



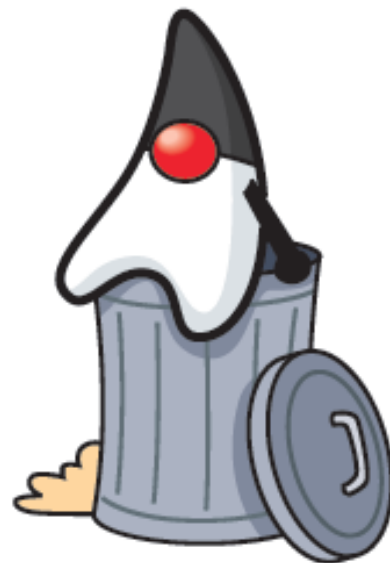
5장 클래스와 객체 2

박숙영

blue@sookmyung.ac.kr

5장의 목표

1. 객체가 언제 생성되고 언제 소멸되는지를 설명할 수 있나요?
2. 가비지 컬렉터의 역할을 설명할 수 있나요?
3. 객체가 메소드에 전달되는 메커니즘을 설명할 수 있나요?
4. 정적 멤버를 선언하고 사용할 수 있나요?



객체의 생성과 소멸

- 객체도 생성되어서 사용되다가 사용이 끝나면 파괴된다. 객체의 일생은 객체를 참조하는 변수와 밀접한 관련이 있다.



