Power Java

제9장 클래스와 객체 II

이번 장에서 학습할 내용

- 생성자
- 정적 변수
- 정적 메소드
- 접근제어
- this
- 클래스간의 관계

생성자

- 생성자(contructor): 객체가 생성될 때에 필드에게 초기값을 제공하고 필요한 초기화 절차를 실행하는 메소드

그림 9.1 생성자의 역할

생성자의 예제

CarTest.java

```
class Car {
        public int speed;          // 속도
        public int mileage;                // 주행 거리
        public String color;       // 색상

        // 첫 번째 생성자
        public Car(int s, int m, String c) {
                speed = s;
                mileage = m;
                color = c;
        }
        // 두 번째 생성자
        public Car() {
              speed = mileage = 0;
          color = "red";
        }
}
```

생성자의 예제

```
public class CarTest {
        public static void main(String args[]) {
                Car c1 = new Car(100, 0, "blue"); // 첫 번째 생성자 호출
                Car c2 = new Car(); // 두 번째 생성자 호출
        }
}
```

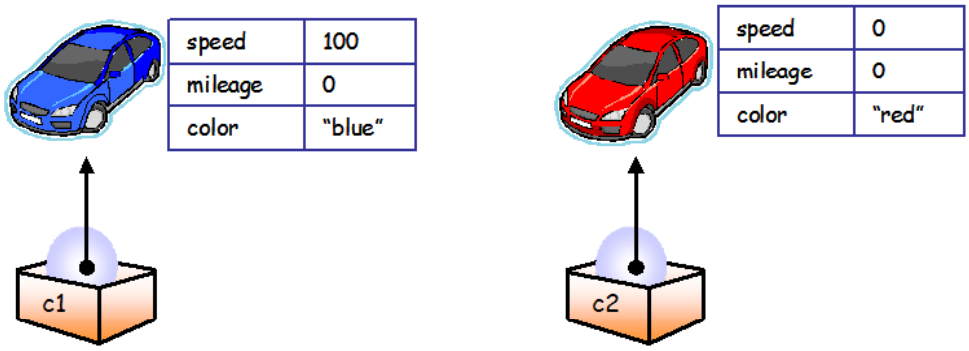


그림 9.2 생성자를 통한 객체의 초기화

디폴트 생성자

- 만약 클래스 작성시에 생성자를 하나도 만들지 않는 경우에는 자동적으로 메소드의 몸체 부분이 비어있는 생성자가 만들어진다.

CarTest1.java

```
class Car {
      public int speed;           // 속도
      public int mileage;         // 주행 거리
      public String color; // 색상
}
public class CarTest1 {
      public static void main(String args[]) {
          Car c1 = new Car(); // 디폴트 생성자 호출
      }
}
```

생성자에서 메소드 호출

Car.java

```
public class Car {
    public int speed; // 속도
    public int mileage; // 주행 거리
    public String color; // 색상

    // 첫 번째 생성자
    public Car(int s, int m, String c) {
        speed = s;
        mileage = m;
        color = c;
    }
    // 색상만 주어진 생성자
    public Car(String c) {
        this(0, 0, c);          // 첫 번째 생성자를 호출한다.
    }
}
```


예제: Date 클래스

DateTest.java

```
import java.util.Scanner;

class Date {
        private int year;
        private String month;
        private int day;

        public Date() {                 // 기본 생성자
                month = "1월";
                day = 1;
                year = 2009;
        }

        public Date(int year, String month, int day) {     // 생성자
                setDate(year, month, day);
        }

        public Date(int year) {                            // 생성자
                setDate(year, "1월", 1);
        }
```

예제: Date 클래스

```
        public void setDate(int year, String month, int day) {
                this.month = month;                    // this는 현재 객체를 가리킨다.
                this.day = day;
                this.year = year;
        }
}
public class DateTest {

        public static void main(String[] args) {
                Date date1=new Date(2009,"3월", 2);        // 2009.3.2
                Date date2=new Date(2010);                 // 2010.1.1
                Date date3=new Date();                     // 2009.1.1
        }
}
```


