Java

제 12 강

추상클래스 & 객체지향개념 II-3

java.lang 패키지

1. 상속
2. 오버라이딩
3. package와 import

→ 객체지향개념 II-1

4. 제어자
5. 다형성

→ 객체지향개념 II-2

6. 추상클래스
7. 인터페이스

→ 객체지향개념 II-3

6. 추상클래스(abstract class)

6.1 추상클래스(abstract class)란?

6.2 추상메서드(abstract method)란?

6.3 추상클래스의 작성

1. Object클래스
    1.1 Object클래스의 메서드
    1.2 equals(Object obj)
    1.3 hashCode()
    1.4 toString()
    1.5 clone()
    1.6 getClass()
2. String클래스
    2.1 String클래스의 특징
    2.2 빈 문자열(empty string)
    2.3 String클래스의 생성자와 메서드
    2.4 문자열과 기본형간의 변환
3. StringBuffer클래스
    3.1 StringBuffer클래스의 특징
    3.2 StringBuffer클래스의 생성자와 메서드
4. Math & wrapper클래스
    4.1 Math클래스
    4.2 wrapper클래스
    4.3 Number클래스

6. 추상클래스(abstract class)

6.1 추상클래스(abstract class)란?

- 클래스가 설계도라면 추상클래스는 '미완성 설계도'
- 추상메서드(미완성 메서드)를 포함하고 있는 클래스

  * 추상메서드 : 선언부만 있고 구현부(몸통, body)가 없는 메서드

```
abstract class Player {
     int currentPos;          // 현재 Play되고 있는 위치를 저장하기 위한 변수

     Player() {               // 추상클래스도 생성자가 있어야 한다.
          currentPos = 0;
     }

     abstract void play(int pos);      // 추상메서드
     abstract void stop();             // 추상메서드

     void play() {
          play(currentPos);            // 추상메서드를 사용할 수 있다.
     }
     ...
}
```

- 일반메서드가 추상메서드를 호출할 수 있다.(호출할 때 필요한 건 선언부)
- 완성된 설계도가 아니므로 인스턴스를 생성할 수 없다.
- 다른 클래스를 작성하는 데 도움을 줄 목적으로 작성된다.

6.2 추상메서드(abstract method)란?

- 선언부만 있고 구현부(몸통, body)가 없는 메서드

```
/* 주석을 통해 어떤 기능을 수행할 목적으로 작성하였는지 설명한다. */
abstract 리턴타입 메서드이름();

Ex)
/* 지정된 위치(pos)에서 재생을 시작하는 기능이 수행되도록 작성한다.*/
abstract void play(int pos);
```

- 꼭 필요하지만 자손마다 다르게 구현될 것으로 예상되는 경우에 사용
- 추상클래스를 상속받는 자손클래스에서 추상메서드의 구현부를 완성해야 한다.

```
abstract class Player {
    ...
     abstract void play(int pos);      // 추상메서드
     abstract void stop();             // 추상메서드
    ...
}

class AudioPlayer extends Player {
    void play(int pos) { /* 내용 생략 */ }
    void stop() { /* 내용 생략 */ }
}

abstract class AbstractPlayer extends Player {
    void play(int pos) { /* 내용 생략 */ }
}
```

6.3 추상클래스의 작성

- 여러 클래스에 공통적으로 사용될 수 있는 추상클래스를 바로 작성하거나 기존클래스의 공통 부분을 뽑아서 추상클래스를 만든다.

```
class Marine {    // 보병
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void stimPack()         { /* 스팀팩을 사용한다.*/}
}

class Tank {      // 탱크
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void changeMode()       { /* 공격모드를 변환한다. */}
}

class Dropship {  // 수송선
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void load()             { /* 선택된 대상을 태운다.*/ }
    void unload()           { /* 선택된 대상을 내린다.*/ }
}
```

```
abstract class Unit {
    int x, y;
    abstract void move(int x, int y);
    void stop() {      /* 현재 위치에 정지 */ }
}

class Marine extends Unit {  // 보병
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stimPack()         { /* 스팀팩을 사용한다.*/}
}

class Tank extends Unit {    // 탱크
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void changeMode()       { /* 공격모드를 변환한다. */}
}

class Dropship extends Unit { // 수송선
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void load()             { /* 선택된 대상을 태운다.*/ }
    void unload()           { /* 선택된 대상을 내린다.*/ }
}
```

```
Unit[] group = new Unit[4];
group[0] = new Marine();
group[1] = new Tank();
group[2] = new Marine();
group[3] = new Dropship();

for(int i=0;i< group.length;i++) {
    group[i].move(100, 200);
}
```

추상메서드가 호출되는 것이 아니라 각 자손들에 실제로 구현된 move(int x, int y)가 호출된다.

1. Object클래스

1.1 Object클래스의 메서드

- 모든 클래스의 최고 조상. 오직 11개의 메서드만을 가지고 있다.
- notify(), wait() 등은 쓰레드와 관련된 메서드이다.
- equals(), hashCode(), toString()은 적절히 오버라이딩해야 한다.

