

자바의 기본 클래스

20-1. 래퍼 클래스

기본 자료형의 값을 감싸는 래퍼 클래스

```
class UseWrapperClass {  
    public static void showData(Object obj) {  
        System.out.println(obj);  
    }  
  
    public static void main(String[] args) {  
        Integer iInst = new Integer(3);  
        showData(iInst);  
        showData(new Double(7.15));  
    }  
}
```

인스턴스를 요구하는 메소드

이 메소드를 통해서 정수나 실수를 출력하려면 해당 값을 인스턴스화 해야 한다.

이렇듯 기본 자료형의 값을 인스턴스로 감싸는 목적의 클래스를 가리켜 래퍼 클래스라 한다.

래퍼 클래스의 종류와 생성자

Boolean `public Boolean(boolean value)`

Character `public Character(char value)`

Byte `public Byte(byte value)`

Short `public Short(short value)`

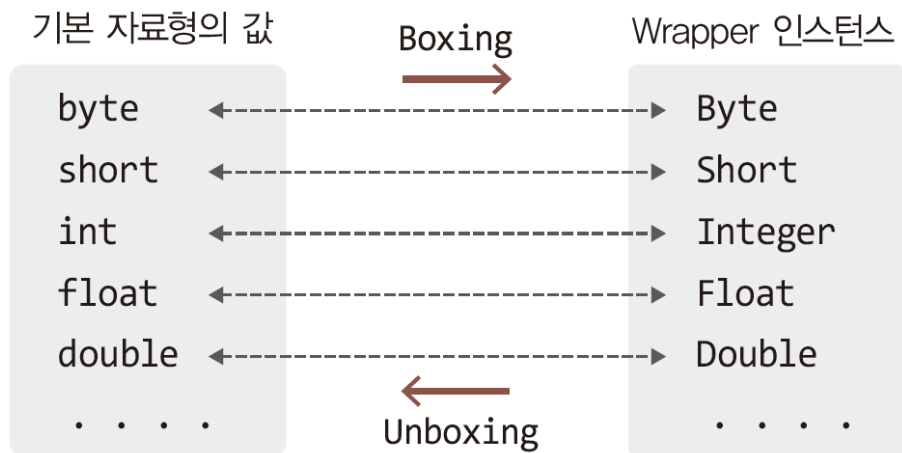
Integer `public Integer(int value)`

Long `public Long(long value)`

Float `public Float(float value), public Float(double value)`

Double `public Double(double value)`

래퍼 클래스의 두가지 기능



박싱과 언박싱 예

```
public static void main(String[] args) {  
    Integer iObj = new Integer(10);    // 박싱  
    Double dObj = new Double(3.14);    // 박싱  
    . . . .  
  
    int num1 = iObj.intValue();         // 언박싱  
    double num2 = dObj.doubleValue();   // 언박싱  
    . . . .  
  
    // 래퍼 인스턴스 값의 증가 방법  
    iObj = new Integer(iObj.intValue() + 10);  
    dObj = new Double(dObj.doubleValue() + 1.2);  
    . . . .  
}
```

언박싱 메소드의 이름

```
Boolean                                public boolean
```

booleanValue()

```
Character      public char charValue()
```

```
Integer                public int
```

```
intValue()
```

Long public long

longValue()

```
Double                                public double
```

doubleValue()

오토 박싱과 오토 언박싱

```
class AutoBoxingUnboxing {  
    public static void main(String[] args) {  
        Integer iObj = 10;    // 오토 박싱 진행  
        Double dObj = 3.14;   // 오토 박싱 진행  
  
        . . . . .    인스턴스가 와야 할 위치에 기본 자료형 값이 오면 오토 박싱 진행  
  
        int num1 = iObj;      // 오토 언박싱 진행  
        double num2 = dObj;   // 오토 언박싱 진행  
  
        . . . . .    기본 자료형 값이 와야 할 위치에 인스턴스가 오면 오토 언박싱 진행  
    }  
}
```


