

3부 클래스와 객체

- 12장 메서드 살피기

최문환



12장 메소드 살피기

1. 다형성의 의미



1. 다형성의 의미

타원
draw() {타원을 그린다}

사각형
draw() {사각형을 그린다}

삼각형
draw() {삼각형을 그린다}

1.1 메소드의 오버로딩

메서드 오버로딩이란 한 클래스 내에 같은 이름의 메서드를 여러 개 정의하는 것을 뜻한다.

메소드를 구분하는 시그너처

메소드를 구분하는 시그너처 중에서 메소드의 이름이 같아야 메소드의 오버로딩이므로 다음 세 가지 조건 중 하나를 만족해야 한다.

- 1 메소드의 전달인자 자료형이 달라야 한다.
2. 메소드의 전달인자 개수가 달라야 한다.
3. 메서드의 전달인자 순서를 다르게 한다.

<예제> 메소드의 오버로딩 살펴보기

```
001:public class MethodTest01{
002: public static void main(String[] args) {
003:     //(1) 논리값 : true, false
004:     System.out.println(true);
005:     //(2) 문자 : 단일 따옴표로 묶어줌
006:     System.out.println('A');
007:     //(3) 정수 : 소수점이 없는 수
008:     System.out.println(128);
009:     //(4) 실수 : 소수점이 있는 수
010:     System.out.println(3.5);
011:     //(5) 문자열 : 이중 따옴표로 묶어줌
012:     System.out.println("Hello");
013: }
014:}
```

<예제> 전달인자 자료형이 다른 메소드 오버로딩

```
001: public class MethodTest02{
002:     //int형 데이터에 대해서 절대값을 구하는 메소드 정의
003:     int abs(int num){
004:         if(num<0)
005:             num=-num;
006:         return num;
007:     }
008:     //long형 데이터에 대해서 절대값을 구하는 메소드 정의
009:     long abs(long num){
010:         if(num<0)
011:             num=-num;
012:         return num;
013:     }
014:     //double 데이터에 대해서 절대값을 구하는 메소드 정의
015:     double abs(double num){
016:         if(num<0)
017:             num=-num;
018:         return num;
019:     }
```

<예제> 전달인자 자료형이 다른 메소드 오버로딩

```
020:
021: public static void main(String[] args) {
022:     MethodTest02 mt=new MethodTest02();
023:
024:     //전달인자가 int형이므로 03:의 int형 데이터에 대해서 절대값을 구하는 메소드 호출
025:     int var01=-10, var02;
026:     var02=mt.abs(var01);
027:     System.out.println(var01 + "의 절대값은-> " + var02);
028:
029:     //전달인자가 long형이므로 09:의 long형 데이터에 대해서 절대값을 구하는 메소드호출
030:     long var03=-20L, var04;
031:     var04=mt.abs(var03);
032:     System.out.println(var03 + "의 절대값은-> " + var04);
033:
034:     //전달인자가double형이므로 15:의 double형에 대해서 절대값을 구하는 메소드 호출
035:     double var05=-3.4, var06;
036:     var06=mt.abs(var05);
037:     System.out.println(var05 + "의 절대값은-> " + var06);
038: }
039: }
```

<예제> 전달인자의 개수가 다른 메소드 오버로딩

```
001:public class MethodTest03{
002: //정수형 데이터 3개를 형식매개변수로 갖는 prn 메소드 정의
003: void prn(int a, int b, int c) {
004:   System.out.println(a + "\t" + b + "\t" + c);
005: }
006: //정수형 데이터 2개를 형식매개변수로 갖는 prn 메소드 정의
007: void prn(int a, int b) {
008:   System.out.println(a + "\t" + b);
009: }
010: //정수형 데이터 1개를 형식매개변수로 갖는 prn 메소드 정의
011: void prn(int a){
012:   System.out.println(a);
013: }
014: public static void main(String[] args){
015:   MethodTest03 mt=new MethodTest03();
016:   mt.prn(10, 20, 30); //정수형 데이터 3개를 실매개변수로 지정
017:   mt.prn(40, 50);    //정수형 데이터 2개를 실매개변수로 지정
018:   mt.prn(60);        //정수형 데이터 1개를 실매개변수로 지정
019: }
020:}
```


