COMP217
JAVA Programming
Summer 2018

Day 13

File I/O

Textbook:
Power Java 2: Chapters 24

File I/O: Goals

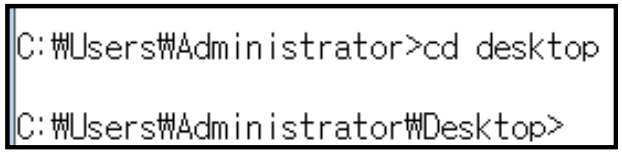
- 오늘 배우게 될 내용 :
  - 파일 클래스
  - 텍스트 입출력

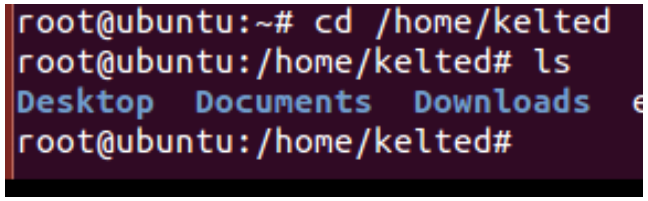
- Motivation
  - 외부 데이터를 자바 프로그램으로 읽어 오려면?
    - 키보드, 파일, …
  - 자바 프로그램 실행 중의 데이터를 외부로 내보내려면?
    - 모니터, 파일, …

- 파워 자바 24장(입출력)

File 클래스

- 파일 or 디렉토리를 나타내는 클래스
  - directory seperator
    - 윈도우 : ₩, 리눅스 : /
  - 플랫폼에 독립적인 표현 가능
    - File클래스의 static 변수 separator 사용
    - 상대경로 사용





File 클래스

반환형	메소드	설명
boolean	canExecute()	파일을 실행할 수 있는지의 여부
boolean	canRead()	파일을 읽을 수 있는지의 여부
boolean	canWrite()	파일을 변경할 수 있는지의 여부
static File	createTempFile(String prefix, String suffix)	임시 파일을 생성한다.
boolean	delete()	파일을 삭제한다.
void	deleteOnExit()	가상 기계가 종료되면 파일을 삭제한다.
boolean	exists()	파일의 존재 여부
String	getAbsolutePath()	절대 경로를 반환
String	getCanonicalPath()	정규 경로를 반환
String	getName()	파일의 이름을 반환
String	getParent()	부모 경로 이름을 반환
File	getParentFile()	부모 파일을 반환
boolean	isDirectory()	디렉토리이면 참
boolean	isFile()	파일이면 참
long	lastModified()	파일이 변경되었는지 여부
long	length()	파일 길이 반환
String[]	list()	디렉토리 안에 포함된 파일과 디렉토리를 반환
boolean	mkdir()	디렉토리를 생성한다.
boolean	renameTo(File dest)	파일 이름을 변경한다.
boolean	renameTo(File dest)	파일 이름을 변경한다.
boolean	setExecutable(boolean executable)	파일을 실행 가능하게 설정
boolean	setLastModified(long time)	파일을 변경된 것으로 설정

파일 클래스 테스트

```
import java.io.File;
import java.util.Date;

class FileTestLoop{
  public static void main(String [] args){
    File[] f = {new File("F:\\COMP217\\강의자료\\13일차"), new
   File("C:"+File.separator+"Users"), new File("FileTest.java")};
    for(File file:f){
      System.out.println("존재? " + file.exists());
      System.out.println("파일 크기: " + file.length() + " 바이트");
      System.out.println("읽기여부: " + file.canRead());
      System.out.println("쓰기여부: " + file.canWrite());
      System.out.println("폴더여부: " + file.isDirectory());
      System.out.println("파일여부: " + file.isFile());
      System.out.println("절대경로? " + file.isAbsolute());
      System.out.println("절대경로: " + file.getAbsolutePath());
      System.out.println("마지막수정일: " + new Date(file.lastModified()));

   System.out.println("==========================================")
   ;
    }
  }
}
```

Text I/O

A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file. In order to perform I/O, you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file. This section introduces how to read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes.

Writing Data Using PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §3.6, "Formatting Console Output and Strings."

WriteData.java

```
public class WriteData {
  public static void main(String[] args) throws Exception {
    java.io.File file = new java.io.File("scores.txt");
    if (file.exists()) {
      System.out.println("File already exists");
      System.exit(0);
    }

    // Create a file
    java.io.PrintWriter output = new java.io.PrintWriter(file);

    // Write formatted output to the file
    output.print("John T Smith ");
    output.println(90);
    output.print("Eric K Jones ");
    output.println(85);

    // Close the file
    output.close();
  }
}
```

WriteDataWithAutoClose.java

```
public class WriteDataWithAutoClose {
  public static void main(String[] args) throws Exception {
    java.io.File file = new java.io.File("scores.txt");
    if (file.exists()) {
      System.out.println("File already exists");
      System.exit(0);
    }

    try (
      // Create a file
      java.io.PrintWriter output = new java.io.PrintWriter(file);
    ) {
      // Write formatted output to the file
      output.print("John T Smith ");
      output.println(90);
      output.print("Eric K Jones ");
      output.println(85);
    }
  }
}
```

Reading Data Using Scanner

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner object to read data from the specified file.
+Scanner(source: String)	Creates a Scanner object to read data from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

ReadData.java

```
import java.util.Scanner;

public class ReadData {
  public static void main(String[] args) throws Exception {
    // Create a File instance
    java.io.File file = new java.io.File("scores.txt");

    // Create a Scanner for the file
    Scanner input = new Scanner(file);

    // Read data from a file
    while (input.hasNext()) {
      String firstName = input.next();
      String mi = input.next();
      String lastName = input.next();
      int score = input.nextInt();
      System.out.println(
        firstName + " " + mi + " " + lastName + " " + score);
    }

    // Close the file
    input.close();
  }
}
```

Problem: Replacing Text

Write a class named ReplaceText that replaces a string in a text file with a new string. The filename and strings are passed as command-line arguments as follows:

```
java ReplaceText sourceFile targetFile oldString newString
```

For example, invoking

```
java ReplaceText FormatString.java t.txt StringBuilder
  StringBuffer
```

replaces all the occurrences of StringBuilder by StringBuffer in FormatString.java and saves the new file in t.txt.

ReplaceText.java

```
import java.io.*;
import java.util.*;

public class ReplaceText {
  public static void main(String[] args) throws Exception {
    // Check command line parameter usage
    if (args.length != 4) {
      System.out.println("Usage: java ReplaceText sourceFile targetFile oldStr newStr");
      System.exit(1);
    }

    // Check if source file exists
    File sourceFile = new File(args[0]);
    if (!sourceFile.exists()) {
      System.out.println("Source file " + args[0] + " does not exist");
      System.exit(2);
    }

    // Check if target file exists
    File targetFile = new File(args[1]);
    if (targetFile.exists()) {
      System.out.println("Target file " + args[1] + " already exists");
      System.exit(3);
    }

    try (
      // Create input and output files
      Scanner input = new Scanner(sourceFile);
      PrintWriter output = new PrintWriter(targetFile);
    ) {
      while (input.hasNext()) {
        String s1 = input.nextLine();
        String s2 = s1.replaceAll(args[2], args[3]);
        output.println(s2);
      }
    }
  }
}
```