The Java API and User Input

For a program to be interactive, it needs to have a way to get input from the user. There are many types of user input; mouse input and keyboard input are probably the ones you use most frequently. What other types of input can you think of? Hint: don't limit yourself to thinking about how you interact with a desktop computer. Consider all of the peripheral devices that might be connected to the computer, or input devices that are in your smartphone, tablet, or video game controller.

Since a mouse and keyboard are used for most input on a traditional desktop computer, we will focus on how to write Java programs that use them. There are many different classes in the Java API that support input to the program; we won't cover all of them, but you are encouraged to familiarize yourself with the Java API and learn about some of the other classes that exist. We'll start by learning how to get user input from the keyboard; later in the class we'll also talk about getting user input from the mouse.

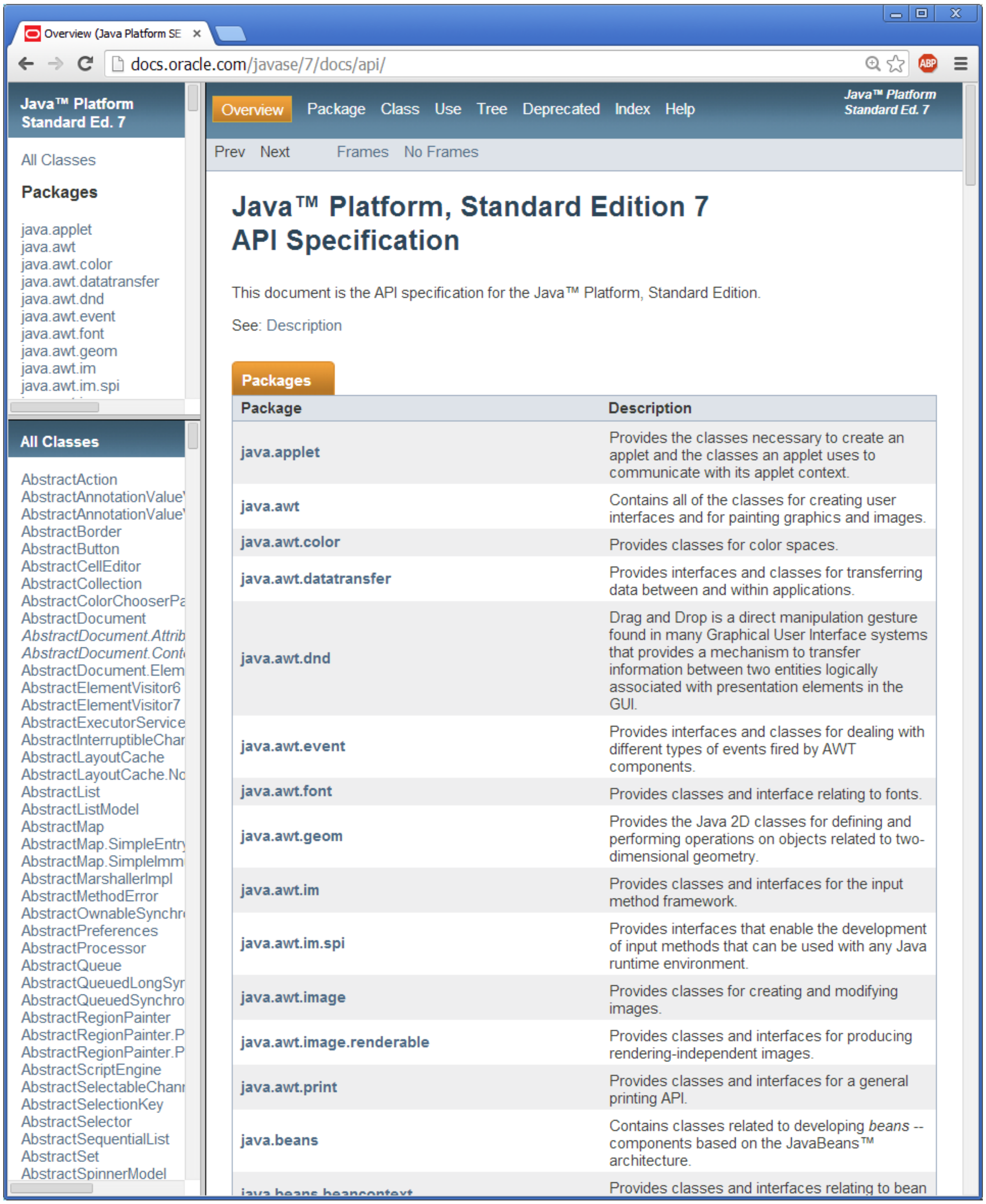
Using the Java API

The Java API is the specification of all of the classes and methods that are part of the Java language. There are a lot of classes that are used in different Java programs – it would be a challenge to keep track of them all without some way to organize them! Think about all of the Java programs you have written so far for this class – if you kept all of the .java files in one folder on your computer, you might find it difficult to locate the one that you want. So you make a folder for each assignment, and put the files relevant to a particular assignment in the correct folder. Java has something similar for all of its classes, but instead of putting classes in folders, it puts them in packages.[1]

Open up a browser and look at the Java API.[2] It should like similar to the image below:

[1]We won't talk much about packages in this course, but you can read more about them here: http://docs.oracle.com/javase/tutorial/java/package/packages.html

[2] The Java API is currently found here: http://docs.oracle.com/javase/7/docs/api/ You can always do a search for "Java API" and look for the most recent version of Java. If you want to see how the Java language has changed with each new version, or if you want to make sure your code can run on a computer with an old version of Java, you can even look at the old APIs!


