Scanner

경복대학교
소프트웨어융합과
배희호 교수
010-2369-4112
031-570-9600
hhbae@kbu.ac.kr

Scanner 클래스

- Scanner는 Data를 Keyboard로부터 입력 받거나, File로부터 입력 받거나, 문자열(String)로 입력을 받는데 도움을 주는 아주 유용한 클래스
- Scanner 클래스는 Keyboard, File, String 등으로부터 입력 받기 위해 다양한 생성자를 제공하며 다양한 Data Type을 입력 받을 수 있음
- 사용할 Scanner의 Type과 저장할 변수의 Type, 내가 직접 입력할 Data의 Type은 항상 통일 시켜야 함
- Scanner 생성자는 다양한데 항상 두 번째 argument는 Encoding 방식의 Charset의 이름 지정
  - 예) “UTF-8”인지 “ANSI”인지 지정할 수 있음
  - 지정하지 않으면 OS에서 지정된 Default Charset으로 됨

Scanner 클래스

- 공란과 줄 바꿈을 모두 입력 값의 경계로 인식하기 때문에 좀 더 쉽게 Data를 입력 받을 수 있음
- Data Type이 입력 받는 시점에서 결정되므로 별도의 Casting이 필요하지 않음

Scanner 클래스

- Regular Expression(정규식)
  - 특정한 규칙을 가진 문자열의 집합을 표현하는 형식 언어로 다양한 문자열의 검색과 치환을 위하여 지원
- 정규식을 사용하여 기본 Data 타입/문자열을 구문 분석할 수 있는 간단한 Text Scanner
- Scanner 클래스는 다양한 타입의 입력 값들을 읽어 들이기 위한 편리한 방법을 제공하는 클래스

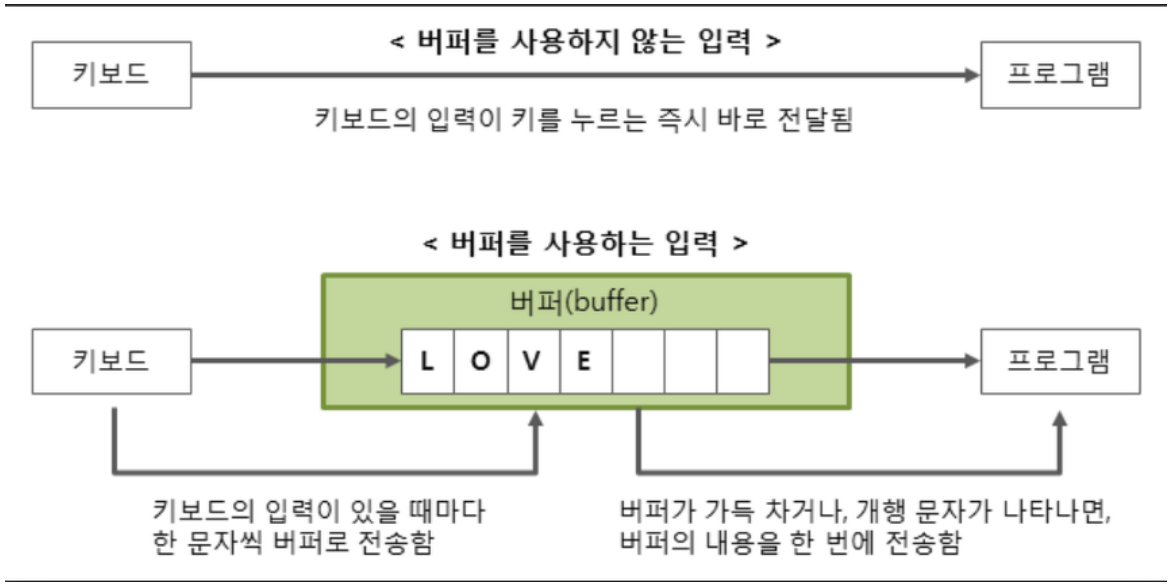
- Scanner 클래스 입력 방법
  - Keyboard (File)
  - File Stream
  - String

