

12. Java의 기본 API

Java SE 6.0에서 제공되는 표준 API들 중에서 기본 API라는 것은 Java 프로그램을 개발할 때 거의 필수로 사용되는 API를 말하며 프로그램의 기본이 되는 클래스와 인터페이스들을 제공하는 `java.lang` 패키지, 프로그래밍시 여러 가지 편리한 기능의 클래스들을 제공하는 `java.util` 패키지 그리고 텍스트, 날짜, 수치 및 메시지 처리 클래스들을 제공하는 `java.text` 패키지로 구성된다.

java.lang의 주요 API

`java.lang`은 자동 import되면서 가장 필수적으로 사용되는 패키지로서 `Object`, `String`, `StringBuffer`, `StringBuilder`, `Comparable` 그리고 `Cloneable`의 기능과 활용 방법을 파악한다.

■ Object

`Object`클래스는 모든 클래스의 최고 조상이기 때문에 `Object`클래스의 멤버들은 모든 클래스에서 바로 사용 가능하다. `Object`클래스는 모든 Java 객체들이 가져야 할 기본적인 기능의 메서드 11개를 가지고 있다. 대부분은 자손 클래스에서 필요에 따라 오버라이딩하여 구현하는 메서드들이다.

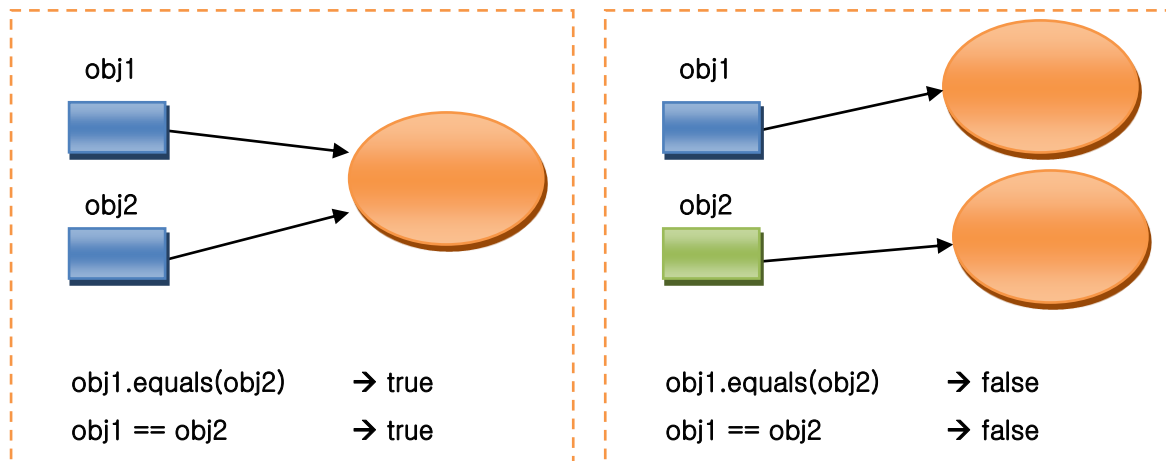
Object클래스의 메서드	설 명
<code>protected Object clone()</code>	객체 자신의 복사본을 반환한다.
<code>public boolean equals(Object obj)</code>	객체 자신과 객체 <code>obj</code> 가 같은 객체인지 알려준다. (같으면 <code>true</code>)
<code>protected void finalize()</code>	객체가 소멸될 때 가비지 컬렉터에 의해 자동적으로 호출된다. 이 때 수행되어야 하는 코드가 있는 경우에만 오버라이딩한다.
<code>public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

▶ equals() 메서드

Object클래스에서 제공되는 equals()메서드는 매개변수로 객체의 참조변수를 받아 등가연산자로 비교하여 그 결과를 boolean값으로 리턴하는 역할을 수행한다. 다음 코드는 Object클래스에 정의되어 있는 equals()메서드의 내용이다.

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

다음 그림에서와 같이 두 참조 형 변수가 동일한 객체를 참조하는 경우에만 true를 리턴한다. 참조되는 객체의 내용을 비교하는 것이 아니고 두 참조 형 변수가 동일한 객체를 참조하는지를 비교하게 되는 결과가 되며 이것은 두 참조 형 변수에 대하여 직접 등가 연산자(==)를 수행하는 것과 동일하다.



