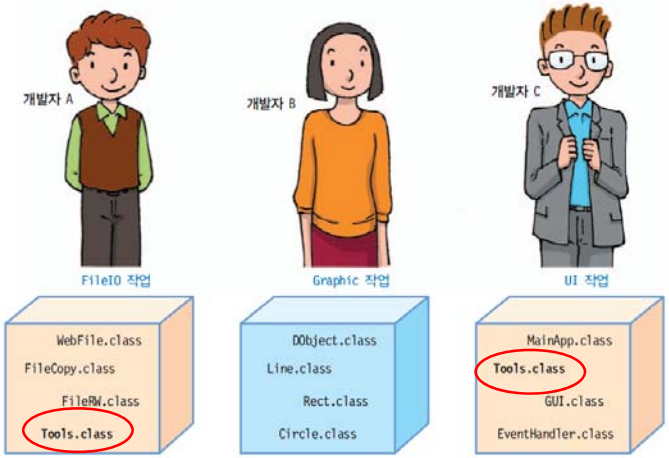
패키지, 기본패키지

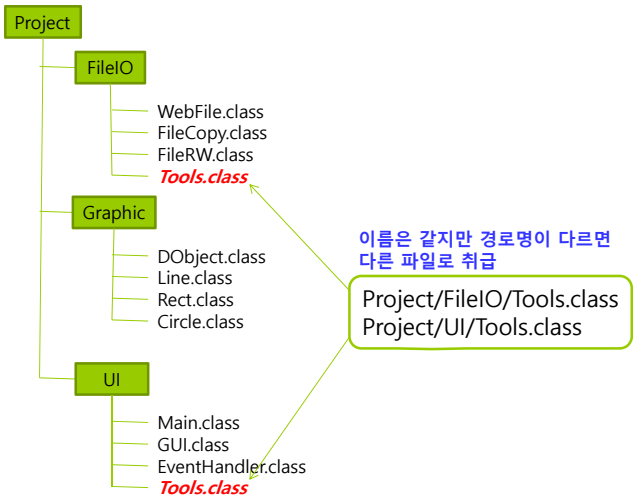
514760
2020년 봄학기
5/12/2020
박경신

패키지 개념과 필요성

3명이 분담하여 자바 응용프로그램을 개발하는 경우,
동일한 이름의 클래스가 존재할 가능성 있음 -> 합칠 때 오류발생



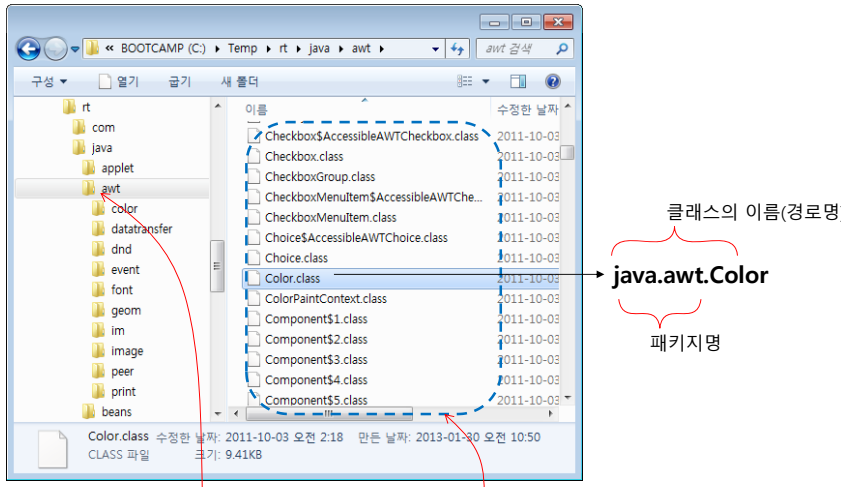
디렉터리로 각 개발자의 코드 관리(패키지)



자바의 패키지 (package)

- 패키지란
  - 서로 관련된 클래스와 인터페이스의 컴파일 된 클래스 파일들을 하나의 디렉터리에 묶어 놓은 것
- 하나의 응용프로그램은 여러 개의 패키지로 작성 가능
  - 하나의 패키지로 만들고 모든 클래스 파일을 넣어 둘 수도 있음
- 패키지는 jar 파일로 압축할 수 있음
  - 예) JDK에서 제공하는 표준 패키지는 rt.jar에 압축

JDK에서 제공되는 패키지



패키지 사용하기, import문

- 다른 패키지에 작성된 클래스 사용
  - import를 이용하지 않는 경우
    - 소스 내에서 패키지 이름과 클래스 이름의 전체 경로명을 써주어야 함

```
public class ImportExample {
   public static void main(String[] args) {
      java.util.Scanner scanner =
            new java.util.Scanner(System.in);
   }
}
```

  - import 키워드 이용하는 경우
    - 소스의 시작 부분에 사용하려는 패키지 명시
      - 소스에는 클래스 명만 명시하면 됨
    - 특정 클래스의 경로명만 포함하는 경우
      - import java.util.Scanner;
    - 패키지 내의 모든 클래스를 포함시키는 경우
      - import java.util.*;
      - *는 현재 패키지 내의 클래스만을 의미하며 하위 패키지의 클래스까지 포함하지 않는다.

```
import java.util.Scanner;
public class ImportExample {
   public static void main(String[] args) {
      Scanner scanner =
            new Scanner(System.in);
   }
}
```

```
import java.util.*;
public class ImportExample {
   public static void main(String[] args) {
      Scanner scanner =
            new Scanner(System.in);
   }
}
```

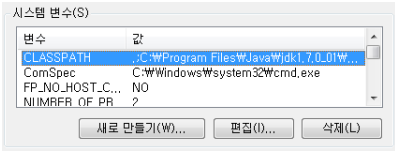
정적 import 문장

- 클래스 안에 정의된 정적 상수나 정적 메소드를 사용하는 경우에 정적 import 문장을 사용하면 클래스 이름을 생략하여도 된다.

```
import static java.lang.Math.*;
double r = cos(PI * theta);
```

클래스 경로

- 클래스의 위치(경로) 지정
  - 클래스 탐색 경로를 지정하는 방법 2가지 - JVM은 항상 현재 작업 디렉토리부터 찾는다.
    1. 시스템 환경 변수 CLASSPATH에 설정된 디렉토리에서 찾는다.

