6. 클래스(Class)

클래스와 객체

- **클래스**는 객체(Object)를 정의해 놓은 코드

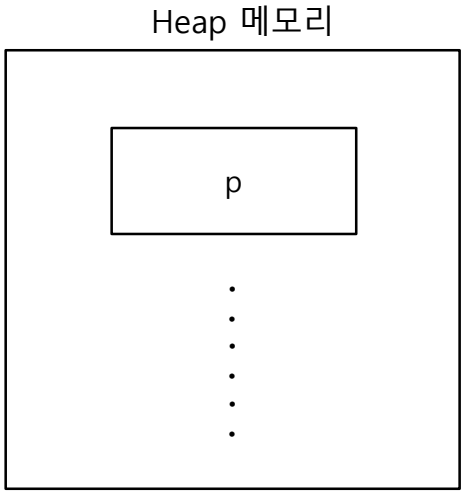
  - 코드가 실행되어 클래스가 메모리에 로드되는 것을 **인스턴스(instance)화** 되었다고 함

  - 해당 인스턴스를 **객체**라고 칭함

Person.java

```
class Person{
    String name;
    int age;
    char gender;

    void walk(){
        …
    }
    void talk(){
        …
    }
}

class MainClass{
    public static void main(…){
        Person p = new Person();
        …
    }
}
```

인스턴스화



클래스 구조

- **클래스**는 객체(Object)를 정의해 놓은 코드

  - 코드가 실행되어 클래스가 메모리에 로드되는 것을 **인스턴스(instance)화** 되었다고 함

  - 해당 인스턴스를 **객체**라고 칭함

```
class Person{
    String name;
    int age;
    char gender;

    Person(){
        ...
    }

    void walk(){
        ...
    }
    void talk(){
        ...
    }
    void eat(){
    }

}
```

→ 멤버변수(Field)
객체가 가지고 있는 속성을 변수로 선언한 것

→ 생성자(Constructor)
인스턴스 생성 시, 실행되는 초기화 블럭

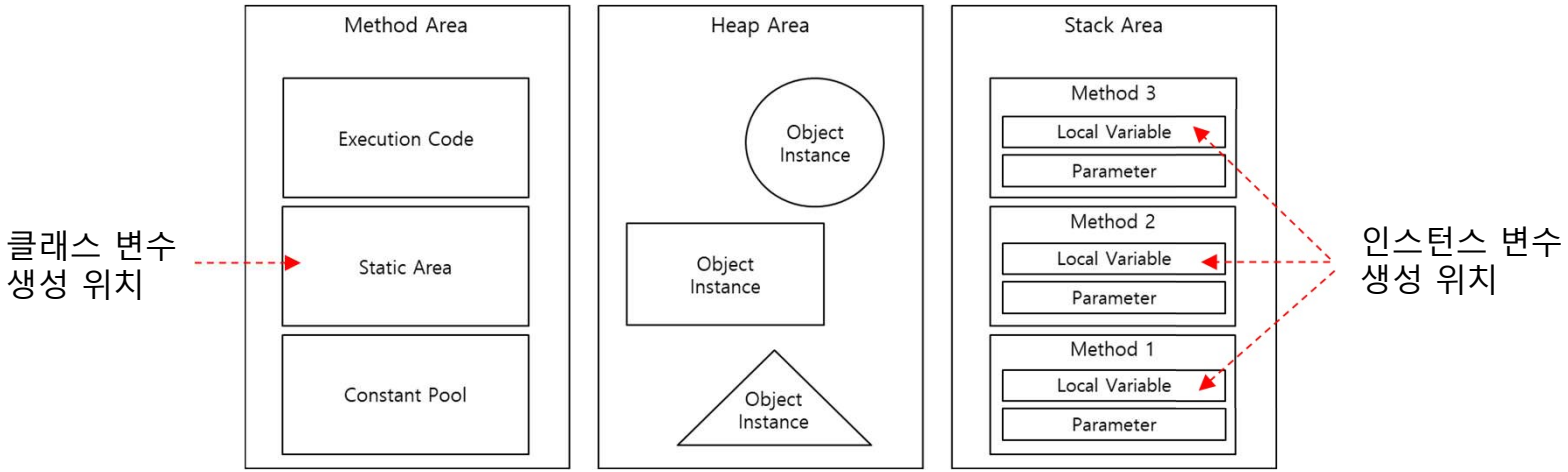
→ 메소드(Method)
객체가 수행하는 특정 작업에 대한 명령문을 모아놓은 그룹으로서 기존 함수와 비슷

