**CS 1302 – Lab 10**

When a program runs, it may accumulate data, for example in an array list. However, that data is gone when the program ends. Clearly, to write real systems, we must have data that persists between running a program again. One way to persist data is in [*text files*](https://en.wikipedia.org/wiki/Text_file), which contain simple text with no formatting. This lab describes how to read data from a text file into your program, and, how to write data you have in your program into text files.

This lab introduces reading and writing text files. There are 9 stages to complete this lab:

|  |  |
| --- | --- |
| **Stage** | **Title** |
| 1 | Where is my File? |
| 2 | More on Reading a Text File |
| 3 | Reading & Creating Objects |
| 4 | String.split() |
| 5 | Parsing Data |
| 6 | More Parsing |
| 7 | Writing Data to a Text File |
| 8 | The *File* Class |
| 9 | Reading & Writing Example |

To make this document easier to read, it is recommended that you turn off spell checking in Word:

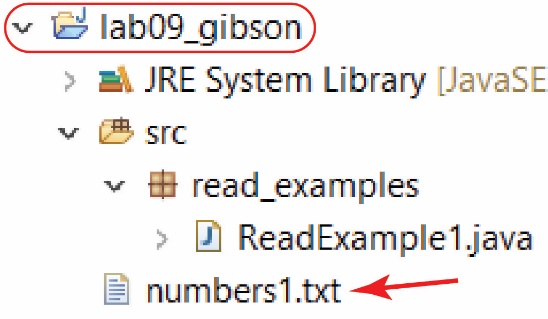
1. Choose: File, Option, Proofing
2. At the very bottom, check: “Hide spelling errors…” and “Hide grammar errors…”
3. **Where is my File?**

Specifying the location of a text file that is to be read or written to, using Eclipse and packages can be a bit confusing. By default, Java is looking for the text file in the project folder. We will want the text file to be in the package folder. To do this, we simply set the path to this location. Below, we illustrate this.

1. **(Read, no action required)**
2. You remember that to read from the keyboard, you use the *Scanner* class with code like this:

Scanner scanner = **new** Scanner(System.***in***);

String input = scanner.nextLine();

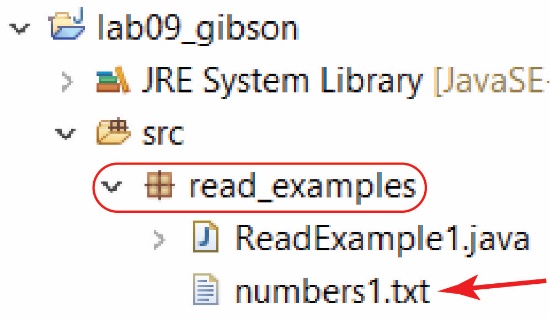
1. We can also use the *Scanner* class to read from a text file. Suppose we want to read a text file named: *numbers1.txt*. We first create a *File* object with the *fileName* and then create a *Scanner* object with the *File* object. For example:

String fileName = "numbers1.txt";

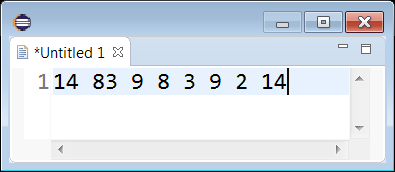
File inFile = **new** File(fileName);

Scanner input = **new** Scanner(inFile);

If the code above is in *ReadExample1.java* located in the *read\_examples* package as shown on the right, then it will be expecting the text file to be located in the project folder, *lab10\_gibson.*

1. For this class, we will want the text file in the same package as the code, as shown on the right. This makes grading a bit easier and eliminates confusion if there are multiple packages and text files. In order to read from the package folder, we must provide a path there: src/package\_name/. For example, to read *numbers1.txt* from the *read\_examples* package:

String fileName = "src/read\_examples/numbers1.txt";

1. Consider a text file that contains a list of numbers separated by a space (the initial, gray, “1” is the line number and is not part of the text file). Next, we describe the code to read and print the values:
2. Create *File* object:

String fileName = "src/read\_examples/numbers1.txt";

File inFile = **new** File(fileName);

1. See if the file exists:

**boolean** doesExist = inFile.exists();

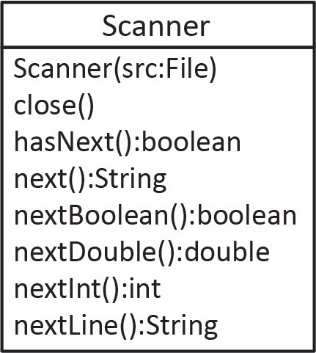
**if**(doesExist) {

**try** {

1. If file exists, create *Scanner* object. Note that Java requires us to use *try/catch* (above) when using a *Scanner* object. Many I/O (input/output) methods can throw exceptions, which are called checked exceptions which must be caught.