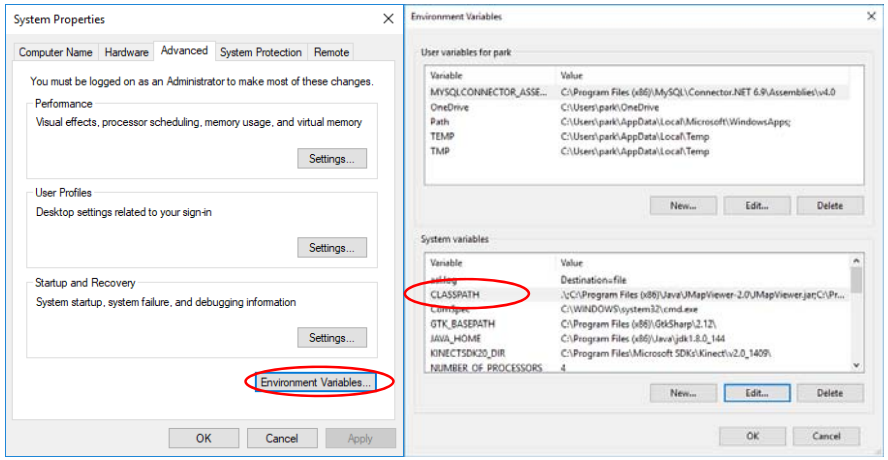


    2. Java의 옵션 –classpath를 사용할 수 있다.

```
C:₩> java -classpath C:₩classes;C:₩lib;. library.Rectangle
```

      - 실행 시 클래스 파일이 존재하는 패키지 디렉터리 정보를 -classpath 옵션에 지정

CLASSPATH 지정 방법



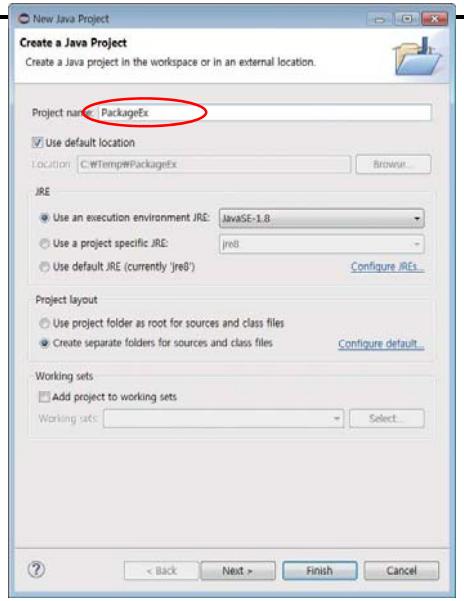
이클립스에서 쉽게 패키지 만들기

- 예제 소스 코드

```
abstract class Calculator {
  public abstract int add(int a, int b);// 두 정수의 합을 구하여 리턴
  public abstract int subtract(int a, int b);// 두 정수의 차를 구하여 리턴
  public abstract double average(int[] a);// 배열에 저장된 정수의 평균을 구해 실수로 반환
}
class GoodCalc extends Calculator {
  public int add(int a, int b) {   return a+b;}
  public int subtract(int a, int b) {     return a - b;  }
  public double average(int[] a) {
    double sum = 0;
    for (int i = 0; i < a.length; i++)
      sum += a[i];
    return sum/a.length;
  }
  public static void main(String [] args) {
    Calculator c = new GoodCalc();
    System.out.println(c.add(2,3));
    System.out.println(c.subtract(2,3));
    System.out.println(c.average(new int [] {2,3,4 }));
  }
}
```

