chapter9 java.lang패키지와 유용한 클래스

ch9-1 Object클래스

-모든 클래스의 최고조상. 11개의 메서드만 가지고 있다.
-notify(), wait()등은 쓰레드와 관련된 메서드이다.

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. 이때 수행되어야하는 코드가 있을 때 오버라이딩한다. (거의 사용안함)
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() public void wait(long timeout) public void wait(long timeout, int nanos)	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)

ch9-2equals(Object obj)

-객체 자신(this)과 주어진 객체(obj)를 비교한다.
같으면 true, 다르면 false
-Object클래스의 equals()는 객체의 **주소**를 비교
서로 다른 두 객체는 값이 같아도 주소가 다르다.

ch9-3 equals(Object obj)의 오버라이딩

-인스턴스 변수(iv)의 값을 비교하도록 equals()를 오버라이딩 해야한다.
주소비교 -> 값비교 하도록!

```
class Person {
   long id;

   public boolean equals(Object obj) {
      if(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.

cv(공유값), iv(개별값) 이기 때문,,

ch9-4 객체의 지문 hashCode()

-객체의 해시코드를 반환하는 메서드
해시코드란? 해싱알고리즘에서 사용하는 정수값으로
Object클래스의 hashCode()는 객체의 **주소**를 int로 변환해서 반환한다.

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

native()는 os의 메서드

equals()를 오버라이딩 하면, hashCode()도 오버라이딩
★ equals()의 결과가 true인 두 객체의 해시코드는 같아야 하기때문

ch9-5~6 toString(), toString()의 오버라이딩

-toString():객체를 문자열로 변환하기 위한 메서드
Object클래스의 toString은 유용하지 못함
>오버라이딩하여 객체(iv값)를 문자열로 변환하여 나타냄
(객체==iv집합이므로 객체를 문자열로 반환한다는 것은 iv의 값을 문자열로 변환한다는 것과 같다)

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

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

   Card() {
      this("SPADE", 1);
   }

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

class Ex9_4 {
   public static void main(String[] args) {
      System.out.println(new Card().toString());
      System.out.println(new Card().toString());
   }
}
```

```
public String toString() {
   return "kind : " + kind + ", number : " + number;
}
```

ch9-7 String 클래스(문자열을 다루기 위한 클래스)

-String클래스는 데이터(char[])+메서드(문자열 관련)로 구성
문자배열 문자열을 다루기 위한 메서드
-내용을 변경할 수 없는 불변 클래스

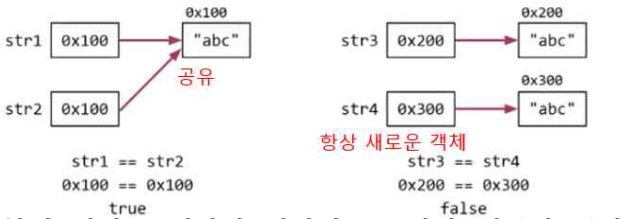
```
String a = "a";
String b = "b";
a = a + b;
```

a에 값이 바뀌는 것처럼 보이나 "ab"라는 문자열이 새로운 저장 공간에 생성 되고 그 저장소의 주소값을 a가 가리킨다.

-덧셈연산자를 이용한 문자열 결합은 성능↓
-문자열의 결합,변경이 잦다면, StringBuffer를 사용

ch9-8 문자열의 비교

```
String str1 = "abc";    // 문자열 리터럴 "abc"의 주소가 str1에 저장됨
String str2 = "abc";    // 문자열 리터럴 "abc"의 주소가 str2에 저장됨
String str3 = new String("abc"); // 새로운 String인스턴스를 생성
String str4 = new String("abc"); // 새로운 String인스턴스를 생성
```



*상황에 따라 논리값이 달라진 수 있기 때문에 문자열 비교시 항상 equals()를 사용해 내용 비교를 해야 한다.
(등가비교는 주소비교)

ch9-9 문자열 리터럴 (리터럴=상수)

문자열 리터럴은 실행시 자동으로 생성 (constant pool에 저장)
(new연산자로 객체 생성 안해도됨) 상수저장소

ch9-10 빈 문자열("", empty string)