Scanner input = **new** Scanner(inFile);

1. Loop until everything has been read. By default, the *Scanner* assumes the file has “tokens” (a character, or string of characters) separated by a space(s). The *Scanner’s hasNext* method returns *true* if there is something else in the file, *i.e.* if there is another token to read.



**while**(input.hasNext()) {

1. The *Scanner’s next* method returns the next token (as a *String*). Some of the *Scanner* methods we will use are shown on the right. Here, we could have use, *nextInt* instead, since the numbers are integers.

String token = input.next();

System.***out***.print(token + " ");

}

1. Close the *Scanner* when through reading.

input.close();

}

**catch**(IOException e) {

System.***out***.println("Error reading file");

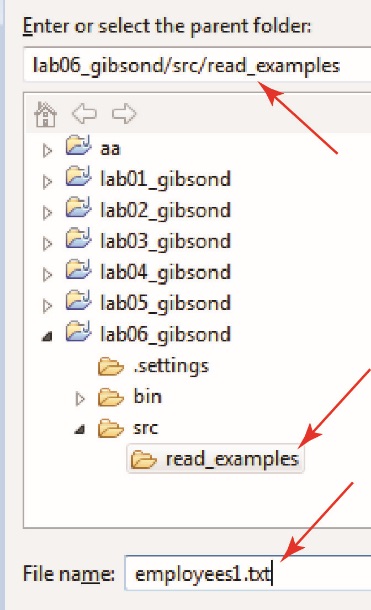
}

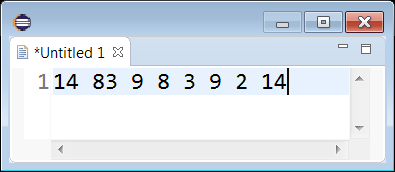
}

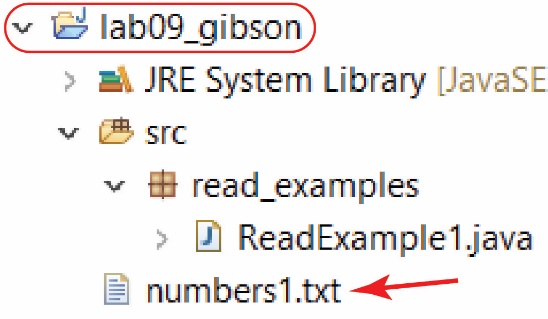
**else** {

System.***out***.println("Can't find file");

}

1. **Setup** – Create a project named, *lab10\_lastName* and create a package named: *read\_examples*.
2. **Create Text File** – Do the following:
3. Select the project folder, *lob10\_lastName*.
4. Choose: File, New, Untitled Text File (or File, New, Other, General, Untitled Text File)
5. Type in some integers on a single line:





1. Choose: File, Save As
2. Select the project folder: *lab10\_lastName*
3. Type in the file name: *numbers1.txt*
4. Press OK. The result is shown in the figure on the right.
5. **Create Class to Read Data** – Create a class named *ReadExample1* (in the *read\_examples* package)and replace everything in the class with:

**package** read\_examples;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.Scanner;

**public** **class** ReadExample1 {

**public** **static** **void** main(String[] args) {

String fileName = "numbers1.txt";

System.***out***.println("Trying to read: " + "\"" + fileName + "\"");

File inFile = **new** File(fileName);

**boolean** doesExist = inFile.exists();

System.***out***.println("Does " + "\"" + fileName + "\"" + " exist? " + doesExist);

**if**(doesExist) {

System.***out***.println("Contents of file:");

**try** {

Scanner input = **new** Scanner(inFile);

**while**(input.hasNext()) {

String token = input.next();

System.***out***.print(token + " ");

}

input.close();

}

**catch**(IOException e) {

System.***out***.println("Error reading file");

}

}

**else** {

System.***out***.println("Can't find file");

}

}

}

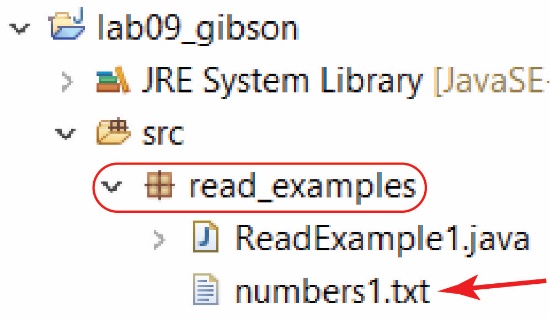
1. Run the program and verify the output by reading the code:

Trying to read: "numbers1.txt"

Does "numbers1.txt" exist? true

Contents of file:

14 83 9 8 3 9 2 14

1. Drag (don’t copy) *numbers1.txt* into the *read\_examples* package as shown on the right.
2. Run the program and verify that the file is not found:

Trying to read: "numbers1.txt"

Does "numbers1.txt" exist? false

Can't find file

1. Change the file name to include the path:

String fileName = "src/read\_examples/numbers1.txt";

1. Run the program and verify that the file is read correctly.
2. **More on Reading a Text File**
3. Close *ReadExamples1.java* and copy and paste into the same package with the name: *ReadExample2.* You can run it to make sure it still works.
4. Change this line:

String token = input.next();

To:

String token = input.nextLine();

This will read the entire line. Thus, the loop will occur only once.

