



한림대학교 SW중심대학

생 초보를 위한 자바 프로그래밍

6장. 클래스

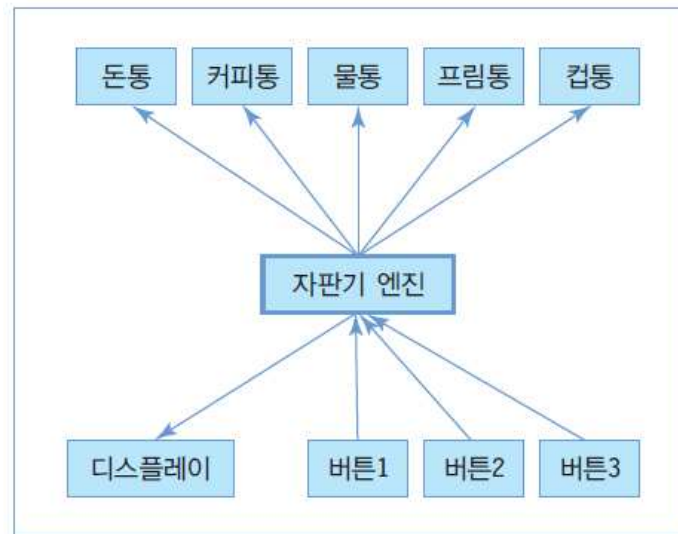
프로그래밍 문법

기본 문법

객체 관련 문법

1절. 객체 지향 프로그래밍

- 객체 지향 프로그래밍
 - OOP: Object Oriented Programming
 - 부품 객체를 먼저 만들고 이것들을 하나씩 조립해 완성된 프로그램을 만드는 기법



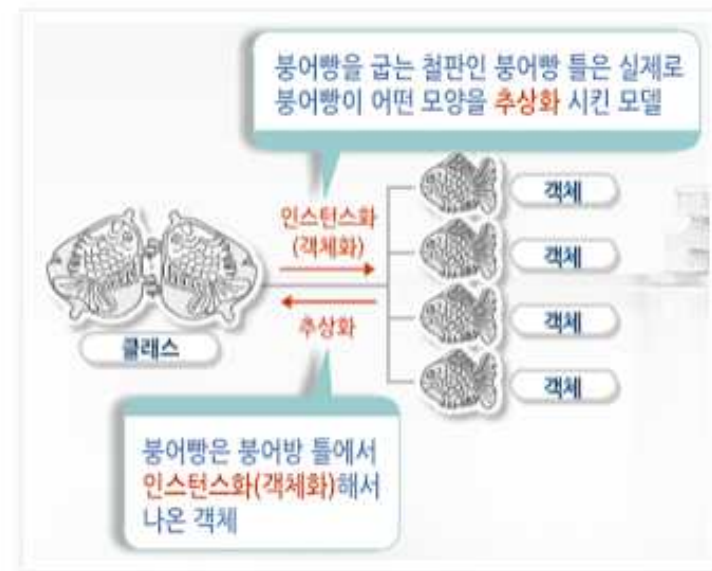
2절. 객체와 클래스

■ 클래스

- 객체의 속성(state)과 행위(behavior) 선언
- 객체의 설계도 혹은 틀

■ 객체

- 클래스의 틀로 찍어낸 실체
 - 프로그램 실행 중에 생성되는 실체
 - 메모리 공간을 갖는 구체적인 실체
 - 인스턴스(instance)라고도 부름



■ 사례

- | | |
|----------------|---------------------|
| ■ 클래스: 소나타자동차, | 객체: 출고된 실제 소나타 100대 |
| ■ 클래스: 벽시계, | 객체: 우리집 벽에 걸린 벽시계들 |
| ■ 클래스: 책상, | 객체: 우리가 사용중인 실제 책상들 |

3절. 클래스 선언

- 클래스의 이름
 - 자바 식별자 작성 규칙에 따라야

번호	작성 규칙	예
1	하나 이상의 문자로 이루어져야 한다.	Car, SportsCar
2	첫 번째 글자는 숫자가 올 수 없다.	Car, 3Car(x)
3	'\$', '_', ' ' 외의 특수 문자는 사용할 수 없다.	\$Car, _Car, @Car(x), #Car(x)
4	자바 키워드는 사용할 수 없다.	int(x), for(x)

- 한글 이름도 가능하나, 영어 이름으로 작성
- 알파벳 대소문자는 서로 다른 문자로 인식
- 첫 글자와 연결된 다른 단어의 첫 글자는 대문자로 작성하는 것이 관례

Calculator, Car, Member, ChatClient, ChatServer, Web_Browser

3절. 클래스 선언

■ 형식

```
public class 클래스명 {  
  
}
```



```
public class Student {  
  
}
```

- 소스 파일당 하나의 클래스를 선언하는 것이 관례
 - 두 개 이상의 클래스도 선언 가능
 - 소스 파일 이름과 동일한 클래스만 public으로 선언 가능