오토 박싱, 오토 언박싱의 또 다른 예

```
public static void main(String[] args) {  
    Integer num = 10;  
    num++;      // new Integer(num.intValue() + 1);  
    . . . .      오토 박싱과 오토 언박싱 동시에 진행!  
  
    num += 3;    // new Integer(num.intValue() + 3);  
    . . . .      오토 박싱과 오토 언박싱 동시에 진행!  
  
    int r = num + 5;    // 오토 언박싱 진행  
    Integer rObj = num - 5;    // 오토 언박싱 진행  
    . . . .  
}
```

Number 클래스

`java.lang.Number`

모든 래퍼 클래스가 상속하는 클래스

`java.lang.Number`에 정의된 추상 메소드들

```
public abstract int intValue()
```

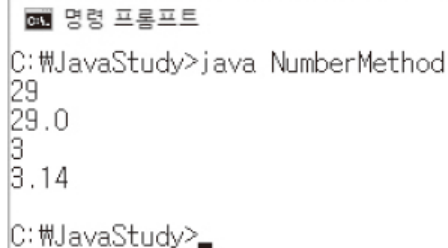
```
public abstract long longValue()
```

```
public abstract double doubleValue()
```

→ 즉 래퍼 인스턴스에 저장된 값을 원하는 형의 기본 자료형 값으로 반환할 수 있다.

Number 클래스의 추상 메소드 호출의 예

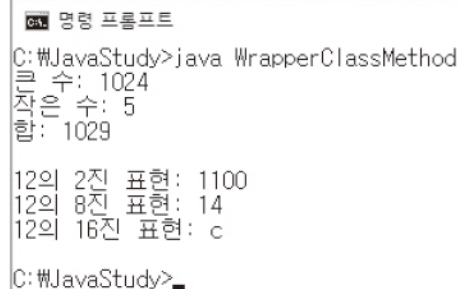
```
public static void main(String[] args) {  
    Integer num1 = new Integer(29);  
    System.out.println(num1.intValue());    // int형 값으로 반환  
    System.out.println(num1.doubleValue()); // double형 값으로 반환  
  
    Double num2 = new Double(3.14);  
    System.out.println(num2.intValue());    // int형 값으로 반환  
    System.out.println(num2.doubleValue()); // double형 값으로 반환  
}
```



```
C:\#JavaStudy>java NumberMethod  
29  
29.0  
3  
3.14  
C:\#JavaStudy>
```

래퍼 클래스의 다양한 static 메소드들

```
public static void main(String[] args) {  
    // 클래스 메소드를 통한 인스턴스 생성 방법 두 가지  
    Integer n1 = Integer.valueOf(5);    // 숫자 기반 Integer 인스턴스 생성  
    Integer n2 = Integer.valueOf("1024");    // 문자열 기반 Integer 인스턴스 생성  
  
    // 대소 비교와 합을 계산하는 클래스 메소드  
    System.out.println("큰 수: " + Integer.max(n1, n2));  
    System.out.println("작은 수: " + Integer.min(n1, n2));  
    System.out.println("합: " + Integer.sum(n1, n2));  
    System.out.println();  
  
    // 정수에 대한 2진, 8진, 16진수 표현 결과를 반환하는 클래스 메소드  
    System.out.println("12의 2진 표현: " + Integer.toBinaryString(12));  
    System.out.println("12의 8진 표현: " + Integer.toOctalString(12));  
    System.out.println("12의 16진 표현: " + Integer.toHexString(12));  
}
```



```
C:\JavaStudy>java WrapperClassMethod  
큰 수: 1024  
작은 수: 5  
합: 1029  
  
12의 2진 표현: 1100  
12의 8진 표현: 14  
12의 16진 표현: c  
  
C:\JavaStudy>
```