Overview (Java Platform SE
docs.oracle.com/javase/7/docs/api/

Java™ Platform Standard Ed. 7

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi

All Classes

AbstractAction
AbstractAnnotationValue
AbstractAnnotationValue
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPa
AbstractDocument
AbstractDocument.Attrib
AbstractDocument.Cont
AbstractDocument.Elem
AbstractElementVisitor6
AbstractElementVisitor7
AbstractExecutorService
AbstractInterruptibleChar
AbstractLayoutCache
AbstractLayoutCache.No
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImm
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchr
AbstractPreferences
AbstractProcessor
AbstractQueue
AbstractQueuedLongSyr
AbstractQueuedSynchro
AbstractRegionPainter
AbstractRegionPainter.P
AbstractRegionPainter.P
AbstractScriptEngine
AbstractSelectableChanr
AbstractSelectionKey
AbstractSelector
AbstractSequentialList
AbstractSet
AbstractSpinnerModel

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

Prev Next Frames No Frames

Java™ Platform, Standard Edition 7
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

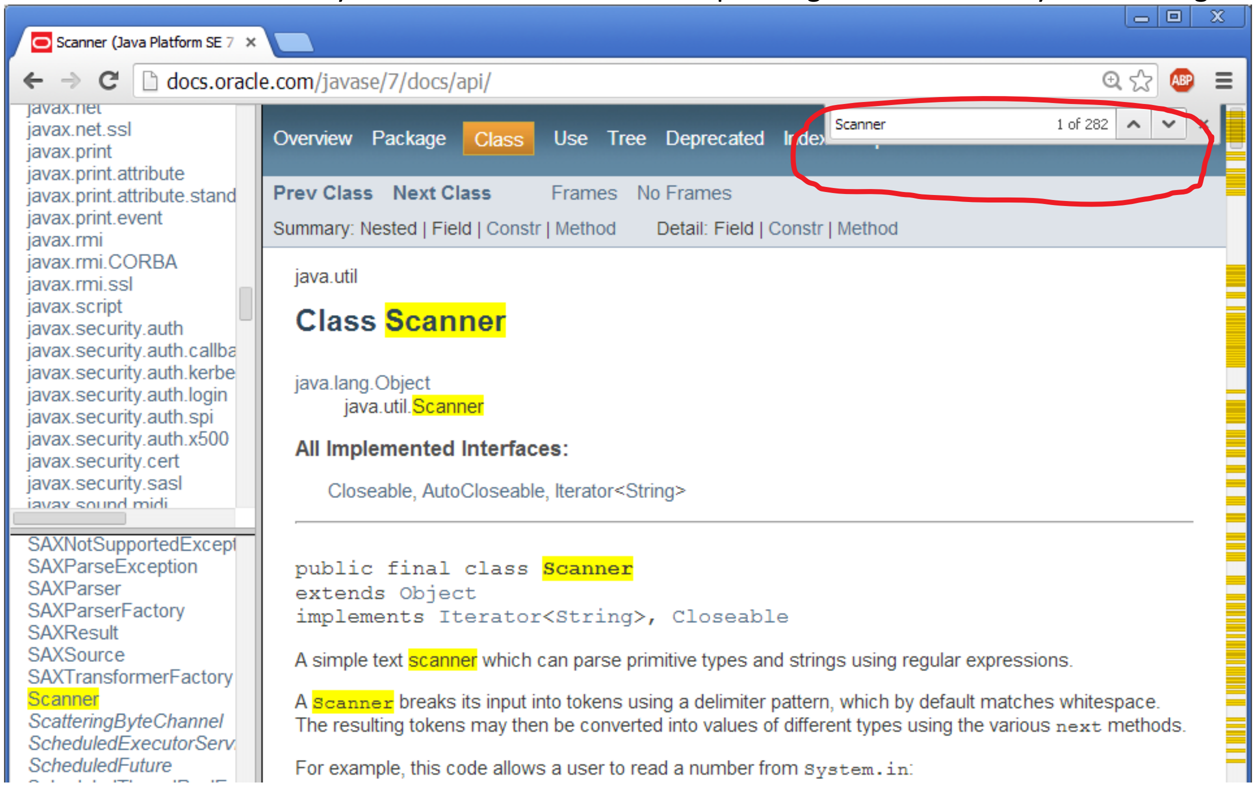
Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean

The version in the picture is using frames.[3] On the left side, there are two ways to navigate the classes in the API; the upper left frame shows all of the Java packages, and the lower left frame

[3] If you don't see frames on your browser, click on the word "Frames" that is near the top of the page.

shows all of the Java classes. If you click on a Java package in the upper left frame, the lower left frame updates to show only the information about classes[4] in that package. Try clicking on java.util in the upper left frame. Then, find the Scanner class in the lower left frame and click on it. Notice that the frame on the right changes now, to tell you about the Scanner class. We'll be using the Scanner class to get keyboard input from the user.