예제

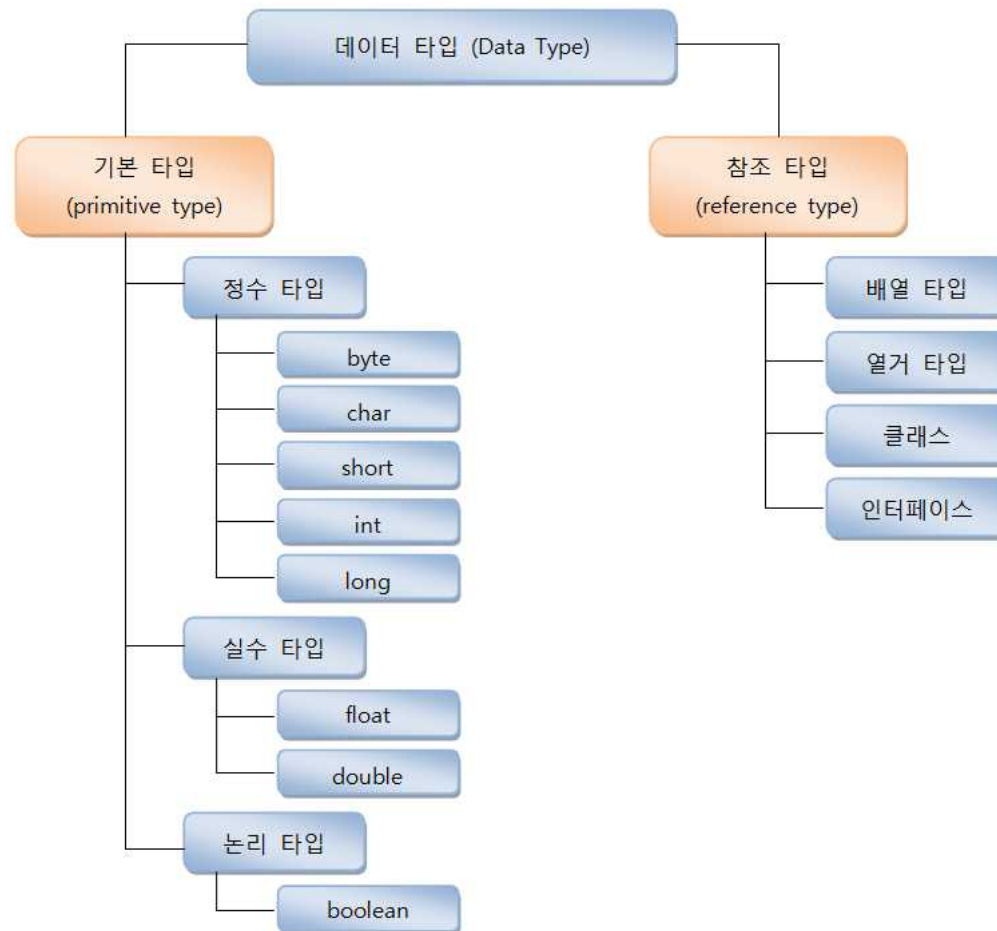
1. 클래스 선언방법
2. 객체 생성 방법 : new 이용

```
public class Student {  
  
}
```

```
public class StudentExample {  
    public static void main(String[] args) {  
        Student s1 = new Student();  
        System.out.println("s1 변수가 Student 객체를 참조합니다.");  
  
        Student s2 = new Student();  
        System.out.println("s2 변수가 또 다른 Student 객체를 참조합니다.");  
    }  
}
```

데이터 타입 분류

■ 데이터 타입 분류

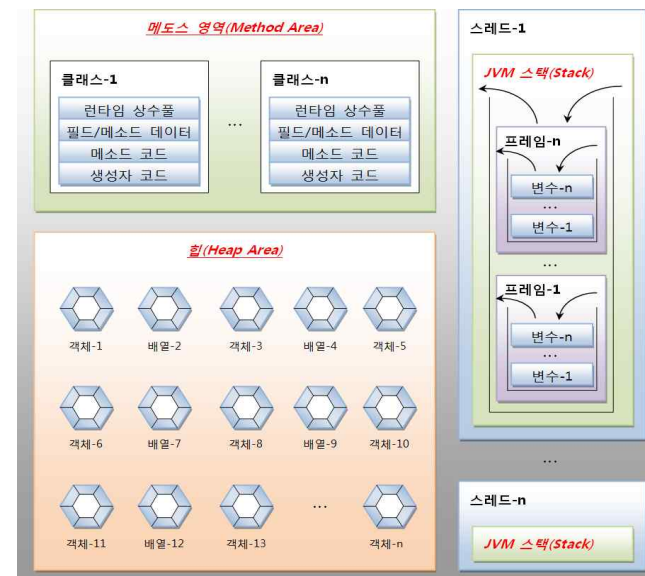


2절. 메모리 사용 영역

■ JVM이 사용하는 메모리 영역 메소드 영역

- **JVM** 시작할 때 생성
- 로딩된 클래스 바이트 코드 내용을 분석 후 저장
- 모든 스레드가 공유

Runtime Data Area



■ 힙 영역

- **JVM** 시작할 때 생성
- 객체/배열 저장
- 사용되지 않는 객체는 **Garbage Collector** 가 자동 제거

■ JVM 스택

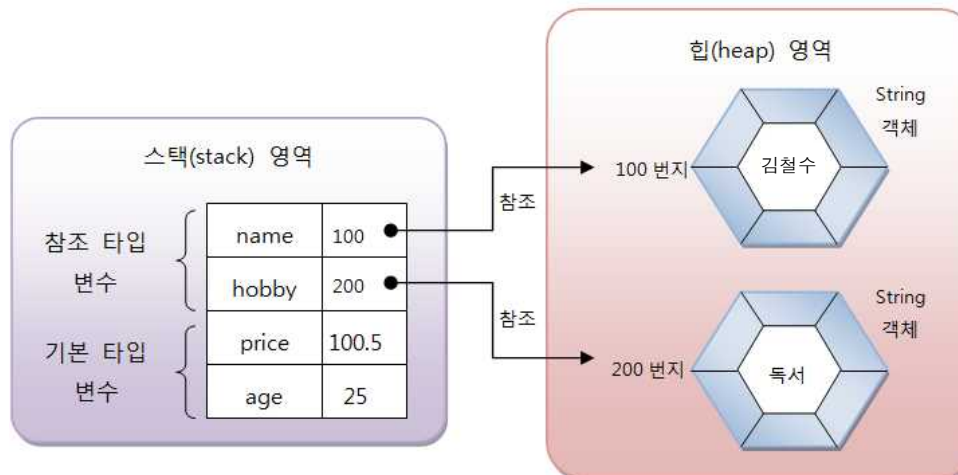
- 스레드 별 생성
- 메소드 호출할 때마다 **Frame**을 스택에 추가(push)
- 메소드 종료하면 **Frame** 제거(pop)