20-2. BigInteger 클래스와 BigDecimal 클래스

매우 큰 정수 표현 위한 java.math.BigInteger 클래스

```
public static void main(String[] args) {
    // long형으로 표현 가능한 값의 크기 출력
    System.out.println("최대 정수: " + Long.MAX_VALUE);
    System.out.println("최소 정수: " + Long.MIN_VALUE);
    System.out.println();

    // 매우 큰 수를 BigInteger 인스턴스로 표현
    BigInteger big1 = new BigInteger("1000000000000000000000000");
    BigInteger big2 = new BigInteger("-999999999999999999999999");

    // BigInteger 기반 덧셈 연산
    BigInteger r1 = big1.add(big2);
    System.out.println("덧셈 결과: " + r1);

    // BigInteger 기반 곱셈 연산
    BigInteger r2 = big1.multiply(big2);
    System.out.println("곱셈 결과: " + r2);
    System.out.println();

    // 인스턴스에 저장된 값을 int형 정수로 반환
    int num = r1.intValueExact();
    System.out.println("From BigInteger: " + num);
}
```

```
C:\JavaStudy>java SoBigInteger
최대 정수: 9223372036854775807
최소 정수: -9223372036854775808
```

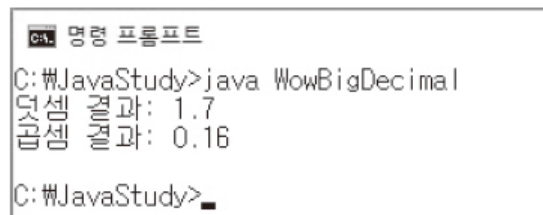
[illegible]

```
|From BigInteger: 1
```

```
C:\#\JavaStudy>
```

오차 없는 실수 표현 위한 BigDecimal 클래스

```
public static void main(String[] args) {  
    BigDecimal d1 = new BigDecimal("1.6");  
    BigDecimal d2 = new BigDecimal("0.1");  
    System.out.println("덧셈 결과: " + d1.add(d2));  
    System.out.println("곱셈 결과: " + d1.multiply(d2));  
}
```



```
C:\JavaStudy>java WowBigDecimal  
덧셈 결과: 1.7  
곱셈 결과: 0.16  
C:\JavaStudy>
```

덧셈

```
public BigDecimal add(BigDecimal augend)
```

뺄셈

```
public BigDecimal subtract(BigDecimal
```

subtrahend)

곱셈

```
public BigDecimal multiply(BigDecimal
```

multiplicand)

나눗셈

```
public BigDecimal divide(BigDecimal divisor)
```

20-3. Math 클래스와 난수의 생성, 그리고 문자열 토큰의 구분

수학 관련 연산 기능을 제공하는 Math 클래스

```
public static void main(String[] args) {  
    System.out.println("원주율: " + Math.PI);  
    System.out.println("2의 제곱근: " + Math.sqrt(2));  
    System.out.println();  
    System.out.println("파이에 대한 Degree: " + Math.toDegrees(Math.PI));  
    System.out.println("2 파이에 대한 Degree: " + Math.toDegrees(2.0 * Math.PI));  
    System.out.println();  
  
    double radian45 = Math.toRadians(45); // 라디안으로의 변환!  
    System.out.println("싸인 45: " + Math.sin(radian45));  
    System.out.println("코싸인 45: " + Math.cos(radian45));  
    System.out.println("탄젠트 45: " + Math.tan(radian45));  
    System.out.println();  
    System.out.println("로그 25: " + Math.log(25));  
    System.out.println("2의 16승: " + Math.pow(2, 16));  
}
```

04. 명령 프롬프트

```
C:\JavaStudy>java SimpleMathUse  
원주율: 3.141592653589793  
2의 제곱근: 1.4142135623730951
```

```
파이에 대한 Degree: 180.0  
2 파이에 대한 Degree: 360.0
```

```
싸인 45: 0.7071067811865475  
코싸인 45: 0.7071067811865476  
탄젠트 45: 0.9999999999999999
```

```
로그 25: 3.2188758248682006  
2의 16승: 65536.0
```

```
C:\JavaStudy>
```

