제 15 강 : 객체지향 - 클래스의 개념 2

※ 학습목표

✓ Call by Value와 Call by Reference의 차이를 설명할 수 있다.

✓ 메서드 오버로딩에 대해서 설명할 수 있다.

✓ 생성자 오버로딩에 대해서 설명할 수 있다.

✓ this와 this()를 활용할 수 있다.

1. 인자값 전달 방시

① Call by Value : 값 호출

✓ 이는 이전 강의 예제 MethodExTwo.java와 같이 메소드를 호출 시 기본 자료형의 값을 인자로 전달하는 방식을 의미한다.

[실습]

```
 1  package tommy.java.exam01;
 2
 3  public class ValueParameter {
 4          public int increase(int n) {
 5                  ++n;
 6                  return n;
 7          }
 8
 9          public static void main(String[] args) {
10                  int var1 = 100;
11                  ValueParameter vp = new ValueParameter();
12                  int var2 = vp.increase(var1);
13                  System.out.println("var1 : " + var1 + ", var2 : " + var2);
14          }
15  }
```

② Call by Reference : 참조 호출

✓ 메소드 호출 시 전달하려는 인자를 참조(객체) 자료형을 사용할 경우를 의미한다. 여기에는 기본 자료형이 아닌 일반 객체 또는 배열들이 여기에 속한다.

[실습]

```
1  package tommy.java.exam02;
2
3  public class ReferenceParameter {
4          public void increase(int[] n) {
5                  for (int i = 0; i < n.length; i++)
6                          n[i]++;
7          }
8
9          public static void main(String[] args) {
10                 int[] ref1 = { 100, 800, 1000 };
11                 ReferenceParameter rp = new ReferenceParameter();
12                 rp.increase(ref1);
13                 for (int i = 0; i < ref1.length; i++)
14                         System.out.println("ref1[" + i + "] : " + ref1[i]);
15         }
16 }
```

③ Variable Arguments

✓ JDK5.0에서 새롭게 추가된 기능

✓ 메소드 정의 시 통일된 인자의 자료형에 '…' 라고 명시하므로 이를 통해 메소드를 수행하는데 필요한 인자의 수를 유연하게 구현할 수 있다.

✓ 내부적으로 배열화 작업을 자동적으로 해 주기 때문!

[실습]

```
1  package tommy.java.exam03;
2
3  public class VariableEx {
4          public void argTest(String... n) {
5                  for (int i = 0; i < n.length; i++)
6                          System.out.println("n[" + i + "]:" + n[i]);
7                  System.out.println("--------------------------");
8          }
9
10         public static void main(String[] args) {
11                 VariableEx ve = new VariableEx();
12                 ve.argTest("Varargs", "Test");
13                 ve.argTest("100", "600", "900", "1000");
14         }
15 }
```

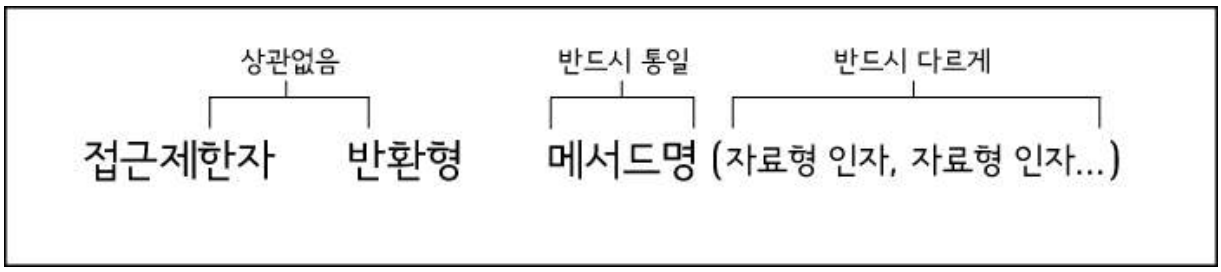
2, 메서드 오버로딩

✓ 하나의 클래스에서 같은 이름을 가진 메서드가 여러 개 정의되는 것을 말함

✓ 같은 이름의 메서드에 인자가 다름

✓ 인자가 다르다는 것은 개수가 다르거나, 자료형이 다르거나, 인수의 순서가 다른 것

✓ 같은 목적으로 비슷한 동작을 수행하는 메서드들을 모아 이름을 같게 만들어 일관성을 유지



[실습] 메서드 비 오버로딩 예제

```
1  package tommy.java.exam04;
2
3  public class OverloadingEx1 {
4          public void intLength(int a) {
5                  String s = String.valueOf(a);
6                  System.out.println("입력한 값의 길이 : " + s.length());
7          }
8
9          public void floatLength(float f) {
10                 String s = String.valueOf(f);
11                 System.out.println("입력한 값의 길이 : " + s.length());
12         }
13
14         public void stringLength(String str) {
15                 System.out.println("입력한 값의 길이 : " + str.length());
16         }
17
18         public static void main(String[] args) {
19                 OverloadingEx1 oe1 = new OverloadingEx1();
20                 oe1.intLength(1000);
21                 oe1.floatLength(3.14f);
22                 oe1.stringLength("10000");
23         }
24 }
```

