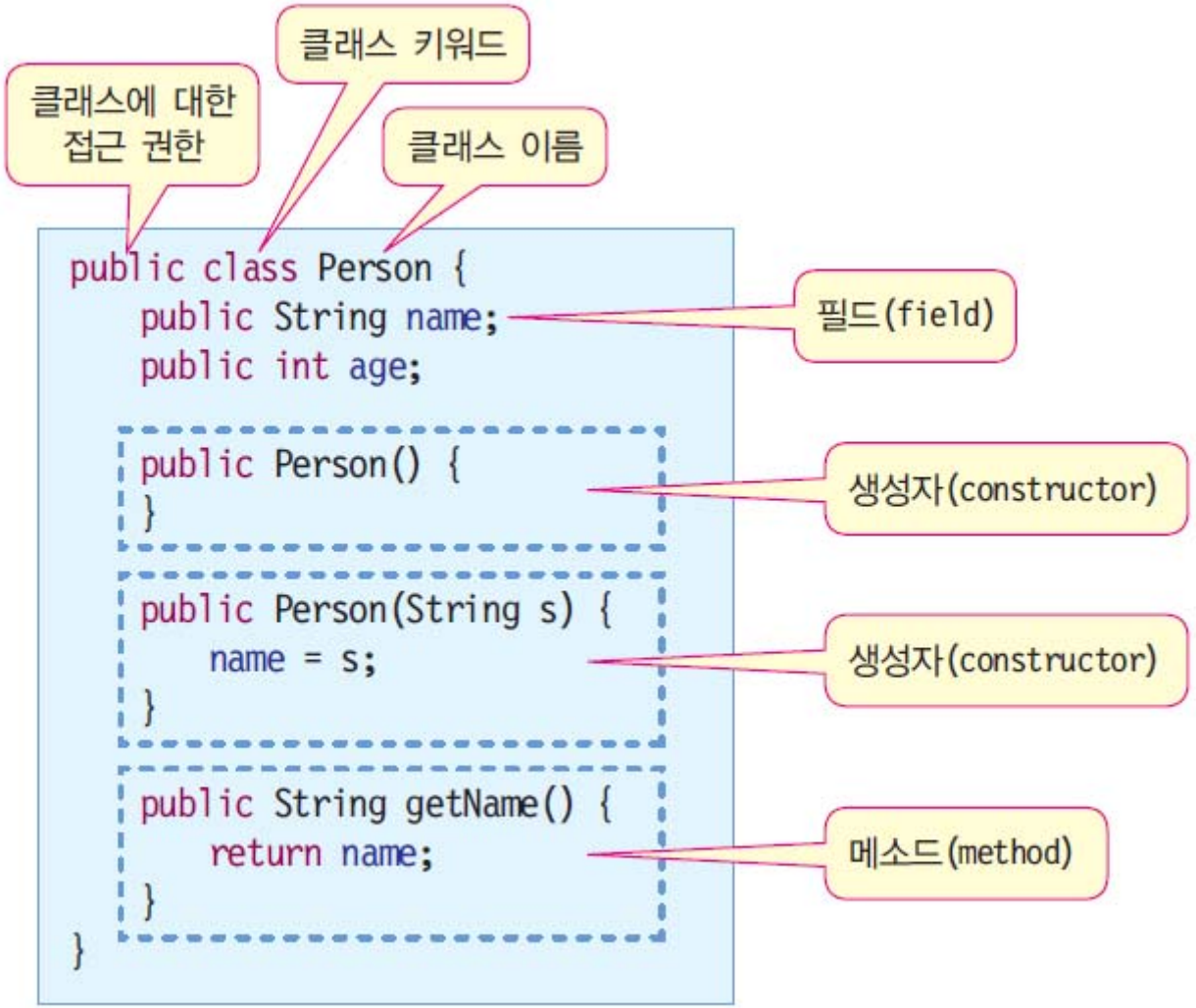
객체와 클래스 II

남 광 우

클래스 구성


클래스 키워드
클래스에 대한 접근 권한
클래스 이름
public class Person {
public String name;
public int age;
public Person() {
}
public Person(String s) {
name = s;
}
public String getName() {
return name;
}
}
필드(field)
생성자(constructor)
생성자(constructor)
메소드(method)

클래스 선언

- 클래스 접근 권한, public
  - public 접근 권한은 다른 클래스들이 이 클래스에 대해 사용 혹은 접근이 가능함을 의미
- class Person
  - Person이라는 이름의 클래스 정의
  - class 다음에 클래스의 이름을 선언
  - 클래스는 {로 시작하여 }로 닫으며 이곳에 모든 멤버 필드와 메소드 구현
- 필드(field)
  - 값을 저장할 멤버 변수를 선언
  - 멤버 변수 혹은 필드라고 함
  - 필드 앞에 붙은 접근 지정자 public
    - 이 필드가 다른 클래스에서 접근될 수 있도록 공개한다는 의미
- 생성자(constructor)
  - 클래스의 이름과 동일한 메소드
  - 클래스의 객체가 생성될 때만 호출되는 메소드
- 메소드(method)
  - 메소드는 함수이며 객체의 행위를 구현
  - 메소드 앞에 붙은 접근 지정자 public
    - 메소드가 다른 클래스에서 접근될 수 있도록 공개한다는 의미

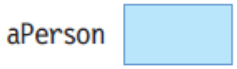
객체 생성

- 객체 생성
  - 객체는 new 키워드를 이용하여 생성
    - new는 객체의 생성자 호출
- 객체 생성 과정
  1. 객체에 대한 레퍼런스 변수 선언
  2. 객체 생성

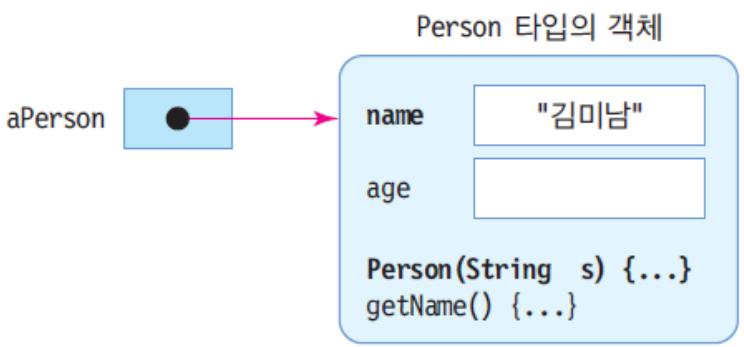
```
public static void main (String args[]) {
        Person aPerson;                    // 레퍼런스 변수 aPerson 선언
        aPerson = new Person("김미남");     // Person 객체 생성

        aPerson.age = 30;                  // 객체 멤버 접근
        int i = aPerson.age;               // 30
        String s = aPerson.getName();      // 객체 메소드 호출
}
```

객체 생성 및 사용 예

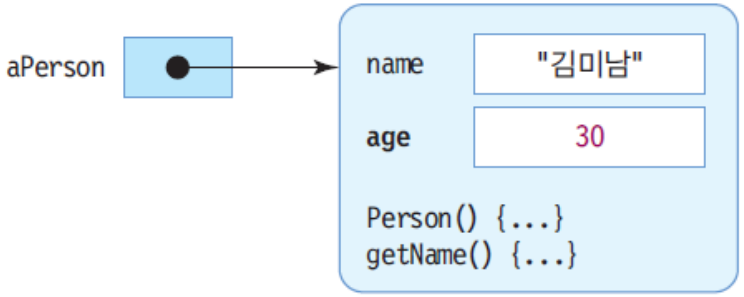
(1) Person aPerson;



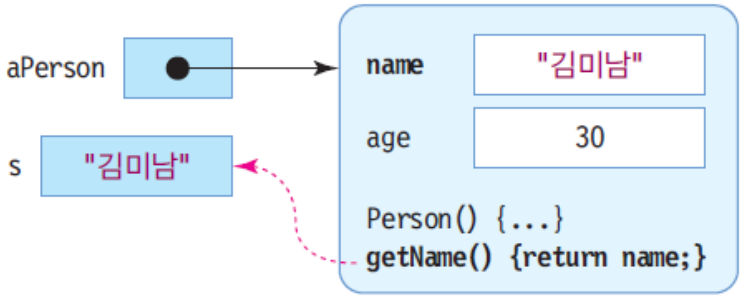
(2) aPerson = new Person("김미남");



(3) aPerson.age = 30;



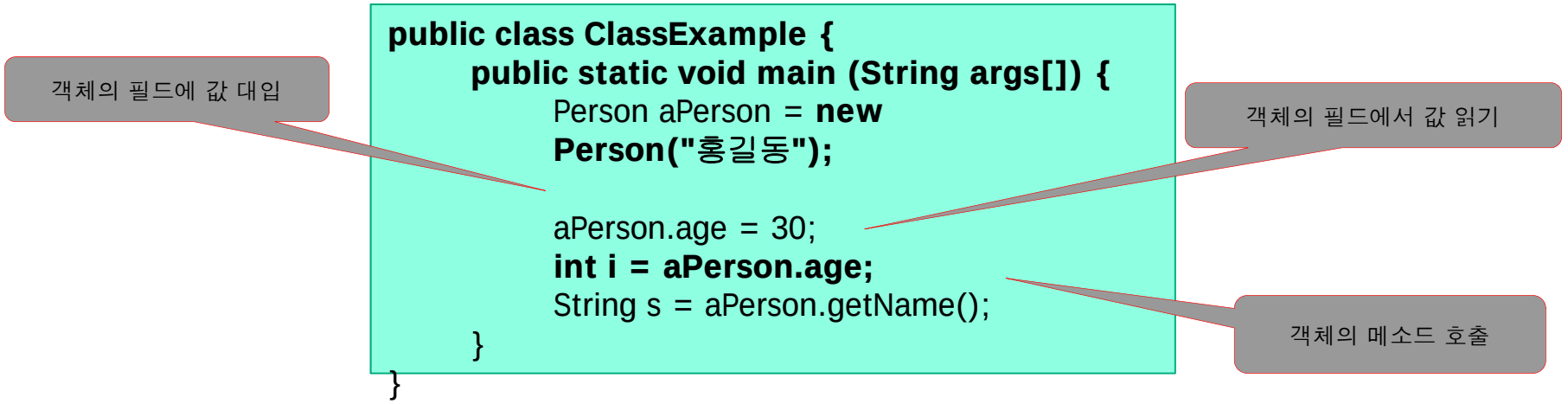
(4) String s = aPerson.getName();