난수의 생성

```
Random rand = new Random();
```

```
public boolean nextBoolean()
```

boolean형 난수 반환

```
public int nextInt()
```

int형 난수 반환

```
public long nextLong()
```

long형 난수 반환

```
public int nextInt(int bound)
```

0 이상 bound 미만 범위의 int형 난수 반환

```
public float nextFloat()
```

0.0 이상 1.0 미만의 float형 난수

반환

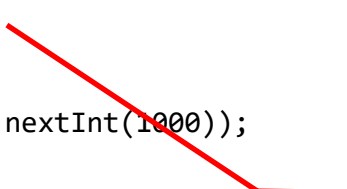
```
public double nextDouble()
```

0.0 이상 1.0 미만의 double형 난수

반환

난수 생성의 예

```
public static void main(String[] args) {  
    Random rand = new Random();  
    for(int i = 0; i < 7; i++)  
        System.out.println(rand.nextInt(1000));  
}
```



```
public Random() {  
    // Random(long seed) 생성자 호출  
    this(System.currentTimeMillis());  
}
```

```
public static void main(String[] args) { 실행할 때마다 같은 결과를 보인다.  
    Random rand = new Random(12);  
    for(int i = 0; i < 7; i++)  
        System.out.println(rand.nextInt(1000));  
}
```

다음 메소드 호출을 통해서 씨드 값을 수시로 바꿀 수 있다.

```
public void setSeed(long seed)
```

문자열의 토큰 구분

"PM:08:45"

이 문자열의 **구분자**가 :일 경우 **토큰**은 다음 세 가지

PM 08 45

위와 같이 토큰을 나누는 방법

```
StringTokenizer st = new StringTokenizer("PM:08:45", ":");  
    public boolean hasMoreTokens()      반환할 토큰이 남아 있는가?  
    public String nextToken()      다음 토큰을 반환
```

문자열의 토큰 구분의 예

```
public static void main(String[] args) {  
    StringTokenizer st1 = new StringTokenizer("PM:08:45", ":");
```

```
    while(st1.hasMoreTokens())  
        System.out.print(st1.nextToken() + ' ');  
    System.out.println();
```

둘 이상의 구분자! 공백도 구분자에 포함!

```
    StringTokenizer st2 = new StringTokenizer("12 + 36 - 8 / 2 = 44", "+-/= ");
```

```
    while(st2.hasMoreTokens())  
        System.out.print(st2.nextToken() + ' ');  
    System.out.println();  
}
```



```
C:\JavaStudy>java TokenizeString  
PM 08 45  
12 36 8 2 44  
C:\JavaStudy>
```

20-4. Arrays 클래스

Arrays 클래스의 배열 복사 메소드

```
public static int[] copyOf(int[] original, int newLength)
```

→ original에 전달된 배열을 첫 번째 요소부터 newLength의 길이만큼 복사

```
public static int[] copyOfRange(int[] original, int from, int to)
```

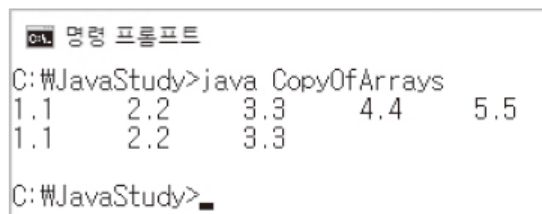
→ original에 전달된 배열을 인덱스 from부터 to 이전 요소까지 복사

```
public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
```

→ 배열 src의 srcPos에서 배열 dest의 destPos로 length 길이만큼 복사

copyOf 메소드 호출의 예

```
public static void main(String[] args) {  
    double[] arOrg = {1.1, 2.2, 3.3, 4.4, 5.5};  
  
    // 배열 전체를 복사  
    double[] arCpy1 = Arrays.copyOf(arOrg, arOrg.length);  
  
    // 세번째 요소까지만 복사  
    double[] arCpy2 = Arrays.copyOf(arOrg, 3);  
  
    for(double d : arCpy1)  
        System.out.print(d + "\t");  
    System.out.println();  
  
    for(double d : arCpy2)  
        System.out.print(d + "\t");  
    System.out.println();  
}
```



```
명령 프롬프트  
C:\JavaStudy>java CopyOfArrays  
1.1    2.2    3.3    4.4    5.5  
1.1    2.2    3.3  
C:\JavaStudy>
```


