

# Libraries and Documentation

## Purpose:

The purpose of this lab is to acquaint you with the standard Java library and with tools for the development and presentation of library class documentation.

## The standard Java library:

The standard Java library consists of collections of classes available for incorporation into your own programs. As such, classes you define are “clients” of these library classes. The class specifications are presented as HTML documents that can be read and navigated with a web browser. To use the library, you must first become familiar with navigating the library documentation.

## Browsing the Java library:

- Open an internet browser. If you are already reading this document with a browser, open another browser window. (You can do this in Netscape by choosing the *Navigator* option from the *Communicator* pull-down that appears on the menu bar at the top of the Netscape window.) Load the document <http://java.sun.com/j2se/1.3/docs/api/>. This link is to the specification of the standard Java library version 1.3 as released by Sun Microsystems, the developers of Java. (“API” means “Application Program Interface.”)

You should have in front of you a split window with 3 sub-windows, called “frames.” The top left frame lists the packages in the library. Scrolling through it you can see that the library has a substantial number of packages. Clicking a package name in this window causes the list of classes in this package to be displayed in lower left frame. Clicking *All Classes* causes all classes in the library to be listed in the lower left frame.

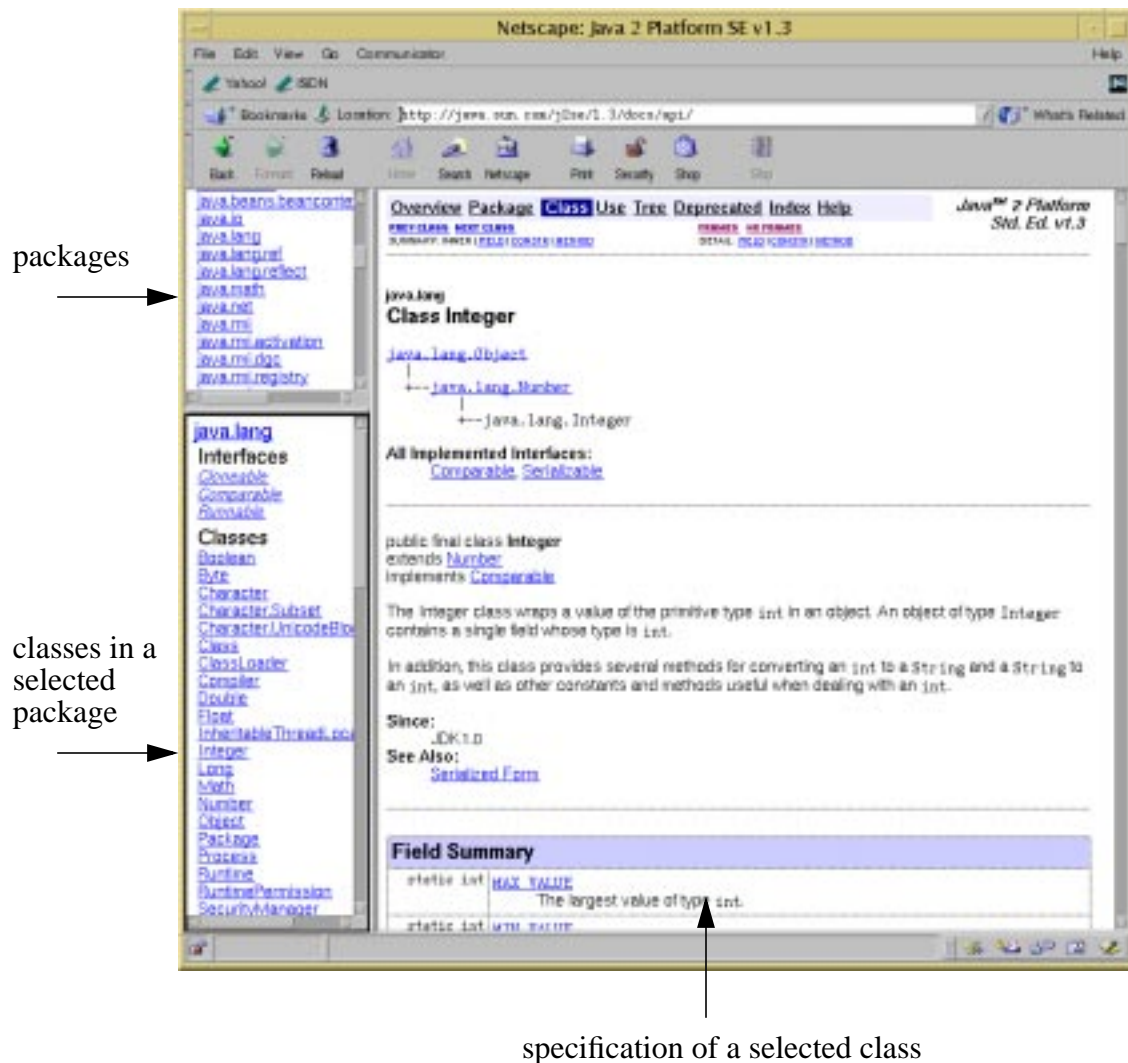
- Select several packages in the top left frame and note how the list in the lower left window changes. Find two packages that contain only one or two classes.

