Class

Java Programming

Byeongjoon Noh

powernoh@sch.ac.kr

SCH SOON CHUN HYANG UNIVERSITY

Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

Contents

1. Classes and objects

Concept of OOP

- Object-oriented programming (OOP)
  - a programming paradigm based on the concept of “objects”
    - can contain data (variables), procedure (methods), and code

- Advantages of OOP
  - increases development speed
  - improves software maintenance and management
  - enhances productivity in software development
  - reduces development costs

구성 요소	설명
클래스	같은 종류의 집단에 속한 속성과 행동을 정의한 틀이다.
객체	클래스의 인스턴스이다.
캡슐화	데이터와 행동을 하나의 단위로 묶는 기법이다.
상속	이미 존재하는 한 클래스의 멤버(변수, 메서드)를 다른 클래스에 물려주는 기법이다.
다형성	변수, 메서드 또는 객체가 여러 형태를 취하는 기법이다.
추상화	불필요한 내부 세부 사항을 숨기고 필수 사항을 표시하는 것을 의미한다.

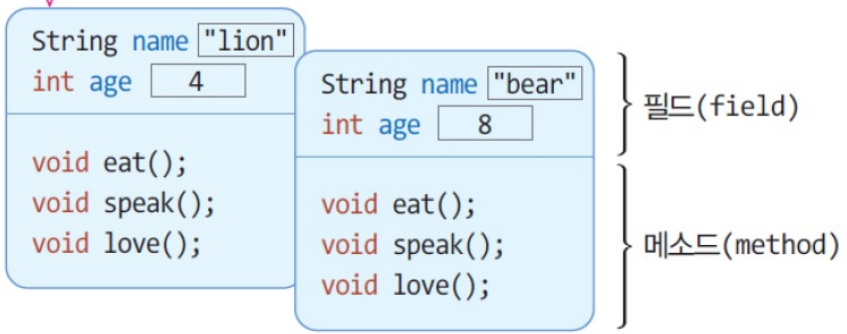
Key features

- Encapsulation
  - bundling the data and the methods that operate on the data into a single unit called a class
  - hiding the details of the information to protect the data and methods of an object
    - making it safe from unauthorized modification
  - examples
    - medicine contains within a capsule
      - encapsulation hides and safely protects the variables and methods of a class

클래스 선언

```
class Animal {
   String name;
   int age;
   void eat() {...}
   void speak() {...}
   void love() {...}
}
```

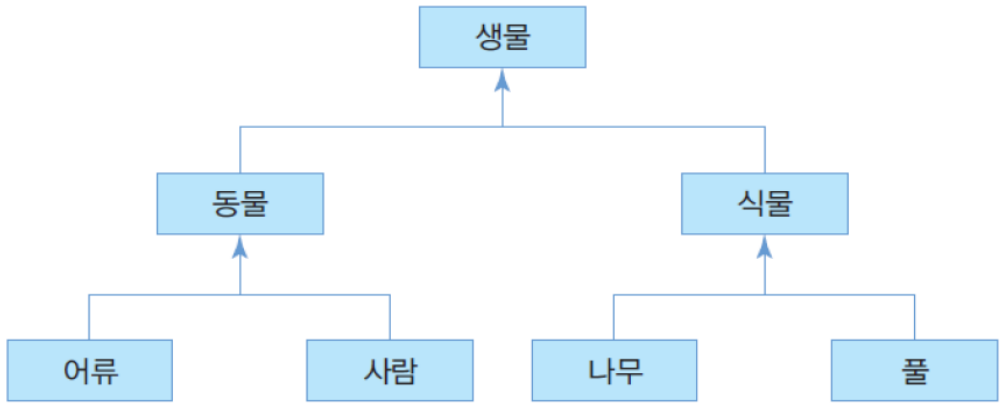
클래스 모양으로 만들어진 객체

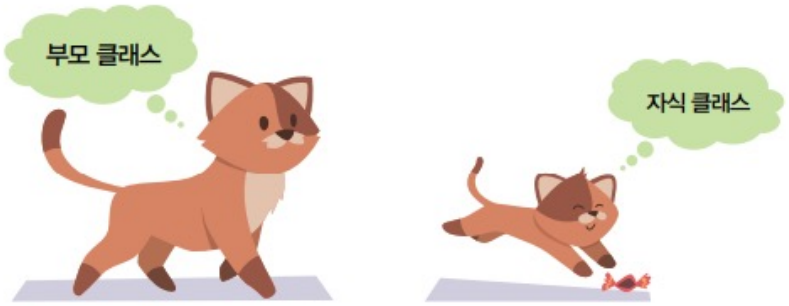


Animal 객체 두 개 생성

Key features

- **Inheritance**
  - a technique where one class shares its members (variables, methods) with another class
  - helps in reusing code and establishing a parent-child relationship between tow classes





슈퍼 클래스

```
class Animal {
    String name;
    int age;
    void eat() {...}
    void speak() {...}
    void love() {...}
}
```

상속

서브 클래스

```
class Human extends Animal {
    String hobby;
    String job;
    void work() {...}
    void cry() {...}
    void laugh() {...}
}
```

Animal의 객체

```
String name;
int age;

void eat();
void speak();
void love();
```

Human의 객체

```
String name;
int age;

void eat();
void speak();
void love();

String hobby;
String job;

void work();
void cry();
void laugh();
```

서브 클래스 객체는 슈퍼 클래스의 멤버와 서브 클래스의 멤버를 모두 가짐

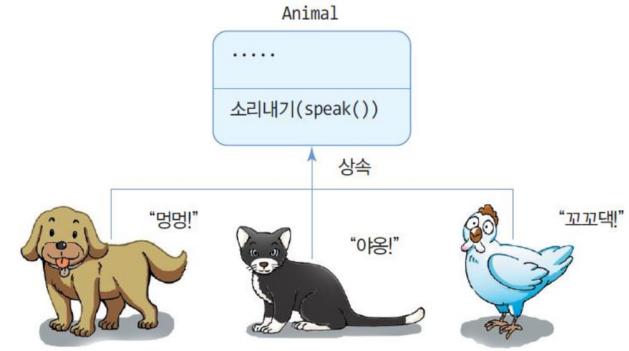
Key features

- Polymorphism
  - “many forms” allowing us to perform a single action in different ways
  - redefining a method of a superclass in the subclass ➔ method overriding
    - Recall: method overloading
      - the same method with the different parameter list

```java
public class CatButler {
    public void myRole(int age) {
        System.out.println("나는 고양이 집사입니다. 나이는 " + age + "입니다");
    }

    public void myRole(String name) {
        System.out.println(name + "은 학생입니다.");
    }

    public void myRole(String name, int age) {
        System.out.println(name + "은 자녀입니다. 나이는 " + age + "입니다");
    }
}
```

메서드 오버로딩

메서드 오버로딩




```java
class Animal {
    public void Sound() {
        System.out.println("동물의 울음소리");
    }
}

class Cat extends Animal {
    public void Sound() {
        System.out.println("고양이는 야옹야옹");
    }
}

class Dog extends Animal {
    public void Sound() {
        System.out.println("강아지는 멍멍멍");
    }
}
```

메서드 오버라이딩

메서드 오버라이딩

Key features

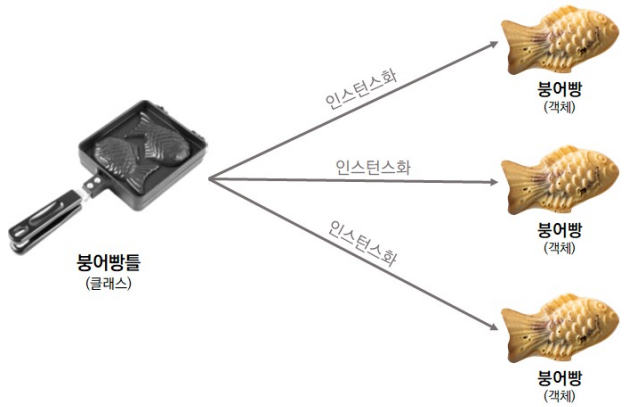
- Abstraction
  - representing the essential features without including the background details
  - hides unnecessary details to reduce complexity and effort in programming

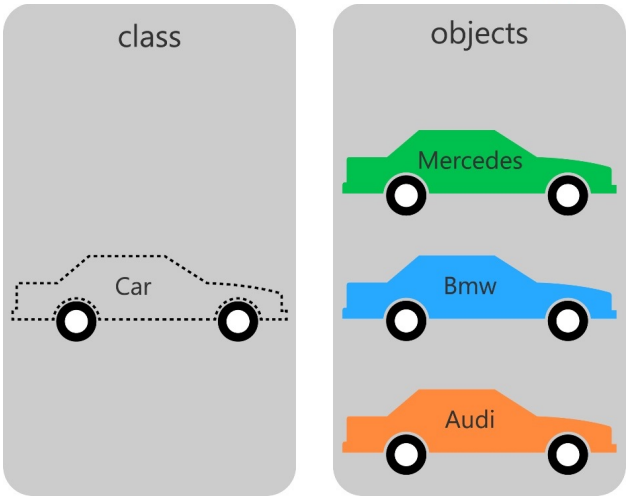
Concept of classes and objects

- Class
  - object의 속성과 행위를 포함한 설계도 (blueprint)
    - 속성과 행위 = variables (data) and methods

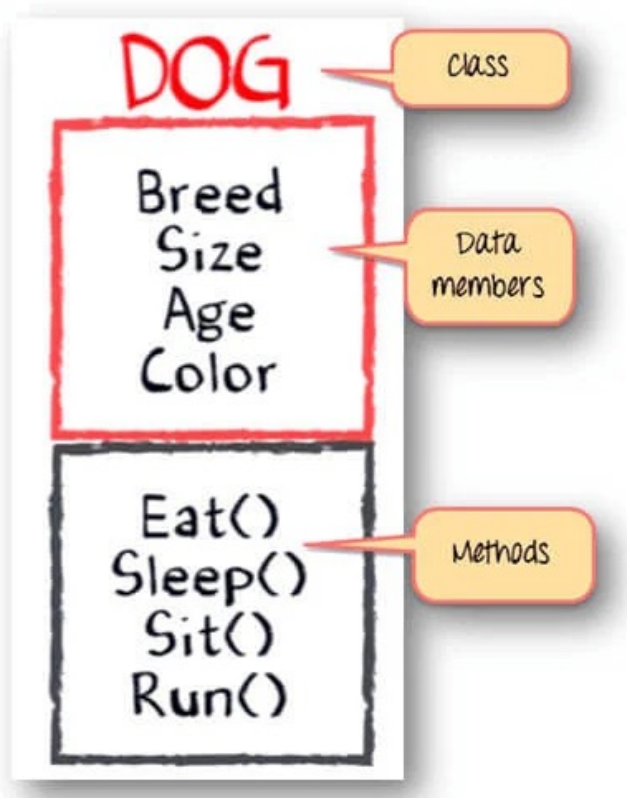
- Object
  - class의 틀로 찍어낸 실체 (instance)
  - 프로그램 실행 중 생성되는 실체 (메모리에 공간을 가짐)