객체의 생성



객체의 사용

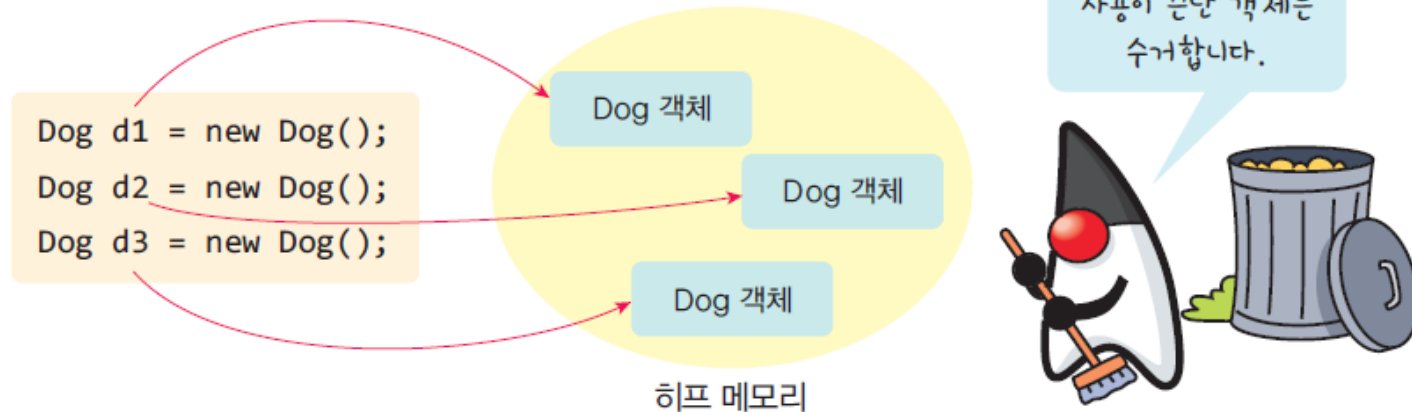


객체의 파괴

그림 5.1 객체의 일생

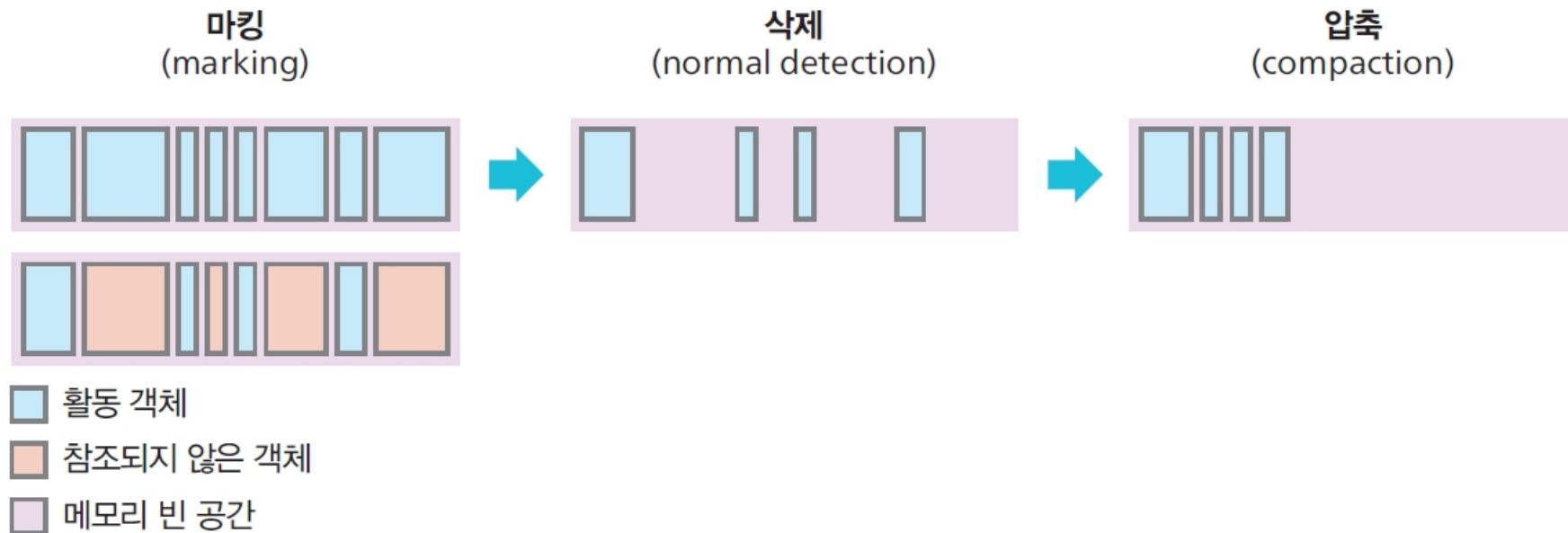
객체의 소멸과 가비지 컬렉션

- 자바에는 객체를 생성하는 연산자는 있지만, 객체를 삭제하는 연산자는 없다.
- 자바에서는 자동 메모리 시스템을 사용하는데 이것을 가비지 컬렉션(garbage collection)이라고 한다.



가비지 컬렉터(Garbage Collector)

- 힙 메모리에서 더 이상 필요 없는 객체를 찾아 지우는 작업을 한다.
- 가비지 컬렉터는 JVM의 중요한 부분이다. JVM 중에서 가장 대표적인 것은 오라클사의 HotSpot이다. HotSpot은 많은 가비지 컬렉션 옵션을 제공한다.



출처: 오라클사

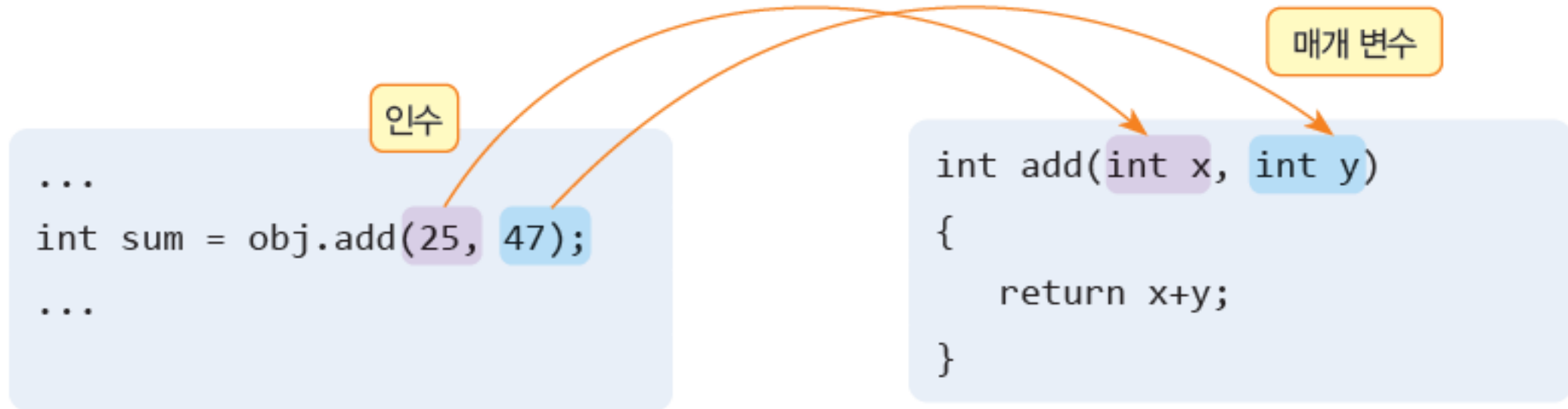
가비지 컬렉션 요청

- 개발자는 System 객체의 gc() 메소드를 호출하여서 가비지 컬렉션을 요청할 수 있다. 가비지 컬렉터가 수행되면 모든 다른 애플리케이션이 멈추기 때문에 가비지 컬렉터의 실행 여부는 JVM이 판단한다.

```
System.gc();           // 가비지 컬렉션 요청
```