What if you didn't know what package Scanner was in, but you wanted to know more about the class? You can use Ctrl+F to bring up a search box, and search for the word "Scanner" when all of the classes are showing. Let's try that now. First we need to display all of the classes again though. In the upper left frame, click on the words "All Classes". Then, hold down the Ctrl key and press the key for the letter F (you should not press the Shift key – just the F key). Do you see the search box? It may be in a different location depending on the browser you are using.


Scanner (Java Platform SE 7
docs.oracle.com/javase/7/docs/api/
Scanner
1 of 282
Overview
Package
Class
Use
Tree
Deprecated
Index
Prev Class
Next Class
Frames
No Frames
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method

javax.net
javax.net.ssl
javax.print
javax.print.attribute
javax.print.attribute.stand
javax.print.event
javax.rmi
javax.rmi.CORBA
javax.rmi.ssl
javax.script
javax.security.auth
javax.security.auth.callba
javax.security.auth.kerbe
javax.security.auth.login
javax.security.auth.spi
javax.security.auth.x500
javax.security.cert
javax.security.sasl
javax.sound.midi

SAXNotSupportedExcept
SAXParseException
SAXParser
SAXParserFactory
SAXResult
SAXSource
SAXTransformerFactory
Scanner
ScatteringByteChannel
ScheduledExecutorServ
ScheduledFuture

java.util

Class Scanner

java.lang.Object
    java.util.Scanner

All Implemented Interfaces:

Closeable, AutoCloseable, Iterator<String>

```
public final class Scanner
extends Object
implements Iterator<String>, Closeable
```

A simple text scanner which can parse primitive types and strings using regular expressions.

A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods.

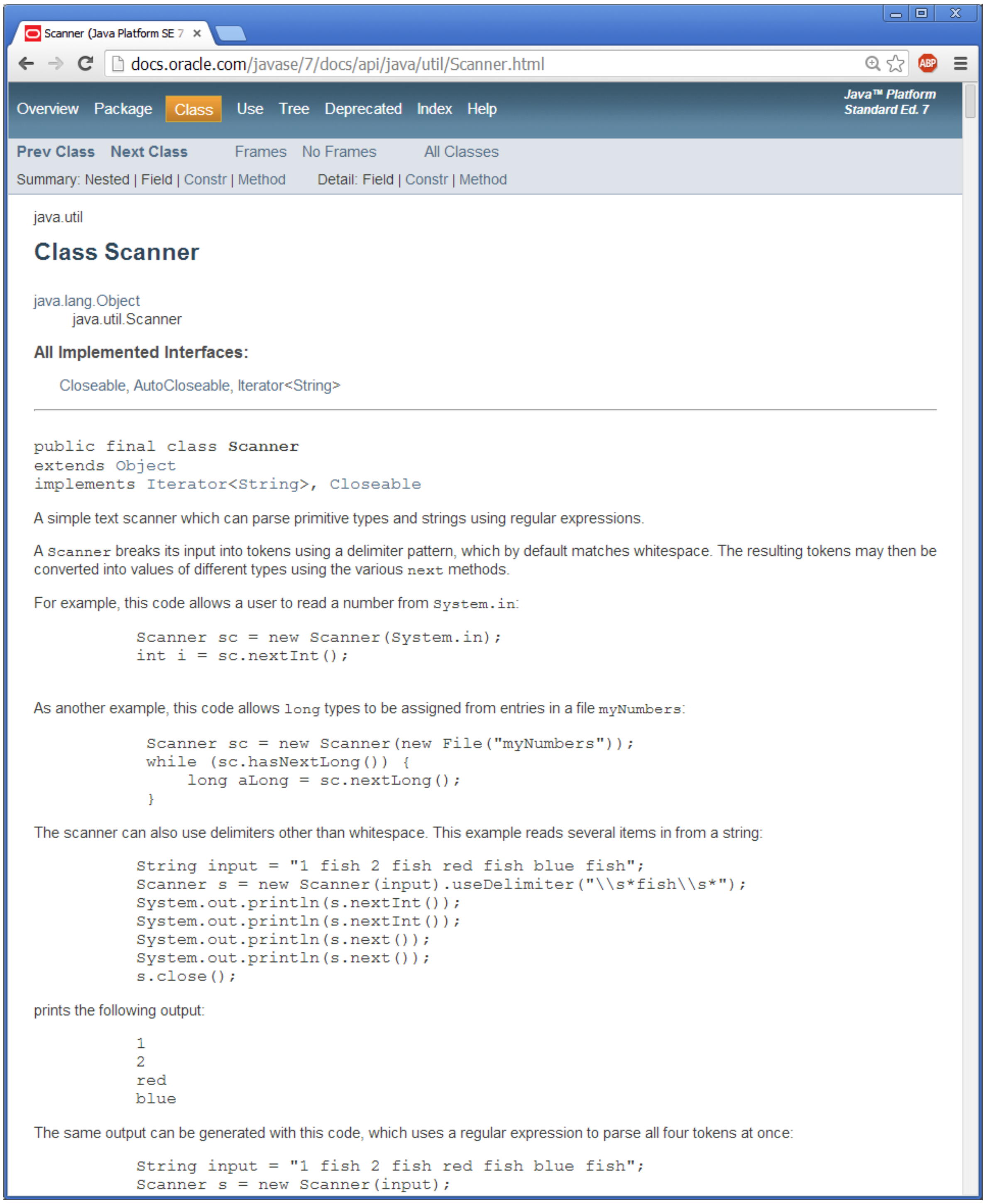
For example, this code allows a user to read a number from `System.in`:

Type the word "Scanner" in the search box. Wow, I see 282 matches in my browser[5]! What I really want is to find the one in the lower left frame. Scroll down to find Scanner in the list (note the classes are in alphabetic order). Now you can click on the word Scanner to see the documentation in the frame on the right.

There is a lot of information in the documentation. We'll mention some of the parts of the documentation now, but we'll talk more about using the API as we learn more about Java. In

[4] You'll note that there are more categories than just "Classes" that are displayed. We'll learn about things like interfaces, enums, and exceptions later in the course.
[5] There are so many matches right now because the documentation for the Scanner class is in the frame on the right.

case you are reading this without following along in your browser[6], here is a screenshot without frames:


Scanner (Java Platform SE 7
docs.oracle.com/javase/7/docs/api/java/util/Scanner.html
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Scanner

java.lang.Object
    java.util.Scanner

All Implemented Interfaces:

Closeable, AutoCloseable, Iterator<String>

```
public final class Scanner
extends Object
implements Iterator<String>, Closeable
```

A simple text scanner which can parse primitive types and strings using regular expressions.

A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods.

For example, this code allows a user to read a number from `System.in`:

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

As another example, this code allows `long` types to be assigned from entries in a file `myNumbers`:

```
Scanner sc = new Scanner(new File("myNumbers"));
while (sc.hasNextLong()) {
    long aLong = sc.nextLong();
}
```

The scanner can also use delimiters other than whitespace. This example reads several items in from a string:

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter("\\s*fish\\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();
```

prints the following output:

```
1
2
red
blue
```

The same output can be generated with this code, which uses a regular expression to parse all four tokens at once:

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input);
```

[6] Just as you should type code and try it out while reading, you should also do each of the steps described in this handout so you will learn how to use the Java API. You'll be using the API a lot for your final project!

The Scanner Class Documentation
Let's look at the documentation for the Scanner class together. At the top of the documentation page, notice the words java.util. This is the package that the Scanner class is in. If we want to use the Scanner class in our code, we need to put the following line in our .java file, before the line that starts `public class …` :

```
    import java.util.Scanner;
```

This line tells Java where to find information about the Scanner class. Any time you want to use a class that is not in the local directory or in the classpath (like the bookClasses folder is), you need to have an import line like this. The import line above imports only the Scanner class. We could instead use a wildcard character * to import all classes in the java.util package:

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

If there is a Java class you want to use in your program, it's helpful to know what package it is in. Now you know how to find it in the Java API!

The next line of the documentation says "Class Scanner" in a big font. This is the class we're reading about. Below this are two lines that say:

```
java.lang.Object
   java.util.Scanner
```

Notice that the second line is indented. These two lines tell us the inheritance hierarchy of the class. Don't worry about this right now – we'll talk about inheritance later in the course. For now you can also ignore everything else before the horizontal line that spans the page.

The first thing below the horizontal line is the class header. We'll talk about this more later too. The text after the class header is a summary of the class. This line is equivalent to what you should put in your comments at the top of each file – it explains the purpose of this class. The first line of the summary says "A simple text scanner which can parse primitive types and strings using regular expressions." Depending how much computer science background you have, this description may not make much sense since you might not know what some of the words mean. If you are looking through the API on your own, you may find it helpful to look up the definition of these words. The word "parse" means that we are going to separate different pieces. Regular expressions are a way to tell the computer what we're looking for in text input. Overall, this description means that we can use the Scanner class to separate text into primitive types (e.g. numbers) and Strings (e.g. words). This looks like a great class to use for getting user input from the keyboard and storing what the user types into the correct type variables!

