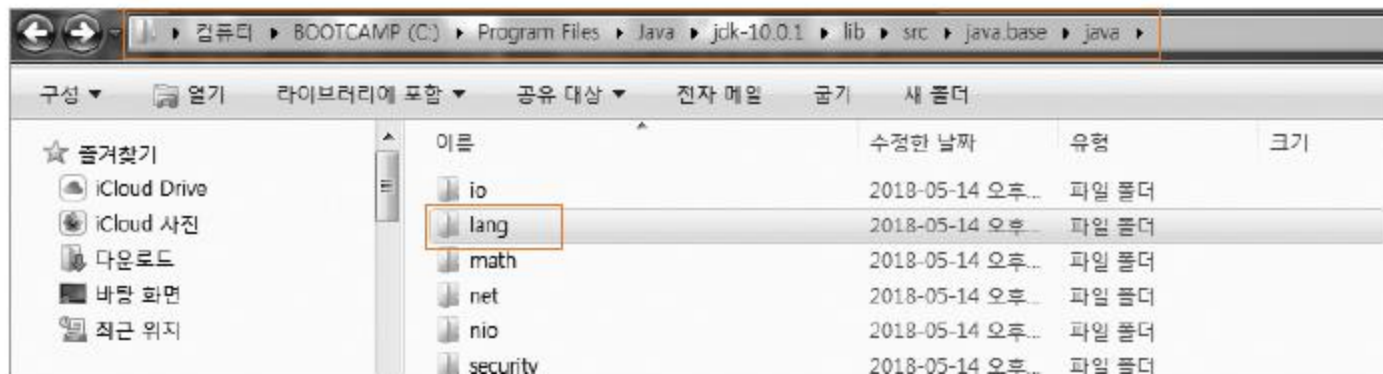


JAVA 프로그래밍

기본 클래스

java.lang 패키지

- 프로그래밍 시 import 하지 않아도 자동으로 import 됨
- `import java.lang.*;` 문장이 추가됨
- 많이 사용하는 기본 클래스들이 속한 패키지
- String, Integer, System 등



Object 클래스

- 모든 클래스의 최상위 클래스
- java.lang.Object 클래스
- 모든 클래스는 Object 클래스에서 상속 받음
- 모든 클래스는 Object 클래스의 메서드를 사용할 수 있음
- 모든 클래스는 Object 클래스의 메서드 중 일부는 재정의할 수 있음
(final로 선언된 메서드는 재정의할 수 없음)
- 컴파일러가 extends Object를 추가함

```
class Student {  
    int studentID;  
    String studentName;  
}
```

코드를 작성할 때



```
class Student extends Object {  
    int studentID;  
    String studentName;  
}
```

컴파일러가 변환

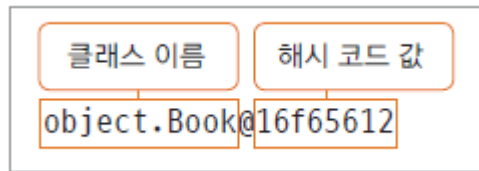
Object 클래스 메서드

메서드	설명
String toString()	객체를 문자열로 표현하여 반환합니다. 재정의하여 객체에 대한 설명이나 특정 멤버 변수 값을 반환합니다.
boolean equals(Object obj)	두 인스턴스가 동일한지 여부를 반환합니다. 재정의하여 논리적으로 동일한 인스턴스임을 정의할 수 있습니다.
int hashCode()	객체의 해시 코드 값을 반환합니다.
Object clone()	객체를 복제하여 동일한 멤버 변수 값을 가진 새로운 인스턴스를 생성합니다.
Class getClass()	객체의 Class 클래스를 반환합니다.
void finalize()	인스턴스가 힙 메모리에서 제거될 때 가비지 컬렉터(GC)에 의해 호출되는 메서드입니다. 네트워크 연결 해제, 열려 있는 파일 스트림 해제 등을 구현합니다.
void wait()	멀티스레드 프로그램에서 사용하는 메서드입니다. 스레드를 '기다리는 상태'(non runnable)로 만듭니다.
void notify()	wait() 메서드에 의해 기다리고 있는 스레드(nonrunnable 상태)를 실행 가능한 상태(runnable)로 가져옵니다.

toString() 메서드

- Object 클래스의 메서드

```
getClass( ).getName( ) + '@' + Integer.toHexString(hashCode( ))
```



- 객체의 정보를 String으로 바꾸어서 사용할 때 많이 쓰임
- String이나 Integer 클래스에는 이미 재정의 되어 있음
- String은 문자열 반환
- Integer는 정수 값 반환

String과 integer의 toString() 메서드

- toString() 메서드가 재정의 되어 있음
- String은 문자열 반환
- Integer는 정수 값 반환

```
String str = new String("test");
```

```
System.out.println(str);
```

test 출력됨

```
Integer i1 = new Integer(100);
```

```
System.out.println(i1);
```