Time 클래스

TimeTest.java

```
class Time {
        private int hour; // 0 - 23
        private int minute; // 0 - 59
        private int second; // 0 - 59

        // 첫 번째 생성자
        public Time() {
                this(0, 0, 0);
        }

        // 두 번째 생성자
        public Time(int h, int m, int s) {
                setTime(h, m, s);
        }

        // 시간 설정 함수
        public void setTime(int h, int m, int s) {
                hour = ((h >= 0 && h < 24) ? h : 0); // 시간 검증
                minute = ((m >= 0 && m < 60) ? m : 0); // 분 검증
                second = ((s >= 0 && s < 60) ? s : 0); // 초 검증
        }
```

```
        // "시:분:초"의 형식으로 출력
        public String toString() {
                return String.format("%02d:%02d:%02d", hour, minute, second);
        }
}
public class TimeTest {
        public static void main(String args[]) {
                // Time 객체를 생성하고 초기화한다.
                Time time = new Time();

                System.out.print("기본 생성자 호출 후 시간:  ");
                System.out.println(time.toString());

                // 두 번째 생성자 호출
                Time time2 = new Time(13, 27, 6);
                System.out.print("두번째 생성자 호출 후 시간: ");
                System.out.println(time2.toString());

                // 올바르지 않은 시간으로 설정해본다.
                Time time3 = new Time(99, 66, 77);
                System.out.print("올바르지 않은 시간 설정 후 시간: ");
                System.out.println(time3.toString());
        }
}
```

실행결과

```
기본 생성자 호출 후 시간:  00:00:00
두번째 생성자 호출 후 시간: 13:27:06
올바르지 않은 시간 설정 후 시간: 00:00:00
```


Circle 클래스

CircleTest.java

```
class Point {
        public int x;
        public int y;

        // 생성자
        public Point(int a, int b) {
                x = a;
                y = b;
        }
}
```

Circle 클래스

```
class Circle {
        public int radius = 0;
        public Point center; // Point 참조 변수가 필드로 선언되어 있다.

        // 생성자
        public Circle() {
                center = new Point(0, 0);
        }

        public Circle(int r) {
                center = new Point(0, 0);
                radius = r;
        }

        public Circle(Point p, int r) {
                center = p;
                radius = r;
        }
}
```


Circle 클래스

```
public class CircleTest {
        public static void main(String args[]) {
                // Circle 객체를 생성하고 초기화한다.
                Point p = new Point(25, 78);
                Circle c = new Circle(p, 10);
        }
}
```

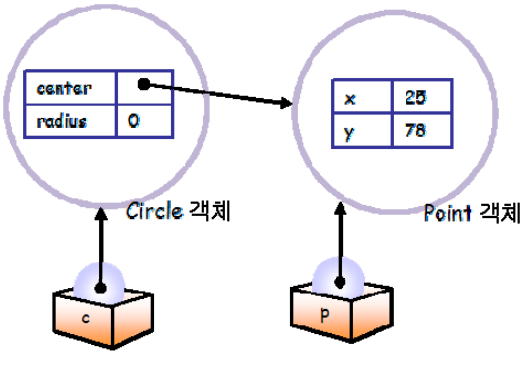


그림 9.3 생성자가 호출된 후의 객체의 상태

정적 변수

- 인스턴스 변수(instance variable): 객체마다 하나씩 있는 변수
- 정적 변수(static variable): 모든 객체를 통틀어서 하나만 있는 변수