데이터 타입 분류

- 변수의 메모리 사용
 - 기본 타입 변수 – 실제 값을 변수 안에 저장
 - 참조 타입 변수 – 주소를 통해 객체 참조

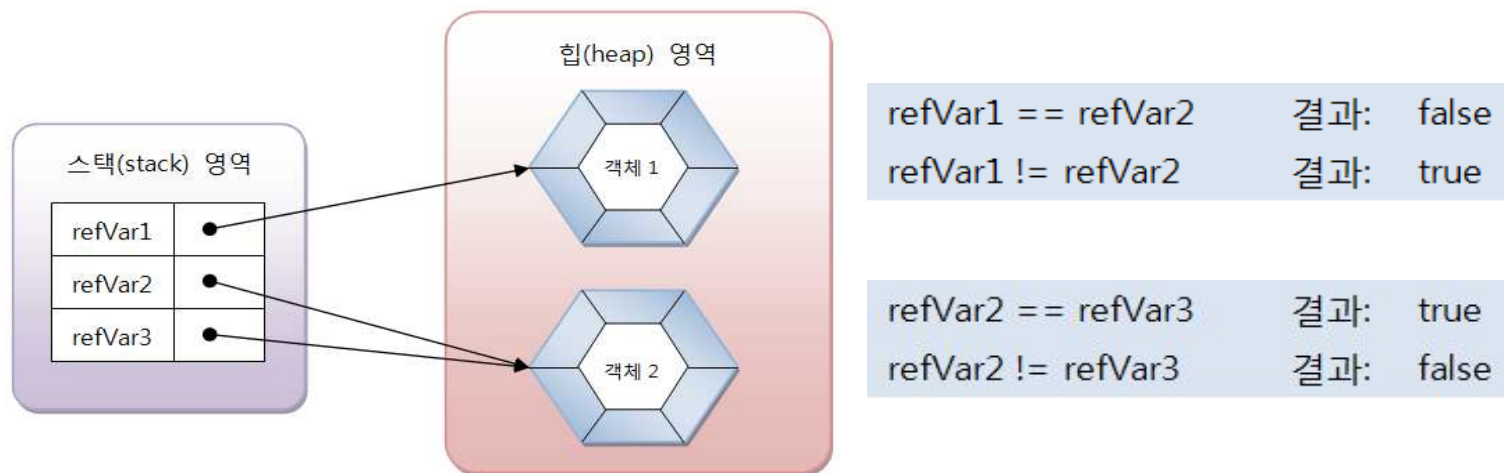
[기본 타입 변수]
int age = 25;
double price = 100.5;

[참조 타입 변수]
String name = "김철수";
String hobby = "독서";



참조 변수의 ==, != 연산

- 변수의 값이 같은지 다른지 비교
 - 기본 타입: **byte, char, short, int, long, float, double, boolean**
 - 의미 : 변수의 값이 같은지 다른지 조사
 - 참조 타입: 배열, 열거, 클래스, 인터페이스
 - 의미 : 동일한 객체를 참조하는지 다른 객체를 참조하는지 조사
→ **==, !=** 이용

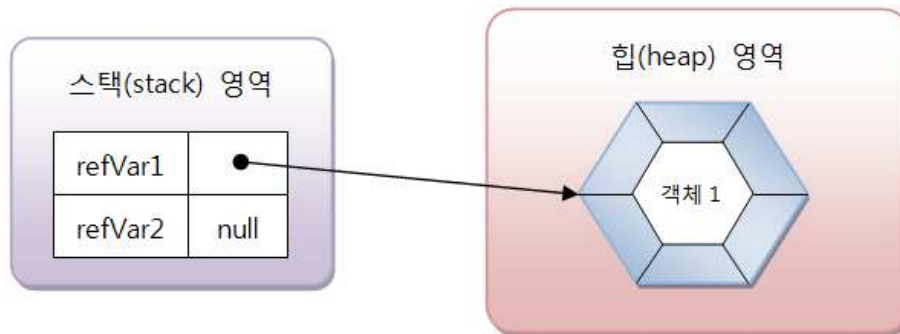


```
if( refVar2 == refVar3 ) { ... }
```

null과 NullPointerException

■ null(널)

- 변수가 참조하는 객체가 없을 경우 초기값으로 사용 가능
- 참조 타입의 변수에만 저장가능
- **null**로 초기화된 참조 변수는 스택 영역 생성



그림에서 refVar1 은 힙 영역의 객체를 참조하므로 연산의 결과는 다음과 같다.

refVar1 == null	결과: false
refVar1 != null	결과: true

refVar2 는 null 값을 가지므로 연산의 결과는 다음과 같다.

refVar2 == null	결과: true
refVar2 != null	결과: false

null과 NullPointerException

■ NullPointerException의 의미

■ 예외 (Exception)

