변수 3개) : 6000
```

Section 4.

메소드

- 클래스의 핵심으로서 클래스의 기능을 나타낸다

[형식] 메소드

```
[public/private/protected][static/final/abstract/synchronized]
                                    반환값형 메소드 이름([매개 변수들])

{
   ..................
   지역 변수 및 메소드의 행위 기술
   ..................
}
```

static : 클래스 메소드
final : 종단 메소드
abstract : 추상 메소드
synchronized : 동기화 메소드

예제 9.5 Box8Test1.java

```
01: class Box8 {
02:     int width;
03:     int height;
04:     int depth;
05:     public Box8(int width, int height, int depth)
06:     {
07:         this.width = width;
08:         this.height = height;
09:         this.depth = depth;
10:     }
11:     int volume()
12:     {
13:         int vol = width * height * depth;
14:         return vol;
15:     }
16: }
```

메소드 volume()에서 부피를 계산하여 결과를 반환

4 메소드

```
17: public class Box8Test1 {
18:     public static void main(String args[]) {
19:         Box8 mybox1 = new Box8(10,20,30);
20:         //mybox1.width = 20;
21:         int vol1 = mybox1.volume();
22:         System.out.println("정수 박스의 부피 : " + vol1);
23:     }
24: }
```

접근 한정자를 사용하지 않았으므로, 값을 변경할 수 있다.

메소드를 호출하여 부피를 구한다.

실행 결과

```
정수 박스의 부피 : 6000
```

4.1 접근 한정자

- 메소드 선언 시 사용되는 접근 한정자는 멤버 변수와 같이 public, private, protected가 사용

- 예

```
01: public class Test1 {
02:     public int a;          <-- public으로 선언된 객체 변수
03:     int b;                 <-- 접근 한정자를 지정하지 않고 선언된 객체 변수
04:     private int c;         <-- private로 선언된 객체 변수
05:     public void method1() { }   <-- public으로 선언된 메소드
06:     viod method2() { }     <-- 접근 한정자를 지정하지 않고 선언된 메소드
07:     private void mothod3() { }  <-- private로 선언된 메소드
08: }
```

4.1 접근 한정자

- 같은 패키지에 속해 있는 클래스에서 사용하는 예

```
01: public class SamePackage {
02:     Test1 t1 = new Test1();   <-- 객체를 생성
03:     t1.a = 3;                 <-- 접근 가능
04:     t1.b = 5;                 <-- 접근 가능
05:     t1.c = 7;                 <-- 접근 불가능
06:     t1.mothod1();             <-- 접근 가능
07:     t1.mothod2();             <-- 접근 가능
08:     t1.mothod3();             <-- 접근 불가능
09: }
```

4.1 접근 한정자

- 다른 패키지에 속해 있는 클래스에서 사용하는 예

```
01: public class OtherPackage {
02:     Test1 t1 = new Test1();   <-- 객체를 생성
03:     t1.a = 3;                 <-- 접근 가능
04:     t1.b = 5;                 <-- 접근 불가능
05:     t1.c = 7;                 <-- 접근 불가능
06:     t1.mothod1();             <-- 접근 가능
07:     t1.mothod2();             <-- 접근 불가능
08:     t1.mothod3();             <-- 접근 불가능
09: }
```

4.1 접근 한정자

예제 9.6 Box9Test1.java

```
01: class Box9 {
02:     private int width;
03:     private int height;
04:     private int depth;
05:     private int vol;
06:     public Box9(int width, int height, int depth)
07:     {
08:         this.width = width;
09:         this.height = height;
10:         this.depth = depth;
11:         volume();
12:     }
13:     private void volume()
14:     {
15:         vol = width * height * depth;
16:     }
17:     public int getvolume() {
18:         return vol;
19:     }
20: }
```

객체의 모든 속성들을 private로 선언

생성자에서 volume() 메소드 호출

volume() 메소드를 private로 선언

부피를 단순하게 반환하는 메소드, public으로 선언

4.1 접근 한정자