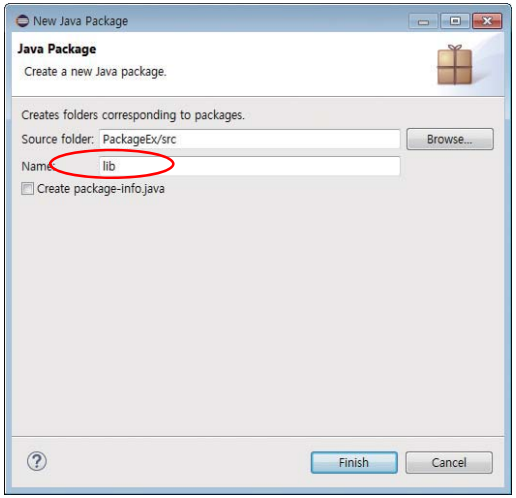
이클립스에서 쉽게 패키지 만들기

- 프로젝트 작성



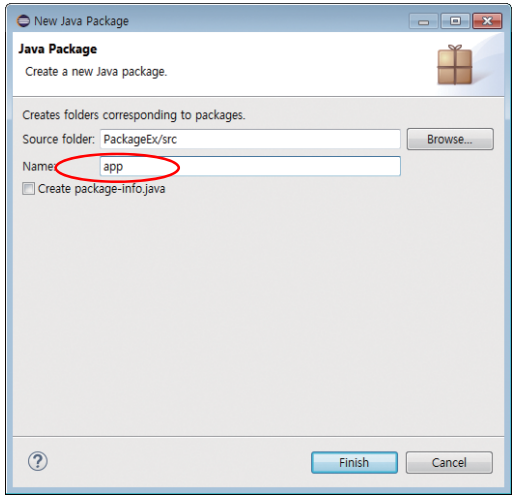
이클립스에서 쉽게 패키지 만들기

- 패키지 lib 작성



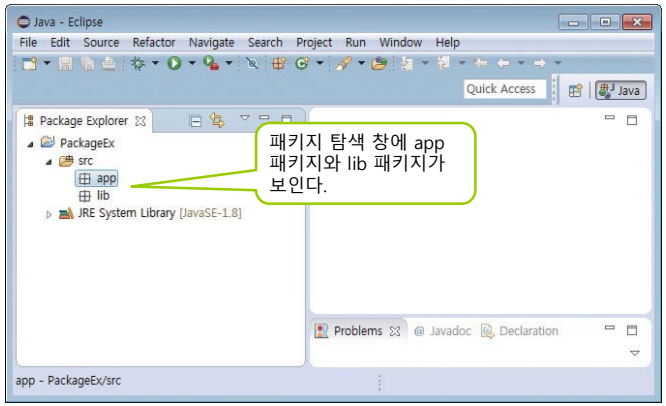
이클립스에서 쉽게 패키지 만들기

- 패키지 app 작성



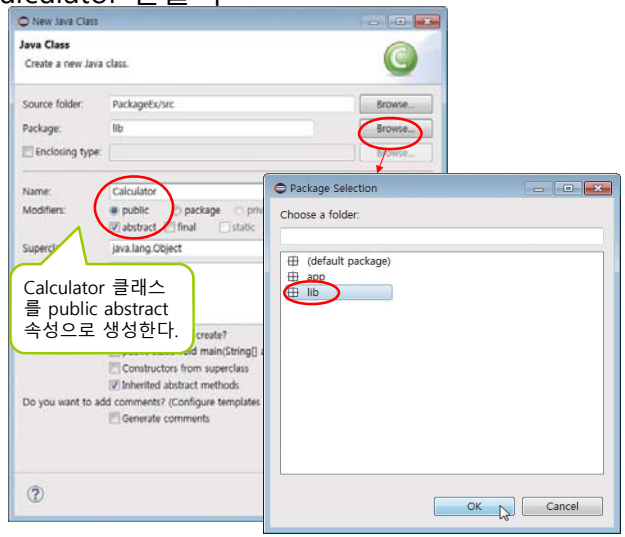
이클립스에서 쉽게 패키지 만들기

- 패키지 작성이 완료된 결과



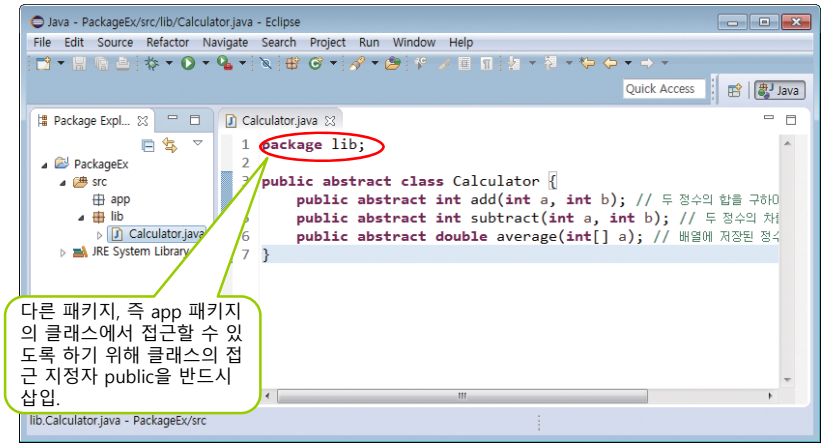
이클립스에서 쉽게 패키지 만들기

- 클래스 Calculator 만들기



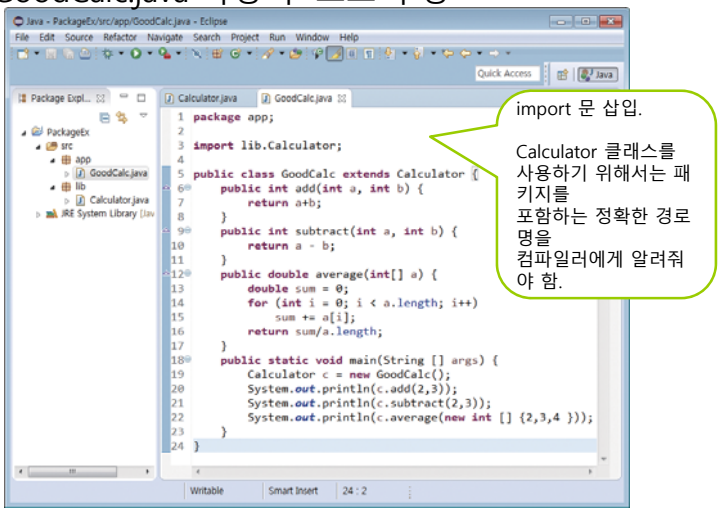
이클립스에서 쉽게 패키지 만들기

- Calculator 클래스의 소스 수정



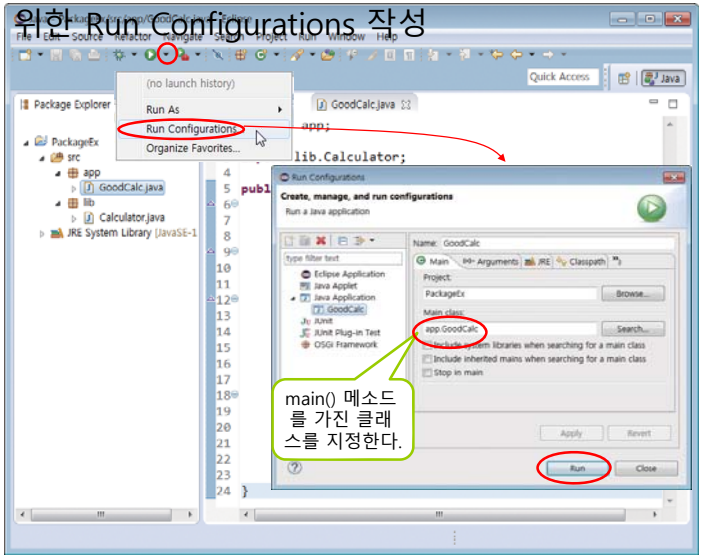
이클립스에서 쉽게 패키지 만들기

- GoodCalc.java 작성 후 소스 수정



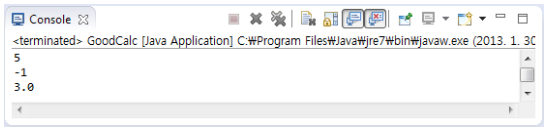
이클립스에서 쉽게 패키지 만들기

- 실행을 위한 Run Configurations 작성



이클립스에서 쉽게 패키지 만들기

- 프로젝트 PackageEx 실행

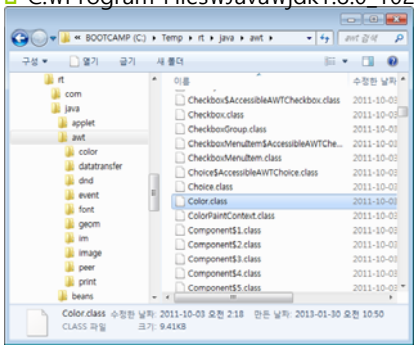


패키지의 특징

- 패키지의 특징
  - 패키지 계층구조
    - 클래스나 인터페이스가 너무 많아지면 관리의 어려움
    - 관련된 클래스 파일을 하나의 패키지로 계층화하여 관리 용이
  - 패키지별 접근 제한
    - default로 선언된 클래스나 멤버는 동일 패키지 내의 클래스들이 자유롭게 접근하도록 허용
  - 동일한 이름의 클래스와 인터페이스의 사용 가능
    - 서로 다른 패키지에 이름이 같은 클래스와 인터페이스 존재 가능
  - 높은 소프트웨어 재사용성
    - 오라클에서 제공하는 자바 API는 패키지로 구성되어 있음
    - java.lang, java.io 등의 패키지들 덕분에 일일이 코딩하지 않고 입출력 프로그램을 간단히 작성할 수 있음

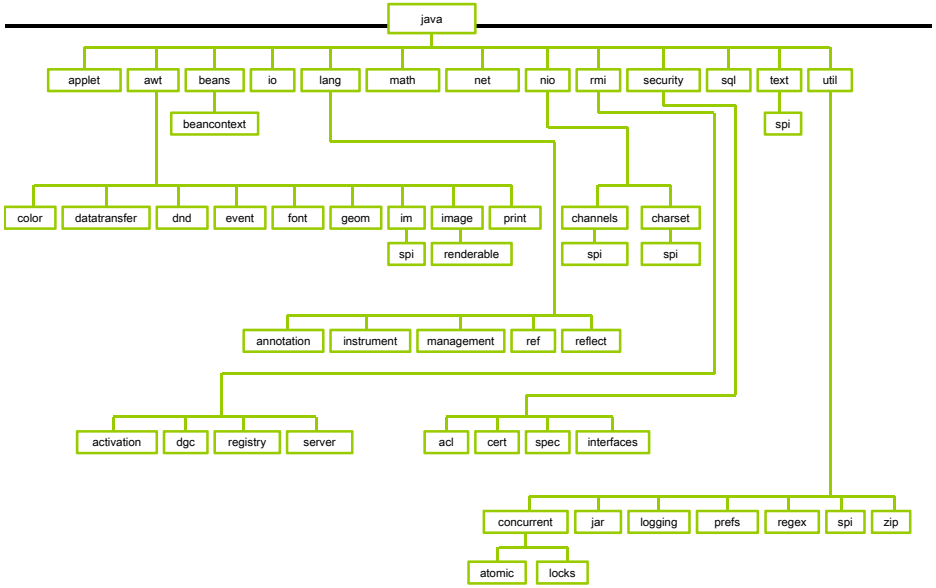
자바 JDK의 패키지 구조

- JDK 패키지
  - 자바에서는 관련된 클래스들을 표준 패키지로 묶어 사용자에게 제공
  - 자바에서 제공하는 패키지는 C언어의 표준 라이브러리와 유사
  - JDK의 표준 패키지는 rt.jar에 담겨 있음
    - C:₩Program Files₩Java₩jdk1.8.0_102₩jre₩lib₩rt.jar



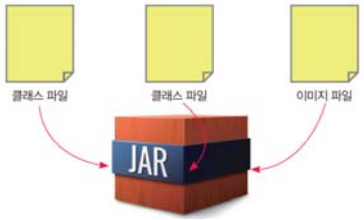
rt.jar의 java.awt 패키지에 컴파일된 클래스들이 들어있다.