-내용X 문자열. 크기가 0인 char형 배열을 저장하는 문자열
-길이가 0인 배열을 생성하는 것은 어느 타입이나 가능

```
char[] ahArr = new char[0];
int[] iArr = {};
```

-문자와 문자열의 초기화

```
String s = "";
char c = ' ';
```

ch9-11 String클래스의 생성자와 메서드

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 String인스턴스를 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'`

int compareTo(String str)
문자열(str)과 사전순서로 비교한다. 같으면 0을, 사전순으로 이전이면 음수를, 이후면 양수를 반환한다.
	`int i = "aaa".compareTo("aaa");`
	`int i2 = "aaa".compareTo("bbb");`
	`int i3 = "bbb".compareTo("aaa");`
		`i = 0`
		`i2 = -1`
		`i3 = 1`

메서드 / 설명	예 제	결 과
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.	`String file = "Hello.txt";` `boolean b = file.endsWith("txt");`	`b = true`

boolean equals(Object obj)
매개변수로 받은 문자열(obj)과 String인스턴스의 문자열을 비교한다. obj가 String이 아니거나 문자열이 다르면 false를 반환한다.
	`String s = "Hello";`
	`boolean b = s.equals("Hello");`
	`boolean b2 = s.equals("hello");`
		`b = true`
		`b2 = false`

boolean equalsIgnoreCase(String str)
문자열과 String인스턴스의 문자열을 대소문자 구분없이 비교한다.
	`String s = "Hello";`
	`boolean b =`
	`  s.equalsIgnoreCase("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`

메서드 / 설명	예 제	결 과

int indexOf(int ch, int pos)
주어진 문자(ch)가 문자열에 존재하는지 지정된 위치(pos)부터 확인하여 위치(index)를 알려준다. 못 찾으면 -1을 반환한다. (index는 0부터 시작)
	`String s = "Hello";`
	`int idx1 = s.indexOf('e', 0);`
	`int idx2 = s.indexOf('e', 2);`
		`idx1 = 1`
		`idx2 = -1`

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`

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`

메서드 / 설명	예 제	결 과

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. (begin ≤ x < end)
	`String s = "java.lang.Object";`
	`String c = s.substring(10);`
	`String p = s.substring(5,9);`
		`c = "Object"`
		`p = "lang"`

메서드 / 설명	예 제	결 과
String toLowerCase() String인스턴스에 저장되어있는 모든 문자열을 소문자로 변환하여 반환한다.	`String s = "Hello";` `String s1 = s.toLowerCase();`	`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 = "Wed Jan 27 21:26:29 KST 2016"`

ch9-12 join()과 StringJoiner

join()은 여러 문자열 사이에 구분자를 넣어서 결합 한다

ch9-13 문자열과 기본형 간의 변환

숫자를 문자로 - > 1. 숫자 + “” //편리
2. String.valueOf(i); //빠름

문자를 숫자로 - > 1. Integer.parsInt("100");
2. Integer.valueOf("100");

기본형 → 문자열	문자열 → 기본형
String String.valueOf(boolean b) String String.valueOf(char c) String String.valueOf(int i) String String.valueOf(long l) String String.valueOf(float f) String String.valueOf(double d)	boolean Boolean.parseBoolean(String s) byte Byte.parseByte(S$ring 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)

참고 byte, short을 문자열로 변경할 때는 String valueOf(int i)를 사용하면 된다.

//이름이 제각각인 래퍼클래스들을 valueOf로 통일시킴

ch9-15 StringBuffer클래스

-String처럼 문자배열을 내부적으로 가지고 있다.
(문자열 저장, 다루는 역할)
-String 과는 달리 내용을 변경 할 수 있다.

ch9-16 StringBuffer의 생성자

-배열의 길이는 변경불가, 공간이 부족하면 새로운 배열을 생성 해야 한다.
1.새로운 배열생성 2.내용복사 3.참조변경(새로운 주소를 가리키도록)