Object클래스의 메서드	설 명
protected Object clone()	객체 자신의 복사본을 반환한다.
public boolean equals(Object obj)	객체 자신과 객체 obj가 같은 객체인지 알려준다.(같으면 true)
protected void finalize()	객체가 소멸될 때 가비지 컬렉터에 의해 자동적으로 호출된다. 이 때 수행되어야하는 코드가 있는 경우에만 오버라이딩한다.
public Class getClass()	객체 자신의 클래스 정보를 담고 있는 Class인스턴스를 반환한다.
public int hashCode()	객체 자신의 해시코드를 반환한다.
public String toString()	객체 자신의 정보를 문자열로 반환한다.
public void notify()	객체 자신을 사용하려고 기다리는 쓰레드를 하나만 깨운다.
public void notifyAll()	객체 자신을 사용하려고 기다리는 모든 쓰레드를 깨운다.
public void wait()	다른 쓰레드가 notify()나 notifyAll()을 호출할 때까지 현재 쓰레드를 무한히 또는 지정된 시간(timeout, nanos)동안 기다리게 한다. (timeout은 천 분의 1초, nanos는 10^9분의 1초)
public void wait(long timeout)	
public void wait(long timeout, int nanos)	

1.2 equals(Object obj)

- 객체 자신과 주어진 객체(obj)를 비교한다. 같으면 true, 다르면 false.
- Object클래스에 정의된 equals()는 참조변수 값(객체의 주소)을 비교한다.

```
public boolean equals(Object obj) {
    return (this==obj);
}
```

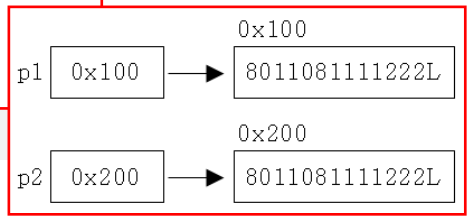
- equals()를 오버라이딩해서 인스턴스변수의 값을 비교하도록 바꾼다.

```
class Person {
   long id;

   public boolean equals(Object obj) {
       if(obj!=null && obj instanceof Person) {
           return id ==((Person)obj).id;
       } else {
           return false;
       }
   }

   Person(long id) {
       this.id = id;
   }
}
```

obj가 Object타입이므로 id값을 참조하기 위해서는 Person타입으로 형변환이 필요하다.

타입이 Person이 아니면 값을 비교할 필요도 없다.

```
Person p1 = new Person(8011081111222L);
Person p2 = new Person(8011081111222L);

System.out.println(p1==p2);
System.out.println(p1.equals(p2));
```



1.3 hashCode()

- 객체의 해시코드(int타입의 정수)를 반환하는 메서드(해시함수)
  다량의 데이터를 저장&검색하는 해싱기법에 사용된다.

- Object클래스의 hashCode()는 객체의 내부주소를 반환한다.

```
public class Object {
    ...
    public native int hashCode();
```

- equals()를 오버라이딩하면, hashCode()도 같이 오버라이딩 해야한다.
  equals()의 결과가 true인 두 객체의 hash code는 같아야하기 때문

```
String str1 = new String("abc");
String str2 = new String("abc");
System.out.println(str1.equals(str2)); // true
System.out.println(str1.hashCode());   // 96354
System.out.println(str2.hashCode());   // 96354
```

- System.identityHashCode(Object obj)는 Object클래스의 hashCode()와 동일한 결과를 반환한다.

```
System.out.println(System.identityHashCode(str1));   // 3526198
System.out.println(System.identityHashCode(str2));   // 7699183
```

1.4 toString()

- 객체의 정보를 문자열(String)로 제공할 목적으로 정의된 메서드

```
public String toString() {  // Object클래스의 toString()
     return getClass().getName() + "@"
            + Integer.toHexString(hashCode());
}
```

오버라이딩

```
class Card {
   String kind;
   int number;

   Card() {
       this("SPADE", 1);
   }
   Card(String kind, int number) {
       this.kind = kind;
       this.number = number;
   }
}

class CardToString
{
   public static void main(String[] args)
   {
       Card c1 = new Card();
       Card c2 = new Card();

       System.out.println(c1.toString());
       System.out.println(c2.toString());
   }
}
```

```
public String toString() {
    // Card인스턴스의 kind와 number를 문자열로 반환한다.
    return "kind : " + kind + ", number : " + number;
}
```

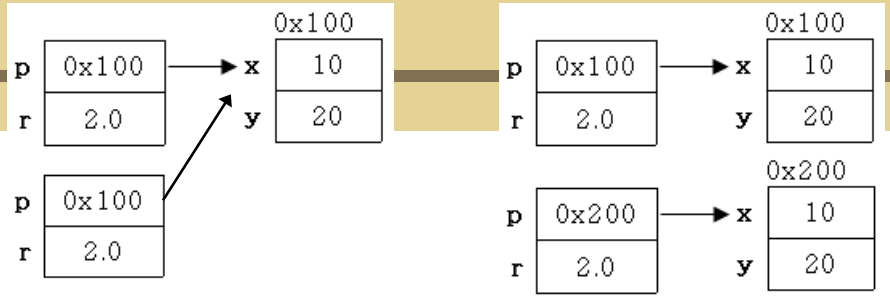
[실행결과]

```
Card@47e553
Card@20c10f
```

→

[실행결과]

```
kind : SPADE, number : 1
kind : SPADE, number : 1
```



1.5 clone()

