

Chapter

06

클래스



06-5. 인스턴스 멤버와 정적 멤버

혼자 공부하는 자바(개정판) (신용권 저)

❖ 목차

- 시작하기 전에
- 인스턴스 멤버와 this
- 정적 멤버와 static
- 싱글톤
- final 필드와 상수
- 키워드로 끝내는 핵심 포인트



시작하기 전에

[핵심 키워드] : 인스턴스 멤버, this, 정적 멤버, static, final 필드, 싱글톤, 상수

[핵심 포인트]

클래스에 선언된 필드와 메소드가 모두 객체 내부에 포함되는 것은 아니다.
객체가 있어야 사용 가능한 멤버가 있고, 그렇지 않는 멤버도 있다.

❖ 인스턴스 멤버

■ 객체 마다 가지고 있는 멤버

- - 인스턴스 필드: 힙 영역의 객체 마다 가지고 있는 멤버, 객체마다 다른 데이터를 저장
- - 인스턴스 메소드: 객체가 있어야 호출 가능한 메소드,
 - 클래스 코드(메소드 영역)에 위치하지만, 이해하기 쉽도록 객체 마다 가지고 있는
 - 메소드라고 생각해도 됨

❖ 정적 멤버

■ 객체와 상관없는 멤버, 클래스 코드(메소드 영역)에 위치

- 정적 필드 및 상수: 객체 없이 클래스만으로도 사용 가능한 필드
- 정적 메소드: 객체가 없이 클래스만으로도 호출 가능한 메소드



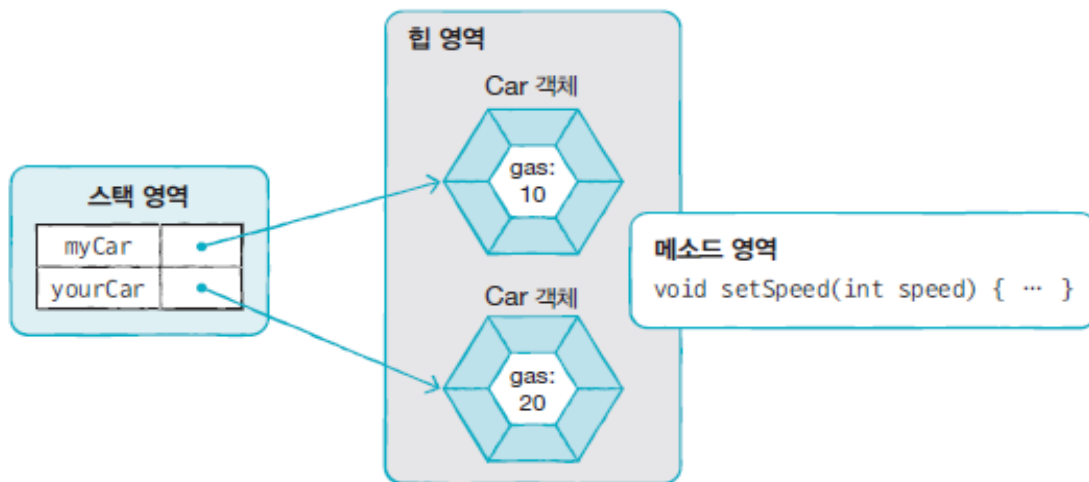
인스턴스 멤버와 this

❖ 인스턴스 (instance) 멤버:

- 객체를 생성한 후 사용할 수 있는 필드와 메소드

```
public class Car {  
    //필드  
    int gas;  
  
    //메소드  
    void setSpeed(int speed) { ... }  
}
```

```
Car myCar = new Car();  
myCar.gas = 10;  
myCar.setSpeed(60);  
  
Car yourCar = new Car();  
yourCar.gas = 20;  
yourCar.setSpeed(80);
```



인스턴스 멤버와 this

❖ this

- 객체 내에서 인스턴스 멤버에 접근하기 위해 사용
- 생성자와 메소드의 매개 변수 이름이 필드와 동일할 경우, 필드 임을 지정하기 위해 주로 사용

```
Car(String model) {  
    this.model = model;  
}  
  
void setModel(String model) {  
    this.model = model;  
}
```



