**Scanner class**

The Scanner class is a class in java.util, which allows the user to read values of various types. There are far more methods in class Scanner than you will need in this course. We only cover a small useful subset, ones that allow us to read in numeric values from either the keyboard or file without having to convert them from strings and determine if there are more values to be read.

**Class Constructors**

There are two constructors that are particularly useful: one takes an InputStream object as a parameter and the other takes a FileReader object as a parameter.

Scanner in = new Scanner(System.in); // System.in is an InputStream

Scanner inFile = new Scanner(new FileReader("myFile"));

If the file ≥myFile≤ is not found, a FileNotFoundException is thrown. This is a checked exception, so it must be caught or forwarded by putting the phrase ≥throws FileNotFoundException≤ on the header of the method in which the instantiation occurs and the header of any method that calls the method in which the instantiation occurs.

**Numeric and String Methods**

|  |  |
| --- | --- |
| *Method* | *Returns* |
| int nextInt() | Returns the next token as an int. If the next token is not an integer, InputMismatchException is thrown. |
| long nextLong() | Returns the next token as a long. If the next token is not an integer, InputMismatchException is thrown. |
| float nextFloat() | Returns the next token as a float. If the next token is not a float or is out of range, InputMismatchException is thrown. |
| double nextDouble() | Returns the next token as a long. If the next token is not a float or is out of range, InputMismatchException is thrown. |
| String next() | Finds and returns the next complete token from this scanner and returns it as a string; a token is usually ended by whitespace such as a blank or line break. If not token exists, NoSuchElementException is thrown. |
| String nextLine() | Returns the rest of the current line, excluding any line separator at the end. |
| void close() | Closes the scanner. |

The Scanner looks for tokens in the input. A token is a series of characters that ends with what Java calls whitespace. A whitespace character can be a blank, a tab character, a carriage return, or the end of the file. Thus, if we read a line that has a series of numbers separated by blanks, the scanner will take each number as a separate token. Although we have only shown four numeric methods, each numeric data type has a corresponding method that reads values of that type.

The numeric values may all be on one line with blanks between each value or may be on separate lines. Whitespace characters (blanks or carriage returns) act as separators. The next method returns the next input value as a string, regardless of what is keyed. For example, given the following code segment and data

int number = in.nextInt();

float real = in.nextFloat();

long number2 = in.nextLong();

double real2 = in.nextDouble();

String string = in.next();

44 23

2222222222

22222.33 End

44 would be stored in number; 23.0 would be stored in real; 2222222222 would be stored in number2; 22222.33 would be stored in real2; and ≥End≤ would be stored in string. Notice that method nextFloat reads an integer and stores it as a float value. This is legal because an integer value can be stored exactly as a real value; there is no ambiguity. If we had keyed a decimal point after the 44, the system would have thrown a InputMismatchException (a checked exception) regardless of whether or not a non-zero value followed the decimal point. An arbitrary real number cannot be stored exactly as an integer; it is ambiguous. Remember that anything ambiguous is illegal.

nextLine reads the rest of the line and returns it as a string. The carriage return is consumed but is not appended to the string. The numeric reads do not consume the whitespace, so if a nextLine is issued at after a numeric read and the numeric value is at the end of the line, nextLine returns the empty string. nextLine never jumps over a carriage return to get the next line of input. For example, the following code fragment

int number = in.nextInt();

String string = in.nextLine();

float real = in.nextFloat();

String string2 = in.nextLine();

and the data shown above, string would contain ≥23≤ and string2 would contain the empty string.

Here is a program that uses these methods, followed by the output. Look over the application carefully to be sure you understand how the output was generated.