- 객체 자신을 복제(clone)해서 새로운 객체를 생성하는 메서드
- Cloneable인터페이스를 구현한 클래스의 인스턴스만 복제할 수 있다.
- Object클래스에 정의된 clone()은 인스턴스변수의 값만을 복제한다.
- 인스턴스변수가 참조형일 때, 참조하는 객체도 복제되게 오버라이딩해야함.

```
class Point implements Cloneable {
    int x;
    int y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String toString()
        return "x="+x +", y="
    }

    public Object clone() {
        Object obj=null;
        try {
            obj = super.clone
        } catch(CloneNotSuppo
        return obj;
    }
}
```

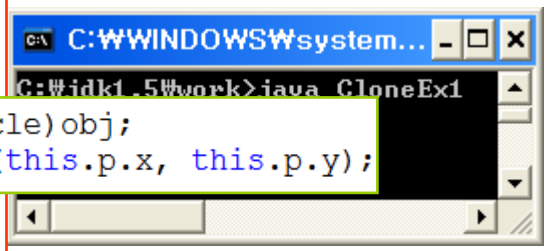
```
class Circle implements Cloneable {
    Point  p;   // 원점
    double r;   // 반지름

    Circle(Point p, double r) {
        this.p = p;
        this.r = r;
    }

    public Object clone() {
        Object obj = null;
        try {
            obj = super.clone();
        } catch(CloneNotSupportedException e) {}

        return obj;
    }

    public String toString() {
        return p.toString() + ", r="+r;
    }
}
```

```
CloneEx1 {
ic static void main(String[] args){
    Circle c1 = new Circle(new Point(10,20), 2.0);
    Circle c2 = (Circle)c1.clone();
System.out.println(copy);
```

```
    Circle c = (Circle)obj;
    c.p = new Point(this.p.x, this.p.y);
```

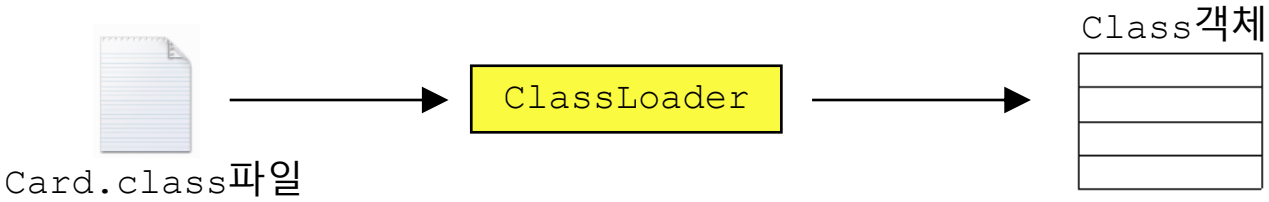
x 3
y 5

x 3
y 5



1.6 getClass()

- 자신이 속한 클래스의 Class객체를 반환하는 메서드
- Class객체는 클래스의 모든 정보를 담고 있으며, 클래스당 단 1개만 존재
  클래스파일(*.class)이 메모리에 로드될때 생성된다.



- Class객체를 얻는 여러가지 방법

```
Card c = new Card();
Class cObj = c.getClass();
```

```
Card c2 = new Card();
Card c2 = (Card)cObj.newInstance();
```

```
Class cObj = Card.class;
String className = cObj.getName();
```

```
String className = Card.class.getName();
```

```
Class cObj = Class.forName("Card");
```

2. String클래스

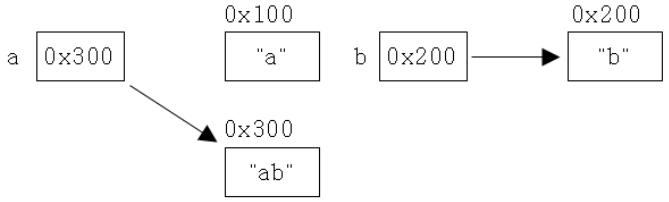
2.1 String클래스의 특징

- 문자형 배열(char[])과 그에 관련된 메서드들이 정의되어 있다.

```
public final class String implements java.io.Serializable, Comparable {
    /** The value is used for character storage. */
    private char[] value;
    ...
```

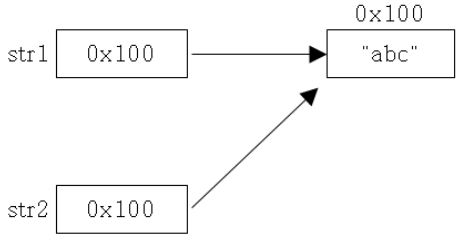
- String인스턴스의 내용은 바꿀 수 없다.(immutable)

```
String a = "a";
String b = "b";
String a = a + b;
```

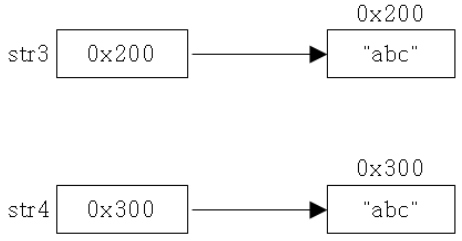


- String str = “abc”;와 String str = new String(“abc”);의 비교

```
String str1 = "abc";
String str2 = "abc";
String str3 = new String("abc");
String str4 = new String("abc");
System.out.println(str1==str2);
System.out.println(str3==str4);
System.out.println(str1.equals(str2));
System.out.println(str3.equals(str4));
```



str1 == str2
0x100 == 0x100
true



str3 == str4
0x200 == 0x300
false

2.2 빈 문자열(“”, empty string)