멤버 변수

- 클래스 내부에 선언된 변수로서 클래스 변수와 인스턴스 변수로 나뉨

  - 인스턴스 변수
    - 클래스의 인스턴스가 생성될 때 만들어지는 변수로서 인스턴스마다 독립적인 메모리 공간에 생성되는 변수
  - 클래스 변수(앞에 static을 붙여서 선언 – '정적 변수'라고도 함)
    - 해당 클래스의 모든 인스턴스가 공유하는 변수로서 인스턴스(객체)를 생성하지 않고도 사용할 수 있음

```
class Man{
    String name;  // 인스턴스 변수
    int age;
    static char gender = 'M';  // 클래스 변수 (정적 변수)
}
```



실습

[Ex1] CardTest.java

```
package chap06.classex;

class CardTest{
    public static void main(String args[]) {
        System.out.println("Card.width = "  + Card.width);
        System.out.println("Card.height = " + Card.height);

        Card c1 = new Card();
        c1.kind = "Heart";
        c1.number = 7;

        Card c2 = new Card();
        c2.kind = "Spade";
        c2.number = 4;

        System.out.println("c1은 " + c1.kind + ", " + c1.number + "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
        System.out.println("c2는 " + c2.kind + ", " + c2.number + "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );

        System.out.println("c1의 width와 height를 각각 50, 80으로 변경합니다.");
        c1.width = 50;
        c1.height = 80;

        System.out.println("c1은 " + c1.kind + ", " + c1.number + "이며, 크기는 (" + c1.width + ", " + c1.height + ")" );
        System.out.println("c2는 " + c2.kind + ", " + c2.number + "이며, 크기는 (" + c2.width + ", " + c2.height + ")" );
    }
 }

class Card {
    String kind ;               // 인스턴스 변수
    int number;                 // 인스턴스 변수
    static int width = 100;     // 클래스 변수
    static int height = 250;    // 클래스 변수
}
```

final static

- 공용상수로서 모든 객체(인스턴스)가 공유하는 상수(final 만 붙이면 객체간 공유 안 됨)

[Ex2] FinalStaticEx.java

```
package chap06.classex;

public class FinalStaticEx {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        FinalStatic fs = new FinalStatic(246);

        System.out.println("A : " + fs.A);
        System.out.println("B : " + fs.B);
        System.out.println("C : " + fs.C);
        System.out.println("D : " + fs.D);
        System.out.println("E : " + fs.E);
    }
}

class FinalStatic{
    final int A = 123;
    final int B;
    final int C;
    final int D;
    final static int E = 135;
    //final static int F; // 에러. final static으로 선언한 상수는 선언과 함께 초기화 해야 함

    FinalStatic(int d){
        B = 456;
        D = d;
    //  F = 246;
    }
    {
        C = 789;
    }
}
```

생성자

- 인스턴스가 생성될 때 자동으로 호출되는 초기화 블럭

- 클래스의 이름과 같아야 하며, 메소드처럼 매개변수를 갖지만, 리턴 값은 없음

```
class Man{
    String name;
    int age;
    static char gender = 'M';

    Man(char n, int a){    // 생성자
        name = n;
        age = a;
    }
}
```

- 모든 클래스는 하나 이상의 생성자를 가져야 하며, 생성자를 생략할 경우, 자동으로 기본 생성자가 만들어짐

  - 생성자가 하나라도 정의되었을 경우, 기본 생성자는 자동 추가되지 않음

```
class Man{
    char name;
    int age;
    static char gender = "Male";

    // 생성자 생략
}
```

컴파일 →

```
class Man{
    char name;
    int age;
    static char gender = "Male";

    Man(){ }  // 자동으로 추가된 기본생성자
}
```

생성자 오버로딩(Overloading)

- 여러 개의 생성자를 정의할 수 있으며, 이 때 각각의 생성자는 매개변수가 달라야 함

  - 생성자의 매개변수 개수와 타입의 차이를 통해 생성자를 구분

  - 생성자 안에서 다른 생성자를 호출할 때 this()를 이용

  - 생성자의 매개변수명과 인스턴스 변수명이 같을 경우, 인스턴스 변수명에 **this**를 붙여 구분

```
class Man{
    String name;
    int age;
    static char gender = 'M';

    Man(){  // 생성자 1
        this("noname", 25);
    }

    Man(String name){  // 생성자 2
        this(n, 25);
    }

    Man(String name, int age){   // 생성자 3
        this.name = n;
        this.age = a;
    }
}
```

실습(1)