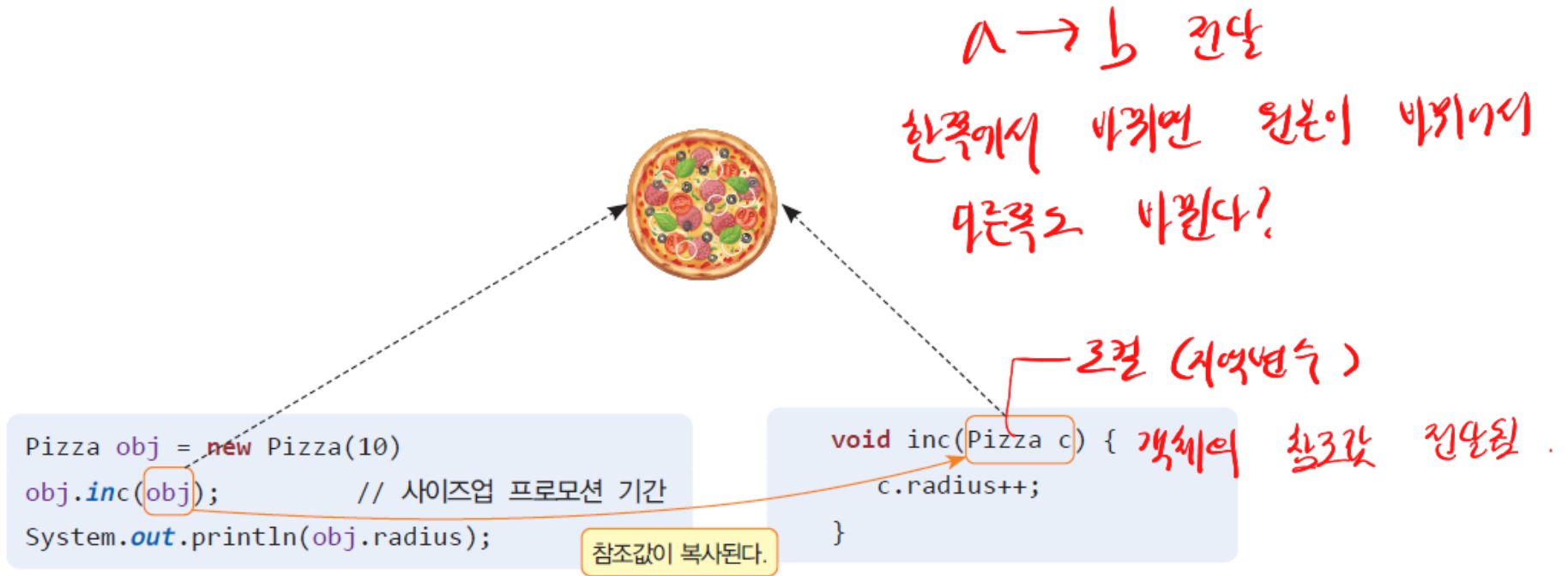
인수전달 방법

- 자바에서 메소드로 인수가 전달되는 방법은 기본적으로 "값에 의한 호출(call-by-value)"이다



객체가 전달되는 경우

- 우리가 객체를 메소드로 전달하게 되면 객체 자체가 복사되어 전달되는 것이 아니고 객체의 참조값만 복사되어서 전달된다. 참조 변수는 참조값(주소)을 가지고 있다.
- 참조값이 매개 변수로 복사되면 메소드의 매개 변수도 동일한 객체를 참조하게 된다.



예제

- 피자 객체 2개를 받아서
더 큰 피자 객체를 반환하
는 메소드

Pizza whosLargest(Pizza other)를
작성하고 테스트하라.

18인치 피자가 더 큼.

```
class Pizza {
    int radius;
    Pizza(int r) {
        radius = r;
    }
    Pizza whosLargest(Pizza other) {
        if (this.radius > other.radius)
            return this;
        else
            return other;
    }
}

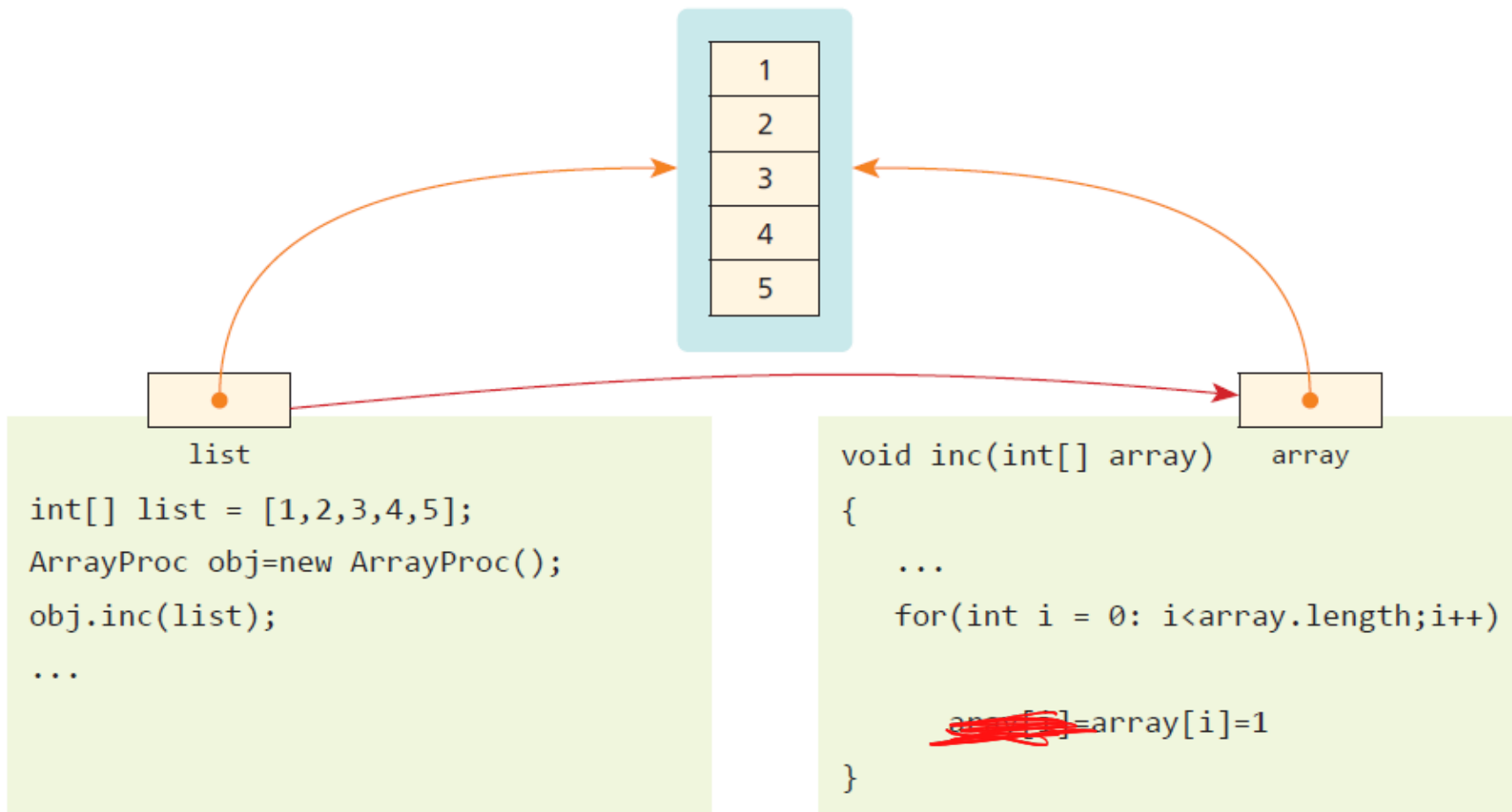
public class PizzaTest {
    public static void main(String args[]) {
        Pizza obj1 = new Pizza(14);
        Pizza obj2 = new Pizza(18);

        Pizza largest = obj1.whosLargest(obj2);
        System.out.println(largest.radius + "인치 피자가 더 큼.");
    }
}
```