1. Run and verify the output.
2. Close *ReadExamples2.java* and copy and paste into the same package with the name: *ReadExample3.* You can run it to make sure it still works.
3. Replace the body of the *while* loop with:

**int** val = input.nextInt();

System.***out***.print(val + " ");

Notice now that we are reading the “tokens” as integers using the *Scanner’s nextInt* method. This is fine as long as we know they are integers. The *Scanner* class also has *nextDouble* and *nextBoolean* methods.

1. Run and verify the output.
2. Open *numbers1.txt* and add a character somewhere in the file. For example:

14 83a 9 8 3 9 2 14

1. Run and verify that the program crashes by throwing an un-caught *InputMismatchException*.

Trying to read: "src/read\_examples/numbers1.txt"

Does "src/read\_examples/numbers1.txt" exist? true

Contents of file:

14 Exception in thread "main" java.util.InputMismatchException

at java.util.Scanner.throwFor(Unknown Source)

at java.util.Scanner.next(Unknown Source)

at java.util.Scanner.nextInt(Unknown Source)

at java.util.Scanner.nextInt(Unknown Source)

at read\_examples.ReadExample3.main(ReadExample3.java:20)

1. We could handle this situation by using *try/catch* around the call to *input.nextInt()*. Replace the code inside the *while* loop with:

**try** {

**int** val = input.nextInt();

System.***out***.print(val + " ");

}

**catch**(InputMismatchException ie) {

// Read the token as a string. Note, since, nextInt, which

// cause the exception, didn't actually read a value, the marker (cursor)

// for where to begin reading has not been advanced. Thus, the call to next

// will read the value.

String badValue = input.next();

System.***out***.print("Skipped:" + badValue + " ");

}

1. Resolve the compilation error by importing: **import** java.util.InputMismatchException; Or, you could change the exception to: RuntimeException and then no import would be needed.
2. Run, and verify the output:

Trying to read: "src/read\_examples/numbers1.txt"

Does "src/read\_examples/numbers1.txt" exist? true

Contents of file:

14 Skipped:83a 9 8 3 9 2 14

1. Remove the character from *numbers1.txt* and save.
2. **Reading and Creating Objects**
3. **Create Text File** – Do the following:
4. Create a new package: *read\_examples2*.
5. Create a text file named: *employees1.txt* and copy this data in:

Dave

44.55

33

Sue

33.44

45

Sherry

55.34

18

Mike

23.45

23

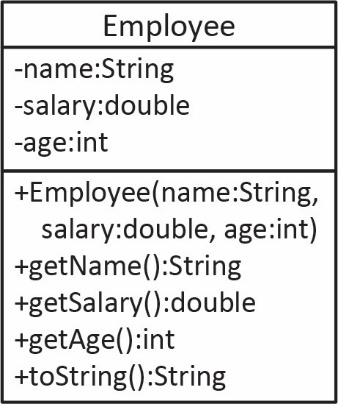
Nicole

76.34

19

Note: You will probably end up with a blank line in the file (line 16). I don’t think it will affect the reading, but probably better to use backspace to delete that line.

1. **Read (no action required)** –



1. The *Employee* class is shown on the right. Each three lines in the text file above, represents an *Employee* object: name, salary, age.
2. We will write code to read the values in, create *Employee* objects, put them in an array list, and finally print them as the algorithm below shows:
3. Create an arraylist to hold employees
4. Loop over the file
   1. Read the name, salary, and age
   2. Use them to build an *Employee* object
   3. Add employee to the arraylist.
5. Loop over the arraylist and print each employee.
6. **Create Classes to Read and Store Data** – Do the following:
7. Create a class named *Employee* (in the *read\_examples2* package)and replace everything in the class (exceptthe package statement at the top) with:

**public** **class** Employee {

**private** String name;

**private** **double** salary;

**private** **int** age;

**public** Employee(String name, **double** salary, **int** age) {

**this**.name = name;

**this**.salary = salary;

**this**.age = age;

}

**public** String getName() { **return** name; }

**public** **double** getSalary() { **return** salary; };

**public** **int** getAge() { **return** age; }

@Override

**public** String toString() {

**return** "Employee [name=" + name + ", salary=" + salary + ", age="

+ age + "]";

}

}

1. Create a class named *EmployeeReader* and replace everything in the class (except the package statement at the top) with:

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** EmployeeReader {

**public** **static** **void** main(String[] args) {

ArrayList<Employee> employees = **new** ArrayList<>();

File inFile = **new** File( "src/read\_examples2/employees1.txt" );

**try** {

Scanner input = **new** Scanner( inFile );

**while**(input.hasNext()) {

String name = input.next();

**double** salary = input.nextDouble();

**int** age = input.nextInt();

Employee e = **new** Employee(name, salary, age);

employees.add(e);

}

input.close();

}

**catch**( IOException e ) {

System.***out***.println(e);

}

**for**( Employee e : employees )

System.***out***.println(e);

}

}

Study the code in the *try* block.