그런데 실질적인 객체와 객체의 비교는 객체들의 내용을 비교해야 하는 경우를 흔히 볼 수 있으며 이러한 경우에는 Object클래스에서 제공되는 equals()메서드로는 해결을 할 수가 없으며 내용을 비교하려는 객체에 알맞게 Object클래스의 equals()메서드를 오버라이딩하여 비교하려는 내용에 알맞게 구현해주어야 한다.

다음에 제시된 Member클래스는 equals()메서드를 오버라이딩하지 않았다.

```
class Member {  
    int id;  
    String name;  
    String password;  
  
    Member(int id, String name, String password) {  
        this.id = id;  
        this.name = name;  
        this.password = password;  
    }  
}
```

이런 경우에는 Object에서 제공되는 equals()메서드가 대신 호출되는 경우가 되며 다음과 같이 두 개의 Member클래스의 객체를 생성하여 equals()를 호출하게 되면 false를 리턴하게 된다.

```
Member obj1 = new Member(10, "Java", "duke");  
Member obj2 = new Member(10, "Java", "duke");  
  
System.out.println(obj1.equals(obj2)); // false
```

다음은 Member클래스 안에 Member객체들의 비교하고자 하는 내용에 알맞게 equals()메서드를 오버라이딩하여 구현한 소스이다.

```
public boolean equals(Object o) {  
    if(o != null && o instanceof Member) {  
        Member m = (Member)o;  
        if(id == m.id && name.equals(m.name) &&  
            password.equals(m.password))  
            return true;  
    }  
    return false;  
}
```

위와 같은 equals()메서드를 Member클래스에 정의하게 되면 Member클래스의 객체를 생성해서 equals()를 호출하게 되면 오버라이딩된 equals()가 호출되어 두 객체에 초기화되어 있는 id, name, password 변수를 각각 비교하여 동일한 내용의 객체인지를 비교하게 되는 결과가 되어 다음과 같이 호출하면 true가 리턴된다.

```
Member obj1 = new Member(10, "Java", "duke");
Member obj2 = new Member(10, "Java", "duke");

System.out.println(obj1.equals(obj2)); // true
```

equals() 메서드를 통해 두 객체의 내용이 동일한지를 비교하고자 하는 경우에는 해당 객체에 알맞게 클래스안에 equals()를 오버라이딩해야 한다.

▶ hashCode() 메서드

객체의 해쉬코드값을 리턴하는 기능의 메서드로서 객체의 내용과 내용을 비교할 수 있는 또 다른 메서드이다. boolean타입의 값을 리턴하는 equals()와는 다르게 hashCode()는 객체에 대한 해쉬코드값을 정수형으로 리턴한다.

다음은 Object클래스에 정의되어 있는 hashCode()메서드의 선언부이다.

```
public int hashCode()
```

Object클래스에 정의되어 있는 hashCode()메서드는 이 객체의 참조 값 즉, 논리적인 주소값으로 생성하기 때문에 각각의 객체들은 무조건 서로 다른 해쉬코드값을 가지게 되는 결과가 된다.

서로 다른 종류의 객체를 구분할 공통기준이 참조 값외에는 없기 때문이기도 하지만, 객체의 멤버변수의 값들을 하나하나 비교하는 것보다는 4byte의 참조 값을 비교하는 것이 더 빠르기 때문이다.

다음은 hashCode()를 오버라이딩하지 않은 Member클래스의 객체를 두 개 생성하여 각각의 hashCode()메서드를 호출한 부분 예제이다. Member클래스의 객체가 생성될 때마다 할

당되는 메모리상의 논리적인 주소 값이 서로 다르므로 서로 다른 해쉬코드값이 리턴되는 것을 볼 수 있다.

```
Member obj1 = new Member(10, "Java", "duke");
Member obj2 = new Member(10, "Java", "duke");

System.out.println(obj1.equals(obj2));
System.out.println(obj1.hashCode()); // 12677476
System.out.println(obj2.hashCode()); // 33263331
```

equals()메서드와 마찬가지로 동일한 타입, 동일한 내용의 객체에 대하여 동일한 해쉬코드 값이 추출되도록 하고자 하는 경우에는 해당 객체의 클래스내에 hashCode()메서드를 오버라이딩해서 구현해야 한다.

hashCode()메서드를 오버라이딩할 때는 다음 사항들을 고려한다.