- 내용이 없는 문자열. 크기가 0인 char형 배열을 저장하는 문자열
- 크기가 0인 배열을 생성하는 것은 어느 타입이나 가능

```
char[] cArr = new char[0];   // 크기가 0인 char배열
int[]  iArr = {};            // 크기가 0인
```

```
public static void main(String[] args)
```

```
if(args==null||args.length!=2){
    System.out.println("매개변수는 2개 필요");
    System.exit(0);
}
```

- String str=“”;은 가능해도 char c = ‘’;는 불가능
- String은 참조형의 기본값인 null 보다 빈 문자열로 초기화하고 char형은 기본값인 ‘₩u0000’보다 공백으로 초기화하자.

```
String s = null;
char c = '\u0000';
```

→

```
String s = ""; // 빈 문자열로 초기화
char c = ' ';  // 공백으로 초기화
```

```
String str1 = "";
String str2 = "";
String str3 = "";
```

```
String str4 = new String("");
String str5 = new String("");
String str6 = new String("");
```

2.3 String클래스의 생성자와 메서드(1/3)

메서드 / 설명	예 제	결 과
String(String s) 주어진 문자열(s)을 갖는 String인스턴스를 생성한다.	`String s = new String("Hello");`	`s = "Hello"`
String(char[] value) 주어진 문자열(value)을 갖는 String인스턴스 를 생성한다.	`char[] c = {'H','e','l','l','o'}` `String s = new String(c);`	`s = "Hello"`

String(StringBuffer buf)
StringBuffer인스턴스가 갖고 있는 문자열과 같은 내용의 String인스턴스를 생성한다.
	`StringBuffer sb =`
	`new StringBuffer("Hello");`
	`String s = new String(sb);`
		`s = "Hello"`

char charAt(int index)
지정된 위치(index)에 있는 문자를 알려준다. (index는 0부터 시작)
	`String s = "Hello";`
	`String n = "0123456";`
	`char c = s.charAt(1);`
	`char c2 = n.charAt(1);`
		`c = 'e'`
		`c2 = '1'`

String concat(String str) 문자열(str)을 뒤에 덧붙인다.	`String s = "Hello";` `String s2 = s.concat(" World");`	`s2 = "Hello World"`
boolean contains(CharSequence s) 지정된 문자열(s)이 포함되었는지 검사한다.	`String s = "abcedfg";` `boolean b = s.contains("bc");`	`b = true`
boolean endsWith(String suffix) 지정된 문자열(suffix)로 끝나는지 검사한다.	`("txt");`	`b = true`
boolean equals(Object obj) 매개변수로 받은 문자열(obj)과 String인스턴 스의 문자열 아니 거나 문자열이 다르면 false를 반환한다.	`lo");` `llo");`	`b = true` `b2 = false`
boolean equalsIgnoreCase(String str) 문자열과 String인스턴스의 문자열을 대소문자 구분없이	`eCase("HELLO");` `boolean b2 = s.equalsIgnoreCase("heLLo");`	`b = true` `b2 = true`

int indexOf(int ch)
주어진 문자(ch)가 문자열에 존재하는지 확인하여 위치(index)를 알려준다. 못 찾으면 -1을 반환한다.(index는 0부터 시작)
	`String s = "Hello";`
	`int idx1 = s.indexOf('o');`
	`int idx2 = s.indexOf('k');`
		`idx1 = 4`
		`idx2 = -1`

H	e	l	l	o

java.lang
Interface CharSequence

All Known Subinterfaces:
Name

All Known Implementing Classes:
CharBuffer, Segment, String, StringBuffer, StringBuilder

2.3 String클래스의 생성자와 메서드(2/3)

int indexOf(String str) 주어진 문자열이 존재하는지 확인하여 그 위치(index)를 알려준다. 없으면 -1을 반환한다. (index는 0부터 시작)	`String s = "ABCDEFG";` `int idx = s.indexOf("CD");`	`idx = 2`

String intern()
문자열을 constant pool에 등록한다. 이미 constant pool에 같은 내용의 문자열이 있을 경우 그 문자열의 주소값을 반환한다.
	`String s = new String("abc");`
	`String s2 = new String("abc");`
	`boolean b = (s==s2);`
	`boolean b2 = s.equals(s2);`
	`boolean b3 = (s.intern()==s2.intern());`
		`b = false`
		`b2 = true`
		`b3 = true`

int lastIndexOf(int ch)
지정된 문자 또는 문자코드를 문자열의 오른쪽 끝에서부터 찾아서 위치(index)를 알려준다. 못 찾으면 -1을 반환한다.
	`String s = "java.lang.Object";`
	`int idx1 = s.lastIndexOf('.');`
	`int idx2 = s.indexOf('.');`
		`idx1 = 9`
		`idx2 = 4`

int lastIndexOf(String str)
지정된 문자열을 인스턴스의 문자열 끝에서 부터 찾아서 위치(index)를 알려준다. 못 찾으면 -1을 반환한다.
	`String s = "java.lang.java";`
	`int idx1 = s.lastIndexOf("java");`
	`int idx2 = s.indexOf("java");`
		`idx1 = 10`
		`idx2 = 0`