arraycopy 메소드 호출의 예

```
public static void main(String[] args) {  
    double[] org = {1.1, 2.2, 3.3, 4.4, 5.5};  
    double[] cpy = new double[3];  
  
    // 배열 org의 인덱스 1에서 배열 cpy 인덱스 0으로 세 개의 요소 복사  
    System.arraycopy(org, 1, cpy, 0, 3);  
  
    for(double d : cpy)  
        System.out.print(d + "\t");  
    System.out.println();  
}
```



```
명령 프롬프트  
C:\JavaStudy>java CopyOfSystem  
2.2 3.3 4.4  
C:\JavaStudy>
```

두 배열의 내용 비교

```
public static boolean equals(int[] a, int[] a2)
```

→ 매개변수 a와 a2로 전달된 배열의 내용을 비교하여 true 또는 false 반환

```
public static void main(String[] args) {  
    int[] ar1 = {1, 2, 3, 4, 5};  
    int[] ar2 = Arrays.copyOf(ar1, ar1.length);  
    System.out.println(Arrays.equals(ar1, ar2));  
}
```

명령 프롬프트

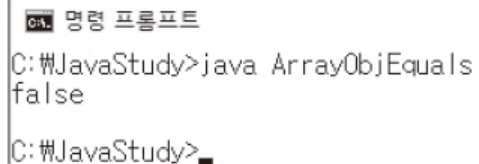
```
C:\JavaStudy>java ArrayEquals  
true
```

```
C:\JavaStudy>
```

인스턴스 저장 배열의 비교 예

```
class INum {  
    private int num;  
    public INum(int num) {  
        this.num = num;  
    }  
}  
  
class ArrayObjEquals {  
    public static void main(String[] args) {  
        INum[] ar1 = new INum[3];  
        INum[] ar2 = new INum[3];  
        ar1[0] = new INum(1); ar2[0] = new INum(1);  
        ar1[1] = new INum(2); ar2[1] = new INum(2);  
        ar1[2] = new INum(3); ar2[2] = new INum(3);  
        System.out.println(Arrays.equals(ar1, ar2));  
    }  
}
```

결과가 의미하는 바는? 어떤 식으로 비교?



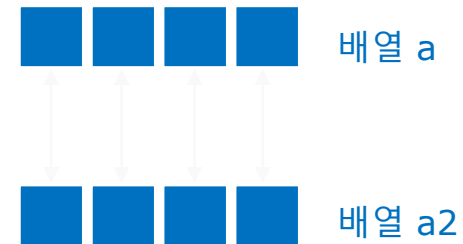
```
명령 프롬프트  
C:\JavaStudy>java ArrayObjEquals  
false  
C:\JavaStudy>
```

Arrays의 equals 메소드가 내용을 비교하는 방식

```
public static boolean equals(Object[] a, Object[] a2)
```

다음은 실제 Java의 Arrays.equals 메소드의 일부!

```
for (int i=0; i<length; i++) {  
    Object o1 = a[i];  
    Object o2 = a2[i];  
  
    if (!(o1==null ? o2==null : o1.equals(o2)))  
        return false;  
}
```



각 요소별로 Object 클래스의 equals 메소드로 비교

Object 클래스의 equals 메소드는?

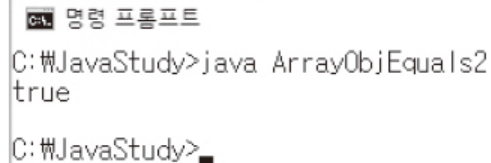
```
public boolean equals(Object obj) {  
    if(this == obj) // 두 인스턴스가 동일 인스턴스이면  
        return true;  
    else  
        return false;  
}    // 이렇듯 Object 클래스에 정의된 equals 메소드는 참조 값 비교를 한다.
```

따라서 Arrays 클래스의 equals 메소드가 두 배열의 내용 비교를 하도록 하려면
비교 대상의 equals 메소드를 내용 비교의 형태로 오버라이딩 해야 한다.