[실습] 메서드 오버로딩 예제

```
1  package tommy.java.exam05;
2
3  public class OverloadingEx2 {
4          public void getLength(int n) {
5                  String s = String.valueOf(n);
6                  getLength(s);
7          }
8
9          void getLength(float n) {
10                 String s = String.valueOf(n);
11                 getLength(s);
12         }
13
14         private int getLength(String str) {
15                 System.out.println("입력한 값의 길이 : " + str.length());
16                 return 0;
17         }
18
19         public static void main(String[] args) {
20                 OverloadingEx2 oe2 = new OverloadingEx2();
21                 oe2.getLength(1000);
22                 oe2.getLength(3.14f);
23                 oe2.getLength("10000");
24         }
25 }
```

3. 생성자

✓ 메모리 내에 객체가 생성될 때 호출되어 객체의 구조를 인식하게 하고 생성되는 멤버변수들을 초기화하는 데 목적을 둠

✓ 생성자명은 클래스명과 같아야 하고, return type를 정의하지 말아야 함

✓ 프로그래머가 어떠한 생성자도 정의하지 않았을 경우 컴파일러가 default 생성자를 자동으로 정의해 줌(default 생성자) : 인자가 없는 생성자

① 생성자 접근제한의 의미

✓ 생성자의 접근제한을 둘 경우 해당 객체를 생성할 수 있는 접근권한을 가짐

✓ 클래스의 접근제한이 public으로 정의 되어도 생성자를 private로 정의 하면 클래스 내부에서만 접근 가능 하다.

✓ 만약 protected로 정의되는 클래스는 상속관계의 객체들만 생성할 수 있음

② 생성자의 구성

```
[접근제한] [생성자명](자료형 인자1, 자료형 인자2,…){
                수행문1;
                수행문2;
                … ;
}
```

③ 생성자의 특징

✓ 클래스 명과 똑같다.

✓ 반환형 void를 명시 할 수 없다.

✓ 클래스 내부에 생성자가 없을 때는 컴파일러가 default 생성자를 만들어 놓는다.

✓ 하나의 클래스에는 인자의 수가 다르거나 인자의 자료형이 다른 생성자들이 여러 개 있을 수 있다.(생성자 오버로딩)

✓ 생성자의 첫 번째 라인에서 this(인자) 생성자를 사용해서 다른 생성자 하나를 호출 가능하다.

④ 생성자 오버로딩

✓ 생성자의 Overloading은 객체를 생성할 수 있는 방법의 수를 제공하는 것과 같다

✓ 메소드 오버로딩법과 문법적으로 다를 것이 없어 각 생성자의 구분 또한 인자로 구별함을 잊지 말자!

[실습] 생성자를 변경해 가며 실행 해 보자.

```
 1  package tommy.java.exam06;
 2
 3  class MyClass {
 4          private String name;
 5          private int age;
 6
 7          public MyClass() {
 8                  name = “무명“;
 9          }
10
11          public MyClass(String n) {
12                  name = n;
13          }
14
```

```
15          public MyClass(int a, String n) {
16                  age = a;
17                  name = n;
18          }
19
20          public MyClass(String n, int a) {
21                  age = a;
22                  name = n;
23          }
24
25          public String getName() {
26                  return name;
27          }
28
29          public int getAge() {
30                  return age;
31          }
32  }
33
34  public class MyClassEx {
35          public static void main(String[] args) {
36                  MyClass mc1 = new MyClass();
37                  MyClass mc2 = new MyClass("아라치");
38                  MyClass mc3 = new MyClass("마루치", 46);
39                  MyClass mc4 = new MyClass(23, "오자바");
40                  System.out.println(mc1.getName() + "," + mc1.getAge());
41                  System.out.println(mc2.getName() + "," + mc2.getAge());
42                  System.out.println(mc3.getName() + "," + mc3.getAge());
43                  System.out.println(mc4.getName() + "," + mc4.getAge());
44          }
45  }
```

⑤ this와 this()

✓ this란 특정 객체 내에서 자신이 생성되었을 때의 주소 값 변수

✓ 객체의 주소는 생성 전까지는 모르기 때문에 객체 생성 후 자신의 주소로 대치됨

✓ this()는 현재 객체의 생성자를 의미함

✓ 생성자 안에서 오버로딩 된 다른 생성자를 호출할 경우에 this()라는 키워드로 호출함

[실습]

```
1   package tommy.java.exam07;
```

```
2
3  class ThisEx {
4          String name, jumin, tel;
5
6          public ThisEx() {
7                  this.name = “Guest“;
8                  this.jumin = “000000-0000000“;
9                  tel = “000-0000-0000“;
10         }
11
12         public ThisEx(String name) {
13                 this();
14                 this.name = name;
15         }
16
17         public ThisEx(String name, String jumin) {
18                 this(name);
19                 this.jumin = jumin;
20         }
21
22         public ThisEx(String name, String jumin, String tel) {
23                 this(name, jumin);
24                 this.tel = tel;
25         }
26
27         public String getJumin() {
28                 return jumin;
29         }
30
31         public String getName() {
32                 return name;
33         }
34
35         public String getTel() {
36                 return tel;
37         }
38 }
39
40 public class ThisExOne {
41         public static void main(String[] ar) {
42                 ThisEx ref = new ThisEx();
43                 System.out.println(“Name : “ + ref.getName());
44                 System.out.println(“TEL : “ + ref.getTel());
45                 System.out.println(“Jumin : “ + ref.getJumin());
46         }
47 }
```