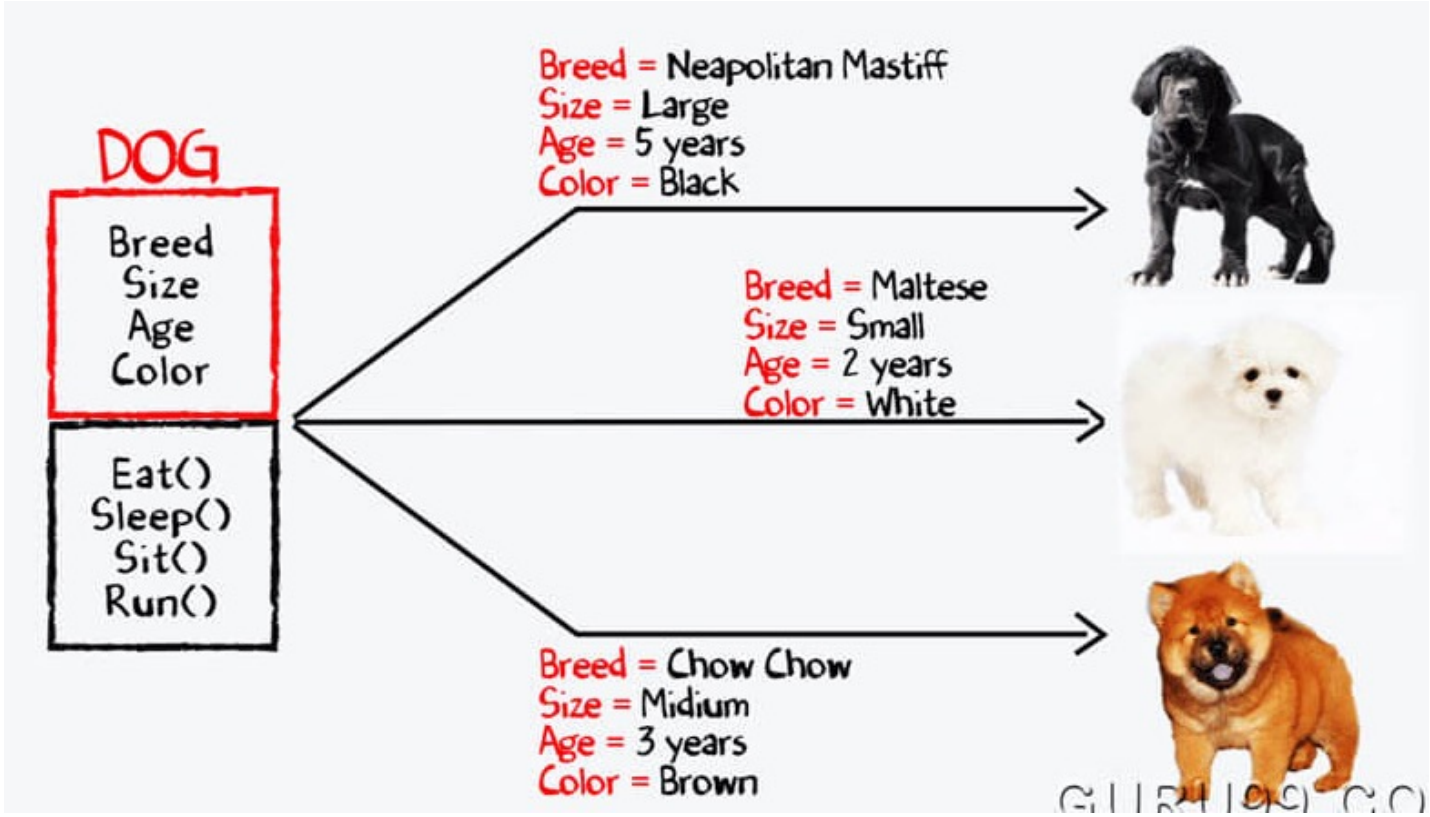
- Examples
  - class = 소나타 자동차 ➔ object = 생산된 소나타 100대
  - class = 동물 ➔ object = 개 5마리, 고양이 2마리, 사람 10명





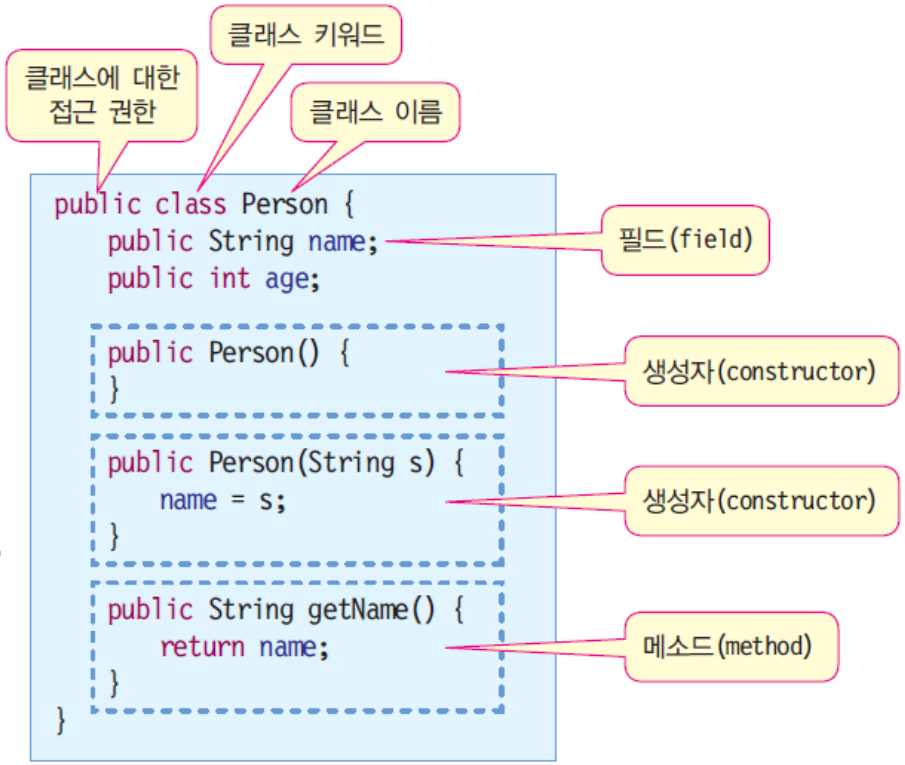
Concept of classes and objects





Class design and instantiation

- Class design
  - use ‘class’ keyword
  - bundles the code in class by using curly braces { }
  - components: fields (variables), constructor, methods
    - not all classes contain both variables and methods
      - 변수만 존재하는 클래스나 메소드만 존재하는 클래스도 있음
      - depending on the tasks that the class performs



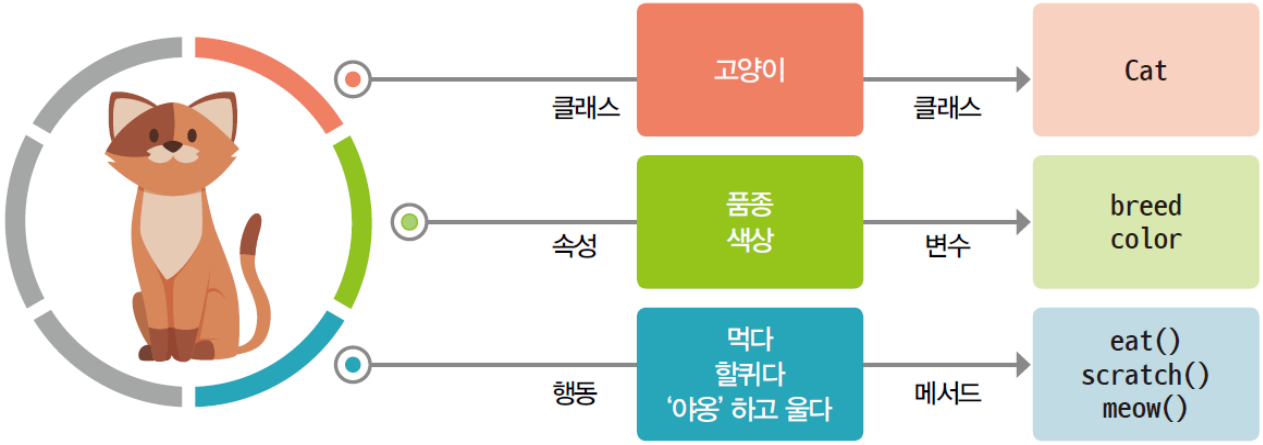
Class design and instantiation

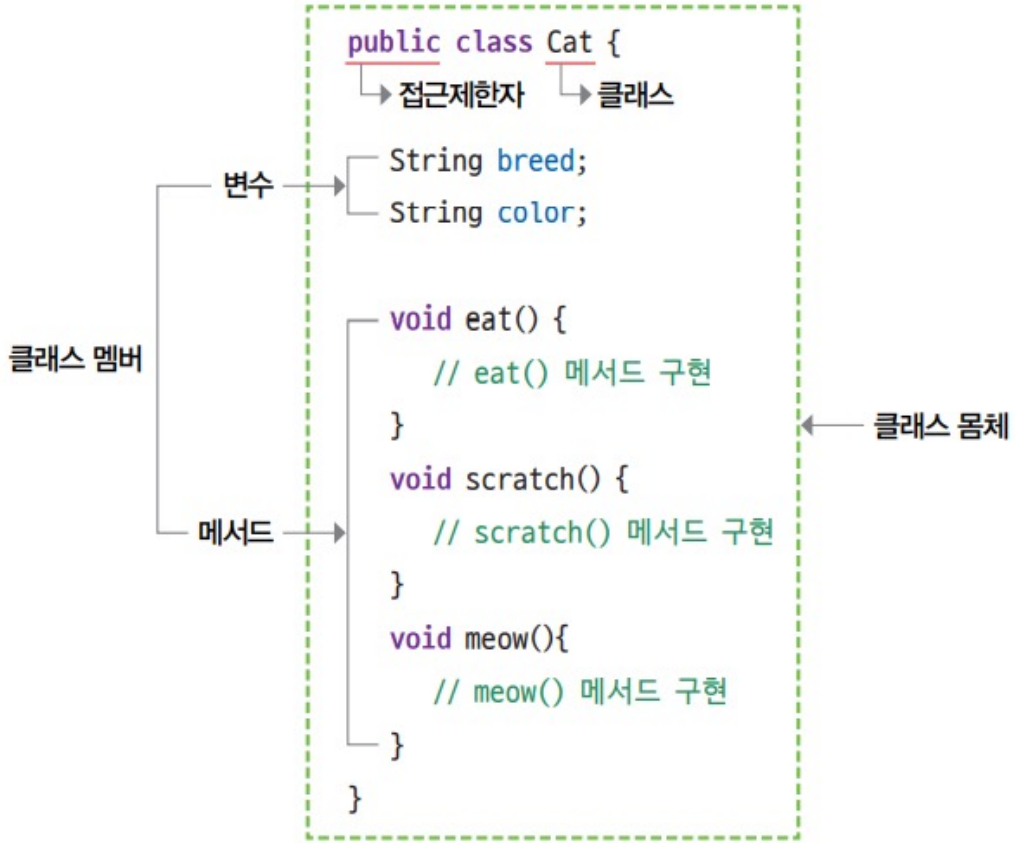
- Components of the class
  - member variable (field)
    - declared in a class, but outside a method
    - attributes that each object created from the class can have
  - member method
    - defines the behavior or functionality that objects of a class can perform
  - constructor
    - special member methods used to initialize new objects
    - to set initial values for the member variables of new object