Handwritten notes:
- Red arrows point from `obj1` to `this` and from `obj2` to `other` in the `whosLargest` method.
- Red text "대기변수 참조" (reference to instance variable) is written above `obj2` in the `main` method.
- The `obj2` argument in `obj1.whosLargest(obj2)` is circled in red.

배열이 전달되는 경우

- 배열도 객체이기 때문에 배열을 전달하는 것은 배열 참조 변수를 복사하는 것이다.

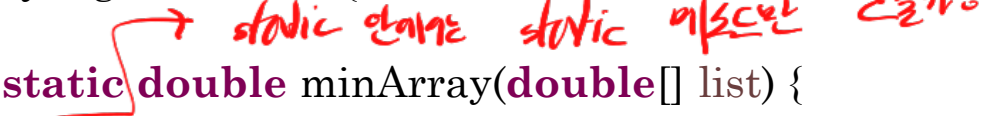


예제: 배열을 받는 메소드 작성하기

- 배열을 받아서 최소값을 계산하여 반환하는 메소드 `minArray(double[] list)`를 작성하고 테스트해보자.

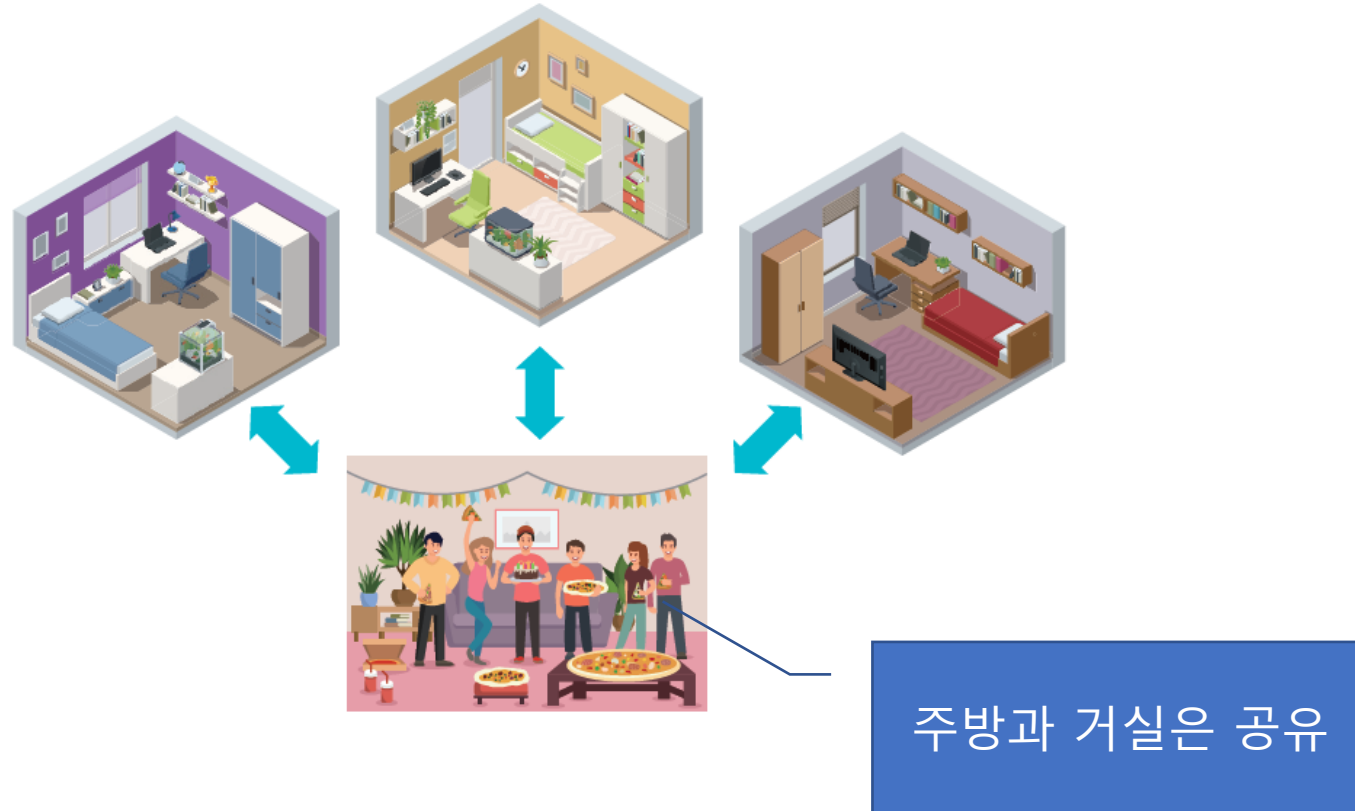
첫 번째 배열의 최소값=0.1

두 번째 배열의 최소값=-9.0

```
public class ArrayArgumentTest {  
      
    public static double minArray(double[] list) {  
        double min = list[0];  
  
        for (int i = 1; i < list.length; i++) {  
            if (list[i] < min)  
                min = list[i];  
        }  
        return (min);  
    }  
    public static void main(String args[]) {  
        double[] a = { 1.1, 2.2, 3.3, 4.4, 0.1, 0.2 };  
        double[] b = { -2.0, 3.0, -9.0, 2.9, 1.5 };  
  
        double min;  
  
        min = minArray(a);  
        System.out.println("첫 번째 배열의 최소값=" + min);  
        min = minArray(b);  
        System.out.println("두 번째 배열의 최소값=" + min);  
    }  
}
```