- 사용자의 잘못된 조작 이나 잘못된 코딩으로 인해 발생하는 프로그램 오류

■ NullPointerException

- 참조 변수가 **null** 값을 가지고 있을 때
 - 객체의 필드나 메소드를 사용하려고 했을 때 발생

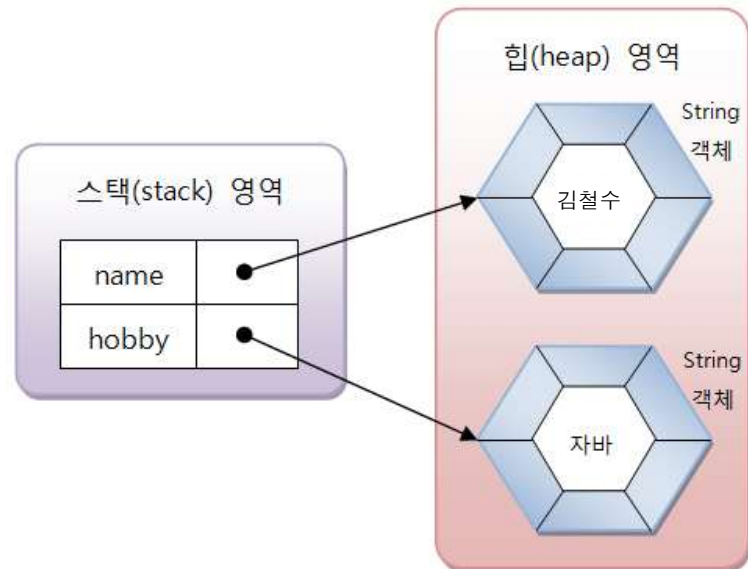
```
int[] intArray = null;  
intArray[0] = 10;      //NullPointerException
```

```
String str = null;  
System.out.println("총 문자수: " + str.length()); //NullPointerException
```

String 타입

- **String** 타입
 - 문자열을 저장하는 클래스 타입

```
String name = "김철수";  
String hobby = "독서";
```

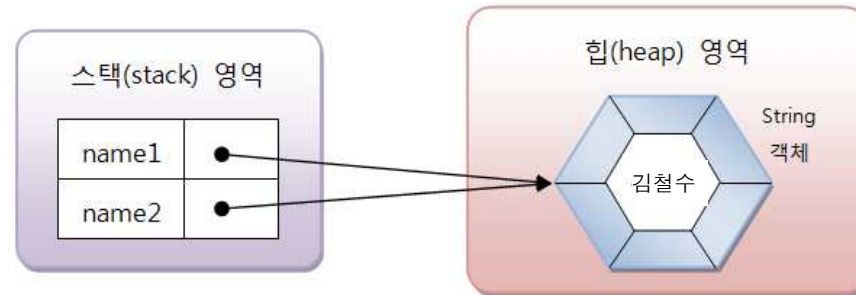


String 타입

■ String 타입

- 문자열 리터럴 동일하다면 **String** 객체 공유

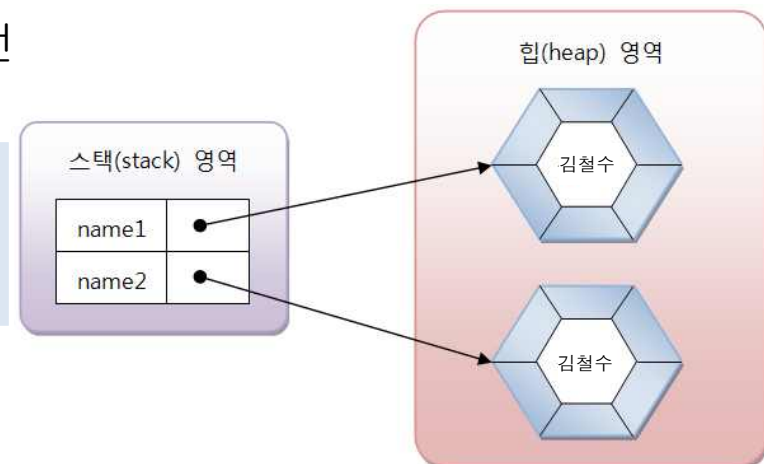
```
String name1 = "김철수";  
String name2 = "김철수";
```



■ **new** 연산자를 이용한 **String** 객체 생성

- 힙 영역에 새로운 **String** 객체 생성
- **String** 객체를 생성한 후 번지 리턴

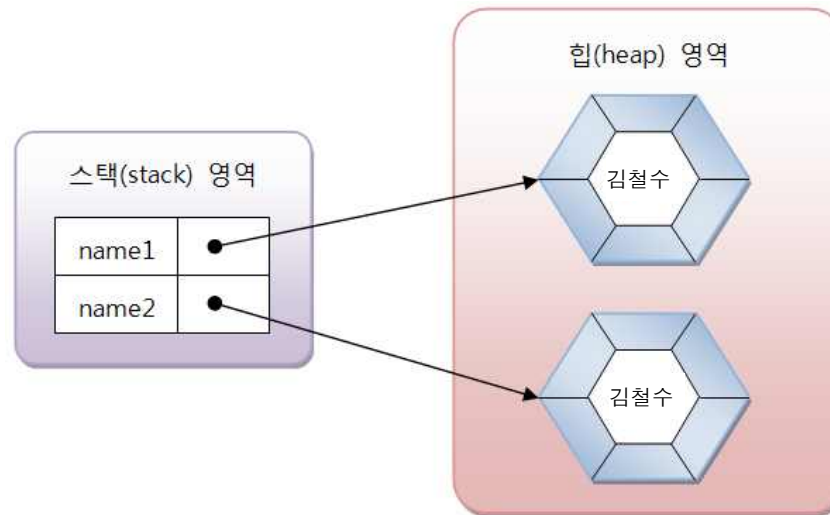
```
String name1 = new String("김철수");  
String name2 = new String("김철수");
```