[Ex3-1] Car.java

```
package chap06.classex;

public class Car {
    //필드
    String company = "현대자동차";
    String model;
    String color;
    int maxSpeed;

    //생성자
    Car() {
    }

    Car(String model) {
        this(model, null, 0);
    }

    Car(String model, String color) {
        this(model, color, 0);
    }

    Car(String model, String color, int maxSpeed) {
        this.model = model;
        this.color = color;
        this.maxSpeed = maxSpeed;
    }
}
```

실습(2)

[Ex3-2] CarExample.java

```
package chap06.classex;

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

        Car car2 = new Car("자가용");
        System.out.println("car2.company : " + car2.company);
        System.out.println("car2.model : " + car2.model);
        System.out.println();

        Car car3 = new Car("자가용", "빨강");
        System.out.println("car3.company : " + car3.company);
        System.out.println("car3.model : " + car3.model);
        System.out.println("car3.color : " + car3.color);
        System.out.println();

        Car car4 = new Car("택시", "검정", 200);
        System.out.println("car4.company : " + car4.company);
        System.out.println("car4.model : " + car4.model);
        System.out.println("car4.color : " + car4.color);
        System.out.println("car4.maxSpeed : " + car4.maxSpeed);
    }
}
```

실습(3)

[Ex4-1] Car2.java

```
package chap06.classex;

public class Car2 {
    String color;
    String gearType;
    int door;

    Car2() {
        this("white", "auto", 4);
    }

    Car2(Car2 c) {  // Car2의 인스턴스를 매개변수로 받음
        color    = c.color;
        gearType = c.gearType;
        door     = c.door;
    }

    Car2(String color, String gearType, int door) {
        this.color    = color;
        this.gearType = gearType;
        this.door     = door;
    }
}
```

실습(4)

[Ex4-2] CarExample2.java

```
package chap06.classex;

public class CarExample2 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Car2 c1 = new Car2();
        Car2 c2 = new Car2(c1); // c1의 복사본 c2를 생성한다.
        System.out.println("c1의 color=" + c1.color + ", gearType=" + c1.gearType+ ", door="+c1.door);
        System.out.println("c2의 color=" + c2.color + ", gearType=" + c2.gearType+ ", door="+c2.door);

        c1.door=100;    // c1의 인스턴스변수 door의 값을 변경한다.
        System.out.println("c1.door=100; 수행 후");
        System.out.println("c1의 color=" + c1.color + ", gearType=" + c1.gearType+ ", door="+c1.door);
        System.out.println("c2의 color=" + c2.color + ", gearType=" + c2.gearType+ ", door="+c2.door);
    }
}
```

초기화 블럭

- 클래스 변수와 인스턴스 변수를 초기화하기 위한 블럭

  - 클래스 변수 초기화 블록은 클래스가 메모리에 처음 로딩될 때 한번만 수행됨

  - 인스턴스 변수 초기화 블록은 인스턴스가 생성될 때마다 수행됨

```
class Man{
    String name;
    int age;
    static char gender;

    // 인스턴스 변수 초기화 블럭
    {
        name = "Kim";
        age = "25";
    }

    // 클래스 변수 초기화 블럭
    static {
        gender = 'M';
    }
}
```

  - 변수 초기화 방법

    - 명시적 초기화 (변수 선언 시, 초기화 하는 방법)
    - 생성자를 이용한 초기화
    - 초기화 블록을 이용한 초기화

실습(1)

[Ex5] InitBlockTest.java

```
package chap06.classex;

public class InitBlockTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        InitBlock ib = new InitBlock();
        System.out.println("iVar : " + ib.iVar);
        System.out.println("sVar : " + ib.sVar);
        InitBlock ib2 = new InitBlock();
        System.out.println("iVar : " + ib2.iVar);
        System.out.println("sVar : " + ib2.sVar);
    }
}

class InitBlock{
    static int sVar = 1;
    int iVar = 10;

    static {
        sVar = 5;
        System.out.println("static 초기화 블럭이 실행되었습니다.");
    }
    {
        iVar = 100;
        System.out.println("instance 초기화 블럭이 실행되었습니다.");
    }
}
```

실습(2)

[Ex6] CustomerTest.java

```
package chap06.classex;

class Customer {
    static int cCount = 0;
    int myNum;
    {
        cCount++;
        myNum = cCount;
    }
}

public class CustomerTest {
    public static void main(String args[]) {
        Customer c1 = new Customer();
        Customer c2 = new Customer();
        Customer c3 = new Customer();

        System.out.println(c1.myNum + "번 고객이 방문했습니다.");
        System.out.println(c2.myNum + "번 고객이 방문했습니다.");
        System.out.println(c3.myNum + "번 고객이 방문했습니다.");
        System.out.println("지금까지 방문한 고객수는 "+ c3.cCount + "명 입니다.");
    }
}
```