100 출력됨

toString() 메서드 재정의하기

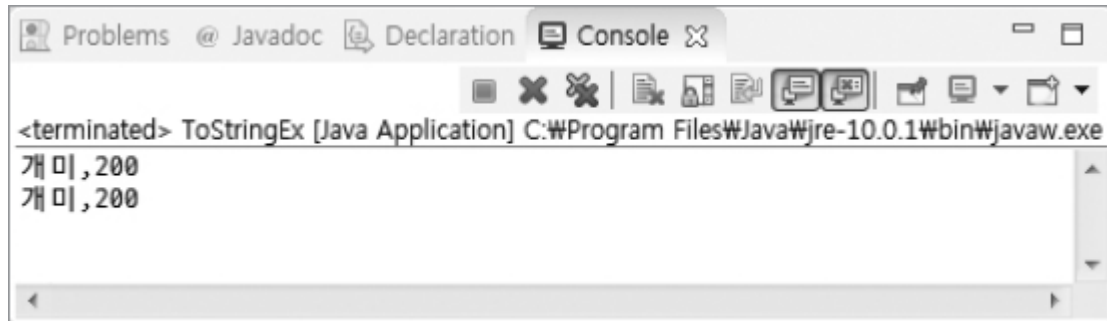
- ✓ 책의 제목과 번호를 반환 하도록 재정의

```
class Book {  
    int bookNumber;  
    String bookTitle;  
  
    Book(int bookNumber, String bookTitle) {  
        this.bookNumber = bookNumber;  
        this.bookTitle = bookTitle;  
    }  
  
    @Override  
    public String toString( ) {  
        return bookTitle + "," + bookNumber;  
    }  
}
```

toString() 메서드 재정의

toString() 메서드 재정의 테스트

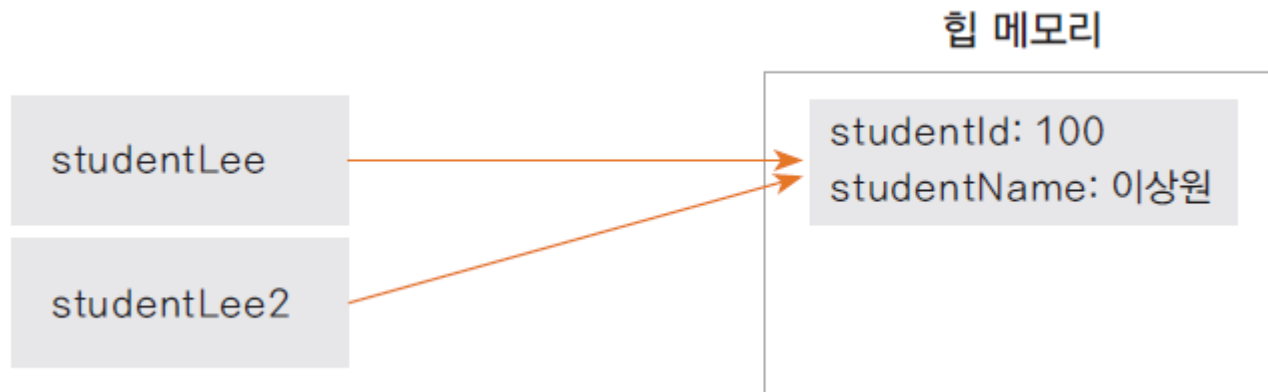
```
public class ToStringEx {  
    public static void main(String[ ] args) {  
        Book book1 = new Book(200, "개미");  
  
        System.out.println(book1);  
        System.out.println(book1.toString( ));  
    }  
}
```



equals() 메서드

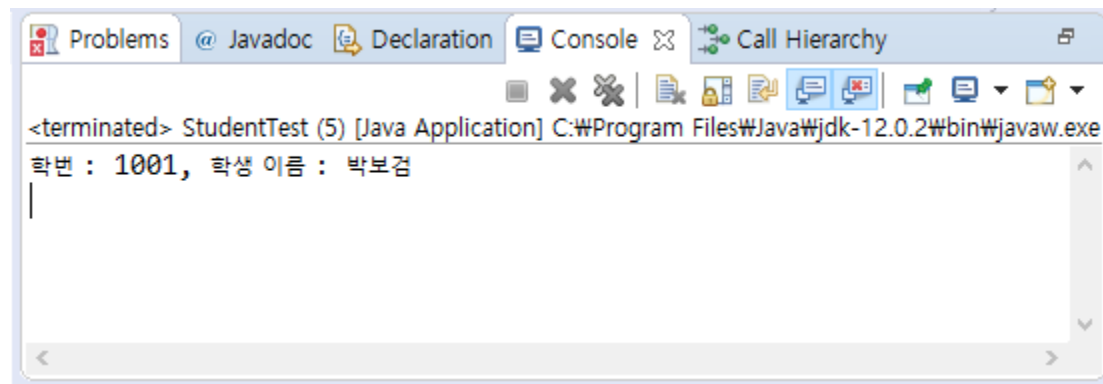
- 두 인스턴스의 주소 값을 비교하여 true/false를 반환
- 재정의하여 두 인스턴스가 논리적으로 동일함의 여부를 반환
- 같은 학번의 학생인 경우 여러 인스턴스의 주소 값은 다르지만 같은 학생으로 처리해야 학점이나 정보 산출에 문제가 생기지 않으므로 이런 경우, equals() 메서드를 재정의함

```
Student studentLee = new Student(100, "이상원");  
Student studentLee2 = studentLee; // 주소 복사
```



Quiz toString()

- Student 클래스에는 학생 이름과 학번을 멤버 변수로 가진다.
- 이 클래스에 toString() 메서드를 오버라이딩(재정의)하여 Student 클래스의 참조 변수를 출력할 때, 학생의 이름과 학번이 출력되도록 구현해 보세요.