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

  void eat() { … }
  void speak() { … }
  void love() { … }

  public Animal() { … }

  public static void main(String[] args) { … }
```

Class design and instantiation

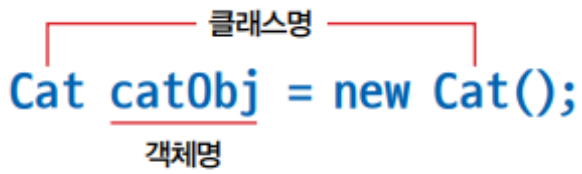
- Class design example

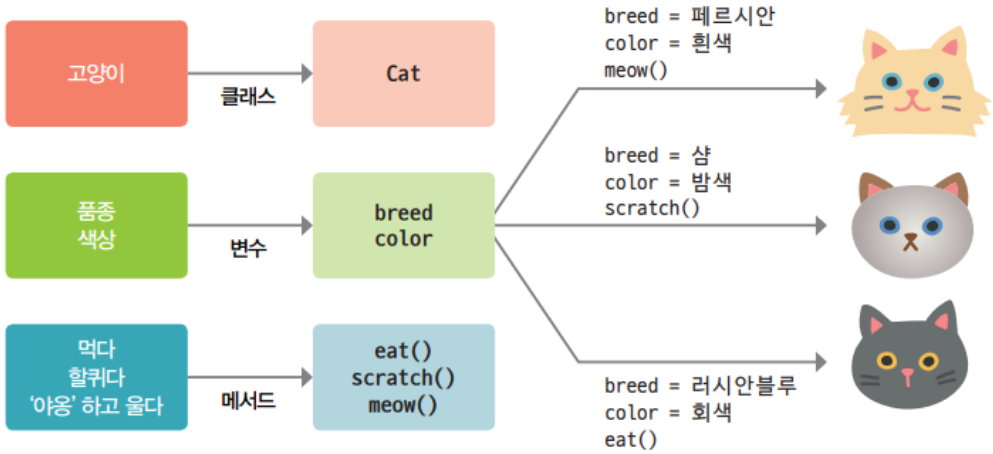




Class design and instantiation

- Object instantiation
  - use ‘new’ keyword





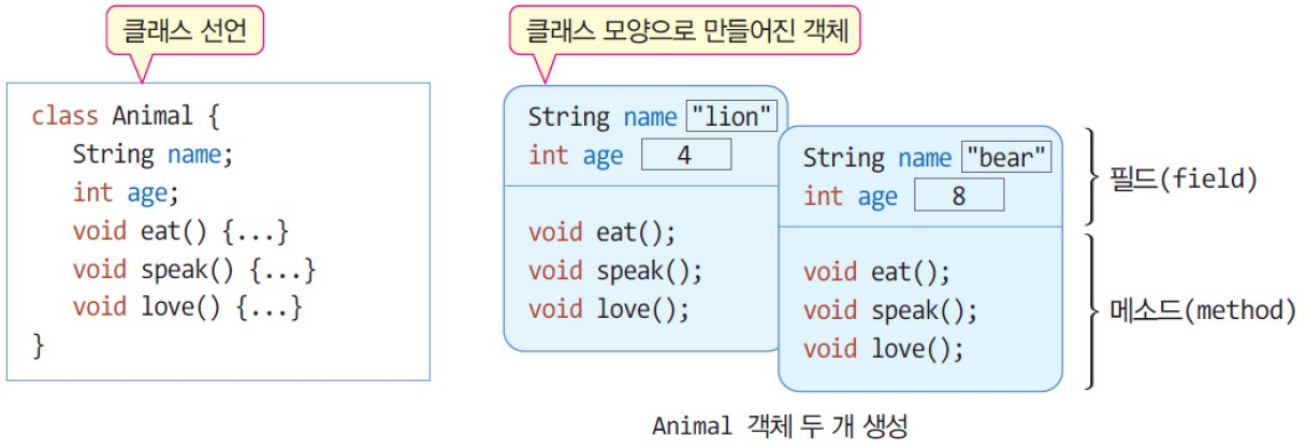

```
public class Cat {
   String breed;
   String color;

   void eat() {
      System.out.println("eat() method");
   }
   void scratch() {
      System.out.println("scratch() method");
   }
   void meow() {
      System.out.println("meow() method");
   }

   public static void main(String[] args) {
      Cat cat = new Cat();
   }
}
```

← object instantiation

Usage of class and object

- ‘Animal’ class design and object instantiation

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

  void eat() {
    System.out.println("eat() method");
  }
  void speak() {
    System.out.println("speak() method");
  }
  void love() {
    System.out.println("love() method");
  }
  public static void main(String[] args) {
    Animal animal_1 = new Animal();
    Animal animal_2 = new Animal();
    animal_1.name = "lion";
    animal_1.age = 4;
    animal_2.name = "bear";
    animal_2.age = 8;
  }
}
```

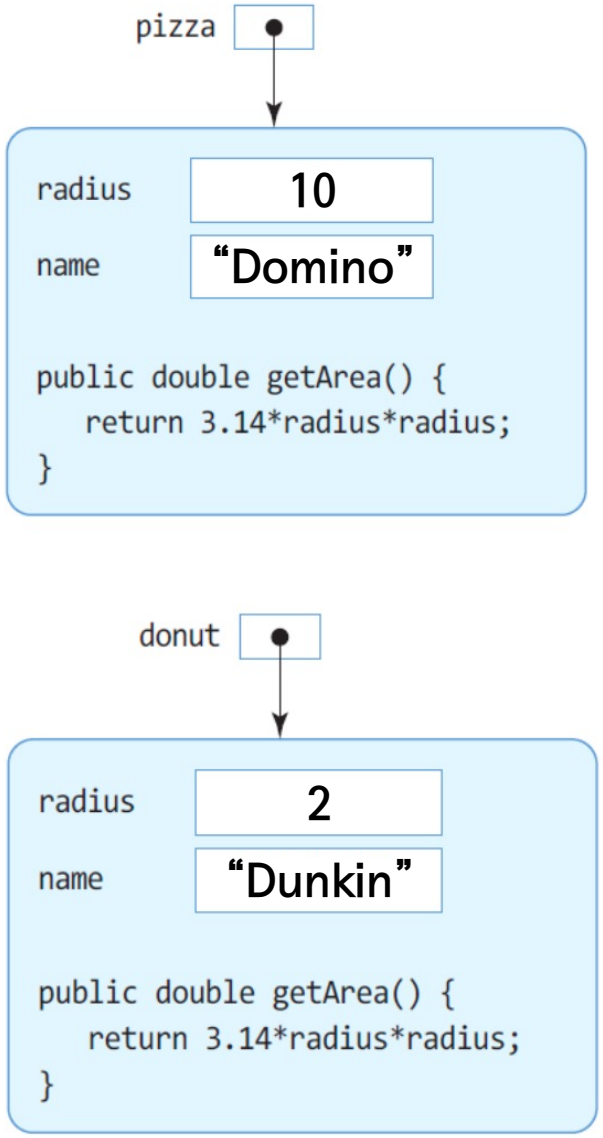


← USE dot (.) operator to access the member variables and methods

Usage of class and object

- ‘Circle’ class design and object instantiation

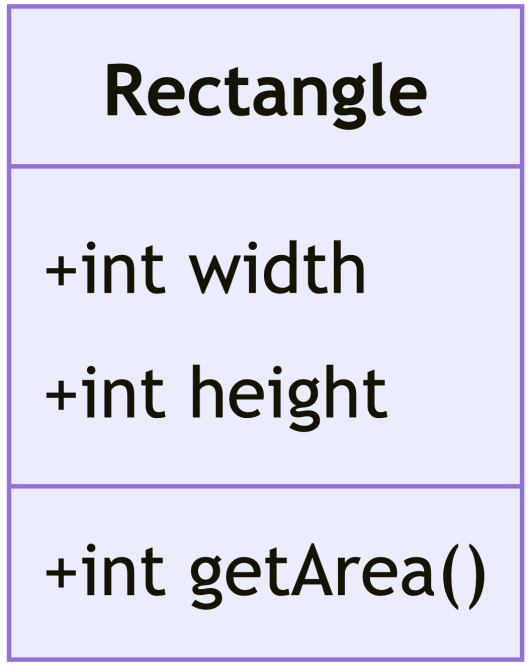
```
public class Circle {
  int radius;
  String name;
  public double getArea() {
    return 3.14 * radius * radius;
  }
  public static void main(String[] args) {
    Circle pizza;
    pizza = new Circle();
    pizza.radius = 10;
    pizza.name = "Domino";
    double area = pizza.getArea();
    System.out.println(pizza.name + "의 면적은 " + area);

    Circle donut = new Circle();
    donut.radius = 2;
    donut.name = "Dunkin";
    area = donut.getArea();
    System.out.println(donut.name + "의 면적은 " + area);
  }
}
```



Usage of class and object

- 'Rectangle' class design and object instantiation
  - requirements: 너비, 높이 값을 갖고, 면적값을 계산하는 getArea() 메소드를 가진 Rectangle 클래스를 작성하라

```
public class Rectangle {
  int width, height;
  int getArea() {
    return width * height;
  }

  public static void main(String[] args) {
    Rectangle rect = new Rectangle();
    rect.width = 4;
    rect.height = 5;
    System.out.println("사각형의 면적은 " + rect.getArea() + "입니다.");
  }
}
```

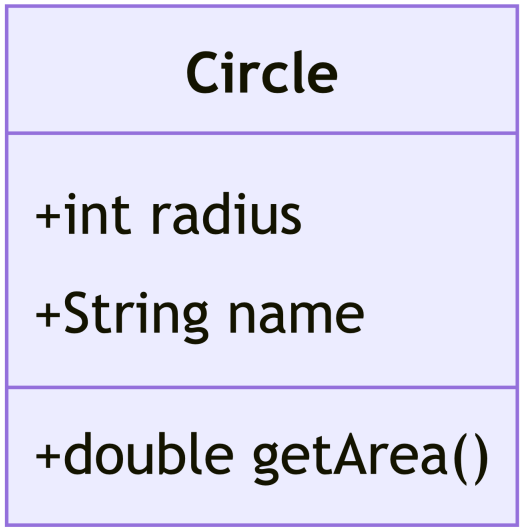


class diagram

Usage of class and object

- How can we represent the 'Circle' class into class diagram

```
public class Circle {
  int radius;
  String name;
  public double getArea() {
    return 3.14 * radius * radius;
  }
  public static void main(String[] args) {
    Circle pizza;
    pizza = new Circle();
    pizza.radius = 10;
    pizza.name = "Domino";
    double area = pizza.getArea();
    System.out.println(pizza.name + "의 면적은 " + area);

    Circle donut = new Circle();
    donut.radius = 2;
    donut.name = "Dunkin";
    area = donut.getArea();
    System.out.println(donut.name + "의 면적은 " + area);
  }
}
```



2. Constructors

What is a constructor?

- Constructor
  - a special method that has the same name as the class serving to set up new instances of the class
    - method ➔ a class can have more than once constructor, each with different parameters
      - “constructor overloading”
  - no return type (not even void)
  - only called once per object at the time of object creation (객체가 생성되는 순간에 자동으로 딱 한 번 호출)
- Example of constructor for ‘Circle’ class

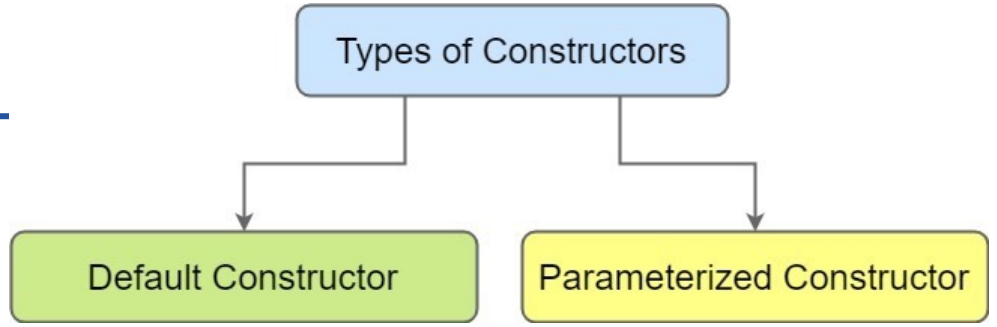
```
public Circle(int r, String n) {...}

public static void main(String[] args) {
  Circle pizza = new Circle(10, "Domino");
}
```

```
public Circle() {...}
public Circle(int r, String n) {...}

public static void main(String[] args) {
  Circle pizza = new Circle(10, "Domino");
  Circle donut = new Circle();
}
```

Usage of the constructor

```
public class Circle {
  int radius;
  String name;

  public Circle() {
    radius = 1;
    name = "No Name";
  }

  public Circle(int r, String n) {
    radius = r;
    name = n;
  }

  public double getArea() {
    return 3.14 * radius * radius;
  }

  public static void main(String[] args) {
    Circle pizza = new Circle(10, "Domino");
    Circle donut = new Circle();
  }
}
```

Types of constructors



- Default constructor (기본 생성자)
  - no-argument (parameter) constructor
  - automatically provided by Java if no other constructors are explicitly defined within the class
    - class 내에 생성자가 선언되어 있지 않을 때 자동으로 호출됨

```
public class Example {
  public Example() {...}
}
```

```
public class Circle {
   int radius;
   void set(int r) { radius = r; }
   double getArea() { return 3.14*radius*radius; }


   public static void main(String[] args) {
      Circle pizza = new Circle();
      pizza.set(5);
      System.out.println(pizza.getArea());
   }
}
```