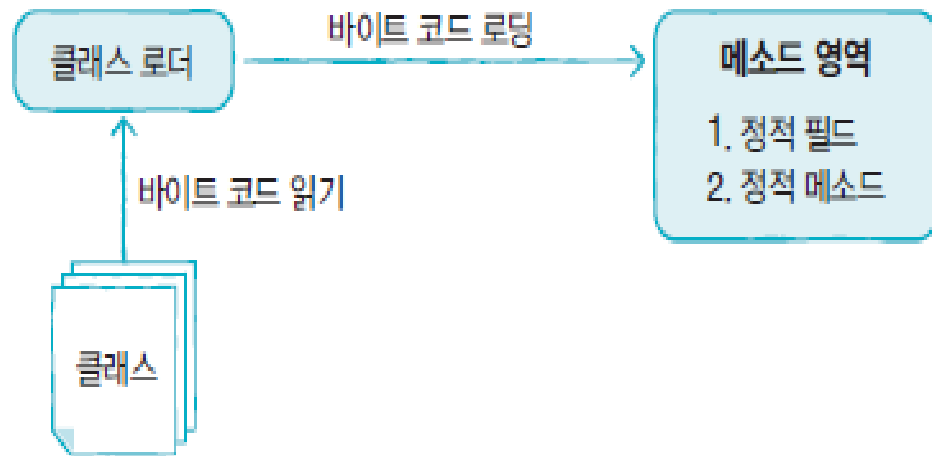
정적 멤버와 static

❖ 정적 (static) 멤버

- 클래스에 고정된 멤버로서 객체 생성하지 않고 사용할 수 있는 필드와 메소드

❖ 정적 멤버 선언

```
public class 클래스 {  
    //정적 필드  
    static 타입 필드 [= 초기값];  
  
    //정적 메소드  
    static 리턴 타입 메소드( 매개변수선언, ... ) { ... }  
}
```



정적 멤버와 static

❖ 정적 멤버 사용

- 클래스 이름과 함께 도트 연산자로 접근

```
클래스.필드;  
클래스.메소드( 매개값, ... );
```

```
public class Calculator {  
    static double pi = 3.14159;  
    static int plus(int x, int y) { ... }  
    static int minus(int x, int y) { ... }  
}
```

```
double result1 = 10 * 10 * Calculator.pi;  
int result2 = Calculator.plus(10, 5);  
int result3 = Calculator.minus(10, 5);
```



정적 멤버와 static

■ 인스턴스 멤버와 정적 멤버 선택 기준

- 객체마다 다를 수 있는 필드값 -> 인스턴스 필드로 선언
- 그렇지 않고 객체마다 다를 필요가 없는 필드값 -> 정적 필드로 선언

```
public class Calculator {  
    String color;           //계산기별로 색깔이 다를 수 있습니다.  
    static double pi = 3.14159; //계산기에서 사용하는 파이( $\pi$ ) 값은 동일합니다.  
}
```

- 메소드 블록에 인스턴스 필드 또는 인스턴스 메소드를 사용할 경우 -> 인스턴스 메소드로 선언
 그렇지 않을 경우 -> 정적 메소드로 선언

```
public class Calculator {  
    String color;           //인스턴스 필드  
    void setColor(String color) { this.color = color; } //인스턴스 메소드  
    static int plus(int x, int y) { return x + y; }      //정적 메소드  
    static int minus(int x, int y) { return x - y; }    //정적 메소드  
}
```



정적 멤버와 static

❖ 정적 메소드 선언 시 주의할 점

- 정적 메소드 선언 시 그 내부에 인스턴스 필드 및 메소드 사용 불가
- 정적 메소드 선언 시 그 객체 자신 참조인 this 키워드 사용 불가

```
public class ClassName {  
    //인스턴스 필드와 메소드  
    int field1;  
    void method1() { ... }  
    //정적 필드와 메소드  
    static int field2;  
    static void method2() { ... }  
  
    //정적 메소드  
    static void Method3 {  
        this.field1 = 10; // (x)  
        this.method1(); // (x)  
        field2 = 10;      // (o)  
        method2();        // (o)  
    }  
}
```

