CSE2016 Programming Methodology

Week 10: Text & File Processing

Instructor: Jinyoung Han (jinyounghan@hanyang.ac.kr)

HANYANG UNIVERSITY

Contents

HANYANG UNIVERSITY

Today's Schedule

01. String Tokenizer

Tokenizer

- We want to divide a String into multiple sub-strings with semantics
  - Example
    - “20170001,John,Male”
      - “20070001”, “John”, “Male”

- Token
  - A word with semantic, or a string
  - Can be delimited by a “delimiter”
    - E.g., “,”, “ ”, “/”

01. String Tokenizer

Tokenizer

- Java.util.StringTokenizer

```
import java.util.*;

public class Test1
{
    public static void main(String[] args)
    {
        String s = "20180001,John,Male";
        StringTokenizer t = new StringTokenizer(s, ",");
        System.out.println(t.nextToken());
        System.out.println(t.nextToken());
        System.out.println(t.nextToken());
    }
}
```

01. String Tokenizer

Tokenizer

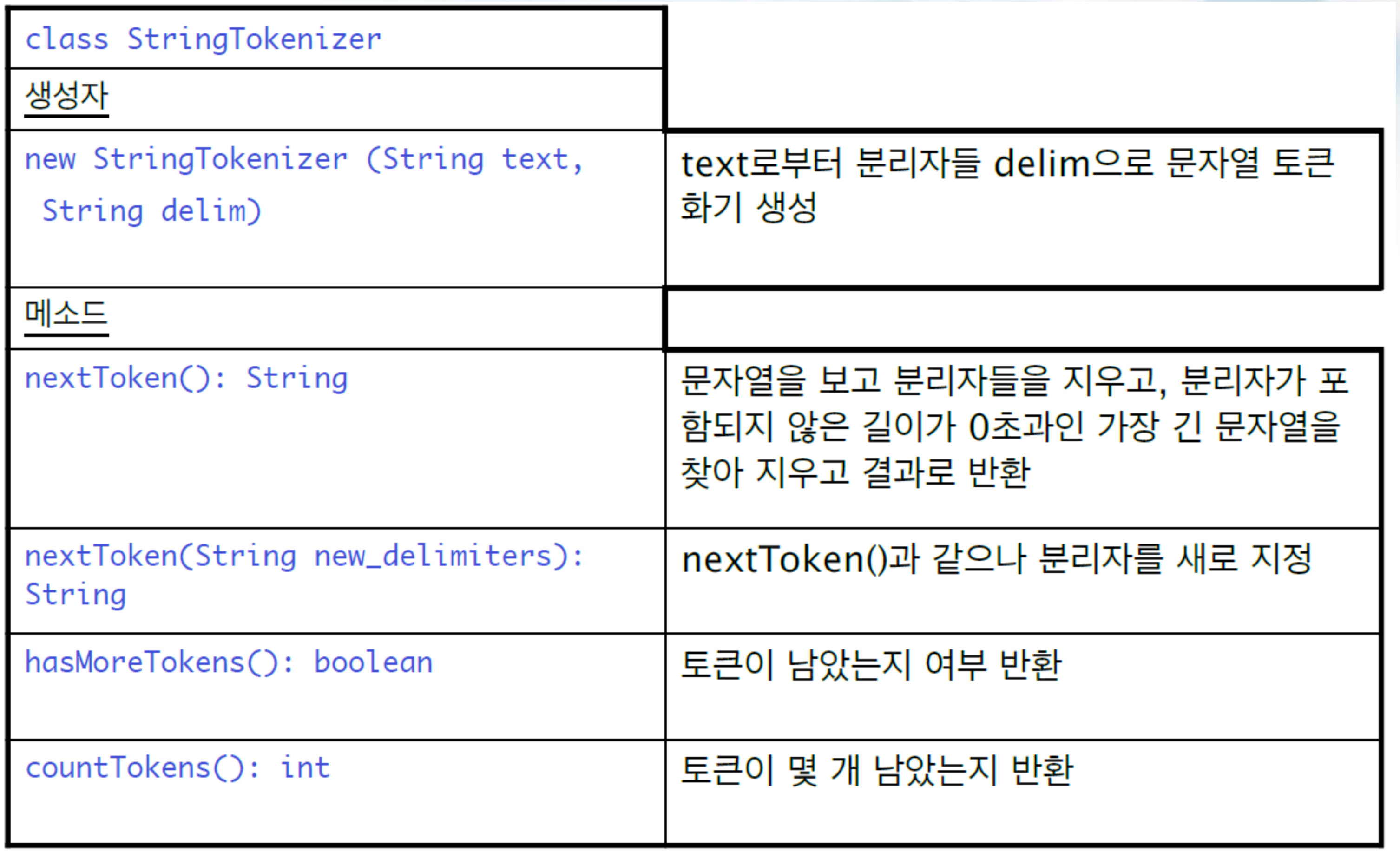
- Multiple delimiters can be applied

```
String s2 = "$13.46";
StringTokenizer t2 = new StringTokenizer(s2, "$.");
System.out.println(t2.nextToken());
System.out.println(t2.nextToken());
```