- 동일 객체에 대해서 hashCode()를 여러 번 호출해도 동일한 값을 반환해야 한다.
- equals()로 비교해서 true를 얻은 두 객체의 hashCode()값은 일치해야 한다.

다음은 Member클래스에 구현할 수 있는 hashCode()메서드의 구현 예이다.

```
public int hashCode() {
    return id+name.hashCode()+password.hashCode();
}
```

hashCode()를 오버라이딩할 때는 객체에 정의되어 있는 멤버변수들 중 비교 대상되는 것들 것 값을 덧셈연산하여 리턴하는 것이다. 멤버 변수의 타입이 숫자형이면 바로 덧셈 연산에 사용하면 되고 문자열 타입과 같은 참조 타입의 경우에는 해당 클래스에서 제공되는 hashCode()메서드를 호출하며 리턴된 값을 더하면 된다. 다음은 hashCode()를 오버라이딩 하고 있는 Member클래스의 hashCode()호출하고 있는 예제의 부분 소스로서 동일한 내용을 같은 Member객체의 경우에는 동일한 해쉬코드 값이 리턴되고 다른 내용을 갖는 Member객체는 서로 다른 해쉬코드를 리턴하는 것을 볼 수 있다.

```

Member obj1 = new Member(10, "Java", "duke");
Member obj2 = new Member(10, "Java", "duke");

System.out.println(obj1.equals(obj2));           // true
System.out.println(obj1.hashCode());             // 4726841
System.out.println(obj2.hashCode());             // 4726841

Member obj3 = new Member(20, "OCJP", "test");
System.out.println(obj1.equals(obj3));           // false
System.out.println(obj3.hashCode());             // 5976768

```

▶ toString()메서드

toString()은 객체에 대한 정보를 문자열(String)로 제공할 목적으로 사용되는 메서드이다. 객체타입을 문자열 타입으로 변환해야 하는 경우에도 호출되며 객체 타입을 화면에 표준 출력할 때로 자동으로 호출되는 메서드이다.

```

"일시 : " + new Date()           // Date객체에 대하여 toString()이 자동 호출
System.out.println(new Date()); // Date객체에 대하여 toString()이 자동 호출

```

Object클래스에 정의된 toString()메서드는 다음과 같이 구현되어 있다. 객체의 클래스명에 해쉬코드값을 16진수로 변환하여 @ 기호와 함께 조합하여 리턴한다.

```

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

```

그러므로 toString()을 오버라이딩하지 않은 Member클래스에 대해서 toString()을 호출하면 다음과 같이 클래스명, @ 기호 그리고 객체의 해쉬코드값이 16진수로 출력되는 것을 볼 수 있다.

```
Member obj = new Member(10, "Java", "duke");

System.out.println(obj.toString());           //exam11.Member@482039
System.out.println(obj);                      // exam11.Member@482039
System.out.println("Member 객체의 내용 : " + obj);
// Member 객체의 내용 : exam11.Member@482039
```

그러므로 toString()의 경우에도 객체에 따른 적당한 문자열이 리턴되기를 원한다면 해당 객체의 클래스안에 toString()을 오버라이딩하여 구현한다. 다음은 Member클래스에 추가하는 toString()메서드의 구현 내용이다.

```
public String toString() {
    return "계정 : "+id+" 이름 : "+ name+" 암호 : "+password;
}
```

위와 같이 toString()을 오버라이딩하면 오버라이딩된 toString()이 대신 호출되어 객체에 대한 정보를 하나의 문자열로 표현하고자할 때 다음과 같이 해당 클래스에 알맞은 내용의 문자열이 리턴된다.

```
Member obj = new Member(10, "Java", "duke");

System.out.println(obj);    // 계정 : 10 이름 : Java 암호 : duke
```

