Reading Text Files

Berk Gökberk

Reading Data from Text Files

- You can read data from text files using File and Scanner classes

- Place the input file at location:
  `YourEclipseWorkspace\YourApp\input.txt`

- Import the following libraries:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
```

input.txt

```
#City xCoordinate yCoordinate Population
Istanbul 41.00 28.97 32000
Izmir 38.41 27.12 18000
Athens 37.98 23.72 9000
Rome 41.90 12.49 3200
```

Example: Reading Text File Part 1/2

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class AppFile {
 public static void main(String[] args) throws FileNotFoundException {

  // Filename of the input file
  String fileName = "input.txt";

  // file object is required to open the file
  File file = new File(fileName);

  // if the file is not found, issue an error message and quit
  if (!file.exists()) {
   System.out.printf("%s can not be found.", fileName);
   System.exit(1); // exit the program
  }
  // code continues
}
```

Example: Reading Text File Part 2/2

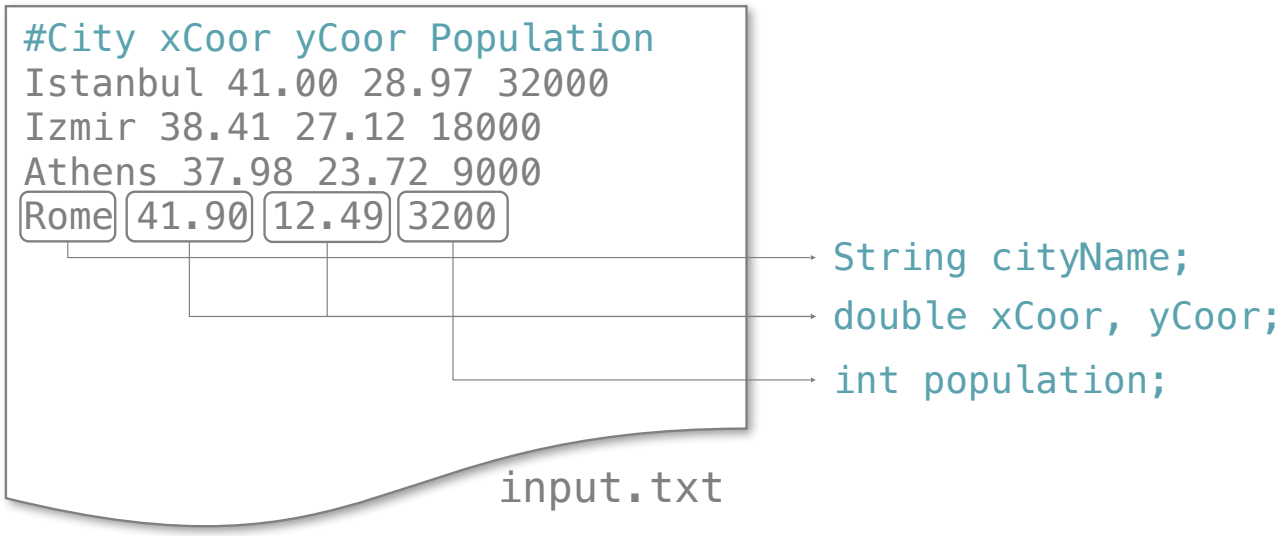
```
public class AppFile {
 public static void main(String[] args) throws FileNotFoundException {
  // code continues from here

  // scanner object is required to read the contents of the file
  Scanner inputFile = new Scanner(file);

  // continue reading file contents if there is a line to be read
  while (inputFile.hasNextLine()) {
   String line = inputFile.nextLine(); // get the current line as a string
   System.out.println(line); // print the line
  }
  inputFile.close(); // close the scanner object
 }
}
```

Program output

```
#City xCoordinate yCoordinate Population
Istanbul 41.008240 28.978359 32000
Izmir 38.418720 27.129601 18000
Athens 37.983810 23.727539 9000
Rome 41.902782 12.496365 3200
```

Reading Data from Text Files

- In the previous example, we use `inputFile.nextLine()` to read the complete line and store it in a string variable
- Now, let's try to get each city information in a corresponding variable:
  - City name to a String variable: `cityName`
  - x and y coordinates to double variables: `xCoor, yCoor`
  - Population to an integer variable: `population`