int length() 문자열의 길이를 알려준다.	`String s = "Hello";` `int length = s.length();`	`length = 5`
String replace(char old, char nw) 문자열 중의 문자(old)를 새로운 문자(nw)로 바꾼 문자열을 반환한다.	`String s = "Hello";` `String s1 = s.replace('H', 'C');`	`s1 = "Cello"`
String replace(CharSequence old, CharSequence nw) 문자열 중의 문자열(old)을 새로운 문자열(nw)로 모두 바꾼 문자열을 반환한다.	`String s = "Hellollo";` `String s1 = s.replace("ll","LL");`	`s1 = "HeLLoLLo"`
String replaceAll(String regex, String replacement) 문자열 중에서 지정된 문자열(regex)과 일치하는 것을 새로운 문자열(replacement)로 모두 변경한다.	`String ab = "AABBAABB";` `String r = ab.replaceAll("BB","bb");`	`r = "AAbbAAbb"`
String replaceFirst(String regex, String replacement) 문자열 중에서 지정된 문자열(regex)과 일치 하는 것 중, 첫 번째 것만 새로운 문자열(replacement)로 변경한다.	`String ab = "AABBAABB";` `String r = ab.replaceFirst("BB","bb");`	`r = "AAbbAABB"`

2.3 String클래스의 생성자와 메서드(3/3)

String[] split(String regex) 문자열을 지정된 분리자(regex)로 나누어 문자열 배열에 담아 반환한다.	String animals = "dog,cat,bear"; String[] arr = animals.split(","); 	arr[0] = "dog" arr[1] = "cat" arr[2] = "bear"
String[] split(String regex, int limit) 문자열을 지정된 분리자(regex)로 나누어 문자열배열에 담아 반환한다. 단, 문자열 전체를 지정된 수(limit)로 자른다.	String animals = "dog,cat,bear"; String[] arr = animals.split(",",2); 	arr[0] = "dog" arr[1] = "cat,bear"

boolean startsWith(String prefix)
주어진 문자열(prefix)로 시작하는지 검사한다.
	String s = "java.lang.Object";
	boolean b =s.startsWith("java");
	boolean b2=s.startsWith("lang");
	
		b = true
		b2 = false

String substring(int begin) **String substring(int begin, int end)** 주어진 시작위치(begin)부터 끝 위치(end) 범위에 포함된 문자열을 얻는다. 이 때, 시작위치의 문자는 범위에 포함되지만, 끝 위치의 문자는 포함되지 않는다.	String s = "java.lang.Object"; Stirng c = s.substring(10); String p = s.substring(5,9); 	c = "Object" p = "lang"
String toLowerCase() String인스턴스에 저장되어있는 모든 문자열을 소문자로 변환하여 반환한다.	String s = "Hello"; String s1 = s.toLowerCase(); 	s1 = "hello"
String toString() String인스턴스에 저장되어 있는 문자열을 반환한다.	String s = "Hello"; String s1 = s.toString(); 	s1 = "Hello"
String toUpperCase() String인스턴스에 저장되어있는 모든 문자열을 대문자로 변환하여 반환한다.	String s = "Hello"; String s1 = s.toUpperCase(); 	s1 = "HELLO"
String trim() 문자열의 왼쪽 끝과 오른쪽 끝에 있는 공백을 없앤 결과를 반환한다. 이 때 문자열 중간에 있는 공백은 제거되지 않는다.	String s = " Hello World "; String s1 = s.trim(); 	s1 = "Hello World"
static String valueOf(boolean b) **static String valueOf(char c)** **static String valueOf(int i)** **static String valueOf(long l)** **static String valueOf(float f)** **static String valueOf(double d)** **static String valueOf(Object o)** 지정된 값을 문자열로 변환하여 반환한다. 참조변수의 경우, toString()을 호출한 결과를 반환한다.	String b = String.valueOf(true); String c = String.valueOf('a'); String i = String.valueOf(100); String l = String.valueOf(100L); String f = String.valueOf(10f); String d = String.valueOf(10.0); java.util.Date dd =new java.util.Date(); String date = String.valueOf(dd); 	b = "true" c = "a" i = "100" l = "100" f = "10.0" d = "10.0" date = "Sun Jan 27 21:26:29 KST 2008"

2.4 문자열과 기본형간의 변환

- 기본형 값을 문자열로 바꾸는 두 가지 방법(방법2가 더 빠름)

```
int i = 100;
String str1 = i + "";              // 100을 "100"으로 변환하는 방법1
String str2 = String.valueOf(i); // 100을 "100"으로 변환하는 방법2
```

- 문자열을 기본형 값으로 변환하는 방법

```
int i = Integer.parseInt("100"); // "100"을 100으로 변환하는 방법1
int i2 = Integer.valueOf("100"); // "100"을 100으로 변환하는 방법2(JDK1.5이후)
char c = "A".charAt(0);   // 문자열 "A"를 문자 'A'로 변환하는 방법
```

기본형 → 문자열	문자열 → 기본형
String valueOf(boolean b) String valueOf(char c) String valueOf(int i) String valueOf(long l) String valueOf(float f) String valueOf(double d)	boolean Boolean.getBoolean(String s) byte Byte.parseByte(String s) short Short.parseShort(String s) int Integer.parseInt(String s) long Long.parseLong(String s) float Float.parseFloat(String s) double Double.parseDouble(String s)

[예제9-18]/ch9/StringCount.java

```
public class StringCount
{
    private int count;
    private String source ="";

    public StringCount(String source) {
        this.source = source;
    }

    public int stringCount(String s) {
        return stringCount(s, 0);
    }

    public int stringCount(String s, int pos) {
        int index = 0;
        if (s == null || s.length() == 0)
            return 0;
        if ( (index = source.indexOf(s, pos))!=-1) {
            count++;
            stringCount(s, index +s.length());
        }
        return count;
    }
```

