# java.lang package

- String 클래스
- 배열
- 콘솔 출력

# String 클래스

# ☞ String 클래스의 인스턴스 생성

· Java는 큰 따옴표로 묶여서 표현되는 문자열을 모두 인스턴스화 한다.

· 문자열은 String 이라는 이름의 클래스로 표현된다.

```
String str1 = "String Instance";
String str2 = "My String";
```

두 개의 String 인스턴스 생성,
그리고 참조변수 str1과 str2로 참조.

```
System.out.println("Hello JAVA!");
System.out.println("My Coffee");
```

println 메소드의 매개 변수형이 String이기때문에 이러한 문장의 구성이 가능하다.

```
class StringInstance {
    public static void main(String[] args) {
        java.lang.String str="My name is Sunny";

        int strLen1=str.length();
        System.out.println("길이 1: "+strLen1);

        int strLen2 = "한글의 길이는 어떻게?".length();
        System.out.println("길이 2: "+strLen2);
    }
}
```

문자열의 선언은 인스턴스의 생성으로 이어짐을 보이는 문장

## ☞ String 인스턴스는 상수 형태의 인스턴스다.

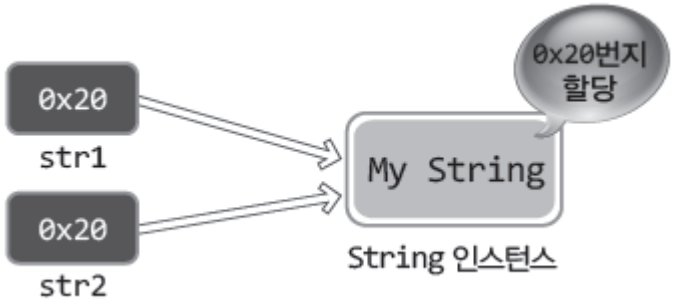
· String 인스턴스에 저장된 문자열의 내용은 변경이 불가능하다.

· 이는 동일한 문자열의 인스턴스를 하나만 생성해서 공유하기 위함이다.

```
class ImmutableString {
    public static void main(String[] args) {
        String str1="My String";
        String str2="My String";
        String str3="Your String";

        if(str1 == str2)
            System.out.println("동일 인스턴스 참조");
        else
            System.out.println("다른 인스턴스 참조");


        if(str2 == str3)
            System.out.println("동일 인스턴스 참조");
        else
            System.out.println("다른 인스턴스 참조");
    }
}
```




⇒String 인스턴스의 문자열 변경이 불가능하기 때문에 둘 이상의 참조 변수가 동시에 참조를 해도 문제가 발생하지 않는다!

## ☞ String 클래스가 제공하는 유용한 메소드들

- 문자열의 길이 반환 — public int length()
- 두 문자열의 결합 — public String concat(String str)
- 두 문자열의 비교 — public int compareTo(String anotherString)

```
class StringMethod {
        public static void main(String[] args) {
                String str1="Smart";
                String str2=" and ";
                String str3="Simple";
                String str4=str1.concat(str2).concat(str3);

                System.out.println(str4);
                System.out.println("문자열 길이: "+str4.length());

                if(str1.compareTo(str3)<0)
                        System.out.println("str1이 앞선다");
                else
                        System.out.println("str3이 앞선다");
        }
}
```

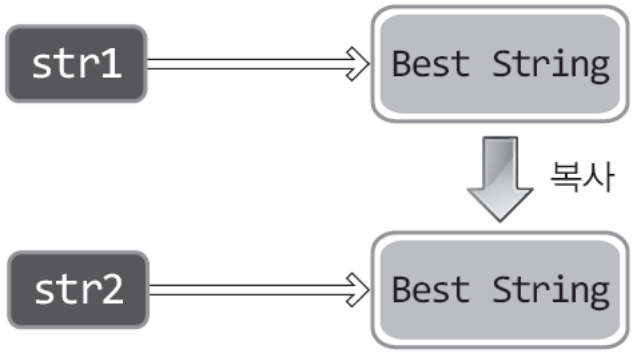
# ☞ 자바에서의 문자열 복사!