Scanner 클래스

- Scanner는 File을 한번에 전부 읽어 들임

```
Scanner input = new Scanner(file);
```

  - File을 한 번에 읽어서 Scanner 객체인 input이 가지고 있고, 해당 Data Type에 맞추어 읽어 들이면 됨



Scanner 클래스

- 메소드

메소드	설명
nextBoolean()	boolean의 자료형을 입력 받음 자료형은 두 개인 true, false로 대소문자를 구분하지 않음
nextByte()	byte의 자료형을 입력으로 받음 입력 범위(127 ~ -128) 밖이면 InputMismatchException이 발생
nextShort()	short의 자료형을 입력 받음
nextInt()	int형의 자료형을 입력 받음
nextLong()	long형의 자료형을 입력 받음
nextDouble()	double형의 자료형을 입력 받음
nextFloat()	float형의 자료형을 입력 받음
next()	String형의 문자열을 입력 받음 (이때 공백 문자까지 입력 받음)
nextLine()	문자열을 입력 받는데 다른 next~() 메소드와 다른 점은 줄 단위로 입력 받음

Scanner 클래스

- 주의 사항
  - Scanner 클래스의 객체를 만들려면 표준 입력 Stream을 나타내는 System.in이라는 이미 정의 되어있는 객체를 전달해야 함 (File로부터의 입력은 File 클래스의 객체를 전달하면 됨)
  - 특정 Data Type의 값을 읽기 위해 사용하는 메소드는 next~() 메소드
    - 예) nextInt(), nextDouble() 등과 같은 메소드를 사용
  - 문자열, 한 줄(Enter Key 기준으로)을 모두 읽기 위해서는 nextLine() 메소드 사용
  - 단일 문자(char)를 읽기 위해서 next()와 charAt() 메소드를 함께 사용

Scanner 클래스

- nextLine() 메소드
  - Scanner를 이용한 입력에서 특히 주의 해야하는 것은 next(), nextInt() 등의 메소드는 띄어 쓰기를 기준으로 입력 값들을 읽고, nextLine() 메소드의 경우 한 Line을 기준으로 입력 값들을 읽음
  - 이 경우 문제가 발생할 수 있는데, next() 계열 메소드를 입력한 후 <Enter>를 입력하는 경우, next() 계열 메소드는 <Enter>의 줄 바꿈 문자를 처리하지 못하고 Buffer에 남겨 둠
  - 이 남겨둔 줄 바꿈 문자를 다음 Line에서 nextLine() 메소드가 인식하는 순간, 의도하지 않은 출력 결과를 발생함

Scanner와 BufferedReader

- Scanner와 BufferedReader는 모두 문자열을 입력 받는데 사용되는 클래스
- BufferedReader는 일정한 크기(8192 chars)의 Data를 한 번에 읽어와 Buffer에 보관 후 사용자 요청 시 Buffer에서 Data를 읽어오는 방식. 그래서 시간 부하를 줄일 수 있고, 입력 받는 모든 형식은 String임
- Scanner는 Data를 입력 받는 시점 정규식을 이용해서 Data Type이 결정되므로 별도의 Casting이 필요치 않고, 사용자 요청 시 바로바로 Data를 주기 때문에 Data 용량이 클 경우 BufferedReader보다 느림
- 속도는 BufferedReader가 Scanner보다 빠름

Scanner와 BufferedReader

- 두 Class 모두 문자열을 입력 받는데 사용된다는 공통점이 있지만, 사용 방법, 속도 등 차이점이 존재

	BufferedReader	Scanner
나온 시점	jdk 1.1	jdk 1.5
Buffer Size	8192	1024
Syncronized	O	X
문자열 파싱	단순히 읽어 들임	문자열 Parsing 가능
정규 표현식 사용	X	O
Exception	IOException 던짐	IOException 숨김

- JAVA에선 BufferedReader를 사용하는게 성능 향상에 더 좋음

Delimiter

- JAVA에서 구분 기호는 문자열을 Token으로 분할(분리)하는 문자
- 공백 구분 기호는 JAVA의 기본 구분 기호

- 방법
  - Scanner.next() 메소드 : 기본 공백 분리자만 적용
  - String.split() 메소드
  - StringTokenizer/StreamTokenizer Class 사용
  - Scanner.useDelimiter() 메소드

Delimiter

- split() 메소드
  - split()는 String 클래스의 메소드로 String에 존재하는 특정한 Pattern을 구분자로 문자열들을 추출할 때 사용

String split(String regex)	java의 정규 표현식 또는 특정 문자열을 구분자로 문자열들을 추출
String split(String regex, int limit)	정규 표현식 또는 특정 문자열을 구분자로 limit 만큼 문자열들을 추출

