객체지향 프로그래밍

Object-Oriented Programming

Chapter 04. 객체 지향

- 객체 지향의 개요
- 객체 지향 프로그래밍의 특징
- 클래스의 선언과 객체 생성
- 클래스의 구성 요소와 멤버 접근
- 접근자와 설정자
- 생성자
- 정적 멤버

생성자

생성자 (constructor)

- 생성자의 역할 : 객체를 생성하는 시점에서 필드를 다양한 방식으로 초기화
- 생성자의 선언 방식

```
클래스이름 ( … ) { … }
```

일반적으로 공개되어야 하므로 public으로 선언되지만 아닐 수도 있다.

- 생성자 사용

```
클래스이름 변수 = new 클래스이름(…);
```

생성자

- 생성자 이름은 클래스 이름과 같다.
- 생성자의 반환 타입은 없다.
- 생성자는 new 연산자와 함께 사용하며, 객체를 생성할 때 호출한다.
- 생성자도 오버로딩할 수 있다.

생성자

디폴트 생성자

- 모든 클래스는 최소한 하나의 생성자가 있음
- 만약 생성자를 선언하지 않으면 컴파일러가 자동으로 디폴트 생성자를 추가

```
class Circle {
   private double radius;
   public double getRadius() { … }
   public void setRadius(double radius) { … }
   …
}

public class CircleDemo {
   public static void main(String[] args) {
      Circle myCircle = new Circle();
      …
   }
}
```

생성자를 하나도 선언하지 않았지만 컴파일러가 디폴트 생성자인 Circle( )를 자동으로 추가한다.

Circle 클래스에서 생성자를 선언하지 않았지만, Circle 생성자를 사용해 객체를 생성할 수 있다.

생성자

생성자 오버로딩

- 메서드와 같이 생성자 또한 오버로딩 가능
- 다양한 매개변수를 갖는 여러 개의 생성자 선언 가능

생성자

Practice

- **예제 4-5**: 생성자 추가 (p.147)
  - Package: chap04
  - Class: CircleDemo

- **예제 4-6**: 생성자 오버로딩 (p.147)
  - Package: chap04
  - Class: CircleDemo

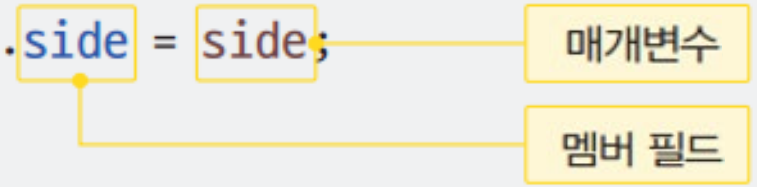
생성자

this 키워드

- 객체 자기 자신을 참조할 수 있도록 하는 키워드

```
class Square {
    private double side;

    public void setRadius(double side) {
        this.side = side;
    }
}
```



생성자

this() 키워드

- 생성자에서 다른 생성자를 호출할 수 있도록 하는 키워드
- 오버로딩된 생성자에서 생기는 중복 코드 제거 가능
- 생성자의 가장 첫 번째 줄에서 실행해야 함

```
public Circle() {
   radius = 10.0;
   this("빨강");
}
```

기존 생성자를 호출하기 전에 다른 실행문이 있어 오류가 발생한다. 따라서 순서를 바꿔야 한다.

생성자

Practice

- **예제 4-7**: this() 키워드 실습 (p.150)
  - Package: chap04
  - Class: CircleDemo

생성자

연속 호출

- 중복 코드를 줄이고 가독성 향상을 위해 연속 호출을 적용할 수 있다

setName( )이 this를 반환하므로 Person 객체이다. 따라서 person.setName( )은 setAge( )를 호출할 수 있다.

```
Person person = new Person();
person.setName("민국").setAge(21).sayHello();
```

setAge( )도 this를 반환하므로 Person 객체이다. 따라서 person.setName( ).setAge( )는 sayHello( )를 호출할 수 있다.