String과 integer의 equals() 메서드

- String은 같은 문자열의 경우 true를 반환
- Integer는 정수 값이 같은 경우 true를 반환

```
public class StringEquals {  
    public static void main(String[ ] args) {  
        String str1 = new String("abc");  
        String str2 = new String("abc");
```

```
        System.out.println(str1 == str2);
```

두 인스턴스 주소 값이 같은지 비교하여 출력

```
        System.out.println(str1.equals(str2));
```

String 클래스의 equals() 메서드 사용. 두 인스턴스의 문자열 값이 같은지 비교하여 출력

```
        Integer i1 = new Integer(100);
```

```
        Integer i2 = new Integer(100);
```

```
        System.out.println(i1 == i2);
```

두 인스턴스 주소 값이 같은지 비교하여 출력

```
        System.out.println(i1.equals(i2));
```

Integer 클래스의 equals() 메서드 사용. 두 인스턴스의 정수 값이 같은지 비교하여 출력

```
    }  
}
```

:: 출력 화면

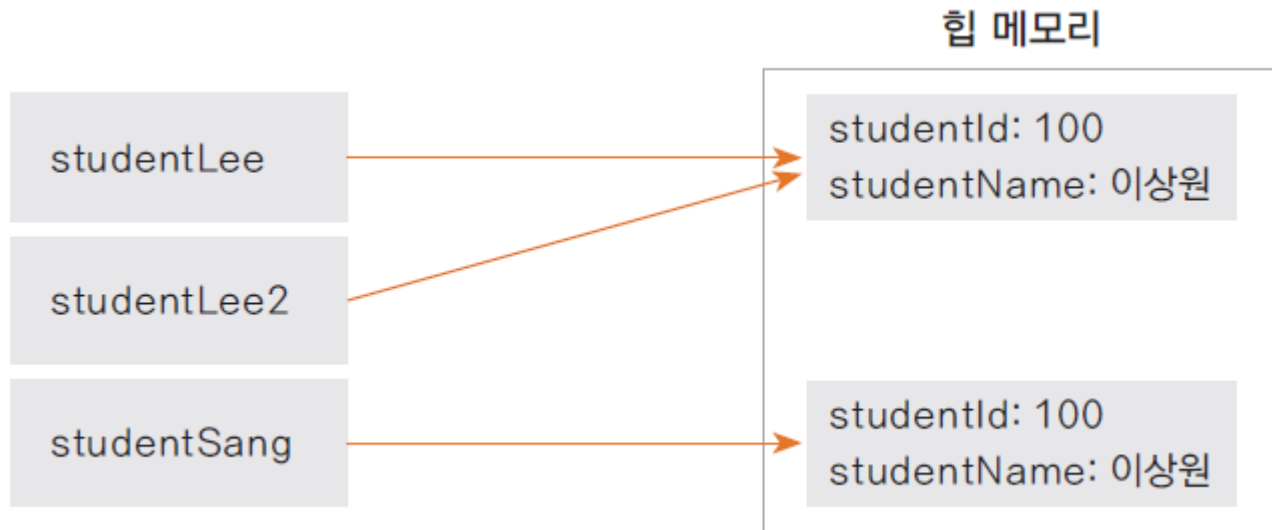
Problems @ Javadoc

<terminated> StringEquals

```
false  
true  
false  
true
```

equals() 메서드

```
Student studentLee = new Student(100, "이상원");  
Student studentLee2 = studentLee;  
Student studentSang = new Student(100, "이상원");
```



- ✓ studentLee와 studentSang은 인스턴스는 다르지만 같은 학번의 같은 학생임
- ✓ 두 학생이 논리적으로 같다는 것을 구현해야 함

equals() 메서드 재정의하기

```
class Student {  
    ...
```

```
@Override
```

```
public boolean equals(Object obj) {
```

```
    if(obj instanceof Student) {
```

```
        Student std = (Student)obj;
```

```
        if(this.studentId == std.studentId)
```

```
            return true;
```

```
        else return false;
```

```
    }
```

```
    return false;
```

```
}
```