```
21: public class Box9Test1 {
22:     public static void main(String args[]) {
23:         Box9 mybox1 = new Box9(10,20,30);
24:         // mybox1.width = 20;
25:         // int vol1 = mybox1.volume();
26:         System.out.println("정수 박스의 부피 : " + mybox1.getvolume());
27:     }
28: }
```

객체의 속성값을 변경하거나 메소드를 호출하면 오류 발생

부피를 읽어 오는 메소드를 호출하여 값을 출력

실행 결과

```
정수 박스의 부피 : 6000
```

4.2 클래스 메소드

- **클래스 변수와 같이 메소드에도 static을 붙여 클래스 메소드로 선언**
  - 클래스 변수와 같이 클래스 이름으로 접근
  - 객체를 생성하지 않아도 사용 가능한 함수 같은 메소드

```
01: Arrays.toString(a);   <-- Arrays 클래스의 toString() 메소드를 함수처럼 직접 사용
02: Arrays.sort(b);       <-- Arrays 클래스의 sort() 메소드(정렬 메소드)를 함수처럼 직접 사용
03: Integer.parseInt(args[0]);   <-- Integer 클래스의 parseInt() 메소드를 함수처럼 직접 사용
04: String.valueOf(number);      <-- String 클래스의 valueOf() 메소드를 함수처럼 직접 사용
```

4.2 클래스 메소드

- 클래스 메소드 내에서는 클래스 변수만 사용이 가능

```
01: class Box {
02:     int width;
03:     int height;
04:     int depth;
05:     long idNum;          <-- 일반 객체 변수로 정의
06:     static long boxID = 100;          <-- 클래스 변수로 정의
07:     static long getcurrentID() {      <-- 클래스 메소드 정의
08:         int count=1;                  <-- 지역 변수 정의. 사용 가능
09:         idNum = idNum+count;          <-- 객체 변수 사용 불가능. 오류 발생
10:         boxID = boxID+count;          <-- 클래스 변수와 지역 변수 사용 가능
11:         return boxID++;
12:     }
13: }
```

4.2 클래스 메소드

예제 9.7 Box10Test1.java

```
01: class Box10 {
02:     private int width;
03:     private int height;
04:     private int depth;
05:     private int vol;
06:     private long idNum;
07:     private static long boxID;
08:     public Box10(int width, int height, int depth)
09:     {
10:         this.width = width;
11:         this.height = height;
12:         this.depth = depth;
13:         idNum = ++boxID;
14:         volume();
15:     }
16:    private void volume()
17:    {
```

객체의 모든 속성들을 private로 선언

클래스 변수도 private로 선언 가능

생성자에서 idNum에 고유 번호를 부여

4.2 클래스 메소드

```
18:         vol = width * height * depth;
19:     }
20:     public String getvolume() {          ← 박스의 고유 번호와 부피를 반환
21:         return idNum +"번 박스의 부피 : "+ vol;
22:     }
23:     public static long getCurrentID() {  ← 현재의 박스 번호를 반환하는 클래스 메소드
24:         // return idNum;                 ← idNum을 사용하면 오류 발생
25:         return boxID;                    ← 클래스 변수 boxID 반환
26:     }
27: }
28: public class Box10Test1 {
29:     public static void main(String args[]) {
30:         Box10 mybox1;
31:         for(int i=1 ; i <= 5 ; i++) {
32:             mybox1 = new Box10(i,i+1,i+2);
33:             System.out.println(mybox1.getvolume());
34:         }
35:         System.out.println("마지막 생성된 박스 번호는 "+ Box10.getCurrentID()
    + "번입니다");                           ← 클래스 메소드를 이용하여 현재의 박스 번호 출력
36:         // System.out.println(Box10.boxID);
37:     }                                    클래스 변수를 private로 선언함으로써 오류 발생
38: }
```

실행 결과