▶ clone()메서드와 Cloneable

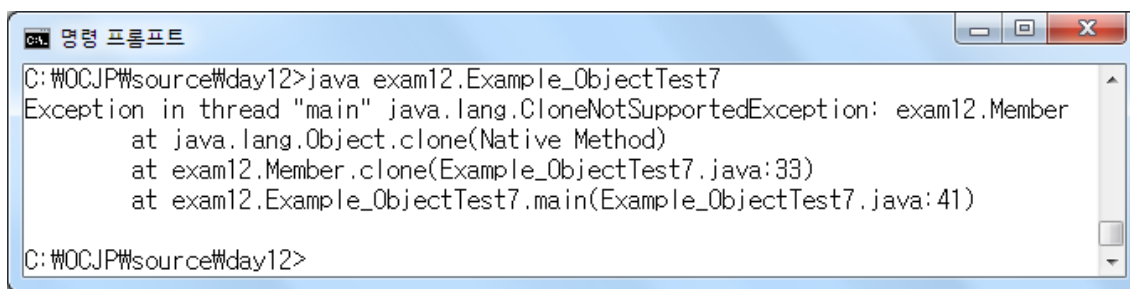
Java에서는 객체의 실제 메모리 주소를 직접 사용하지는 않는다. 객체 사용시에는 생성된 객체의 메모리로 작업을 하는 것이 아니라 메모리에 연결된 참조 값으로 작업을 하게 된다. 참조 값을 이용하기 때문에 객체를 메모리 차원에서 직접 복사하는 행위는 불가능하지만 Object클래스에서 제공되는 clone()메서드를 사용하여 메모리 차원의 복제를 처리할 수 있다.

Object클래스에서 제공되는 clone()메서드는 protected형으로서 자손에서는 호출할 수 있어도

외부에서 멤버 연산자로는 호출할 수 없다. 복제가 가능한 클래스를 만들고자 하는 경우에는 우선 Cloneable인터페이스를 추가 상속하도록 구현한 다음 clone()메서드를 오버라이딩하고 이 안에서 조상의 clone()을 호출하도록 구현한다.

Cloneable인터페이스는 아무런 추상 메서드 없이 해당 객체가 clone()메서드 호출을 통해 복제가 가능하다고 알리는 목적으로 사용되는 인터페이스이다. Cloneable인터페이스를 구현한 객체는 Object클래스의 protected 메서드인 clone()메서드를 super.clone()과 같이 호출하여 해당 클래스의 복제본을 만들 수 있다.

clone()메서드를 가지고 있다 하더라도 Cloneable인터페이스를 추가 상속하지 않은 클래스의 객체에 대하여 clone()메서드를 호출하면 다음과 같이 CloneNotSupportedException을 발생 시킨다.



```
C:\WOCJP\source\day12>java exam12.Example_ObjectTest7
Exception in thread "main" java.lang.CloneNotSupportedException: exam12.Member
    at java.lang.Object.clone(Native Method)
    at exam12.Member.clone(Example_ObjectTest7.java:33)
    at exam12.Example_ObjectTest7.main(Example_ObjectTest7.java:41)
C:\WOCJP\source\day12>
```

clone() 메서드를 오버라이딩 하여 구현할 때는 다음 규약을 적용한다.

1. x.clone() != x
2. x.clone().getClass() == x.getClass()
3. x.clone().equals(x) 이지만 필수는 아니다.
4. clone() 메서드에서는 조상의 clone() 메서드를 호출하여 메모리 차원의 복제가 가능하도록 구현한다.(Object클래스의 clone()메서드만 메모리차원의 복제가 가능하다.)

다음은 Member클래스에 clone()메서드를 추가한 부분 예제이다.


```

class Member implements Cloneable {
    int id;
    String name;
    String password;

    Member(int id, String name, String password) {
        this.id = id;
        this.name = name;
        this.password = password;
    }

    :

    public Object clone() throws CloneNotSupportedException {
        // Object의 clone()을 호출하여 메모리 차원의 복제 수행
        return super.clone();
    }
}

```

■ String 과 StringBuffer/StringBuilder

기존의 다른 언어에서는 문자열을 char형의 배열로 다루었으나 Java에서는 문자열 처리를 위한 클래스를 API로 크게 두 가지로 나누어 제공한다.

- 초기화된 문자열의 내용을 읽을 수만 있는 기능을 제공하는 String클래스
- 초기화된 문자열의 내용을 편집하는 용도로 사용하는 StringBuffer/StringBuilder클래스

▶ String클래스

String클래스에는 문자열을 저장하기 위해서 문자 타입의 배열 변수(char[]) value를 인스턴스 변수로 정의해놓고 객체 생성 시 생성자의 매개변수로 입력받은 문자열을 이 인스턴스 변수(value)에 문자 타입 배열(char[])로 저장한다.