예제

```
public class StringEqualsExample {  
    public static void main(String[] args) {  
        String strVar1 = "김철수";  
        String strVar2 = "김철수";  
  
        if(strVar1 == strVar2) {  
            System.out.println("strVar1 과 strVar2는 참조가 같음");  
        } else {  
            System.out.println("strVar1 과 strVar2는 참조가 다름");  
        }  
  
        if(strVar1.equals(strVar2)) {  
            System.out.println("strVar1 과 strVar2는 문자열이 같음");  
        }  
  
        String strVar3 = new String("김철수");  
        String strVar4 = new String("김철수");  
  
        if(strVar3 == strVar4) {  
            System.out.println("strVar3과 strVar4는 참조가 같음");  
        } else {  
            System.out.println("strVar3과 strVar4는 참조가 다름");  
        }  
  
        if(strVar3.equals(strVar4)) {  
            System.out.println("strVar3과 strVar4는 문자열이 같음");  
        }  
    }  
}
```


- 그림과 같은 형태로 두개의 변수를 생성다음을 비교하시오
 - 두개가 동일한 객체인지 여부
 - 두개의 문자열이 같은지 여부



4절. 객체 생성과 클래스 변수

■ new 연산자

■ 객체 생성 역할

```
new 클래스();
```



- 클래스()의 객체를 생성
- 생성자를 호출하는 코드
- 생성된 객체는 힙 메모리 영역에 생성

■ new 연산자는 객체를 생성 후, 객체 생성 번지 리턴

4절. 객체 생성과 클래스 변수

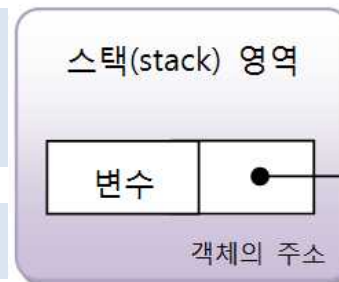
■ 클래스 변수

- new 연산자에 의해 리턴 된 객체의 번지 저장 (참조 타입 변수)
- 힙 영역의 객체를 사용하기 위해 사용

클래스 변수;

변수 = new 클래스();

클래스 변수 = new 클래스();



참조



- SmartPhone0이라는 클래스를 선언
- Main라는 클래스의 main 메소드에서 sp1, sp2 라는 Smartphone 객체를 생성하시오

5절. 클래스 선언

- 클래스의 구성 멤버
 - 필드(Field)
 - 생성자(Constructor)
 - 메소드(Method)

- 필드(Field) ————— 객체의 데이터가 저장되는 곳
- 생성자(Constructor) ————— 객체 생성시 초기화 역할 담당
- 메소드(Method) ————— 객체의 동작에 해당하는 실행 블록

접근지정자 클래스이름

```
public class ClassName {  
  
    //필드  
    int fieldName;  
  
    //생성자  
    ClassName() { ... }  
  
    //메소드  
    void methodName() { ... }  
  
}
```

6절. 필드(field)

- 클래스내에 선언된 변수
- 필드 선언

타입 필드 [= 초기값] ;

```
String company = "현대자동차";  
String model = "그랜저";  
int maxSpeed = 300;  
int productionYear;  
int currentSpeed;  
boolean engineStart;
```

6절. 필드(field)

- 필드의 기본 초기값
 - 초기값 지정되지 않은 필드
 - 객체 생성시 자동으로 기본값으로 초기화

분류		데이터 타입	초기값
기본 타입	정수 타입	byte	0
		char	₩u0000 (빈 공백)
		short	0
		int	0
		long	0L
	실수 타입	float	0.0F
		double	0.0
	논리 타입	boolean	false
참조 타입		배열	null
		클래스(String 포함)	null
		인터페이스	null

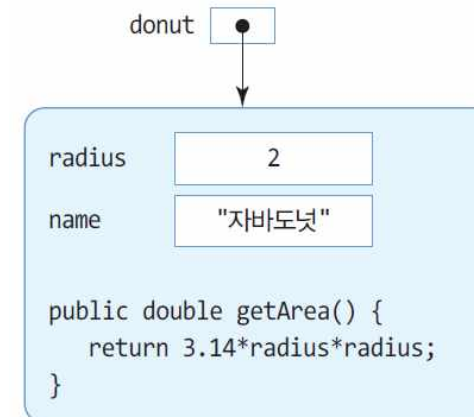
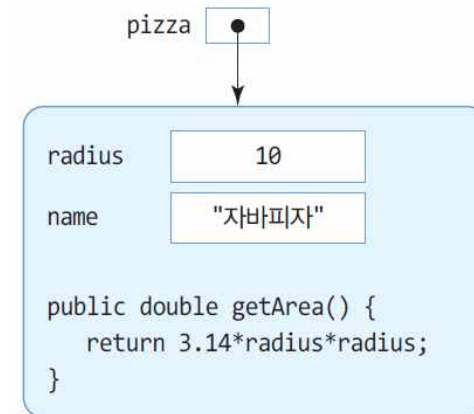
6절. 필드(field)

- 필드 사용
 - 필드 값을 읽고, 변경하는 작업을 말한다.
 - 필드 사용 위치
 - 선언된 클래스 내부: “**필드이름**” 으로 바로 접근
 - 선언된 클래스 외부: “**객체.필드이름**” 으로 접근

