제 4 장
클래스와 객체 Part 3
Sttic Members

Static member

눈은 각 사람마다 있고 공기는 모든 사람이 소유(공유)한다

사람은 모두 각각 눈을 가지고 태어난다.

세상에는 이미 공기가 있으며 태어난 사람은 모두 공기를 공유한다.
그리고 공기 역시 각 사람의 것이다.

static 멤버와 non-static 멤버

- non-static 멤버의 특성
  - 공간적 - 멤버들은 객체마다 독립적으로 별도 존재
    - 인스턴스 멤버라고도 부름
  - 시간적 - 필드와 메소드는 객체 생성 후 비로소 사용 가능
  - 비공유의 특성 - 멤버들은 여러 객체에 의해 공유되지 않고 배타적

- static 멤버란?
  - 객체를 생성하지 않고 사용가능
  - 객체마다 생성되는 것이 아니며
  - 클래스당 하나만 생성됨
    - 클래스 멤버라고도 부름
  - 특성
    - 공간적 특성 - static 멤버들은 클래스 당 하나만 생성.
    - 시간적 특성 - static 멤버들은 클래스가 로딩될 때 공간 할당.
    - 공유의 특성 - static 멤버들은 동일한 클래스의 모든 객체에 의해 공유

```
class StaticSample {
   int n; // non-static 필드
   void g() {…} //non-static 메소드
   static int m; // static 필드
   static void f() {…} //f()는 static 메소드
}
```

non-static 멤버와 static 멤버의 차이

	non-static 멤버	static 멤버

선언	class Sample {
	  int n;
	  void g() {…}
	}
		class Sample {
		  static int m;
		  static void g() {…}
		}

공간적 특성	멤버는 객체마다 별도 존재. - 인스턴스 멤버라고 부름.	멤버는 클래스 당 하나 생성 - 멤버는 객체 내부가 아닌 별도의 공간에 생성 - 클래스 멤버라고 부름
시간적 특성	객체 생성 시 함께 멤버 생성됨 - 객체가 생길 때 멤버도 생성 - 객체 생성 후 멤버 사용 가능 - 객체가 사라지면 멤버도 사라짐	클래스 로딩 시에 멤버 생성 - 객체가 생기기 전에 이미 생성 - 객체가 생기기 전에도 사용 가능 - 객체가 사라져도 멤버는 사라지지 않음 - 멤버는 프로그램이 종료될 때 사라짐
공유의 특성	동일한 클래스의 객체들에 의해 공유되지 않음. - 멤버는 객체 내에 각각 공간 유지	동일한 클래스의 객체들에 의해 공유됨

예 제

```
public class Car {
        private String model;
        private String color;
        private int speed;

         // 자동차의 시리얼 번호
        private int id;
        private static int numbers = 0;

        public Car(String m, String c, int s) {
              model = m;
              color = c;
              speed = s;
               // 자동차의 개수를 증가하고 id에 대입한다.
              id = ++numbers;
        }
}
```

```
public class CarTest {
        public static void main(String args[]) {
                Car c1 = new Car("S600", “white”, 80);          // 첫 번째 생성자 호출
                Car c2 = new Car("E500", “blue”, 20);           // 첫 번째 생성자 호출
                int n = Car.numbers;          // 정적 변수
                System.out.println("지금까지 생성된 자동차 수 = " + n);
        }
}
```

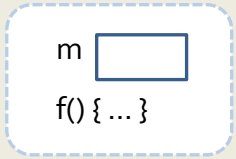
지금까지 생성된 자동차 수 = 2

static 멤버를 객체의 멤버로 접근하는 사례

```
class StaticSample {
    public int n;
    public void g() {
        m = 20;
    }
    public void h() {
        m = 30;
    }
    public static int m;
    public static void f() {
        m = 5;
    }
}

public class Ex {
    public static void main(String[] args) {
        StaticSample s1, s2;
        s1 = new StaticSample();
        s1.n = 5;
        s1.g();
        s1.m = 50; // static
        s2 = new StaticSample();
        s2.n = 8;
        s2.h();
        s2.f(); // static
        System.out.println(s1.m);
    }
}
```

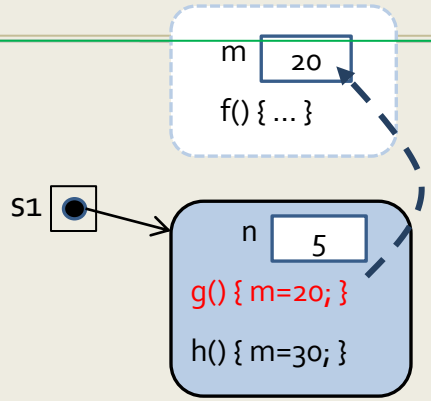
5

StaticSample s1, s2;



