컴퓨터 프로그래밍1

실습 11주차

- 클래스의 생성자 -

생성자 (1/4)

생성자

- 생성자는 클래스 안에 선언되어서 객체가 생성될 때 필드에 초기값을 제공하고 필요한 초기화 절차 진행

```
class Car {
    private String color; // 색상
    private int speed; // 속도
    private int gear; // 기어

    // 첫 번째 생성자
    public Car(String c, int s, int g) {
        color = c;
        speed = s;
        gear = g;
    }

    // 두 번째 생성자
    public Car() {
        color = "red";
        speed = 0;
        gear = 1;
    }
}
```

생성자의 이름 앞에는 public 수식어가 반드시 필요하다.

생성자는 클래스와 이름이 같고 반환값이 없다.

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

생성자 (2/4)

디폴트 생성자

- 생성자를 하나도 만들지 않은 경우 컴파일러는 자동으로 디폴트 생성자를 만든다

```
class Car {
    private String color;    // 색상
    private int speed;       // 속도
    private int gear;        // 기어

    public Car() {}
}

public class CarTest1 {
    public static void main(String args[]) {
        Car c1 = new Car(); // 디폴트 생성자 호출
    }
}
```

디폴드 생성자는 컴파일러에 의해 자동으로 추가된다.

```
class Car {

    private String color; // 색상
    private int speed; // 속도
    private int gear; // 기어

    public Car(String c, int s, int g) {
        color = c;
        speed = s;
        gear = g;
    }
}

public class CarTest2 {
    public static void main(String args[]) {
        Car c1 = new Car(); // 오류!
    }
}
```

생성자 (3/4)

생성자에서 메소드 호출하기

- 생성자도 메소드이기때문에 다른 메소드들을 호출 가능 특히 중복 정의된 생성자들은 비슷한 초기화작업을 수행하기 때문에 하나의 생성자가 다른 생성자를 호출하는 경우도 많다

```
public class Car {
	private int speed; // 속도
	private int gear; // 기어
	private String color; // 색상

	// 첫 번째 생성자
	public Car(String c, int s, int g) {
		color = c;
		speed = s;
		gear = g;
	}
	// 색상만 주어진 생성자
	public Car(String c) {
		this(c, 0, 1);		// 첫 번째 생성자를 호출한다.
	}
}
```

생성자 (4/4)

프로그래밍 예제 – P.201

```
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){
        setDate(year, month, day);
    }

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

    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
    }
}
```

정적 변수와 메소드 (1/5)

정적 변수

- 정적 변수는 클래스 변수라고 하며 해당 클래스의 모든 객체들이 공유하는 변수이다

```
public class Car {
	private String color;
	private int speed;
	private int gear;
	 // 자동차의 시리얼 번호
	private int id;
	private static int numberOfCars = 0;

	public Car(String c, int s, int g) {
		color = c;
		speed = s;
		gear = g;
		 // 자동차의 개수를 증가하고 id 번호를 할당한다.
		id = ++numberOfCars;
	}
}
```

정적 변수와 메소드 (2/5)

상수

- 상수를 인스턴스 변수로 선언하면 각 객체마다 하나씩 생성되므로 저장공간이 낭비된다
  상수를 정적변수로 선언하여 저장공간을 절약한다

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

정적 변수와 메소드 (3/5)

정적 메소드

- 정적 메소드는 클래스메소드라고도 하며 객체를 생성할 필요없이 클래스 이름을 통해 호출한다

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

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

```
public class CarTest3 {
      public static void main(String args[]) {
            Car c1 = new Car("blue", 100, 1);  // 첫 번째 생성자 호출
            Car c2 = new Car("white", 0, 1);   // 첫 번째 생성자 호출
            int n = Car.getNumberOfCars(); // 정적 메소드 호출
            System.out.println("지금까지 생성된 자동차 수 = " + n);
      }
}
```

```
지금까지 생성된 자동차 수 = 2
```

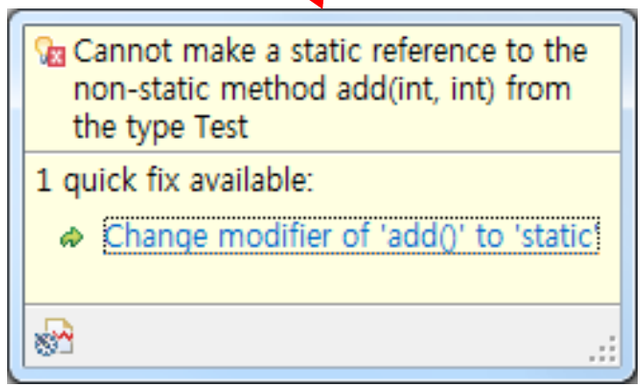
정적 변수와 메소드 (4/5)