6절. 필드(field)

```
public class Circle {  
    int radius;           // 원의 반지름을 저장하는 멤버 변수  
    String name;          // 원의 이름을 저장하는 멤버 변수  
  
    public double getArea() { // 멤버 메소드  
        return 3.14*radius*radius;  
    }  
}
```

```
public class CircleExample {  
    public static void main(String[] args) {  
        Circle pizza;  
        pizza = new Circle();           // Circle 객체 생성  
        pizza.radius = 10;              // 피자 반지름을 10으로 설정  
        pizza.name = "자바피자";        // 피자의 이름 설정  
        double area = pizza.getArea();   // 피자의 면적 알아내기  
        System.out.println(pizza.name + "의 면적은 " + area);  
  
        Circle donut = new Circle();     // Circle 객체 생성  
        donut.radius = 2;                // 도넛의 반지름을 2로 설정  
        donut.name = "자바도넛";          // 도넛의 이름 설정  
        area = donut.getArea();           // 도넛의 면적 알아내기  
        System.out.println(donut.name + "의 면적은 " + area);  
    }  
}
```



객체 생성과 활용

1. 레퍼런스 변수 선언

`Circle pizza;`

2. 객체 생성

- new 연산자 이용

`pizza = new Circle();`


3. 객체 멤버 접근

- 점(.) 연산자 이용

`pizza.radius = 10;`


`area = pizza.getArea();`

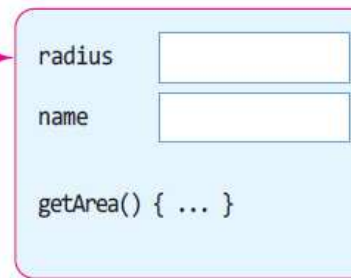
(1) `Circle pizza;`

pizza 

Circle 타입의 객체


(2) `pizza = new Circle();`

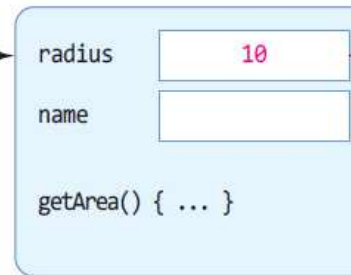
pizza 



객체 메모리
할당 및
객체 생성


(3) `pizza.radius = 10;`

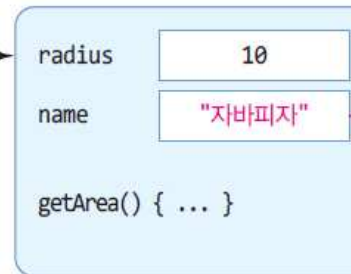
pizza 



radius 값 변경


(4) `pizza.name = "자바피자";`

pizza 

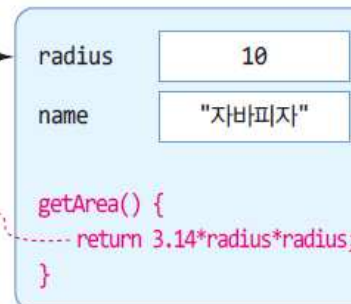


name 값 변경

(5) `double area = pizza.getArea();`

pizza 

area



getArea()
메소드 실행

- Main 클래스에서 sp1, sp2객체 생성
- 다음과 SmartPhone 클래스와 필드를 만들고 객체 생성 후 초기값과 변경된 값을 출력하세요

sp1

필드명	disSize	color	bandSpeed	company
타입	int	String	int	String
초기값	60	FHD	1000	LL
값변경		HD	500	

sp2

필드명	disSize	color	bandSpeed	company
타입	int	String	int	String
값변경	80	QHD	2000	SS

- ❖ 클래스명 : Tree
- ❖ Main에서 객체 생성후 다음을 실행
 - ❖ obj1 객체 생성

필드명	height	color	name
타입	int	String	String
초기값	60	yellow	pinetree
값변경	10	red	oaktree

- ❖ 필드값 출력

- ❖ obj2 객체 생성

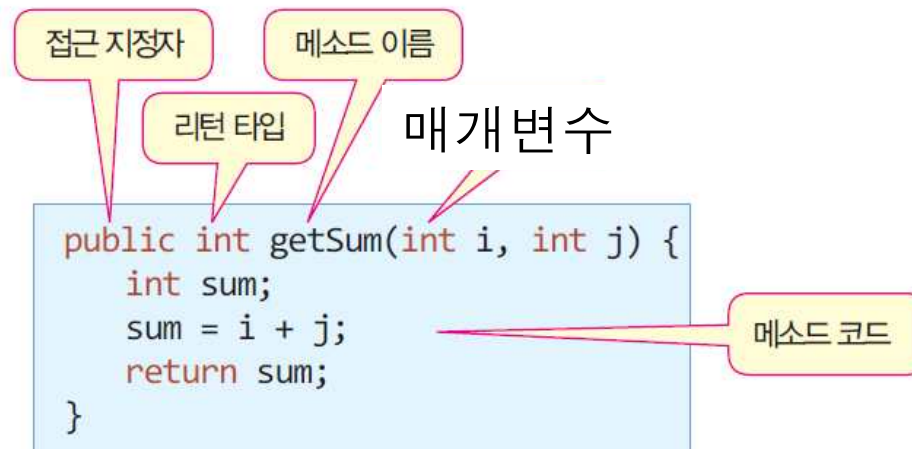
필드명	height	color	name
타입	int	String	String
값변경	100	green	lemontree

- ❖ Obj1.color : green 로 변경을 하는데. Obj2의 객체를 이용

7절. 메소드(method)