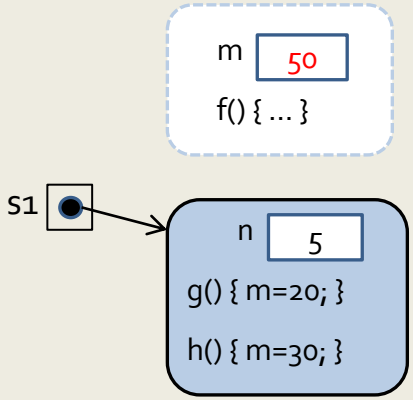
static 멤버 생성

s1 = new StaticSample();
s1.n = 5;
s1.g();



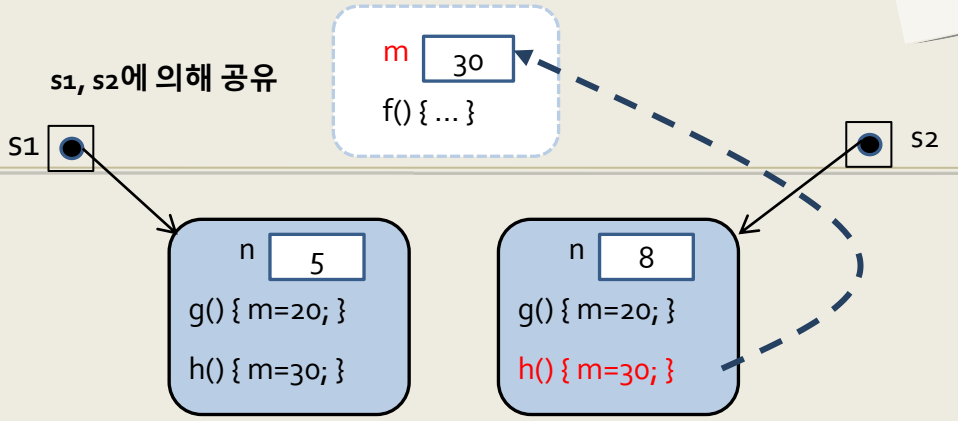
s1.g()호출에 의해 static 멤버 m의 값이 20으로 설정

s1.m = 50;



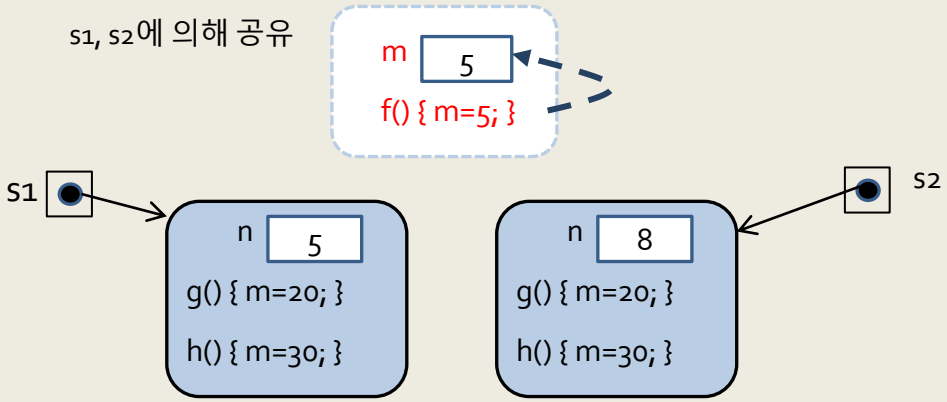
s1.m=50;에 의해 static 멤버 m의 값이 50으로 설정

다음 페이지 계속



s2 = new StaticSample();
s2.n = 8;
s2.h();

s2.h()호출에 의해
static 멤버 m의 값이
30으로 설정



s2.f();

s2.f()호출에 의해
static 멤버 m의 값이
5로 설정

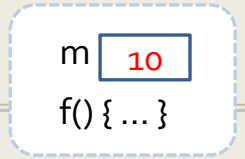
System.out.println(s1.m);

5 출력

static 멤버를 클래스 이름으로 접근하는 사례

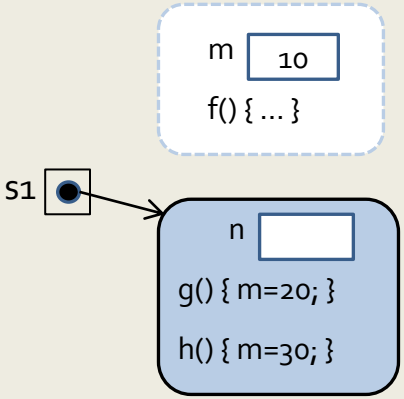
```
class StaticSample {
    public int n;
    public void g() {
        m = 20;
    }
    public void h() {
        m = 30;
    }
    public static int m;
    public static void f() {
        m = 5;
    }
}

public class Ex {
    public static void main(String[] args) {
        StaticSample.m = 10;

        StaticSample s1;
        s1 = new StaticSample();
        System.out.println(s1.m);
        s1.f();
        StaticSample.f();
    }
}
```