StringTokenizer 클래스

- 문자 Stream과 구분자(Delimiter)를 입력 받아 Token을 생성하는 기능을 가진 클래스
- 문자를 입력 받아 처리하는 Program에 유용하게 사용될 수 있음
- StringTokenizer는 문자열을 추출하기 위한 클래스
- StringTokenizer 생성자에 전달할 매개 변수에 따라 다양한 방법으로 문자열을 추출할 수 있음
- split()와 차이점은 정규 표현식이 아닌 구분자(Delimiter)를 이용해서 문자열을 추출한다는 것이고 추출한 문자열을 String 배열로 반환해주는 split()와는 달리 Iterator를 이용해서 추출된 문자열을 일일이 읽어 들여야 하는 번거로운 과정과 Code를 사용해야 함

StringTokenizer 클래스

- 생성자

StringTokenizer(String str)	구분자(delimiter)를 인자로 받지않는 생성자. 디폴트 구분자로 공백문자(₩t, ₩n, ₩r, ₩f)를 가짐
StringTokenizer(String str, String delim)	구분자(delimiter)를 인자로 받는 생성자 구분자는 2자리 이상도 설정할 수 있음 구분자를 $%라고 설정하면 $, %를 기준으로 분리함 %%라고 설정하면 %를 기준으로 분리함 구분자는 길이가1로 고정이고 길이가 2이상인 구분자를 받으면 구분자가 여러 개가 됨
StringTokenizer(String str, String delim, boolean returnDelims)	구분자(delimiter)를 인자로 받는 생성자 returnDelims가 true이면 구분자도 토큰으로 간주 각 구분자는 길이가 1인 String이 됨 예) delimiter가 %%라도 % 2개로 저장 returnDelims가 false면 구분자를 token으로 사용하지 않는다. 위의 두 생성자는 디폴트로 false를 가짐

StringTokenizer 클래스

- 제공되는 클래스 상수

상수	설명
TT_EOF	파일의 끝
TT_EOF	라인의 끝
TT_NUMBER	읽은 토큰이 숫자임
TT_WORD	읽은 토큰이 단어임

StreamTokenizer 클래스

- 메소드

메소드	설명
int lineno()	현재의 라인 번호를 반환
int nextToken()	다음 토큰이 숫자이면, TT_NUMBER를 반환. 단어이면 TT_WORD를 반환, 그 외에는 읽은 문자를 반환
void lowerCaseMode(boolean flag)	flag가 true이면, 모든 토큰이 소문자로 취급
String toString()	현재의 토큰과 동일한 문자열을 반환
void whitespaceChars(int c1, int c2)	c1-c2 사이의 모든 문자를 공란으로 취급
void wordChars(int c1, int c2)	c1-c2사이의 모든 문자를 단어로 취급
void ordinaryChar(int i)	i를 정규 문자로 지정

StreamTokenizer 클래스

- 메소드

메소드	설명
void commentChar(int ch)	ch가 주석문(한 라인)의 시작 문자임
void eolIsSignificant(boolean flag)	flag가 true이면 end-of-line을 토큰으로 취급하고, false이면 공백으로 취급
void parseNumbers()	현재 객체의 숫자를 파싱
void quoteChar(int ch)	ch를 quote문자로 설정
void resetSyntax()	모든 문자를 정규 문자로 취급
void slashSlashComments(boolean flag)	flag가 true이면 // 주석문이 무시
void slashStarComments(boolean flag)	*flag*가 true이면 / 주석문이 무시

StreamTokenizer 클래스

- StringTokenizer와 split() 차이
  - StringTokenizer는 java.util에 포함되어 있는 클래스, split()는 String 클래스에 속해있는 메소드
  - StringTokenizer는 문자 또는 문자열로 문자열을 구분한다면, split()는 정규 표현식으로 구분
  - StringTokenizer는 빈 문자열을 Token으로 인식하지 않지만 split()는 빈 문자열을 Token으로 인식하는 차이가 있음
  - Stringtokenizer는 결과값이 문자열이라면 split는 결과값이 문자열 배열. 따라서 StringTokenizer를 이용할 경우 전체 Token을 보고 싶다면 반복문을 이용해서 하나 하나 출력할 수 밖에 없음
  - 배열에 담아 반환하는 split()는 Data를 바로바로 잘라서 반환해주는 StringTokenizer보다 성능이 약간 떨어짐