1. Run and verify the output as shown below. It would be useful to run through the debugger and inspect the values as they are read.

Employee [name=Dave, salary=44.55, age=33]

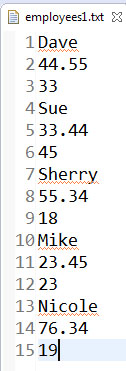
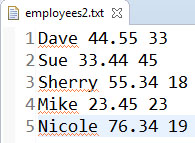
Employee [name=Sue, salary=33.44, age=45]

Employee [name=Sherry, salary=55.34, age=18]

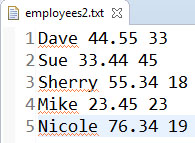
Employee [name=Mike, salary=23.45, age=23]

Employee [name=Nicole, salary=76.34, age=19]

1. **Read (no action required)** – Consider the three text files shown below. The code we wrote above to read *employees1.txt* will read any one of these properly. We will show this next.

****

**E:\Data-Classes\CS 1302 - Programming 2-Fall 2016\notes\06_ch12_Exceptions\c5.jpg**

1. **Test other Layouts of the Data** – Do the following:
2. Copy *employees1.txt* and give it the new name: *employees2.txt*.
3. Open *employees2.txt* and reformat so that it looks as shown in the figure on the right: each employee on a separate line.
4. Open *EmployeeReader* and change the reference to *employees1.txt* to *employees2.*txt.

File inFile = **new** File( "src/read\_examples2/employees2.txt" );

1. Run and verify that the output is the same as before.
2. Copy *employees2.txt* and give it the new name: *employees3.txt*.
3. Open *employees3.txt* and reformat so that it looks as shown in the figure below: all the employees on a single line.

**E:\Data-Classes\CS 1302 - Programming 2-Fall 2016\notes\06_ch12_Exceptions\c5.jpg**

1. Open *EmployeeReader* and change the reference to *employees2.txt* to *employees3.txt*.
2. Run and verify that the output is the same as before.
3. **Read (no action required)** –What is the point of this? The *Scanner* class is very easy to use and specifically, it is not concerned with the end of a line (unless *nextLine* is called). When you call any of the *next* methods (except *nextLine*), you are simply returned the next token no matter whether it is on the same line or the next line (or even separated by multiple blank lines). In more advanced situation, we have to use other classes for reading such as: *Reader, BufferedReader, FileReader*. A short discussion of the different classes is found [here](https://stackoverflow.com/questions/19430071/proper-java-classes-for-reading-and-writing-files).
4. ***String.split()***

Sometimes it is useful and/or necessary to read a text file line-by-line. As we read a line, we then use code to break it into tokens ourselves, figure out what it is, and then do the appropriate action with it. To learn how to parse data, we must learn how to break a string into tokens using the *String* class’s *split* method.

1. (**Read, no action required)** –
2. The *String* class has a *split* method that breaks a string into “tokens” based on a delimiter and returns the tokens in an array.For example:

String s1 = "43.85 66.239 8.223";

String[] vals = s1.split(" ");

Results in the *vals* array containing: "43.85", "66.239", "8.223"

The delimiter in this case is a space (" "). Thus, everything between the spaces is a token.

1. Above, we say that we “split the string on a space.” Though this will work for our examples, technically there are lots of different types of spaces[[1]](#footnote-1). This delimiter, “\s” will catch more of them. Remember, since a backslash is a delimiter in a Java string, it must be delimited with a “\”. Thus, we would write it like this:

String[] vals = s1.split("\\s");

1. The two examples above will split the string on a single space.

String[] vals = s1.split(" ");

String[] vals = s1.split("\\s");

If we want to split on any number of spaces, we do this:

String s1 = "43.85 66.239 8.223";

String[] vals = s1.split("\\s+");

Which results in the *vals* array containing: "43.85", "66.239", "8.223"

1. To split a string any of multiple characters, we surround the characters with “[ ]”. For example:

String s1 = "4,3 5,,,2, 8";

String[] vals = s1.split("[,\\s]+");

Results in the *vals* array containing: "4", "3", "5", "2", "8"

1. To split on common punctuation characters:

String s1 = "This. Is,,, \nfunny; \t\tyes: why? now!";

String[] vals = s1.split("[.,;:?!\\s]+");

Results in the *vals* array containing: "This", "is", "funny", "yes", "why", "now"

1. The *delimiter* (the argument to the *split* method) is actually a *regular expression*, which can be composed to delimit and pattern match sophisticated situations[[2]](#footnote-2).