· 자바에서는 문자열을 상수의 형태로 관리하고, 또 동일한 유형의 문자열을 둘 이상 유지하지 않으므로 문자열의 복사라는 표현이 흔하지 않다.

· 무엇보다도 자바에서는 문자열을 복사할 필요가 없다.

⇒ 그러나 원하는 것이 인스턴스를 새로 생성해서 문자열의 내용을 그대로 복사하는 것이라면 다음과 같이 코드를 구성하면 된다.

```
String str1="Best String";
String str2=new String(str1);
```




```
public String(String original)
```

✓ 새로운 문자열 인스턴스의 생성에 사용되는 생성자

```
class StringCopy {
    public static void main(String[] args) {
        String str1="Lemon";
        String str2="Lemon";
        String str3=new String(str2);

        if(str1 == str2) // 비교 연산자는 참조값 비교
            System.out.println("str1과 str2는 동일 인스턴스 참조");
        else
            System.out.println("str1과 str2는 다른  인스턴스 참조
");
        if(str2 == str3)
            System.out.println("str2와 str3는 동일 인스턴스 참조");
        else
            System.out.println("str2와 str3는 다른  인스턴스 참조
");
    }
}
```

# ☞ + 연산과 += 연산의 진실

```
class StringAdd {
	public static void main(String[] args) {
		String str1="Lemon"+"ade";
		String str2="Lemon"+'A';
		String str3="Lemon"+3;
		String str4=1+"Lemon"+2;
		str4 += '!';

		System.out.println(str1);
		System.out.println(str2);
		System.out.println(str3);
		System.out.println(str4);
	}
}
```

```
String str1="Lemon".concat("ade");
String str2="Lemon".concat(String.valueOf('A'));
String str3="Lemon".concat(String.valueOf(3));
```

위 예제의 str4의 선언이 다음과 같이 처리 된다면?

```
String str4 = String.valueOf(1).concat("Lemon").concat(String.valueOf(2));
```

아무리 많은 + 연산을 취하더라도 딱 두 개의 인스턴스만 생성된다. StringBuilder 클래스의 도움으로...

# ☞ StringBuilder

· StringBuilder는 문자열의 저장 및 변경을 위한 메모리 공간을 지니는 클래스

· 문자열 데이터의 추가를 위한 append와 삽입을 위한 insert 메소드 제공

```
class BuilderString {
    public static void main(String[] args) {
        StringBuilder strBuf = new StringBuilder("AB"); // buf : AB
        strBuf.append(25); // buf : AB25
        strBuf.append('Y').append(true); // buf : AB25Ytrue
        System.out.println(strBuf);

        strBuf.insert(2, false); // buf : ABfalse25Ytrue
        strBuf.insert(strBuf.length(), 'Z'); // buf : ABfalse25YtrueZ
        System.out.println(strBuf);
    }
}
```

✓ 연속해서 함수 호출이 가능한 이유는 append 메소드가 strBuf의 참조값을 반환하기 때문이다.

# ☞ 참조를 반환하는 메소드

· this의 반환은 인스턴스 자신의 참조 값 반환을 의미한다.

· 그리고 이렇게 반환되는 참조 값을 대상으로 연이은 함수 호출이 가능하다.

```
class SimpleAdder {
        private int num;
        public SimpleAdder(){num=0;}

        public SimpleAdder add(int num) {
                this.num+=num;
                return this;
        }
        public void showResult() {
                System.out.println("add result: "+num);
        }
}
class SelfReference {
        public static void main(String[] args)  {
                SimpleAdder adder=new SimpleAdder();
                adder.add(1).add(3).add(5).showResult();
        }
}
```

add 함수는 adder의 참조 값을 반환한다.

# ☞ StringBuilder의 버퍼와 문자열 조합

· 추가되는 데이터 크기에 따라서 버퍼의 크기가 자동으로 확장된다.

· 생성자를 통해서 초기 버퍼의 크기를 지정할 수 있다.