컴파일러에 의해 기본 생성자 자동 삽입

```
public Circle() { }
```

호출

Types of constructors

- Parameterized constructor
  - allows passing arguments to set the initial state of an object when it is created
  - can take any number of parameters
  - class 내에 생성자가 선언되어 있을 때 ➔ default constructor 생성하지 않음

```
public class Example {
  public Example(int value, ...) {...}
}
```

```
public class Circle {
   int radius;
   void set(int r) { radius = r; }
   double getArea() { return 3.14*radius*radius; }

   public Circle(int r) {
      radius = r;
   }

   public static void main(String[] args) {
      Circle pizza = new Circle(10);
      System.out.println(pizza.getArea());

      Circle donut = new Circle();
      System.out.println(donut.getArea());
   }
}
```

public Circle() { }

호출

오류

컴파일 오류.
해당하는 생성자 없음

Keyword ‘this’

- keyword ‘this’
  - a reference variable that refers to the current object (객체 자신에 대한 레퍼런스)
    - the object whose method of constructor is being called
  - usage: this.[member] or this()

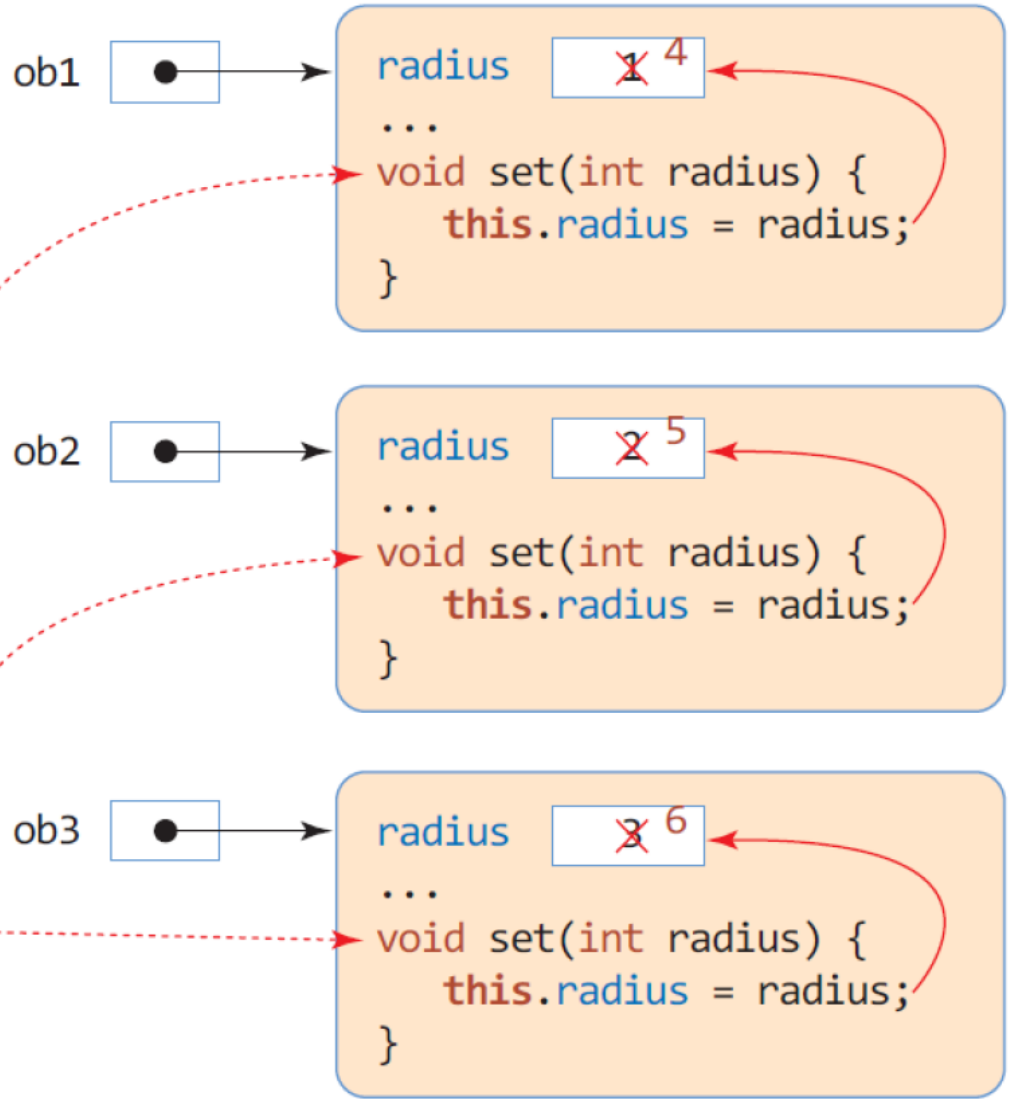
```
public class Circle {
  int radius;
  String name;

  public Circle() {
    radius = 1;
    name = "No Name";
  }

  public Circle(int r, String n) {
    radius = r;
    name = n;
  }
}
```

```
public class Circle {
  int radius;
  String name;

  public Circle() {
    this.radius = 1;
    this.name = "No Name";
  }

  public Circle(int radius, String name) {
    this.radius = radius;
    this.name = name;
  }
}
```

Keyword ‘this’

- Example of ‘this’

```
public class Circle {
    int radius;
    public Circle(int radius) {
        this.radius = radius;
    }
    void set(int radius) {
        this.radius = radius;
    }

    public static void main(String[] args) {
        Circle ob1 = new Circle(1);
        Circle ob2 = new Circle(2);
        Circle ob3 = new Circle(3);

        ob1.set(4);
        ob2.set(5);
        ob3.set(6);
    }
}
```



Keyword ‘this’

- this()
  - call the other constructor in the same class
  - just use in the constructor (can not refer out of the constructor method)
  - MUST BE the FIRST STATEMENT in the constructor body where it is used

오류

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

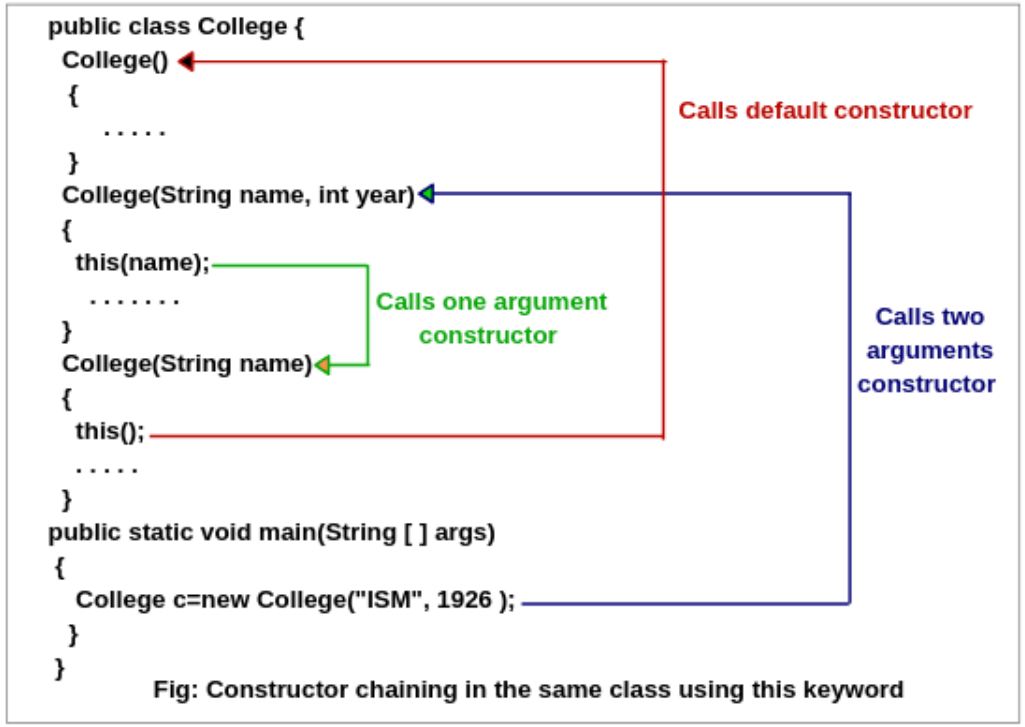


Fig: Constructor chaining in the same class using this keyword

Keyword ‘this’

- Example of this() for ‘Book’ class

```
public class Book {
  String title;
  String author;
  void show() {
    System.out.println(title + " " + author);
  }

  public Book() {
    this("", "");
    System.out.println("생성자 호출");
  }

  public Book(String title) {
    this(title, "작자미상");
  }

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

```
  public static void main(String[] args) {
    Book javaBook = new Book("Java", "김기태");
    Book bible = new Book("Bible");
    Book emptyBook = new Book();
    bible.show();
  }
```

```
생성자 호출
Bible 작자미상
```

Keyword 'this'

Rectangle
-int width -int height
+Rectangle() +Rectangle(int size) +Rectangle(int width, int height) +int getArea()

- Example of this() for 'Rectangle' class

```
public class Rectangle {
  private int width, height;
  public Rectangle() { this(0, 0); }
  public Rectangle(int size) { this(size, size); }
  public Rectangle(int width, int height) { this.width = width;  this.height = height;  }
  public int getArea() { return width * height; }
```

```
public static void main(String[] args) {
  Rectangle rect1 = new Rectangle(5, 20);
  Rectangle rect2 = new Rectangle(5);
  Rectangle rect3 = new Rectangle();

  System.out.println("Width:"+rect1.width+", height:"+rect1.height+", area:"+rect1.getArea());
  System.out.println("Width:"+rect2.width+", height:"+rect2.height+", area:"+rect2.getArea());
  System.out.println("Width:"+rect3.width+", height:"+rect3.height+", area:"+rect3.getArea());
}
```

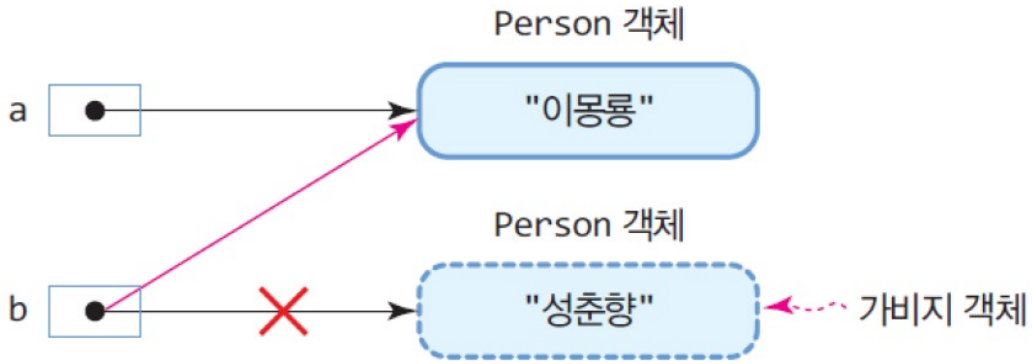
```
Width:5, height:20, area:100
Width:5, height:5, area:25
Width:0, height:0, area:0
```

Destruction of objects

- Objects are managed by “Garbage Collector (GC)”, a part of the Java Virtual Machine (JVM)
  - C/C++ ➔ objects are managed by programmers themselves using ‘delete’ keyword

- Garbage
  - memory that is occupied by objects that are no longer accessible or needed by a program
  - 가리키는 레퍼런스가 하나도 없는 객체 또는 더 이상 접근할 수 없어서 사용할 수 없는 메모리

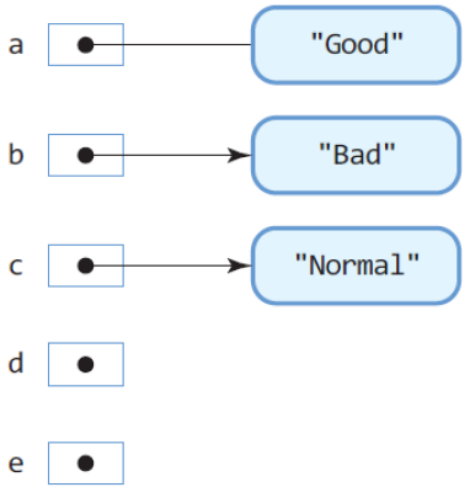
```
Person a, b;
a = new Person("이몽룡");
b = new Person("성춘향");

b = a; // b가 가리키던 객체는 가비지가 됨
```



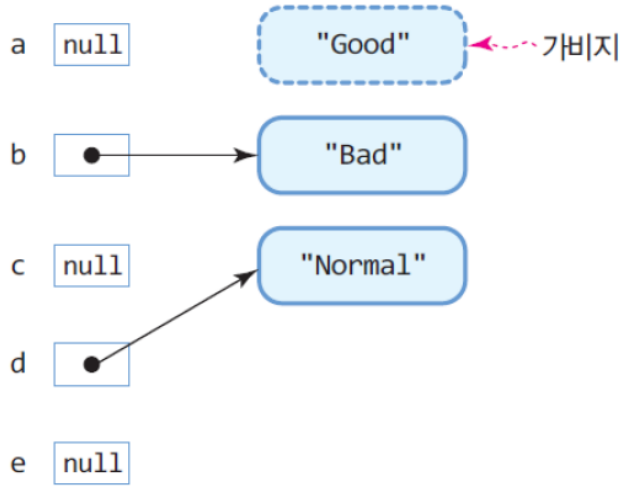
Destruction of objects

- **Garbage 발생 예제**