정적 멤버

- 프로그램을 작성하다 보면 여러 개의 객체가 하나의 변수를 공유해야 되는 경우가 있다. 이러한 멤버를 정적 멤버(static member) 또는 클래스 멤버(class member)라고 한다.



인스턴스 멤버 vs 정적 멤버 - *static* 변수.

```
class Television {  
    int channel;  
    int volume;  
    boolean onOff;  
    static int count;  
}
```

*같은 클래스 내
각각의 객체들이 공유하는 변수.*

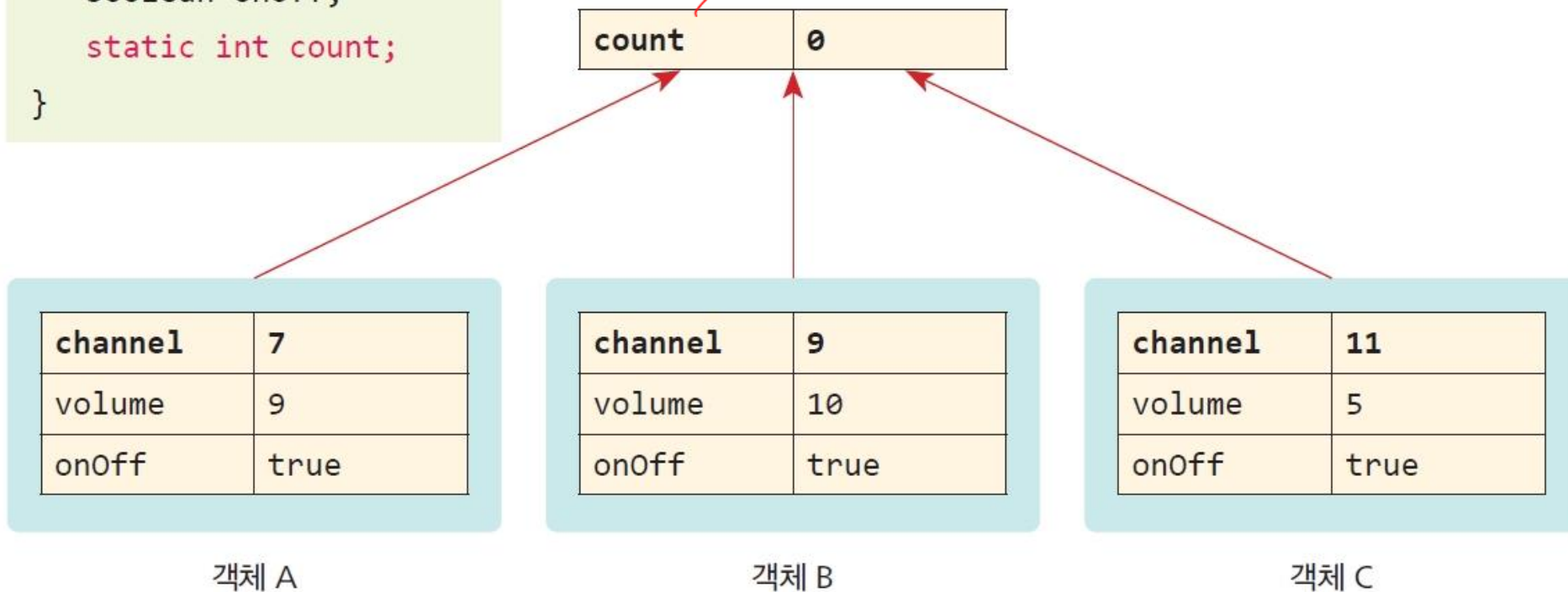


그림 5.2 정적 변수

예제: 정적 변수 사용하기

- 어떤 가게에서 하루에 판매되는 피자의 개수를 알고 싶다고 하자. 피자의 개수를 알기 위해서는 지금까지 피자가 얼마나 생성되었는지를 알아야 한다. 이러한 경우에 정적 변수를 선언하고 생성자에서 개수를 증가시키면 된다.

지금까지 판매된 피자 개수 = 3

```
public class Pizza {  
    private String toppings;  
    private int radius;  
    static final double PI = 3.141592;  
    static int count = 0;  
    public Pizza(String toppings){  
        this.toppings = toppings;  
        count++;  
    }  
}
```

// 상수 정의
// 정적 필드

```
public class PizzaTest {  
    public static void main(String args[]) {  
        Pizza p1 = new Pizza("Super Supreme");  
        Pizza p2 = new Pizza("Cheese");  
        Pizza p3 = new Pizza("Pepperoni");  
        int n = Pizza.count;  
        System.out.println("지금까지 판매된 피자 개수 = " + n);  
    }  
}
```

정적 메소드

- 정적 메소드도 정적 변수와 마찬가지로 static 수식자를 메소드 앞에 붙여서 만든다.

```
public class Math {  
    public static double sqrt(double a) {  
        ...  
    }  
}  
...  
double value = Math.sqrt(9.0);
```