객체의 활용

- 객체의 멤버 접근

객체레퍼런스 . 멤버



예제 4-2 : 지수 클래스 MyExp 만들기

클래스 MyExp를 작성하라. MyExp는 지수값을 표현하는 클래스로서 두 개의 정수형 멤버 필드 base와 exp를 가진다. 2^3의 경우 base는 2이며, exp는 3이 된다. base와 exp는 양의 정수만을 가지는 것으로 가정한다. 또한 MyExp는 정수값을 리턴하는 getValue()라는 멤버 메소드를 제공한다. getValue()는 base와 exp 값으로부터 지수를 계산하여 정수 값으로 리턴한다. 예를 들어 MyExp객체의 base 필드가 2이고 exp가 3이라면 getValue()는 8을 리턴한다.

```
public class MyExp {
    int base;
    int exp;
    int getValue() {
        int res=1;
        for(int i=0; i<exp; i++)
            res = res * base;
        return res;
    }
    public static void main(String[] args) {
        MyExp number1 = new MyExp();
        number1.base = 2;
        number1.exp = 3;
        MyExp number2 = new MyExp();
        number2.base = 3;
        number2.exp = 4;

        System.out.println("2의 3승 = " + number1.getValue());
        System.out.println("3의 4승 = " + number2.getValue());
    }
}
```

```
2의 3승 = 8
3의 4승 = 81
```

예제 4-1 : 상품 하나를 표현하는 클래스 Goods 만들기

상품 하나를 표현하는 클래스 Goods를 작성하라.
상품은 String 타입의 name,
int 타입의 price, numberOfStock, sold 등 네 개의 필드를 갖는다.
Goods 클래스 내에 main() 메소드를 작성하여 Goods 객체를 하나 생성하고 이 객체에 대한 레퍼런스 변수 명을 camera로 하라. 그리고 나서 camera의 상품 이름(name 필드)을 "Nikon", 값(price)을 400000, 재고 갯수(numberOfStock)를 30, 팔린 개수(sold)를 50으로 설정하라. 그리고 설정된 이들 값을 화면에 출력하라.

```
public class Goods {
    String name;
    int price;
    int numberOfStock;
    int sold;

    public static void main(String[] args) {
        Goods camera = new Goods();

        camera.name = “Nikon”;
        camera.price = 400000;
        camera.numberOfStock = 30;
        camera.sold = 50;

        System.out.println("상품 이름:" + camera.name);
        System.out.println("상품 가격:" + camera.price);
        System.out.println("재고 수량:" + camera.numberOfStock);
        System.out.println("팔린 수량:" + camera.sold);
    }
}
```

```
상품 이름:Nikon
상품 가격:400000
재고 수량:30
팔린 수량:50
```

예제 4-1 : 상품 하나를 표현하는 클래스 Goods 만들기

생성자의 추가

```
public class Goods {
    String name;
    int price;
    int numberOfStock;
    int sold;
        Goods(String name, int price, int numberOfStack, int sold) {
        this.name = name;
        this.price = price;
        this.numberOfStock = numberOfStock;
        this.sold = sold;
    }

    public static void main(String[] args) {
        Goods camera1 =  new Goods("Nikon", 400000, 30, 50);
        Goods camera2 = null;

        camera2 =  new Goods("Nikon", 400000, 30, 50);


        System.out.println("상품 이름:" + camera.name);
        System.out.println("상품 가격:" + camera.price);
        System.out.println("재고 수량:" + camera.numberOfStock);
        System.out.println("팔린 수량:" + camera.sold);
    }
}
```

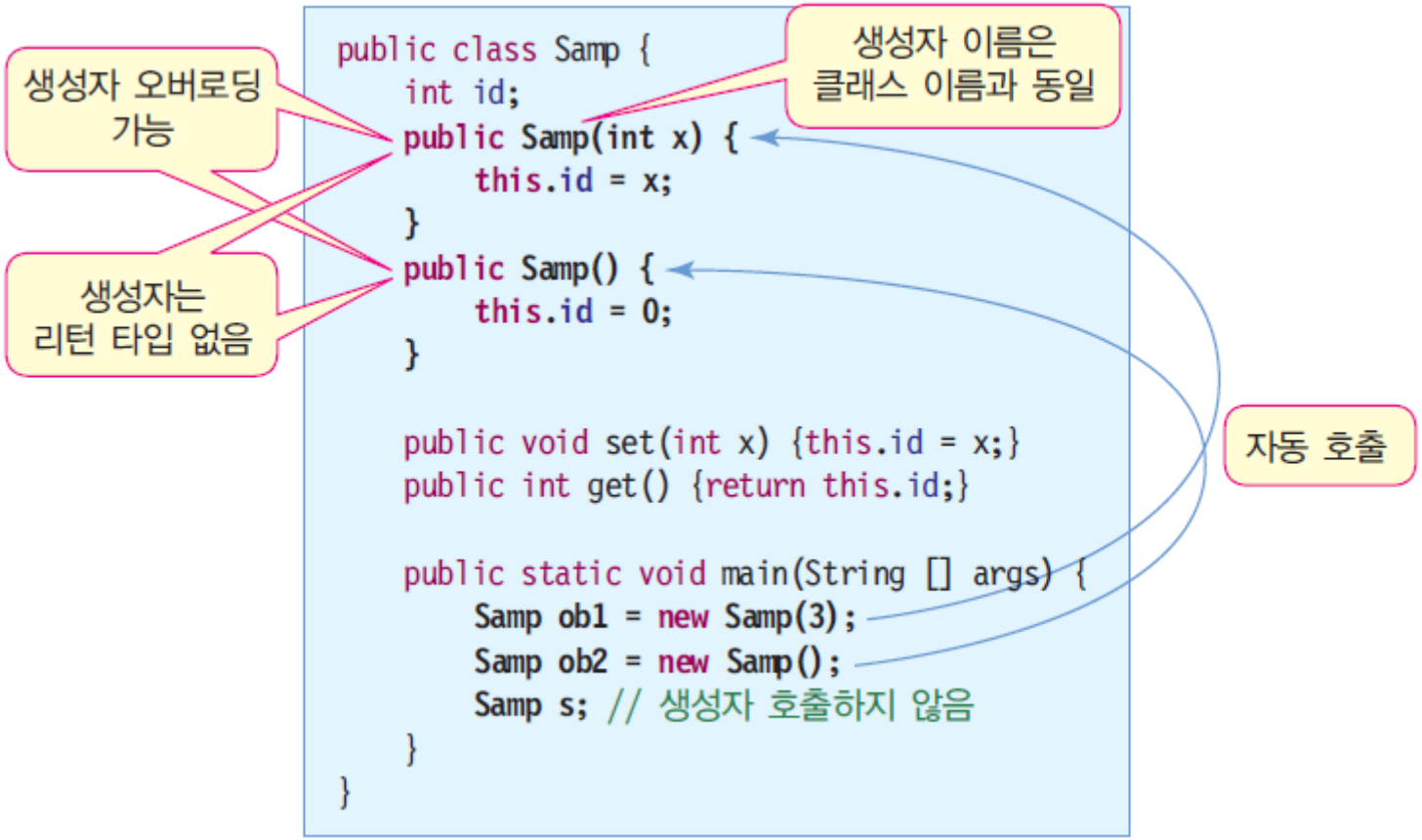
```
상품 이름:Nikon
상품 가격:400000
재고 수량:30
팔린 수량:50
```

생성자(Constructor)

- 생성자 메서드의 사용 이유
  - 할당받은 멤버변수를 초기화할 때
  - 객체가 생성되기 전의 미리 해야 할 작업이 있을 때

- 생성자 메소드
  - 객체의 초기화를 위해 객체가 생성될 때 호출되어 실행
  - 생성자 메서드의 이름은 클래스의 이름과 동일
  - new 연산자가 호출된 직후에 호출
    - new 연산자가 메모리를 생성하면 멤버변수들이 메모리를 할당받음
    - 따라서 변수들에 대한 초기화 작업이 가능해짐
  - 클래스에 기본 생성자 메서드가 없다면 컴파일러가 자동 생성
    - 컴파일러가 생성해주는 기본 생성자 메서드는 매개변수가 없고 블록이 빔
  - 메서드이지만 유일하게 리턴형이 없음

생성자 정의와 생성자 호출



예제 4-5 : 생성자 정의와 호출

클래스 Book을 String title, String author, int ISBN의 3개의 필드를 갖도록 정의하라.

```
public class Book {
   String title;
   String author;
   int ISBN;
   public Book(String title, String author, int ISBN) {
      this.title = title;
      this.author = author;
      this.ISBN = ISBN;
   }

   public static void main(String [] args) {
      Book javaBook = new Book("Java JDK", "황기태", 3333);
   }
}
```

기본 생성자

- 기본 생성자(default constructor)
  - 클래스에 생성자가 하나도 선언되지 않은 경우
  - 컴파일러에 의해 자동으로 생성
    - 인자 없는 생성자
    - 아무 작업 없이 단순 리턴
  - 디폴트 생성자라고도 부름

```
class DefaultConstructor{
    int x;
    public  void setX(int x) {this.x = x;}
    public  int getX() {return x;}

    public static void main(String [] args) {
        DefaultConstructor p= new DefaultConstructor();
        p.setX(3);
    }
}
```

개발자가 작성한 코드

```
class DefaultConstructor{
    int x;
    public  void setX(int x) {this.x = x;}
    public  int getX() {return x;}

    public DefaultConstructor() { }

    public static void main(String [] args) {
        DefaultConstructor p= new DefaultConstructor();
        p.setX(3);
    }
}
```

컴파일러에 의해 자동 삽입된 기본 생성자

컴파일러가 자동으로 기본 생성자를 삽입한 코드

기본 생성자가 자동 생성되지 않는 경우

- 클래스에 생성자가 하나라도 존재하면 자동으로 기본 생성자가 생성되지 않음

```
class DefaultConstructor{
   int x;
   public  void setX(int x) {this.x = x;}
   public  int getX() {return x;}

   public DefaultConstructor(int x) {
      this.x = x;
   }
   public static void main(String [] args) {
      DefaultConstructor p1= new DefaultConstructor(3);
      int n = p1.getX();

      DefaultConstructor p2= new DefaultConstructor();
      p2.setX(5);
   }
}
```

컴파일러가 기본 생성자를 자동 생성하지 않음

public DefaultConstructor() { }

컴파일 오류.
해당하는 생성자가 없음 !!!

