생성자

#01. 생성자의 이해

생성자는 객체가 생성될 때 자동으로 실행되는 특수한 형태의 메서드이다.

객체가 생성될 때 멤버변수의 값을 초기화(가장 처음에 값을 할당하는 행위)하기 위해 사용한다.

1) 생성자 규칙

- 생성자는 클래스의 이름과 동일해야 한다.
- 리턴형을 명시하지 않는다.
- 필요하다면 파라미터를 정의할 수 있다.
- 모든 클래스는 반드시 하나 이상의 생성자를 포함해야 한다.

```
class Foo {
    /** 생성자 */
    Foo() {

    }
}
```

2) 기본 생성자

모든 클래스는 반드시 하나 이상의 생성자를 포함해야 한다. (하지만 지금까지의 예제들은 생성자를 포함하지 않고 있다.)

만약 소스코드 상에 생성자가 정의되지 않았다면 자바 컴파일러는 다음과 같이 파라미터와 구현부({})가 비어 있는 생성자가 있다고 간주한다.

이를 **기본 생성자**라고 한다.

```
class Foo {
  Foo() {  // <-- 실제로는 코딩되어 있지 않지만 컴파일러가 이런 생성자가 있다고 간주한다.
    // 생성자는 있지만, 내용이 비어 있다.
  }
}
```

기본생성자 라고함

3) 파라미터를 갖는 생성자

멤버변수 초기화

생성자도 메서드의 한 종류이므로 필요하다면 파라미터를 함께 정의하는 것이 가능하다.

생성자 파라미터 정의하기

일반 메서드와 같이 필요한 파라미터를 괄호(())안에 정의한다.

```
class 클래스이름 {
    클래스이름(int a, int b) {
        ...
    }
}
```

Ex01_SimpleConstructor.java

생성자가 파라미터를 갖는 경우의 객체 생성

new 키워드를 사용해 객체를 생성할 때 생성자 파라미터를 함께 전달해야 한다.

```
                                    생성자 호출
클래스이름 객체 = new 클래스이름(a에_대한_값, b에_대한_값);
```

생성자 파라미터를 사용하는 이유

생성자 파라미터는 객체가 생성될 때 전달받는 파라미터를 클래스 내의 맴버변수에 복사하는 것으로 객체의 초기화를 외부적인 요인에 의해 처리하기하고자 할 때 사용한다.

4) 생성자를 사용하는 경우의 이점

클래스를 작성하는 단계에서는 생성자를 포함하는 것이 소스코드가 더 길어진다. 하지만 전체적으로 봤을 때 생성자를 작성하는 것이 코드 효율성에 더 유리하다.

생성자가 없는 클래스의 예시

클래스 자체는 매우 간결하다.

```
class Student {
    String name = "";
    int level = 0;
    int age = 0;
}
```

이 클래스를 활용하는 과정에서는 소스코드가 더 길어지게 된다. 예를 들어 100개의 객체를 생성하고 각 객체의 멤버변수를 모두 변경해야 한다면 아래와 같이 총 400라인의 코드가 필요하게 된다.

```
Student s1 = new Student();
s1.name = "이름1";
s1.level = 1;
s1.age = 1;
...
Student s100 = new Student();
s100.name = "이름100";
```

```
s100.level = 100;
s100.age = 100;
```

생성자가 있는 클래스의 예시

```
class Student {
    String name;
    int level;
    int age;

    // 생성자 --> 주로 멤버변수에 할당할 값들을 파라미터로 정의
    Student(String n, int l, int a) {
        System.out.println("----- " + n + "이의 생성자가 실행되었습니다. -----");
        name = n;
        level = l;
        age = a;
    }
}
```

위의 클래스는 객체가 포함해야 할 멤버변수의 값을 객체 생성과 동시에 전달할 수 있기 때문에 각 객체당 1라인으로 모든 처리가 종료될 수 있다. 그러므로 객체를 생성하고 초기화 하는데 총 400라인에서 100라인으로 줄어들게 된다.

```
Student s1 = new Student("이름1", 1, 1);
...
Student s100 = new Student("이름100", 100, 100);
```

> Ex02_ConstructorParams.java

#02. 예약어 this

`this`는 클래스 안에서 **자기 자신**을 의미하는 키워드이다.

1) 지역 변수와 전역 변수의 차이

메서드 내의 지역변수와 전역으로서의 멤버변수 이름이 같은 경우

메서드 안에서는 멤버변수와 동일한 이름의 변수를 선언할 수 있다. 이 경우에는 유효성 범위가 가까운 변수가 우선적으로 인식되기 때문에 멤버변수를 식별할 수 없다.

> Ex03_MyNameText.java

파라미터와 멤버변수 이름이 같은 경우

파라미터 역시 메서드 안에서 선언되는 지역변수의 일종이므로 위와 같은 현상이 발생한다.

> Ex04_MyNameText2.java

2) 지역 변수와 전역 변수를 구분하기 위한 this

어떤 메서드 안에서 변수에 값을 대입하거나 변수를 활용한 연산을 수행할 때 변수 이름 앞에 `this.`을 명시하면 자기 자신(=현재 클래스)에 속한 변수를 찾는다. 즉, 멤버변수를 지정하고자 할 때 `this` 를 사용한다.

파라미터 역시 메서드 안에서 선언되는 지역변수의 일종이므로 `this`를 사용하여 멤버변수와 구별 할 수 있다.

> Ex05_MyNameTest3.java

3) 메서드를 가리키기 위한 this

멤버변수와 마찬가지로 메서드 이름 앞에 `this` 가 적용되면 현재 클래스 안에 속해 있는 메서드를 가리킨다.

하지만 메서드의 경우 지금까지 사용한 것과 같이 메서드 이름 앞에 아무런 식별자가 없더라도 모두 현재 클래스 안에 속해 있는 메서드 외에 다른 메서드를 지정할 수 있는 방법이 없으므로 메서드 이름 앞에 this를 명시하는 것은 크게 의미가 없다.

#03. 생성자 오버로딩

1) 객체 생성 방법의 다양화

생성자도 메서드의 일종이므로 파라미터의 갯수나 타입을 다르게 정의하여 오버로딩을 적용할 수 있다.

생성자 오버로딩이 적용될 경우 객체의 생성 방법이 다양해 진다.

> Ex06_생성자_오버로딩.java

2) `this`를 활용한 생성자 오버로딩

`this`를 메서드로 사용할 경우 현재 클래스의 생성자를 의미한다.

모든 멤버변수에 대한 완전체 메서드를 정의하고 `this()`를 활용하여 멤버변수에 대한 파라미터를 하나씩 제거하는 순서로 오버로딩을 적용할 수 있다.

> Ex07_생성자_오버로딩_this.java