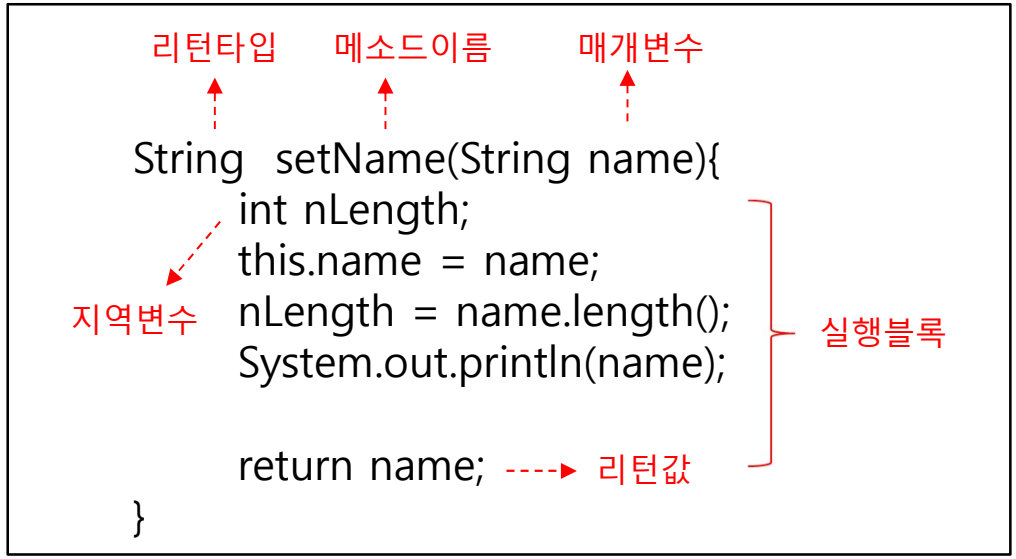
메소드

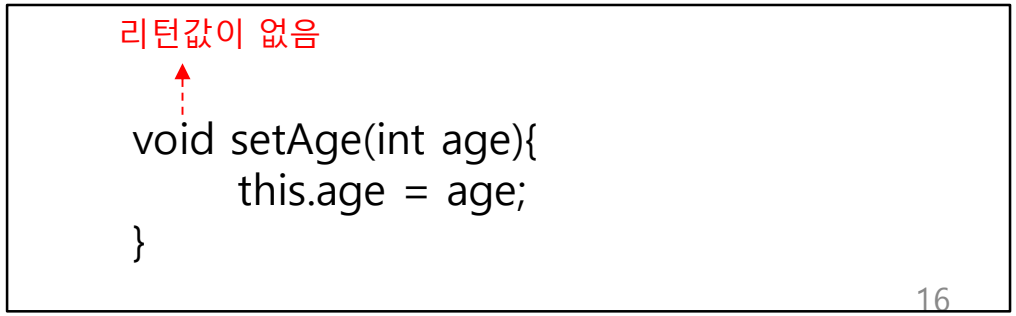
- 객체가 수행하는 특정 작업에 대한 명령문을 모아놓은 블럭으로서 코드의 중복을 제거하고 재사용성을 높이기 위한 목적

  - 메소드는 리턴타입, 메소드이름, 매개변수, 실행블록으로 구성됨

    - 메소드의 리턴값이 없을 경우에는 메소드의 리턴타입은 void로 정의

```
class Person{

    String name;
    int age;
    char gender;

    Person(){
        ...
    }

    String setName(String name){
        int nLength;
        this.name = name;
        nLength = name.length()
        System.out.println(name);

        return name;
    }

    void setAge(int age){
        this.age = age;
    }

}
```





실습(1)

[Ex7-1] Calculator.java

```
package chap06.classex;

public class Calculator {
    boolean power = false;

    void powerOn() {
        if(!power) {
            power = true;
            System.out.println("계산기 전원을 켭니다.");
        }else {
            System.out.println("계산기 전원이 이미 켜져있습니다.");
        }
    }
    void powerOff() {
        if(power) {
            power = false;
            System.out.println("계산기 전원을 끕니다.");
        }
    }
    void showStatus() {
        if(power)
            System.out.println("계산기 전원이 켜져 있습니다.");
        else
            System.out.println("계산기 전원이 꺼져 있습니다.");
    }
    double add(double a, double b) { return a + b; }
    double subtract(double a, double b) { return a - b; }
    double multiply(double a, double b) { return a * b; }
    double divide(double a, double b) { return a / b; }
}
```