Notice that the first line in the lower frame displays the name of the package. The lower listing can contain three categories of items: *interfaces*, *classes* and *exceptions*. We will limit our attention to *classes*.

## Choosing a class for browsing:

- To look at the specification for a class, we choose the package containing the class in the top left frame and then choose the class in the lower left frame. The specification of the class will appear in the main frame of the display.
- Browse several classes and note the format of the documentation. Methods are described in “summary” form and in “detailed” form. Note that only the specification of a method is given. There is no “implementation.” The description in the “summary” is the first sentence from the doc comment preceding the method definition. The description in the “detail” is the entire doc comment. Sufficient information must be provided so that a programmer can use the method

and understand what it will do in all circumstances.



### Browsing the *Dimension* class

- Choose the package `java.awt` in the top left frame.
- Browse the class `Dimension`. Some of the information we must ignore, since we haven't yet discussed all the relevant topics. Don't expect to understand everything in the documentation yet! The actual specification of the class starts with the lines:

```
public class Dimension
    extends Dimension2D
    implements Serializable
```

The specification begins with an explanation of the class, followed by links to related classes.

- Next are three tables labeled *Field Summary*, *Constructor Summary*, and *Method Summary*. (We can ignore *Field Summary* for now.) Clicking a method name brings you to a more

detailed explanation of the method. Spend sometime reading and understanding how to navigate the window containing the specification of the *Dimension* class.

*Understanding the specifications:*

- How many constructors are available for the class *Dimension*?
- How many *Dimension* constructors have one parameter only?
- How many *Dimension* constructors have more than one parameter?
- What is the width of the *Dimension* constructed in the following statement?

```
Dimension d = new Dimension();
```

- Which of the methods in the class *Dimension* are commands?
- Which method would you use to determine the width of a *Dimension* instance? Write the specification of this method.
- Which method would you use to change the width and height of a *Dimension* instance? Write the specification of this method.

**Browsing the *String* class:**

- Choose the package *java.lang* in the top left frame.
- How many classes does this package have?
- Browse the *String* class in the package *java.lang*. As before, the introductory class documentation is followed by three summary tables. Spend sometime reading and navigating the *String* class specification. Don't worry that you don't understand everything in the specification.

*Understanding the specifications:*

- How many constructors are there available in the class *String*?
- How many *String* constructors have one parameter only?
- How many *String* constructors have more than one parameter?
- The literal "Hello There" denotes a *String*. Write the specification of the *String* constructor being used in the following statement:

```
String message = new String("Hello There");
```

- Given a *String* value, which method can you use to get the 5th character of the *String*? Write the specification of this method.
- Given a *String* reference containing "Hello There", which method would you use to produce a *String* in which all the characters are in upper case? Write the specification of such method.
- Are there any commands that change the state of a *String*?

## Using javadoc:

Java comes equipped with a tool called *javadoc* that produces HTML documents similar to the ones you've browsed above. *javadoc* can be used to produce documentation for a single class, a package, or a group of packages. The tool uses the source files written by the programmer, and extracts public features and associated documentation.

As an example, you will produce HTML documentation for a class you wrote in a previous laboratory.

- In a terminal window, with your *Java* directory as the working directory, enter the command:

```
mkdir figuresDocs
```

to create a directory in which the HTML files will be placed.

- You should have a *Java* subdirectory named *figures5* from a previous lab. This directory should contain files *Circle.java* and *Rectangle.java*. Now generate the HTML documentation for either of these classes. For instance, the to produce documentation for the class *Rectangle*, key:

```
javadoc -d figuresDocs figures5/Rectangle.java
```

The “-d” option specified the directory in which the HTML files are to be placed. If it is omitted, they are placed in the current working directory.

*javadoc* can be given one or more source files, as in the above example, or a package. For instance,

```
javadoc -d figuresDocs figures5
```

will produce documentation for all the classes in the package *figures5*.

- If you have only one browser window open, press *Alt-1* to open another. Using the browser, open the file *Java/figuresDocs/index.html*. (Select the *Open Page* option from the *File* menu. Make your selection by either keying the file name in the sub-window text field, or selecting *Chose File* from the sub-window. Finish by pressing *Open in Navigator*.)
- Read the documentation. Could someone adequately use your class by reading what you have written?
- Have the browser generate a printout of your class specification. (Click in the “class” frame to make it active, then choose *Print Frame* from the *File* menu.)

## Developing good specifications:

The specification of the class and its methods should be clear and complete. A programmer who wants to use the class must get precise information about the purpose of the class and its methods, what is required from the client to use a method, the meaning of values returned from queries, the initial state of the object after creation, and the expected changes of state resulting from commands. The meaning of parameters and the range of possible values that they can be provided by the client must also be clearly explained.