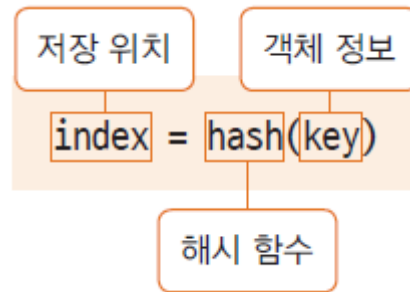
```
}
```

equals() 메서드 재정의

재정의한 equals() 메서드는 학생의
학번이 같으면 true 반환

hashCode() 메서드

- hash : 정보를 저장, 검색하기 위해 사용 하는 자료구조
- 자료의 특정 값(키 값)에 대해 저장 위치를 반환해주는 해시 함수를 사용함



- 해시 함수는 어떤 정보인가에 따라 다르게 구현됨
- hashCode() 메서드는 인스턴스의 저장 주소를 반환함
- 힙 메모리에 인스턴스가 저장되는 방식이 hash

hashCode() 메서드

- hashCode()의 반환 값
: 자바 가상머신이 저장한 인스턴스의 주소 값을 10진수로 나타냄
- 서로 다른 메모리의 두 인스턴스가 같다면???
 - 재정의된 equals() 메서드의 값이 true
 - 동일한 hashCode() 반환 값을 가져야 함
- 논리적으로 동일함을 위해 equals() 메서드를 재정의 하였다면,
hashCode() 메서드도 재정의하여 동일한 값이 반환 되도록 함
- String 클래스 : 동일한 문자열 인스턴스에 대해 동일한 정수가 반환됨
- Integer 클래스 : 동일한 정수 값의 인스턴스에 대해 같은 정수 값이 반환됨

String과 integer의 hashCode() 메서드

```
public class HashCodeTest {  
    public static void main(String[ ] args) {  
        String str1 = new String("abc");  
        String str2 = new String("abc");
```

```
        System.out.println(str1.hashCode( ));  
        System.out.println(str2.hashCode( ));
```

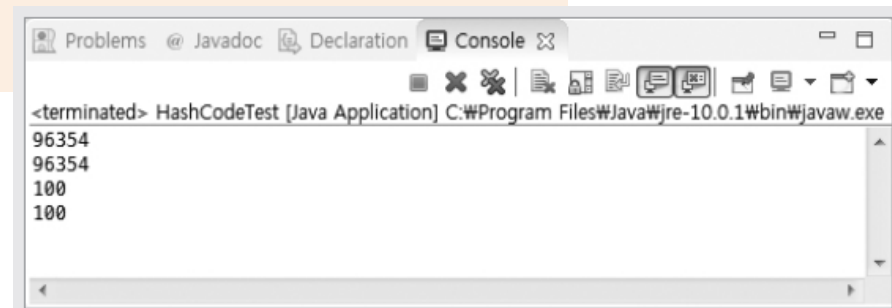
abc 문자열의 해시 코드 값 출력

```
        Integer i1 = new Integer(100);  
        Integer i2 = new Integer(100);
```

```
        System.out.println(i1.hashCode( ));  
        System.out.println(i2.hashCode( ));
```

Integer(100)의 해시 코드 값 출력

```
    }  
}
```



hashCode() 메서드 재정의하기

- ✓ 같은 학생의 경우 같은 정수 값을 반환해야 하므로 학번을 반환하도록 재정의함

```
class Student {
```

```
...
```

```
@Override
```

```
public int hashCode( ) {  
    return studentId;  
}
```

해시 코드 값으로 학번을 반환
하도록 메서드 재정의

```
}
```

```
public class EqualsTest {
```

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

```
        ...
```

```
        System.out.println("studentLee의 hashCode : " + studentLee.hashCode( ));
```

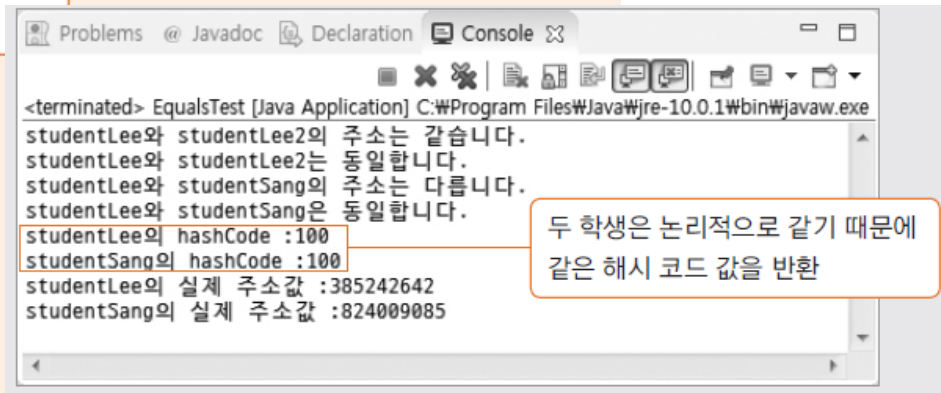
```
        System.out.println("studentSang의 hashCode : "+ studentSang.hashCode( ));
```

```
        System.out.println("studentLee의 실제 주소값 : "+ System.identityHashCode(studentLee));
```

```
        System.out.println("studentSang의 실제 주소값 : "+ System.identityHashCode(studentSang));
```

```
    }
```

