Understanding the Java API (Application Programmer Interface)

Java provides documentation on its classes and methods in the API (Application Programmer Interface). The repository for this information is at the Sun website: java.sun.com. Since the website changes its format from time to time, the actual webpage organization and URLs may change, so here are general directions to find and use the API, but remember that the look of the webpages at the Sun website may change in the future.

How to get to the API

1. Go to the main sun website: java.sun.com
2. Find “Resources” and in that list find “APIs”
3. In the “List of API technologies”, find the Java version you are using. For this document, I chose the Standard Edition J2SE 1.5.0 link.
4. For the J2SE 1.5.0 API, three frames are presented. The upper left frame lists the packages. The lower left frame lists all the Java classes. The right frame lists all the packages with a short description of each one.

How the API is organized

Java contains a vast number of classes. The classes are grouped into packages. A package is like a library and it contains similar classes. Many package names begin with java, for instance, java.applet. Not all packages begin with java however. The Java packages contain all of the classes that comprise Java. Many of the packages are automatically installed when you install Java, but not all of them are installed.

How Java packages are accessed in a Java program

There is one package that is used so often that Java is configured so that it is automatically accessible to all Java programs. This package is named java.lang. Since it is automatically included for you, you do not need to do anything to access it in your program.

If your program needs to use a class in any other package, however, the package name must be included in your source code by using an import statement. For example, to include the package named java.util, you would code this import statement:

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

The asterisk says that you want to have access to all classes in the package. If you want access to only one class, you can specify that class:

```java
import java.util.ArrayList;
```

You can have as many import statements as you need:

```java
import java.util.ArrayList;
import java.text;
import java.nio;
```

How a package is organized

Let’s look at the java.lang package. Scroll down the list of packages and click on the java.lang link. You will see something like this at the top:

- Overview
- Package
- Class
- Use
- Tree
- Deprecated
- Index
- Help
Below that will be some tables. The table titled \textit{Interface Summary} lists all the Interfaces included in the package. We are interested in the table titled \textit{Class Summary}. Scroll down to that table. This table lists all the classes that are included in the package. Scroll down and find the \textit{String} class and click on that link. You will be taken to a description of the String class.

\textbf{Explanation of the String class}

At the top of the documentation you will see:

\texttt{Overview Package Class Use Tree Deprecated Index Help}

\texttt{Summary: Nested Field Constr Method}

Method will take you to a table that lists and explains all the methods that are in the String class. Click on that link. You will see a \textit{Method Summary} table.

\textbf{Explanation of the Method Summary table}

The method summary table lists all the methods included in the class and gives further information about each method. Sometimes, but not always, the information will include some code samples on how to use the method. Scroll down the table, find the \texttt{length} method, and click on it. You will see the \textit{method detail}. For the length method it looks like this:

\begin{verbatim}
length

public int length()
    Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.
Specified by:
    \texttt{length} in interface \texttt{CharSequence}
Returns:
    the length of the sequence of characters represented by this object.
\end{verbatim}

\textbf{How to read the method detail information}

For each method in a class, the \textit{method signature} is listed. For the length method, the signature is:

\begin{verbatim}
public int length()
\end{verbatim}

- \texttt{public} says the method has public access (anyone can use it)
- \texttt{int} says the method returns an int value
- \texttt{length} is the name of the method
- the empty parentheses say the method does \texttt{not} need any inputs

What is very important is what is not shown. Notice the absence of the word \texttt{static} before public. Whenever you see a method signature \texttt{without the word static}, it means the method is \texttt{an instance method}. \textbf{This is very important.} It means that, in order to use the method, it requires an \texttt{instance} of the class (same thing as an \texttt{object} of the class.) Since the length method resides in the String class, length requires you to place a String object name in front of the method name.
objectname.length();

So, to correctly use the length method, you must create a String object first. You can do this by simply creating a String variable:

```java
String city;
city = "Portland";
```

Now that you have an object, named city, you can find its length:

```java
int result;
result = city.length(); //result will contain the length of city, which is 8 in this example
```

Remember, all methods must be followed by a set of parentheses, and if any inputs are required, they must be included inside the parentheses. In this case, we know that length does not require any inputs, so the () are empty. In fact, if you place anything inside the () for the length method, the compiler will give you an error.

Let's look at the use of some other methods in the String class.

The **concat** method actually has some examples listed with its documentation.

```java
public String concat(String str)
    Concatenates the specified string to the end of this string.

    If the length of the argument string is 0, then this String object is returned. Otherwise, a new String object is created, representing a character sequence that is the concatenation of the character sequence represented by this String object and the character sequence represented by the argument string.

    Examples:
    "cares".concat("s") returns "caress"
    "to".concat("get").concat("her") returns "together"
```

Notice the signature:

- **String** indicates the method returns a String object
- **concat** is the name
- **(String str)** indicates this method requires one String object as input

Here is how to use it:

```java
String city;
city = "Portland";
String state;
state = "Oregon";
String cityState;
      cityState = city.concat(state); //cityState will equal: PortlandOregon
```

This will not work:

```java
      cityState = concat(state);
```
because you must specify an objectname in front of the method name.

Likewise, this will not work:
```java
    cityState  city.concat( );
```
because you must specify an input String to the method.

There is another method named `indexOf`.

```java
    public int indexOf(int ch)
        Returns the index within this string of the first occurrence of the specified character.
```

To use it:

- an `int` value is returned
- the name of the method is `indexOf`
- a `character` variable is required as input

So,
```java
    int result;
    String city = "Bend";;
    char something = 'e';
    result = city.indexOf(something);  //result will equal 1 which is the position of 'e' in "Bend".
```

(0 is always the first position in Strings, not 1)

Once again, these will not work:
```java
    result = indexOf(something);    //missing an objectname
```
or
```java
    result = city.indexOf( );      //missing the required character variable input
```

In summary,

- always specify the **correct name** of the method
  ```java
  length       correct
  lenght       incorrect
  ```

- always specify any and all **required inputs**
  ```java
  city.indexOf(something)     correct
  city.indexOf( )             incorrect
  ```

- always include a variable to capture the **return value** if one is returned
  ```java
  result = indexOf(something);     correct
  indexOf(something);             incorrect
  ```