- Open and browse the specifications for the class *Counter*, located in *docs/*.

A user of the class *Counter* should be able to answer the following questions easily, by reading the class specifications. (Note that some information in a class specification is implicit. For instance, if a method returns an `int`, in the absence of any documentation to the contrary, you must assume that any `int` value is possible.)

- Read the specifications and attempt to answer the following questions. If you cannot answer the question, explain why.
  - What is the initial count of a *Counter* instance, immediately after creation?
  - What is the purpose of the constructor argument `number`?
  - Can the constructor argument `number` be negative?
  - By how much will the `incr` command increment the count?
  - Can the count become negative? (In particular, is it legal to decrement a *Counter* with a zero count?)
  - How large can the count become?
  - How small can the count become?
  - What is `tally`?
  - What does it mean if the query `oddOrEven` returns *true*?

### Post-lab:

As directed by your instructor, submit answers to all the questions and a printout of specification of one of your classes.

### UNIX Review:

In this section, we review some UNIX commands and concepts from the previous labs. It's worth while spending a bit of time experimenting to make sure you understand this.

#### Files:

- A *file* is a named collection of information.
- A *directory* is a file that contains (references to) other files.
- An ordinary (non-directory) file can contain printable text (ASCII characters) or binary data.
- A file name can be *absolute* or *relative*.

An absolute file name starts at the root directory (`'/'`), and names all the directories on the path from the root to the file. For instance, the absolute file name for the Perl interpreter is `/usr/local/bin/perl`.

- In a terminal window, key

```
ls -l /usr/local/bin/perl
```

You will see the following output:

```
-rwxr-xr-x  2 root    other      897576 Dec 16  1999 /usr/local/bin/perl
```

Some information included in this line:

- The leading ‘-’ means that this is an ordinary file, and not a directory.
- The next nine characters have to do with file permissions.

The first three characters (‘`rwX`’) mean that the owner of the file has read, write, and execute permissions.

The next three (‘`r-x`’) mean that the owner’s group has read and execute permissions, but not write permission.

The final three (‘`r-x`’) means that everyone else (you, for instance) has read and execute permissions, but not write permission. (In particular, you can run the Java compiler and copy it, but you can’t change it or delete it.)

- ‘2’ is the number of links - don’t worry about what this means.
- ‘root’ is the owner of the file, ‘other’ the owner’s group.
- ‘897576’ is the size of the file in bytes.
- ‘Dec 16 1999’ is the date and time the file was last modified.
- ‘/usr/local/bin/perl’ is the name of the file.

A relative file name names a file with respect to some directory. An absolute file name always starts with a ‘/’ and a relative file name never does.

- Type

```
cd /
```

in the terminal window to change the working directory to the root directory, and type

```
ls
```

Note that the root directory contains a file (a directory) name `usr`.

- Type

```
cd usr
pwd
```

and note that the working directory is now `/usr`. The name of the Perl interpreter relative to this directory is `local/bin/perl`.

- Type

```
ls -l local/bin/perl
```

and note that the information you see is the same as before.

- Type

```
cd local
pwd
ls -l bin/perl
cd bin
pwd
ls -l perl
```

and note the relative file name is different with respect to different directories.

- Finally, type

```
cd
```

to return to your home directory.

### Commands:

Some useful commands, most of which you've seen before, are listed below. Each is followed by an example or two of use.

**ls** -- display information about a directory or file.

Some options include

**-a** list all entries, including those that begin with a dot (.), which are normally not listed.

**-l** long listing.

```
ls -l
ls -a /
```

**mkdir** -- create a new directory.

```
mkdir hwk1
```

**cp** -- make a copy of a file.

```
cp old new
```

**mv** -- move (rename) a file.

```
mv old new
```

**rm** -- remove (delete) a file.

```
rm new
```

**pwd** -- print the current working directory.

```
pwd
```

**cd** -- change the working directory.

```
cd /usr
```

**more** -- display the contents of a text file.

```
more sample.html
```

**file** -- display information regarding the kind of file.

```
file /usr/local/bin/perl
```

A particularly useful command that you should become familiar with is the manual command, **man**. This command give you all available information about a command.

- Type

```
man ls
```

and scan the output. (The output is displayed using **more**, so you need to press the space bar to move to the next window full.)

The trick to using **man** is to realize that you don't have to read and understand everything. For instance, you should be able to answer the following questions by scanning the output from 'man ls', even though you don't understand much of what is written.

Included in the output from the command

```
ls -l /usr
```

are the following two lines:

```
drwxrwxr-x  6 root    bin          5120 Apr 19  2000 sbin
lrwxrwxrwx  1 root    root         12 Jan 12  2000 spool -> ../var/spool
```

What does the 'd' at the beginning of the first line mean?

What does the 'l' at the beginning of the second line mean?