Object 클래스의 equals 메소드 오버라이딩 결과

```
class INum {  
    private int num;  
    public INum(int num) {  
        this.num = num;  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        if(this.num == ((INum)obj).num)  
            return true;  
        else  
            return false;  
    }  
}
```

```
public static void main(String[] args) {  
    INum[] ar1 = new INum[3];  
    INum[] ar2 = new INum[3];  
    ar1[0] = new INum(1); ar2[0] = new INum(1);  
    ar1[1] = new INum(2); ar2[1] = new INum(2);  
    ar1[2] = new INum(3); ar2[2] = new INum(3);  
    System.out.println(Arrays.equals(ar1, ar2));  
}
```



```
C:\JavaStudy>java ArrayObjEquals2  
true  
C:\JavaStudy>
```

배열의 정렬: Arrays 클래스의 sort 메소드

```
public static void sort(int[] a)
```

→ 매개변수 a로 전달된 배열을 오름차순(Ascending Numerical Order)으로 정렬

```
public static void main(String[] args) {  
    int[] ar1 = {1, 5, 3, 2, 4};  
    double[] ar2 = {3.3, 2.2, 5.5, 1.1, 4.4};  
    Arrays.sort(ar1);  
    Arrays.sort(ar2);  
  
    for(int n : ar1)  
        System.out.print(n + "\t");  
    System.out.println();  
  
    for(double d : ar2)  
        System.out.print(d + "\t");  
    System.out.println();  
}
```



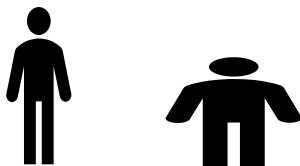
```
C:\WJavaStudy>java ArraySort  
1      2      3      4      5  
1.1    2.2    3.3    4.4    5.5  
C:\WJavaStudy>
```

오름차순 정렬이란?

값이 작은 순에서 큰 순으로 세워 나가는 정렬

ex) 1, 2, 3, 4, 5, 6, 7

수의 경우는 오름차순에 대한 기준이 명확하다. 그렇다면 다음 두 사람을 오름차순으로 정렬해서 줄을 세운다고 하면? 한 사람은 키가 크고 다른 한 사람은 무게가 많이 나간다.



이 때에는 기준이 필요하다 오름차순 순서상 크고 작음에 대한 기준이 필요하다.

그래서 클래스를 정의할 때 오름차순 순서상 크고 작음에 대한 기준을 정의해야 한다.

compareTo 메소드 정의 기준

```
interface Comparable
```

```
→ int compareTo(Object o)
```

인자로 전달된 o가 작다면 양의 정수 반환

인자로 전달된 o가 크다면 음의 정수 반환

인자로 전달된 o와 같다면 0을 반환

클래스에 정의하는 오름차순 기준

```
class Person implements Comparable {  
    private String name;  
    Private int age;  
  
    . . . .  
  
    @Override  
    public int compareTo(Object o) {  
        Person p = (Person)o;  
        if(this.age > p.age)  
            return 1;    // 인자로 전달된 o가 작다면 양의 정수 반환  
        else if(this.age < p.age)  
            return -1;    // 인자로 전달된 o가 크다면 음의 정수 반환  
        else  
            return 0;    // 인자로 전달된 o와 같다면 0을 반환  
    }  
    . . . .  
}
```

Comparable 인터페이스를 구현한다는 것은
오름차순 순서상 크고 작음에 대한 기준을 제공한다는 의미

나이가 어릴수록 오름차순 순서상 작은 것으로 정의됨

다음과 같이 구현도 가능!

```
public int compareTo(Object o) {  
    Person p = (Person)o;  
  
    if(this.age > p.age)  
        return 1;    // 인자로 전달된 o가 작다면 양의 정수 반환  
    else if(this.age < p.age)  
        return -1;    // 인자로 전달된 o가 크다면 음의 정수 반환  
    else  
        return 0;    // 인자로 전달된 o와 같다면 0을 반환  
}
```