```
1번 박스의 부피 : 6
2번 박스의 부피 : 24
3번 박스의 부피 : 60
4번 박스의 부피 : 120
5번 박스의 부피 : 210
마지막 생성된 박스 번호는 5번입니다
```

4.3 final, abstract, synchronized 메소드

final로 선언된 메소드

- 서브 클래스에서 오버라이딩overriding될 수 없음을 의미(10장 설명)

abstract로 선언된 메소드

- 추상 메소드로서 추상 클래스 내에서 선언될 수 있습니다. 추상 메소드는 선언 부분만 가지고 몸체 부분이 없는 메소드로서 하위 클래스에서 오바라이딩됩니다. 추상 메소드는 11장에서 설명합니다.

synchronized 메소드

- 스레드를 동기화할 수 있는 기법을 제공하기 위해 사용되는 메소드

4.4 메소드의 호출과 반환 값

- **메소드는 상호 호출될 수 있다**
  - 하나의 메소드에서 다른 메소드가 호출되면, 그 메소드는 수행이 중지되고 호출된 메소드로 제어가 넘어단다. 호출된 메소드의 수행이 완료되면 호출한 메소드는 중지된 시점에서 다시 실행된다
- **메소드 호출의 예**

```
01: class Sample3 {
02:     void methodA() {
03:         System.out.println("B 메소드 호출 전");
04:             methodB();  <-- methodB()를 호출
05:         System.out.println("B 메소드 호출 후");
06:     }
07:     void methodB() {
08:         System.out.println("C 메소드 호출 전");
09:             methodC();  <-- methodC()를 호출
10:         System.out.println("C 메소드 호출 후");
11:     }
12:     void methodC() {
13:         System.out.println("C 메소드 수행 완료");
14:     }
15: }
16: class SampleTest {
17:     public static void main(String args[]) {
18:          Sample3 s = new Sample3();
19:          s.methodA();
20:     }
21: }
```

실행 결과

```
B 메소드 호출 전
C 메소드 호출 전
C 메소드 수행 완료
C 메소드 호출 후
B 메소드 호출 후
```

4.4 메소드의 호출과 반환 값

- **메소드 선언부에서 반환값의 자료형이 지정되어야 한다**
  - 반환값의 자료형이 지정된 경우, 명시적으로 return 문에 의해 값이 반환 되어야 한다
  - 반환값은 기본 자료형 뿐만 아니라 참조 자료형도 반환 될 수 있다
- **반환 값이 없는 경우에는 void로 지정**
- **반환값의 예**

```
01: public int sum(int a, int b) {   <-- 반환되는 값의 형을 int로 지정
02:     nt c;
03:     c = a + b;
04:     return c; //                 <-- 정수값을 반환
05: }
```

```
01: public void calc(int x, int y) {   <-- 반환되는 값이 없다는 의미
02:         ..............
03:     if ( a < 0 ) return;           <-- 조건이 참이면 메소드의 실행을 종료
04:         ..............
05: }
```

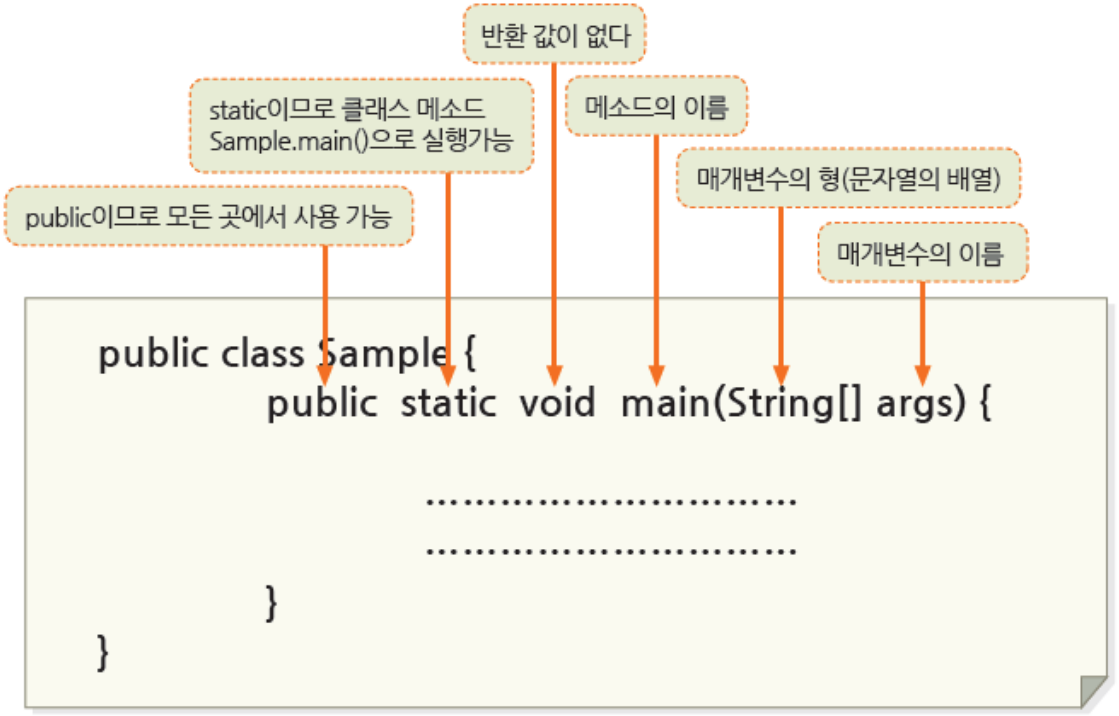
4.4 메소드의 호출과 반환 값

- **반환값의 예**

