Lab 6  

Name:_________________________  Checked:_______

Objectives:

• Learn about the Java API
• Test code snippets interactively to explore data representation and casts
• Practice using Math, Random, String, and other classes from the Java API
• Practice writing code to do basic String processing

Preparation: Java API exercise

1. Go through the following exercise to learn about the Java API, writing in your answers. When finished, scan (or take a picture) and submit through Blackboard.
2. Bring it to class and check your answers with your partner.

Learning about the Java API - Exercise

Java derives much of its power from the many classes already defined in the Java Application Programming Interface (aka Java API). But how are we ever to learn and use these classes if we don’t know about them? Any textbook on Java can only begin to cover these classes and the methods defined in them. For a complete listing of these classes and methods you will need to visit the Java 6 API: http://docs.oracle.com/javase/7/docs/api/

Although the information covered in the textbook is sufficient to complete all of the programming and lab assignments for this course, you may find yourself wishing for a “better” class or method, or just more information on a known class or method. The Java API website (see link above) is the place to find that information!

All class definitions are found in the Java API Specifications. API stands for application programming interfaces and is more simply a set of existing “building blocks” for programmers to use to develop programs. The API is divided into packages. Packages contain classes. Classes contain methods. Methods contain Java code. We use methods in our Java programs.

Access the Java API at the link above. Why is it abbreviated to Java SE (what does the SE stand for)?

The API Specifications page is divided into 3 sections. The upper left-hand section is used to navigate to different packages (collections of classes). Below this section is a listing of all Java classes in alphabetical order. The largest section of the page displays details about a selected package or class. At present (before selecting a class or package), all Java packages are listed.

Scroll down the main display section of the page until you find the java.lang package. What does it provide?

The java.lang package is automatically provided/imported for all Java programs. Find the java.util package. What does it provide?
Clicking on any package will get a detailed description of the package. Click on java.util. This detailed description provides 5 summaries of items contained in this package. List the four summaries which are written in the orange background:

_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________

For now, we are interested in the **Class Summary**. This summary lists the classes that are contained in the package. The left column contains the name of the class. Notice that all class names start with a capital letter. The right column contains the description of the class. Scroll down until you find the **Scanner** class. What does it contain?

_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________

Click on the **Scanner** class. You will get a detailed description of what is contained in the **Scanner** class. Notice that the package name - java.util - appears (in small print) above the class name. Scroll down a few pages to see the two summaries available for the Scanner class. What are they?

_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________

Scroll down to the **Method Summary**. The left column indicates the type of information the method will return. The right column contains the method name (underlined), the parameters (in parentheses) and a brief description of the method.

Examine the first method listed for the Scanner class. It is the **close()** method. The left column contains **void**, indicating that this particular method does not return anything. All methods have a return type, even if the return type is simply **void**. The right column tells us the name of the method is **close** and the empty () indicates that this method does not require any parameters to be used. The name of the method is located immediately before the open parenthesis. All methods require parentheses.

Based on this information, you could invoke this method using the programming statement

```
scan.close();
```

where **scan** is an already declared and initialized **Scanner** object.

Let’s look at another **Scanner** method. Locate the method **findInLine()**. As you can see, there are two versions of this method, both of which return a **String**. Look at the version with a parameter of type **String** named **pattern**. The definition tells us that this method “attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.”

Based on this information, you could invoke this method using the programming statement

```
String result = scan.findInLine("xx");
```

where **scan** is an already declared and initialized Scanner object. The variable **result** will then reference the String produced/returned by the method.
Click on the name of the Scanner method `findInLine()`. This will provide you additional information about the method. Notice the line at the top of the page:

**public String findInLine(String pattern)**

This line is known as the method header. This is the same information that we saw in the method summary with the added word `public`. The word public indicates that this method is “publically accessible” so that we can use it. The return type follows and is a `String`. A method only ever returns one type. The word located immediately before the parentheses is the name of the method. Everything listed inside of the parentheses are the parameter specifications.

Choose your browser’s back button to return to the Scanner class’s Method Summary. Let’s look at one more method of the Scanner class. To date, we have used the `nextInt()` method to capture integer input from the user. Locate the `nextInt()` method. This method is interesting because it is listed twice. The first appearance of this method does not specify a parameter and the second appearance of the method does. Note that both `nextInt()` methods return an integer. If you have a Scanner object declared and initialized called `scan` and an integer declared and initialized call `num`, the `nextInt()` method could be invoked one of two ways:

```java
int inputA = scan.nextInt();
int inputB = scan.nextInt(num);
```

Ok … now let’s look at another class – the `String` class. To locate the `String` class, use the left hand alphabetical listing of classes. What package is the `String` class part of?