01. String Tokenizer

Tokenizer

- class StringTokenizer



class StringTokenizer	
생성자	
new StringTokenizer (String text, String delim)	text로부터 분리자들 delim으로 문자열 토큰화기 생성
메소드	
nextToken(): String	문자열을 보고 분리자들을 지우고, 분리자가 포함되지 않은 길이가 0초과인 가장 긴 문자열을 찾아 지우고 결과로 반환
nextToken(String new_delimiters): String	nextToken()과 같으나 분리자를 새로 지정
hasMoreTokens(): boolean	토큰이 남았는지 여부 반환
countTokens(): int	토큰이 몇 개 남았는지 반환

02. Files

Files

- For saving data permanently, we can use files

- For using files
  - **Open**, and read/write
    - We can open a file for (i) reading, (ii) writing, and (iii) both
  - **Close,** after all work

02. Files

File Output

- FileWriter object
  - Has the file address
  - Has methods for writing the file
  - Opens the file with write-mode
    - If not exist, a new file is created

- PrintWriter object
  - Has the FileWriter object
  - Has methods such as print, println, etc.

```
PrintWriter ofile = new PrintWriter(new FileWriter("file.txt"));
```

02. Files

Example

```
import java.io.*;

public class Test2
{
    public static void main(String[] args) throws IOException
    {
        PrintWriter outfile = new PrintWriter(new
        FileWriter("test.txt"));
        outfile.println("Hello to you!");
        outfile.print("How are");
        outfile.println(" you?");
        outfile.println(47+2);
        outfile.close();
    }
}
```

02. Files

HANYANG UNIVERSITY

File Input

- FileReader object
  - Has the file address
  - Has methods for reading the file
  - Opens the file with read-mode
    - If not exist, an exception occurs

- BufferedReader object
  - Has the FileReader object
  - Has methods such as readLine, etc.

```
BufferedReader ifile = new BufferedReader(new FileReader("file.txt"));
```

02. Files

Example

```
String f = JOptionPane.showInputDialog("Input filename, please: ");
BufferedReader infile = new BufferedReader(new FileReader(f));
PrintWriter outfile2 = new PrintWriter(new FileWriter(f + ".out"));
while (infile.ready())
{
        outfile2.println(infile.readLine());
}
infile.close();
outfile2.close();
```

ready(): Tells whether this stream is ready to be read.
readLine(): Reads a line of text.

02. Files

Chooser

```
JFileChooser chooser = new JFileChooser();
chooser.setDialogTitle("Select a file.");
int result = chooser.showDialog(null, "Copy");
if(result != JFileChooser.APPROVE_OPTION)
       System.exit(0);
String f = chooser.getSelectedFile().toString();
BufferedReader infile = new BufferedReader(new FileReader(f));
PrintWriter outfile = new PrintWriter(new FileWriter(f + ".out"));
while (infile.ready())
{
       outfile.println(infile.readLine());
}
infile.close();
outfile.close();
```

showDiaglog: Pops a custom file chooser dialog with a custom approve button.
APPROVE_OPTION : Return value if approve (yes, ok) is chosen.
getSelectedFile: Returns the selected file.

02. Files

An Example: Payroll

- Write PayrollReader and PayrollWriter
  - PayrollReader: reading the file
  - PayrollWriter: writing the file

```
A|31|20250
B|42|24500
C|18|18000
!
```

Name|#hours|hourly wage

$\Rightarrow$

```
A|627750
B|1029000
C|324000
!
```

Name|total salary

02. Files

Specification

class PayrollReader	
methods:	
getNextRecord(): boolean	Reading the next record
nameOf(): String	Return the name
hourseOf(): int	Return the hours
payrateOf(): int	Return the par rate
close()	Closing the file

class PayrollWriter	
methods:	
printCheck()	Writing the name and his/her salary
close()	Closing the file

02. Files

PayrollReader

```
public class PayrollReader
{
    private BufferedReader infile;
    private String END_OF_FILE = "!";
    private String name;
    private int hours, payrate;

    public PayrollReader(String file_name)
    {
        infile = new BufferedReader(new FileReader(file_name));
    }
    public String nameOf() { return name; }
    public int hoursOf() { return hours; }
    public int payrateOf() { return payrate; }
    public void close() { infile.close(); }
```

02. Files

PayrollReader

```
public boolean getNextRecord()
{
    if (!infile.ready()) return false;
    String line = infile.readLine();
    StringTokenizer t = new StringTokenizer(line, "|");
    String s = t.nextToken().trim();
    if (s.equals(END_OF_FILE) || t.countTokens() != 2) return false;
    name = s;
    hours = new Integer(t.nextToken().trim()).intValue();
    payrate = new Integer(t.nextToken().trim()).intValue();
    return true;
}
```