Reading Data from Text Files

```java
// read the explanation line first
String firstLine = inputFile.nextLine();

while (inputFile.hasNextLine()) {

 String cityName = inputFile.next();
 double xCoor = inputFile.nextDouble();
 double yCoor = inputFile.nextDouble();
 int population = inputFile.nextInt();

 System.out.printf("City name: %10s, ", cityName);
 System.out.printf("x: %.2f, y: %.2f, ", xCoor, yCoor);
 System.out.printf("Population: %4d\n", population);

}
inputFile.close(); // close the scanner object
```

```
#City xCoor yCoor Population
Istanbul 41.00 28.97 32000
Izmir 38.41 27.12 18000
Athens 37.98 23.72 9000
Rome 41.90 12.49 3200
```

input.txt

Reading Data from Text Files

```
// read the explanation line first
String firstLine = inputFile.nextLine();

while (inputFile.hasNextLine()) {

 String cityName = inputFile.next();
 double xCoor = inputFile.nextDouble();
 double yCoor = inputFile.nextDouble();
 int population = inputFile.nextInt();

 System.out.printf("City name: %10s, ", cityName);
 System.out.printf("x: %.2f, y: %.2f, ", xCoor, yCoor);
 System.out.printf("Population: %4d\n", population);

}
inputFile.close(); // close the scanner object
```

next() reads a string

```
#City xCoor yCoor Population
Istanbul 41.00 28.97 32000
Izmir 38.41 27.12 18000
Athens 37.98 23.72 9000
Rome 41.90 12.49 3200
```

input.txt

Reading Data from Text Files

```
// read the explanation line first
String firstLine = inputFile.nextLine();

while (inputFile.hasNextLine()) {

 String cityName = inputFile.next();
 double xCoor = inputFile.nextDouble();
 double yCoor = inputFile.nextDouble();
 int population = inputFile.nextInt();

 System.out.printf("City name: %10s, ", cityName);
 System.out.printf("x: %.2f, y: %.2f, ", xCoor, yCoor);
 System.out.printf("Population: %4d\n", population);

}
inputFile.close(); // close the scanner object
```

nextDouble() reads a double

```
#City xCoor yCoor Population
Istanbul 41.00 28.97 32000
Izmir 38.41 27.12 18000
Athens 37.98 23.72 9000
Rome 41.90 12.49 3200
```

input.txt

Reading Data from Text Files

```
// read the explanation line first
String firstLine = inputFile.nextLine();

while (inputFile.hasNextLine()) {

 String cityName = inputFile.next();
 double xCoor = inputFile.nextDouble();
 double yCoor = inputFile.nextDouble();
 int population = inputFile.nextInt();

 System.out.printf("City name: %10s, ", cityName);
 System.out.printf("x: %.2f, y: %.2f, ", xCoor, yCoor);
 System.out.printf("Population: %4d\n", population);

}
inputFile.close(); // close the scanner object
```

nextDouble() reads a double

```
#City xCoor yCoor Population
Istanbul 41.00 28.97 32000
Izmir 38.41 27.12 18000
Athens 37.98 23.72 9000
Rome 41.90 12.49 3200
```

input.txt

Reading Data from Text Files

```
// read the explanation line first
String firstLine = inputFile.nextLine();

while (inputFile.hasNextLine()) {

 String cityName = inputFile.next();
 double xCoor = inputFile.nextDouble();
 double yCoor = inputFile.nextDouble();
 int population = inputFile.nextInt();

 System.out.printf("City name: %10s, ", cityName);
 System.out.printf("x: %.2f, y: %.2f, ", xCoor, yCoor);
 System.out.printf("Population: %4d\n", population);

}
inputFile.close(); // close the scanner object
```

nextInt() reads an integer

```
#City xCoor yCoor Population
Istanbul 41.00 28.97 32000
Izmir 38.41 27.12 18000
Athens 37.98 23.72 9000
Rome 41.90 12.49 3200
```

input.txt

Reading Data from Text Files

```
// read the explanation line first
String firstLine = inputFile.nextLine();

while (inputFile.hasNextLine()) {

 String cityName = inputFile.next();
 double xCoor = inputFile.nextDouble();
 double yCoor = inputFile.nextDouble();
 int population = inputFile.nextInt();
 System.out.printf("City name: %10s, ", cityName);
 System.out.printf("x: %.2f, y: %.2f, ", xCoor, yCoor);
 System.out.printf("Population: %4d\n", population);
}
inputFile.close(); // close the scanner object
```