1.2 메소드 오버로딩에 포함되지 않는 메소드 구성요소

1. 접근 지정자
2. 리턴값

<예제> 리턴형이 다른 메소드-[파일 이름 : MethodTest03.java]

```
001: class MethodTestA{
002:   void prn(int a){
003:     System.out.println(a);
004:   }
005:   int prn(int a){
006:     return a;
007:   } }
      public class MethodTest03{
0008:   public static void main(String[] args){
0009:     MethodTestA mt=new MethodTestA();
0010:     mt.prn(10);      //정수형 데이터 1개를 실매개변수로 지정
0011:     int k;
0012:     k=mt.prn(10);
0013:   }
0014: }
```

1.3 Varargs(가변인자: Variable Argument List)

<예제> Varargs를 이용하여 다양한 형태로 메소드 호출하기

```
001:public class MethodTest04{
002: void prn(int ... num){    //int형 데이터를 출력하는 메소드의 정의
003:   for(int i=0; i<num.length; i++) //전달인자의 개수만큼 반복하면서
004:     System.out.print(num[i]+"Wt"); //배열 형태로 출력한다.
005:   System.out.println();
006: }
007:
008: public static void main(String[] args)
009: {
010:   MethodTest04 mt=new MethodTest04();
011:   mt.prn(10, 20, 30); //개수에 상관없이 메소드를 호출할 수 있다.
012:   mt.prn(40, 50);
013:   mt.prn(60);
014: }
015:}
```

2. 기본 자료형과 레퍼런스 형의 차이점

<예제> 레퍼런스 변수만 선언하기-[파일 이름 : MethodTest05.java]

```
001:class MyDate{ //클래스의 초기값을 지정함
002: int year=2023;
003: int month=4;
004: int day=1;
005:}
006:public class MethodTest05 {
007: public static void main(String[] args) {
008:
009:     MyDate d; //1. 레퍼런스 변수
010:     System.out.println(d.year+ "/" +d.month+ "/" +d.day);
011:
012:     new MyDate(); //2. 객체 생성되었지만 사용되지 못함
013:
014:     d=new MyDate();
015:     System.out.println(d.year+ "/" +d.month+ "/" +d.day);
016: }
017:}
```

2. 기본 자료형과 레퍼런스 형의 차이점

<예제> 기본 자료형과 레퍼런스 형의 차이점

```
001: class MyDate{
002:   int year=2022;
003:   int month=4;
004:   int day=1;
005: }
006: public class MethodTest06{
007:   public static void main(String[] args) {
008:     //기본 자료형 중에서 int형으로 선언한 두 개의 변수
009:     int x=7;
010:     int y=x; //7을 저장하고 있는 변수 x의 값을 변수 y에 복사
011:     //두 개는 MyDate 형으로 선언된 레퍼런스 변수
012:     MyDate d=new MyDate(); //객체 생성
013:     MyDate t=d;
```

2. 기본 자료형과 레퍼런스 형의 차이점

<예제> 기본 자료형과 레퍼런스 형의 차이점

```
014:
015: System.out.println( "x->" +x+ " y->" +y);
016: System.out.println(d.year+ "/" +d.month+ "/" +d.day); //2022/4/1
017: System.out.println(t.year+ "/" +t.month+ "/" +t.day); //2022/4/1
018:
019: //변수 x와 y는 독립된 두 개의 변수이므로
020: y=10;//변수 y의 값을 변경시켜도 x의 값에 영향을 주지 못함
021: System.out.println( "x->" +x+ " y->" +y);
022:
023: //레퍼런스 변수 t로 접근해서 MyDate 객체의 값을 변경하면
024: t.year=2023; t.month=7; t.day=19;
025: //레퍼런스 변수 d로 접근했을 때에도 변경되어진 값이 출력
026: System.out.println(d.year+ "/" +d.month+ "/" +d.day); //2023/7/19
027: System.out.println(t.year+ "/" +t.month+ "/" +t.day); //2023/7/19
028: }
029: }
```