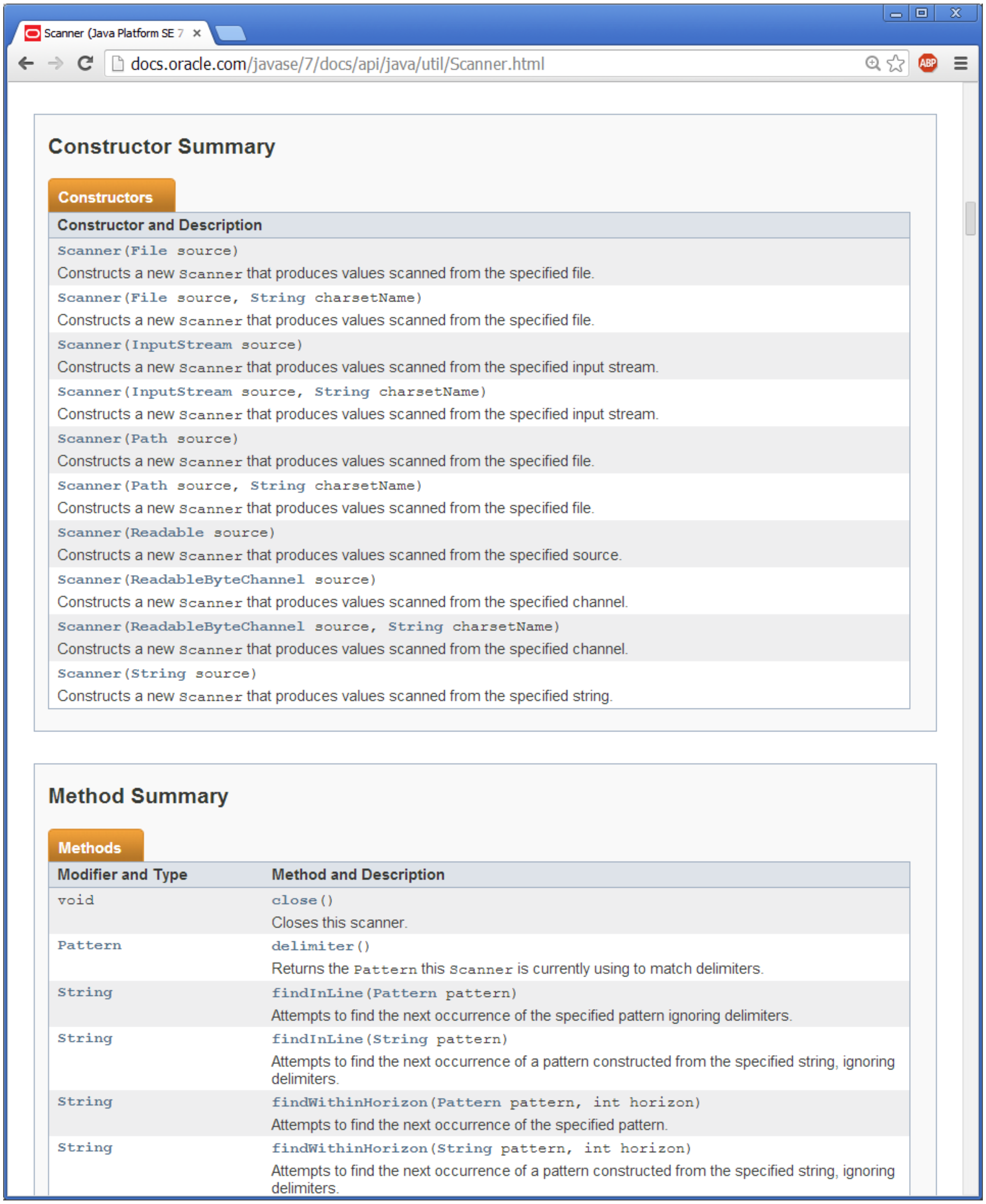
The summary continues for quite a while, with more details and some examples. You should skim the rest of the summary now, then read it more carefully after you have finished reading this handout.[7] Depending on what class you are looking at, there may be some information at the end of the summary with labels like “Since:” or “See Also:”

Did you notice the first example in the summary? It gives example code for reading a number from System.in, which you can think of as being the keyboard:

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

If you scroll down to the end of the summary, you’ll see some shaded boxes. The first shaded box in the Scanner documentation is labeled “Constructor Summary”. This lists all of the constructors that can be used to create a Scanner object. We’ll talk more about these later too, but notice that depending which constructor we use, we can use the Scanner object to read data from different sources, including files, input streams, and Strings. We want to use the constructor that takes an InputStream as a parameter, as shown in the code above (System.in is an already defined InputStream object).

[7] Caution – you will probably not understand all of the summary even after reading this handout. That’s okay. You won’t be tested on it, but you should get used to reading these summaries because you will be reading them while working on the final project.