Next, we will show some examples:

1. **Create Example** – Do the following:
2. Create a new package: *string\_split*.
3. Create a class named: *StringSplitExamples* and replace everything in the class (except the package statement at the top) with:

**public** **class** StringSplitExamples {

**public** **static** **void** main(String[] args) {

*split\_example1*();

*split\_example2*();

*split\_example3*();

*split\_example4*();

*split\_example5*();

}

// Splits on a single space

**public** **static** **void** split\_example1() {

System.***out***.println("split\_example1()");

String s1 = "43.85 66.239 8.223";

String[] vals = s1.split("\\s");

*printArray*(s1, "\\\\s", vals);

}

// There are two spaces between the first and second numbers.

// Thus, one of the tokens is an empty string, "".

**public** **static** **void** split\_example2() {

System.***out***.println("\nsplit\_example2()");

String s1 = "43.85 66.239 8.223";

String[] vals = s1.split("\\s");

*printArray*(s1, "\\\\s", vals);

}

// Splits on one or more spaces

**public** **static** **void** split\_example3() {

System.***out***.println("\nsplit\_example3()");

String s1 = "43.85 66.239 8.223";

String[] vals = s1.split("[\\s]+");

*printArray*(s1, "[\\\\s]+", vals);

}

// Splits on or more commas or spaces

**public** **static** **void** split\_example4() {

System.***out***.println("\nsplit\_example4()");

String s1 = "4,3 5, ,,2, 8";

String[] vals = s1.split("[,\\s]+");

*printArray*(s1, "[,\\\\s]+", vals);

}

// Splits on common punctuation

**public** **static** **void** split\_example5() {

System.***out***.println("\nsplit\_example5()");

String s1 = "This. Is,,, \nfunny; \t\tyes: why? now!";

String[] vals = s1.split("[.,;:?!\\s]+");

*printArray*(s1, "[.,;:?!\\\\s]+", vals);

}

**public** **static** **void** printArray(String str, String delimeter, String[] vals) {

String msg = "\"" + str + "\".split(\"" + delimeter + "\")=\n";

**int** i=1;

**for**(String s : vals) {

msg += (i++) + ". \"" + s + "\"\n";

}

System.***out***.println(msg);

}

}

1. Run, examine each method and the corresponding output.
2. **Parsing Data**

In this stage we show an example of *parsing* data.

1. (**Read, no action required)** –
2. Suppose we have a file that contains integers and words in any arbitrary order as shown in the figure below:

E:\Data-Classes\CS 1302 - Programming 2-Fall 2016\notes\06_ch12_Exceptions\c6.jpg

We would like to read the entire file and put all the integers in one *ArrayList* and all the words in another.

1. One way to do this is to read each *token*:

String token = input.next();

And then use a method we can write to see if the token is an int.

**public** **static** **boolean** isAnInteger(String strNum) {

**if** (strNum==**null**) {

**return** **false**;

}

**try** {

**int** val = Integer.*parseInt*(strNum);

**return** **true**;

}

**catch** (NumberFormatException nfe) {

**return** **false**;

}

}

Note that this method would classify a double as a string. If we wanted to allow doubles, we could simply change to *Double.parseDouble.*

1. Thus, the body of the read loop will look like this (where *ints* and *words* are the *ArrayLists* used to store the data):

**while**(input.hasNext()) {

String token = input.next();

**if**(*isAnInteger*(token)) {

**int** x = Integer.*parseInt*(token);

ints.add(x);

}

**else** {

words.add(token);

}

}

1. **Create Classes to Read and Store Data** – Do the following:
2. Create a new package: *read\_examples3*.
3. Create a text file (see directions above if necessary) named: *numbersAndWords.txt*.
4. Copy this data into the file:

33 992 guitar 8 fire 29 344 5 hat pick rain 1 horse 18

1. Create a class named *ReadNumbersAndWords* (in the *read\_examples* package)and replace everything in the class (exceptthe package statement at the top) with:

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** ReadNumbersAndWords {

**public** **static** **void** main(String[] args) {

ArrayList<Integer> ints = **new** ArrayList<>();

ArrayList<String> words = **new** ArrayList<>();

File inFile = **new** File( "src/read\_examples3/numbersAndWords.txt" );

**try** {

Scanner input = **new** Scanner( inFile );

**while**( input.hasNext() ) {

String token = input.next();

**if**(*isAnInteger*(token)) {

**int** x = Integer.*parseInt*(token);

ints.add(x);

}

**else** {

words.add(token);

}

}

input.close();

}

**catch**( IOException e ) {

System.***out***.println(e);

}

System.***out***.println(" ints: " + ints);

System.***out***.println("words: " + words);

}

**public** **static** **boolean** isAnInteger(String strNum) {

**if** (strNum==**null**) {

**return** **false**;

}

**try** {

**int** val = Integer.*parseInt*(strNum);

**return** **true**;

}

**catch** (NumberFormatException nfe) {

**return** **false**;

}

}

}

This approach is not the best, throwing an exception is expensive. However, it illustrates the idea of parsing. And, next we introduce a more efficient way.