자바 패키지 구조



JAR 압축 파일

- JAR 파일은 여러 개의 클래스 파일을 디렉토리 계층 구조를 유지한 채로 압축하여서 가지고 있을 수 있다.



- Jar 파일을 생성하는 방법 예시

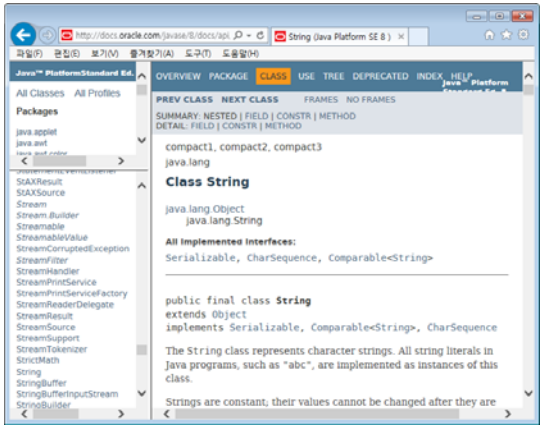
```
c> jar cvf Game.jar *.class icon.png
```

주요 패키지

- java.lang
  - 자바 language 패키지
    - 스트링, 수학 함수, 입출력 등 자바 프로그래밍에 필요한 기본적인 클래스와 인터페이스
  - 자동으로 import 됨 - import 문 필요 없음
- java.util
  - 자바 유틸리티 패키지
    - 날짜, 시간, 벡터, 해시맵 등과 같은 다양한 유틸리티 클래스와 인터페이스 제공
- java.io
  - 키보드, 모니터, 프린터, 디스크 등에 입출력을 할 수 있는 클래스와 인터페이스 제공
- java.awt
  - 자바 GUI 프로그래밍을 위한 클래스와 인터페이스 제공
- javax.swing
  - 자바 GUI 프로그래밍을 위한 스윙 패키지

자바 API 참조

- 자바 API의 상세 정보
  - Oracle Technology Network(http://docs.oracle.com/javase/8/docs/api/)에서 온라인제공



Object 클래스

- 특징
  - java.lang 패키지에 포함
  - 자바 클래스 계 층 구조의 최상위에 위치
  - 모든 클래스의 수퍼 클래스
- 주요 메소드

메소드	설명
protected Object clone()	현 객체와 똑같은 객체를 만들어 리턴
boolean equals(Object obj)	obj가 가리키는 객체와 현재 객체가 비교하여 같으면 true 리턴
Class getClass()	현 객체의 런타임 클래스를 리턴
int hashCode()	현 객체에 대한 해시 코드 값 리턴
String toString()	현 객체에 대한 스트링 표현을 리턴
void notify()	현 객체에 대해 대기하고 있는 하나의 스레드를 깨운다.
void notifyAll()	현 객체에 대해 대기하고 있는 모든 스레드를 깨운다.
void wait()	다른 스레드가 깨울 때까지 현재 스레드를 대기하게 한다.

Object의 메소드 활용 예

```
class Point {
  int x, y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
public class ObjectProperty {
  public static void main(String [] args) {
    Point p = new Point(2,3);
    System.out.println(p.getClass().getName());
    System.out.println(p.hashCode());
    System.out.println(p.toString());
    System.out.println(p);
  }
}
```

```
Point
12677476
Point@c17164
Point@c17164
```

객체를 문자열로 변환

- String toString()
  - 객체를 문자열로 반환
  - Object 클래스에 구현된 toString()이 반환하는 문자열
    - 클래스 이름@객체의 hash code
  - 각 클래스는 toString()을 오버라이딩하여 자신만의 문자열 리턴 가능
- 컴파일러에 의한 자동 변환
  - '객체 + 문자열' -> '객체.toString() + 문자열'로 자동 변환

```
Point a = new Point(2,3);
String s = a + "점";
System.out.println(s);
```

변환

```
Point a = new Point(2,3);
String s = a.toString()+ "점";
System.out.println(s.toString());
```

```
Point@c17164점
```

새로운 toString() 만들기

```
class Point {
  int x, y;
  public Point(int x, int y) {
    this.x = x; this.y = y;
  }
  public String toString() {
    return "Point(" + x + "," + y+ ")";
  }
}
public class ObjectProperty {
  public static void main(String [] args) {
    Point a = new Point(2,3);
    System.out.println(a.toString());
  }
}
```

System.out.println(a); 라고 해도 동일

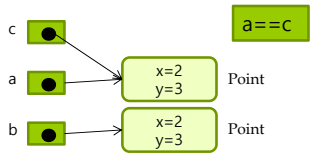
Point(2,3)

객체 비교(==과 equals())

- 객체 레퍼런스의 동일성 비교 == 연산자 이용
- 객체 내용(즉, 서로 다른 두 객체가 같은 내용물인지) 비교
  - boolean equals(Object obj) 이용

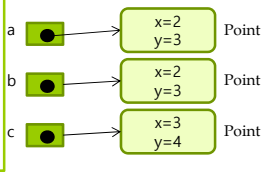
```
class Point {
  int x, y;
  public Point(int x, int y) {
    this.x = x; this.y = y;
  }
}
```

```
Point a = new Point(2,3);
Point b = new Point(2,3);
Point c = a;
if(a == b) // false
  System.out.println("a==b");
if(a == c) // true
  System.out.println("a==c");
```




```
class Point {
  int x, y;
  public Point(int x, int y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Point p){
    if(x == p.x && y == p.y)
      return true;
    else
      return false;
  }
}
```

```
Point a = new Point(2,3);
Point b = new Point(2,3);
Point c =  new Point(3,4);
if(a == b) // false
  System.out.println("a==b");
if(a.equals(b)) // true
  System.out.println("a is equal to b");
if(a.equals(c)) // false
  System.out.println("a is equal to c");
```



a is equal to b

예제 : Rect 클래스 만들고 equals() 만들기

int 타입의 width, height의 필드를 가지는 Rect 클래스를 작성하고, 두 Rect 객체의 width, height 필드에 의해 구성되는 면적이 같으면 두 객체가 같은 것으로 판별하도록 equals()를 작성하라. Rect 생성자에서 width, height 필드를 인자로 받아 초기화한다.

```
class Rect {
  int width;
  int height;
  public Rect(int width, int height) {
    this.width = width;
    this.height = height;
  }
  public boolean equals(Rect p) {
    if (width*height ==
p.width*p.height)
      return true;
    else
      return false;
  }
}
```

```
public class EqualsEx {
  public static void main(String[] args) {
    Rect a = new Rect(2,3);
    Rect b = new Rect(3,2);
    Rect c = new Rect(3,4);
    if(a.equals(b))  System.out.println("a is equal
to b");
    if(a.equals(c))  System.out.println("a is equal
to c");
    if(b.equals(c))  System.out.println("b is
equal to c");
  }
}
```

a is equal to b

Wrapper 클래스

- 자바의 기본 타입을 클래스화한 8개 클래스

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

- 용도
  - 기본 타입의 값을 사용할 수 없고 객체만 사용하는 컬렉션 등에 기본 타입의 값을 Wrapper 클래스 객체로 만들어 사용

Wrapper 객체 생성

- 기본 타입의 값을 인자로 Wrapper 클래스 생성자 호출

```
Integer i = new Integer(10);
Character c = new Character('c');
Float f = new Float(3.14);
Boolean b = new Boolean(true);
```