- 객체를 생성하지 않고 클래스 이름으로 접근해서 사용 가능

예제: 정적 메소드 사용하기

- 간단한 연산을 제공하는 MyMath 클래스를 작성하여 보자. MyMath 클래스는 값을 계산하는 power() 함수와 절대값 함수를 제공한다. 모두 정적 메소드로 정의해보자.

```
public class MyMath {  
    public static int abs(int x) { return x>0?x:-x; }  
    public static int power(int base, int exponent) {  
        int result = 1;  
        for (int i = 1; i <= exponent; i++)  
            result *= base;  
        return result;  
    }  
}
```

```
public class MyMathTest {  
    public static void main(String args[]) {  
        System.out.println("10의 3승은 "+MyMath.power(10, 3));  
    }  
}
```

10의 3승은 1000

정적 변수의 활용

- 정적 메소드는 정적 멤버만 사용할 수 있다.

```
class Test {  
    int a;          // 인스턴스 변수  
    static int b;   // 정적 변수  
  
    void sub1() { a = 0; }    // OK!  
    static void sub2() { a = 0; } // 오류! 정적 메소드에서는 인스턴스 멤버를 사용하면 안 됨.  
}
```

```
public class Test {  
    public static void main(String args[]) {  
        add(10,20); // 오류!! 정적 메소드 안에서 인스턴스 메소드 호출  
    }  
    int add(int x, int y) {  
        return x + y;  
    }  
}
```

- 정적 메소드에서 정적 메소드를 호출하거나 정적 멤버를 사용하는 것은 가능하다.

정적 변수의 활용

- 정적 메소드는 this를 사용할 수 없다. (객체를 생성하지 않아서)

```
class Test {  
    static int a;                // 인스턴스 변수  
  
    static void sub(int x) { this.a = x; } // 오류! 정적 메소드에서는 this 사용 안 됨  
}
```

예제: 정적 메소드 사용하기

- main()도 정적 메소드이기 때문에 인스턴스 메소드를 호출할 수 없다. 하지만 정적 메소드는 main()에서 호출할 수 있다. 우리는 3제곱 계산 프로그램을 정적 메소드만을 이용하여 작성해보자.

```
public class Test {  
    public static int cube(int x) {                // 정적 메소드  
        int result = x*x*x;  
        return result;  
    }  
    public static void main(String args[]) {        // 정적 메소드  
        System.out.println("10*10*10은 "+cube(10)); // 정적 메소드 호출  
    }  
}
```

10*10*10은 1000

final 키워드

- 어떤 필드에 final을 붙이면 상수가 된다. 상수를 정의할 때 static과 final 수식어를 동시에 사용하는 경우가 많다

```
public class Car {
```

```
...
```

```
    static final int MAX_SPEED = 350;
```

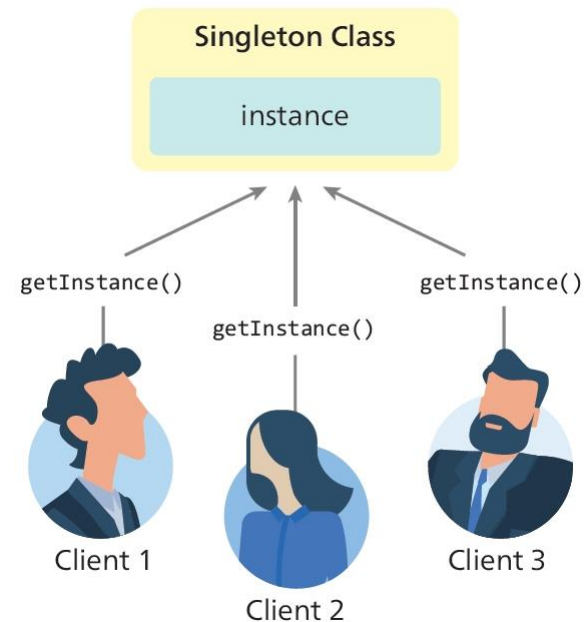
```
    ..
```

```
}
```

상수는 클래스 변수로 만들어서 공유하는 것이 메모리 공간을 절약한다.

Lab: 싱글톤 패턴

- 객체 중에는 **전체** 시스템을 통틀어서 딱 하나만 존재하여야 하는 것들이 있다. 예를 들어 환경설정 클래스나 혹은 네트워크 연결 풀(Pool), 스레드 풀(Pool)을 관리하는 클래스들이다.
- 이럴 경우에 사용할 수 있는 디자인 패턴이 있다. 싱글톤 패턴(singleton design pattern)은 하나의 프로그램 내에서 하나의 인스턴스만을 생성해야 하는 경우에 사용된다.



Sol.