- `public StringBuilder()` // 기본 16개의 문자 저장 버퍼 생성
- `public StringBuilder(int capacity)` // capacity개의 문자 저장 버퍼 생성
- `public StringBuilder(String str)` // str.length()+16 크기의 버퍼 생성

· 문자열의 복잡한 조합의 과정에서는 StringBuilder의 인스턴스가 활용된다.

· 때문에 추가로 생성되는 인스턴스의 수는 최대 두 개이다.

```
String str4=1+"Lemon"+2;
```

```
new StringBuilder().append(1).append("Lemon").append(2).toString();
```

StringBuilder 인스턴스의 생성에서 한 개

toString 메소드의 호출에 의해서 한 개

# ☞ StringBuffer 클래스

· StringBuffer 클래스와 StringBuilder 클래스의 공통점

- 메소드의 수(생성자 포함)
- 메소드의 기능
- 메소드의 이름과 매개변수형

· StringBuffer 클래스와 StringBuilder 클래스의 차이점

- StringBuffer는 thread에 안전, StringBuilder는 thread에 불안전

=> BuilderString 클래스에서,
StringBuilder를 StringBuffer로 바꿔도 컴파일 및 실행이 된다.

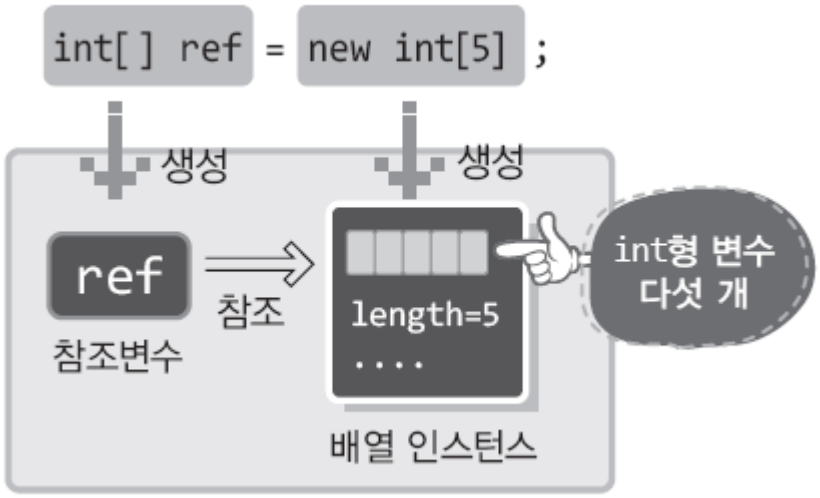
# 배 열

# 배열 인스턴스의 생성

- 일반적인 인스턴스 생성 방법
  AAA ref = new AAA(5);

- 배열 생성
  int[] ref = new int[5];

- 클래스 기반의 배열 선언 가능
  FruitSeller[] arr4 = new FruitSeller[5];
  FruitBuyer[] arr5 = new FruitBuyer[8];

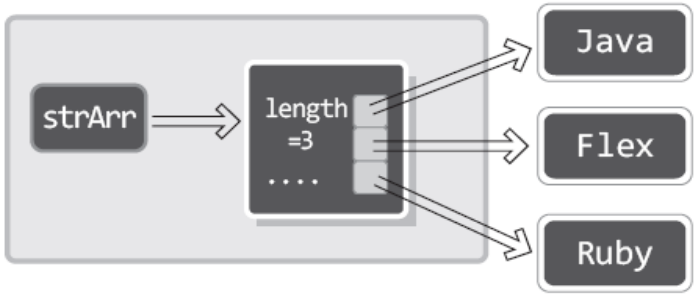


# 배열의 접근 방법

```
1.  class AccessArray {
2.      public static void main(String[] args) {
3.          int[] arr = new int[3];
4.          arr[0] = 1;
5.          arr[1] = 2;
6.          arr[2] = 3;
7.
8.          int sum = arr[0] + arr[1] + arr[2];
9.          System.out.println(sum);
10.     }
11. }
```