1. Run and verify the output:

ints: [33, 992, 8, 29, 344, 5, 1, 18]

words: [guitar, fire, hat, pick, rain, horse]

1. **(Read, no action required)** – Another way to test each token to see if it is an integer is to use the *matches* method of the *String* class which accepts a regular expression. The regular expression below checks to see if all the characters in the token are digits.

**if**( token.matches("\\d+") ) {

Note: this will only detect positive integers; negative integers will be treated as strings. To match negative integers, you would use: ("-?\\d+"). The “?” means optional, thus, the “-“ is optional. There is a lot to understanding regular expressions. We will only consider the one above.

I will not be testing you on the *matches* method, but we will consider it in the next example.

1. Open *ReadNumbersAndWords* and comment out the *if* statement and replace with:

**if**(token.matches("\\d+")) {

The while loop will look like this:

**while**( input.hasNext() ) {

String token = input.next();

// if(isAnInteger(token)) {

**if**(token.matches("\\d+")) {

**int** x = Integer.*parseInt*(token);

ints.add(x);

}

**else** {

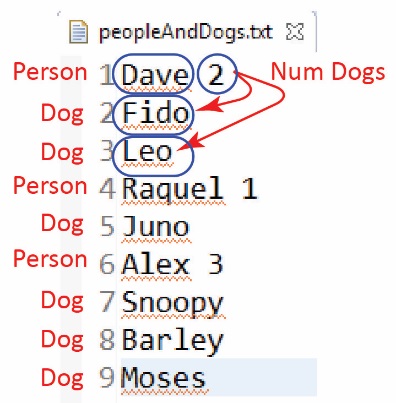
words.add(token);

}

}

1. Run and verify that the output is the same as before.
2. **More Parsing**

In this stage we show another example of parsing data.

1. (**Read, no action required)** –
2. Suppose we have a file as shown on the right. A person (Dave) is shown on line 1 and the number of dogs he has (2). The next two lines show the two dogs.
3. Thus, to read in this data, where each person can have a different number of dogs, we could read a person’s name

String name = input.next();

and then read their number of dogs

**int** numDogs = input.nextInt();

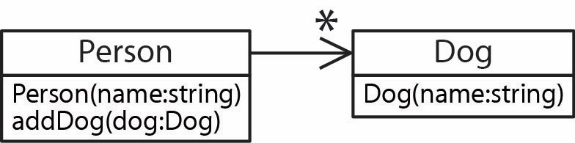
and then loop over the number of dogs reading each dog’s name:

**for**(**int** i=0; i<numDogs; i++){

String dogName = input.next();

...

}

1. Now, suppose we have *Person* and *Dog* classes as shown in the class diagram on the right. Then, we can build an *ArrayList<Person>* with code like this:

ArrayList<Person> people = **new** ArrayList<>();

...

**while**( input.hasNext() ) {

String name = input.next();

Person p = **new** Person(name);

**int** numDogs = input.nextInt();

**for**(**int** i=0; i<numDogs; i++){

String dogName = input.next();

Dog dog = **new** Dog(dogName);

p.addDog(dog);

}

people.add(p);

}

1. **Create Classes to Read and Store Data** – Do the following:
2. Create a text file (see directions above if necessary) named: *peopleAndDogs.txt* in the *read\_examples3* package.
3. Copy this data into the file:

Dave 2

Fido

Leo

Raquel 1

Juno

Alex 3

Snoopy

Barley

Moses

1. Create a class named *Dog* (in the *read\_examples3* package)and replace everything in the class (exceptthe package statement at the top) with:

**public** **class** Dog {

**private** String name;

**public** Dog(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

}

1. Create a class named *Person* (in the *read\_examples3* package)and replace everything in the class (exceptthe package statement at the top) with:

**import** java.util.ArrayList;

**public** **class** Person {

ArrayList<Dog> dogs = **new** ArrayList<>();

**private** String name;

**public** Person(String name){

**this**.name = name;

}

**public** **void** addDog(Dog dog) {

dogs.add(dog);

}

@Override

**public** String toString(){

StringBuilder msg = **new** StringBuilder();

msg.append("Person: " + name + " - Dogs: ");

**for**(Dog dog : dogs)

msg.append(dog.getName() + ", ");

msg.delete(msg.length()-2,msg.length());

**return** msg.toString();

}

}

1. Create a class named *ReadPeopleAndDogs* (in the *read\_examples* package)and replace everything in the class (exceptthe package statement at the top) with:

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** ReadPeopleAndDogs {

**public** **static** **void** main(String[] args) {

ArrayList<Person> people = **new** ArrayList<>();

File inFile = **new** File( "src/read\_examples3/peopleAndDogs.txt" );

**try** {

Scanner input = **new** Scanner( inFile );

**while**( input.hasNext() ) {

String name = input.next();

Person p = **new** Person(name);

**int** numDogs = input.nextInt();

**for**(**int** i=0; i<numDogs; i++){

String dogName = input.next();

Dog dog = **new** Dog(dogName);

p.addDog(dog);

}

people.add(p);