Scanner (Java Platform SE 7

docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Constructor Summary

Constructors

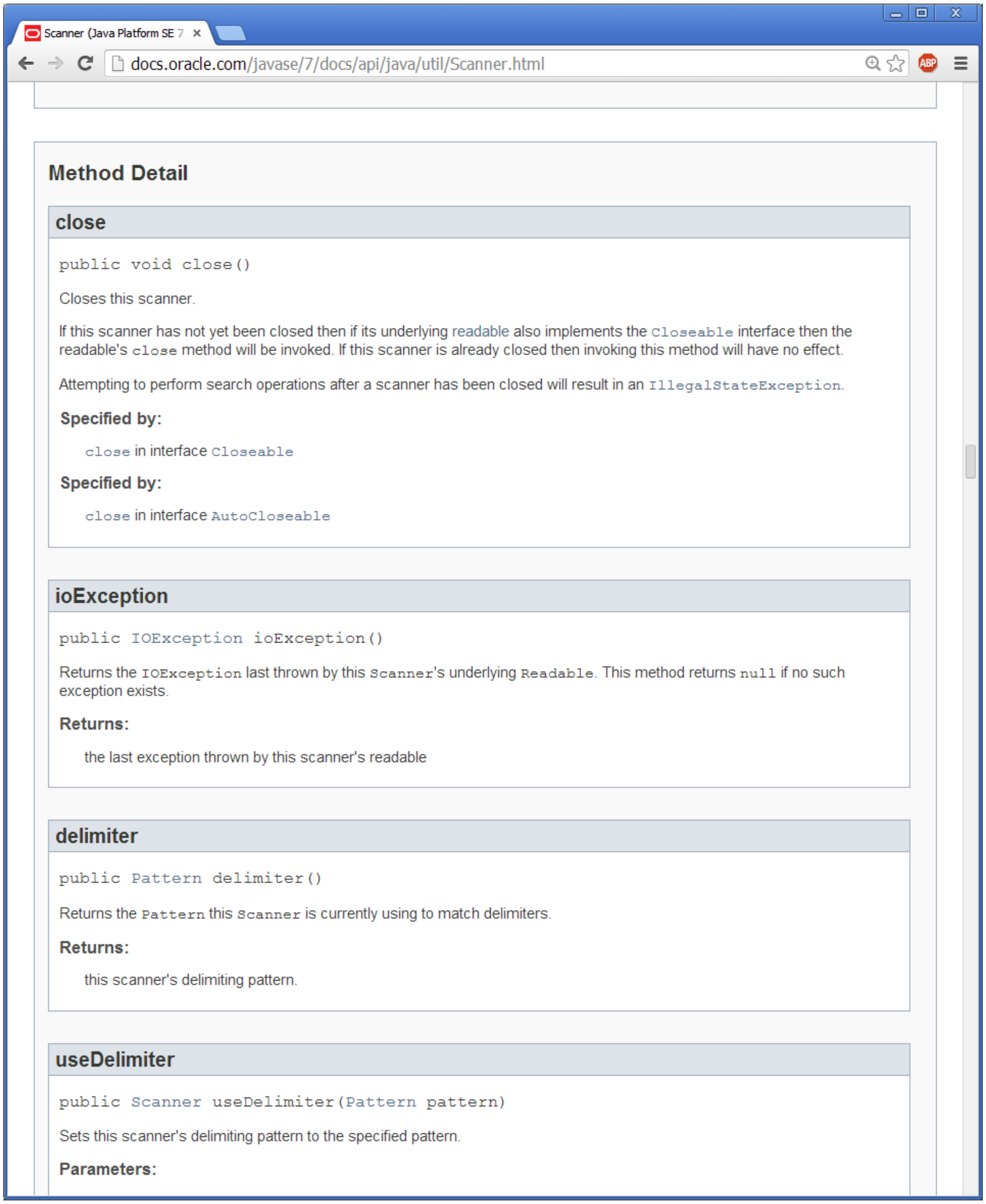
Constructor and Description
`Scanner(File source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(File source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(InputStream source)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(InputStream source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(Path source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Path source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Readable source)` Constructs a new `Scanner` that produces values scanned from the specified source.
`Scanner(ReadableByteChannel source)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(ReadableByteChannel source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(String source)` Constructs a new `Scanner` that produces values scanned from the specified string.

Method Summary

Methods

Modifier and Type	Method and Description
`void`	`close()` Closes this scanner.
`Pattern`	`delimiter()` Returns the `Pattern` this `Scanner` is currently using to match delimiters.
`String`	`findInLine(Pattern pattern)` Attempts to find the next occurrence of the specified pattern ignoring delimiters.
`String`	`findInLine(String pattern)` Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
`String`	`findWithinHorizon(Pattern pattern, int horizon)` Attempts to find the next occurrence of the specified pattern.
`String`	`findWithinHorizon(String pattern, int horizon)` Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.