← 컴파일 에러

- 정적 메소드에서 인스턴스 멤버 사용하려는 경우
객체 우선 생성 후 참조 변수로 접근

```
static void Method3() {  
    ClassName obj = new ClassName();  
    obj.field1 = 10;  
    obj.method1();  
}
```



정적 멤버와 static

- main() 메소드 정적 메소드이므로 동일 규칙 적용

```
public class Car {  
    int speed;  
  
    void run() { ... }  
  
    public static void main(String[] args) {  
        speed = 60; // (x)  
        run();      // (x)  
    }  
}
```

← 컴파일 에러

```
public static void main(String[] args) {  
    Car myCar = new Car();  
    myCar.speed = 60;  
    myCar.run();  
}
```

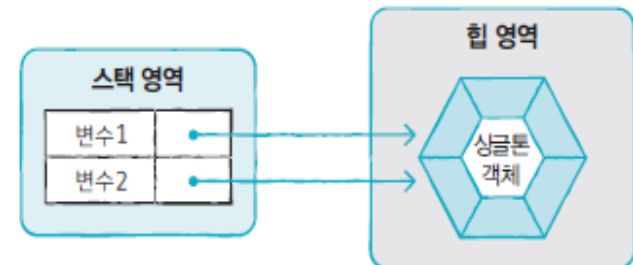


❖ 싱글톤 (singleton)

- 전체 프로그램에서 단 하나의 객체만 만들도록 보장하는 코딩 기법
- 싱글톤 작성 방법
 - 클래스 외부에서 new 연산자 통해 생성자 호출하는 것 불가능하도록 **private 접근 제한자** 사용
 - 자신의 타입인 정적 필드 선언 후 자신의 객체 생성해 초기화
 - 외부에서 호출할 수 있는 getInstance() 선언
 - 정적 필드에서 참조하는 자신의 객체 리턴

```
public class 클래스 {  
    //정적 필드  
    private static 클래스 singleton = new 클래스();  
  
    //생성자  
    private 클래스() {}  
  
    //정적 메소드  
    static 클래스 getInstance() {  
        return singleton;  
    }  
}
```

```
클래스 변수1 = 클래스.getInstance();  
클래스 변수2 = 클래스.getInstance();
```



final 필드와 상수

❖ final 필드

- 초기값이 저장되면 최종값이 되어 프로그램 실행 도중 수정 불가
- final 필드의 초기값 주는 방법
 - 단순 값일 경우 필드 선언 시 초기화(주로 정적 필드(상수)일 경우)
 - 객체 생성 시 외부 데이터로 초기화 필요한 경우 생성자에서 초기화(주로 인스턴스 필드일 경우)
- 인스턴스 final 필드
 - 객체에 한번 초기화된 데이터를 변경 불가로 만들 경우: ex) 주민 번호

```
final 타입 필드 [= 초기값];
```

```
final String ssn; //생성자에서 초기화
```

- 정적 final 필드 (관례적으로 모두 대문자로 작성)
 - 불편의 값인 상수를 만들 경우: ex)

```
static final 타입 상수 = 초기값;
```

```
static final double PI = 3.14159;  
static final double EARTH_RADIUS = 6400;  
static final double EARTH_AREA = 4 * Math.PI * EARTH_RADIUS * EARTH_RADIUS;
```

키워드로 끝내는 핵심 포인트

- **인스턴스 멤버** : 객체를 생성한 후 사용할 수 있는 필드와 메소드.

인스턴스 필드. 인스턴스 메소드

- **this** : 객체 내부에서도 인스턴스 멤버에 접근하기 위해 this를 사용할 수 있음.

주로 생성자와 메소드의 매개 변수 이름이 필드와 동일한 경우, 인스턴스 멤버인 필드임을 명시

- **정적 멤버** : 클래스에 고정된 멤버로서 객체 생성하지 않고 사용할 수 있는 필드와 메소드

- **static** : 정적 멤버를 선언할 때 사용하는 키워드입니다.

- **싱글톤** : 전체 프로그램에서 단 하나의 객체만 만들도록 보장해야 하는 경우 사용하는 코드 패턴

- **final 필드** : 초기값 저장되면 이것이 최종값이 되어 프로그램 실행 도중 수정할 수 없는 필드.

- **상수** : 불변의 값을 저장하는 정적 필드. final static 키워드로 선언





Thank You!