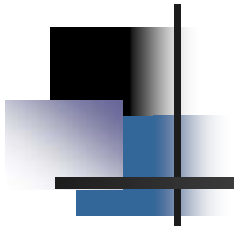
this(), 생성자에서 다른 생성자 호출

- this()
  - 같은 클래스의 다른 생성자 호출
  - 생성자 내에서만 사용 가능
    - 다른 메소드에서는 사용 불가
  - 반드시 생성자 코드의 제일 처음에 수행

```
public class Book {
   String title;
   String author;
   int ISBN;

   public Book(String title, String author, int ISBN) {
   this.title = title;
   this.author = author;
   this.ISBN = ISBN;
   }
   public Book(String title, int ISBN) {
      this(title, "Anonymous", ISBN);
   }
   public Book() {
      this(null, null, 0);
      System.out.println("생성자가 호출되었음");
   }
   public static void main(String [] args) {
      Book javaBook = new Book("Java JDK", "황기태", 3333);
      Book holyBible = new Book("Holy Bible", 1);
      Book emptyBook = new Book();
   }
}
```

title = "Holy Bible"
author = "Anonymous"
ISBN = 1

title = "Holy Bible"
ISBN = 1



this() 사용 실패 예

```
public Book() {
   System.out.println("생성자가 호출되었음");
   this(null, null, 0); // 생성자의 첫 번째 문장이 아니기 때문에 컴파일 오류
}
```

객체 배열

- 객체 배열 생성 및 사용

```
Person[] pa;
pa = new Person[10];
for(int i=0; i<pa.length; i++) {
   pa[i] = new Person();
   pa[i].age = 30 + i;
}
```

배열에 대한 레퍼런스 선언

레퍼런스 배열 생성

배열의 원소 객체 생성

객체 배열 사용

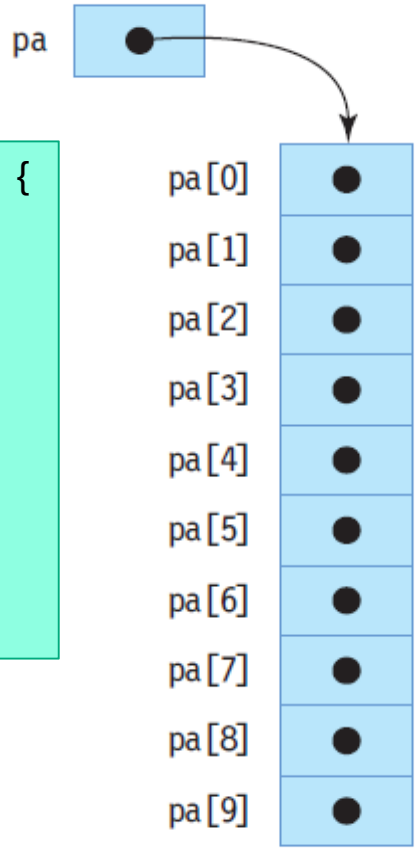
```
for(int i=0; i<pa.length; i++) // 배열 pa의 모든 원소 객체의 age를 출력한다.
   System.out.print(pa[i].age + " ");
```

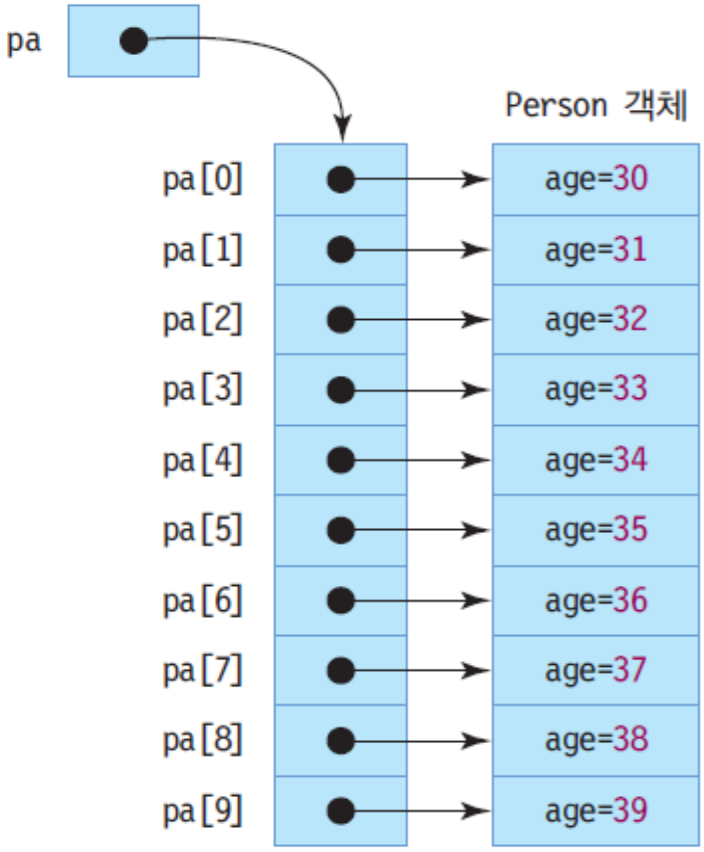
객체 배열 선언과 생성 사례

```
Person[] pa;
```

```
pa = new Person[10];
```

```
for(int i=0; i<pa.length; i++) {
    pa[i] = new Person();
    pa[i].age = 30 + i;
}
```






```
public static void main(String [] args) {
   Person[] pa;
   pa = new Person[10];
   for (int i=0;i<pa.length;i++) {
      pa[i] = new Person();
      pa[i].age = 30 + i;
   }

   for (int i=0;i<pa.length;i++)
      System.out.print(pa[i].age+" ");
}
```

```
30 31 32 33 34 35 36 37 38 39
```

예제 4-3 : 객체 배열 생성

java.util.Scanner 클래스를 이용하여 상품을 입력 받아 Goods 객체를 생성하고 이들을 Goods 객체 배열에 저장하라. 상품 즉 Goods 객체를 3개 입력 받으면 이들을 모두 화면에 출력하라.

```
import java.util.Scanner;

public class GoodsArray {
    public static void main(String[] args) {
        Goods [] goodsArray;
        goodsArray = new Goods [3];

         Scanner s = new Scanner(System.in);
        for(int i=0; i<goodsArray.length; i++) {
            String name = s.next();
            int price = s.nextInt();
            int n = s.nextInt();
            int sold = s.nextInt();
            goodsArray[i] = new Goods(name, price, n, sold);
        }

        for(int i=0; i<goodsArray.length; i++) {
            System.out.print(goodsArray[i].getName()+" ");
            System.out.print(goodsArray[i].getPrice()+" ");
            System.out.print(goodsArray[i].getNumberOfStock()+"
");
            System.out.println(goodsArray[i].getSold());
        }
    }
}
```

```
class Goods {
    private String name;
    private int price;
    private int numberOfStock;
    private int sold;

    Goods(String name, int price, int numberOfStack, int sold) {
        this.name = name;
        this.price = price;
        this.numberOfStock = numberOfStock;
        this.sold = sold;
    }

    String getName() {return name;}
    int getPrice() {return price;}
    int getNumberOfStock() {return numberOfStock;}
    int getSold() {return sold;}
}
```

```
콜라 500 10 20
사이다 1000 20 30
맥주 2000 30 50
콜라 500 10 20
사이다 1000 20 30
맥주 2000 30 50
```

키 입력 부분

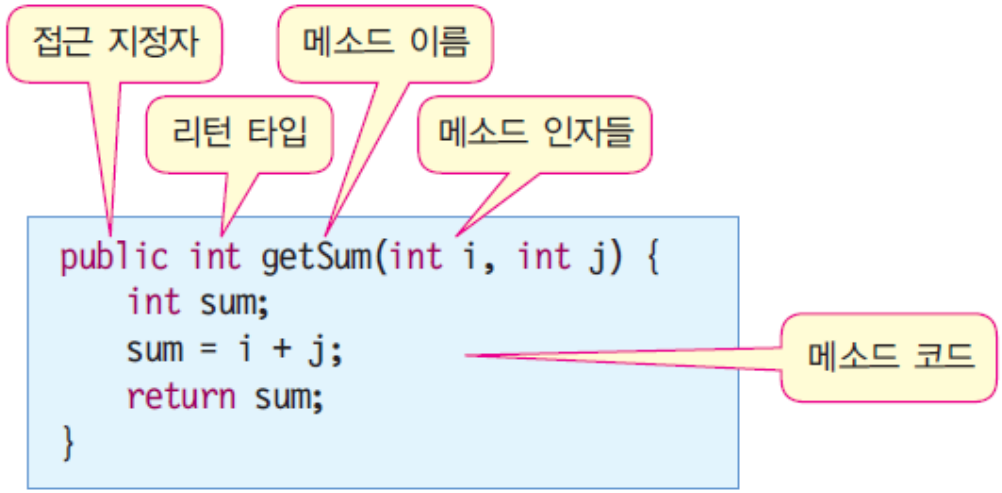
실습 : Song 클래스 만들기

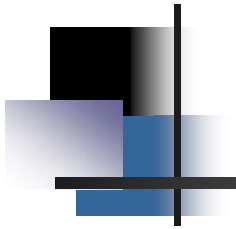
- 다음의 조건을 만족하는 Song 클래스를 만드시오
  - 노래의 제목을 나타내는 title
  - 가수를 나타내는 artist
  - 노래가 속한 앨범 제목을 나타내는 album
  - 노래의 작곡가를 나타내는 composer(작곡가는 여러 명 가능)
  - 노래가 발표된 연도를 나타내는 year
  - 노래가 속한 앨범의 트랙번호를 나타내는 track

  - 생성자는 모든 필드를 초기화하는 생성자, 빈생성자()후 set* 2개
  - 노래의 정보를 출력하는 show() 메소드
  - main()에서 Song 객체 5개를 저장할수 있는 배열을 생성한후
    - 아는노래로 Song 객체를 5개 생성한후
    - show() 하시오

메소드 형식

- 메소드
  - 메소드는 함수이며 함수 만드는 방법과 동일하게 작성
  - 모든 메소드는 반드시 클래스 안에 있어야 함(캡슐화 원칙)
- 메소드 구성 형식
  - 접근 지정자
    - public. private, protected, default(접근 지정자 생략된 경우)
  - 리턴 타입
    - 메소드가 반환하는 결과값의 데이터 타입



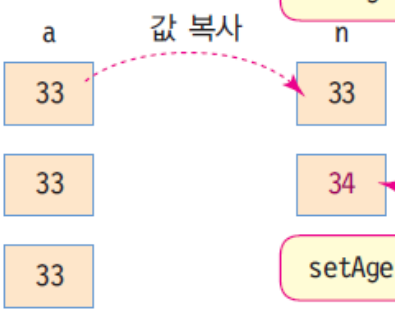


인자 전달 – call by value

- 자바의 인자 전달 방식
  - 값에 의한 호출(call by value)
  - 기본 타입의 값을 전달하는 경우
    - 값이 복사되어 전달
    - 매개 변수 값이 변경되어도 호출한 인자 값은 변경되지 않음
  - 객체 혹은 배열을 전달하는 경우
    - 객체나 배열의 레퍼런스 만 전달됨
      - 객체 혹은 배열이 통째로 복사되어 전달되는 것이 아님
    - 매개 변수와 호출한 실인자가 서로 객체 혹은 배열 공유

call by value : 기본 데이터의 값 전달 사례

```
public class CallByValue {
    public static void main (String args[]) {
        Person aPerson = new Person("홍길동");
        int a = 33;

        aPerson.setAge(a);

        System.out.println(a);
    }
}
```




```
public class Person {
    public String name;
    public int age;

    public Person(String s) {
        name = s;
    }

    public void setAge(int n) {
        age = n;
        n++;
    }
}
```

setAge()가 호출되면 매개변수 n이 생성된다.