# 배열의 접근 방법

```
1.  class InstanceArray {
2.      public static void main(String[] args) {
3.          String[] strArr = new String[3];
4.          strArr[0] = new String("Java");
5.          strArr[1] = new String("Flex");
6.          strArr[2] = new String("Ruby");
7.
8.          for(int i = 0; i < strArr.length; i++)
9.              System.out.println(strArr[i]);
10.     }
11. }
```



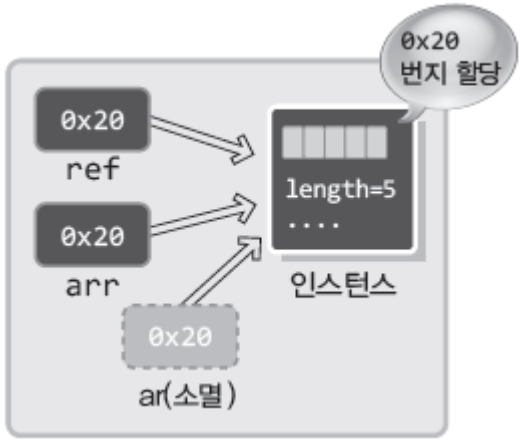

- 인스턴스 배열에는 인스턴스가 저장되는 것이 아니라, 인스턴스의 참조값이 저장된다.

# 배열의 선언과 동시에 초기화

- int[] arr = new int[3] {1, 2, 3}; // error
- int[] arr = new int[] {1, 2, 3};
- int[] arr = {1, 2, 3};

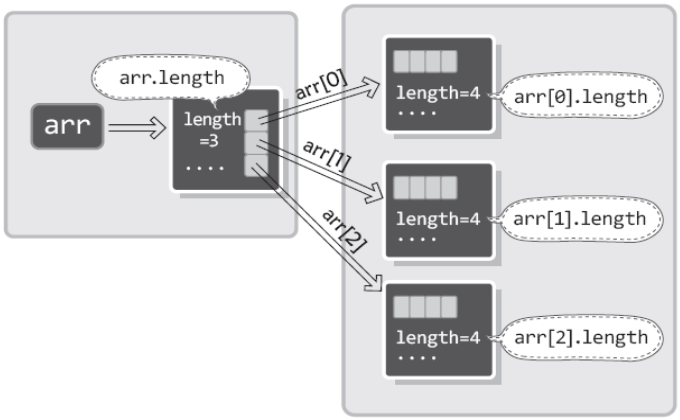
# 배열과 메소드

```
1.    class ArrayAndMethods {
2.        public static int[] addAllArray(int[] ar, int addVal) {
3.            for(int i=0; i<ar.length; i++)
4.                ar[i] += addVal;
5.            return ar;
6.        }
7.        public static void main(String[] args) {
8.            int[] arr={1, 2, 3, 4, 5};
9.            int[] ref;
10.           ref=addAllArray(arr, 7);
11.           if(arr==ref)
12.               System.out.println("동일 배열 참조");
13.           else
14.               System.out.println("다른 배열 참조");
15.
16.           for(int i=0; i<ref.length; i++)
17.               System.out.print(arr[i]+" ");
18.       }
19.   }
```



# 다차원 배열

```
1.  class TwoDimenArrayInstance {
2.      public static void main(String[] args) {
3.          int[][] arr=new int[3][4];
4.
5.          for(int i=0; i < arr.length; i++)
6.              for(int j=0; j<arr[i].lengtn; j++)
7.                  arr[i][j]=i+j;
8.
9.          for(int i=0; i < arr.length; i++) {
10.             for(int j=0; j<arr[i].length; j++)
11.                 System.out.print(arr[i][j]+" ");
12.
13.             System.out.println("");
14.         }
15.     }
16. }
```



# 2차원 배열의 선언 및 초기화

- int[][] arr = {
  {1, 2, 3, 4},
  {5, 6, 7, 8},
  {9, 10, 11, 12}
  };

- int[][] arr = {
  {1, 2},
  {3, 4, 5},
  {6, 7, 8, 9}
  };