Below the constructor summary is the method summary box. This lists all of the methods of the Scanner class. The left column indicates the return type; the right column has the name of the method and a short description. Note that you can click on the name of a method to go to the full, detailed description. Different classes will have different labels for shaded boxes, but the constructor summary and the method summary are common to see. The summary boxes are typically followed by the detail boxes. Keep scrolling down in the Scanner documentation, and you'll see the constructor detail box, and below that you'll find the method detail box.


Scanner (Java Platform SE 7
docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Method Detail

close

`public void close()`

Closes this scanner.

If this scanner has not yet been closed then if its underlying readable also implements the `Closeable` interface then the readable's `close` method will be invoked. If this scanner is already closed then invoking this method will have no effect.

Attempting to perform search operations after a scanner has been closed will result in an `IllegalStateException`.

Specified by:

`close` in interface `Closeable`

Specified by:

`close` in interface `AutoCloseable`

ioException

`public IOException ioException()`

Returns the `IOException` last thrown by this `Scanner`'s underlying `Readable`. This method returns `null` if no such exception exists.

Returns:

the last exception thrown by this scanner's readable

delimiter

`public Pattern delimiter()`

Returns the `Pattern` this `Scanner` is currently using to match delimiters.

Returns:

this scanner's delimiting pattern.

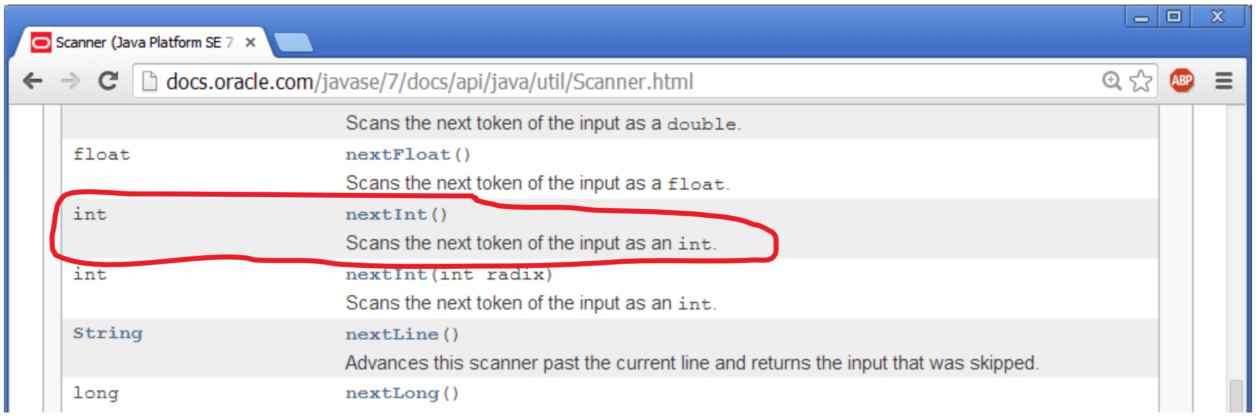
useDelimiter

`public Scanner useDelimiter(Pattern pattern)`

Sets this scanner's delimiting pattern to the specified pattern.

Parameters:

In the method summary box, find the entry for nextInt(), which was used in the example code in the class summary. You may find it helpful to search with Ctrl+F.


Scanner (Java Platform SE 7
docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

	Scans the next token of the input as a `double`.
`float`	`nextFloat()` Scans the next token of the input as a `float`.
`int`	`nextInt()` Scans the next token of the input as an `int`.
`int`	`nextInt(int radix)` Scans the next token of the input as an `int`.
`String`	`nextLine()` Advances this scanner past the current line and returns the input that was skipped.
`long`	`nextLong()`

Note that there are two methods that are both called `nextInt`. Your textbook will talk about why this is okay. For now, select the one that does not have a parameter (it's circled in the screenshot above). Click on the word `nextInt()`. This takes you to the detailed description of the method. The detailed description shows the method header and repeats the summary description, then gives a lot more details. Any expected parameters, any return value, and any exceptions that may occur are described here.[8]

This documentation is pretty handy, and it's nice that it is uniform from class to class. How was this documentation created? Do you think that somebody went through each class and typed the HTML for each web page by hand? Of course not! Remember that computer scientists are lazy when the computer can do something for them instead! All of this documentation was generated automatically, using Javadoc. Chapter 4 of your textbook discussed comments, including the Javadoc style of comment. More information about writing Javadoc style comments for the Javadoc Tool are found here: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

You are not required to write Javadoc style comments for this course, but you should read the instructions and try to use this style – it will make your code look much more impressive to potential employers when they see that you already know how to comment your code properly! You can generate Javadoc documentation for your correctly commented code in Dr. Java, using the Tools->Javadoc menu. In fact, documentation for all of the classes in the bookClasses folder was created this way. You can find the documentation by going to the bookClasses folder, then clicking on the doc folder. Double click the package-summary.html file to open it in your browser, then click on "FRAMES" near the top of the page to see the list of all classes on the left. Note that the classes in the bookClasses folder do not belong to any particular package. This is generally not a good programming decision for real-world programming, but works well for beginning programming courses.