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

public StringBuffer() {
    this(16);          // 버퍼의 크기를 지정하지 않으면 버퍼의 크기는 16이 된다.
}

public StringBuffer(String str) {
    this(str.length() + 16);   // 지정한 문자열의 길이보다 16이 더 크게 버퍼를 생성한다.
    append(str);
}
```

//재생성은 기능↓, 저장할 문자열의 길이를 고려해서 적절한 크기로 생성해야 한다. 기본은 16

ch9-17 StringBuffer의 변경

append():끝에 문자열 추가
delete(): 삭제
insert(): 삽입
는 반환타입이 StringBuffer이고 주소를 반환한다.

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

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

-> 때문에 오른쪽 코드와 같이 가능

ch9-18 StringBuffer의 비교

StringBuffer는 equals()가 오버라이딩 되어 있지 않다.
-> 주소비교를 한다 (String은 내용비교)
StringBuffer를 String으로 변환후에 equals()로 비교해야 한다.

```
String s  = sb.toString();      // sb를 String으로 변환
String s2 = sb2.toString();

System.out.println(s.equals(s2)); // true
```

ch9-19 StringBuffer의 생성자와 메서드

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

메서드 / 설명	예 제 / 결 과

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 길이를 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"

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

메서드 / 설명	예 제 / 결 과
void setCharAt(int index, char ch)	StringBuffer sb = new StringBuffer("0123456"); sb.setCharAt(5, 'o');
지정된 위치의 문자를 주어진 문자(ch)로 바꾼다.	sb = "01234o6"

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

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

ch9-21 StringBuilder

-StringBuffer와 똑같다. 한가지 차이가 있는데 Builder은 동기화가 안되어 있고 Buffer은 되어있다.
-동기화란 멀티스레드에 안전하게 하는 것

스레드 - 싱글스레드 : 한번에 1개 작업
- 멀티스레드 : 한번에 n개의 작업
(데이터 공유-데이터 꼬임 발생가능성↑)

동기화는 멀티스레드 프로그램이 아닌 경우 불필요하기 때문에 싱글스레드 프로그램일 때 StringBuffer대신 StringBuilder를 사용하면 성능 향상!

ch9-22 Math클래스

-수학 관련 static메서드의 집합 (객체 생성 불필요)
math클래스의 메서드

메서드 / 설명	예 제	결 과
static double abs(double a) **static float abs(float f)** **static int abs(int f)** **static long abs(long f)** 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(10.000015);
		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 double max(double a, double b) **static float max(float a, float b)** **static int max(int a, int b)** **static long max(long a, long 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

메서드 / 설명	예 제	결 과
static double min(double a, double b) **static float min(float a, float b)** **static int min(int a, int b)** **static long min(long a, long b)** 주어진 두 값을 비교하여 작은 쪽을 반환한다.	double d = Math.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. (1.0은 범위에 포함되지 않는다.)	double d = Math.random(); int i = (int)(Math.random()*10)+1	0.0<=d<1.0 1<=i<11

static double rint(double a)
주어진 double값과 가장 가까운 정수값을 double형으로 반환한다. 단, 두 정수의 정가운데 있는 값(1.5, 2.5, 3.5 등)은 짝수를 반환.
	double d = Math.rint(1.2);
	double d2 = Math.rint(2.6);
	double d3 = Math.rint(3.5);
	double d4 = Math.rint(4.5);
		d = 1.0
		d2 = 3.0
		d3 = 4.0
		d4 = 4.0

static long round(double a)
static long round(float a)
소수점 첫째자리에서 반올림한 정수값(long)을 반환한다. 두 정수의 정가운데있는 값은 항상 큰 정수를 반환.(rint()의 결과와 비교)
	long l = Math.round(1.2);
	long l2 = Math.round(2.6);
	long l3 = Math.round(3.5);
	long l4 = Math.round(4.5);
	double d= 90.7552;
	double d2 = Math.round(d*100)/100.0;
		l = 1
		l2 = 3
		l3 = 4
		l4 = 5
		d = 90.7552
		d2 = 90.76

ch9-25 래퍼(wrapper) 클래스 (기본형을 감싸는 class)

-8개의 기본형을 객체로 다뤄야할 때 사용하는 클래스

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");

예제 9-14