```
01: public Box volume_compute(Box instance_box) {   <-- 반환되는 값의 형을 Box로 지정
02:     ..................
03:     Box v_box = new Box();   <-- Box 객체를 생성하여 반환 값으로 사용
04:     v_box.width = instance_box.width;
05:     v_box.height = instance_box.height;
06:     v_box.depth = instance_box.depth;
07:     v_box.volume= v_box.width * v_box.height * v_box.depth;
08:     return v_box;   <-- Box 객체를 반환
09: }
```

4.5 main() 메소드

- main() 메소드는 특수한 메소드로서 자바 프로그램의 실행이 시작되는 첫 메소드를 의미



4.5 main() 메소드

- **클래스 메소드 main()의 의미**
  - main() 메소드는 클래스 메소드이므로 이 메소드에서 사용 가능한 속성 역시 클래스 속성만 사용할 수 있다.

- **클래스 메소드 main()의 의미 : 예**

```
01: public class Sample {
02:     int count=10;                          ← 객체 변수 선언
03:     static int num=20;                     ← 클래스 변수 선언
04:     public int sum(int x, int y) {         ← 메소드 선언
05:         return x+y;
06:     }
07:     static int mul(int x, int y) {         ← 클래스 메소드 선언
08:         return x*y;
09:     }
10:
11:     public static void main(String[] args) {
12:         int same;
13:         same = count;                      ← 오류 발생. 클래스 메소드는 클래스 변수만 사용 가능
14:         same = num;                        ← 사용 가능
15:         same = sum(5,5);                   ← 오류 발생. 클래스 메소드에서는 클래스 메소드만 사용 가능
16:         same = mul(5,5);                   ← 사용 가능
17:     }
18: }
```

4.5 main() 메소드

- 클래스 메소드 main()의 의미 : 객체를 생성하여 사용하는 예