한번 생성된 String 객체가 가지고 있는 문자열은 읽어 올 수만 있고, 변경할 수는 없다. 예를 들어 "a" + "b"와 같이 '+'연산자를 이용해서 문자열을 결합하는 경우 객체내의 문자열이 바뀌는 것이 아니라 새로운 문자열("ab")이 담긴 새로운 String객체가 생성되는 것이다.

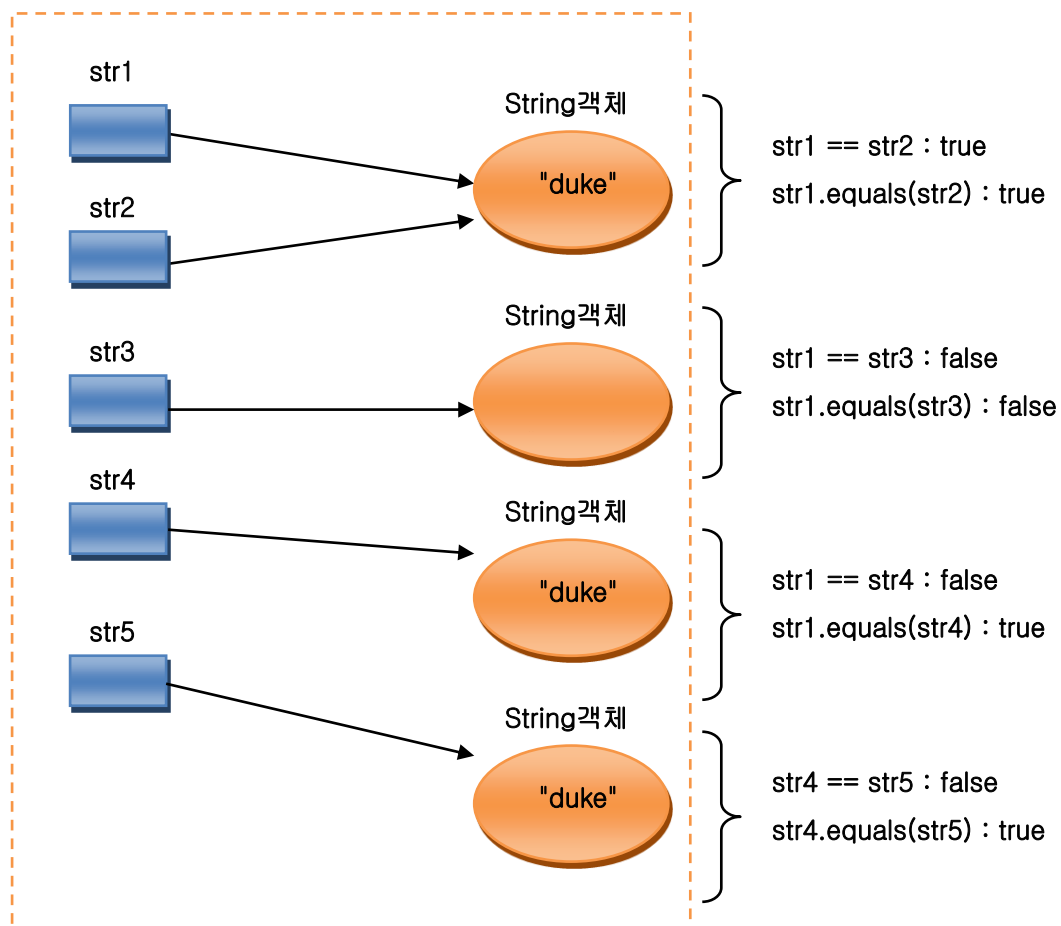
이처럼 덧셈연산자(+)를 사용해서 문자열을 결합하는 것은 매 연산 시 마다 새로운 문자열을 가

진 String객체가 생성되어 메모리공간을 차지하게 되므로 가능한 한 결합횟수를 줄이는 것이 좋으며 문자열간의 결합이나 추출 등 문자열을 편집하는 작업이 많이 필요한 경우에는 String클래스 대신 StringBuffer클래스를 사용하는 것이 더욱 적당하다.