setAge()가 끝나면 n은 사라진다.

```
33
```

call by value : 객체 전달 사례

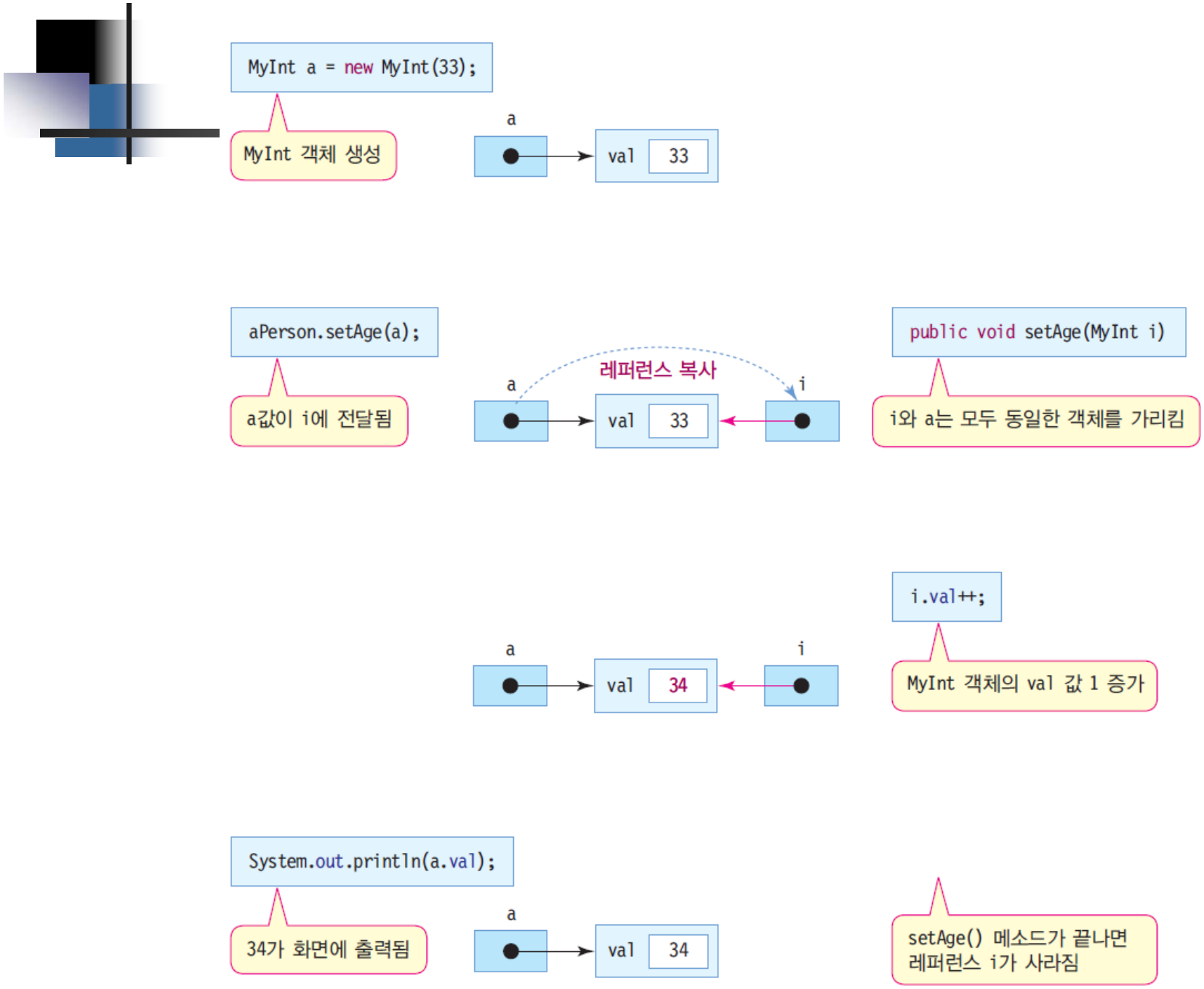
```
class MyInt {
    int val;
    MyInt(int i) {
        val = i;
    }
}
public class CallByValueObject {
    public static void main(String args[]) {
        Person aPerson = new Person("홍길동");
        MyInt a = new MyInt(33);

        aPerson.setAge(a);

        System.out.println(a.val);
    }
}
```

호출

```
public class Person {
    public String name;
    public int age;
    public Person(String s) {
        name = s;
    }

    public void setAge(MyInt i) {
        age = i.val;
        i.val++;
    }

}
```

34

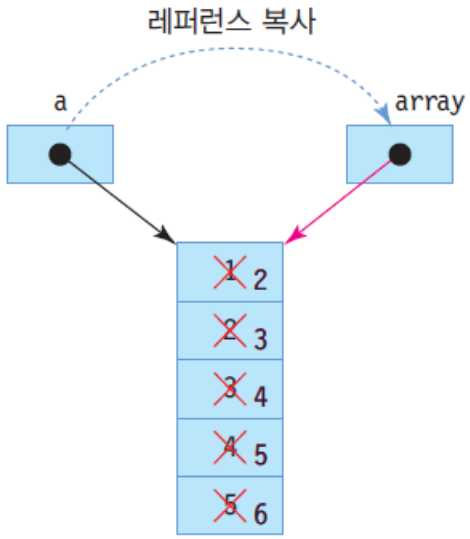
* 객체가 복사되어 전달되는 것이 아님
객체에 대한 레퍼런스 만이 복사되어 전달


MyInt a = new MyInt(33);
MyInt 객체 생성
a
val
33
aPerson.setAge(a);
a값이 i에 전달됨
레퍼런스 복사
a
i
val
33
public void setAge(MyInt i)
i와 a는 모두 동일한 객체를 가리킴
i.val++;
a
i
val
34
MyInt 객체의 val 값 1 증가
System.out.println(a.val);
34가 화면에 출력됨
a
val
34
setAge() 메소드가 끝나면
레퍼런스 i가 사라짐

call by value : 배열 전달 사례

- 인자로 배열을 전달하면 배열의 레퍼런스만이 전달됨

```
public class ArrayParameter {

    public static void main(String args[]) {
        int a[] = {1, 2, 3, 4, 5};


        increase(a);


        for(int i=0; i<a.length; i++)
            System.out.print(a[i]+" ");
    }
```




```
    static void increase(int[] array) {
        for(int i=0; i<array.length; i++) {
            array[i]++;
        }
    }
}
```

2 3 4 5 6

예제 4-4 : 배열의 전달

char 배열을 메소드의 인자로 전달하여 배열 속의 공백(' ')문자를 ',' 로 대치하는 프로그램을 작성하라.

```
public class ArrayParameter {
   static void replaceSpace(char a[]) {
      for (int i = 0; i < a.length; i++)
         if (a[i] == ' ')
            a[i] = ',';
   }
   static void printCharArray(char a[]) {
      for (int i = 0; i < a.length; i++)
         System.out.print(a[i]);
      System.out.println();
   }
   public static void main (String args[]) {
      char c[] = {'T','h','i','s',' ','i','s',' ','a',' ','p','e','n','c','i','l','.'};
      printCharArray(c);
      replaceSpace(c);
      printCharArray(c);
   }
}
```

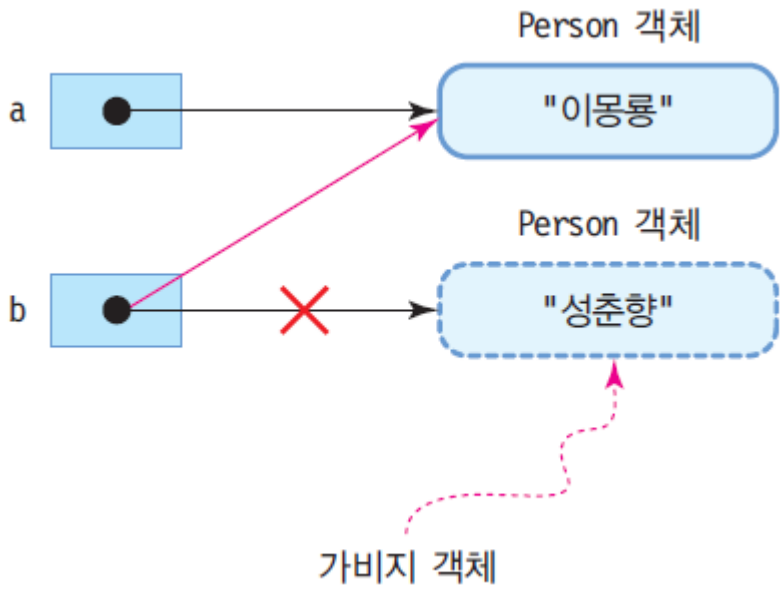
```
This is a pencil.
This,is,a,pencil.
```

객체의 소멸과 가비지

- 객체 소멸
  - new에 의해 생성된 객체 메모리를 자바 가상 기계에게 되돌려 주는 행위
  - 소멸된 객체 공간은 가용 메모리에 포함
- 자바는 객체 삭제 기능 없음
  - 개발자에게는 매우 다행스러운 기능
    - C/C++에서는 할당 받은 객체를 개발자가 프로그램 내에서 삭제해야 함
- 가비지
  - 가비지
    - 자신에 대한 레퍼런스가 없는 객체
  - 가비지 켈렉션
    - 자바 가상 기계의 가비지 컬렉터가 자동으로 가비지를 수집하여 반환

가비지 사례

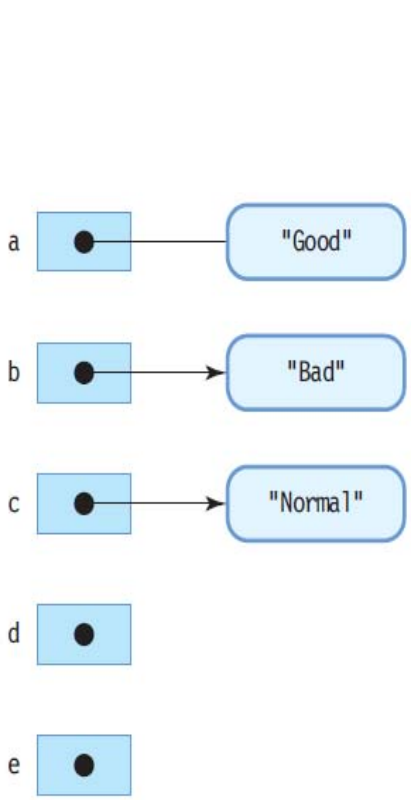
```
Person a, b;
a = new Person("이몽룡");
b = new Person("성춘향");
b = a; // b가 가리키던 객체는 가비지가 됨
```