StreamTokenizer 클래스

- StringTokenizer 객체는 객체 속성 변수로 nval, sval, ttype를 사용 가능
  - nval, sval, ttype의 접근 제한자는 public이므로 Program에서 자유로이 참조 가능
  - 읽은 Token의 값이 숫자이면 nval에 저장되고, 문자열이면 sval에 저장됨
  - 읽은 Token의 유형이 숫자나 문자열이 아닌 경우에는 ttype에 저장됨

Delimiter

- useDelimiter() 메소드
  - 구분자는 기본적으로 공백(space)인데 필요에 따라서는 우리가 정해야 될 필요가 있음
  - 구분자 지정 메소드가 useDelimiter() 메소드
  - useDelimiter()는 2가지로 Overloading이 되어있는 메소드인데, 단순 문자열로만 구분할 수 있는 메소드와 정규표현식을 사용하여 쓸 수 있는 구분자가 사용
  - Scanner를 활용하여 정규식 표현의 Data를 구분하는 구분 문자(delimiter)에도 정규식 표현을 사용할 수 있어서 복잡한 형태의 구분 문자도 처리가 가능

Delimiter

- 정규 표현식 작성 방법
  - 정규 표현식은 문자 또는 숫자 기호와 반복 기호가 결합된 문자열

기호	설명		
[ ]	한 개의 문자	[abc]	a, b, c 중 하나의 문자
		[^abc]	a, b, c 이외의 하나의 문자
		[a-zA-Z]	a~z, A~Z 중 하나의 문자
\d	한 개의 숫자 [0-9] 와 동일		
\s	공백		
\w	한 개의 알파벳 또는 한 개의 숫자, [a-zA-Z_0-9] 와 동일		
?	없음 또는 한 개		
*	없음 또는 한 개 이상		
+	한 개 이상		
{n}	정확히 n개		
{n,}	최소한 n개		
{n,m}	n개에서부터 m개까지		
( )	그룹핑		

Delimiter

- useDelimiter() 메소드
  - 예)
    - ₩s : 공백 1개
    - * : 앞의 값이 없거나 한번 이상 나옴
      - ₩s* : 공백(₩s)이 없거나 한번 이상 나옴
    - | : or
    - ₩r₩n : 개행 문자
      - (₩s*,₩s*)|(₩r₩n) : 처음에 공백(₩s)이 없거나 한번 이상 나오고, 그 다음 쉼표(,)가 나오고, 다시 공백(₩s)이 없거나 한번 이상 나옴
      - | : 혹은
      - ₩r₩n : 개행 문자가 옴

Delimiter 예제 1

```
public static void main(String[] args) {
    String str = "Javasite is the best website to learn new technologies";

    Scanner scanner = new Scanner(str);
    while (scanner.hasNext()) {
        String tokens = scanner.next();
        System.out.println(tokens);
    }
    scanner.close();
}
```

✓기본 분리자인 공백 분리자 적용
✓Scanner.next() 메소드 사용

Delimiter 예제 2

```
public static void main(String[] args) {
    String str = "Javasite is the best website to learn new technologies";
    String[] stringarray = str.split(" ");

    for(int  i = 0; i < stringarray.length; i++) {
        System.out.println(stringarray[i]);
    }
}
```

✓ String.split() 메소드 사용하여 문자열을 분리

Delimiter 예제 3

```
public static void main(String[] args) {
    String str = "Javasite is the best website to learn new technologies";

    StringTokenizer tokens = new StringTokenizer(str, " ");
    while (tokens.hasMoreTokens()) {
        System.out.println(tokens.nextToken());
    }
}
```

✓ StringTokenizer Class 사용

Delimiter 예제 4

```
public static void main(String[] args) {
    String str = "Javasite is the best website to learn new technologies";

    Scanner scan = new Scanner(str).useDelimiter(" ");
    while(scan.hasNext()) {
        System.out.println(scan.next());
    }
    scan.close();
}
```

- ✓ useDelimiter() 메소드 사용
  - ✓ useDelimiter(" ")
  - ✓ useDelimiter("₩₩s")

Delimiter 예제 5

- Keyboard로 입력 받은 주민번호에서 문자열을 '–‘ 또는 공백(‘ ‘)으로 분리하여 주민번호 앞자리만 출력하는 프로그램을 작성하라.