- 데이터 값을 나타내는 문자열을 생성자 인자로 사용

```
Boolean b = new Boolean("false");
Integer I = new Integer("10");
Double d = new Double("3.14");
```

- Float는 double 타입의 값을 생성자의 인자로 사용

```
Float f = new Float((double) 3.14);
```

주요 메소드

- 가장 많이 사용하는 Integer 클래스의 주요 메소드

메소드	설명
static int bitCount(int i)	인자 i의 이진수 표현에서 1의 개수를 리턴
float floatValue()	float 타입으로 변환된 값 리턴
int intValue()	int 타입으로 변환된 값 리턴
long longValue()	long 타입으로 변환된 값 리턴
short shortValue()	short 타입으로 변환된 값 리턴
static int parseInt(String s)	스트링 s를 10진 정수로 변환된 값 리턴
static int parseInt(String s, int radix)	스트링 s를 지정된 진법의 정수로 변환된 값 리턴
static String toBinaryString(int i)	인자 i를 이진수 표현으로 변환된 스트링 리턴
static String toHexString(int i)	인자 i를 16진수 표현으로 변환된 스트링 리턴
static String toOctalString(int i)	인자 i를 8진수 표현으로 변환된 스트링 리턴
static String toString(int i)	인자 i를 스트링으로 변환하여 리턴

Wrapper 활용

- Wrapper 객체로부터 기본 데이터 타입 알아내기

```
Integer i = new Integer(10);
int ii = i.intValue(); // ii = 10

Character c = new Character('c' );
char cc = c.charValue(); // cc = 'c'
```

```
Float f = new Float(3.14);
float ff = f.floatValue(); // ff = 3.14

Boolean b = new Boolean(true);
// bb = true
boolean bb = b.booleanValue();
```

- 문자열을 기본 데이터 타입으로 변환

```
int i = Integer.parseInt("123"); // i = 123
boolean b = Boolean.parseBoolean("true"); // b = true
float f = Float.parseFloat("3.141592" ); // f = 3.141592
```

Wrapper 활용

- 기본 데이터 타입을 문자열로 변환

```
// 정수 123을 문자열 "123" 으로 변환
String s1 = Integer.toString(123);

// 정수 123을 16진수의 문자열 "7b"로 변환
String s2 = Integer.toHexString(123);

// 실수 3.141592를 문자열 "3.141592"로 변환
String s3 = Float.toString(3.141592f);

// 문자 'a'를 문자열 "a"로 변환
String s4 = Charater.toString('a');

// 불린 값 true를 문자열 "true"로 변환
String s5 = Boolean.toString(true);
```

예제 : Wrapper 클래스 활용

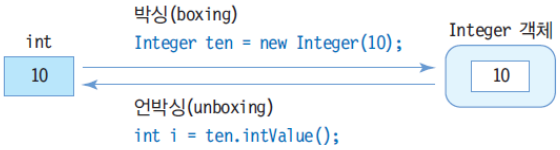
다음은 Wrapper 클래스를 활용하는 예이다. 다음 프로그램의 결과는 무엇인가?

```
public class WrapperClassEx {
  public static void main(String[] args) {
    Integer i = new Integer(10);
    char c = '4';
    Double d = new Double(3.1234566);
    System.out.println(Character.toLowerCase('A'));
    if (Character.isDigit(c))
      System.out.println(Character.getNumericValue(c));
    System.out.println(Integer.parseInt("-123"));
    System.out.println(Integer.toBinaryString(28));
    System.out.println(Integer.toHexString(28));
    System.out.println(i.doubleValue());
    System.out.println(d.toString());
    System.out.println(Double.parseDouble("44.13e-6"));
  }
}
```

```
a
4
-123
16
11100
3
1c
10.0
3.1234566
4.413E-5
```

박싱과 언박싱

- 박싱(boxing)
  - 기본 타입의 값을 Wrapper 객체로 변환하는 것
- 언박싱(unboxing)
  - Wrapper 객체에 들어 있는 기본 타입의 값을 빼내는 것



자동박싱/자동언박싱

- JDK 1.5부터 지원
- 자동 박싱(Auto boxing)
  - 기본 타입의 값을 자동으로 Wrapper 객체로 변환
- 자동 언박싱(Auto unboxing)
  - Wrapper 객체를 자동으로 기본 타입 값으로 변환

```
Integer ten = 10; // 자동 박싱. 10 -> new Integer(10)으로 자동 박싱
int i = ten; // 자동 언박싱. ten -> ten.getIntValue();로 자동 언박싱
```

예제 : 박싱 언박싱의 예

다음 코드에 대한 결과는 무엇인가?

```
public class AutoBoxingUnBoxing {
  public static void main(String[] args) {
    int i = 10;
    Integer intObject = i; // auto boxing
    System.out.println("intObject = " + intObject);

    i = intObject + 10; // auto unboxing
    System.out.println("i = " + i);
  }
}
```

```
intObject = 10
i = 20
```

String의 생성과 특징

- String - java.lang.String
  - String 클래스는 하나의 스트링만 표현

```
// 스트링 리터럴로 스트링 객체 생성
String str1 = "abcd";

// String 클래스의 생성자를 이용하여 스트링 생성
char data[] = {'a', 'b', 'c', 'd'};
String str2 = new String(data);
String str3 = new String("abcd"); // str2와 str3은 모두 "abcd" 스트링
```

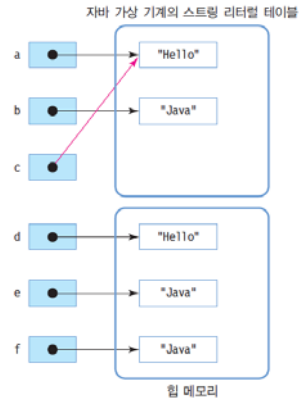
  - String 생성자

생성자	설명
String()	빈 스트링 객체 생성
String(char[] value)	문자 배열에 포함된 문자들을 스트링 객체로 생성
String(String original)	인자로 주어진 스트링과 똑같은 스트링 객체 생성
String(StringBuffer buffer)	스트링 버퍼에 포함된 문자들을 스트링 객체로 생성

스트링 리터럴과 new String()

- 스트링 생성
  - 단순 리터럴로 생성, String s = "Hello";
    - JVM이 리터럴 관리, 응용프로그램 내에서 공유됨
  - String 객체로 생성, String t = new String("Hello");
    - 힙에 String 객체 생성

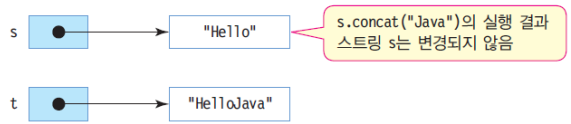
```
String a = "Hello";
String b = "Java";
String c = "Hello";
String d = new String("Hello");
String e = new String("Java");
String f = new String("Java");
```



스트링 객체의 주요 특징

- 스트링 객체는 수정 불가능

```
String s = new String("Hello");
String t = s.concat("Java"); // 스트링 s에 "Java"를 덧붙인 스트링을 리턴함.
```



- ==과 equals()
  - 두 스트링을 비교할 때 반드시 equals()를 사용하여야 함
    - equals()는 내용을 비교하기 때문