대신

```
public int compareTo(Object o) {  
    Person p = (Person)o;  
    return this.age - p.age;  
}
```

배열에 저장된 인스턴스들의 정렬의 예

```
class Person implements Comparable {  
    private String name;  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    public int compareTo(Object o) {  
        Person p = (Person)o;  
        return this.age - p.age;  
    }  
    public String toString() {  
        return name + ": " + age;  
    }  
}
```

```
public static void main(String[] args) {  
    Person[] ar = new Person[3];  
    ar[0] = new Person("Lee", 29);  
    ar[1] = new Person("Goo", 15);  
    ar[2] = new Person("Soo", 37);  
  
    Arrays.sort(ar);  
    for(Person p : ar)  
        System.out.println(p);  
}
```

명령 프롬프트

```
C:\JavaStudy>java ArrayObjSort  
Goo: 15  
Lee: 29  
Soo: 37  
C:\JavaStudy>
```

배열의 탐색: 기본 자료형 값 대상

```
public static int binarySearch(int[] a, int key)
```

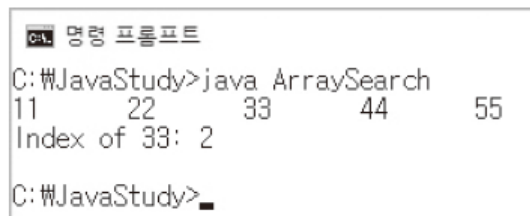
→ 배열 `a`에서 `key`를 찾아서 있으면 `key`의 인덱스 값, 없으면 `0`보다 작은 수 반환

`binarySearch`는 이진 탐색을 진행!

그리고 이진 탐색을 위해서는 탐색 이전에 데이터들이 오름차순으로 정렬되어 있어야 한다.

배열의 탐색: 기본 자료형 값 대상의 예

```
class ArraySearch {  
    public static void main(String[] args) {  
        int[] ar = {33, 55, 11, 44, 22};  
        Arrays.sort(ar);    // 탐색 이전에 정렬이 선행되어야 한다.  
  
        for(int n : ar)  
            System.out.print(n + "\t");  
        System.out.println();  
  
        int idx = Arrays.binarySearch(ar, 33); // 배열 ar에서 33을 찾아라.  
        System.out.println("Index of 33: " + idx);  
    }  
}
```



```
명령 프롬프트  
C:\JavaStudy>java ArraySearch  
11    22    33    44    55  
Index of 33: 2  
C:\JavaStudy>
```

배열의 탐색: 인스턴스 대상

```
public static int binarySearch(Object[] a, Object key)
```

마찬가지로 탐색 대상들은 오름차순으로 정렬되어 있어야 한다.

그리고 탐색 대상의 확인 여부는 `compareTo` 메소드의 호출 결과를 근거로 한다.

즉, 탐색 방식은 인스턴스의 내용 비교이다!

배열의 탐색: 인스턴스 대상의 예

```
class Person implements Comparable {
    private String name;
    private int age;
    . . .
    @Override
    public int compareTo(Object o) {
        Person p = (Person)o;
        return this.age - p.age;    // 나이가 같으면 0을 반환
    }
    . . .
}

public static void main(String[] args) {
    Person[] ar = new Person[3];
    ar[0] = new Person("Lee", 29);
    ar[1] = new Person("Goo", 15);
    ar[2] = new Person("Soo", 37);
    Arrays.sort(ar);    // 탐색에 앞서 정렬을 진행
    int idx = Arrays.binarySearch(ar, new Person("Who are you?", 37));
    System.out.println(ar[idx]);
}
```

명령 프롬프트

C:\JavaStudy>java ArrayObjSearch
Soo: 37

C:\JavaStudy>