**Class NumericInput demonstrates reading numeric values.**

import java.util.Scanner;

import java.io.\*; // Access System.out

public class NumericInput

{

public static void main(String[] args)

{

// Declarations

Scanner in = new Scanner(System.in);

int integer;

long longInteger;

float realNumber;

double doubleReal;

String string1;

String string2;

// Prompts

System.out.println("Enter an integer, a long integer, " + "a floating-point "); System.out.println("number, another floating-point number, " + "and a string.");

System.out.println("Separate each with a blank or return.");

// Read in values

integer = in.nextInt();

longInteger = in.nextLong();

realNumber = in.nextFloat();

doubleReal = in.nextDouble();

string1 = in.nextLine();

System.out.println("Now enter another value.");

string2 = in.next();

System.out.println("Here is what you entered: ");

System.out.println(integer + " " + longInteger + " " + realNumber + " " + doubleReal + " " + string1 + " and " + string2);

}

}

**Output:**

Enter an integer, a long integer, a floating-point

number, another floating-point number, and a string.

Separate each with a blank or return.

23

24

25.0 233333333333333.444 Hello

Now enter another value.

23.4

Here is what you entered:

23 24 25.0 2.3333333333333344E14 Hello and 23.4

Boolean Methods

We said that the Scanner methods that read numeric data throw a InputMismatchException exception if the next value isnπt what the method expects. We can avoid that problem using Boolean methods. Here are four useful Boolean methods that allow us to check to be sure that the next value is what we expect.

Method

Returns

boolean hasNextLine()

Returns true if the scanner has another line in its input; false otherwise.

boolean hasNextInt()

Returns true if the next token in the scanner can be interpreted as an int value.

boolean hasNextFloat()

Returns true if the next toke in the scanner can be interpreted as a float value.

Let's write a code fragment that instantiates a scanner and reads and prints an integer value and a second integer value if there is one.

Scanner in = new Scanner(System.in);

System.out.println(in.nextInt());

if (in.hasNextInt())

System.out.println(in.nextInt());

There are methods equivalent to these for each of the Java built-in types.

The following application applies the appropriate reading method to the data that is keyed in.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// MixedTypeInput

// This application demonstrates testing before reading to be

// sure to use the correct input method for the data.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.io.\*;

import java.util.Scanner;

public class MixedTypeInput

{

public static void main(String[] args)

{

double number;

Scanner in = new Scanner(System.in);

System.out.println("Enter your gross income: ");

if (in.hasNextInt())

{

number = (double)in.nextInt();

System.out.println("You entered " + number);

}

else if (in.hasNextFloat())

{

number = (double)in.nextFloat();

System.out.println("You entered " + number);

}

else if (in.hasNextDouble())

{

number = in.nextDouble();

System.out.println("You entered " + number);

}

else

System.out.println("Token not an integer or a real value.");

}

}

The application was run four times. The input is shown in red.

Enter your gross income:

55000

You entered 55000.0

Enter your gross income:

55000.0

You entered 55000.0

Enter your gross income:

55E10

You entered 5.50000001024E11

Enter your gross income:

Fifty Five Hundred

Token not an integer or a real value.

What would happen if there were no token in the file in the previous example? Each of the boolean methods would return false. They return true if and only if the next token in the scanner can be interpreted as a value of their type. We return to the subject of reading data from files later in this chapter and show how to use these Scanner methods to allow us to read multiple values from a line in a file. Except for some trivial cases, we must combine reading operations with loops to read through all of the data on a file.

Files

To read from a file rather than the keyboard, you instantiate a Scanner object with a FileReader object rather than System.in.

Scanner in = new Scanner(System.in); // Reading from the keyboard

Scanner inFile = new Scanner(new FileReader(≥inFile.dat≤)); // Reading from a file

Although all of the methods applied to keyboard input can be applied to file input, there are methods that are usually applied only to files. These are the methods that ask of there are more values in the file. If there are no more values in a file, we say that the file is at the end of the file (EOF). For example,

inFile.hasNext();

inFile.hasNextLine();

return true if inFile has another token in the file or if there is another line in the file. What about the methods hasNextInt and so forth that we used to look ahead at the type of the next input token? These can be used to determine if there are more data values in the file, provided you know exactly how the files are organized

Be sure to close all files. If you forget to close System.in, no harm is done, but forgetting to close a file can cause problems.