}

input.close();

}

**catch**( IOException e ) {

System.***out***.println(e);

}

**for**(Person p : people) {

System.***out***.println(p);

}

}

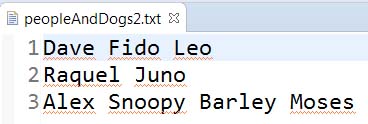
}

1. Run and verify the output:

Person: Dave - Dogs: Fido, Leo

Person: Raquel - Dogs: Juno

Person: Alex - Dogs: Snoopy, Barley, Moses

1. (Read, no action required) Next, let’s consider the same problem except that the text file doesn’t contain the number of dogs, and each person (and their dogs are on the same line) as shown on the right. Here is the approach, inside the loop:
2. Read the entire line:

String line = input.nextLine();

1. Split the line on a space:

String[] tokens = line.split("\\s");

1. The first token has the name:

String name = tokens[0];

Person p = **new** Person(name);

1. The rest of the tokens are dog names. Notice that the loop below starts at 1.

**for**(**int** i=1; i<tokens.length; i++){

String dogName = tokens[i];

Dog dog = **new** Dog(dogName);

p.addDog(dog);

}

people.add(p);

1. Code the example. Do the following:
2. Create a text file named: *peopleAndDogs2.txt* in the *read\_examples3* package.

Dave Fido Leo

Raquel Juno

Alex Snoopy Barley Moses

1. Create a class named *ReadPeopleAndDogs2* (in the *read\_examples* package)and replace everything in the class (exceptthe package statement at the top) with:

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** ReadPeopleAndDogs2 {

**public** **static** **void** main(String[] args) {

ArrayList<Person> people = **new** ArrayList<>();

File inFile = **new** File( "src/read\_examples3/peopleAndDogs2.txt" );

**try** {

Scanner input = **new** Scanner(inFile);

**while**(input.hasNext()) {

String line = input.nextLine();

String[] tokens = line.split("\\s");

String name = tokens[0];

Person p = **new** Person(name);

**for**(**int** i=1; i<tokens.length; i++){

String dogName = tokens[i];

Dog dog = **new** Dog(dogName);

p.addDog(dog);

}

people.add(p);

}

input.close();

}

**catch**( IOException e ) {

System.***out***.println(e);

}

**for**(Person p : people) {

System.***out***.println(p);

}

}

}

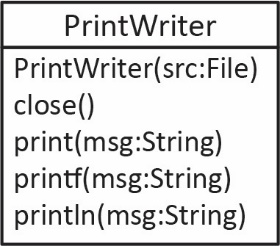
1. Run and verify the output is correct.
2. **Writing Data to a Text File**

In this stage we demonstrate how to write data to a text file using the *PrintWriter* class.

1. (**Read, no action required)** –
2. One way to write data to a text file is to use the [*PrintWriter*](https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html)class. It accepts a *File* as an argument:

File outFile = **new** File("src/write\_examples/numbers.txt");

PrintWriter writer = **new** PrintWriter( outFile );



1. Probably the three most useful (at least for our class) methods of the *PrintWriter* class are *print(…), println(…),* and *printf(…)* which work identically to the *System.out.print(…)* methods. For example, to loop over an array of integers (*nums*) and write them space-delimited to a text file:

**for**( **int** i=0; i<nums.length; i++ ) {

writer.print( nums[i] + " " );

}

1. When we are done writing, we must close the writer: writer.close(); And, as with the *Scanner* class, we must *try/catch* (or *throws…*) all file manipulations.
2. **Create Classes to Read and Store Data** – Do the following:
3. Create a package named *write\_examples.*
4. Create a class named *WriteArray* (in the *write\_examples* package)and replace everything in the class (exceptthe package statement at the top) with:

**import** java.io.File;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**public** **class** WriteArray {

**public** **static** **void** main(String[] args) {

**int**[] nums = {33, 44, 55, 66, 12, 33, 55, 66, 77, 22};

File outFile = **new** File( "src/write\_examples/numbers.txt" );

**try** {

PrintWriter writer = **new** PrintWriter( outFile );

**for**( **int** i=0; i<nums.length; i++ ) {

writer.print( nums[i] + " " );

}

writer.close();

System.***out***.println( "File written" );

}

**catch** (IOException ioe) {

System.***out***.println("Problem creating file or writing");

}

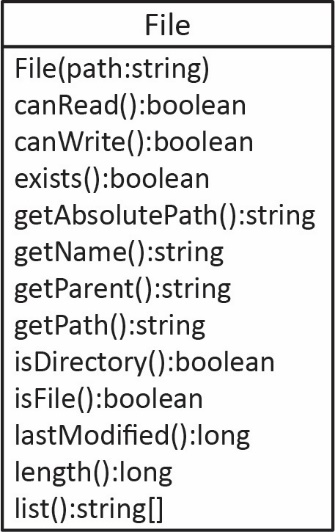
}

}

1. Run and verify the output in the console:

File written

1. Select the *write\_examples* package in the Package Explorer and choose: File, Refresh (or F5) and you should see the output file, *numbers.txt* appear. Open it and verify the contents.
2. Find the array declaration in *WriteArray* and add a number to the beginning of the array.
3. Run the program and examine the contents of *numbers.txt*. Note that the previous file was overwritten completely. In many situations you would want to check before overwriting a file. This is easily handled with the *File* class as we will see later.
4. **The *File* Class**

In this stage we demonstrate a few methods of the *File* class.