Program output

```
City name:   Istanbul, x: 41.01, y: 28.98, Population:   32
City name:      Izmir, x: 38.42, y: 27.13, Population:   18
City name:     Athens, x: 37.98, y: 23.73, Population:    9
City name:       Rome, x: 41.90, y: 12.50, Population:   55
```

Methods of the Scanner Class

- You can read string, double, integer, float etc. using the scanner class
  - `nextDouble()` for reading a double number
  - `nextInt()` for reading an integer
  - `next()` for reading a single string
  - `nextLine()` for reading the complete line as a string
- `hasNext()` method returns true if there is more data to be read
  - Usually used as a loop condition
- `close()` closes the scanner object.
  - It is recommended to use the `close()` method when reading is finished

hasNext() vs hasNextLine()

- You can also read data using hasNext() as a loop condition
- hasNext() returns true if there is still data to be read in the input text file

```
while (inputFile.hasNext()) {

  String studentName = inputFile.next();
  int age = inputFile.nextInt();

}
```

Scanner Class

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner that produces values scanned from the specified file.
+Scanner(source: String)	Creates a Scanner that produces values scanned from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has more data to be read.
+next(): String	Returns next token as a string from this scanner.
+nextLine(): String	Returns a line ending with the line separator from this scanner.
+nextByte(): byte	Returns next token as a byte from this scanner.
+nextShort(): short	Returns next token as a short from this scanner.
+nextInt(): int	Returns next token as an int from this scanner.
+nextLong(): long	Returns next token as a long from this scanner.
+nextFloat(): float	Returns next token as a float from this scanner.
+nextDouble(): double	Returns next token as a double from this scanner.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern and returns this scanner.

File Class

- The File class can be used to obtain file and directory properties, to delete and rename files and directories, and to create directories

java.io.File	
+File(pathname: String)	Creates a File object for the specified path name. The path name may be a directory or a file.
+File(parent: String, child: String)	Creates a File object for the child under the directory parent. The child may be a file name or a subdirectory.
+File(parent: File, child: String)	Creates a File object for the child under the directory parent. The parent is a File object. In the preceding constructor, the parent is a string.
+exists(): boolean	Returns true if the file or the directory represented by the File object exists.
+canRead(): boolean	Returns true if the file represented by the File object exists and can be read.
+canWrite(): boolean	Returns true if the file represented by the File object exists and can be written.
+isDirectory(): boolean	Returns true if the File object represents a directory.
+isFile(): boolean	Returns true if the File object represents a file.
+isAbsolute(): boolean	Returns true if the File object is created using an absolute path name.
+isHidden(): boolean	Returns true if the file represented in the File object is hidden. The exact definition of *hidden* is system dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period (.) character.
+getAbsolutePath(): String	Returns the complete absolute file or directory name represented by the File object.
+getCanonicalPath(): String	Returns the same as getAbsolutePath() except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
+getName(): String	Returns the last name of the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getName() returns test.dat.
+getPath(): String	Returns the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getPath() returns c:\book\test.dat.
+getParent(): String	Returns the complete parent directory of the current directory or the file represented by the File object. For example, new File("c:\\book\\test.dat").getParent() returns c:\book.
+lastModified(): long	Returns the time that the file was last modified.
+length(): long	Returns the size of the file, or 0 if it does not exist or if it is a directory.
+listFile(): File[]	Returns the files under the directory for a directory File object.
+delete(): boolean	Deletes the file or directory represented by this File object. The method returns true if the deletion succeeds.
+renameTo(dest: File): boolean	Renames the file or directory represented by this File object to the specified name represented in dest. The method returns true if the operation succeeds.
+mkdir(): boolean	Creates a directory represented in this File object. Returns true if the the directory is created successfully.
+mkdirs(): boolean	Same as mkdir() except that it creates directory along with its parent directories if the parent directories do not exist.