Java에서 문자열을 만드는데 사용되는 방법은 두 가지이다. 하나는 String클래스의 생성자를 사용해서 원하는 내용으로 초기화되는 String객체를 직접 생성해서 문자열을 만드는 것이고 다른 하나는 문자열 리터럴을 사용하는 방법이다. Java에서는 문자열 리터럴을 String타입의 객체로 생성하여 처리하기 때문이다.

```
String str1 = "duke";
String str2 = "duke";
String str3 = "DUKE";
String str4 = new String("duke");
String str5 = new String("duke");
```

문자열 리터럴을 사용하는 경우의 특징은 동일한 내용으로 구성되는 문자열 리터럴의 경우에는 다음 그림과 같이 String타입의 객체를 하나만 생성하여 공유하게 된다는 것이다.



프로그램 안에서 사용하고자 하는 문자열을 문자열 리터럴로 생성하여 사용하는 경우에는 동일 내용으로 구성되는 문자열 리터럴의 경우 String타입의 객체가 하나만 생성되어 공유되므로 등가연산의 결과는 true가 되지만 직접 String타입의 객체를 생성하여 사용하는 경우에는 등가연산의 결과가 false이다. 그러므로 두 문자열의 내용으로 비교하고자 하는 경우에는 equals()메서드를 호출하여 비교해야 한다. String클래스의 경우에도 String클래스에 알맞게 equals()메서드를 오버라이딩하고 있다 그런데 대소문자를 구분하므로 대소문자 구별하지 않고 비교하고자 하는 경우에는 equalsIgnoreCase()메서드 사용한다.

이외에도 String클래스에는 문자열 처리와 관련된 다음과 같은 주요 메서드들을 제공하고 있다.

char charAt(int index)	Index번째 문자를 반환한다.
int compareTo(Object o)	o와 비교하여 결과로 양수, 음수, 0의 값을 반환한다.
int compareTo(String anotherString)	2개의 문자열을 비교하여 결과로 양수, 음수, 0의 값을 반환한다.
int compareToIgnoreCase(String str)	대소문자를 구분하지 않고 2개의 문자열을 비교하여 결과로 양수, 음수, 0의 값을 반환한다.
String concat(String str).	'+' 연산자처럼 문자열을 결합한다.
boolean equals(Object anObject)	2개의 문자열값을 비교하여 true, false를 반환한다.
boolean equalsIgnoreCase (String anotherString)	대소문자를 구분하지 않고 2개의 문자열값과 비교하여 true, false를 반환한다.
int hashCode()	Object 클래스의 메서드 오버라이딩이다.
int length()	문자열의 길이를 반환한다.
boolean matches(String regex)	두 개의 문자열이 일치하는지의 여부를 비교한다.
String replace (char oldChar, char newChar)	oldChar를 newChar로 변환한다.
String[] split(String regex)	지정된 regex를 중심으로 여러 개의 문자열 배열로 분리한다.
String[] split(String regex, int limit)	지정된 개수만큼 지정된 regex를 중심으로 여러 개의 문자열 배열로 분리한다.
boolean startsWith(String prefix)	특정 문자열로 시작하는지 비교한다.
String substring(int beginIndex)	특정 위치부터의 문자열만 반환한다.
String substring (int beginIndex, int endIndex)	지정된 특정 위치부터 특정 위치까지의 문자열만 반환한다.
char[] toCharArray()	문자 배열을 반환한다.
String toLowerCase()	소문자로 변환한다.
String toString()	문자열로 변환한다. Object 클래스의 메서드 오버라이딩이다.
String toUpperCase()	대문자로 변환한다.
String trim()	문자열 양쪽의 공백을 제거한다.

[빈 문자열(empty string)]

빈 문자열이란 크기가 0인 문자 타입의 배열을 갖는 String 타입의 객체이다. 더불인용부호 두 개를 이어서("") 사용한다.

```
String s = "";
```