```
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

Destruction of objects

- Garbage collection
  - JVM의 garbage collector가 자동으로 garbage 값 수집 및 반환
  - 가용 메모리 공간이 일정 이하로 부족할 때 작동될 수 있음
    - ➔ garbage 수거하여 가용 메모리 공간 확보
  - 강제로 garbage collection 수행 할 수 있음
    - ➔ System.gc();
      - 강제로 모든 garbage 회수 요청 (JVM 판단 하에 진행되지 않을 수도 있음)

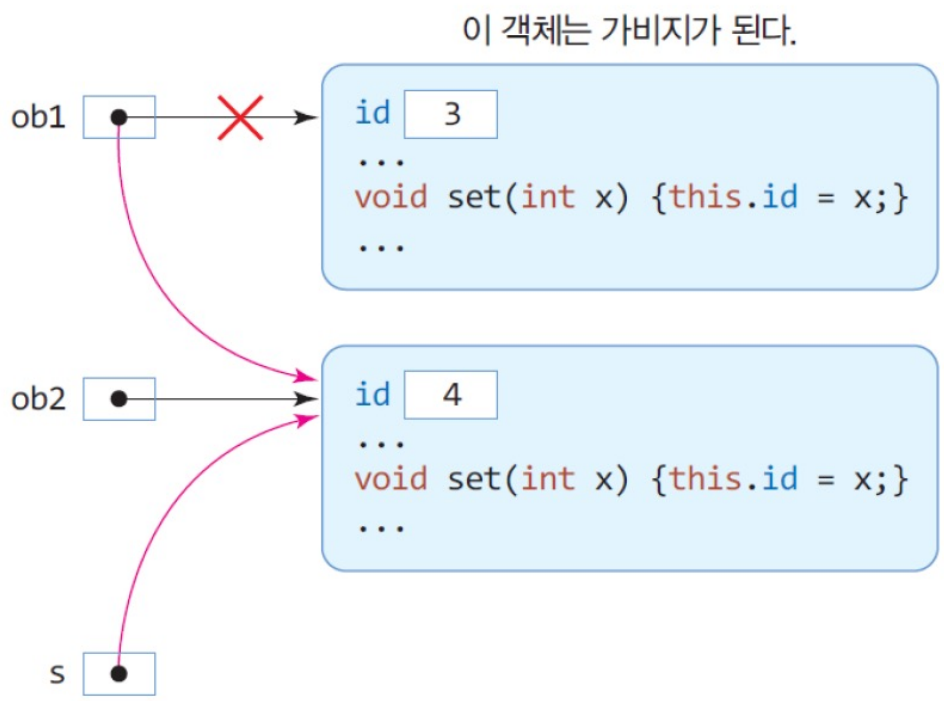
Object substitution (객체 치환)

- Garbage is generated by object substitution

```
public class Samp {
    int id;
    public Samp(int x) {this.id = x;}
    public void set(int x) {this.id = x;}
    public int get() {return this.id;}

    public static void main(String [] args) {
        Samp ob1 = new Samp(3);
        Samp ob2 = new Samp(4);
        Samp s;

        s = ob2;
        ob1 = ob2; // 객체의 치환
        System.out.println("ob1.id="+ob1.get());
        System.out.println("ob2.id="+ob2.get());
    }
}
```

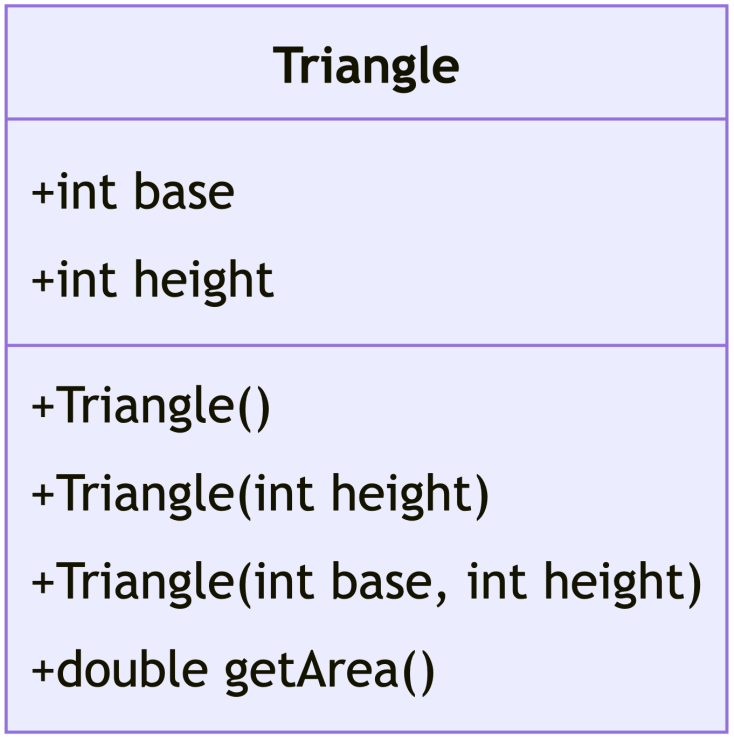
객체 치환

```
ob1.id=4
ob2.id=4
```



Examples and practices for class design and constructors

- 다음 class diagram에 따라 'Triangle' class와 메소드의 기능을 구현하고 main()에서 테스트 해보세요.
  - file path and name: Chap06Example/Triangle.java
  - class diagram
    - base: 밑변
    - height: 높이
    - Triangle(): 밑변과 높이가 5로 같음
    - Triangle(int height): 밑변과 높이가 같음

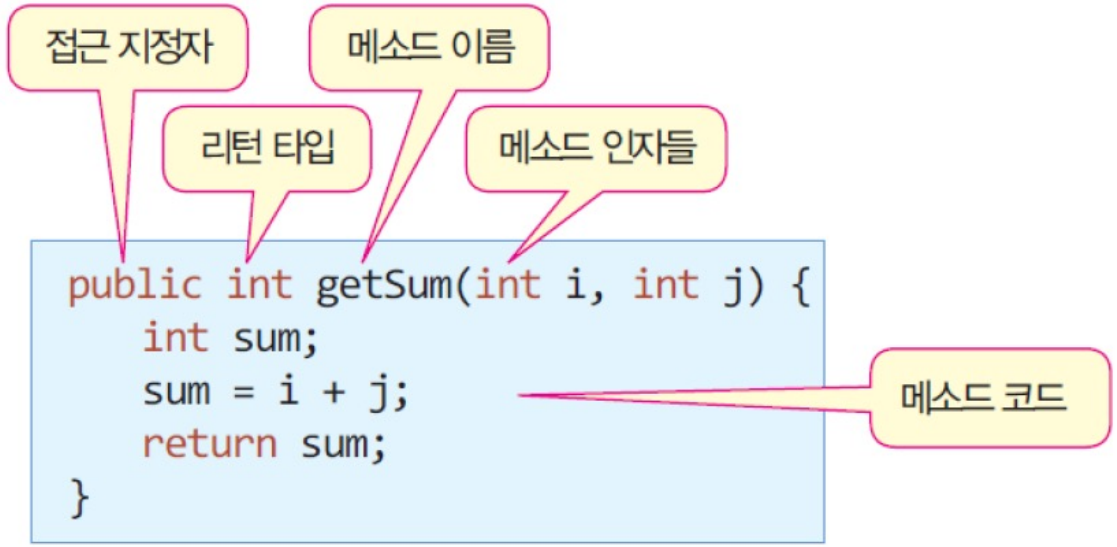


3. Methods with objects

Recall: method

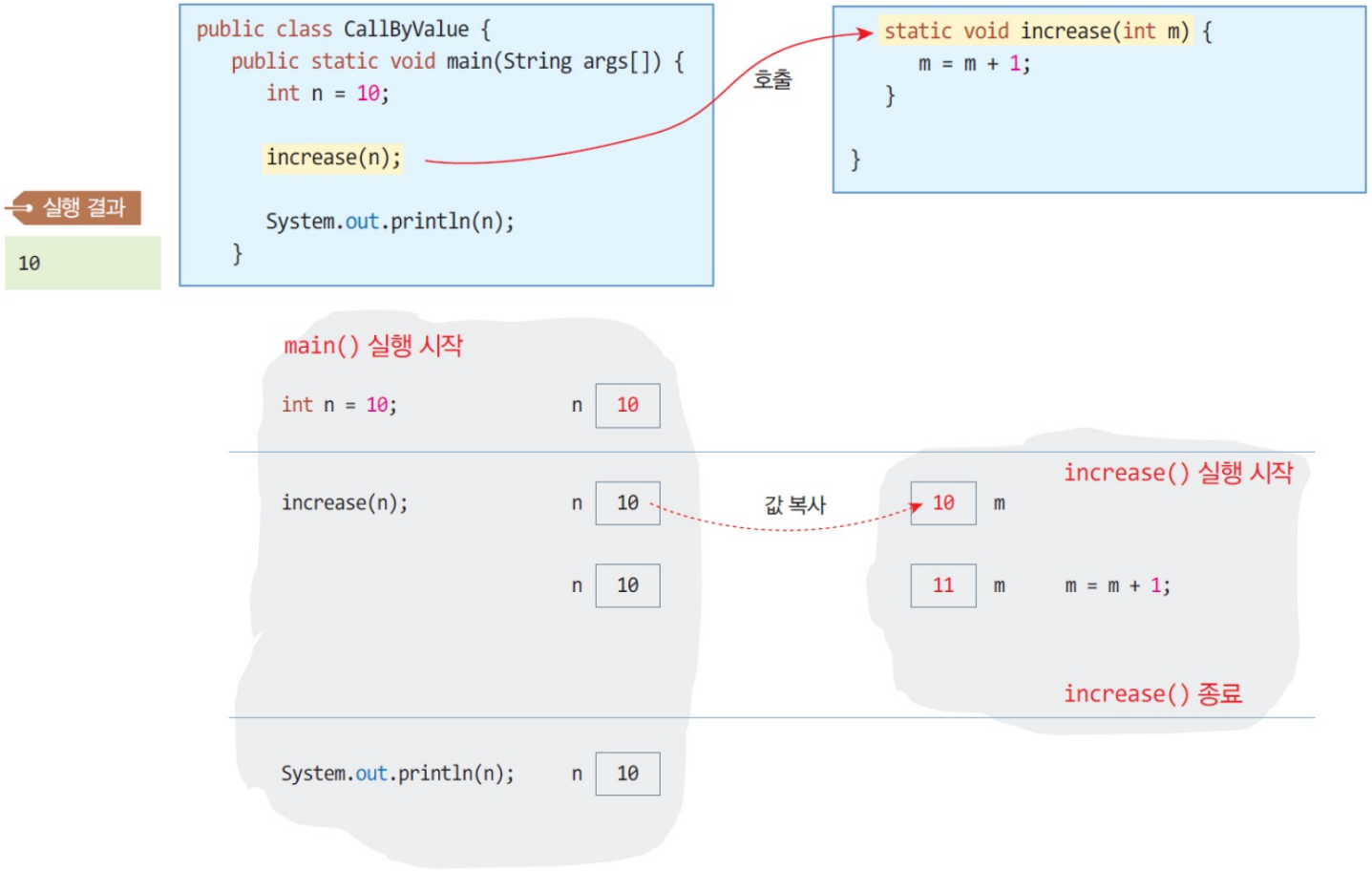
- Rule
  - all methods should exist in class ← Encapsulation