- int[][] arr = new int[][]
  {
  {1, 2, 3, 4},
  {5, 6, 7, 8},
  {9, 10, 11, 12}
  };

# 행과 열의 분리선언

- int[][] arr = new int[3][]; // ragged array
  arr[0] = new int[2];
  arr[1] = new int[3];
  arr[2] = new int[4];

# for – each 문

- 배열의 전체를 참조할 필요가 있는 경우에 유용.
- for(int i = 0; i < arr.length; i++)
  System.out.print(arr[i] + “ ”);

- for(int e : arr)
  System.out.print(e + “”);

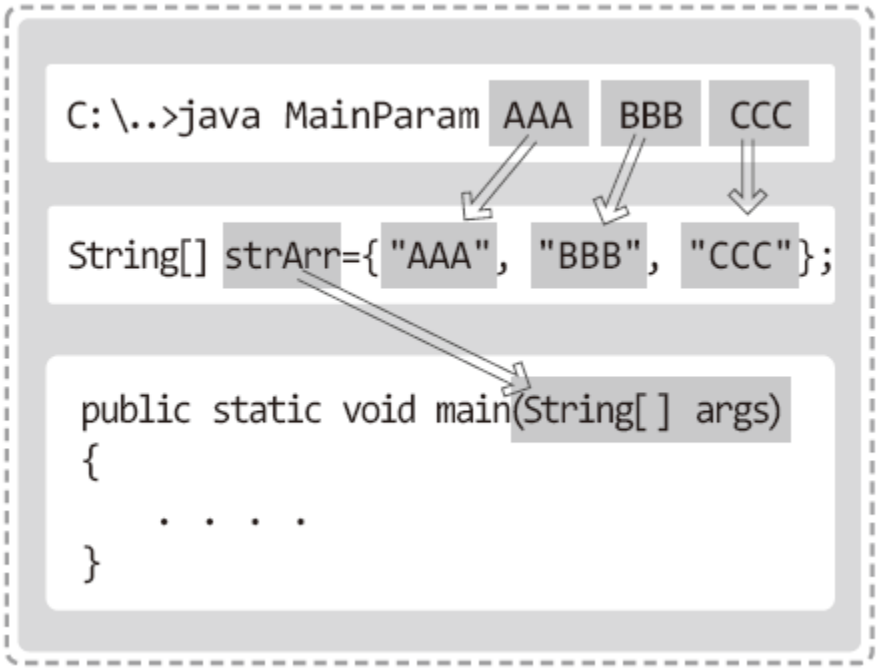
- for-each 문을 통한 값의 변경은 실제 배열에 반영되지 않으니, 값의 참조를 목적으로만 사용해야 한다.

# 인스턴스 배열에 대한 for-each문

```
1.  class Number {
2.      public int num;
3.      public Number(int num) { this.num=num; }
4.      public int getNum() { return num; }
5.  }
6.  class EnhancedForInst {
7.      public static void main(String[] args) {
8.          Number[] arr = new Number[]{
9.                                  new Number(2),
10.                                 new Number(4),
11.                                 new Number(8)
12.                             };
13.         for(Number e: arr)
14.             e.num++;
15.         for(Number e: arr)
16.             System.out.print(e.getNum()+" ");
17.         System.out.println("");
18.         for(Number e: arr) {
19.             e=new Number(5);
20.             e.num+=2;
21.             System.out.print(e.getNum()+" ");
22.         }
23.         System.out.println("");
24.         for(Number e: arr)
25.             System.out.print(e.getNum()+" ");
26.     }
27. }
```

# main으로의 데이터 전달방법

```
C:\JavaStudy>java MainParam AAA BBB CCC
AAA
BBB
CCC
```




그림에서 보이듯이 명령프롬프트 상에서 전달되는, 공백으로 구분되는 문자열로 String 배열이 구성되어 이 배열의 참조값이 전달된다.