```
01: public class Sample1 {
02:     int count=10;
03:     static int num=20;
04:     public int sum(int x, int y) {
05:         return x+y;
06:     }
07:     static int mul(int x, int y) {
08:         return x*y;
09:     }
10:
11:     public static void main(String[] args) {
12:         Sample1 s = new Sample1();
13:         int same = s.count;
14:         same = s.num; // Sample1.num도 가능
15:         same = s.sum(5, 5);
16:         same = s.mul(5, 5); // Sample1.mul(5,5)도 가능
17:     }
18: }
```

main() 메소드가 속한 클래스로부터 객체 생성

객체(객체의 속성)에 접근 가능

클래스 변수이므로 클래스 이름과 객체 이름으로 접근 가능

객체(객체의 메소드)에 접근 가능

클래스 메소드이므로 클래스 이름과 객체 이름으로 접근 가능

4.5 main() 메소드

main() 메소드의 매개 변수

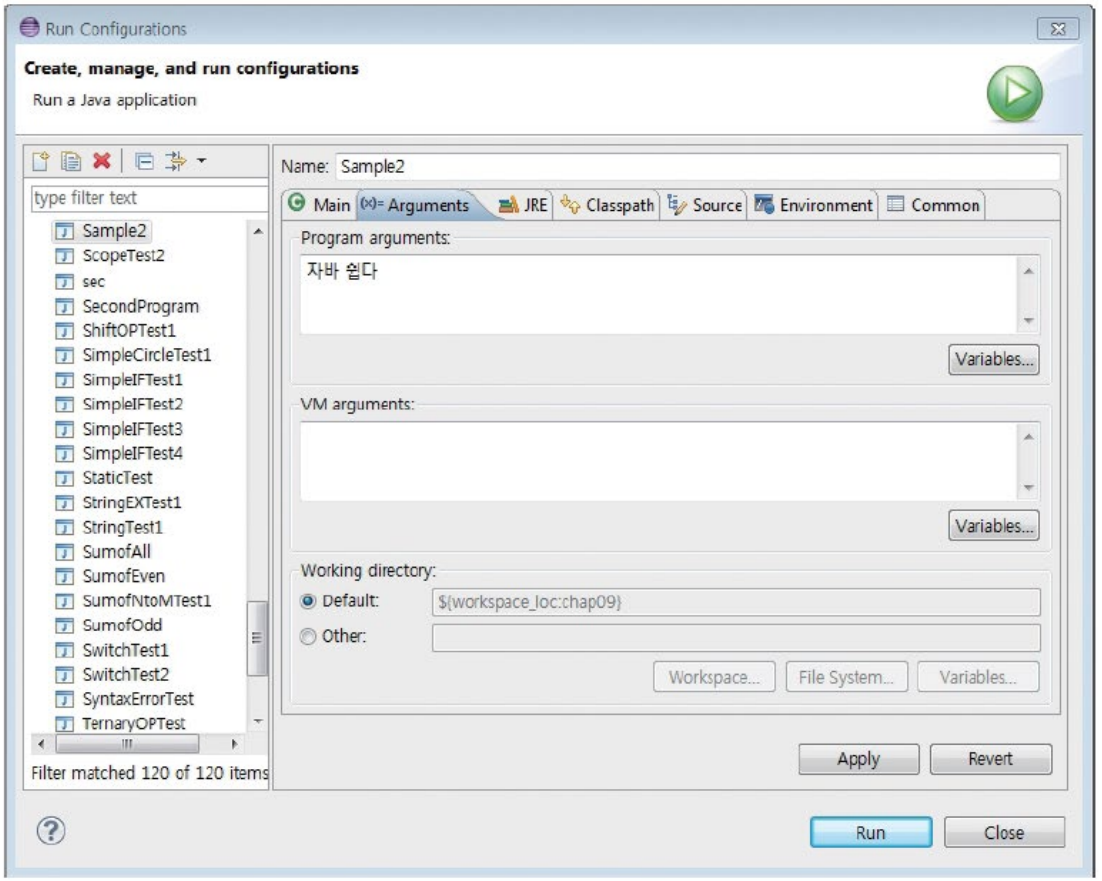
- main 메소드의 매개 변수는 문자열의 배열로 정의

```
01: public class Sample2 {
02:     public static void main(String[] args) {
03:         String s1 = args[0];<-------- 프로그램 실행 시 지정한 첫 번째 값이 s1에 저장
04:         String s2 = args[1];<-------- 프로그램 실행 시 지정한 두 번째 값이 s2에 저장
05:         System.out.println("첫 번째 매개 변수값 : "+ s1);
06:         System.out.println("두 번째 매개 변수값 : "+ s2);
07:     }
08: }
```

4.5 main() 메소드

이클립스에서 실행 시 매개 변수를 입력하는 방법

이클립스의 "Run" 메뉴에서 "Run Configuration"을 선택하면 창이 나타납니다. 나타난 창에서 "Arguments" 탭을 선택하여 다음과 같이 입력합니다. 처음 입력되는 항목이 args[0]이고 각 항목은 공백(space)으로 구분합니다.



4.5 main() 메소드

예제 9.8 SumTwoNumber.java