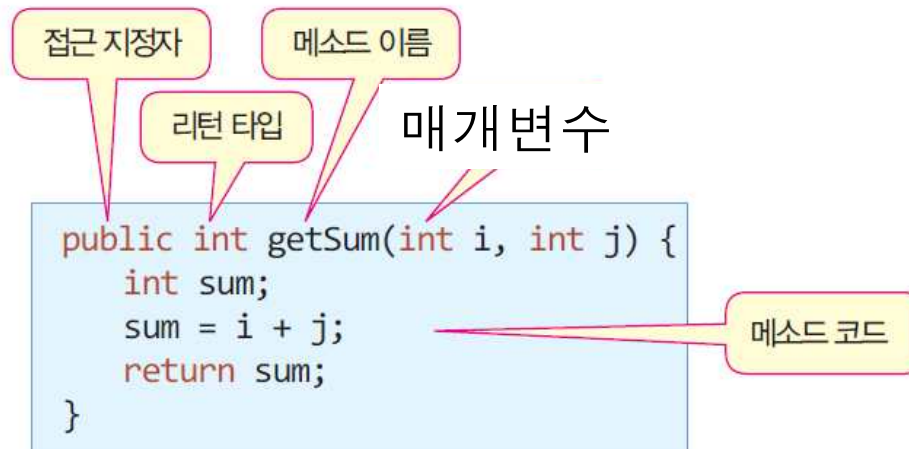
- 메소드란?
 - 객체의 동작(기능)
 - 호출해서 실행할 수 있는 중괄호 { } 블록
 - 메소드 호출하면 중괄호 { } 블록에 있는 모든 코드들이 일괄 실행

- 메소드 선언



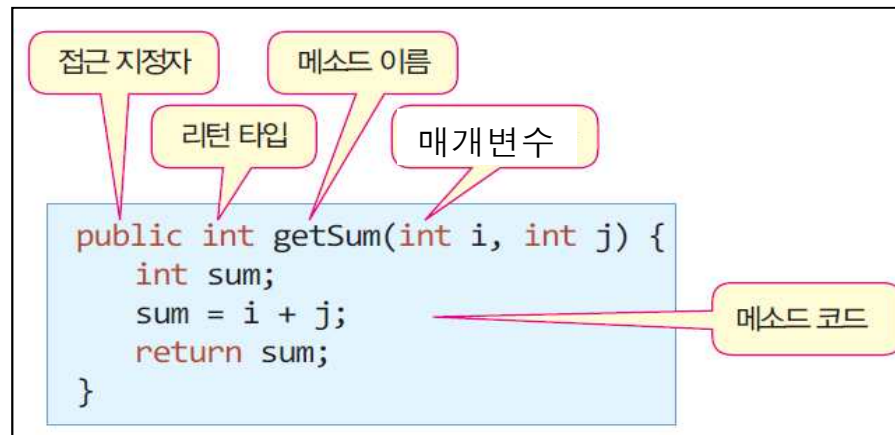
- 접근 지정자
 - 다른 클래스에서 메소드를 접근할 수 있는지 여부 선언
 - `public`, `private`, `protected`, 디폴트(접근 지정자 생략)

7절. 메소드(method)



7절. 메소드(method)

- 메소드 리턴 타입
 - 메소드 실행된 후 리턴하는 값의 타입
 - 메소드는 리턴값이 있을 수도 있고 없을 수도 있음



[메소드 선언]

```
void powerOn() { ... }  
double divide(int x, int y) { ... }
```

[메소드 호출]

```
powerOn();  
double result = divide( 10, 20 );
```

- 메소드 이름
 - 자바 식별자 규칙에 맞게 작성

7절. 메소드(method)

- 메소드 매개변수 선언
 - 매개변수는 메소드를 실행할 때 필요한 데이터를 외부에서 받기 위해 사용
 - 매개변수도 필요 없을 수 있음

[메소드 선언]

```
void powerOn() { ... }  
double divide(int x, int y) { ... }
```

[메소드 호출]

```
powerOn();  
double result = divide( 10, 20 );
```

```
byte b1 = 10;  
byte b2 = 20;  
double result = divide(b1, b2);
```


7절. 메소드(method)

- 리턴(return) 문
 - 메소드 실행을 중지하고 리턴값 지정하는 역할
 - 리턴값이 있는 메소드
 - 반드시 리턴(return)문 사용해 리턴값 지정해야

```
int plus(int x, int y) {  
    int result = x + y;  
    return result;  
}
```

- return 문 뒤에 실행문 올 수 없음
- 리턴값이 없는 메소드
 - 메소드 실행을 강제 종료 시키는 역할

```
boolean isLeftGas() {  
    if(gas==0) {  
        System.out.println("gas 가 없습니다.");  
        return false;  
    }  
    System.out.println("gas 가 있습니다.");  
    return true;  
}
```

7절. 메소드(method)

리턴타입이 있다면 세개의
타입이 모두 일치 해야 함

```
class Calculator {  
    int radius;  
    String name;  
    int getSum(int x, int y)  
    {  
        int sum = x+y;  
        printSum(sum);  
        return sum;  
    }  
    void printSum(int s)  
    {  
        System.out.println(s);  
    }  
}  
public class MainClass  
{  
    public static void main(String[] args) {  
        int ret=0, x=1, y=2;  
        Calculator cal = new Calculator ();  
        ret=cal.getSum(x,y);  
    }  
}
```

매개변수가 있다면 호출시
매개변수의 개수 타입이
모두 일치 해야 함

7절. 메소드(method)

■ 메소드 호출 방법

■ 같은 클래스 내에서 호출

- 리턴값이 있는 경우
- 형식 : 변수 = 메소드명(매개변수)
- 리턴값이 없는 경우
- 형식 : 메소드명(매개변수);

```
class Calculator {  
    int radius;  
    String name;  
    int getSum(int x, int y)  
    {  
        int sum = x+y;  
        printSum(sum);  
        return sum;  
    }  
    void printSum(int s)  
    {  
        System.out.println(s);  
    }  
}  
public class MainClass  
{  
    public static void main(String[] args) {  
        int x=1, y=2, ret=0;  
        Calculator cal = new Calculator ();  
        ret=cal.getSum(x,y);  
    }  
}
```