Example

Reading text from a file using delimiters

Example: Using Delimiters to Parse Data

- When the line to be read contains data separated by a special delimiter symbol, you can use `split()` (in the String class) method to automatically split string into parts
- Example:
  - File contains student info name, surname and grades separated by the ; symbol
  - Each student has different number of grades

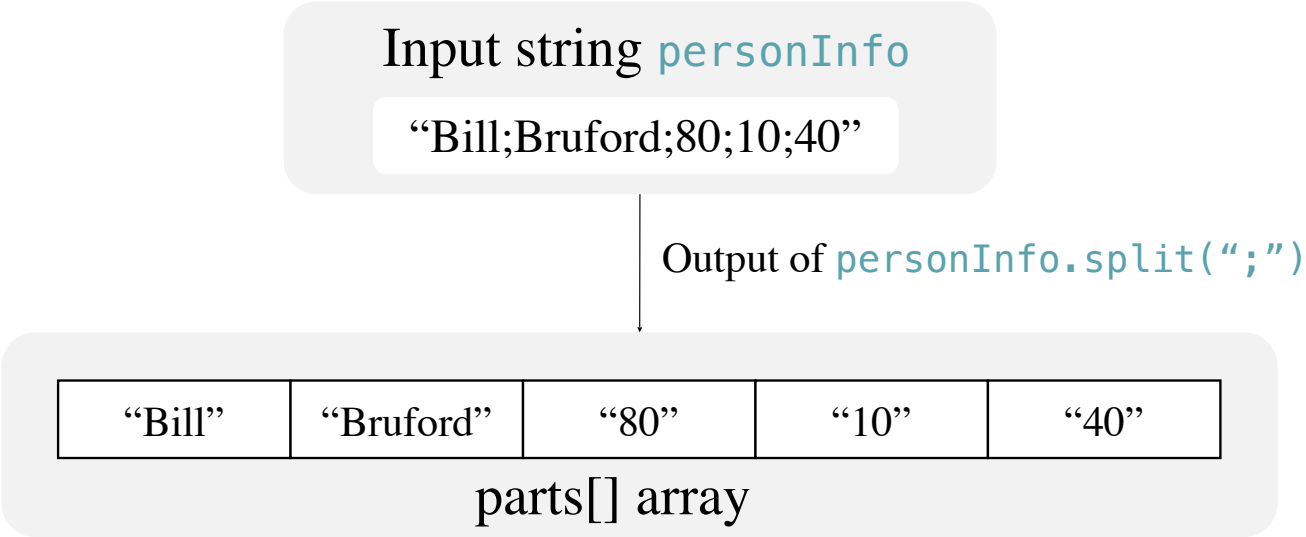
```
%Explanation of rows: Name;Surname;Grades
Robert;Fripp;30;40;60;10;20;30;55;33;12;40;12
John;Wetton;20;90;70
Bill;Bruford;80;10;40;40;40
```

input.txt

Example: Using Delimiters to Parse Data

- When the line to be read contains data separated by a special delimiter symbol, you can use `split()` (in the String class) method to automatically split string into parts
- Split() method returns a string array:

```
String personInfo = “Bill;Bruford;80;10;40”;
String[] parts = personInfo.split(“;”) // ; is the delimiter
```



Converting String to Integer/Double

- You can convert a string to an integer using `Integer.parseInt()` method
  `int grade = Integer.parseInt("69") // converts the string "69" to an integer`
- Similarly, you can convert a string to a double using `Double.parseInt()`
  `double height = Double.parseInt("1.92") // converts the string "1.92" to double`

```
public class AppStringConversion {
 public static void main(String[] args) {
  String str1 = "12"; // string containing an integer value
  String str2 = "2.4"; // string containing a double value

  int val1 = Integer.parseInt(str1);
  double val2 = Double.parseDouble(str2);

  System.out.println(2 * val1);
  System.out.println(2 * val2);
 }
}
```

Example: Using Delimiters to Parse Data

```
String firstLine = inputFile.nextLine(); // read the first explanation line

while (inputFile.hasNextLine()) {

 String line = inputFile.nextLine(); // get the current line as a string
 String[] strParts = line.split(";"); // split the line into strings
 String name = strParts[0]; // first and second items are name and surname
 String surname = strParts[1];

 // rest of the array elements are grades. Place them into an integer array
 int[] grades = new int[strParts.length-2]; // create the array
 for (int i = 2; i < strParts.length; i++)
  grades[i-2] = Integer.parseInt(strParts[i]); // convert string to an integer

 System.out.printf("Name %7s, Surname: %7s, " , name, surname);
 System.out.println("Grades: " + Arrays.toString(grades));
}
```

Program output

```
Name  Robert, Surname:   Fripp, Grades: [30, 40, 60, 10, 20, 30, 55, 33, 12, 40, 12]
Name    John, Surname:  Wetton, Grades: [20, 90, 70]
Name    Bill, Surname: Bruford, Grades: [80, 10, 40, 40, 40]
```