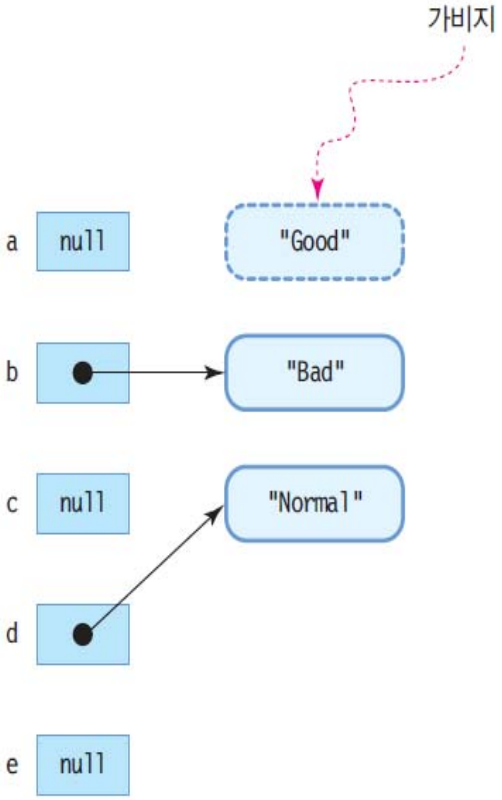
예제 4-6 : 가비지 발생

다음 소스에서 언제 가비지가 발생하는지 설명하라.

```
public class GarbageEx {
    public static void main(String[] args) {
        String a = new String("Good");
        String b = new String("Bad");
        String c = new String("Normal");
        String d, e;
        a = null;
        d = c;
        c = null;
    }
}
```



(a) 초기 객체 생성 시(라인 6까지)



(b) 코드 전체 실행 후

가비지 컬렉션

- 가비지 컬렉션
  - 자바에서는 가비지 자동 회수
    - 가용 메모리 공간으로 확보
  - 가비지 컬렉터(garbage collector)에 의해 자동 수행
- 개발자에 의한 강제 가비지 컬렉션
  - System 또는 Runtime 객체의 gc() 메소드 호출

```
System.gc(); // 가비지 컬렉션 작동 요청
```

    - 자바 가상 기계에 강력한 가비지 컬렉션을 요청
      - 그러나 자바 가상 기계가 가비지 컬렉션 시점을 전적으로 판단

생성자(Constructor)의 사용 예 : 같은 파일

```
//Student.java – 매개변수없는 생성자
public class Student {
   private String name = null;
   private String address = null;
   private int age = 0;
   public Student() {
      this.name = "이름없음";
      this.address="주소없음";
      this.age=0;
   }
   public void setStudent(String n, String addr, int a)
  {
      this.name = n;
      this.address = addr;
      this.age = a;
   }
   public String getName() { return name }

   public String getStudentString() {
      String str = name + "  " + address + "   " + age;
      return str;
   }
}
```

```
public static void main(String[] args) {
    Student st = new Student();
    String s;
    s = st.getStudentString();
    System.out.println(s);
    st.setStudent("홍길동", "군산시", 25);
    s = st.getStudentString();
    System.out.println(s);
 }
```

```
결과화면>
C:₩04>javac Student.java
C:₩04>java Student
이름없음  주소없음   0
홍길동  군산시   25
```

- 실행결과
  - 객체생성시 생성자메서드 실행
    - 각각의 멤버변수에 값을 할당
  - 멤버변수의 값을 도스창에 출력
    - 생성자메서드에 의해 값이 할당됨 증명

생성자(Constructor)의 사용 예 : 다른 파일

```
//Student.java – 매개변수없는 생성자
public class Student {
   private String name = null;
   private String address = null;
   private int age = 0;
   public Student() {
      this.name = "이름없음";
      this.address="주소없음";
      this.age=0;
   }
   public void setStudent(String n, String addr, int a)
  {
      this.name = n;
      this.address = addr;
      this.age = a;
   }
   public String getName() { return name }

   public String getStudentString() {
      String str = name + "  " + address + "   " + age;
      return str;
   }
}
```

```
// StudentTest.java
public class StudentTest{

  public static void main(String[] args) {
     Student st = new Student();
     String s;
     s = st.getStudentString();
     System.out.println(s);
     st.setStudent("홍길동", "군산시", 25);
     s = st.getStudentString();
     System.out.println(s);
   }
}
```

결과화면>
C:₩04>javac Student.java
C:₩04>javac StudentTest.java
C:₩04>java StudentTest
이름없음 주소없음 0
홍길동 군산시 25

- 실행결과
  - 객체생성시 생성자메서드 실행
    - 각각의 멤버변수에 값을 할당
  - 멤버변수의 값을 도스창에 출력
    - 생성자메서드에 의해 값이 할당됨 증명

생성자(Constructor)의 사용 예

```
//Student2.java – 매개변수 있는 생성자
public class Student2 {
   private String name = null;
   private String address = null;
   private int age = 0;

   public Student() {
      this.name = "이름없음";
      this.address="주소없음";
      this.age=0;
   }

   public Student(String n, String addr, int a) {
      this.name = n;
      this.address=addr;
      this.age=a;
   }
   public String getStudentString() {
      String str = name + "  " + address + "
      return str;
   }
}
```

```
// StudentTest2.java
public class StudentTest2{

  public static void main(String[] args) {
      Student st1 = new Student();
      Student st2 = new Student2("홍길동",
                    "군산", 23);

      System.out.println(st1.getStudentString());
      System.out.println(st2.getStudentString());
   }
}
```

```
결과화면>
C:₩04>javac Student2.java
C:₩04>javac StudentTest2.java
C:₩04>java StudentTest2
이름없음 주소없음 0
홍길동  군산 23
```

- 실행결과
  - 객체 생성시 매개변수가 있는 생성자 메서드실행
    - 각각의 멤버변수에 값을 할당
    - 반드시 매개변수의 개수와 형을 맞추어 줄것
  - 멤버변수의 값을 도스창에 출력
    - 생성자메서드에 의해 값이 할당됨 증명

오버로딩(Overloading)

- overloading이란?
  - 같은 이름을 가진 여러 개의 메서드
    - 객체의 다형성 지원
    - 한가지 이름으로 여러가지 기능을 제공
    - 중복메서드, 다중정의메서드 라고도 함
- overloading을 사용할 때의 규칙
  - 매개변수의 개수가 다를 것
  - 매개변수의 형이 다를 것
  - 위의 두 가지 조건중 하나만 달라도 overloading은 성립
  - 메서드의 리턴형은 overloading을 구분할 때 사용하지 않음

Overloading의 예

```
println()
println(boolean)
println(char)
println(double)
...
```

오버로딩(Overloading)의 사용 예

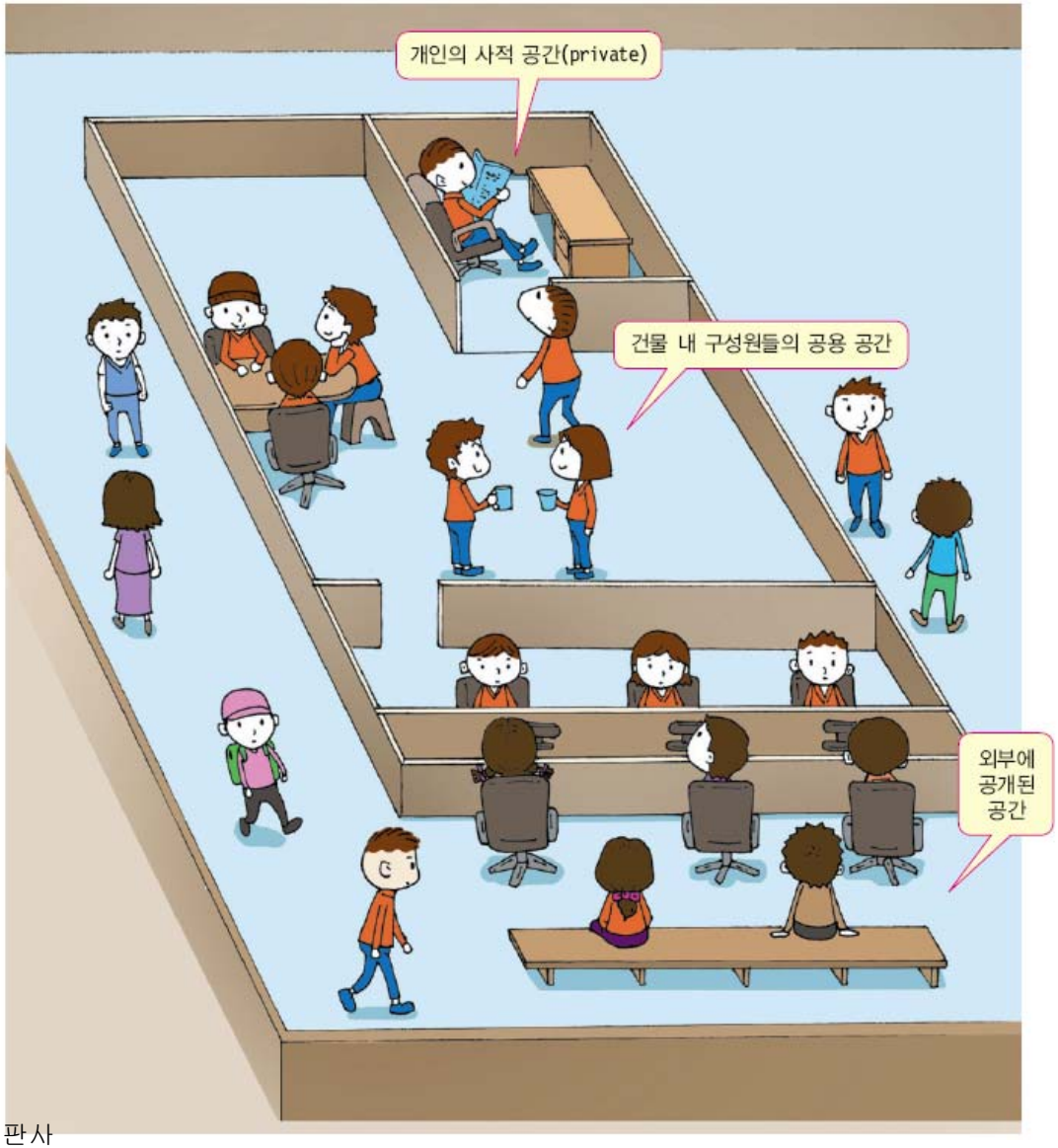
```
//OverloadCalc.java
public class OverloadCalc {
   public int plus(int a, int b) {
      return(a+b);
   }
   public float plus(float a, float b) {
      return(a+b);
   }
   public double plus(double a, double b) {
      return(a+b);
   }
}
```

```
// OverloadTest.java
public class OverloadTest{

  public static void main(String[] args) {
      OverloadCalc oc=new OverloadCalc();
      int i=oc.plus(3,5);
      float j=oc.plus(0.1f,0.2f);
      double k=oc.plus(0.5,0.7);
      System.out.println("int합:"+i);
      System.out.println("float합:"+j);
      System.out.println("double합:"+k);
   }
}
```

```
결과화면>
C:₩04>javac OverloadCalc.java
C:₩04>javac OverloadTest.java
C:₩04>java OverloadCalc
int합:8
float합:0.3
double합:1.2
```

- 실행결과
  - plus()메서드 호출
    - 각각의 메서드의 매개변수에 맞는 형태의 plus()메서드가 호출되어 실행
    - 프로그램을 만드는 사람은 각각의 경우를 모두 생각해야 하므로 귀찮음
    - 하지만 사용자는 쉽게 사용하게 됨