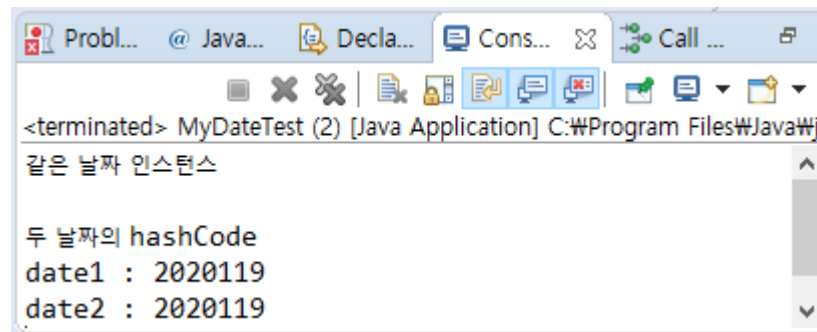
```
}
```



Quiz hashCode()

<MyDateTest.java>

- 날짜를 구현한 MyDate 클래스의 멤버 변수로는 day, month, year을 가집니다.
 - 3개의 멤버 변수를 매개변수로 갖는 생성자로 두 날짜 인스턴스 생성
- 생성한 두 인스턴스의 날짜가 같으면, “같은 날짜 인스턴스”가 출력되도록 equals() 메서드를 재정의 하세요.
- equals() 메서드를 재정의할 때 사용한 멤버 변수를 활용하여 hashCode() 메서드를 재정의 해 보세요.



clone() 메서드

- 객체의 원본 복제하는데 사용하는 메서드
- 원본을 유지해 놓고 복사본을 사용할 때
- 기본 틀(prototype)을 두고 복잡한 생성과정을 반복하지 않고 복제
- clone() 메서드를 사용하면 객체의 정보(멤버변수 값)가 같은 인스턴스가 또 생성되는 것이므로 객체지향 프로그램의 정보은닉, 객체 보호의 관점에서 위배될 수 있음
- 객체의 clone() 메서드 사용을 허용한다는 의미로 Cloneable 인터페이스를 명시해 줌

```
class Circle implements Cloneable {
```

```
    Point point;
```

```
    int radius;
```

객체를 복제해도 된다는 의미로 Cloneable 인터페이스를 함께 선언

clone() 메서드 재정의하기

```
class Circle implements Cloneable {  
    Point point;  
    int radius;  
  
    Circle(int x, int y, int radius) {  
        this.radius = radius;  
        point = new Point(x, y);  
    }  
  
    public String toString( ) {  
        return "원점은 " + point + "이고," + "반지름은 " + radius + "입니다";  
    }  
  
    @Override  
    public Object clone( ) throws CloneNotSupportedException {  
        return super.clone( );  
    }  
}
```

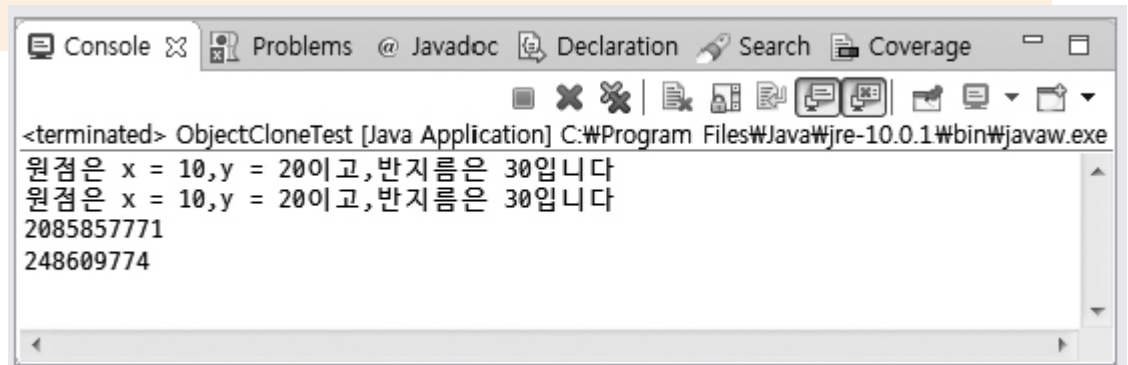
객체를 복제해도 된다는 의미로 Cloneable 인터페이스를 함께 선언

clone() 메서드를 사용할 때 발생할 수 있는 오류를 예외 처리함

clone() 메서드 재정의 테스트하기

```
public class ObjectCloneTest {  
    public static void main(String[ ] args) throws CloneNotSupportedException {  
        Circle circle = new Circle(10, 20, 30);  
        Circle copyCircle = (Circle)circle.clone( );  
  
        System.out.println(circle);  
        System.out.println(copyCircle);  
        System.out.println(System.identityHashCode(circle));  
        System.out.println(System.identityHashCode(copyCircle));  
    }  
}
```

clone() 메서드를 사용해 circle 인스턴스를 copyCircle에 복제함



The screenshot shows a Java IDE console window with the following output:

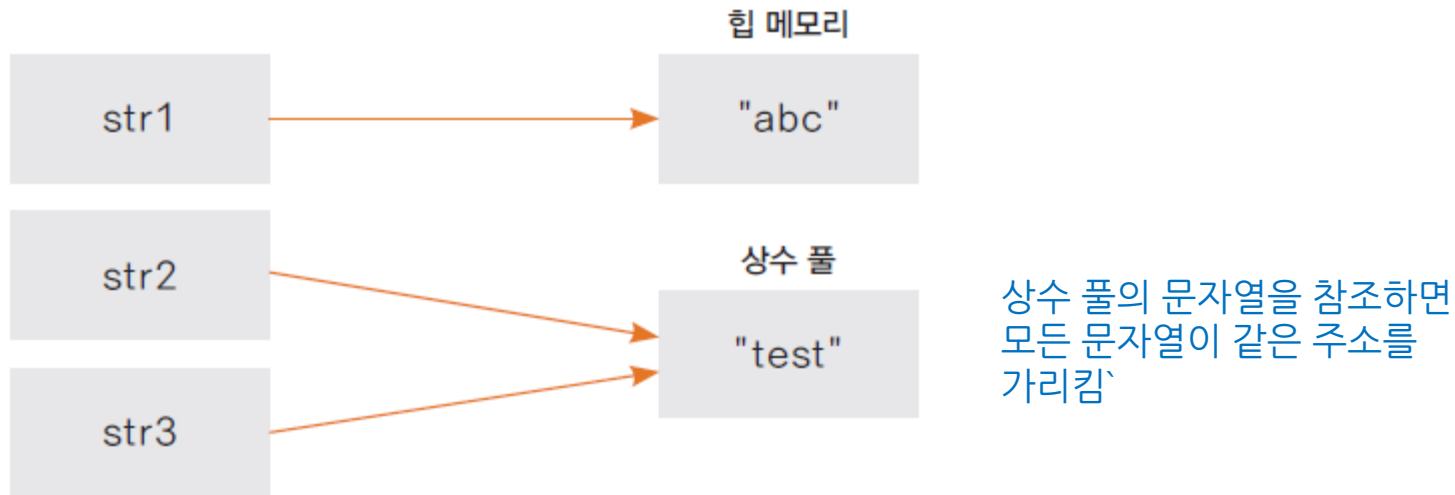
```
<terminated> ObjectCloneTest [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe  
원점은 x = 10,y = 20이고,반지름은 30입니다  
원점은 x = 10,y = 20이고,반지름은 30입니다  
2085857771  
248609774
```

String 클래스

- String을 선언하는 두 가지 방법

```
String str1 = new String("abc"); // 생성자의 매개변수로 문자열 생성  
String str2 = "test";           // 문자열 상수를 가리키는 방식
```

- 힙 메모리에 인스턴스로 생성되는 경우와 상수 풀(constant pool)에 있는 주소를 참조하는 방법 두 가지



문자열 주소 값 비교하기

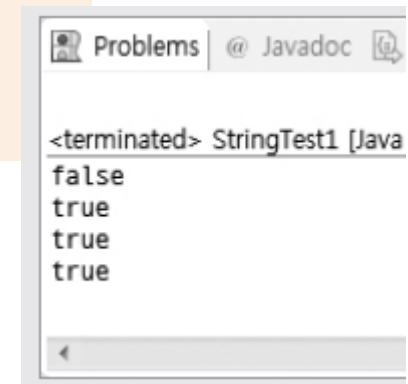
```
public class StringTest1 {  
    public static void main(String[ ] args) {  
        String str1 = new String("abc");  
        String str2 = new String("abc");  
  
        System.out.println(str1 == str2);  
        System.out.println(str1.equals(str2));  
  
        String str3 = "abc";  
        String str4 = "abc";  
  
        System.out.println(str3 == str4);  
        System.out.println(str3.equals(str4));  
    }  
}
```

인스턴스가 매번 새로 생성되므로 str1과 str2의 주소 값이 다름

문자열 값은 같으므로 true 반환

문자열 abc는 상수 풀에 저장되어 있으므로 str3과 str4가 가리키는 주소 값이 같음

문자열 값도 같으므로 true 반환



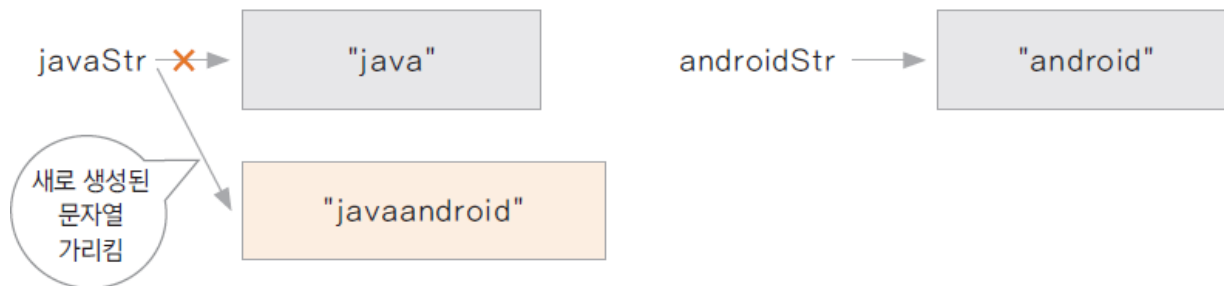
String 클래스로 문자열 연결

- 한 번 생성된 String 값(문자열)은 불변 (immutable)
- 두 개의 문자열을 연결하면 새로운 인스턴스가 생성됨
- 문자열 연결을 계속하면 메모리에 garbage가 많이 생길 수 있음

```
String javaStr = new String("java");  
String androidStr = new String("android");  
System.out.println(javaStr);  
System.out.println("처음 문자열 주소 값: " + System.identityHashCode(javaStr));
```

```
javaStr = javaStr.concat(androidStr);
```