02. Files

Controller

```
public class Payroll
{
    public static void main(String[] args)
    {
        String in_name = JOptionPane.showInputDialog("Please type input payroll name: ");
        String out_name = JOptionPane.showInputDialog("Please type output payroll name: ");
        if ( in_name != null  &&  out_name != null )
                processPayroll(in_name, out_name);
    }

    private static void processPayroll(String in, String out)
    {
        PayrollReader reader = new PayrollReader(in);
        PayrollWriter writer = new PayrollWriter(out);
        while (reader.getNextRecord())
        {
            double pay = reader.hoursOf() * reader.payrateOf();
             writer.printCheck(reader.nameOf(), pay);
        }
        reader.close();
        writer.close();
    }
}
```

03. Secure Coding

Secure Coding

- What if there are some problems in getNextRecord?
    - If there is a problem in reading the file?
    - If a record is an unexpected form?

- Expected risk handling ways
    - If there is a problem in the file, return “false”
    - If there is a problem in the record, read the next record
    - If there is any problem, write error messages in the log file

03. Secure Coding

Secure Coding

- How can we know/handle “errors”?
  - Problem in “infile.readLine”
    - IOException error
  - t.countToken()!=2
    - Error in the record
    - Need to read the next record

03. Secure Coding

Try and catch

- Try and catch
  - If there is a problem in the body of “try”
    - Escape from the body
  - exception handling in “catch”
    - Only for a few types of errors

```
try{
		statements;
		…
}
catch (type..){
		statements;
		…
}
```

03. Secure Coding

Not Secure

```
public boolean getNextRecord()
{
    if (!infile.ready()) return false;
    String line = infile.readLine();
    StringTokenizer t = new StringTokenizer(line, "|");
    String s = t.nextToken().trim();
     if (s.equals(END_OF_FILE) || t.countTokens() != 2) return false;
     name = s;
     hours = new Integer(t.nextToken().trim()).intValue();
     payrate = new Integer(t.nextToken().trim()).intValue();
     return true;
}
```

03. Secure Coding

HANYANG UNIVERSITY

File Error Handling

```
try
{
    if (infile.ready())
    {
        String line = infile.readLine();
        StringTokenizer t = new StringTokenizer(line, "|");
        String s = t.nextToken().trim();
        …
}
catch (IOException e)
{
       System.out.println("PayrollReader error: " + e.getMessage());
}
```

03. Secure Coding

Record Error Handling

```
…
if (t.countTokens() == 2)
{
        name = s;
        hours = new Integer(t.nextToken().trim()).intValue();
        payrate = new Integer(t.nextToken().trim()).intValue();
        result = true;
}
else
{
        // printing error messages
        result = getNextRecord();
}
```

03. Secure Coding

Error Handlings

```
try
{
     if (infile.ready())
     {
           …
           if (!s.equals(END_OF_FILE))
           {
                      if (t.countTokens() == 2)
                      {
                                 …
                      }
                      else
                      {
                                 throw new RuntimeException(line);
                      }
           }
     }
}
catch (IOException e){ System.out.println("PayrollReader error: " + e.getMessage()); }
catch (RuntimeException e)
{
           System.out.println("PayrollReader error: bad record format: " + e.getMessage() + "  Skipping 
record");
           result = getNextRecord();  // try again
}
```

03. Secure Coding

Exception

- Exception is an object
  - Explanations of error, e.g., where it occurs, etc.
  - Some methods
    - getMessage: info of error
    - toString: concerting the exception object to String
    - printStackTrace: tracking the exception
    - …
- catch(<type> e){…}
  - Only catching the type(or subtype) of exception
  - Type - Exception: every exception
  - Type - RuntimeException: runtime exception
  - Type - IOException: file exception

03. Secure Coding

Reminder: Standard I/O

- Standard output
  - System.out
    - System.out.println, …, etc.

- Standard input
  - System.in
  - We can use:
    - BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
      - String s = keyboard.readLine();
    - Scanner sc = new Scanner(System.in)
      - String s = sc.next();

03. Secure Coding

Example

```
public int readAnIntFrom(BufferedReader view) throws IOException
{
    int num;
    try
    {
        System.out.print("Type an int: ");
        String s = view.readLine();
        num = new Integer(s).intValue();
    }
    catch (Exception e)
    {
        System.out.println("Error: " + e.getMessage() + " not an
        integer; try again.");
        num = readAnIntFrom(view); // restart readAnIntFrom
    }
    return num;
}
```

04. Summary

Summary

- String Handing
- File Handling
- Secure Coding

Thanks

Week 10: Text & File Processing
Instructor: Jinyoung Han (jinyounghan@hanyang.ac.kr)