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
source	a	a	b	b	c	c	A	A	B	B	C	C	a	a

```
while((index = source.indexOf(s, pos))!=-1) {
    count++;
    pos = index +s.length();
}
```

```
public static void main(String[] args)
{
    String str = "aabbccAABBCCaa";
    System.out.println(str);
    StringCount sc = new StringCount(str);
    System.out.println("aa를 " + sc.stringCount("aa") +"개 찾았습니다.");
}
```

3. StringBuffer클래스

3.1 StringBuffer클래스의 특징

- String처럼 문자형 배열(char[])을 내부적으로 가지고 있다.

```
public final class StringBuffer implements java.io.Serializable
{
   private char[] value;
   ...
```

- 그러나, String클래스와 달리 내용을 변경할 수 있다.(mutable)

```
StringBuffer sb = new StringBuffer("abc");
sb.append("123");
```

- 인스턴스를 생성할 때 버퍼(배열)의 크기를 충분히 지정해주는 것이 좋다.
  (버퍼가 작으면 성능 저하 - 작업 중에 더 큰 배열의 생성이 필요)

```
public StringBuffer(int length) {
   value = new char[length];
   shared = false;
}
```

```
public StringBuffer(String str) {
   this(str.length() + 16);
   append(str);
}
```

- String클래스와 달리 equals()를 오버라이딩하지 않았다.

```
StringBuffer sb = new StringBuffer("abc");
StringBuffer sb2 = new StringBuffer("abc");
System.out.println(sb==sb2);        // false
System.out.println(sb.equals(sb2)); // false
```

```
String s  = sb.toString();
String s2 = sb2.toString();
System.out.println(s.equals(s2)); // true
```

3.2 StringBuffer클래스의 생성자와 메서드(1/2)

메서드 / 설명	예 제 / 결 과
StringBuffer()	StringBuffer sb = new StringBuffer();
16문자를 담을 수 있는 버퍼를 가진 StringBuffer 인스턴스를 생성한다.	sb = ""
StringBuffer(int length)	StringBuffer sb = new StringBuffer(10);
지정된 개수의 문자를 담을 수 있는 버퍼를 가진 StringBuffer인스턴스를 생성한다.	sb = ""
StringBuffer(String str)	StringBuffer sb = new StringBuffer("Hi");
지정된 문자열 값(str)을 갖는 StringBuffer 인스턴스를 생성한다.	sb = "Hi"

StringBuffer append(boolean b) **StringBuffer append(char c)**
StringBuffer append(char[] str) **StringBuffer append(double d)**
StringBuffer append(float f) **StringBuffer append(int i)**
StringBuffer append(long l) **StringBuffer append(Object obj)**
StringBuffer append(String str)
	StringBuffer sb = new StringBuffer("abc");
	StringBuffer sb2 = sb.append(true);
	sb.append('d').append(10.0f);
	StringBuffer sb3 =
	sb.append("ABC").append(123);

매개변수로 입력된 값을 문자열로 변환하여 StringBuffer인스턴스가 저장하고 있는 문자열의 뒤에 덧붙인다.	sb = "abctrued10.0ABC123" sb2 = "abctrued10.0ABC123" sb3 = "abctrued10.0ABC123"

int capacity()	StringBuffer sb = new StringBuffer(100);
	sb.append("abcd");
	int bufferSize = sb.capacity();
	int stringSize = sb.length();

StringBuffer인스턴스의 버퍼크기를 알려준다. length()는 버퍼에 담긴 문자열의 크기를 알려준다.	bufferSize = 100 stringSize = 4 (sb에 담긴 문자열이 "abcd"이므로)
char charAt(int index)	StringBuffer sb = new StringBuffer("abc"); char c = sb.charAt(2);
지정된 위치(index)에 있는 문자를 반환한다.	c='c'
StringBuffer delete(int start, int end)	StringBuffer sb = new StringBuffer("0123456"); StringBuffer sb2 = sb.delete(3,6);
시작위치(start)부터 끝 위치(end) 사이에 있는 문자를 제거한다. 단, 끝 위치의 문자는 제외.	sb = "0126" sb2 = "0126"
StringBuffer deleteCharAt(int index)	StringBuffer sb = new StringBuffer("0123456"); sb.deleteCharAt(3);
지정된 위치(index)의 문자를 제거한다.	sb = "012456"

3.2 StringBuffer클래스의 생성자와 메서드(2/2)

메서드 / 설명	예 제 / 결 과 StringBuffer sb = new StringBuffer("0123456");
StringBuffer insert(int pos, boolean b) StringBuffer insert(int pos, char c) **StringBuffer insert(int pos, char[] str) StringBuffer insert(int pos, int i)** **StringBuffer insert(int pos, float f) StringBuffer insert(int pos, long l)** **StringBuffer insert(int pos, double d) StringBuffer insert(int pos, String str)** **StringBuffer insert(int pos, Object obj)**	sb.insert(4,'.');
두 번째 매개변수로 받은 값을 문자열로 변환하여 지정된 위치(pos)에 추가한다. pos는 0부터 시작	sb = "0123.456"
int length()	int length = sb.length();
StringBuffer인스턴스에 저장되어 있는 문자열의 길이를 반환한다.	length = 7
StringBuffer replace(int start, int end, String str)	sb.replace(3, 6, "AB");
지정된 범위(start~end)의 문자들을 주어진 문자열로 바꾼다. end위치의 문자는 범위에 포함안됨.	sb = "012AB6" "345"를 "AB"로 바꿨다.
StringBuffer reverse()	sb.reverse();
StringBuffer인스턴스에 저장되어 있는 문자열의 순서를 거꾸로 나열한다.	sb = "6543210"
void setCharAt(int index, char ch)	sb.setCharAt(5, 'o');
지정된 위치의 문자를 주어진 문자(ch)로 바꾼다.	sb = "01234o6"