문자열 javaStr와 문자열 androidStr를
연결하여 javaStr에 대입



StringBuilder, StringBuffer 사용하기

- 내부적으로 가변적인 char[] 배열을 가지고 있는 클래스
- 문자열을 여러 번 연결하거나 변경할 때 사용하면 유용함
- 매번 새로 생성하지 않고 기존 배열을 변경하므로 garbage가 생기지 않음
- StringBuffer는 멀티 쓰레드 프로그래밍에서 동기화(synchronization)를 보장
- 단일 쓰레드 프로그램에서는 StringBuilder를 사용하기를 권장
- toString() 메서드로 String 반환

StringBuilder 테스트

```
public class StringBuilderTest {  
    public static void main(String[] args) {  
        String javaStr = new String("Java");  
        System.out.println("javaStr 문자열 주소 :" + System.identityHashCode(javaStr));  
  
        StringBuilder buffer = new StringBuilder(javaStr);  
        System.out.println("연산 전 buffer 메모리 주소:" + System.identityHashCode(buffer));  
  
        buffer.append(" and");  
        buffer.append(" android");  
        buffer.append(" programming is fun!!!");  
        System.out.println("연산 후 buffer 메모리 주소:" + System.identityHashCode(buffer));  
  
        javaStr = buffer.toString();  
        System.out.println(javaStr);  
        System.out.println("연결된 javaStr 문자열 주소 :" + System.identityHashCode(javaStr));  
    }  
}
```

인스턴스가 처음 생성됐을 때
메모리 주소

String으로부터 StringBuilder 생성

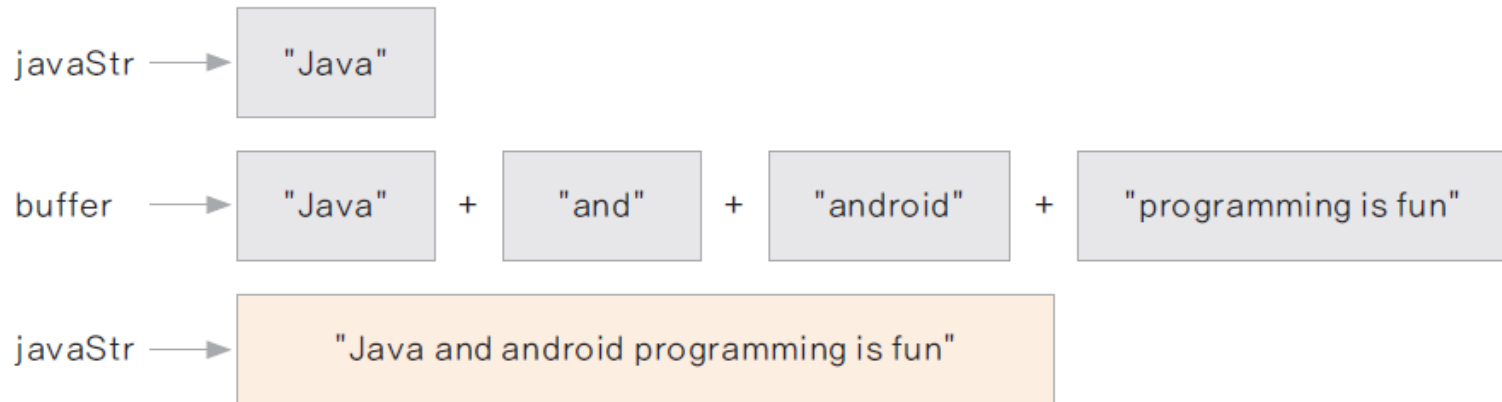
문자열 추가

String 클래스로 반환

```
<terminated> StringBuilderTest [Java Application] C:\WProgram  
javaStr 문자열 주소 :385242642  
연산 전 buffer 메모리 주소:824009085  
연산 후 buffer 메모리 주소:824009085  
Java and android programming is fun!!!  
새로 만들어진 javaStr 문자열 주소 :2085857771
```

StringBuilder 테스트

❖ 메모리 구조



Wrapper 클래스

- 기본 자료형(primitive data type)에 대한 클래스

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

오토박싱(autoboxing)과 언박싱(unboxing)

- Integer는 객체, int는 4바이트 기본 자료형
- 두 개의 자료를 같이 연산할 때, 자동으로 변환이 일어남

```
Integer num = new Integer(100);  
int num2 = 200;  
int sum = num + num2;  
Integer num3 = num2;
```

num.intValue()로 변환(언박싱)

Integer.valueOf(num2)로 변환(오토박싱)