String 클래스 주요 메소드

메소드	설명
char charAt(int index)	지정된 index 인덱스에 있는 문자 값 리턴
int codePointAt(int index)	지정된 index 인덱스에 있는 유니코드 값 리턴
int compareTo(String anotherString)	두 스트링을 사전적 순서를 기준으로 비교. 두 스트링이 같으면 0, 현 스트링이 지정된 스트링보다 사전적으로 먼저 나오면 음수, 아니면 양수를 리턴
String concat(String str)	str 스트링을 현재 스트링 뒤에 덧붙인 스트링 리턴
boolean contains(CharSequence s)	s에 지정된 일련의 문자들을 포함하고 있으면 true 리턴
int length()	스트링의 길이 리턴
String replace(Charsequece target, Charsequence replacement)	target이 지정하는 일련의 문자들을 replacement가 지정하는 문자들로 변경한 스트링 리턴
String[] split(String regex)	정규식 regex에 일치하는 부분을 중심으로 스트링을 분리하고 분리된 스트링을 배열에 저장하여 리턴
String subString(int beginIndex)	beginIndex 인덱스부터 시작하는 서브 스트링 리턴
String toLowerCase()	스트링을 소문자로 변경한 스트링 리턴
String toUpperCase()	스트링을 대문자로 변경한 스트링 리턴
String trim()	스트링 앞뒤의 공백 문자들을 제거한 스트링 리턴

문자열 비교

- int compareTo(String anotherString)
  - 문자열이 같으면 0 리턴
  - 이 문자열이 anotherString 보다 사전에 먼저 나오면 음수 리턴
  - 이 문자열이 anotherString 보다 사전에 나중에 나오면 양수 리턴

```
String a = "java";
String b = "jasa";
int res = a.compareTo(b);
if(res == 0)
  System.out.println("the same");
else if(res < 0)
  System.out.println(a +"<"+b);
else
  System.out.println(a +">"+b);
```

"java" 가 "jasa" 보다 사전에 나중에 나오기 때문에 양수 리턴

```
java>jasa
```

- 비교 연산자 ==는 문자열 비교에는 사용할 수 없음

문자열 연결

- + 연산자로 문자열 연결
  - + 연산의 피연산자에 문자열이 있는 경우
  - + 연산에 객체가 포함되어 있는 경우
    - 객체.toString()을 호출하여 객체를 문자열로 변환한 후 문자열 연결
  - 기본 타입 값은 문자열로 변환된 후에 연결

```
System.out.print("abcd" + 1 + true + 3.13e-2 + 'E'+ "fgh" );
// abcd1true0.0313Efgh  출력
```

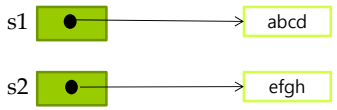
- String concat(String str)를 이용한 문자열 연결

```
"abcd".concat("efgh");
// "abcdefg" 리턴
```

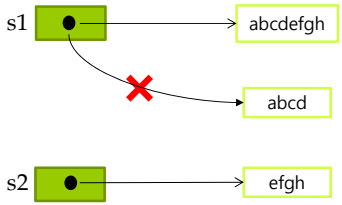
  - 기존 String 객체에 연결되지 않고 새로운 스트링 객체 생성 리턴

concat()은 새로운 문자열을 생성

```
String s1 = "abcd";
String s2 = "efgh";
```




```
s1 = s1.concat(s2);
```



문자열 내의 공백 제거, 문자열의 각 문자 접근

- 공백 제거
  - String trim()
    - 문자열 앞 뒤 공백 문자(tab, enter, space) 제거한 문자열 리턴

```
String a = " abcd def ";
String b = "₩txyz₩t";
String c = a.trim(); // c = "abcd def"
String d = b.trim(); // d = "xyz"
```

- 문자열의 문자
  - char charAt(int index)
    - 문자열 내의 문자 접근

```
String a = "class";
char c = a.charAt(2); // c = 'a'
```

```
// "class"에 포함된 's'의 개수를 세는 코드
int count = 0;
String a = "class";
// a.length()는 5
for(int i=0; i<a.length(); i++) {
  if(a.charAt(i) == 's')
    count++;
}
System.out.println(count); // 2 출력
```

예제 : String 클래스 메소드 활용

String 클래스의 다양한 메소드를 활용하는 예를 보여라.

```
public class StringEx {
  public static void main(String[] args) {
    String a = new String(" abcd");
    String b = new String(",efg");
    // 문자열 연결
    a = a.concat(b);
    System.out.println(a);
    // 공백 제거
    a = a.trim();
    System.out.println(a);
    // 문자열 대치
    a = a.replace("ab","12");
    System.out.println(a);
```

예제 : String 클래스 메소드 활용

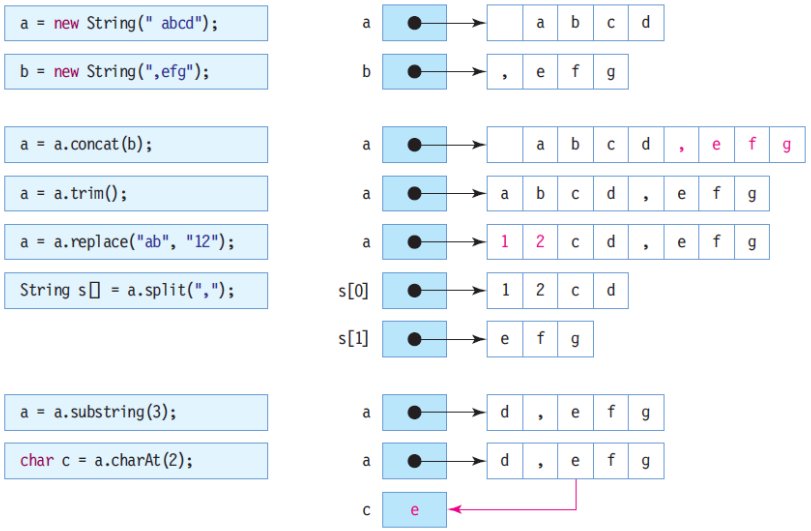
```
    // 문자열 분리
    String s[] = a.split(",");
    for (int i=0; i<s.length; i++)
      System.out.println("분리된 " + i + "번 문자열: " + s[i]);

    // 서브 스트링
    a = a.substring(3);
    System.out.println(a);

    // 문자열의 문자
    char c = a.charAt(2);
    System.out.println(c);
  }
}
```

```
abcd,efg
abcd,efg
12cd,efg
분리된 0번 문자열: 12cd
분리된 1번 문자열: efg
d,efg
e
```

예제 실행 과정



StringBuffer 클래스

- java.lang.StringBuffer
  - 스트링과 달리 객체 생성 후 스트링 값 변경 가능
  - append와 insert 메소드를 통해 스트링 조작
  - StringBuffer 객체의 크기는 스트링 길이에 따라 가변적
- 생성자