접근 지정자 이해



클래스 접근 지정자

- 클래스 앞에 올 수 있는 접근 지정자
  - public 접근 지정자

```
public class Person {}
```

    - 다른 모든 클래스가 접근 가능
  - 접근 지정자 생략 (default 접근 지정자)

```
class Person {}
```

    - 또는 package-private라고도 함
    - 같은 패키지 내에 있는 클래스에서만 접근 가능
      - 다른 말로 같은 디렉토리에 있는 클래스끼리 접근 가능

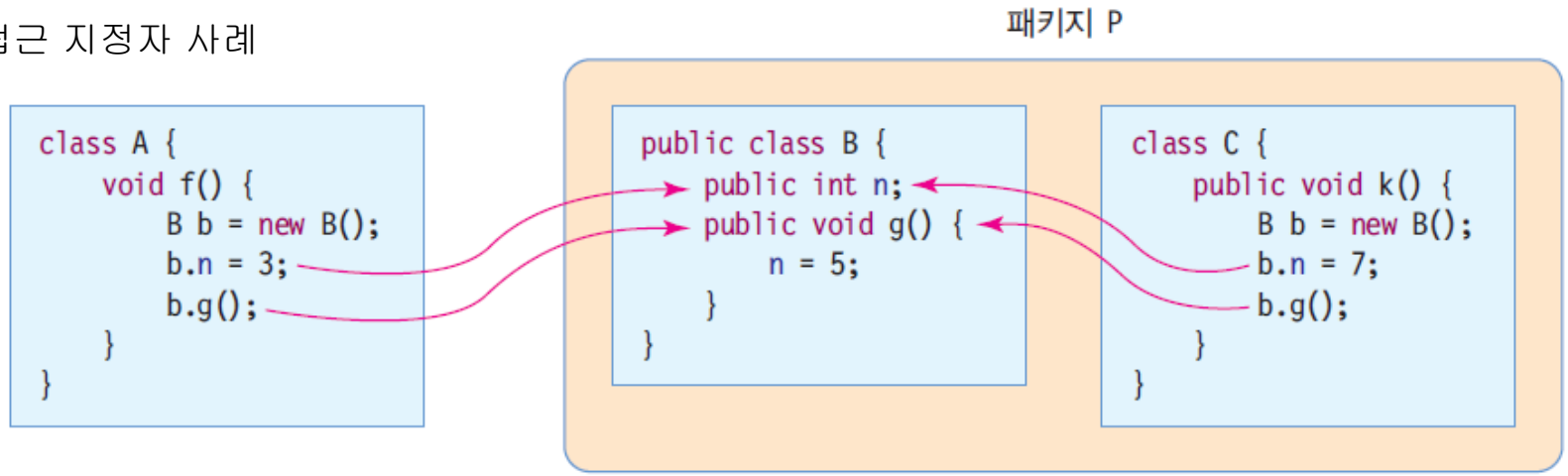
멤버 접근 지정자

default (또는 package-private)	•같은 패키지 내에서 접근 가능
public	•패키지 내부, 외부 클래스에서 접근 가능
private	•정의된 클래스 내에서만 접근 가능 •상속 받은 하위 클래스에서도 접근 불가
protected	•같은 패키지 내에서 접근 가능 •다른 패키지에서 접근은 불가하나 상속을 받은 경우 하위 클래스에서는 접근 가능

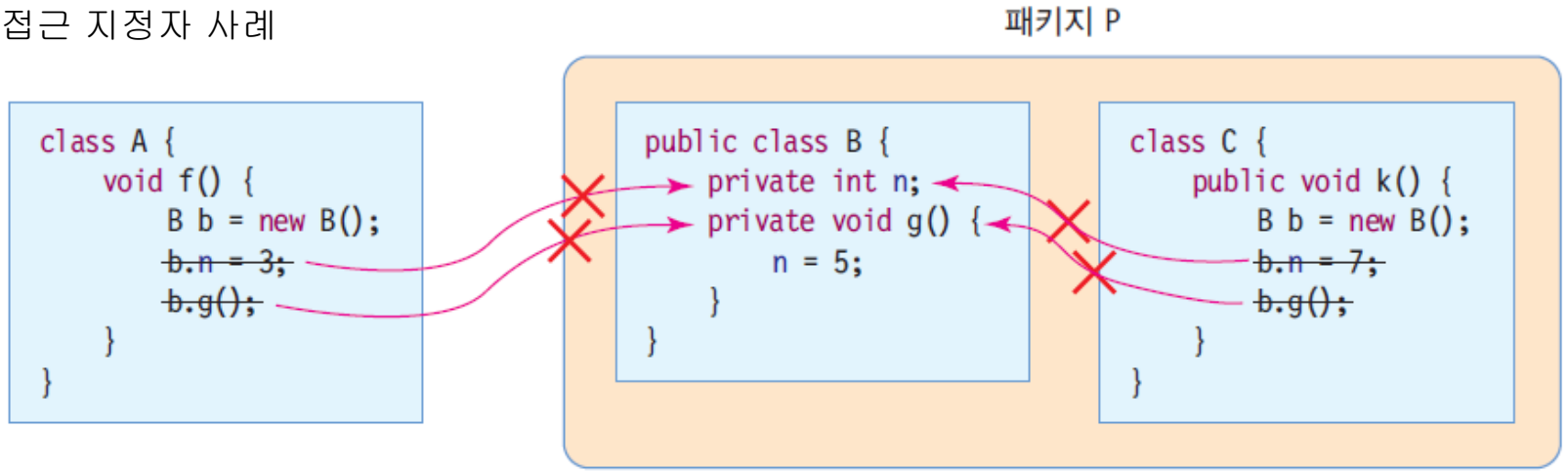
멤버에 접근하는 클래스	멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O

접근 지정자의 이해

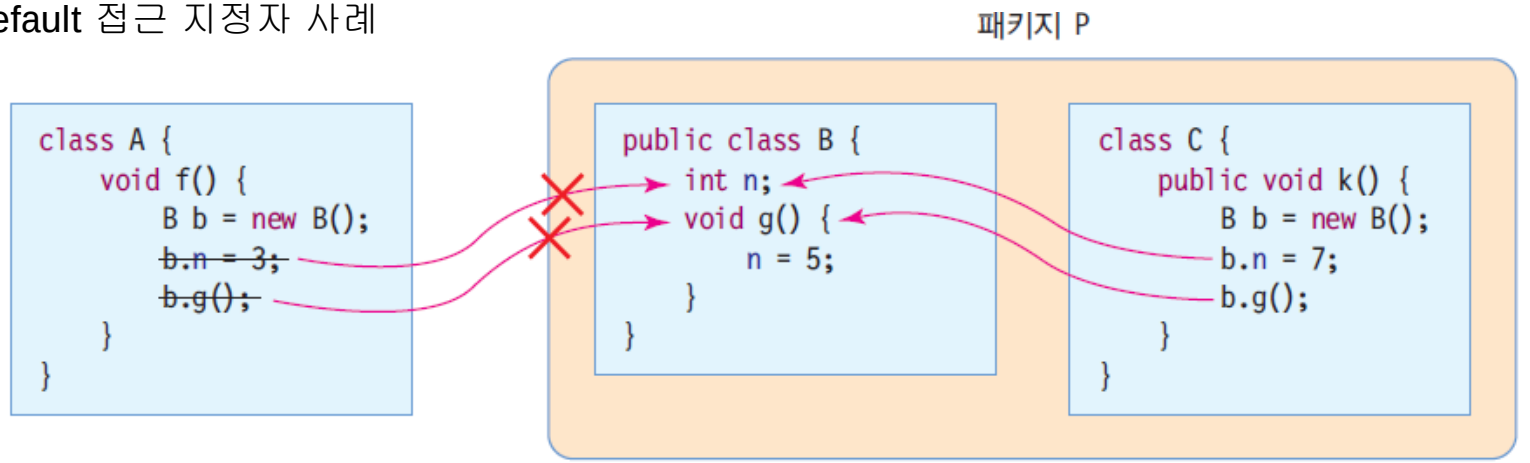
public 접근 지정자 사례



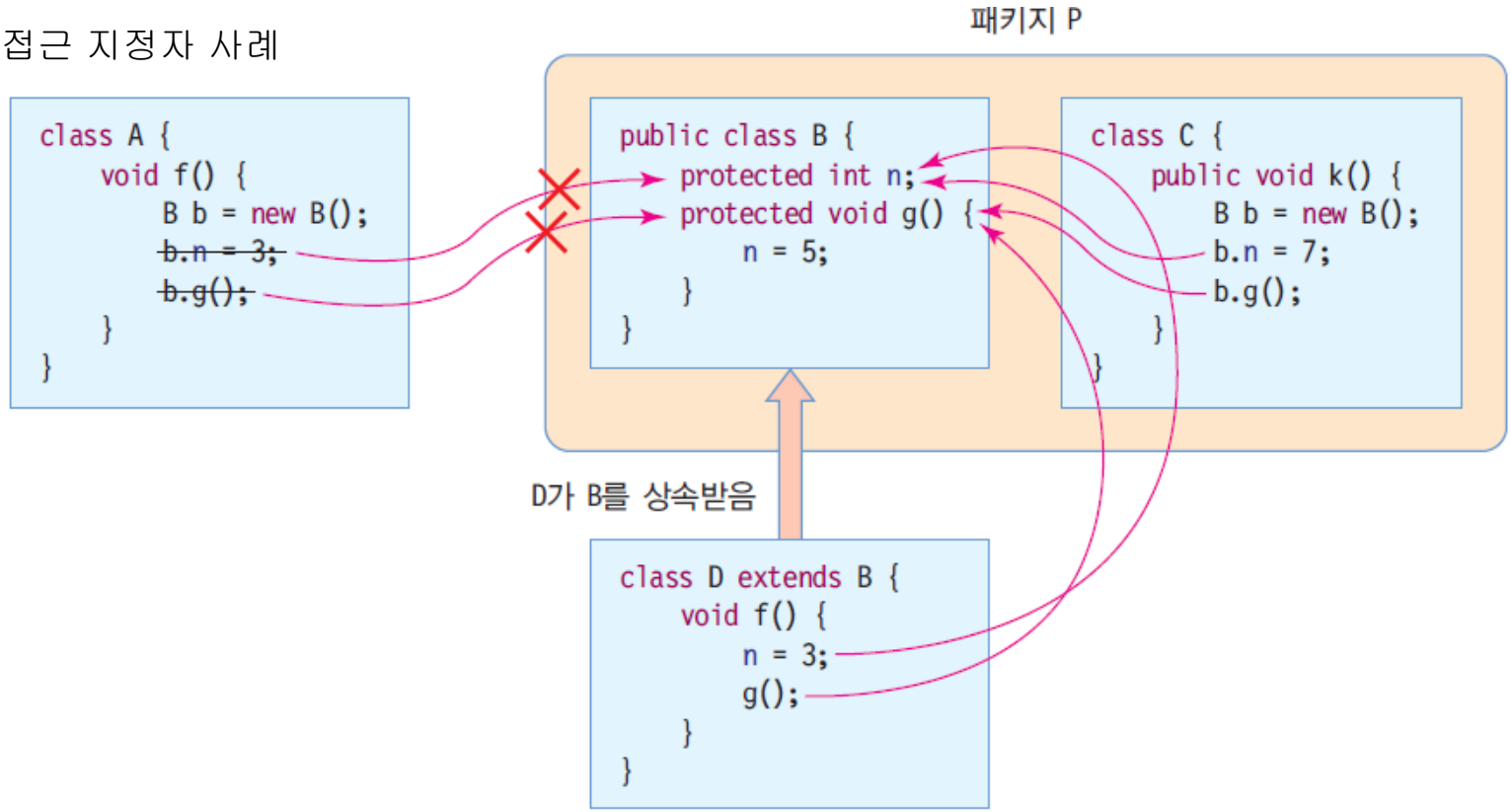
private 접근 지정자 사례



default 접근 지정자 사례



protected 접근 지정자 사례



예제 4-7 : 접근 지정자의 사용

다음의 소스를 컴파일 해보고 오류가 난 이유를 설명하고 오류를 수정하시오.

```
class Sample {
   public int a;
   private int b;
   int c;
}

public class AccessEx {
   public static void main(String[] args) {
      Sample aClass = new Sample();
      aClass.a = 10;
      aClass.b = 10;
      aClass.c = 10;
   }
}
```