```
01: public class SumTwoNumber {
02:     public static void main(String args[]) {
03:         System.out.println("매개 변수로 받은 두 수의 합은 : "
    +(args[0]+args[1])); ◄---------- 매개 변수의 값을 직접 더하여 출력
04:         int a = Integer.parseInt(args[0]); ◄
05:         int b = Integer.parseInt(args[1]); ◄ ------ 매개 변수의 값(문자열)을 정수로 변환
06:         System.out.println("매개 변수로 받은 두 수의 합은 : " +(a+b));
07:     }
08: }
```

실행 결과 입력값을 10과 20으로 지정

```
매개 변수로 받은 두 수의 합은 : 1020
매개 변수로 받은 두 수의 합은 : 30
```

Section 5.

메소드 오버로딩

● 생성자의 오버로딩과 같이 메소드도 오버로딩 할 수 있다.

- 같은 클래스에 같은 이름의 메소드를 중첩하여 사용
- 메소드 매개 변수의 개수와 형이 달라야 한다

예제 9.9 Box11Test1.java

```
01: class Box11 {
02:     private int ivol;
03:     private double dvol;
04:     public Box11(int w, int h, int d)
05:     {
06:         volume(w,h,d); <---------- 오버로딩된 메소드 volume()을 호출함
07:     }
08:     public Box11(double w, double h, double d)
09:     {
10:         volume(w,h,d); <---------- 오버로딩된 메소드 volume()을 호출함
11:     }
12:     private void volume(int w, int h, int d) <----------
13:     {
14:         ivol = w * h * d;
15:     }
16:    private void volume(double w, double h, double d)
```

5 메소드 오버로딩

```
17:     {
18:         dvol = w * h * d;
19:     }
20:     public int get_ivol() {
21:         return ivol;
22:     }
23:     public double get_dvol() {
24:         return dvol;
25:     }
26: }
27: public class Box11Test1 {
28:     public static void main(String args[]) {
29:         Box11 mybox1 = new Box11(10,20,30);
30:         System.out.println("박스의 부피(정수 매개 변수) : " +
    mybox1.get_ivol());
31:         mybox1 = new Box11(10.5, 20.5, 30.5);
32:         System.out.println("박스의 부피(실수 매개 변수) : " +
    mybox1.get_dvol());
33:         mybox1 = new Box11(10,20,30.5);
34:         System.out.println("박스의 부피(정수와 실수 혼합) : " +
    mybox1.get_dvol());
35:     }
36: }
```

동일한 이름의 메소드가 오버로딩으로 선언

실행 결과