- Hint
  - 구분 문자를 설정할 때에는 Scanner 클래스의 useDelimiter() 메소드를 사용

Delimiter 예제 5

```
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in).useDelimiter("(₩₩s)|-");
    String start, last;

    System.out.print("주민등록번호를 입력 ");
    start = keyboard.next();
    last = keyboard.next();
    System.out.println("주민등록번호 앞자리 : " + start);
    System.out.println("주민등록번호 뒷자리 : " + last);
}
```

Scanner 예제 1

- 문자열에서 입력

```
public static void main(String[] args) {
    String source = "2 boys and 2 girls";
    Scanner input = new Scanner(source);
    int num1 = input.nextInt();
    String word1 = input.next();
    String word2 = input.next();
    int num2 = input.nextInt();
    String word3 = input.next();

    System.out.println(num1 + " " + word1 + " " + word2 + " " +
                                                    num2 + " " + word3);
}
```

- Delimeiter를 추가해보자

```
.useDelimiter("₩₩s");
```

Scanner 예제 2

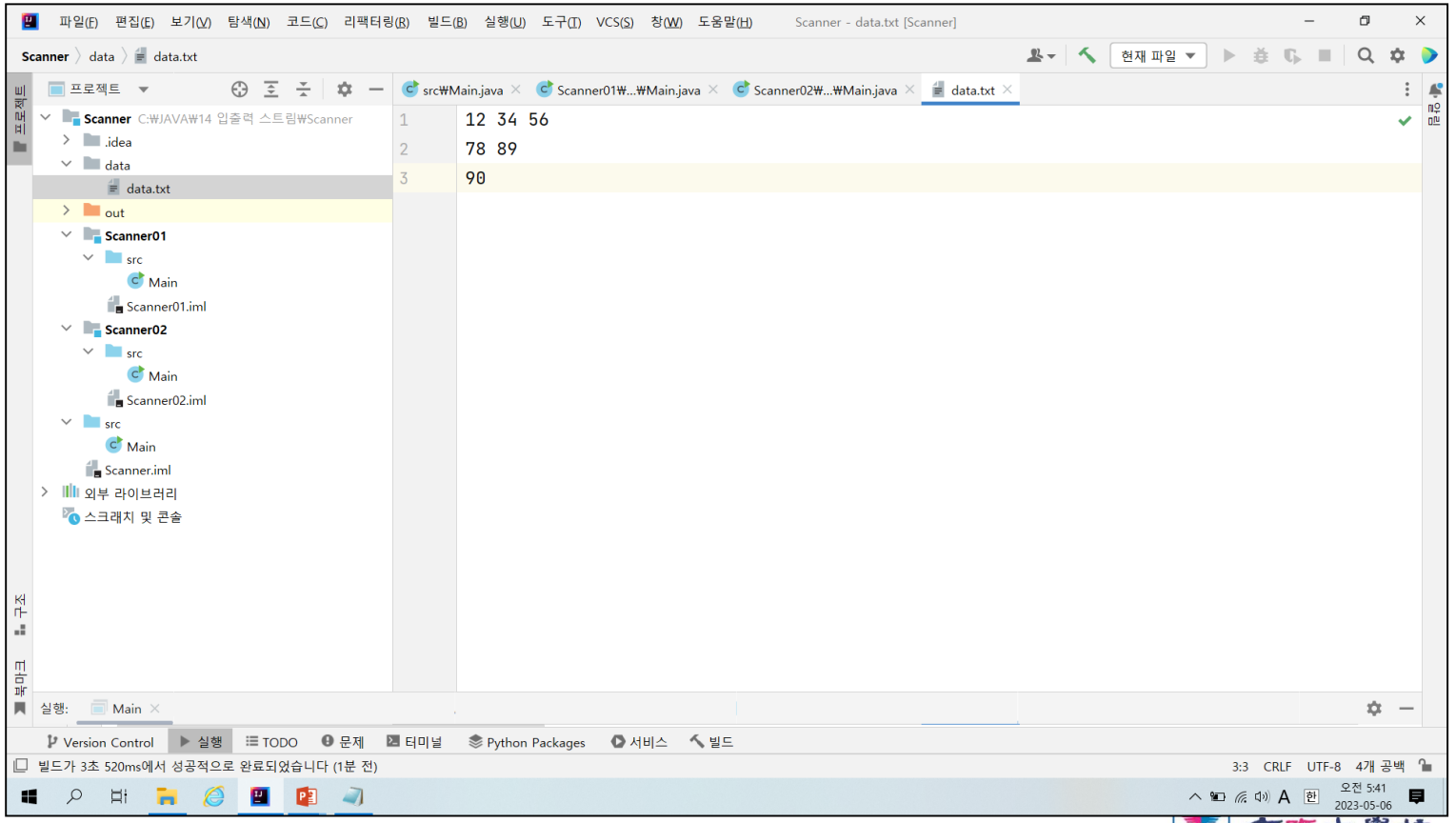
```
public static void main(String[] args) {
    String source = "생각하는 JAVA, 재미있는 JAVA, 즐거운 JAVA, 신나는 JAVA";

    Scanner input = new Scanner(source);
    input.useDelimiter(" JAVA, ");
    String result = "";
    while (input.hasNext())
        result += input.next() + ", ";
    result = result.substring(0, result.length()-2);
    System.out.print(result);

    input.close();
}
```