실습(2)

[Ex7-2] CalculatorTest.java

```
package chap06.classex;

public class CalculatorTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Calculator cal = new Calculator();
        cal.powerOn();
        cal.showStatus();

        System.out.println("30 + 10 = " + cal.add(30, 10));
        System.out.println("30 - 10 = " + cal.subtract(30, 10));
        System.out.println("30 * 10 = " + cal.multiply(30, 10));
        System.out.println("30 / 10 = " + cal.divide(30, 10));

        cal.powerOff();
        cal.showStatus();
    }
}
```

실습(3)

[Ex8] AverageEx.java

```
package chap06.classex;

public class AverageEx {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Average a = new Average();
        double result = a.avg(10, 50);
        System.out.println(result);
    }
}

class Average{
    int plus(int x, int y) {
        return x + y;
    }
    double avg(int x, int y) {
        double sum = plus(x, y);  // 클래스 내부 메소드 호출
        return sum / 2;
    }
}
```

메소드 매개변수

- mutable 속성 매개변수 vs immutable 속성 매개변수

  - Immutable 속성을 가진 매개변수는 메소드에서 값을 변경해도 원본에 영향을 미치지 않음

```
class Man{
    void increaseAge(int age){
        age = age + 1;
    }
}
```

호출

```
class MainClass {
    int age = 10;
    Man m = new Man();
    m.increaseAge(age)
    System.out.println("age = " + age); // 10
}
```

  - MainClass의 age의 값을 복사해서 넘기는 것이므로 increaseAge 메소드에서 age를 증가시켜도 전혀 영향을 받지 않음
  - 기본형, String, Integer, Double 등의 기본형 wrapper 클래스, LocalDate, LocalTime, LocalDateTime, Enum, BigInteger, BigDecimal 등의 수치가 큰 데이터를 다루는 클래스 등

  - Mutable 속성을 가진 매개변수는 메소드에서 값을 변경하면 원본에 영향을 미침

```
class Man{
    void changeName(char[] name){
        name[0] = 'K';
    }
}
```

호출

```
class MainClass {
    char[] name = "Lee";
    Man m = new Man();
    m.changeName(name)
    System.out.println("name = " + name); // Kee
}
```

  - MainClass의 name은 배열변수로서 changeName 메소드로 "Lee" 값의 주소를 넘기며, changeName 메소드에서는 그 주소를 직접 액세스하여 "L"을 "K"로 변경 가능
  - 배열, 사용자 정의 클래스, ArrayList, LinkedList, StringBuilder, StringBuffer, HashMap, Hashtable 등

실습(1)

[Ex9] ManTest.java

```
package chap06.classex;

public class ManTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        char[] name = new char[] {'L', 'e', 'e'};
        Man m = new Man();

        m.changeName(name);
        System.out.println(name);
    }
}

class Man{
    void changeName(char[] n) {
        n[0] = 'K';
    }
}
```

실습(2)

[Ex10] ReferenceParamEx.java

```
package chap06.classex;

public class ReferenceParamEx {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Data d = new Data();
        d.x = 10;
        System.out.println("main() : x = " + d.x);

        changeMutable(d.x);
        System.out.println("After change(d.x)");
        System.out.println("main() : x = " + d.x);

        changeImmutable(d);
        System.out.println("After change(d.x)");
        System.out.println("main() : x = " + d.x);
    }

    static void changeMutable(int x) {
        System.out.println("====  Mutable  ====");
        x = 1000;
        System.out.println("change() : x = " + x);
    }
    static void changeImmutable(Data d) {
        System.out.println("====  Immutable  ====");
        d.x = 1000;
        System.out.println("change() : x = " + d.x);
    }
}

class Data { int x; }
```

클래스 메소드

- 앞에 static을 붙여 선언한 메소드로서 클래스 변수와 같이 인스턴스(객체)를 생성하지 않고도 호출이 가능('정적 메소드'라고도 함)

```
class MyClass{
    static int sVal; // 클래스 변수
    int iVal; // 인스턴스 변수

    static void myMethod(int a){  // 클래스 메소드 (정적 메소드)
        .....
    }

    int myMethod2(int a){  // 인스턴스 메소드
        .....
    }
```