StaticSample.m = 10;



static 멤버 생성

StaticSample s1;
s1 = new StaticSample();

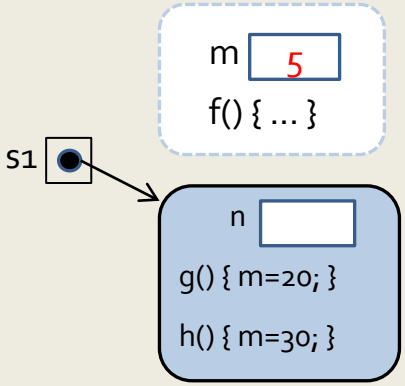


객체 s1 생성

System.out.println(s1.m);

10 출력

s1.f();



StaticSample.f();

s1.f();의 호출과 동일함

static의 활용

- 전역 변수와 전역 함수를 만들 때 활용
  - 자바에서의 캡슐화 원칙
    - 다른 모든 클래스에서 공유하는 전역 변수나 전역 함수도 클래스 내부에만 정의
- java.lang.Math 클래스
  - JDK와 함께 배포되는 java.lang.Math 클래스
  - 모든 메소드가 static으로 정의되어 다른 모든 클래스에서 사용됨
  - 객체를 생성하지 않고 바로 호출할 수 있는 상수와 메소드 제공

```
public class Math {
  static int abs(int a);
  static double cos(double a);
  static int max(int a, int b);
  static double random();
  …
}
```

// 권하지 않는 사용법

```
Math m = new Math();
int n = m.abs(-5);
```

// 바른 사용법

```
int n = Math.abs(-5);
```

static 메소드의 제약 조건

- static 메소드는 오직 static 멤버만 접근 가능
  - 객체가 생성되지 않은 상황에서도 사용이 가능하므로 객체에 속한 인스턴스 메소드, 인스턴스 변수 등 사용 불가
  - 인스턴스 메소드는 static 멤버들을 모두 사용 가능
- static 메소드에서는 this 키워드를 사용할 수 없음
  - 객체가 생성되지 않은 상황에서도 호출이 가능하기 때문에 현재 실행 중인 객체를 가리키는 this 레퍼런스를 사용할 수 없음

static을 이용한 달러와 우리나라 원화 사이의 변환 예제

static 필드와 메소드를 이용하여 달러와 한국 원화 사이의 변환을 해주는 환율 계산기를 만들어 보자.

```
class CurrencyConverter {
    private static double rate; // 한국 원화에 대한 환율
    public static double toDollar(double won) {
        return won/rate; // 한국 원화를 달러로 변환
    }
    public static double toKWR(double dollar) {
        return dollar * rate; // 달러를 한국 원화로 변환
    }
    public static void setRate(double r) {
        rate = r; // 환율 설정. KWR/$1
    }
}
public class StaticMember {
    public static void main(String[] args) {
        CurrencyConverter.setRate(1121); // 미국 달러 환율 설정
        System.out.println("백만원은 " + CurrencyConverter.toDollar(1000000) + "달러
입니다.");
        System.out.println("백달러는 " + CurrencyConverter.toKWR(100) +"원입니
다.");
    }
}
```

```
백만원은 892.0606601248885달러입니다.
백달러는 112100.0원입니다.
```

final

- final 클래스 - 더 이상 클래스 상속 불가능

```
final class FinalClass {
        .....
}
class DerivedClass extends FinalClass { // 컴파일 오류 발생
        .....
}
```

- final 메소드 - 더 이상 오버라이딩 불가능

```
public class SuperClass {
        protected final int finalMethod() { ... }
}

class DerivedClass extends SuperClass {
        protected int finalMethod() { ... } // 컴파일 오류, 오버라이딩 할 수 없음
}
```

final 필드

- final 필드, 상수 정의
  - 상수를 정의할 때 사용

```
class SharedClass {
        public static final double PI = 3.141592653589793;
}
```

  - 상수 필드는 선언 시에 초기 값을 지정하여야 한다
  - 상수 필드는 한 번 정의되면 값을 변경할 수 없다

```
public class FinalFieldClass {
        final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정
        final int COLS; // 컴파일 오류, 초기값을 지정하지 않았음
        void f() {
                int [] intArray = new int [ROWS]; // 상수 활용
                ROWS = 30; // 컴파일 오류 발생, final 필드 값을 변경할 수 없다.
        }
}
```