■ 다른 클래스 에서 호출

- 1. 객체 생성
- 2. 메소드 호출
- 리턴값이 있는 경우
- 형식 : 변수 = 객체.메소드명(매개변수);
- 리턴값이 없는 경우
- 형식 : 객체.메소드명(매개변수);

■ 클래스 이름 : BasicCalculator

리턴타입	메소드명	매개변수	호출시매개변수값
int	plus	int x, int y	1, 2
int	minus	int x, int y	4,2
int	mul	int x, int y	3,3
double	divide	int x, int y	10,4

■ Main 클래스 생성 – main()

- 객체를 이용 각 메소드 호출(메소드에는 디버그 메시지 추가)
- 리턴값 저장 후 출력

리턴타입 변수명 = 객체. 메소드(매개변수들)

8절. 생성자(Constructor)

■ 생성자

- new 연산자에 의해 호출되어 객체의 초기화 담당(호출시기)

```
Car c = new Car();
```

- 생성자 이름은 클래스 이름과 동일
- 생성자는 여러 개 작성 가능(생성자 중복)
- 목적 : 객체 생성 시 필드의 초기화 및 초기화 관련 메소드 호출

■ 기본 생성자(default constructor)

- 매개 변수 없고, 아무 작업 없이 단순 리턴하는 생성자
- 디폴트 생성자라고도 불림
- 생성자 선언을 생략하면 컴파일러는 다음과 같은 기본 생성자 추가

소스 파일(Car.java)

```
public class Car {  
  
}
```

→

바이트 코드 파일(Car.class)

```
public class Car {  
    public Car() { } //자동 추가  
}
```

기본 생성자

```
Car myCar = new Car();
```

기본 생성자

8절. 생성자(Constructor)

```
public class Car {  
    //생성자  
    Car() {  
    }  
}
```

```
public class CarExample {  
    public static void main(String[] args) {  
        Car myCar = new Car();  
    }  
}
```

8절. 생성자(Constructor)

- 생성자 선언 : 메소드와의 차이점 -> 리턴값이 없음
 - 디폴트 생성자 대신 개발자가 직접 선언
 - 개발자 선언한 생성자 존재 시 컴파일러는 기본 생성자 추가하지 않음
 - 생성자 선언 형식

```
클래스명(매개변수들){  
}
```

- 메소드 선언 형식

```
리턴타입 메소드명(매개변수들){  
}
```

예제

```
public class Car {  
    //생성자  
    Car() {  
    }  
}
```



```
public class CarExample {  
    public static void main(String[] args) {  
        Car myCar = new Car();  
        //Car myCar = new Car(); (x)  
    }  
}
```

```
public class Car {  
    String color = "red";  
    int cc = 1100;  
  
    public Car()  
    {  
  
    }  
  
    public Car(String col, int c)  
    {  
        color = col;  
        cc=c;  
    }  
}
```



```
public class CarExample {  
    public static void main(String[] args)  
    {  
        Car cno = new Car();  
        System.out.println(cno.color);  
        System.out.println(cno.cc);  
        Car c = new Car("black", 2000);  
        System.out.println(c.color);  
        System.out.println(c.cc);  
    }  
}
```


9절. 인스턴스 멤버

■ this

- 객체 내부에서 인스턴스 멤버임을 명확히 하기 위해 this. 사용
- 필드
 - 객체내부에서 인스턴스 필드에 접근시 사용(객체 생성 없이 this를 이용 바로 사용 가능)

```
public class Car {  
    String color="red";  
    int cc=1100;  
    public Car(String color, int cc) {  
        this.color=color;  
        this.cc=cc;  
    }  
}
```

■ 메소드

- 객체내부에서 인스턴스 메소드에 접근시 사용(객체 생성 없이 this를 이용 바로 사용 가능)

예제

```
public class Car {  
    String color="red";  
    int cc=1100;  
    public Car() {  
  
    }  
    public Car(String color, int cc) {  
        this.color=color;  
        this.cc=cc;  
    }  
    public String getColor() {  
        return this.color;  
    }  
}
```

```
public class CarExample {  
    public static void main(String[] args)  
    {  
  
        Car c = new Car("black",2000);  
        System.out.println(c.color);  
        System.out.println(c.cc);  
    }  
}
```

10절. 패키지(package)

■ 패키지란?

- 클래스를 기능별로 묶어서 그룹 이름을 붙여 놓은 것
 - 파일들을 관리하기 위해 사용하는 폴더(디렉토리)와 비슷한 개념
 - 패키지의 물리적인 형태는 파일 시스템의 폴더
- 클래스명이 같아도 패키지명이 다르면 다른 클래스로 취급
- 클래스 선언할 때 포함될 패키지 선언

예제

```
package ch06.exam05_package;
import ch06.exam05_package.hy.Car;

public class CarExample {
    public static void main(String[] args) {
        Car car1 = new Car();
        System.out.println("car1.company : " + car1.company);
        System.out.println();
    }
}
```

```
package ch06.exam05_package.hy;
```

```
public class Car {
    //필드
    public String company = "h_motor";
    public String model;
    public String color;

    //생성자
    public Car() {
    }
    public Car(String model, String color) {
        this.model = model;
        this.color = color;
    }

    public String getCompany() {
        return this.company;
    }
}
```

```
package ch06.exam05_package.ki;
```

```
public class Car {
    //필드
    public String company = "k_motor";
    public String model;
    public String color;

    //생성자
    public Car() {
    }
    public Car(String model, String color) {
        this.model = model;
        this.color = color;
    }

    public String getCompany() {
        return this.company;
    }
}
```