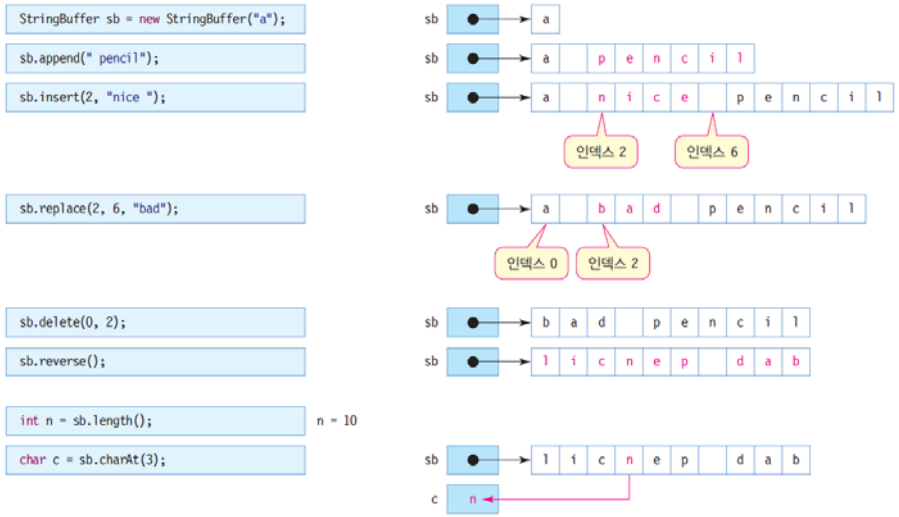
```
StringBuffer sb = new StringBuffer("java");
```

생성자	설명
StringBuffer()	초기 버퍼의 크기가 16인 스트링 버퍼 객체 생성
StringBuffer(charSequence seq)	seq가 지정하는 일련의 문자들을 포함하는 스트링 버퍼 생성
StringBuffer(int capacity)	지정된 초기 크기를 갖는 스트링 버퍼 생성
StringBuffer(String str)	지정된 스트링으로 초기화된 스트링 버퍼 생성

SringBuffer 주요 메소드

메소드	설명
StringBuffer append(String str)	str 스트링을 스트링 버퍼에 덧붙인다.
StringBuffer append (StringBuffer sb)	sb 스트링 버퍼를 현재의 스트링 버퍼에 덧붙인다. 이 결과 현재 스트링 버퍼의 내용이 변한다.
int capacity()	현재 스트링 버퍼의 크기 리턴
StringBuffer delete (int start, int end)	start 위치에서 end가 지정하는 문자의 앞까지 스트링 제거
StringBuffer insert (int offset, String str)	str 스트링을 스트링 버퍼의 offset 위치에 삽입
StringBuffer replace (int start, int end, String str)	스트링 버퍼 내의 start가 지정하는 문자부터 end가 지정하는 문자 앞의 서브 스트링을 str로 대치
StringBuffer reverse()	스트링 버퍼 내의 문자들을 반대 순서로 변경
void setLength(int newLength)	스트링 버퍼 내 문자열 길이를 newLength로 재설정. 현재 길이보다 큰 경우 널 문자로 채우며 작은 경우는 기존 문자열이 잘린다.

StringBuffer의 메소드 활용 예



예제 : StringBuffer 클래스 메소드 활용

StringBuffer 클래스의 메소드를 이용하여 문자열을 조작하는 예를 보이자. 다음 코드의 실행 결과는?

```
public class StringBufferEx {
  public static void main(String[] args) {
    StringBuffer sb = new StringBuffer("This");
    System.out.println(sb.hashCode());
    sb.append(" is pencil"); // 문자열 덧붙이기
    System.out.println(sb);
    sb.insert(7, " my"); // 문자열 삽입
    System.out.println(sb);
    sb.replace(8, 10, "your"); // 문자열 대치
    System.out.println(sb);
    sb.setLength(5); // 스트링 버퍼 내 문자열 길이 설정
    System.out.println(sb);
    System.out.println(sb.hashCode());
  }
}
```

```
14576877
This is pencil
This is my pencil
This is your pencil
This
14576877
```

StringTokenizer 클래스

- java.util.StringTokenizer
  - 구분 문자를 기준으로 문자열 분리
    - 문자열을 구분할 때 사용되는 문자를 구분 문자(delimiter)라고 함

```
String query = "name=kitae&addr=seoul&age=21";
StringTokenizer st = new StringTokenizer(query, "&");
```

      - 위의 예에서 '&'가 구분 문자
  - 토큰(token)
    - 구분 문자로 분리된 문자열
  - String 클래스의 split() 메소드를 이용하여 동일한 구현 가능

StringTokenizer 주요 메소드

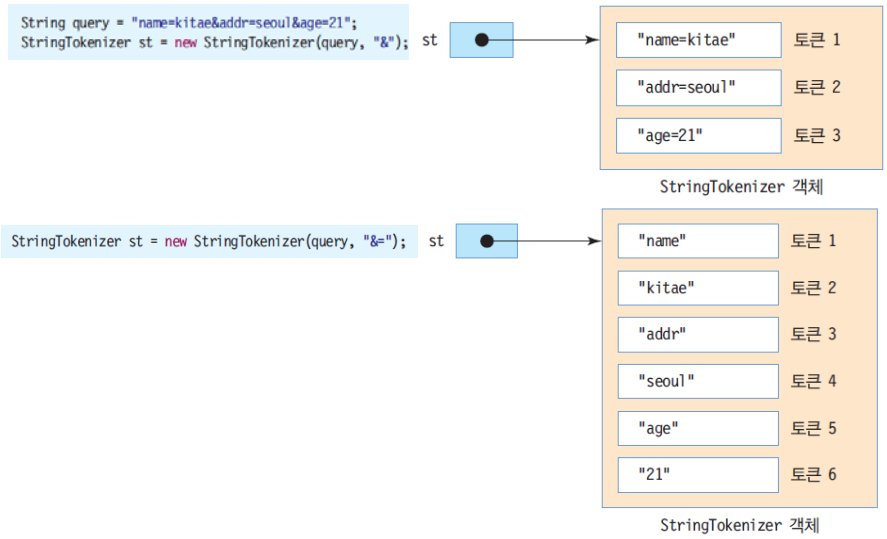
- StringTokenizer 생성자

생성자	설명
StringTokenizer(String str)	str 스트링으로 파싱한 스트링 토크나이저 생성
StringTokenizer(String str, String delim)	str 스트링과 delim 구분 문자로 파싱한 스트링 토크나이저 생성
StringTokenizer(String str, String delim, boolean returnDelims)	str 스트링과 delim 구분 문자로 파싱한 스트링 토크나이저 생성. returnDelims가 true이면 delim이 포함된 문자도 토큰에 포함된다.

- 주요 메소드

메소드	설명
int countTokens()	스트링 토크나이저에 포함된 토큰의 개수 리턴
boolean hasMoreTokens()	스트링 토크나이저에 다음 토큰이 있으면 true 리턴
String nextToken()	다음 토큰 리턴

StringTokenizer 객체 생성과 문자열 분리



예제 : StringTokenizer 클래스 메소드 활용

"홍길동/장화/홍련/콩쥐/팥쥐"문자열을'/'를 구분 문자로 하여 토큰을 분리하여 각 토큰을 출력하라.

```
import java.util.StringTokenizer;

public class StringTokenizerEx {
  public static void main(String[] args) {
    StringTokenizer st =
                new StringTokenizer("홍길동/장화/홍련/콩쥐/팥쥐", "/");
    while (st.hasMoreTokens())
      System.out.println(st.nextToken());
  }
}
```

```
홍길동
장화
홍련
콩쥐
팥쥐
```

Math 클래스

- 기본적인 산술 연산을 수행하는 메소드 제공
  - java.lang.Math
  - 모든 메소드는 static으로 선언
  - 클래스 이름으로 바로 호출 가능