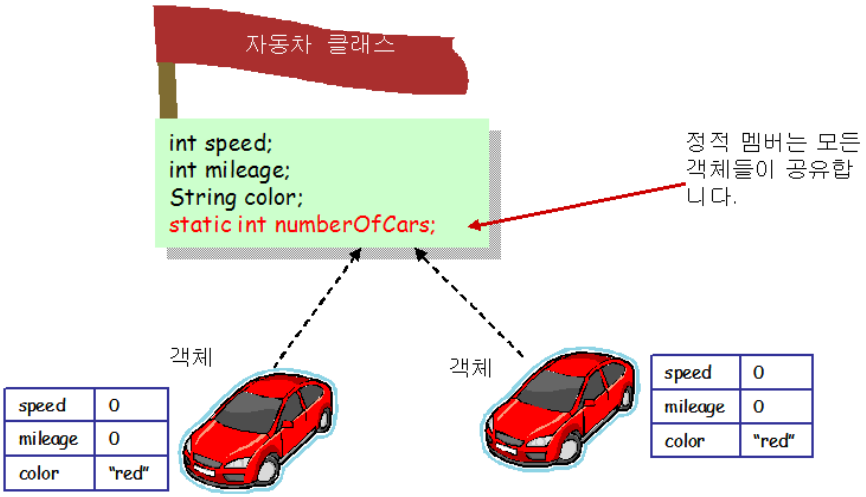


그림 9.4 정적 멤버

정적 변수의 예

Car.java

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

        // 자동차의 시리얼 번호
        private int id;

        // 실제화된 Car 객체의 개수를 위한 정적 변수
        private static int numberOfCars = 0;

        public Car(int s, int m, String c) {
                speed = s;
                mileage = m;
                color = c;

                // 자동차의 개수를 증가하고 id 번호를 할당한다.
                id = ++numberOfCars;
        }
}
```

정적 메소드

- 정적 메소드(static method): 객체를 생성하지 않고 사용할 수 있는 메소드
- (예) Math 클래스에 들어 있는 각종 수학 메소드 들

```
double value = Math.sqrt(9.0);
```


정적 메소드의 예

CarTest3.java

```
class Car {
        private int speed;
        private int mileage;
        private String color;
         // 자동차의 시리얼 번호
        private int id;
         // 실체화된 Car 객체의 개수를 위한 정적 변수
        private static int numberOfCars = 0;

        public Car(int s, int m, String c) {
                speed = s;
                mileage = m;
                color = c;
                 // 자동차의 개수를 증가하고 id 번호를 할당한다.
                id = ++numberOfCars;
        }
       // 정적 메소드
        public static int getNumberOfCars() {
                return numberOfCars; // OK!
        }
}
```

정적 메소드의 예

```
public class CarTest3 {
        public static void main(String args[]) {
                Car c1 = new Car(100, 0, "blue");          // 첫 번째 생성자 호출
                Car c2 = new Car(0, 0, "white"); // 첫 번째 생성자 호출
                int n = Car.getNumberOfCars();   // 정적 메소드 호출
                System.out.println("지금까지 생성된 자동차 수 = " + n);
        }
}
```

실행결과

```
지금까지 생성된 자동차 수 = 2
```


상수

- 공간을 절약하기 위하여 정적 변수로 선언된다.

```
public class Car {
   ...
   static final int MAX_SPEED = 350;
   ...
}
```

EmployeeTest.java

```
import java.util.*;

class Employee {
        private String name;
        private double salary;

        private static int count = 0;     // 정적 변수

       // 생성자
        public Employee(String n, double s) {
               name = n;
               salary = s;
               count++; // 정적 변수인 count를 증가
        }

       // 객체가 소멸될 때 호출된다.
        protected void finalize() {
               count--; // 직원이 하나 줄어드는 것이므로 count를 하나 감소
        }

        // 정적 메소드
        public static int getCount() {
               return count;
        }
}
```


```
public class EmployeeTest {
        public static void main(String[] args) {
               Employee e1,e2,e3;
               e1 = new Employee("김철수", 35000);
               e2 = new Employee("최수철", 50000);
               e3 = new Employee("김철호", 20000);

               int n = Employee.getCount();
               System.out.println("현재의 직원수=" + n);
        }
}
```

실행결과