- public static void main(String[] args) 도 클래스 메소드
  - 클래스 인스턴스를 생성하지 않아도 main 메소드가 자동 호출됨

- System.out.println(), Integer.parseInt(), Math.random() 등도 모두 클래스 메소드

- 클래스 메소드에서는 인스턴스 변수를 사용할 수 없음

- 클래스 메소드에서는 인스턴스 메소드를 호출할 수 없음

실습(1)

[Ex11-1] MyMath.java

```
package chap06.classex;

public class MyMath {
    long a, b;

    // 인스턴스 메소드
    long add()      { return a + b; }
    long subtract() { return a - b; }
    long multiply() { return a * b; }
    double divide() { return a / b; }

    // 클래스 메소드
    //static long add_instanceVar() { return a + b; } // 에러. a와 b가 인스턴스 변수 이므로
    static long add(long a, long b)          { return a + b; } // a, b는 지역변수
    static long subtract(long a, long b)   { return a - b; }
    static long multiply(long a, long b)     { return a * b; }
    static double divide(double a, double b) { return a / b; }
}
```

실습(2)

[Ex11-2] MyMathTest.java

```
package chap06.classex;

public class MyMathTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // 클래스메서드 호출. 인스턴스 생성없이 호출가능
        System.out.println("==== 클래스 메소드 호출 ====");
        System.out.println(MyMath.add(200L, 100L));
        System.out.println(MyMath.subtract(200L, 100L));
        System.out.println(MyMath.multiply(200L, 100L));
        System.out.println(MyMath.divide(200.0, 100.0));

        // 인스턴스메서드는 객체생성 후에만 호출이 가능함.
        System.out.println("==== 인스턴스 메소드 호출 ====");
        MyMath mm = new MyMath(); // 인스턴스를 생성
        mm.a = 200L;
        mm.b = 100L;
        System.out.println(mm.add());
        System.out.println(mm.subtract());
        System.out.println(mm.multiply());
        System.out.println(mm.divide());
    }
}
```

실습(3)

[Ex12] SingletonEx.java

```
package chap06.classex;

public class SingletonEx {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Singleton sInstance = Singleton.getInstance();
        sInstance.increaseInstanceVal();
        System.out.println("instanceVal : " + sInstance.instanceVal);

        Singleton sInstance2 = Singleton.getInstance();
        sInstance2.increaseInstanceVal();
        System.out.println("instanceVal : " + sInstance2.instanceVal);

        if(sInstance == sInstance2)
            System.out.println("sInstance와 sInstance2는 같은 객체입니다.");
        else
            System.out.println("sInstance와 sInstance2는 다른 객체입니다.");
    }
}

class Singleton{
    private static Singleton instance;
    int instanceVal = 0;

    // 생성자를 private으로 선언하여 인스턴스를 만들지 못하도록 막음
    private Singleton() {}

    public static Singleton getInstance() {
        if(instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
    void increaseInstanceVal() {
        instanceVal++;
    }
}
```

메소드 오버로딩

- 같은 클래스 안에 메소드의 이름은 동일하지만 매개변수의 개수나 타입이 다른 메소드를 여러 개 정의하는 것

  - 메소드의 리턴 타입이 다른 것은 오버로딩이 아님
  - 가장 많이 사용하는 System.out의 println(), print() 메소드 등이 오버로딩의 대표적인 예

PrintStream.java

```
public void println() { … }
public void println(boolean x) { … }
public void println(char x) { … }
public void println(int x) { … }
public void println(long x) { … }
public void println(float x) { … }
public void println(double x) { … }
public void println(char[] x) { … }
public void println(String x) { … }
public void println(Object x) { … }
```

실습(1)

[Ex13] RetangleEx.java

```
package chap06.classex;

public class RectangleEx {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Rectangle r = new Rectangle();
        System.out.println(r.areaRectangle(10));
        System.out.println(r.areaRectangle(10, 20));
    }
}

class Rectangle{
    // 정사각형의 넓이
    double areaRectangle(double width) {
        return width * width;
    }
    // 직사각형의 넓이
    // areaRectangle(double width)를 오버로딩
    double areaRectangle(double width, double height) {
        return width * height;
    }
}
```

실습(2)