[8] We'll talk about exceptions later in the course, but it's good to know you can find this information in the API.

Using the Scanner class to get keyboard input:
Now that we've explored the Scanner API, let's look at a program that uses the Scanner class to read user input. Here is a sample program that reads a number the user types, and outputs it to the screen.

```
/**
 * @author Tasha
 * This is an example program that accepts an integer from the user,
 * then repeats it.
 */

// Import the package needed to use the Scanner class
import java.util.Scanner;

public class Repeat
{
  public static void main(String[] args)
  {
    // Declare and initialize variables
    // Create a scanner object that is connected to the keyboard.
    Scanner input = new Scanner(System.in);
    int x;

    // Prompt the user for input
    System.out.print("Enter an integer: ");
    // Read the value that the user provided and store it in x
    x = input.nextInt();

    // Output the user's integer
    System.out.println("You input the number " + x);
  }
}
```

Let's do a walkthrough of the code. The first thing in the code is a comment (in fact, a Javadoc style of comment) indicating the author and the purpose of the program. The line

```
      import java.util.Scanner;
```

tells Java where to find the class definition for Scanner. The program won't compile without this line.

The class header is in the next lines:

```
      public class Repeat
      {
```

and then comes the method header for main():

```
      public static void main(String[] args)
```

The body of main() is where the most interesting stuff happens. The first thing that happens in main() is the declaration and initialization of a few variables. The line

```
      Scanner input = new Scanner(System.in);
```
creates a Scanner object that is connected to System.in – that is, it is connected to the keyboard. We also create space in memory to store an int, and label it x.

Next, we prompt the user for input with the line:
```
      System.out.print("Enter an integer: ");
```
Good program design for interacting with the user always ensures that the user knows what is expected. If we left out this line, the program would sit around waiting for user input, but the user wouldn't know why it appeared that the program was hanging.[9]

Finally we get to the line where we actually get information from the user:
```
      x = input.nextInt();
```
This line causes the program to wait until the user types something and presses the enter key. When the user presses the enter key, the characters that they typed are converted to type int and stored in the variable x.

The last thing this program does is print the number the user typed to the screen:
```
System.out.println("You input the number " + x);
```

Try running this program – make sure you type an integer and press enter.[10] It's not a very exciting program, is it? Can you figure out how to make it more exciting by using a loop to print the number out as many times as the number is large?

Let's try a different type of input – we'll ask the user for their name, then welcome them personally to the class.

```
/**
 * @author Tasha
 * This program asks the user for a name, then greets the user
 * by name.
 */

// Import the package needed to use the Scanner class
import java.util.Scanner;

public class Greeting
{
 public static void main(String[] args)
 {
```

[9] Try this out yourself. After you've run the correct version of the program, comment out the prompt to the user. What happens? Can you tell the program is running? Note that the box that Dr. Java draws in the Interactions pane when it is waiting for keyboard input is unique to Dr. Java. When running this program anywhere else, nothing will appear! Go ahead and type an integer in the box and press enter to see what happens. The rest of the program run should look like the run of the correct version of the program.

[10] What happens if you type a non-integer? Try a decimal number, and also try typing non-numeric characters. What if you type an integer followed by a space followed by a non-numeric character? Why do you think the program behaves this way? You may find it helpful to refer back to the Scanner documentation in the Java API.

```
  // Declare and initialize variables
  // Create a scanner object that is connected to the keyboard.
  Scanner input = new Scanner(System.in);
  String name;

  // Prompt the user for input.
  System.out.print("What is your name? ");

  // Read the output. You can use the next() or the nextLine()
  // methods from the Scanner class. How do they behave
  // differently? To test each line, uncomment one, and comment
  // out the other.
  // Remember that you can use Ctrl+/ to comment, and Ctrl+Shift+/
  // to uncomment lines.
  name = input.next();
//    name = input.nextLine();
  // Greet the user by name.
  System.out.printf("Welcome to class, %s!\n", name);
 }
}
```

This program is fairly similar to the previous program. It still has the import statement and it still creates a Scanner object that is connected to the keyboard input. This time we are creating a String reference variable called `name` instead of an int so we can store the user input. To read a String that the user typed, we can use:

```
      name = input.next();
```

or we can use

```
      name = input.nextLine();
```

There is a slight difference in the behavior between the next() method and the nextLine() method. Can you figure it out? Try typing just a first name when running each version program, and also try typing a first name and a last name separated by a space. You may find it helpful to refer to the Java API again.

Test your understanding:

How can you find the package that a Java class is in?
How do you use the Java API to find out what a class does?
How do you use the Java API to find the parameters a method requires, and the value that a method returns (if any)?
How does Java know where to find information about the Scanner class?
How do you create a Scanner object that gets input from the keyboard?
What method do you call to read an int that the user typed? A double? A String?