- Sample 클래스의 a와 c는 각각 public, default 지정자로 선언이 되었으므로, 같은 패키지에 속한 AccessEx 클래스에서 접근 가능
- b는 private으로 선언이 되었으므로 AccessEx 클래스에서 접근 불가능

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The field Sample.b is not visible

        at AccessEx.main(AccessEx.java:11)
```

예제 4-7 결과

오류가 수정된 소스

```
class Sample {
   public int a;
   private int b;
   int c;
   public int getB() {
      return b;
   }
   public void setB(int value) {
      b = value;
   }
}
public class AccessEx {
   public static void main(String[] args) {
      Sample aClass = new Sample();
      aClass.a = 10;
      aClass.setB(10);
      aClass.c = 10;
   }
}
```

- private 접근 지정자를 갖는 멤버 b를 위해 클래스 내부에 getB()/setB() 메소드 만들어 접근

static 이해를 위한 그림

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 생기는 것이 아님
  - 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

예제

```
class  MyObj {
        static int s = 40;   // class variable ( static variable )
        int tot = 20;        // object variable

        void f()
        {
          short s = 300;   // Local variable
          int tot = 100;    // Local variable
          System.out.println( "s=" + s + " tot="+tot);
        }
        void g()
       {
            System.out.println("s=" + s + " tot="+tot);
            { int s= 500;    // S는 블록 지역 변수
              System.out.println( "s=" + s + " tot=" + tot );
            }
            System.out.println("s=" + s + " tot=" + tot );
         }
        public static void main( String args[] )
       {
              MyObj  m = new MyObj();
              m.f();
              m.g();
       }
      }
```

- 실행 결과

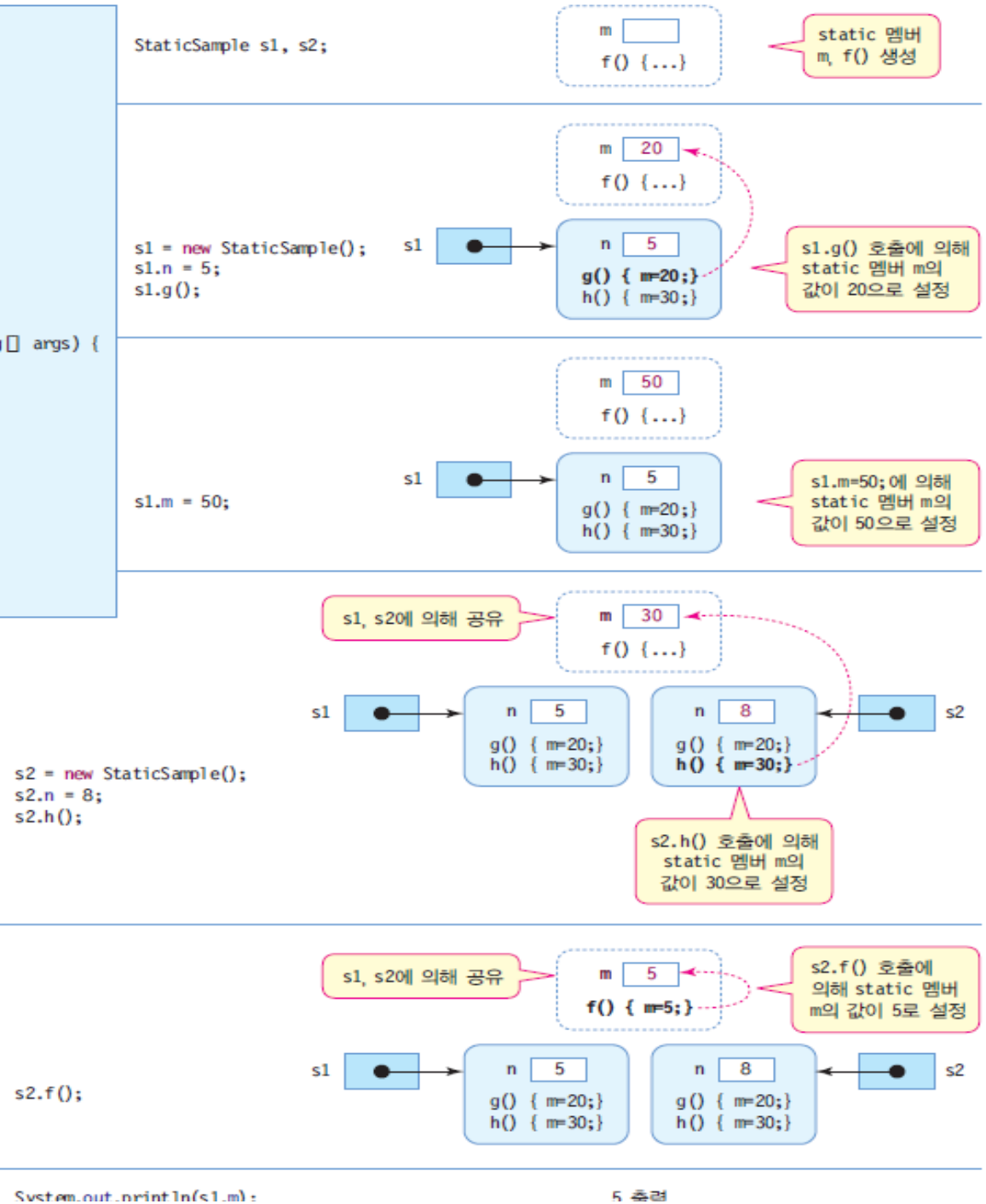
```
C:₩java₩c4> java  MyObj
s= 300   tot = 100
s= 40    tot = 20
s= 500   tot = 20
s= 40    tot = 20
```

static 멤버를 객체의 멤버로 접근하는 사례

```java
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);
    }
}
```

실행 결과

```
5
```



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();
    }
}
```

실행 결과

```
10
```

static의 활용

- 전역 변수와 전역 함수를 만들 때 활용
  - 자바의 캡슐화 원칙 지킴
    - 다른 클래스에서 공유하는 전역 변수나 전역 함수도 반드시 클래스 내부에 구현해야 함
- static 멤버를 가진 클래스 사례
  - 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 메소드

static 메소드 (정적 메소드, 클래스 메소드)

- 변수와 유사하게 메소드(method)에도 static이라는 한정자가 붙을 수 있음. 이를 클래스 메소드라고 함.

- 클래스 메소드는 static 한정자가 붙은 클래스 변수만을 접근 가능. 객체 변수들은 액세스 할 수 없음.

- 클래스 메소드를 사용하기 위해서는 다음과 같이 클래스 이름(또는 객체 이름)과 메소드 이름을 같이 명시함.

```
ClassName.static_methodName();
```

또는

```
ObjectInstance.static_methodName();
```

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

Static 메소드

- 다음은 클래스 메소드를 사용한 예제 프로그램.
  static변수 total과 static메소드 st_inc()가 선언되어 사용되고 있음에 유의.

```
class Count {
    static int total = 10;    // static variable
    int cnt = 6;
    public static void st_inc()  {    // 정적 메소드
                total++;
    }
```

Static 메소드

```
    public static inc() {  // 일반 메소드
                cnt++;
    }
    public void out() {    // print two variables
        System.out.println("total="+total+"₩tcnt="+춧);
    }
}
public class CountTest {
   public static void main( String[] args ) {
            Count m = new Count();
            m.out();          // print 메소드 호출
            m.inc();        error
            m.st_inc();       // 정적 메소드 호출
            Count.st_inc();
            m.out();          // print 메소드 호출
   }
}
```

Static 메소드

- 실행 결과

```
C:₩java₩c4> java CountTest
total=10          cnt=6
total=12          cnt=7
```

- 가장 많이 사용되는 static 메소드는 바로 main()이다.

  main()은 클래스의 객체가 생성되기 전에 실행되는 메소드이므로 static은 선언되어야 함.

- 메소드의 반환값.

  메소드는 객체를 포함하여 자바에서 유효한 모든 데이터형을 반환 가능.
  메소드 선언부에서 반환 값의 자료형이 명시되어야 함.

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) + "달러입니다.");
        System.out.println("백달러는 " + CurrencyConverter.toKWR(100) +"원입니다.");
    }
}
```

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

객체를 반환하는 메소드

- 다음은 정수형 데이터를 반환하는 메소드 sum()의 예제 코드.

```
public int sum( int max ) {

    int total;

    for( int i=1; i<= max; i++ ) total +=i;

    return(total);
}
```

객체를 반환하는 메소드

- 다음은 객체를 반환하는 예제 프로그램.

Test클래스의 NewOne() 메소드는 새로운 Test 객체를 생성하고, 생성된 객체의 참조(= 주소)를 반환함.