Scanner 예제 3

```
public static void main(String[] args) {
    String source = "1  fish  2  fish  red  fish   blue   fish";

    Scanner input = new Scanner(source);
    input.useDelimiter("₩₩s*fish₩₩s*");
    // "₩₩s*"는 정규표현식으로서 한 개이상의 스페이스를 의미
    System.out.println(input.nextInt());
    System.out.println(input.nextInt());
    System.out.println(input.next());
    System.out.println(input.next());

    input.close();
}
```

Scanner 예제 4

- File에서 입력
  - Scanner는 file에서도 입력을 받을 수 있음
  - data.txt 파일 만들어야 함



Scanner 예제 4

- File에서 입력

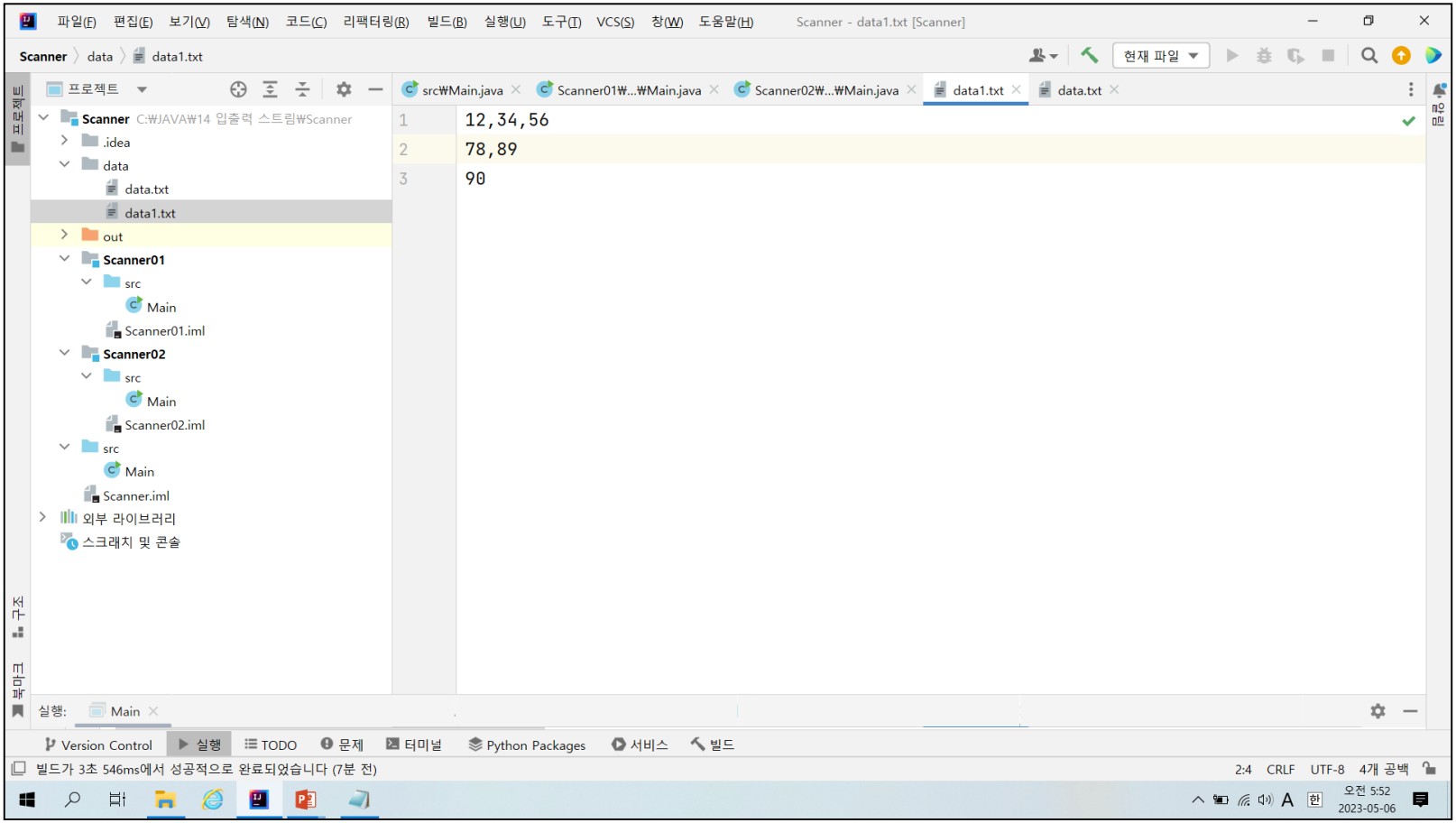
```
public static void main(String[] args) throws FileNotFoundException {
    String path = ".₩₩data₩₩data.txt";

    File file = new File(path);
    if (file.exists()) {
        Scanner input = new Scanner(file).useDelimiter("(₩₩r₩n)|(₩₩s)");
        System.out.println("파일로부터 정수 입력 더하기");
        int sum = 0;
        while (input.hasNextInt()) {
            int data = input.nextInt();
            sum += data;
            System.out.print(data + " + ");
        }
        System.out.print("₩b₩b= " + String.format("%,d", sum));
    }else {
        System.out.println("파일이 존재하지 않아요");
    }
}
```