```
결과
i==i2 ? false
i.equals(i2) ? true
i.compareTo(i2)=0
i.toString()=100
MAX_VALUE=2147483647
MIN_VALUE=-2147483648
SIZE=32 bits
BYTES=4 bytes
TYPE=int
```

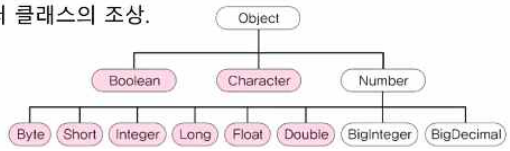
```
class Ex9_14 {
    public static void main(String[] args) {
        Integer i  = new Integer(100);
        Integer i2 = new Integer(100);

        System.out.println("i==i2 ? "+(i==i2));
        System.out.println("i.equals(i2) ? "+i.equals(i2));
        System.out.println("i.compareTo(i2)="+i.compareTo(i2));
        System.out.println("i.toString()="+i.toString());

        System.out.println("MAX_VALUE="+Integer.MAX_VALUE);
        System.out.println("MIN_VALUE="+Integer.MIN_VALUE);
        System.out.println("SIZE="+Integer.SIZE+" bits");
        System.out.println("BYTES="+Integer.BYTES+" bytes");
        System.out.println("TYPE="+Integer.TYPE);
```

ch9-27 Numver클래스

모든 숫자 래퍼 클래스의 조상.




```
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();
    }
}
```

래퍼객체가 가지고 있는 값을 기본형으로 바꿀 때 쓰는 메서드를 가지고 있다.

ch9-28 문자열을 숫자로 변환하기

-문자열을 숫자로 반환하는 다양한 방법

```
int     i  = new Integer("100").intValue(); // floatValue(), longValue(), ...
int     i2 = Integer.parseInt("100");       // 주로 이 방법을 많이 사용.
Integer i3 = Integer.valueOf("100");
```

문자열 → 기본형	문자열 → 래퍼클래스

byte b = Byte.parseByte("100");
short s = Short.parseShort("100");
int i = Integer.parseInt("100");
long l = Long.parseLong("100");
float f = Float.parseFloat("3.14");
double d = Double.parseDouble("3.14");
	Byte b = Byte.valueOf("100");
	Short s = Short.valueOf("100");
	Integer i = Integer.valueOf("100");
	Long l = Long.valueOf("100");
	Float f = Float.valueOf("3.14");
	Double d = Double.valueOf("3.14");

n진법의 문자열을 숫자로 변환하는 방법

```
int i4 = Integer.parseInt("100",2);  // 100(2) -> 4
int i5 = Integer.parseInt("100",8);  // 100(8) -> 64
int i6 = Integer.parseInt("100",16); // 100(16)-> 256
int i7 = Integer.parseInt("FF", 16); // FF(16) -> 255
//  int i8 = Integer.parseInt("FF");     // NumberFormatException발생
```

ch9-30 오토박싱 & 언박싱

기본형을 래퍼클래스(객체)로 -> 오토박싱
래퍼클래스를 기본형으로 -> 언박싱
//JDK1.5이전에는 기본형과 참조형간의 연산이 불가능
래퍼형을 기본형으로 컴파일러가 자동으로 전환

예제 9-16

```
class Ex9_16 {
    public static void main(String[] args) {
        int i = 10;

        // 기본형을 참조형으로 형변환(형변환 생략가능)
        Integer intg = (Integer)i; // Integer intg = Integer.valueOf(i);
        Object obj = (Object)i;    // Object obj = (Object)Integer.valueOf(i);

        Long lng = 100L; // Long lng = new Long(100L);

        int i2 = intg + 10;  // 참조형과 기본형간의 연산 가능
        long l = intg + lng; // 참조형 간의 덧셈도 가능

        Integer intg2 = new Integer(20);
        int i3 = (int)intg2; // 참조형을 기본형으로 형변환도 가능(형변환 생략가능)
```

컴파일 전의 코드	컴파일 후의 코드
Integer intg = (Integer)i; Object obj = (Object)i; Long lng = 100L;	Integer intg = Integer.valueOf(i); Object obj = (Object)Integer.valueOf(i); Long lng = new Long(100L);