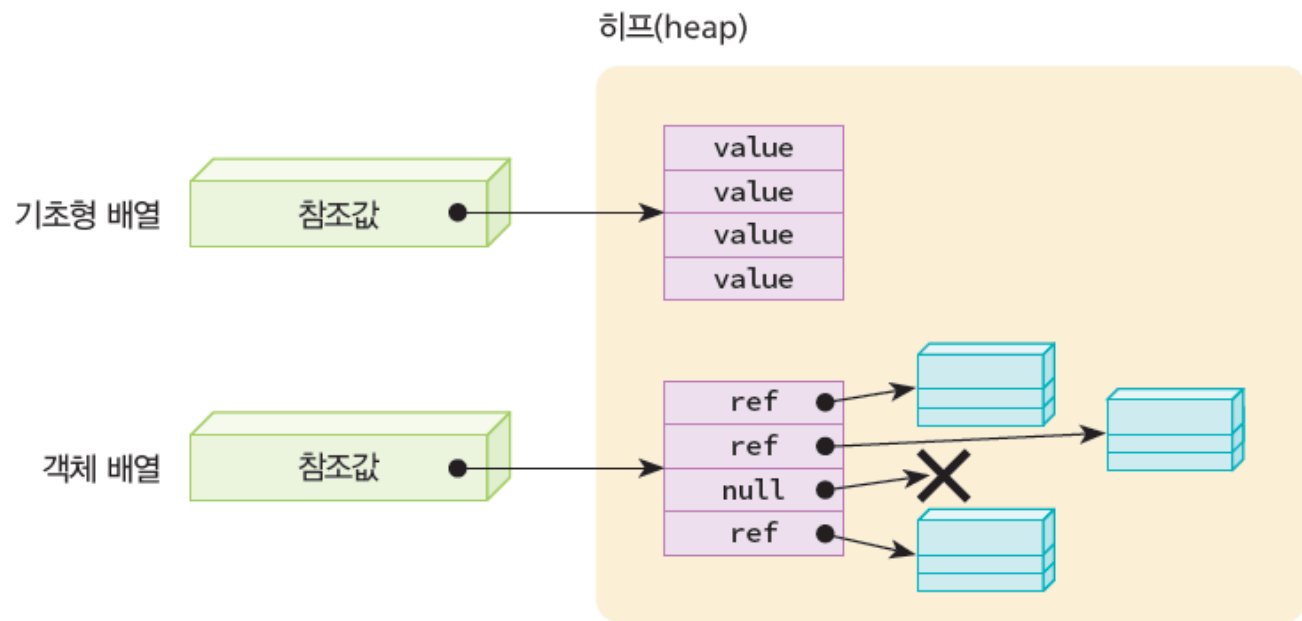
```
class Single {  
    private static Single instance = new Single();  
    private Single() {} // 전용 생성자  
  
    public static Single getInstance() {  
        return instance;  
    }  
}  
  
public class SingleTest {  
    public static void main(String[] args) {  
        Single obj1 = Single.getInstance();  
        Single obj2 = Single.getInstance();  
        System.out.println(obj1);  
        System.out.println(obj2);  
    }  
}
```

test1.Single@4926097b
test1.Single@4926097b

여러번 생성해도 하나의 객체만 생성됨.

객체배열

- 객체를 저장하는 배열
- 객체 배열에는 객체에 대한 참조값이 저장되어 있다.



예제

- Rect 객체를 저장하는 배열을 생성해보자.

```
class Rect {  
    int width, height;  
  
    public Rect(int w, int h){  
        this.width=w;  
        this.height=h;  
    }  
    double getArea() {  
        return (double)width*height;  
    }  
}
```

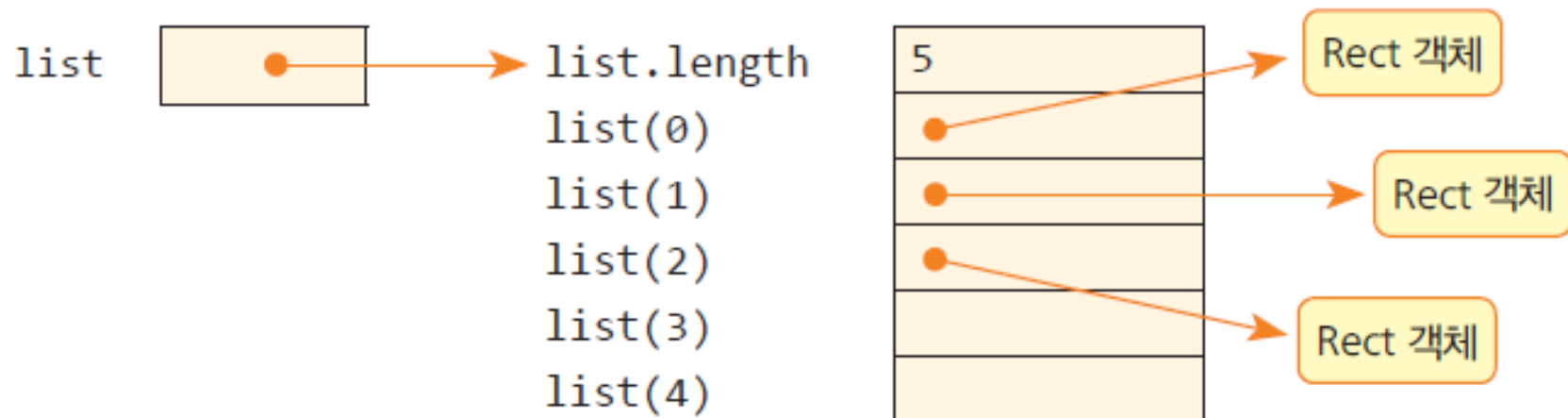
예제

```
public class RectArrayTest {  
    public static void main(String[] args) {  
        Rect[] list;  
        list = new Rect[5]; → 참고 변수용 5칸 배열.  
  
        for(int i=0;i < list.length; i++)  
            list[i] = new Rect(i, i); → 실질적으로 각 칸에 객체 생성.  
  
        for(int i=0;i < list.length; i++)  
            System.out.println(i+"번째 사각형의 면적="+list[i].getArea());  
    }  
}
```

0번째 사각형의 면적=0.0
1번째 사각형의 면적=1.0
2번째 사각형의 면적=4.0
3번째 사각형의 면적=9.0
4번째 사각형의 면적=16.0

객체 배열의 모습

```
for(int i=0; i < list.length; i++)  
    list[i] = new Rect(i, i);
```



예제: 객체 배열 만들기

- 영화의 제목과 감독을 저장하는 Movie 클래스를 정의한다. 몇 개의 영화 정보를 Movie 객체 배열에 저장하고 다시 출력하는 프로그램을 작성해보자.