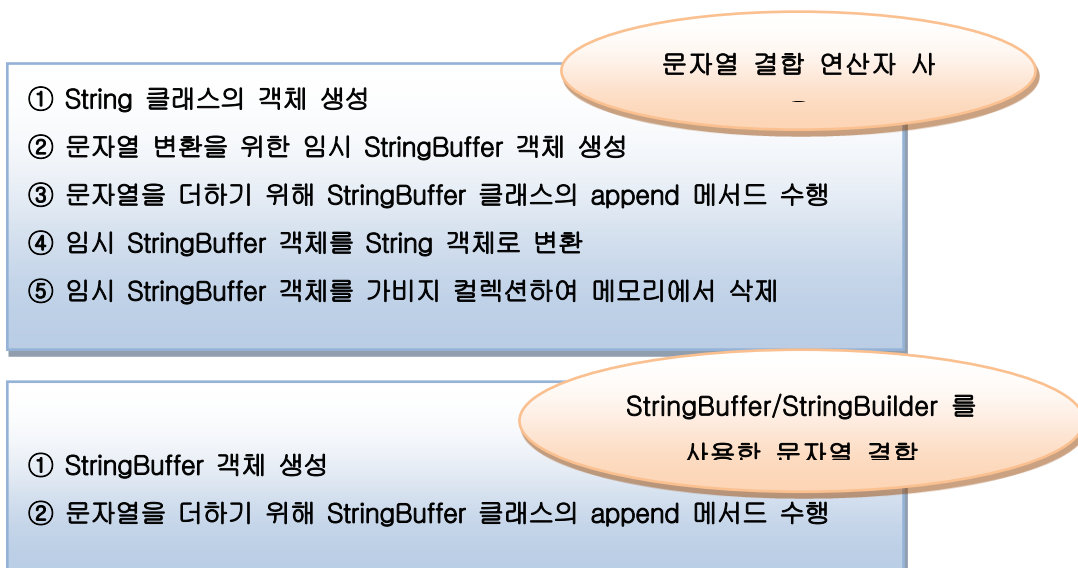
s.length()를 호출해 보면 0이 리턴되고 s를 출력하면 아무것도 출력되지 않는다. String타입의 변수를 초기화할 때 주로 사용되는 문자열 리터럴이다.

▶ StringBuffer/StringBuilder클래스

StringBuffer클래스는 내용이 변할 수 있는 문자열을 다룰 때 사용한다. StringBuffer클래스의 객체는 크기가 동적인데, 객체 생성시 크기를 지정하지 않아도 기본적으로 16개의 문자를 저장할 수 있는 버퍼 공간을 가진다.

String클래스의 객체는 한 번 생성되면 그 내용이 변하지 않는 반면에, StringBuffer클래스의 객체는 한 번 생성된 후에도 계속하여 저장하고 있는 문자열의 내용을 변경할 수 있다. 그러므로, StringBuffer클래스의 메서드는 문자열 처리 후의 결과를 원래의 StringBuffer객체에 반영하고, 메서드 리턴 타입은 대부분 void이다.

여러 문자들을 결합할 때 문자열 결합 연산자인 + 연산자를 사용하는 것도 편하지만 결합할 문자들이 많을 때 StringBuffer/StringBuilder클래스를 사용하는 것이 더욱 효율적이다. 다음은 문자열 결합 연산자(+) 문자열 결합을 수행하는 경우의 처리과정과 StringBuffer/StringBuilder클래스를 사용할 때의 처리과정을 소개하는 내용이다.



다음은 StringBuffer/StringBuilder 클래스에서 제공되는 주요 메서드들의 내용이다.

StringBuffer append(String str)	문자열 데이터를 현재 문자열 끝에 추가한다.
StringBuffer delete(int start, int end)	start에서 end-1까지의 인덱스 위치 문자들을 삭제하고 문자열을 반환한다.
StringBuffer deleteCharA (int index)	index 위치의 문자 중 삭제한 문자열을 반환한다.
StringBuffer insert(int offset, String str)	offset 위치에 문자열 데이터를 삽입한다.
int length()	문자열 내의 문자 개수를 반환한다.
StringBuffer replace (int start, int end, String str)	start에서 end-1까지의 인덱스 위치 문자열을 str 문자열로 대체하여 반환한다.
StringBuffer reverse()	문자열의 역순으로 된 문자열을 반환한다.
void setCharAt(int index, char ch)	index 위치의 문자를 ch 문자로 설정한다.
void setLength(int newLength)	문자열의 버퍼 크기를 새롭게 설정한다.
String substring(int start)	start 인덱스 위치로부터의 일부 문자열을 반환한다.
String substring(int start, int end)	start에서 end-1까지의 인덱스 위치의 일부 문자열을 반환한다.