- Format of general method



Recall: Call by value

- 기본 데이터 타입의 값이 전달되는 경우
  - 호출자가 건네는 값이 매개변수(parameter)에 복사되어서 전달
  - 함수 내에서 실제 인자의 값은 변경 불가능



Recall: Call by value

- main 메소드 내 x 값은 외부 함수에서 바뀌지 않음

  ➔ call by value (값에 의한 호출)

```
public static void printX(int x) {
  System.out.println("X in printX method = " + x);
  x++;
  System.out.println("X in printX method = " + x);
}

public static void main(String[] args) {
  int x = 10;
  System.out.println("X in main method = " + x);
  printX(x);
  x = 50;
  System.out.println("X in main method = " + x);
}
```

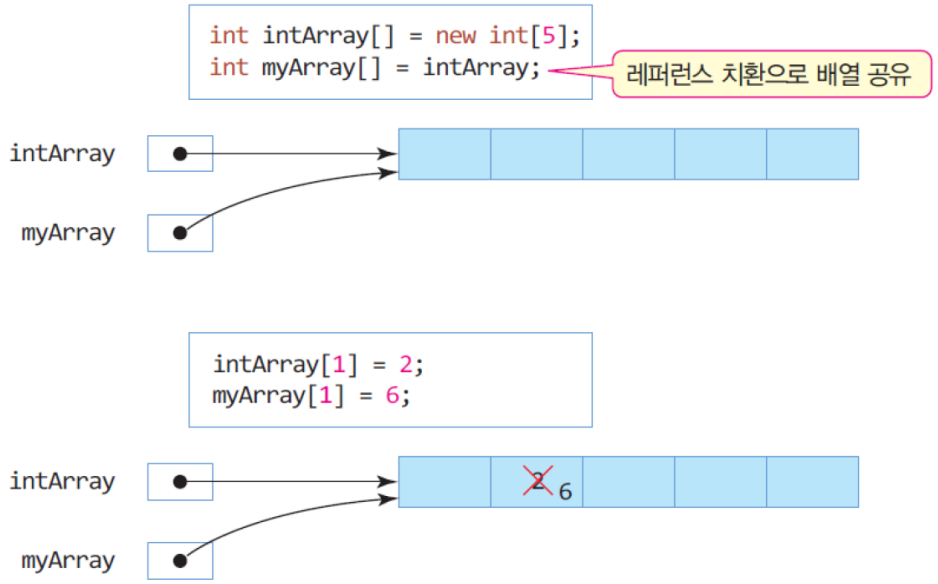
```
X in main method = 10
X in printX method = 10
X in printX method = 11
X in main method = 50
```

Recall: Call by reference

- Array reference assignment and sharing
  - one array can be manipulated and accessed through different reference variables
    - when you assign one array reference variable to another, both variables refer to the same array in the memory

```
int[] intAray = {1, 2, 3, 4, 5};
int[] myArray = intAray;

intAray[2] = 2;
intAray[2] = 6;

System.out.println("intArray: " + intAray[2]);
System.out.println("myArray: " + myArray[2]);
```

```
intArray: 6
myArray: 6
```



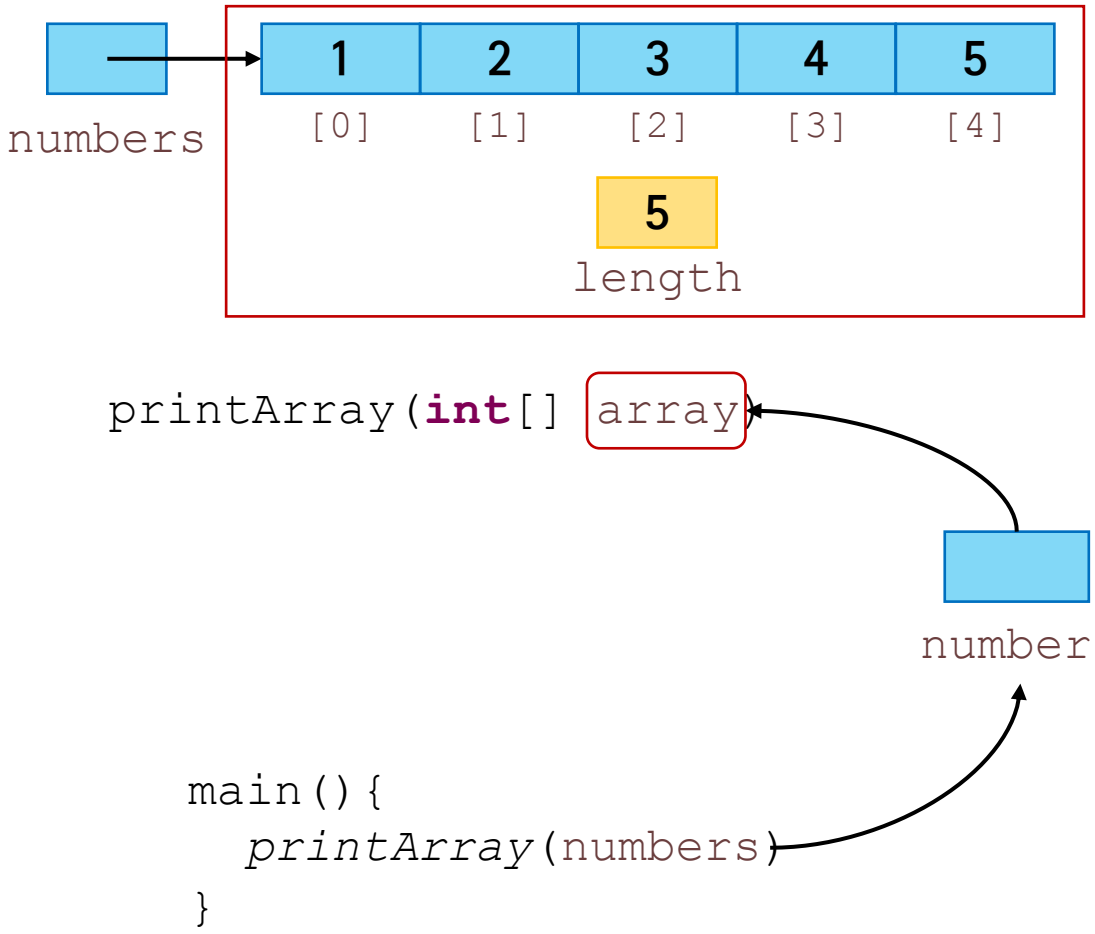
Recall: Call by reference

- Arrays can be passed to methods as "reference"
  - not the actual array itself

```
public static void printArray(int[] array) {
  for (int element : array) {
    System.out.print(element + " ");
  }
  System.out.println();
}
```

```
public static void main(String[] args) {
  int[] numbers = {1, 2, 3, 4, 5};
  System.out.println("Array :");
  printArray(numbers);
}
```

```
Array :
1 2 3 4 5
```



array의 reference가 method에 전달 ➔ 함수 내에서 값을 조작하면 실제 그 값이 바뀜

➔ call by reference (레퍼런스에 의한 호출)

Objects as parameters

- 객체가 메소드의 인자로 전달되는 경우
  - 객체의 reference만 전달
  - 매개 변수가 실제 인자 객체를 공유함 ➔ 값 변경 가능

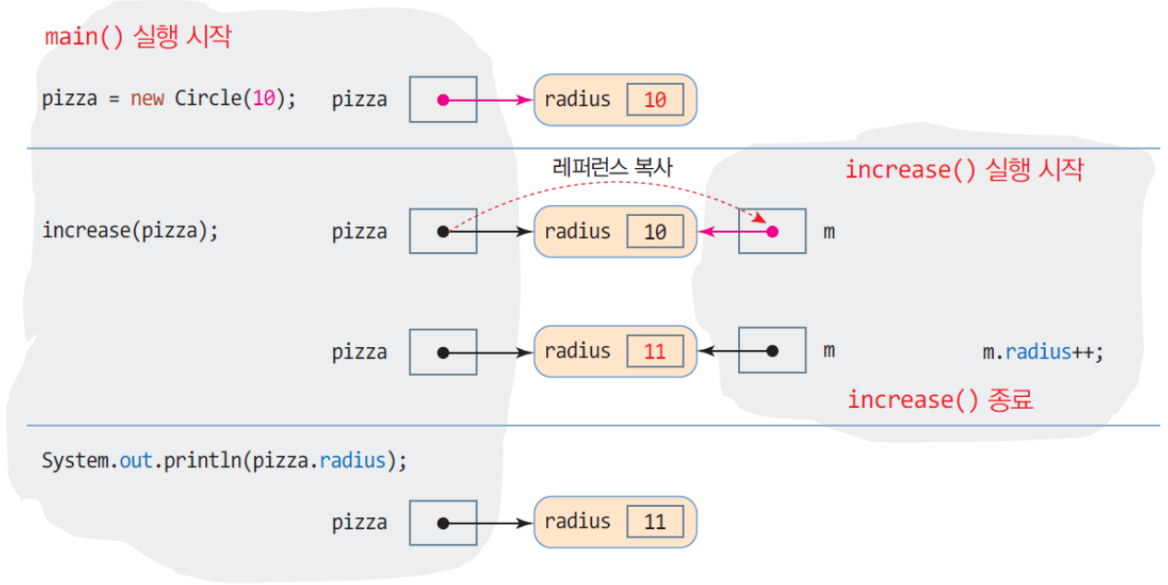
```
public class ReferencePassing {
   public static void main (String args[]) {
      Circle pizza = new Circle(10);

      increase(pizza);

      System.out.println(pizza.radius);
   }
```

호출

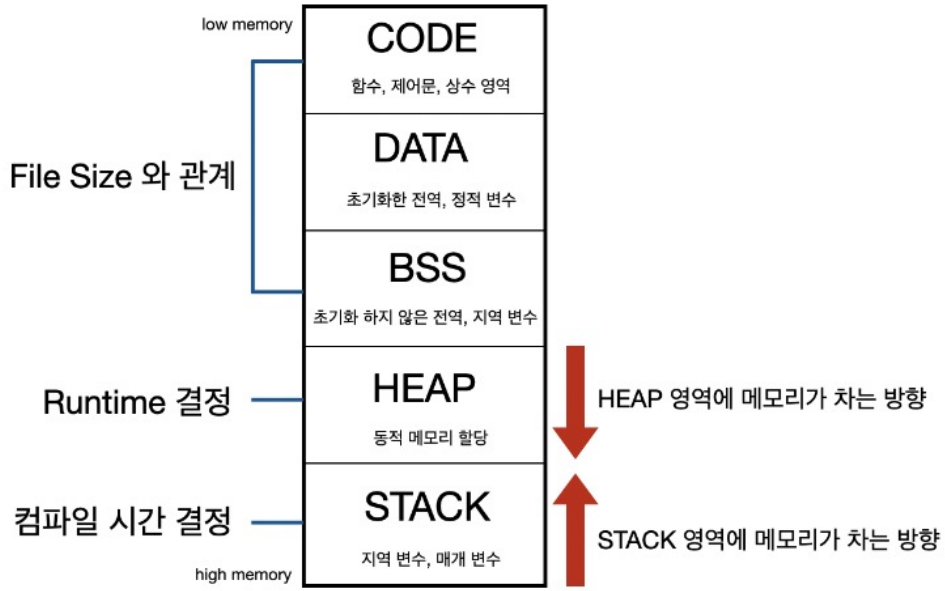
```
   static void increase(Circle m) {
      m.radius++;
   }

}
```

실행 결과

```
11
```



Keyword ‘static’ and ‘final’



- Note: Memory 구조
  - code 영역
    - 실제 실행되는 프로그램의 코드가 저장되는 영역
    - CPU는 code영역에 저장된 명령여를 하나씩 처리
  - data 영역
    - 전역 변수와 정적(static) 변수가 저장되는 영역 (프로그램 실행 시 할당, 프로그램 종료 후 소멸)
  - stack 영역
    - 메소드 호출과 관계되는 지역 변수와 매개 변수가 저장되는 영역 (함수의 호출이 완료되면 소멸)
  - heap 영역
    - reference data type (array, string etc.) 저장
    - ‘new’로 생성되는 모든 변수/객체 저장

Keyword ‘static’ and ‘final’