```
박스의 부피(정수 매개 변수) : 6000
박스의 부피(실수 매개 변수) : 6565.125
박스의 부피(정수와 실수 혼합) : 6100.0
```

- 예제 9.9를 변형하여 객체의 배열을 사용한 예

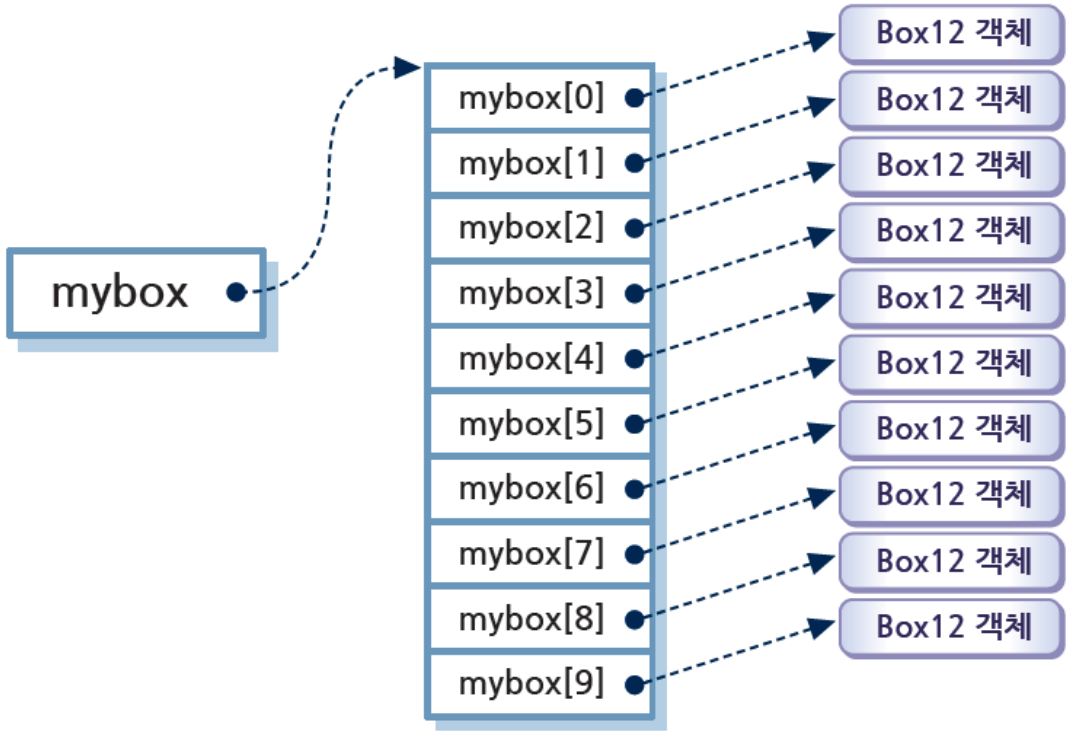


그림 9-3 객체배열

5 메소드 오버로딩

예제 9.10 Box12Test1.java

```
01: class Box12 {
02:     private int ivol;
03:     public Box12(int w, int h, int d)
04:     {
05:     ivol = w * h * d;
06:     }
07:     public String get_ivol() {
08:         return " 박스의 부피는 " + ivol;
09:     }
10: }
11:
12: public class Box12Test1 {
13:     public static void main(String args[]) {
14:         Box12 mybox[] = new Box12[10];
15:         for(int i = 0 ; i < mybox.length ; i++) {
16:             mybox[i] = new Box12(i*10, i*20, i*30);
17:         }
18:         for(Box12 mb : mybox) {
19:             System.out.println(mb.get_ivol());
20:         }
21:     }
22: }
```

14: 10개의 요소를 가진 Box12 형의 배열 선언과 생성

15~17: 배열의 각 요소에 Box12형의 객체를 생성

18~20: 배열의 각 요소의 메소드를 호출하여 결과 출력

실행 결과

```
박스의 부피는 0
박스의 부피는 6000
박스의 부피는 48000
박스의 부피는 162000
박스의 부피는 384000
박스의 부피는 750000
박스의 부피는 1296000
박스의 부피는 2058000
박스의 부피는 3072000
박스의 부피는 4374000
```

5 메소드 오버로딩

예제 9.11 OverloadTest1.java

```
01: class Overload{
02:     public void calc(){
03:         System.out.println("매개 변수가 없습니다.");
04:     }
05:     public void calc(int width){
06:         System.out.println("정사각형의 넓이 : " + width * width);
07:     }
08:     public void calc(int width, int height){
09:         System.out.println("직사각형의 넓이 : " + width * height);
10:     }
11:     public void calc(int width, int height, int depth){
12:         System.out.println("직육면체의 부피 : " + width * height * depth);
13:     }
14: }
15:
16: public class OverloadTest1 {
17:     public static void main(String args[]){
18:         Overload ol = new Overload();
19:         int input[] = new int[args.length];
20:         for(int i=0; i<args.length; i++)
21:             input[i] = Integer.parseInt(args[i]);
22:         switch (args.length){
23:         case 0:
24:             ol.calc();
25:             break;
26:         case 1:
27:             ol.calc(input[0]);
28:             break;
29:         case 2:
30:             ol.calc(input[0], input[1]);
31:             break;
32:         case 3:
33:             ol.calc(input[0], input[1], input[2]);
34:             break;
35:         default:
36:             System.out.println("인수의 개수가 많습니다.");
37:         }
38:     }
39: }
```

오버로딩된 메소드

객체를 생성

입력한 매개 변수의 크기와 같은 정수 배열 생성

실행 시 입력한 문자열 배열의 요소를 정수로 바꾸어 정수 배열에 저장

정수 배열의 길이를 기준으로 switch문 수행

배열의 길이에 따라 메소드 호출

실행 결과 값을 달리하여 여러 번 실행