■ Wrapper 클래스와 Autoboxing

Java에서는 8가지 기본형 타입을 지원하고 있다. 기본형이란 프로그램내에서 처리하려는 데이터의 값을 저장하여 처리하는 데이터 타입으로서 char, boolean, byte, short, int, long, float 그리고 double형이 사용되고 있다.

Java는 또한 기본형을 값(value)이 아닌 객체로 사용할 수 있게 지원하는데, 기본형을 객체로 사용한다는 것은 기본형과 연관된 클래스가 제공된다는 의미이다. 기본형을 값으로 사용하는 것과 객체로 사용하는 것은 실행시간의 효율성 측면에서 차이가 있다. 객체는 주소를 참조하여 값에 접근하게 되므로 값을 직접 사용하는 것보다는 실행시간의 효율성이 떨어진다. 반면에 연산자로는 할 수 없는 클래스에서 제공되는 유용한 메서드들을 사용할 수 있다.

Java는 기본형에 따른 객체를 지원하기 위해 각각의 기본형과 관련된 클래스를 제공하는데, 이를 **Wrapper 클래스**라고 한다.

다음은 8가지의 기본형과 각 기본형에 매핑되는 Wrapper 클래스의 내용이다.

기본형 명칭	Wrapper 클래스 명칭
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

8가지의 Wrapper 클래스는 많은 메서드들이 제공되지만, 특히 **사용자의 문자열 입력을 다양한 기본 데이터형으로 변환**하는 편리한 메서드들을 제공한다. Wrapper 클래스 객체는 한번 생성되면 그 값이 변할 수 없다.

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");

▶ AutoBoxing/AutoUnBoxing

참조형(객체형)을 요구하는 자리에 기본형 데이터가 오면 자동으로 기본형 타입에 매핑되는 Wrapper클래스의 객체로 변환하여 참조형으로 사용할 수 있도록 하는 기능을 AutoBoxing 이라고 하며 기본형을 요구하는 자리에 Wrapper클래스의 객체가 오면 자동으로 매핑되는 기본형으로 변환해 주는 것을 AutoUnBoxing이라고 한다. 다음에 제시된 부분 소스를 점검한다.

AutoBoxing/AutoUnBoxing은 기본형 타입과 각 기본형 타입에 매핑되는 Wrapper클래스 타입간의 자동 형 변환이라고 할 수 있다.

```
int su = 1000;
Integer obj1 = new Integer(100);
Integer obj2 = 100;           // int타입 데이터 100이 자동으로 Integer객체가 된다.
Integer obj3 = su;           // int타입 su의 값이 자동으로 Integer객체가 된다.
int number = obj1+obj2+obj3;  // Integer객체들이지만 int으로 변환되어 연산된다.
System.out.println(obj1+10); // Integer객체이지만 int으로 변환되어 연산된다.
System.out.println(number);
```

■ Comparable

객체 정렬을 필요로 하는 객체의 클래스를 정의할 때 상속하는 인터페이스로서 compareTo()라는 메서드를 구현하도록 제시하고 있다.

```
public Interface Comparable {
    int compareTo(Object o);
}
```

TreeSet 객체에 객체를 보관하거나 sort()메서드를 사용하여 객체들을 정렬하고자 하는 경우에는 해당 객체가 Comparable을 상속하고 있어야 한다.

두 객체를 비교해서 같으면 0을 작으면 음수, 크면 양수를 리턴하도록 구현한다. 이 리턴 값을 통해 두 객체의 정렬순서가 결정된다. 객체의 어떤 값을 가지고 비교할 것인지를 결정하여 비교하며 a.compareTo(b)와 같이 수행 시켰을 때 참조형 변수 a를 기본으로 하여 참조형 변수 b가 참조하는 값을 비교했을 때 a가 더 크면 1을, 동일하면 0을 더 작으면 -1을 리턴하도록 구현하는 것이 일반적인 구현이다.

다음은 Member클래스에 정의될 수 있는 compareTo()메서드의 구현 내용이다.

```
class Member implements Comparable {
    public int compareTo(Object o) {
        int result = 0;
        if (o != null && o instanceof Member) {
            Member m = (Member)o;
            if (this.id > m.id)           // Member객체의 id값을 가지고 비교하
                result = 1;              // 고 있다.
            else if (this.id == m.id)
                result = 0;
            else
                result = -1;

        } else
            throw new ClassCastException();
        return result;
    }
}
```