# 콘솔 출력

# ☞ 콘솔 출력 : System.out.println과 System.out.print

- println 메소드는 출력 후에 개행을 한다.
- print 메소드는 출력 후에 개행을 하지 않는다.
- println, print 메소드의 인자로 인스턴스의 참조 값이 전달 될 수 있다.

```
class Friend {
    String myName;
    public Friend(String name) {
        myName=name;
    }

    public String toString() {
        return "제 이름은 "+myName+"입니다.";
    }
}
```

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

        Friend fnd1=new Friend("이종수");
        Friend fnd2=new Friend("현주은");

        System.out.println(fnd1);
        System.out.println(fnd2);

        System.out.print("출력이 ");
        System.out.print("종료되었습니다.");
        System.out.println("");
    }
}
```

인스턴스의 참조값이 전달

println 메소드에 인스턴스의 참조 값이 전달되면, 인스턴스의 toString 메소드가 반환하는 문자열이 출력된다!

# ☞ Escape Sequence

· 문자열 안에서 특별한 의미로 해석되는 문자를 가리켜 '이스케이프 시퀀스'라 한다.

- \n 개행
- \t 탭(Tab)
- \" 큰 따옴표(Quatation mark)
- \\ 역슬래쉬(Backslash)

대표적인 이스케이프 시퀀스

```
System.out.println("제가 어제 "당신 누구세요?" 라고 물었더니");
```

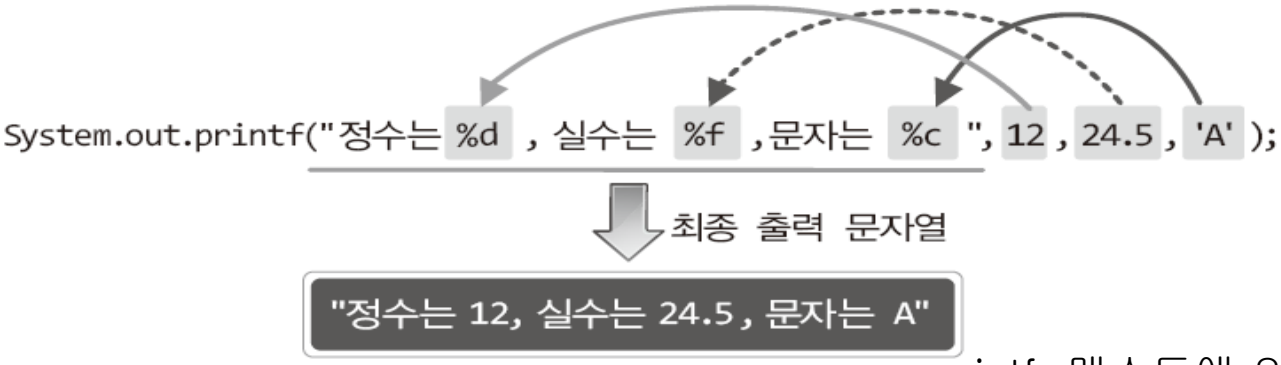
문자열 안에 큰따옴표가 들어가면 이는 문자열의 구분자로 인식된다.

```
System.out.println("제가 어제 \"당신 누구세요?\"라고 물었더니");
```

문자열 안에 큰 따옴표를 삽입하려면 이스케이프 시퀀스 사용!

# ☞ 문자열을 조합해서 출력하는 printf

- System.out.printf 메소드는 문자열의 중간에 삽입될 데이터를 가지고 하나의 문자열을 조합해서 출력한다.




printf 메소드에 의한 문자열의 조합

| 서식문자 | 출력의 형태 |
|---|---|
| %d | 10진수 정수 형태의 출력 |
| %o | 8진수 정수 형태의 출력 |
| %x | 16진수 정수 형태의 출력 |
| %f | 실수의 출력 |
| %e | e 표기법 기반의 실수 출력 |
| %g | 출력의 대상에 따라서 %e 또는 %f 형태의 출력 |
| %s | 문자열 출력 |
| %c | 문자 출력 |

문자열의 조합에 사용되는 서식 문자들

# ☞ printf 메소드의 호출 예

```
class FormatString {
    public static void main(String[] args) {
        int age=20;
        double tall=175.7;
        String name="홍자바";