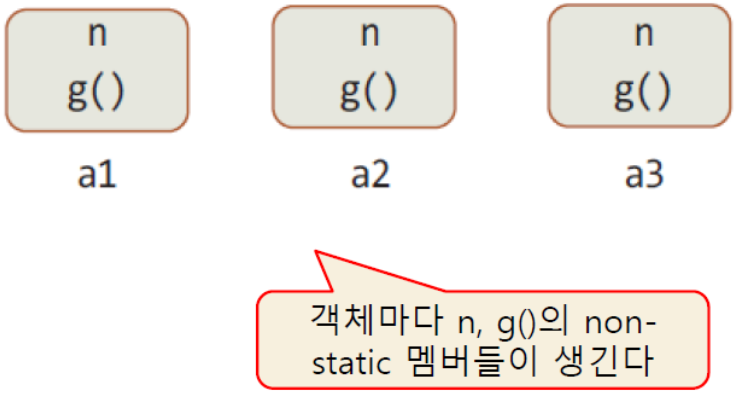
- Static
  - variable, method, class에 사용되어 “정적” 멤버로 변환
    - 정적멤버: class의 instance가 아닌 class 자체에 속하게 되는 것
      - ➔ 객체의 생성 없이 class 이름을 통해 직접 접근할 수 있음
      - 모든 인스턴스에서 공통으로 사용되는 변수
        - ➔ 유틸리티 함수나 상수 값을 지정 시 자주 활용됨
  - lifecycle: 메모리에 할당되어 프로그램이 종료될 때 소멸

Keyword ‘static’ and ‘final’

- Static: 일반 멤버 변수/함수와 정적 멤버 변수/함수의 비교

```
class A {
    int n;
    void g() {...}
}
```

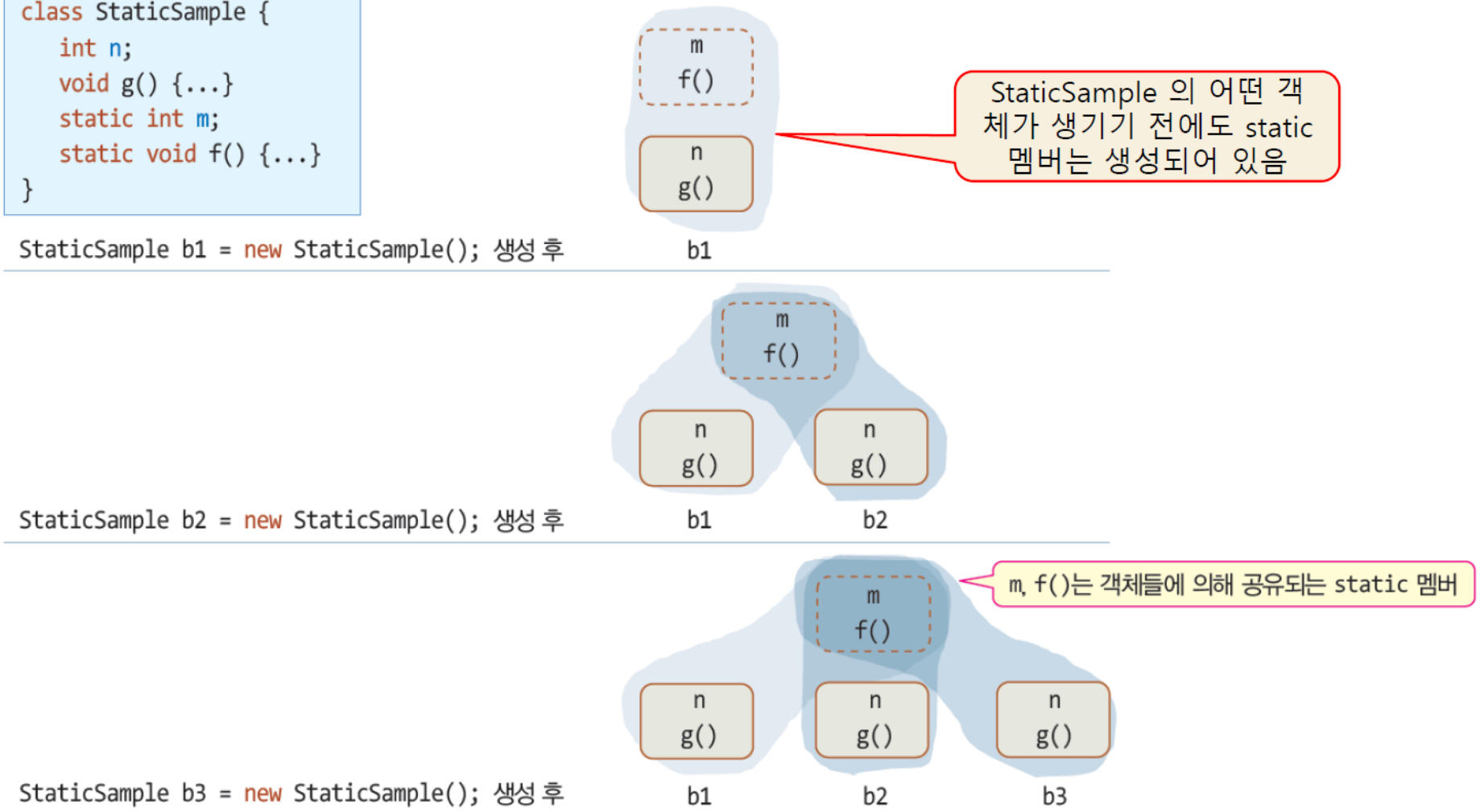
```
A a1 = new A();
A a2 = new A();
A a3 = new A();
```



Keyword ‘static’ and ‘final’

- Static: 일반 멤버 변수/함수와 정적 멤버 변수/함수의 비교

```
class StaticSample {
   int n;
   void g() {...}
   static int m;
   static void f() {...}
}
```



Keyword ‘static’ and ‘final’

- Static 멤버의 사용
  - class 이름으로 접근 가능

```
StaticSample.m = 3;    // 클래스 이름으로 static 필드 접근
StaticSample.f();      // 클래스 이름으로 static 메소드 호출
```

  - 객체의 멤버로도 접근 가능

```
StaticSample b1 = new StaticSample();

b1.m = 3;  // 객체 이름으로 static 필드 접근
b1.f();    // 객체 이름으로 static 메소드 호출
```

  - Non-static 멤버는 클래스 이름으로 접근 불가능


```
StaticSample.n = 5;    // n은 non-static이므로 컴파일 오류
StaticSample.g();      // g()는 non-static이므로 컴파일 오류
```

Keyword ‘static’ and ‘final’

- Static: 일반 멤버 변수/함수와 정적 멤버 변수/함수의 비교

	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 클래스당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간에 생성 • 클래스 멤버라고 부름
시간적 특성	객체 생성 시에 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	공유되지 않음 • 멤버는 객체 내에 각각 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

Keyword ‘static’ and ‘final’

- Static 변수 사용시 제약 조건 및 주의 사항
  - 전역 변수와 전역 함수를 만들 때 주로 활용
  - 클래스 내에서 공유 멤버를 만들 때 활용

```
public class Calc {
  public static int abs(int a) { return a>0?a:-a;}
  public static int max(int a, int b) { return (a>b)?a:b; }
  public static int min(int a, int b) { return (a>b)?b:a; }

  public static void main(String[] args) {
    System.out.println(Calc.abs(-5));
    System.out.println(Calc.max(10, 8));
    System.out.println(Calc.min(-3, -8));
  }
}
```

```
5
10
-8
```

Keyword ‘static’ and ‘final’

- Static 변수 사용시 제약 조건 및 주의 사항
  - 제약 조건
    - 1) static method는 오직 static 멤버만 접근할 수 있음

```
class StaticMethod {
    int n;
    void f1(int x) {n = x;} // 정상
    void f2(int x) {m = x;} // 정상
    static int m;
    static void s1(int x) {n = x;} // 컴파일 오류. static 메소드는 non-static 필드 사용 불가
    static void s2(int x) {f1(3);} // 컴파일 오류. static 메소드는 non-static 메소드 사용 불가
    static void s3(int x) {m = x;} // 정상. static 메소드는 static 필드 사용 가능
    static void s4(int x) {s3(3);} // 정상. static 메소드는 static 메소드 호출 가능
}
```

Keyword ‘static’ and ‘final’

- Static 변수 사용시 제약 조건 및 주의 사항
  - 제약 조건
    - 2) static method는 this 사용 불가
      - static method는 객체 없이도 사용가능 하므로, this reference 사용할 수 없음

오류
```
static void f() { this.n = x;} // 오류. static 메소드에서는 this 사용 불가능
```
오류
```
static void g() { this.m = x;} // 오류. static 메소드에서는 this 사용 불가능
```

Keyword ‘static’ and ‘final’

- final
  - variable, method, class에 사용되어 해당 요소를 더 이상 변경할 수 없도록 함
    - 해당 요소를 “최종적”이라고 표시
  - final + variable ➔ 더 이상 변경할 수 없는 값 == 상수
    - ex) public final double PI = 3.14;
    - ex) public static final double PI = 3.14;
  - final + method ➔ 더 이상 변경할 수 없는 메소드 ➔ overriding 불가능
    - overriding: overloading과 다른 개념 (상속 파트에서 설명)
  - final + class ➔ 더 이상 변경할 수 없는 클래스 ➔ 상속 불가능

Keyword ‘static’ and ‘final’

- final을 이용한 상수
  - 상수는 선언 시 반드시 초기값을 지정해주어야 함
  - 상수는 실행 중 값 변경 불가능

```
class SharedClass {
        public static final double PI = 3.14;
}
```

```
public class FinalFieldClass {
        final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정

        void f() {
                int [] intArray = new int [ROWS]; // 상수 활용
                ROWS = 30; // 컴파일 오류 발생, final 필드 값을 변경할 수 없다.
        }
}
```

오류

Keyword ‘static’ and ‘final’

- Usage of ‘final’ keyword for variables

```
public class Circle {
  public static final double PI = 3.14159;
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public double getArea() {
    return PI * radius * radius;
  }

  public static void main(String[] args) {
    Circle circle = new Circle(5.0);
    System.out.println("원의 넓이: " + circle.getArea());
  }
}
```

4. Class array

Class array

- In Java, “object array” and “class array” are often used interchangeably

- Class array
  - to refer to an array that is specifically intended to hold objects of a particular class or its subclasses
    - 객체에 대한 배열

Class array generation

- 1) variable for the array reference declaration
- 2) generate reference array
- 3) generate element in each object (element) in array

```
Circle [] c;
c = new Circle[5];

for(int i=0; i<c.length; i++) // c.length는 배열 c의 크기로서 5
    c[i] = new Circle(i);
```

❶ Circle 배열에 대한 레퍼런스 변수 c 선언

❷ 레퍼런스 배열 생성

❸ 각 원소 객체 생성

```
for(int i=0; i<c.length; i++) // 모든 객체의 면적 출력
    System.out.print((int)(c[i].getArea()) + " ");
```