```
현재의 직원수=3
```

중간 점검 문제

1. 정적 변수는 어떤 경우에 사용하면 좋은가?

2. 정적 변수나 정적 메소드를 사용할 때, 클래스 이름을 통하여 접근하는 이유는 무엇인가?

3. main() 안에서 인스턴스 메소드를 호출할 수 없는 이유는 무엇인가?

접근 제어

- 접근 제어(access control): 다른 클래스가 특정한 필드나 메소드에 접근하는 것을 제어하는 것

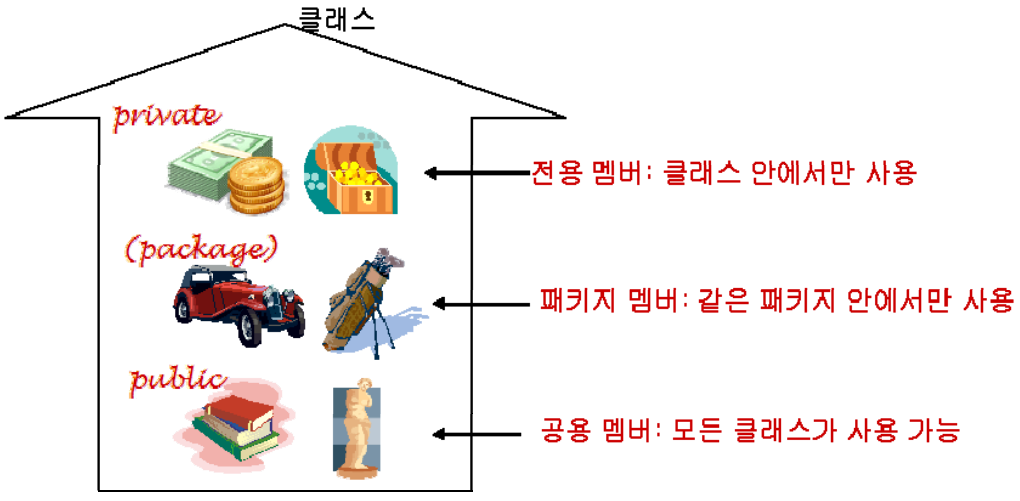


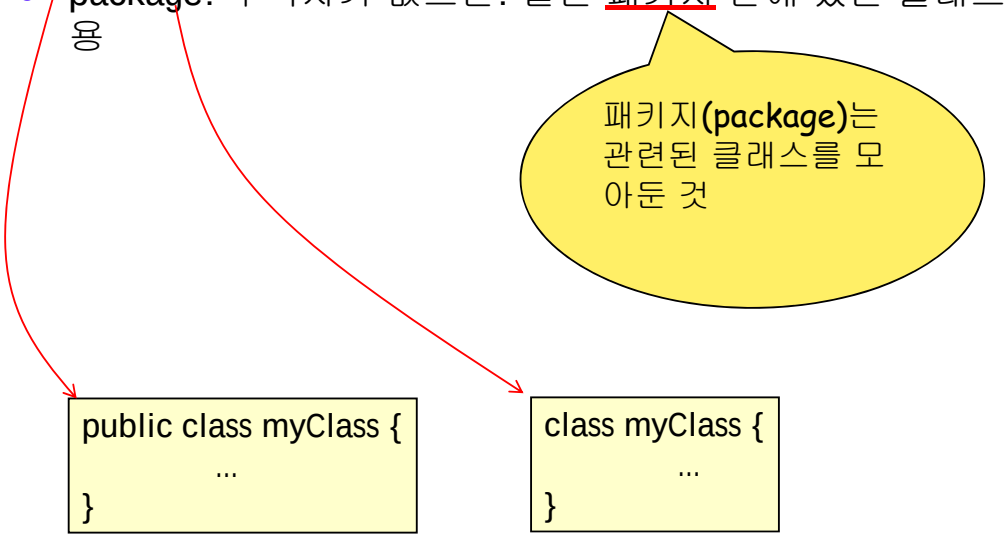
그림 9.5 멤버에 대한 접근 제어

접근 제어의 종류

- 클래스 수준에서의 접근 제어
- 멤버 수준에서의 접근 제어

클래스 수준에서의 접근 제어

- public: 다른 모든 클래스가 사용할 수 있는 공용 클래스
- package: 수식자가 없으면: 같은 패키지 안에 있는 클래스들만이 사용



멤버 수준에서의 접근 제어

분류	접근 지정자	클래스 내부	같은 패키지내의 클래스	다른 모든 클래스
전용 멤버	private	O	X	X
패키지 멤버	없음	O	O	X
공용 멤버	public	O	O	O

EmployeeTest.java