=====

제목: 백투더퓨처

감독: 로버트 저메키스

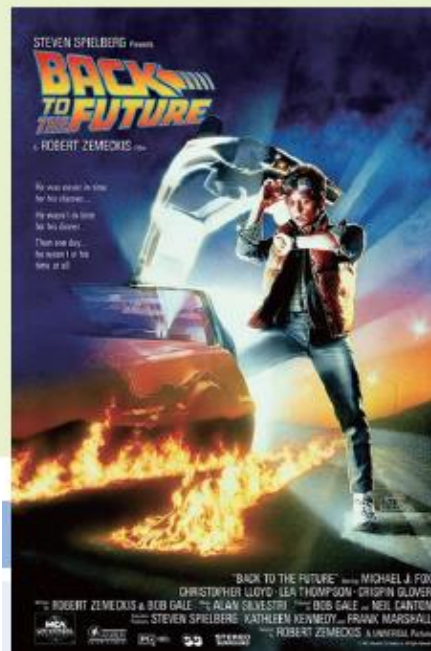
=====

=====

제목: 티파니에서 아침을

감독: 에드워드 블레이크

=====



예제: 객체 배열 만들기

```
import java.util.Scanner;
class Movie {
    String title, director;
    static int count;
    public Movie(String title, String director){
        this.title=title;
        this.director=director;
        count++;
    }
}
```

```
public class MovieArrayTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        Movie [] list= new Movie[10];
        list[0] = new Movie("백투더퓨처", "로버트 저메키스");
        list[1] = new Movie("티파니에서 아침을", "에드워드 블레이크");

        for(int i=0;i < Movie.count; i++) {
            System.out.println("=====");
            System.out.println("제목: "+list[i].title);
            System.out.println("감독: "+list[i].director);
            System.out.println("=====");
        }
    }
}
```

동적 객체 배열

- 자바의 표준 배열은 크기가 결정되면 변경하기 힘들다. 따라서 실제 프로그래밍에서는 동적 배열을 많이 사용한다. 동적 배열 중에서 ArrayList에 객체들을 저장해보자. 첫 번째 예제로 여행지를 저장하고 있다가 랜덤으로 하나를 선택하여서 사용자에게 추천하는 프로그램을 작성해보자.

여행지 추천 시스템입니다.

추천 여행지는 괌

동적 객체 배열

```
import java.util.ArrayList;

public class ArrayListTest {
    public static void main(String[] args) {

        ArrayList<String> list = new ArrayList<String>();    // (1)
        list.add("홍콩");
                                // (2)
        list.add("싱가포르");
        list.add("괌");
        list.add("사이판");
        list.add("하와이");

        System.out.println("여행지 추천 시스템입니다.");
        int index = (int) (Math.random()*list.size());
        System.out.println("추천 여행지는 "+list.get(index));    // (3)
    }
}
```

예제: 연락처 정보 저장하기

- 이름과 전화번호를 Person이라는 클래스로 정의하고 Person 객체를 저장하는 동적 배열을 생성해보자. 몇 사람의 연락처를 저장해본다.

(홍길동,01012345678)

(김유신,01012345679)

(최자영,01012345680)

(김영희,01012345681)

예제: 연락처 정보 저장하기

```
import java.util.ArrayList;
class Person {
    String name;
    String tel;
    public Person(String name, String tel) {
        this.name = name;
        this.tel = tel;
    }
}
public class ArrayListTest2 {
    public static void main(String[] args) {
        ArrayList<Person> list = new ArrayList<Person>(); // (1)
        list.add(new Person("홍길동", "01012345678"));
        list.add(new Person("김유신", "01012345679"));
        list.add(new Person("최자영", "01012345680"));
        list.add(new Person("김영희", "01012345681"));
        for (Person obj : list)
            System.out.println("(" + obj.name + "," + obj.tel + ")");
    }
}
```

Mini Project: 전기차 클래스

- 전기 자동차를 클래스로 작성해보자. 자동차는 완전(100%) 배터리로 시작한다. 자동차를 운전할 때마다 1km를 주행하고 배터리의 10%를 소모한다. 전기 자동차에는 2가지 정보를 보여주는 디스플레이가 있다. 주행한 총 거리는 "주행거리: ...km". 남은 배터리 충전량은 "배터리: ...%"와 같이 표시된다.

1. ECar.getInstance() : 새로운 자동차를 생성하는 정적 메소드이다. 새로운 전기 자동차 인스턴스를 반환한다.

```
ECar car = ECar.getInstance();
```

2. dispDistance(): 주행 거리를 표시한다.
3. dispBattery(): 배터리 백분율 표시한다.
4. drive(): 한번 호출될 때마다 1km를 운행한다.

객체

Mini Project: 책 정보 저장

- 사용자가 읽은 책과 평점을 저장하는 객체 배열을 생성해보자. 다음과 같은 메뉴가 제공된다.

```
=====
1. 책 등록
2. 책 검색
3. 모든 책 출력
4. 종료
=====

번호를 입력하시오: 1
제목: The C Language
평점: 9
...
```

객체 매서드에서
인스턴스 멤버를 사용하고자 하면
객체를 먼저 생성 후
참조변수로 접근해야 한다.

정리: 같은 클래스의 static 메서드에서
인스턴스 메서드 호출 불가.
인스턴스 생성을 통한 호출은 가능하다.