생성자

Practice

- **예제 4-8**: 연속 호출 실습 (p.151)
  - Package: chap04
  - Class: MethodChainDemo

정적 멤버

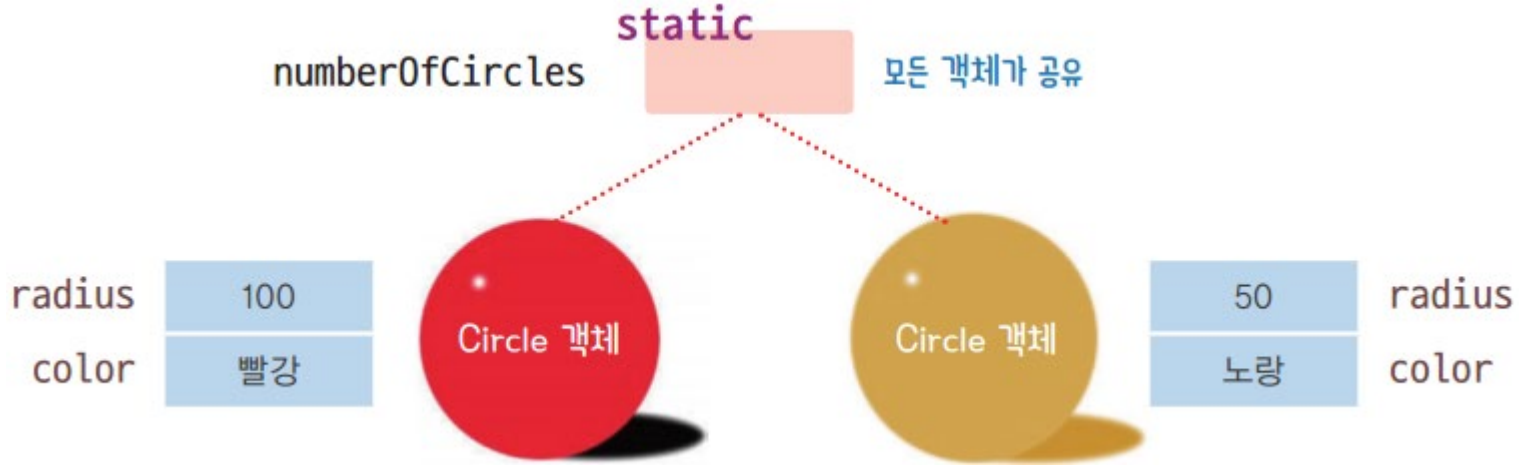
정적 멤버의 필요성

- 필드는 각 객체 간 독립적
- 하지만, 같은 클래스의 객체끼리 공유할 데이터가 필요해질 수도 있음
  - Ex: 생성한 객체 개수, 모든 객체가 공유하는 상수

정적 멤버

정적 멤버

- 자바는 static 키워드로 클래스의 필드를 공유할 수 있도록 지원
- 인스턴스 변수 : static 키워드로 지정되지 않아 공유되지 않은 필드로 인스턴스마다 자신의 필드를 생성
- 정적 변수 혹은 클래스 변수 : static 키워드로 지정하여 모든 인스턴스가 공유하는 필드



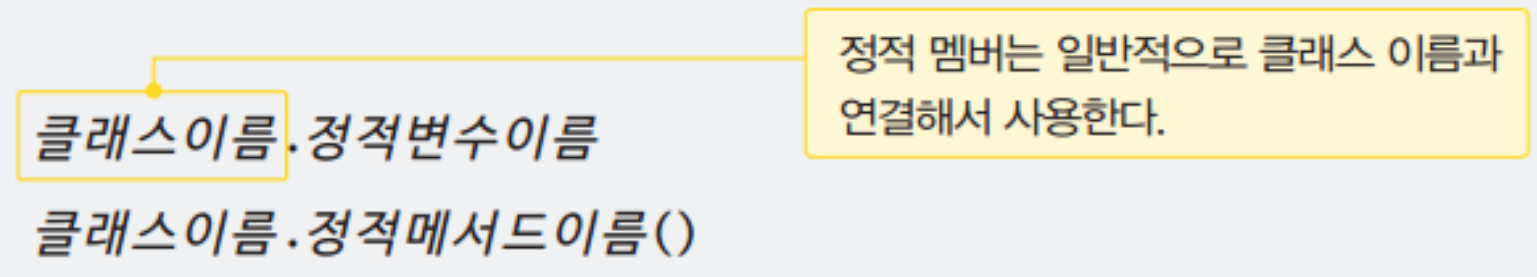
정적 멤버

정적 멤버 – 정적 메서드

- 인스턴스와 무관하게 호출 가능
- static 키워드를 사용하여 선언
- 객체 자신을 가리키는 this 키워드를 사용할 수 없음

정적 멤버

정적 멤버의 활용



정적 멤버

Practice

- **예제 4-9**: 정적 변수 활용 (p.154)
  - Package: chap04
  - Class: CircleDemo

- **예제 4-10**: 정적 메서드 활용 (p.156)
  - Package: chap04
  - Class: UtilDemo

정적 멤버

정적 블록

- 클래스의 초기화 시 실행되는 블록
- 정적 변수의 초기화 과정이 복잡할 때 활용 가능

Practice

- **예제 4-11**: 정적 블록 활용 (p.157)
  - Package: chap04
  - Class: OneToTenDemo

Q & A

객체 지향

Challenge

- **도전 과제 1~3**. 프린터 모델링 (p. 158)