메소드	설명
static double abs(double a)	실수 a의 절댓값 리턴
static double cos(double a)	실수 a의 cosine 값 리턴
static double sin(double a)	실수 a의 sine 값 리턴
static double tan(double a)	실수 a의 tangent 값 리턴
static double exp(double a)	e^a 값 리턴
static double ceil(double a)	실수 a보다 크거나 같은 수 중에서 가장 작은 정수를 실수 타입으로 리턴
static double max(double a, double b)	두 수 a, b 중에서 큰 수 리턴
static double min(double a, double b)	두 수 a, b 중에서 작은 수 리턴
static double random()	0.0보다 크거나 같고 1.0보다 작은 임의의 실수 리턴
static double rint(double a)	지정된 실수 a에 가장 근접한 정수를 실수 타입으로 리턴
static long round(double a)	실수 a를 소수 첫째 자리에서 반올림한 정수를 long 타입으로 반환
static double sqrt(double a)	실수 a의 제곱근 리턴

Math 클래스를 활용한 난수 발생

- 난수 발생
  - static double random()
    - 0.0 이상 1.0 미만의 임의의 double 값을 반환
    - 0에서 100사이의 정수 난수 10개 시키는 샘플 코드

```
for(int x=0; x<10; x++) {
  double d = Math.random()*100; // [0.0 ~ 99.9999] 실수 발생
  // d를 반올림하고 정수로 변환. [0~100] 사이의 정수
  int n = (int)(Math.round(d));
  System.out.println(n);
}
```

    - 위의 코드에서 round() 메소드는 Math. round(55.3)은 55.0을 리턴하며, Math. round(55.9)는 56.0을 리턴
  - java.util.Random 클래스를 이용하면 좀 더 다양한 형태로 난수 발생 가능

예제 : Math 클래스 메소드 활용

Math 클래스의 다양한 메소드 활용 예를 보여라.

```
public class MathEx {
  public static void main(String[] args) {
    double a = -2.78987434;
    // 절대값 구하기
    System.out.println(Math.abs(a));
    System.out.println(Math.ceil(a)); // ceil
    System.out.println(Math.floor(a)); // floor
    System.out.println(Math.sqrt(9.0)); // 제곱근
    System.out.println(Math.exp(1.5)); // exp
    System.out.println(Math.rint(3.141592)); // rint
    // [1,45] 사이의 난수 발생
    System.out.print("이번주 행운의 번호는");
    for (int i=0; i<5; i++)
      System.out.print(Math.round(1 + Math.random() * 44) + " ");
    System.out.println("입니다.");
  }
}
```

```
2.78987434
-2.0
-3.0
3.0
4.4816890703380645
3.0
이번주 행운의 번호는 35 42 18 31 33
```

Calendar 클래스

- Calendar 클래스의 특징
  - java.util 패키지
  - 시간과 날짜 정보 관리
    - 년, 월, 일, 요일, 시간, 분, 초, 밀리초, 오전 오후 등
    - Calendar 클래스의 각 시간 요소를 설정하기나 알아내기 위한 필드들

필드	의미	필드	의미
YEAR	년도	DAY	한 달의 날짜
MONTH	달	DAY_OF_WEEK	한 주의 요일
HOUR	0~11시로 표현한 시간	AM_PM	오전인지 오후인지 구분
HOUR_OF_DAY	24시간을 기준으로 한 시간	MINUTE	분
SECOND	초	MILLISECOND	밀리초

Calendar 객체 생성 및 날짜와 시간

- Calendar 객체 생성
  - Calendar now = Calendar.getInstance(); 이용
    - now객체는 현재 날짜와 시간 정보를 가지고 생성
    - Calendar는 추상 클래스이므로 new Calendar() 하지 않음
- 현재 날짜와 시간

```
int year = now.get(Calendar.YEAR);       // 현재 년도
int month = now.get(Calendar.MONTH) + 1; // 현재 달
```

- 날짜와 시간 설정하기
  - 내가 관리할 날짜와 시간을 Calendar객체를 이용하여 저장
    - Calendar 객체에 날짜와 시간을 설정한다고 해서 컴퓨터의 날짜와 시간을 바꾸는 것은 아님 -> 컴퓨터의 시간과 날짜를 바꾸는 다른 방법 이용

```
// 이성 친구와 처음으로 데이트한 날짜와 시간 저장
Calendar firstDate = Calendar.getInstance();
firstDate.clear(); // 현재 날짜와 시간 정보를 모두 지운다.
firstDate.set(2012, 11, 25); // 2012년 12월 25일. 12월은 11로 설정
firstDate.set(Calendar.HOUR_OF_DAY, 20); // 저녁 8시로 설정
firstDate.set(Calendar.MINUTE, 30); // 30분으로 설정
```

예제 : Calendar를 이용하여 현재 날짜와 시간 출력 및 설정하기

```
import java.util.Calendar;
public class CalendarEx {
  public static void printCalendar(String msg, Calendar cal) {
      int year = cal.get(Calendar.YEAR);
      // get()은 0~30까지의 정수 리턴.
      int month = cal.get(Calendar.MONTH) + 1;
      int day = cal.get(Calendar.DAY_OF_MONTH);
      int dayOfWeek = cal.get(Calendar.DAY_OF_WEEK);
      int hour = cal.get(Calendar.HOUR);
      int hourOfDay = cal.get(Calendar.HOUR_OF_DAY);
      int ampm = cal.get(Calendar.AM_PM);
      int minute = cal.get(Calendar.MINUTE);
      int second = cal.get(Calendar.SECOND);
      int millisecond = cal.get(Calendar.MILLISECOND);
      System.out.print(msg + year + "/" + month + "/" + day + "/");
```

예제 : Calendar를 이용하여 현재 날짜와 시간 출력 및 설정하기

```
      switch(dayOfWeek) {
      case Calendar.SUNDAY : System.out.print("일요일"); break;
      case Calendar.MONDAY : System.out.print("월요일"); break;
      case Calendar.TUESDAY : System.out.print("화요일"); break;
      case Calendar.WEDNESDAY : System.out.print("수요일"); break;
      case Calendar.THURSDAY : System.out.print("목요일"); break;
      case Calendar.FRIDAY: System.out.print("금요일"); break;
      case Calendar.SATURDAY : System.out.print("토요일"); break;
    }
    System.out.print("(" + hourOfDay + "시)");
    if(ampm == Calendar.AM) System.out.print("오전");
    else System.out.print("오후");
    System.out.println(hour + "시 " + minute + "분 " + second + "초 "
     + millisecond +"밀리초");
  }
```

예제 : Calendar를 이용하여 현재 날짜와 시간 출력 및 설정하기

```
public static void main(String[] args) {
      Calendar now = Calendar.getInstance();
      printCalendar("현재 ", now);

      Calendar firstDate = Calendar.getInstance();
      firstDate.clear();
    // 2012년 12월 25일. 12월을 표현하기 위해 month에 11로 설정
      firstDate.set(2012, 11, 25);
      firstDate.set(Calendar.HOUR_OF_DAY, 20); // 저녁 8시
      firstDate.set(Calendar.MINUTE, 30); // 30분
      printCalendar("처음 데이트한 날은 ", firstDate);
  }
}
```

```
현재 2012/12/27/목요일(20시)오후8시 22분 28초 889밀리초
처음 데이트한 날은 2012/12/25/화요일(20시)오후8시 30분 0초 0밀리초
```