        System.out.printf("제 이름은 %s입니다. ₩n", name);
        System.out.printf("나이는 %d이고, 키는 %e입니다. ₩n", age, tall);
        System.out.printf("%d %o %x ₩n", 77, 77, 77);
        System.out.printf("%g %g ₩n", 0.00014, 0.000014);
    }
}
```

0.00000000000000000001 일반표현

$1.0 \times 10^{-20}$ 지수표현

1.0e-20 e표기법

# java.util package

- 콘솔 입력(scanner)
- StringTokenizer 클래스

## ☞ 콘솔 입력

```
Scanner kb = new Scanner(System.in);
int num = kb.nextInt(); // 정수 입력 방식
```

## ☞ Scanner 클래스 생성자

- Scanner(File source)
- Scanner(InputStream source)
- Scanner(Readable source)
- Scanner(String source)

Scanner 클래스는 단순히 키보드의 입력만을 목적으로 디자인된 클래스가 아니다. 스캐너 클래스는 다양한 리소스를 대상으로 입력을 받을 수 있도록 정의된 클래스이다.

ex)

```
import java.util.Scanner;

class StringScanning {
    public static void main(String[] args) {
        String source="1 5 7";
        Scanner sc=new Scanner(source);

        int num1 = sc.nextInt();
        int num2 = sc.nextInt();
        int num3 = sc.nextInt();
        int sum = num1+num2+num3;

        System.out.printf("문자열에 저장된 %d, %d, %d의 합은 %d ₩n",
                                                    num1, num2,num3, sum);
    }
}
```

문자열을 대상으로 Scanner의 인스턴스를 생성한 예

## ☞ 키보드 적용

```
import java.util.Scanner;

class KeyboardScanning
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);

                int num1=sc.nextInt();
                int num2=sc.nextInt();
                int num3=sc.nextInt();
                int sum=num1+num2+num3;

                System.out.printf( "입력된 정수 %d, %d, %d의 합은 %d ₩n",
                                                num1, num2,num3, sum);
        }
}
```

Scanner 클래스를 이용하면, 데이터를 읽어 들일 입력의 대상에 상관없이 동일한 방식으로 데이터를 읽어 들일 수 있다!

# ☞ Scanner 클래스를 구성하는 다양한 메소드

- public boolean nextBoolean()
- public byte nextByte()
- public short nextShort()
- public int nextInt()
- public long nextLong()
- public float nextFloat()
- public double nextDouble()
- public String nextLine()

=> 읽어 들일 데이터의
유형에 따른 메소드 정의

```
import java.util.Scanner;

class ScanningMethods {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        System.out.print("당신의 이름은? ");
        String str = keyboard.nextLine();
        System.out.println("안녕하세요 "+str+'님');

        System.out.print("당신은 스파게티를 좋아한다는데, 진실입니까? ");
        boolean isTrue = keyboard.nextBoolean();
        if(isTrue == true)
            System.out.println("오~ 좋아하는군요.");
         else
            System.out.println("이런 아니었군요.");

        System.out.print("당신과 동생의 키는 어떻게 되나요? ");
        double num1 = keyboard.nextDouble();
        double num2 = keyboard.nextDouble();
        double diff = num1-num2;
        if(diff > 0)
            System.out.println("당신이 "+diff+"만큼 크군요.");
        else
            System.out.println("당신이 "+(-diff)+"만큼 작군요.");
    }
}
```

# ☞ StringTokenizer 클래스 : 문자열 토큰(Token)의 구분

"08 : 45"

"11 : 24"

콜론 : 을 기준으로 문자열을 나눈다고 할 때,
=> 08, 45, 11, 24는 토큰(token)
=> 콜론 : 는 구분자(delimiter)

```
public static void main(String[] args)
{
    String strData="11:22:33:44:55";
    StringTokenizer st=new StringTokenizer(strData, ":");

    while(st.hasMoreTokens())
        System.out.println(st.nextToken());
}
```

```
11
22
33
44
55
```

구분자 정보는 둘 이상 담을 수 있다. 하나의 문자열 안에 둘 이상을 담을 수 있다.