void setLength(int newLength)	sb.setLength(5);
	StringBuffer sb2=new StringBuffer("0123456");
	sb2.setLength(10);
	String str = sb2.toString().trim();

지정된 크기로 문자열의 길이를 변경한다. 크기를 늘리는 경우에 나머지 빈 공간을 널문자 '₩u0000'로 채운다.	sb = "01234" sb2 = "0123456 " str = "0123456"
String toString()	String str = sb.toString();
StringBuffer인스턴스의 문자열을 String으로 반환한다.	str = "0123456"
String substring(int start) **String substring(int start, int end)**	String str = sb.substring(3); String str2 = sb.substring(3, 5);
지정된 범위 내의 문자열을 String으로 뽑아서 반환 한다. 시작위치(start)만 지정하면 시작위치부터 문자열 끝까지 뽑아서 반환한다.	str = "3456" str2 = "34"

4. Math & wrapper클래스

4.1 Math클래스

- 수학계산에 유용한 메서드로 구

```
int x = 50;
x = Math.min(Math.max(10, x), 100);     // 10 <= x <= 100

x = Math.min(Math.max(10, -1), 100);    // x = 10
x = Math.min(Math.max(10, 200), 100);   // x = 100
```

메서드 / 설명	예 제	결 과
static int abs(int f) static float abs(float f) **static long abs(long f) static double abs(double a)** 주어진 값의 절대값을 반환한다.	int i = Math.abs(-10); double d = Math.abs(-10.0);	i=10 d=10.0

static double ceil(double a)
주어진 값을 올림하여 반환한다.
	double d = Math.ceil(10.1);
	double d2 = Math.ceil(-10.1);
	double d3 = Math.ceil(10.0000015);
		d = 11.0
		d2 = -10.0
		d3 = 11.0

static double floor(double a) 주어진 값을 버림하여 반환한다.	double d = Math.floor(10.8); double d2 = Math.floor(-10.8);	d = 10.0 d2=-11.0
static int max(int a, int b) static float max(float a, float b) **static long max(long a, long b) static double max(double a,double b)** 주어진 두 값을 비교하여 큰 쪽을 반환한다.	double d = Math.max(9.5, 9.50001); int i = Math.max(0, -1);	d = 9.50001 i = 0
static int min(int a, int b) static float min(float a, float b) **static long min(long a, long b) static double min(double a,double b)** 주어진 두 값을 비교하여 작은 쪽을 반환한다.	double d = Max.min(9.5, 9.50001); int i = Math.min(0, -1);	d = 9.5 i = -1
static double random() 0.0~1.0범위의 임의의 double값을 반환한다. 0.0은 범위에 포함되지만 1.0은 포함안됨	double d = Math.random(); int i = (int)(Math.random()*10)+1	d = 0.0~1.0의 실수 i = 1~10의 정수

static double rint(double a)
주어진 double값과 가장 가까운 정수값을 double형으로 반환한다.
	double d = Math.rint(5.55);
	double d2 = Math.rint(5.11);
	double d3 = Math.rint(-5.55);
	double d4 = Math.rint(-5.11);
		d = 6.0
		d2 = 5.0
		d3 = -6.0
		d4 = -5.0

static long round(double a)
static long round(float a)
소수점 첫째자리에서 반올림한 정수값(long) 을 반환한다.
	long l = Math.round(5.55);
	long l2 = Math.round(5.11);
	long l3 = Math.round(-5.55);
	long l4 = Math.round(-5.11);
	double d = 90.7552;
	double d2 = Math.round(d*100)/100.0;
		l = 6
		l2 = 5
		l3 = -6
		l4 = -5
		d = 90.7552
		d2 = 90.76

4.2 wrapper클래스

- 기본형을 클래스로 정의한 것. 기본형 값도 객체로 다뤄져야 할 때가 있다.

기본형	래퍼클래스	생성자	활용예
boolean	Boolean	Boolean(boolean value) Boolean(String s)	Boolean b = new Boolean(true); Boolean b2 = new Boolean("true");
char	**Character**	Character(char value)	Character c = new Character('a');
byte	Byte	Byte(byte value) Byte(String s)	Byte b = new Byte(10); Byte b2 = new Byte("10");
short	Short	Short(short value) Short(String s)	Short s = new Short(10); Short s2 = new Short("10");
int	**Integer**	Integer(int value) Integer(String s)	Integer i = new Integer(100); Integer i2 = new Integer("100");
long	Long	Long(long value) Long(String s)	Long l = new Long(100); Long l2 = new Long("100");
float	Float	Float(double value) Float(float value) Float(String s)	Float f = new Float(1.0); Float f2 = new Float(1.0f); Float f3 = new Float("1.0f");
double	Double	Double(double value) Double(String s)	Double d = new Double(1.0); Double d2 = new Double("1.0");

- 내부적으로 기본형(primitive type) 변수를 가지고 있다.