[Ex14] OverloadingEx.java

```java
package chap06.classex;

public class OverloadingEx {

    // 두 개의 정수를 더하는 메소드
    public int add(int a, int b) {
        return a + b;
    }

    // 세 개의 정수를 더하는 메소드
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // 두 개의 실수를 더하는 메소드
    public double add(double a, double b) {
        return a + b;
    }

    public static void main(String[] args) {
        OverloadingEx e = new OverloadingEx();

        // 두 개의 정수를 더하는 메소드 호출
        System.out.println("Sum of 2 integers: " + e.add(5, 3));
        // 세 개의 정수를 더하는 메소드 호출
        System.out.println("Sum of 3 integers: " + e.add(5, 3, 2));
        // 두 개의 실수를 더하는 메소드 호출
        System.out.println("Sum of 2 doubles: " + e.add(5.5, 3.3));
    }
}
```

배열(Array) 매개변수

- 배열을 메소드의 매개변수로 사용하여 메소드로 전달

```
public class VarArgs{
    int vaMethod(int[ ] nums){   // 배열 매개변수
        for(int i = 0; I < nums.length; i++){
            System.out.println(nums[i]);  // nums는 배열로 처리 가능
        }
    }
}

public class MainClass{
    public static void main(String[ ] args){
        int[ ] arr = new int[ ]{1, 2, 3, 4, 5};
        VarArgs va = new VarArgs();
        va.vaMethod(arr);
        va.vaMethod(new int[ ]{1, 2, 3});
    }
}
```

- main() 메소드도 배열을 매개변수로 입력 받음 → public static void main(String[ ] args)
  - 클래스 컴파일 시, 명령 라인으로 입력받는 매개변수

[Ex15] MainArrayArg.java

```
package chap06.classex;

public class MainArrayArgs {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int sum = 0;
        for(int i = 0; i < args.length; i++) {
            sum += Integer.parseInt(args[i]);
        }
        System.out.println("Sum : " + sum);
    }
}
```

실습(2)

[Ex16] ArrayArgsEx.java

```
package chap06.classex;

public class ArrayArgsEx {

    public String concatStr(String[] strs) {
        //StringBuilder result = new StringBuilder();
        String result = "";
        for (String str : strs) {
            result += str;
        }
        return result;
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ArrayArgsEx aArgs = new ArrayArgsEx();

        // 문자열 배열 생성
        String[] words1 = {"Hello", " ", "World", "!!"};
        String[] words2 = {"Java", " ", "is", " ", "fun!"};

        // 문자열 배열을 매개변수로 사용하는 메소드 호출
        System.out.println("Concatenated words1: " + aArgs.concatStr(words1));
        System.out.println("Concatenated words2: " + aArgs.concatStr(words2));
        System.out.println("Concatenated words3: " +
                aArgs.concatStr(new String[]{"Array", " ", "arguments", " ", "example"}));
    }
}
```

가변인자 매소드

- 메소드의 매개변수 개수를 모를 때 가변인자로 선언

  - 호출할 때마다 매개변수의 개수가 달라질 수 있으므로 오버로딩과 같은 효과 발생

  - 메소드에서는 해당 매개변수를 배열로 처리

```
public class VarArgs{
    int vaMethod(int ... nums){    // 매개변수를 가변 인자로 선언
        for(int i = 0; I < nums.length; i++){
            System.out.println(nums[i]);  // nums는 배열로 처리 가능
        }
    }
}

public class MainClass{
    public static void main(String[] args){
        VarArgs va = new VarArgs();
        va.vaMethod(1, 2, 3, 4, 5);
        va.vaMethod(1, 2, 3);
    }
}
```

  - 배열을 매개변수로 넘기는 것과 비슷하나 배열을 생성할 필요가 없다는 장점

실습(1)

[Ex17-1] Sum.java

```
package chap06.classex;

public class Sum {
    int suml(int[] values) {
        int sum = 0;
        for(int value: values) {
            sum += value;
        }
        return sum;
    }

    int sum2(int ... values) {
        int sum = 0;
        for(int i=0; i<values.length; i++) {
            sum += values[i];
        }
        return sum;
    }
}
```

실습(2)

[Ex17-2] Sum.java

```
package chap06.classex;

public class SumEx {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Sum s = new Sum();

        int[] values1 = {1, 2, 3};
        int result1 = s.sum1(values1);
        System.out.println("result1: " + result1);

        int result2 = s.sum1(new int[] {1, 2, 3, 4, 5});
        System.out.println("result2: " + result2);

        int result3 = s.sum2(1, 2, 3);
        System.out.println("result3: " + result3);

        int result4 = s.sum2(1, 2, 3, 4, 5);
        System.out.println("result4: " + result4);
    }
}
```

접근 제한자(Access Modifier)