```
class Test {
    int a;

    Test( int i ) {
          a=i;
     }

    Test NewOne() {  // Test 객체의 참조를 반환
       Test temp = new Test(a*2);
       return temp;
     }
  }
```

객체를 반환하는 메소드

```
class Obj {
    public static void main( String args[] ) {
        Test  ob1 = new Test(3);
        Test  ob2;

        ob2 = ob1.NewOne();
        System.out.println("ob1.a: " + ob1.a);
        System.out.println("ob2.a: " + ob2.a);
        ob2 = ob2.NewOne();
        System.out.println("ob2.a: " + ob2.a);
    }
}
```

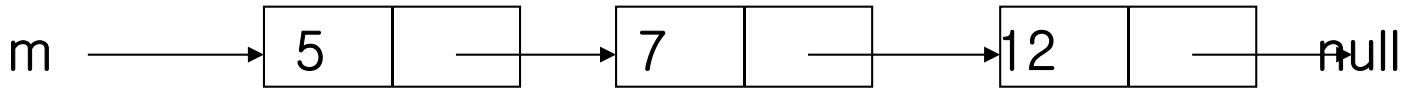
- 실행 결과

```
C:₩java₩c4> java Obj
ob1.a: 3
ob2.a: 6
ob2.a: 12
```

객체를 반환하는 메소드

- 다음은 객체를 반환하는 예제 프로그램.
  Linked List를 구현한 것.

  예제 프로그램의 실행 결과로 생성되는 Linked List를 도식화하면,



객체를 반환하는 메소드

- 예제 프로그램

```
class LinkedList  {
    int  data;
    LinkedList   link;
    LinkedList( int i ) {
            data = i;
            link = null;
    }
    void append( LinkedList l ) {
            link = l;
    }
    static LinkedList newNode( int i ) {
             // 객체를 생성해서 반환하는 메소드
            LinkedList  t= new LinkedList(i);
            return(t);
     }
}
```

객체를 반환하는 메소드

```
class TestLink {

    public static void main( String arg[] ) {
            LinkedList  m = new LinkedList(5);
            m.append( LinkedList.newNode(7) );
            m.link.append( LinkedList.newNode(12) );
            LinkedList t=m;

            while( t!=null ) {   // 링크드 리스트 출력
                System.out.println( "list value = " + t.data );
                t=t.link;
            }
     }
   }
C:₩java₩c4> java TestLink
list value = 5
list value = 7
list value = 12
```

final 클래스와 메소드

- 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

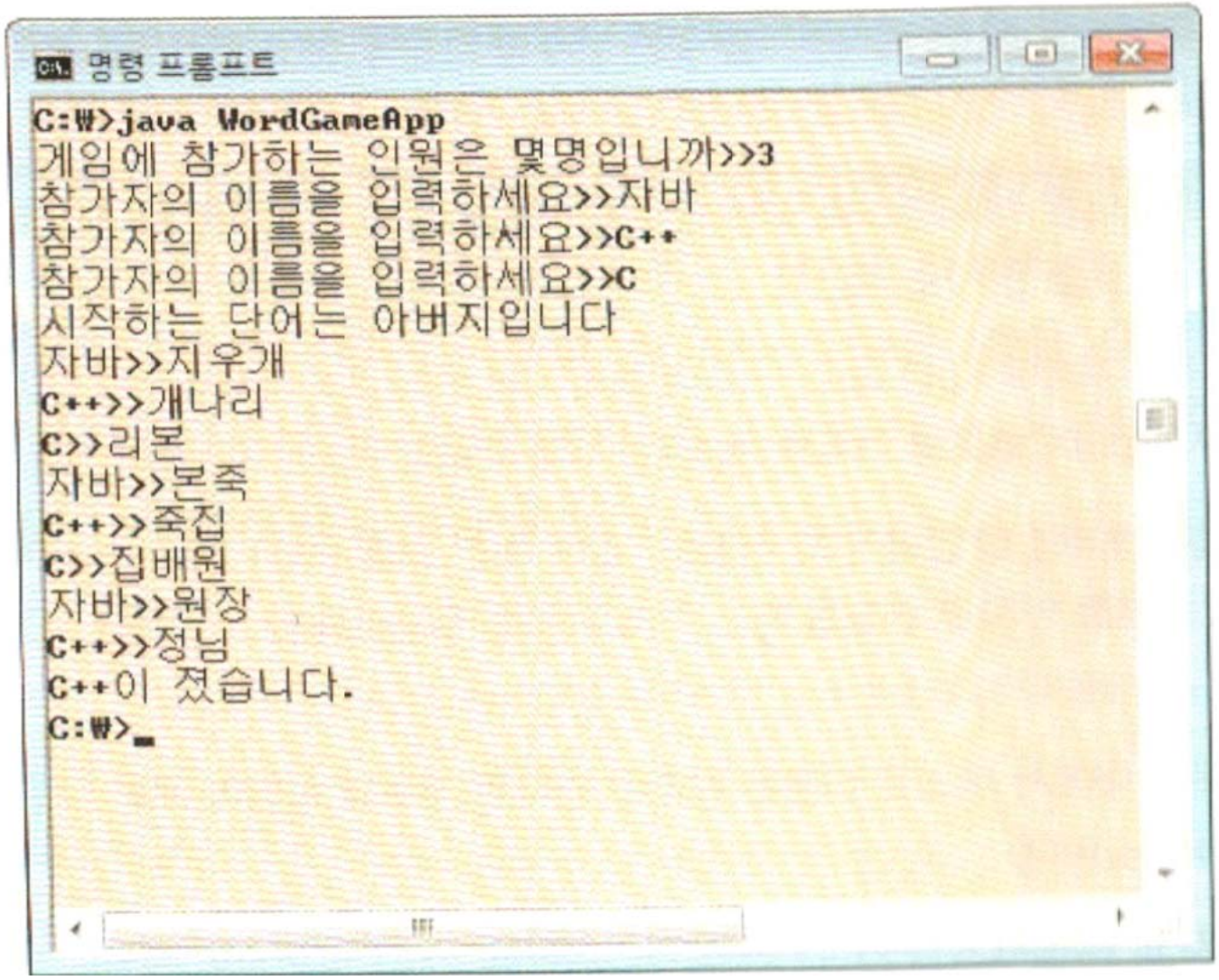
```
public class FinalFieldClass {
        final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정
        final int COLS; // 컴파일 오류, 초기값을 지정하지 않았음
        void f() {
                int [] intArray = new int [ROWS]; // 상수 활용
                ROWS = 30; // 컴파일 오류 발생, final 필드 값을 변경할 수 없다.
        }
}
```

```
class Add {
        private int a, b;
        public void setValue(int a, int b) {
                this.a = a;
                this.b = b;
        }
        public int calculate() {
                return a+b;
        }
}

class Mul {
        private int a, b;
        public void setValue(int a, int b) {
                this.a = a;
                this.b = b;
        }
        public int calculate() {
                return a*b;
        }
}
```

```
class Sub {
        private int a, b;
        public void setValue(int a, int b) {
                this.a = a;
                this.b = b;
        }
        public int calculate() {
                return a-b;
        }
}

class Div {
        private int a, b;
        public void setValue(int a, int b) {
                this.a = a;
                this.b = b;
        }
        public int calculate() {
                return a/b;
        }
}
```

```
public class Calc {
		public static void main (String args[])  {
				Scanner sin = new Scanner(System.in);
				System.out.print("두 정수와 연산자를 입력하시오>>");
				int a = sin.nextInt();
				int b = sin.nextInt();
				char operator = sin.next().charAt(0); // 연산자를 문자로 변환
				switch (operator) {
				case '+':
						Add add = new Add();
						add.setValue(a, b);
						System.out.println(add.calculate());
						break;
				case '-':
						Sub sub = new Sub();
						sub.setValue(a, b);
						System.out.println(sub.calculate());
						break;
				case '*':
						Mul mul = new Mul();
						mul.setValue(a, b);
						System.out.println(mul.calculate());
						break;
				case '/':
						Div div = new Div();
						div.setValue(a, b);
						System.out.println(div.calculate());
						break;
				default:
						System.out.println("잘못된 연산자입니다.");
				}

		}
}
```

끝말잇기 게임


명령 프롬프트
C:W>java WordGameApp
게임에 참가하는 인원은 몇명입니까>>3
참가자의 이름을 입력하세요>>자바
참가자의 이름을 입력하세요>>C++
참가자의 이름을 입력하세요>>C
시작하는 단어는 아버지입니다
자바>>지우개
C++>>개나리
C>>리본
자바>>본죽
C++>>죽집
C>>집배원
자바>>원장
C++>>정님
C++이 졌습니다.
C:W>_

끝말잇기 게임

```
public class WordGameApp {
        private Scanner scanner; // 키보드 입력
        private String startWord = "아버지"; // 시작 단어
        private Player[] players; // 게임 참가자들

        public WordGameApp() {
                scanner = new Scanner(System.in);
        }

        // 게임 참가자 수를 입력받고 Player []을 생성하는 메소드
        private void createPlayers() {
                System.out.print("게임에 참가하는 인원은 몇명입니까>>");
                int nPlayers = scanner.nextInt();
                players = new Player[nPlayers]; // Player [] 레퍼런스 배열 생성

                // 각 참여자의 이름을 입력받아 Player 객체 생성
                for(int i=0; i<nPlayers; i++) {
                        System.out.print("참가자의 이름을 입력하세요>>");
                        String name = scanner.next();
                        players[i] = new Player(name);
                }
        }
```

끝말잇기 게임

```
        // 게임을 진행하는 메소드
        public void run() {
                createPlayers(); // 참가자를 위한 Player []을 생성한다.
                String lastWord = startWord; // startWord에서 부터 시작한다.

                System.out.println("시작하는 단어는 "+lastWord+ "입니다");
                int next =  0; // 참가자들의 순서대로 증가

                // 게임이 끝날 때까지 루프
                while(true) {
                        String newWord = players[next].sayWord(); // next 참가자가 단어를 말하도록 한다.
                        if(!players[next].succeed(lastWord)) { // next 참가자가 성공하였는지 검사.
                                System.out.print(players[next].getName() + "이 졌습니다.");
                                break; // 게임을 종료한다.
                        }
                        next++; // 다음 참가자
                        next %= players.length; // next가 참가자의 개수보다 많게 증가하는 것을 막는다.
                        lastWord = newWord;
                }
        }

        public static void main(String[] args) {
                WordGameApp game = new WordGameApp();
                game.run();
        }
}
```

```
// 한 사람의 참가자를 표현하는 클래스
class Player {
		Scanner scanner; // 키보드 입력
		private String name; // 게임 참가자의 이름
		private String word; // 참가자가 말한 단어

		public Player(String name) {
				this.name = name;
				scanner = new Scanner(System.in);
		}

		public String getName() {return name;}

		public String sayWord() {
				System.out.print(name+">>");
				word = scanner.next();
				return word;
		}

		// lastWord와 참가자가 말한 word를 비교하여 끝말잇기가 잘되었는지 판단.
		// 성공하였으면 true 리턴
		public boolean succeed(String lastWord) {
				int lastIndex = lastWord.length()-1; // 최종 단어의 맨 마지막 문자의 인덱스

				// 최종 단어의 맨 마지막 문자와 참가자가 말한 단어의 첫 문자가 같은지 비교
				if(lastWord.charAt(lastIndex) == word.charAt(0))
						return true;
				else
						return false;
		}
}
```

리포트 : 공연 예약 시스템

- 공연은 하루에 한 번 있다.
- 좌석은 S석, A석, B석 타입이 있으며 모두 10석의 좌석이 있다.
- 공연 예약 시스템의 메뉴는 "예약", "조회", "취소", "끝내기"가 있다.
- 예약은 한 자리만 예약할 수 있고 좌석 타입, 예약자 이름, 좌석 번호를 순서대로 입력받아 예약한다.
- 조회는 모든 종류의 좌석을 표시한다.
- 취소는 예약자의 이름을 입력하여 취소한다.
- 없는 이름, 없는 번호, 없는 메뉴, 잘못된 취소 등에 대해서 오류 메시지를 출력하고 사용자가 다시 시도하도록 한다.