정적 메소드

- 정적 메소드에서는 인스턴스 메소드 호출불가

```
class Test{
    public static void test(){
        add(10, 20);
    }

    int add(int x, int y){
        return x+y;
    }
}
```




```
class Test{
    public static void test(){
        add(10, 20);
    }

    static int add(int x, int y){
        return x+y;
    }
}
```

정적 변수와 메소드 (5/5)

프로그래밍 예제 – P.209

```
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 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);
}
```

```
현재의 직원수=3
```

접근 제어 (1/2)

접근 제어

- 다른 클래스가 특정한 필드나 메소드에 접근하는것을 제어

분류	접근지정자	클래스 내부	같은 패키지 내의 클래스	다른 모든 클래스
전용 멤버	Private	O	X	X
패키지 멤버	없음	O	O	X
공용 멤버	Public	O	O	O

접근 제어 (2/2)

프로그래밍 예제 – P.211

```
class Employee {
    private String name; // 전용 멤버
    private int salary; // 전용 멤버
    int age; // 패키지 멤버

    public Employee(String n, int a, int s) {
        name = n;
        age = a;
        salary = s;
    }

    public String getName() { // 공용 멤버
        return name;
    }

    private int getSalary() { // 전용 멤버
        return salary;
    }

    int getAge() { // 패키지 멤버
        return age;
    }
}
```

```
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
}
```

this 참조

접근 제어

- 모든 객체는 키워드 this를 사용해서 자기 자신을 참조 가능

```
class Car{
    int speed;

    public void setSpeed(int speed){
        this.speed = speed;
    }
}
```

필드

매개변수

클래스와 클래스 간 관계 (1/3)

클래스 간 일반적인 관계

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

클래스와 클래스 간 관계 (2/3)

사용 관계(use)

- 클래스의 메소드에서 다른 클래스의 메소드호출

```
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);
    }
}
```

클래스와 클래스 간 관계 (3/3)

집합 관계(has-a)

■ 하나의 객체 안에 다른 객체들에 대한 참조포함

Time에 대한 참조를 포함한다.

```
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;
    }
}
```

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

    public AlarmClock(Time a, Time c) {
        alarmTime = a;
        currentTime = c;
    }
}
```

실습

프로그래밍 실습 – P.220

- MyMetric이라는 클래스를 작성하라
  - 킬로미터를 마일로 변환하는 정적 메소드인 kiloToMile() 작성
  - 반대로 마일을 킬로미터로 변환하는 정적 메소드 mileToKilo() 작성
  - MyMetricTest 클래스에서 정적 메소드를 호출하여 테스트
  - 1Km = 0.621371 mile, 1mile = 1.609344Km

```
<terminated> MyMetric [Java Application] C:₩Pro
1 km 는 0.621371 mile
1 mile 은 1.609344 km
```

과제 (1/2)

과제I. P.22I I번

- 강아지를 나타내는 Dog 이름의 클래스 설계

- 다음의 필드를 선언
  - name : 강아지의 이름, 전용 맴버
  - breed : 강아지의 종류, 공용(static) 맴버
  - age : 강아지의 나이, 전용 맴버

- 다음의 생성자 메소드를 선언
  - Dog(String name, int age) : 강아지 이름과 나이를 초기화
  - Dog(String name, String breed, int age) : 강아지의 이름, 종류, 나이를 초기화
  - print() : 강아지의 정보(이름, 종류, 나이)를 출력

- main 함수가 있는 Test 클래스를 생성하여 main 함수에서 Dog 클래스의 객체를 생성
- 두 종류의 생성자 메소드를 이용하여 객체 생성 후, print() 메소드로 결과 출력

과제 (2/2)

과제2. P.221 2번

- 비행기를 나타내는 Plane 이름의 클래스 설계
  - 제작사, 모델, 최대 승객의 수를 전용 맴버 필드 선언
  - 모든 필드의 접근자와 설정자 메소드 작성
  - Plane 클래스의 생성자를 몇 개를 중복 정의
    - 모든 데이터를 받거나, 아무런 데이터를 받지 않거나, 특정 데이터를 받도록
  - PlaneTest 클래스를 생성하여 main 함수 내에서 Plane 클래스를 태스트 하는 코드 작성

- 생성된 비행기의 수를 나타내는 정적 변수 planes 추가
- 정적 변수 planes를 반환하는 정적 메소드 getPlanes() 추가하고 main()에서 호출

Q&A

수고하셨습니다