- 클래스, 메소드, 멤버변수 등에 대한 접근을 제한하기 위한 키워드

  - public, protected, private, default 의 접근 제한자가 있음
    - public, protected, private의 접근 제한자가 지정되어 있지 않은 경우, default 접근 제어로 인식됨
  - 클래스는 public과 default 접근 제한만 가질 수 있음
  - public으로 선언된 외부 패키지의 클래스를 접근하기 위해서는 **import**를 통해 선언해야 함

	접근 범위				사용 가능			
	같은 클래스	같은 패키지	자식 클래스	외부 패키지	클래스	생성자	멤버변수	메소드
public	O	O	O	O	O	O	O	O
protected	O	O	O	X	X	O	O	O
default	O	O	X	X	O	O	O	O
private	O	X	X	X	X	O	O	O

실습(1)

[Ex18-1] A.java

```java
package chap07.package1;

class A {}
```

[Ex18-2] B.java

```java
package chap07.package1;

public class B {
    A a = new A();
}
```

[Ex18-3] C.java

```java
package chap07.package2;

import chap07.package1.*;

public class C {
    // A a = new A(); // 에러 (A가 default 제한자를 가지고 있으므로)
    B b = new B();
}
```

실습(2)

[Ex19-1] A.java

```java
package chap07.package1;

public class A {
    A a1 = new A(true);
    A a2 = new A(1);
    A a3 = new A("문자열");

    // 생성자
    public A(boolean b) {}
    A(int b){}  // default 제한자
    private A(String s) {}
}
```

[Ex19-2] B.java

```java
package chap07.package1;

public class B {
    A a1 = new A(true);
    A a2 = new A(1);

    // 에러 (A 클래스의 private 생성자에 대한 접근 시도
    // A a3 = new A("문자열");
}
```

[Ex19-3] C.java

```java
package chap07.package2;

import chap07.package1.*;

public class C {
    A a1 = new A(true);
    // A a2 = new A(1); // 에러 (A 클래스의 default 생성자에 대한 접근 시도)
    // A a3 = new A("문자열");  // 에러 (A 클래스의 private 생성자에 대한 접근 시도)
}
```

실습(3)

[Ex20-1] A.java

```java
package chap07.package1;

class A {  // default 제한자
    A a1 = new A(true);
    A a2 = new A(1);
    A a3 = new A("문자열");

    // 생성자
    public A(boolean b) {}
    A(int b){}  // default 제한자
    private A(String s) {}
}
```

[Ex20-2] B.java

```java
package chap07.package1;

public class B {
    A a1 = new A(true);
    A a2 = new A(1);

    // 에러 (A 클래스의 private 생성자에 대한 접근 시도
    // A a3 = new A("문자열");
}
```

[Ex20-3] C.java

```java
package chap07.package2;

import chap07.package1.*;

public class C {
    // A a1 = new A(true);/ / 에러 (default로 선언된 A 클래스에 대한 접근 시도)
    // A a2 = new A(1); // 에러 (A 클래스의 default 생성자에 대한 접근 시도)
    // A a3 = new A("문자열");  // 에러 (A 클래스의 private 생성자에 대한 접근 시도)
}
```

실습(4)

[Ex21-1] A.java

```java
package chap07.package1;

public class A {
    protected String field;

    protected A() {
    }

    protected void method() {
    }
}
```

[Ex21-2] B.java

```java
package chap07.package1;

public class B {
    public void method() {
        A a = new A();
        a.field = "value";
        a.method();
    }
}
```

[Ex21-3] C.java

```java
package chap07.package2;

import chap07.package1.A;

public class C {
    public void method() {
        /* A의 protected 메소드는 접근 불가능
        A a = new A();
        a.field = "value";
        a.method();
        */
    }
}
```

[Ex21-4] D.java

```java
package chap07.package2;

import chap07.package1.*;

public class D extends A {
    public D() {
        super();  // 생략 가능
        this.field = "value";
        this.method();
    }
}
```

실습(5)

[Ex22-1] Singleton.java

```java
package chap06.classex;

public class Singleton {
    private static Singleton singleton = new Singleton();

    private Singleton() {}

    static Singleton getInstance() {
        return singleton;
    }
}
```

[Ex22-2] SingletonExample.java

```java
package chap06.classex;

public class SingletonExample {
    public static void main(String[] args) {
        /*
        Singleton obj1 = new Singleton();  //컴파일 에러
        Singleton obj2 = new Singleton();  //컴파일 에러
        */

        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();

        if(obj1 == obj2) {
            System.out.println("같은 Singleton 객체 입니다.");
        } else {
            System.out.println("다른 Singleton 객체 입니다.");
        }
    }
}
```