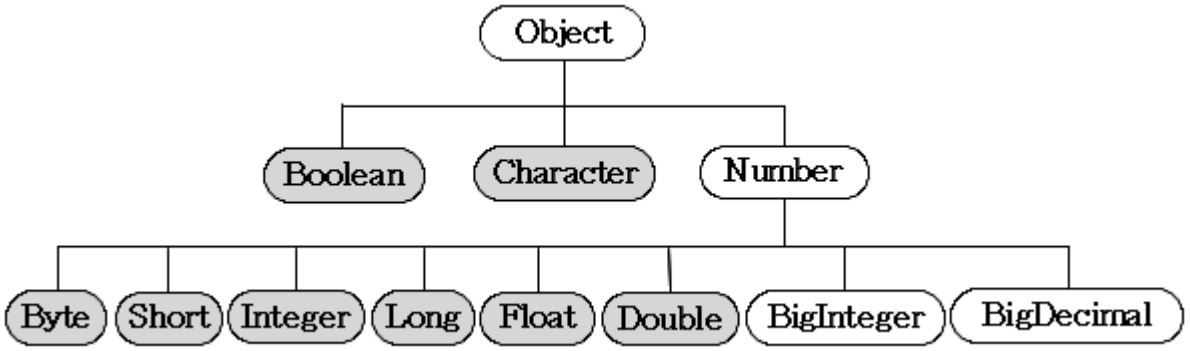
```
public final class Integer extends Number implements Comparable {
   ...
   private int value;
```

- 값을 비교하도록 equals()가 오버라이딩되어 있다.

```
Integer i  = new Integer(100);
Integer i2 = new Integer(100);
System.out.println(i==i2);          // false
System.out.println(i.equals(i2));   // true
```

4.3 Number클래스

- 숫자를 멤버변수로 갖는 클래스의 조상(추상클래스)

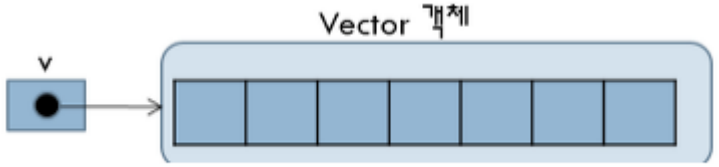



```
public abstract class Number implements java.io.Serializable {
    public abstract int       intValue();
    public abstract long      longValue();
    public abstract float     floatValue();
    public abstract double doubleValue();

    public byte byteValue() {
        return (byte)intValue();
    }
    public short shortValue() {
        return (short)intValue();
    }
}
```

5.1 Vector 소개

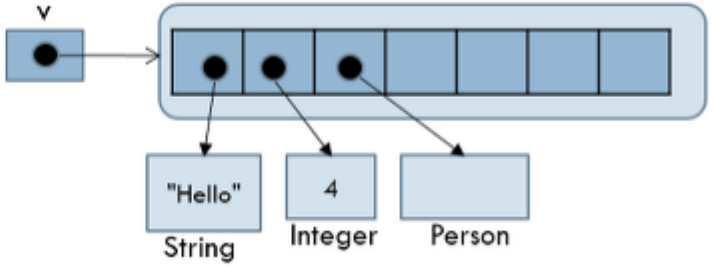
▶ Vector 생성

```
Vector v = new Vector();
```



▶ Vector 요소삽입

```
v.add("Hello");
v.add(new Integer(4));
v.add(new Person());
```



5.1 Vector 소개

▶ Vector 요소 알아내기

get() 메소드를 사용하고 해당 객체타입으로 형변환

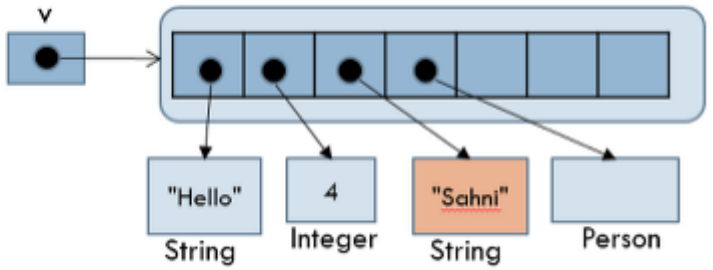
▶ Vector 크기 알아내기

size() 메소드를 사용한다

5.1 Vector 소개

▶ Vector 요소객체 중간 삽입

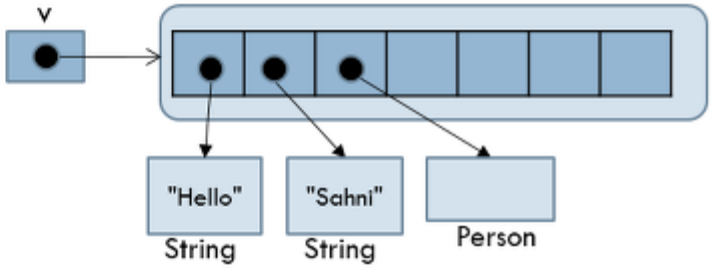
```
v.add(2, "Sahni");
```



▶ Vector 요소삭제

remove() removeAllElement() 메소드 사용

```
v.remove(1);
```



감사합니다.