Scanner 예제 4

- File에서 입력
  - Delimiter를 변경해보자
    - 삭제해보자
    - useDelimiter("(₩₩r₩n)|(₩₩s)")
    - useDelimiter("(₩₩s)|(₩₩r₩n)")

Scanner 예제 5

- File에서 입력
- data1.txt 파일 만들어야 함



Scanner 예제 5

```
public static void main(String[] args) throws FileNotFoundException {
    String path = ".₩₩data₩₩data1.txt";

    File file = new File(path);
    if (file.exists()) {
        Scanner input = new Scanner(file).useDelimiter(",|(₩₩r₩n)");
        System.out.println("파일로부터 정수 입력 더하기");
        int sum = 0;
        while (input.hasNext()) {
            int data = input.nextInt();
            sum += data;
            System.out.print(data + " + ");
        }
        System.out.print("₩b₩b= " + String.format("%,d", sum));
    }else {
        System.out.println("파일이 존재하지 않아요");
    }
}
```

Scanner 예제 5

- File에서 입력
  - Delimiter를 변경해보자
    - 삭제해보자
    - useDelimiter(",|(₩₩r₩n)")
    - useDelimiter("(₩₩r₩n)|,")

Scanner 예제 6

```
public static void main(String[] args) throws FileNotFoundException {
    String path = ".₩₩data₩₩data1.txt";
    File file = new File(path);
    if (file.exists()) {
        Scanner input = new Scanner(file).useDelimiter("₩₩r₩n");
        System.out.println("파일로부터 정수 입력 더하기");
        int sum = 0;
        while (input.hasNextLine()) {
            String line = input.nextLine();
            Scanner inputline = new Scanner(line).useDelimiter(",");
            while (inputline.hasNextInt()) {
                int data = inputline.nextInt();
                sum += data;
                System.out.print(data + " + ");
            }
        }
        System.out.print("₩b₩b= " + String.format("%,d", sum));
    } else {
        System.out.println("파일이 존재하지 않아요");
    }
}
```

Scanner 예제 7

```
public class Main {
    public static void main(String[] args) {
        double sum = 0.0;
        try {
            PrintWriter writer = new PrintWriter(
                                        new PrintWriter(".₩₩data₩₩output.txt"));
            writer.println(9.5);
            writer.println(567.000);
            writer.flush();
            writer.close();
```

Scanner 예제 7

```
            Scanner scanner = new Scanner(new BufferedReader(
                                        new FileReader(".₩₩data₩₩output.txt")));
            while (scanner.hasNext()) {
                if (scanner.hasNextDouble())
                    sum += scanner.nextDouble();
                else
                    scanner.next();
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        }
        System.out.println(sum + "");
    }
}
```