```
import java.util.*;
class Employee {
      private String name;          // private 로 선언
      private int salary;           // private 로 선언
      int age;                      // package 로 선언

      // 생성자
      public Employee(String n, int a, double s) {
          name = n;
          age = a;
          salary = s;
      }
       // 직원의 이름을 반환
      public String getName() {
          return name;
      }
       // 직원의 월급을 반환
      private int getSalary() {     // private 로 선언
          return salary;
      }
       // 직원의 나이를 반환
      int getAge() {                // package로 선언
          return age;
      }
}
```

```
public class EmployeeTest {
    public static void main(String[] args) {
        Employee e;
        e = new Employee("홍길동", 0, 3000);
        e.salary = 300; // 오류! private 변수
        e.age = 26;              // 같은 패키지이므로 OK
        int sa = e.getSalary();  // 오류! private 메소드
        String s = e.getName(); // OK!
        int a = e.getAge();      // 같은 패키지이므로 OK
    }
}
```

실행결과

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field Employee.salary is not visible
        The method getSalary() from the type Employee is not visible
        at EmployeeTest.main(EmployeeTest.java:8)
```


this

- 자기 자신을 참조하는 키워드
  - this.member = 10;
- 생성자를 호출할 때도 사용된다.
  - this(10, 20);

PersonTest.java

```
class Person {
    String lastName;
    String firstName;

    String getLastName() {
        return lastName;
    }

    String getFirstName() {
        return firstName;
    }

    public Person(String lastName, String firstName) {
        this.lastName = lastName;        // this는 현재 객체를 가리킨다.
        this.firstName = firstName;      // this는 현재 객체를 가리킨다.
    }

    public StringbuildName() {
        return String.format("%s %s\n", this.getLastName(), getFirstName());   // ①
    }
}
```


```
public class PersonTest {
    public static void main(String args[]) {
        Person person = new Person("홍", "길동");
        System.out.println(person.buildName());
    }
}
```

중간 점검 문제

1. this의 주된 용도는 무엇인가?

2. this()와 같이 표기하면 무엇을 의미하는가??

클래스와 클래스 간의 관계

- 사용(use): 하나의 클래스가 다른 클래스를 사용한다.
- 집합(has-a): 하나의 클래스가 다른 클래스를 포함한다.
- 상속(is-a): 하나의 클래스가 다른 클래스를 상속한다.

사용 관계

- 클래스 A의 메소드에서 클래스 B의 메소드들을 호출한다.

Complex.java

```
public class Complex {
    private double real;
    private double imag;

    public Complex(double r, double i) {
        real = r;
        imag = i;
    }
    ...
    double getReal() {
        return real;
    }
    double getImag() {
        return imag;
    }
    public Complex add(Complex c) {              // 객체 참조를 매개 변수로 받는다.
        double resultReal = real + c.getReal();
        double resultImag = real + c.getImag();
        return new Complex(resultReal, resultImag);
    }
}
```

집합 관계

- 클래스 A안에 클래스 B가 포함된다.

AlarmClockTest.java

```
class Time {
    private int time;
    private int minute;
    private int second;

    public Time(int t, int m, int s) {
        time = t;
        minute = m;
        second = s;
    }
}
```

```
public class AlarmClock {
    private Time currentTime;
    private Time alarmTime;

    publicAlarmClock(Time a, Time c) {
        alarmTime = a;
        currentTime = c;
    }
}

public class AlarmClockTest {
    public static void main(String args[]) {
        Time alarm = new Time(6, 0, 0);
        Time current = new Time(12, 56, 34);
        AlarmClock c = new AlarmClock(alarm, current);

        System.out.println(c);
    } // end main
} // end class
```


Q & A