- 오토박싱(autoboxing) : 기본형을 객체형으로 바꾸는 것
- 언박싱(unboxing) : 객체형을 기본형으로 꺼내는 것

Class 클래스

- 자바의 모든 클래스와 인터페이스는 컴파일 후 class 파일로 생성됨
- class 파일에는 객체의 정보(멤버변수, 메서드, 생성자 등)가 포함되어 있음
- Class 클래스는 컴파일된 class 파일에서 객체의 정보를 가져올 수 있음

Class 클래스 가져오기

1. Object 클래스의 getClass() 메서드 사용하기

```
String s = new String( );  
Class c = s.getClass( ); //getClass( ) 메서드의 반환형은 Class
```

2. 클래스 파일 이름을 Class 변수에 직접 대입하기

```
Class c = String.Class;
```

3. Class.forName(“클래스 이름”) 메서드 사용하기

```
Class c = Class.forName("java.lang.String");
```

class.forName() 메서드로 동적 로딩하기

- 동적 로딩(dynamic loading)이란 ?
컴파일 시에 데이터 타입이 모두 binding되어 자료형이 로딩되는(static loading) 것이 아니라 실행 중에 데이터 타입을 알고 binding 되는 방식
- 프로그래밍할 때는 어떤 클래스를 사용할지 모를 때 변수로 처리하고, 실행될 때 해당 변수에 대입된 값의 클래스가 실행될 수 있도록 Class 클래스에서 제공하는 static 메서드
- 실행 시에 로딩되므로 경우에 따라 다른 클래스가 사용될 수 있어 유용함
- 컴파일 타임에 체크할 수 없으므로 해당 문자열에 대한 클래스가 없는 경우 예외 (ClassNotFoundException)가 발생할 수 있음

```
String className = "classex.Person"  
Class pClass = Class.forName(className);
```


Class 클래스 가져오기 예제

```
package classex;
```

```
public class ClassTest {
```

```
    public static void main(String[ ] args) throws ClassNotFoundException {
```

```
        Person person = new Person( );
```

```
        Class pClass1 = person.getClass( );
```

```
        System.out.println(pClass1.getName( ));
```

```
        Class pClass2 = Person.class;
```

```
        System.out.println(pClass2.getName( ));
```

```
        Class pClass3 = Class.forName("classex.Person");
```

```
        System.out.println(pClass3.getName( ));
```

```
    }
```

```
}
```

forName() 메서드에서 발생하는 예외를 처리함. 이름과 일치하는 클래스가 없는 경우 ClassNotFoundException 발생

Object의 getClass() 메서드 사용하기

직접 class 파일 대입하기

클래스 이름으로 가져오기

```
<terminated> ClassTest [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe
classex.Person
classex.Person
classex.Person
```

Class 클래스로 정보 가져오기

- Reflection(리플렉션) 프로그래밍 :
Class 클래스를 이용하여 클래스의 정보(생성자, 멤버 변수, 메서드)를 가져오고,
이를 활용하여 인스턴스를 생성하고 메서드를 호출하는 등의 프로그래밍 방식
- 로컬 메모리에 객체가 없어서 객체의 데이터 타입을 직접 알 수 없는 경우(원격에 객체가 있는 경우 등) 객체 정보만을 이용하여 프로그래밍할 수 있음
- Constructor, Method, Filed 등 java.lang.reflect 패키지에 있는 클래스들을 활용하여 프로그래밍
- 일반적으로 자료형을 알 수 있는 경우에는 사용하지 않음

Class 클래스 메서드 활용하기 예제

```
import java.lang.reflect.Constructor;  
import java.lang.reflect.Field;  
import java.lang.reflect.Method;
```

```
public class StringClassTest {  
    public static void main(String[] args) throws ClassNotFoundException {  
        Class strClass = Class.forName("java.lang.String");
```

클래스 이름으로 가져오기

```
        Constructor[] cons = strClass.getConstructors();  
        for(Constructor c : cons) {
```

모든 생성자 가져오기

```
            System.out.println(c);  
        }
```

```
        System.out.println();  
        Field[] fields = strClass.getFields();  
        for(Field f : fields){
```

모든 멤버 변수(필드) 가져오기

```
            System.out.println(f);  
        }
```

```
        System.out.println();  
        Method[] methods = strClass.getMethods();  
        for(Method m : methods){
```

모든 메서드 가져오기

```
            System.out.println(m);  
        }  
    }  
}
```

```
<terminated> StringClassTest [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe  
public java.lang.String(byte[])  
public java.lang.String(byte[],int,int)  
public java.lang.String(byte[],java.nio.charset.Charset)  
public java.lang.String(byte[],java.lang.String) throws java.io.UnsupportedEncodingException  
public java.lang.String(byte[],int,int,java.nio.charset.Charset)  
public java.lang.String(java.lang.StringBuilder)  
public java.lang.String(java.lang.StringBuffer)  
public java.lang.String(char[],int,int)  
public java.lang.String(char[])  
public java.lang.String(java.lang.String)  
public java.lang.String()  
public java.lang.String(byte[],int,int,java.lang.String) throws java.io.UnsupportedEncodingException  
public java.lang.String(byte[],int)  
public java.lang.String(byte[],int,int,int)  
public java.lang.String(int[],int,int)  
  
public static final java.util.Comparator java.lang.String.CASE_INSENSITIVE_ORDER;
```