배열의 원소 객체 사용

Class array generation

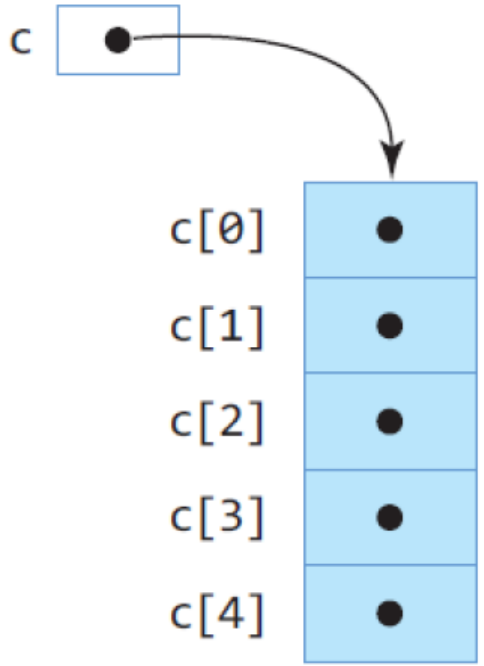
- Details

❶ 배열에 대한 레퍼런스 변수 선언

```
Circle[] c;
```

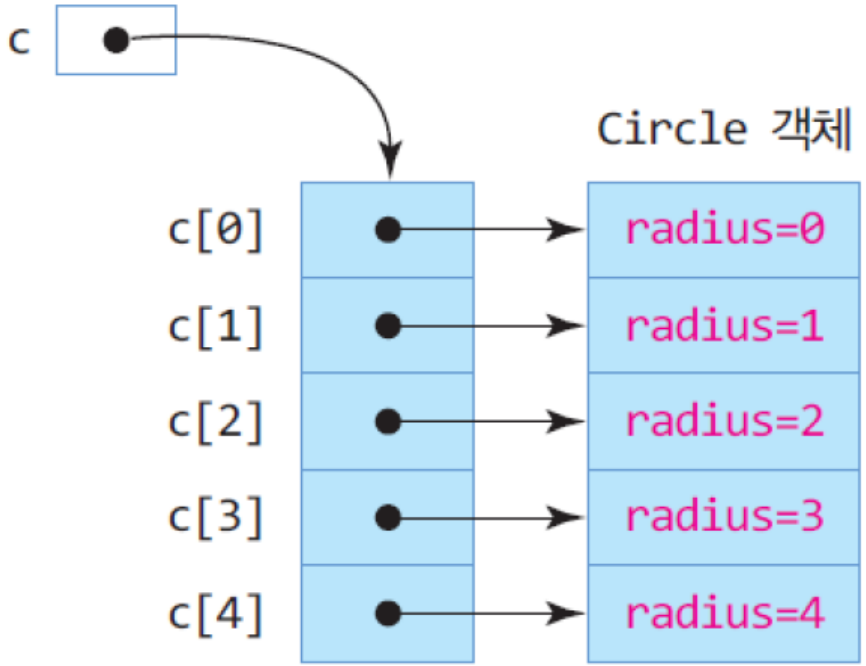
❷ 레퍼런스 배열 생성

```
c = new Circle[5];
```




```
for(int i=0; i<c.length; i++)
    c[i] = new Circle(i);
```

❸ 객체 생성



Class array generation

- Usage of class array for 'Circle' class
  - 기존에 만든 'Circle' class 활용하고, 'CircleArray' class 생성

```
public class CircleArray {

  public static void main(String[] args) {
    Circle[] c;
    c = new Circle[5];

    for (int i = 0; i < c.length; i++) {
      c[i] = new Circle(i);
    }

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

```
0  3  12  28  50
```

Class array generation

- Usage of class array for 'Book' class (+ use 'Scanner' class)
  - 사용자로부터 책의 제목과 저자를 2번씩 입력받고 출력하는 프로그램

```
public class BookArray {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Book[] book = new Book[2];

    for (int i = 0; i < book.length; i++) {
      System.out.print("Title: ");
      String title = scanner.nextLine();
      System.out.print("Author: ");
      String author = scanner.nextLine();
      book[i] = new Book(title, author);
    }
    for (int i = 0; i < book.length; i++) {
      System.out.print("(" + book[i].title + ", " + book[i].author + ")");
    }
  }
}
```

```
Title: 사랑의 기술
Author: 에리히 프롬
Title: 시간의 역사
Author: 스티븐 호킹
(사랑의 기술, 에리히 프롬)(시간의 역사, 스티븐 호킹)
```

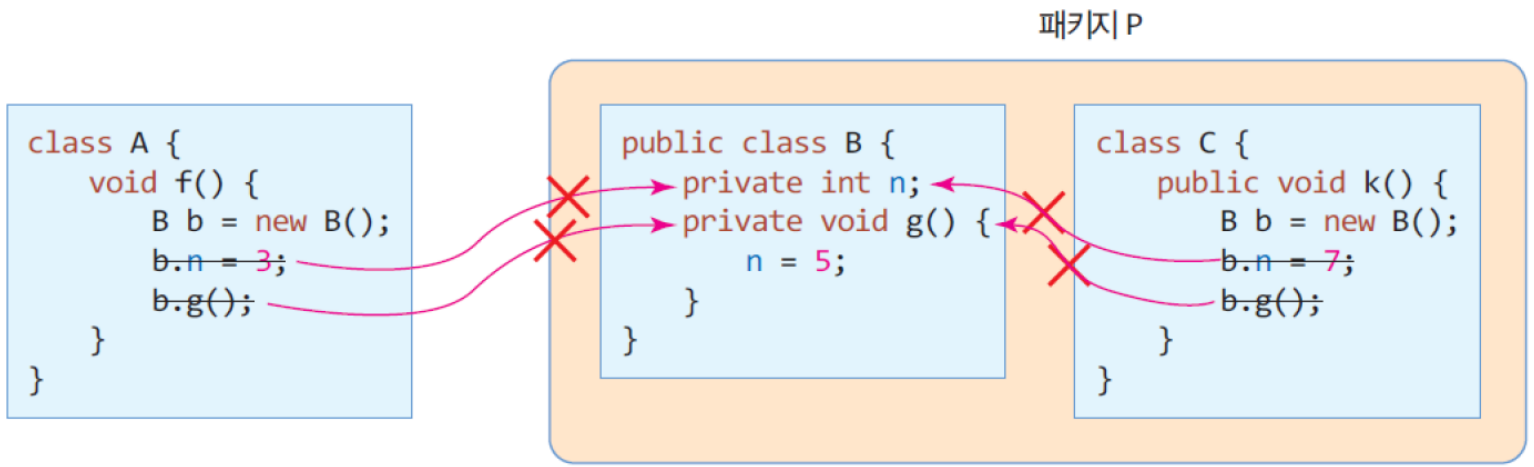
5. Access modifiers

Access modifier

- Access modifier (접근 지정자)
  - keywords that set the accessibility (visibility) of classes, variables, methods, and constructors
  - help to implement encapsulation in OOP by controlling access to the members of a class
  - four types
    - private
    - default (package-private)
    - protected
    - public

Types of access modifiers

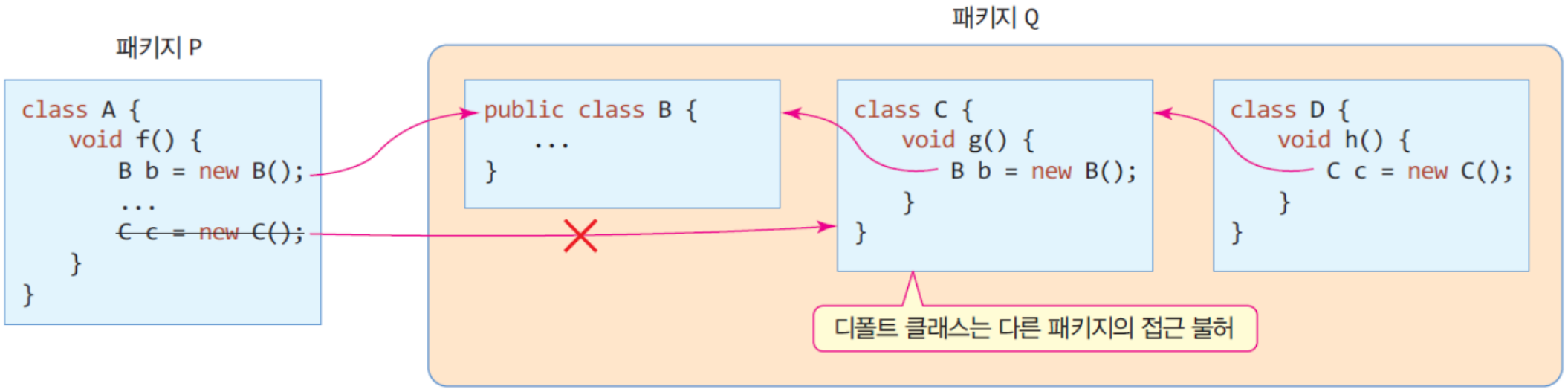
- Private
  - specifies that the member can only be accessed within its own class
  - symbol in class diagram: '-'
  - provides the highest level of encapsulation
  - not visible to any class other than the one in which they are declared
  - commonly used with variables and methods to high the internal data and implementation details of a class



Rectangle
-int width -int height
+Rectangle() +Rectangle(int size) +Rectangle(int width, int height) +int getArea()

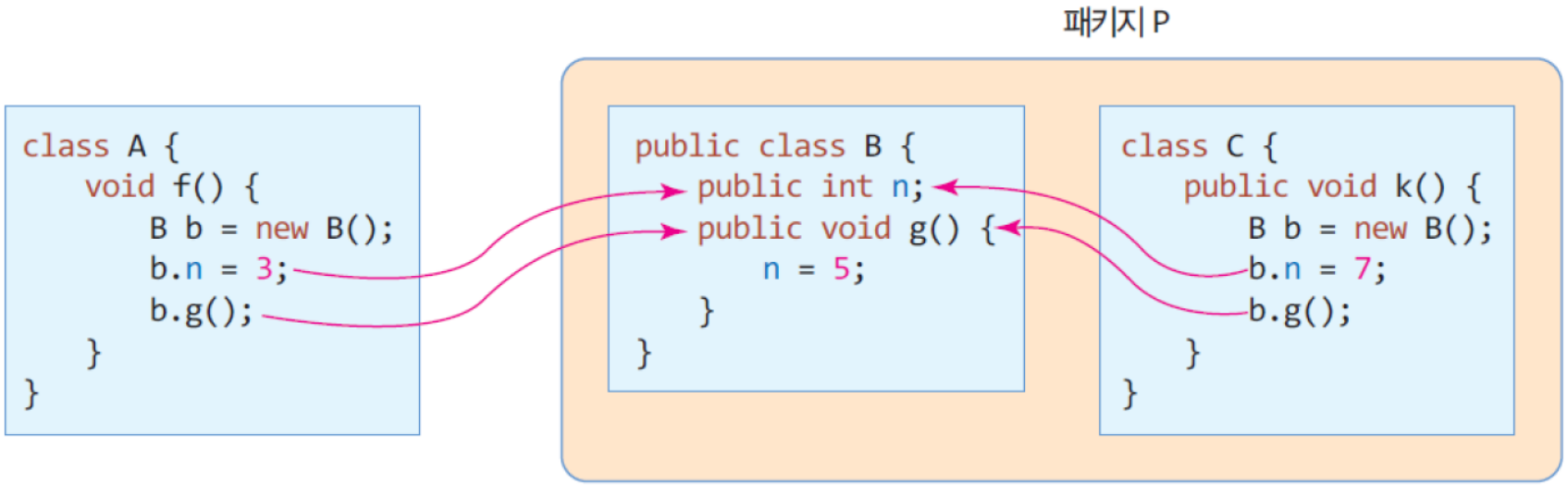
Types of access modifiers

- Default (package-private)
  - if no access modifiers is specified, the default access level is package-private
    - package-private: accessible only within its own package
  - provides more accessibility than 'private' but is still restrictive to outside classes not in the same package
  - useful for allowing classes within the same package to interact with each other without exposing their internals to the outside world



Types of access modifiers

- Public
  - sets no restrictions on access level
  - can be accessed from any other class



Usage of the access modifiers

- **멤버의 접근 지정자 활용 (오류 케이스)**

```
class SampleClass {
   public int field1;
   protected int field2;
   int field3;
   private int field4;
}

public class AccessEx {
   public static void main(String[] args) {
      SampleClass s = new SampleClass();
      s.field1 = 0;
      s.field2 = 1;
      s.field3 = 2;
      s.field4 = 3;
   }
}
```

AccessEx 클래스의 14번 라인에서 컴파일 오류 발생.
field4는 SampleClass의 private 멤버이므로 SampleClass 외의 다른 클래스에서 접근할 수 없다.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The field s.field4 is not visible
at AccessEx.main(AccessEx.java:14)
```

Tips: Scope of variable and method in Java class

- 일반적으로 변수는 'private', 메소드는 'public'
  - private 변수에 접근하기 위한 접근 메소드 (접근자)를 구현
    - setter(), getter()
  - 접근자를 통해 외부에서 변수에 접근함

```
private String name;
private String address;
private int age;
```

```
public String getName() { return this.name; }
public String getAddress() { return this.address; }
public int getAge() { return this.age; }

public void setName(String name) { this.name = name; }
public void setAddress(String address) { this.address = address; }
public void setAge(int age) { this.age = age; }
```

End of slide