1. The [*File*](https://docs.oracle.com/javase/7/docs/api/java/io/File.html)class has many useful methods. The following example will illustrate some of them. Do the following:
2. Create a class named *FileClassTest* (in the *write\_examples* package)and replace everything in the class (exceptthe package statement at the top) with:

**import** java.io.File;

**import** java.io.IOException;

**public** **class** FileClassTest {

**public** **static** **void** main(String[] args) **throws** IOException {

File inFile = **new** File( "src/write\_examples/numbers.txt" );

System.***out***.println(" Does it exist? " + inFile.exists());

System.***out***.println(" Can it be read? " + inFile.canRead());

System.***out***.println("Can it be written? " + inFile.canWrite());

System.***out***.println("Is it a directory? " + inFile.isDirectory());

System.***out***.println(" Is it a file? " + inFile.isFile());

System.***out***.println(" Is it hidden? " + inFile.isHidden());

System.***out***.println(" Number of bytes: " + inFile.length() + " bytes");

System.***out***.println(" Absolute path: " + inFile.getAbsolutePath());

System.***out***.println(" Path: " + inFile.getPath());

System.***out***.println(" Parent: " + inFile.getParent());

System.***out***.println(" Last modified: " + **new** java.util.Date(inFile.lastModified()));

// Derive the absolute path without the file name.

String absPath = inFile.getAbsolutePath();

**int** posEndPath = absPath.lastIndexOf(File.***separator***);

String filePath = absPath.substring(0,posEndPath+1);

System.***out***.println("Abs path (derived): " + filePath);

}

}

1. Run and verify the output. As long as *numbers.txt* is still in your package you will see information about that file.
2. Change the filename to one that doesn’t exist and re-run and verify the output.
3. Change the filename back to the original (*numbers.txt*), re-run and verify that it runs as expected.
4. **Read (no action required)** – A slightly more robust template for reading, where we detect whether the file exists before attempting to read might look like this:

File inFile = **new** File( "src\\write\_examples\\numbers2.txt" );

**if**( inFile.exists()) {

**try** {

Scanner input = **new** Scanner( inFile );

**while**( input.hasNext() ) {

// Read a token...

}

input.close();

}

**catch**( IOException e ) {

System.***out***.println(e);

}

}

**else** {

System.***out***.println("File doesn't exist: " + inFile.getAbsolutePath());

System.***out***.println("Program terminated");

}

To allow the user to supply a new file name would be slightly more involved. A loop would need to be added that continued until a successful file name was supplied or the user indicated that the program should stop. Similarly, before writing, we could detect if the file already exists and ask the user if they want to overwrite or supply a new name.

1. **Reading & Writing Example**

In this stage we illustrate both reading and writing together. The scenario is that we have a file of integers which we want to read and then write the integers that are larger than 99 to another file.

1. Do the following:
2. Create a new text file named *numbers2.txt* and paste these numbers into the file:

8 47 333 555 1 888 9 4 2 222

1. Create a class named *ReadWriteIntegers* (in the *write\_examples* package)and replace everything in the class (exceptthe package statement at the top) with:

**import** java.io.File;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Scanner;

**public** **class** ReadWriteIntegers {

**public** **static** **void** main(String[] args) {

File inFile = **new** File( "src/write\_examples/numbers2.txt" );

File outFile = **new** File( "src/write\_examples/bigNumbers.txt" );

**try** {

Scanner input = **new** Scanner(inFile);

PrintWriter writer = **new** PrintWriter(outFile);

**while**( input.hasNext() ) {

**int** val = input.nextInt();

**if**(val > 99) {

writer.print(val + " ");

}

}

input.close();

writer.close();

System.***out***.println("Program completed successfully");

}

**catch**( IOException e ) {

System.***out***.println(e);

}

}

}

1. Run and verify the console output:

Program completed successfully

1. Select the *write\_examples* package in the Package Explorer and choose: File, Refresh (or F5) and you should see the output file, *bigNumbers.txt* appear. Open it and verify the contents.

**Submission**

1. Do the following:
2. Zip all the folders (packages) under the *src* folder into a zip file named: *lab10\_lastname.zip*
3. Upload your zip file to the *lab10* dropbox in Blazeview.

**You are done!**

1. <https://en.wikipedia.org/wiki/Whitespace_character> [↑](#footnote-ref-1)
2. <https://www.vogella.com/tutorials/JavaRegularExpressions/article.html#common-matching-symbols> [↑](#footnote-ref-2)