```
– 매개 변수 없이 실행
매개 변수가 없습니다.
– 매개 변수로 5를 지정
정사각형의 넓이 : 25
– 매개 변수로 5와 10을 지정
직사각형의 넓이 : 50
– 매개 변수로 5와 10, 15를 지정
직육면체의 부피 : 750
– 매개 변수로 5와 10, 15, 20을 지정
인수의 개수가 많습니다.
```

Section 6.

메소드에 값 전달 기법

- **메소드 호출 시 매개 변수로 지정되는 실매개 변수**
  - 기본 자료형과 참조 자료형
- **자바의 매개 변수 전달 기법은 call by value(값을 복사) 기법을 사용**
  - 값-전달 기법은 메소드 호출 시 실매개 변수의 값을 형식 매개 변수에 복사해 주는 방식

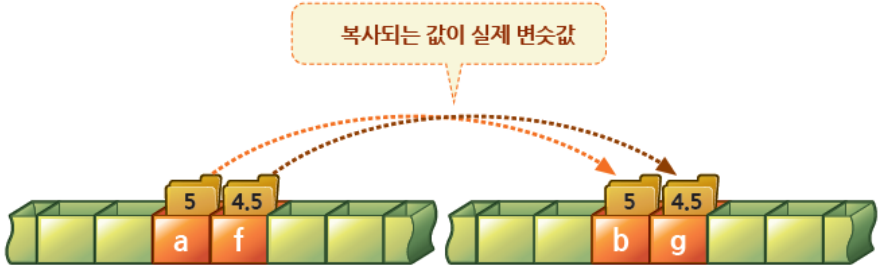


그림 9-2 실매개 변수와 형식 매개 변수로 기본 자료형이 사용되는 경우

5 메소드 오버로딩

예제 9.12 ArgumentTest1.java

```
01: class Argument {
02:     public void change(int i, int j[]) {
03:         i = 20;                          ← 기본 자료형 변수 i의 값을 변경
04:         j[3] = 400;                      ← 참조 자료형 배열의 4번째 요소의 값을 변경
05:     }
06:     public void display(int i, int j[]) {
07:         System.out.println("객체 변수 i의 값 : " + i);
08:         System.out.print("배열의 값 : ");
09:         for(int value : j)
10:             System.out.print(value + " ");
11:         System.out.println();
12:     }                                    ← 변수 i의 값과 배열을 출력
13: }
14: class ArgumentTest1 {
15:     public static void main(String args[]) {
16:         Argument d = new Argument();
17:         int a = 10;                      ← 기본 자료형 변수 a를 생성
18:         int b[]  = { 1, 2, 3, 4 };       ← 참조 자료형 배열 b를 생성
19:         System.out.println("첫 번째 display() 메소드 호출");
20:         d.display(a, b);                 ← display() 메소드를 호출하여 출력
21:         d.change(a, b);                  ← change() 메소드를 이용하여 값을 변경
22:         System.out.println("==========================");
23:         System.out.println("값을 변환한 다음 두 번째 display() 호출");
24:         d.display(a, b);                 ← display() 메소드를 호출하여 출력
25:     }
26: }
```

실행 결과

```
첫 번째 display() 메소드 호출
객체 변수 i의 값 : 10
배열의 값 : 1 2 3 4
==========================
값을 변환한 다음 두 번째 display() 호출
객체 변수 i의 값 : 10
배열의 값 : 1 2 3 400
```

Thank You!