2. 기본 자료형과 레퍼런스 형의 차이점

<예제> 레퍼런스 형 변수가 서로 다른 객체를 가리키도록 하기

```
001: class MyDate{
002:   int year=2022;
003:   int month=4;
004:   int day=1;
005: }
006: public class MethodTest07{
007:   public static void main(String[] args) {
008:     MyDate d=new MyDate(); //객체 생성
009:     MyDate t=d;//t가 이미 선언된 d와 동일한 객체를 참조함
010:     System.out.println(d.year+ "/" +d.month+ "/" +d.day); //2022/4/1
011:     System.out.println(t.year+ "/" +t.month+ "/" +t.day); //2022/4/1
012:     t=new MyDate();
013:     t.year=2023; t.month=7; t.day=19;
014:     System.out.println(d.year+ "/" +d.month+ "/" +d.day); //2022/4/1
015:     System.out.println(t.year+ "/" +t.month+ "/" +t.day); //2023/7/19
016:   }
No.14 017: }
```

2.1 값 전달 방식과 레퍼런스 전달 방식

<예제> 값에 의한 호출 방식 예제

```
001: class ValueMethod{
002: void changeInt(int y){
003:   y=10;
004: }
005: }
006: public class MethodTest08 {
007: public static void main(String[] args) {
008:   ValueMethod vm=new ValueMethod();
009:   int x=7;
010:   System.out.println( " 함수 호출 전 x->" +x);
014:   vm.changeInt(x);
015:   System.out.println( " 함수 호출 후 x->" +x);
016: }
017: }
```

No.15



<예제> 레퍼런스 의한 호출 방식 예제

```
001: class MyDate{
002:   int year=2022;
003:   int month=4;
004:   int day=1;
005: }
006: class RefMethod{
007:   void changeDate(MyDate t){
008:     t.year=2023; t.month=4; t.day=1;
009:   }
010: }
011: public class MethodTest09 {
012:   public static void main(String[] args) {
013:     RefMethod rm=new RefMethod();
014:     MyDate d=new MyDate();
015:     System.out.println(" 함수 호출 전 d->" + d.year+ "/" +d.month+ "/" +d.day);
016:     rm.changeDate(d);
017:     System.out.println(" 함수 호출 후 d->" + d.year+ "/" +d.month+ "/" +d.day);
018:   }
019: }
```


2.2 레퍼런스형 변수의 초기화와 null값

<예제> 초기화하지 않은 레퍼런스형 변수 [파일이름:MethodTestB.java]

```
001:class MyDate{
002:  int year=2023;
003:  int month=3;
004:  int day=10;
005:}
006:class MethodTestB{
007:  public static void main(String[] args){
008:    MyDate d;
009:    System.out.println(d.year+ "/" +d.month+ "/" +d.day);
010:  }
011:}
```

2.2 레퍼런스형 변수의 초기화와 null값

<예제> null 값을 갖는 레퍼런스형 변수 [파일 이름:MethodTestB.java]

```
001:class MyDate{
002:  int year=2023;
003:  int month=3;
004:  int day=10;
005:}
006:class MethodTestC{
007:  public static void main(String[] args){
008:    MyDate d=null;
009:    System.out.println(d.year+ "/" +d.month+ "/" +d.day);
010:    System.out.println("정상 종료");
011:  }
012:}
```

<문제>

1. p() 메소드를 오버로딩하여 두 int 변수에 대해서 두 double에 대해서 최대 값을 구하는 메소드를 정의하시오. (Ex12_01.java)

<문제>

2. 속성으로 메모리 용량과 회사명을 저장하는 변수를 갖는 Mp3 클래스를 설계하고 다음과 같은 결과가 나오도록 객체 생성 후 메소드를 호출하시오.(Ex12_02.java)

회사명 : 갑
을회사
메모리 용
량 : 8G

Mp3
-comp : String -size : int
+getComp() : String +getSize() : int +setComp(new_comp : String) +setSize(new_size : int)