_________________________________________________________________________

Under the `String` class Method Summary, locate the `String` method `trim()`. For this method, provide the following:

Method return type: _______________________________________________________

Required parameters for the method: _________________________________________

Purpose of the method: ___________________________________________________

What would be displayed as a result of executing the following programming statements?

```java
String fname = "Ben        ", lname = "Franklin";
System.out.println( fname + lname);
System.out.println( fname.trim() + lname);
```

_________________________________________________________________________

There are so many great methods to be used from the `String` class that you will surely return to this class’s API many times! But before you review more or the `String` methods, let’s take a look a look at a special type of class.
The **Math** class is a class that only contains static methods. First, locate the Math class. In which Java package can you find the Math class?

________________________________________

_____________________________________

Scroll down to the Method Summary section of the Math class. Examine the first method called `abs()`. The left column contains **static double**. The word **double** tells us that the return type of the method is **double**. But what does **static** mean? Static tells us that this method does not act on an object from the Math class, but that we can just call this method whenever needed. First, answer these questions about `abs()`:

Method return type: ________________________________________________________________

Required parameters for the method: _________________________________________________

Purpose of the method: _____________________________________________________________

Because `abs()` is a static method, to invoke the method you would use the class name and then the method. For example, executing `System.out.println(Math.abs(396 - 400));` would result in 4.

Review the Math method `ceil()` and answer these questions:

Method return type: ________________________________________________________________

Required parameters for the method: _________________________________________________

Purpose of the method: _____________________________________________________________

Example of invoking the method: __________________________________________________

Java API - What have you learned?
- The Java API is divided into packages
- Packages contain classes
- Class names start with a capital letter
- Classes contain methods
- The name of the method is directly to the left of the open parenthesis
- All methods require parentheses
- Parameters are specified with a type followed by an identifier
- All methods have a return type
- The return type of the method is located directly to the left of the method name

**Be Sure To Bring This Worksheet To Class For The Lab To Check With Your Partner**

Partner name:_________________________ Signature:______________________________
**Part A. Use jGrasp to test some code snippets**

**Reminder:** Open jGrasp and click on the **Interactions** tab (lower part of window). You can type Java statements such as variable declarations, assignment statements, and even loops, although the purpose to test out ideas (as opposed to writing substantial pieces of code). You can type any expression to get its value; type variable names to get their values.

**Try with these Java expressions and note the results:**
```java
int a = 1
int b = 2; // Note: semicolon is optional here
int c = 3
a = c
b = c = 5
while (b < c)
{
    System.out.print(b + " ");
    b++;
}
4 + 3
4 / 3
4.0 / 3
0.7 * 0.7
```

**Tips:**
- Watch the Workbench tab on the top/left of jGrasp window; it lists your variables and their values.
- To avoid re-typing a line of code, use the up-arrow (one or more times)—it remembers the previous lines of code you entered.
- Java expressions that have a value can be evaluated directly. **Statements or directives that have no value need a semicolon.**
- Example:
  ```java
  import java.util.Random;
  if (a > 0) ans = "yes";
  ```

**Experiment with expressions involving casts and the Math class:**
```java
(double) 4 / 3
(int) (0.7 * 0.7 * 100)
double phi = Math.PI / 3
Math.cos(phi)
Math.sin(phi)
Math.sin(phi) * 100
Math.round(Math.sin(phi) * 100)
(int) Math.sin(phi) * 100
(int) (Math.sin(phi) * 100)
```

Villanova University    CSC 1051    www.csc.villanova.edu/~map/1051    Dr. Papalaskari
Practice with Strings:

String word = "evolve"

word.length() ______ word.charAt(1) ______ word.charAt(0) ______
word.toUpperCase() ______ word.replace("e","E") ______
word.replaceAll("e","E") ______

String mutation = word.replaceAll("e","")

mutation ______ mutation.substring(3) ______
word.substring(3) ______ word.substring(2,4) ______

String line = "20 0 544"

line.length() ______ line.trim() ______
line.replaceAll(" ","X") ______

String noSpaces = line.replaceAll(" ","")

noSpaces ______

int number = Integer.parseInt(noSpaces);

number ______

noSpaces + 4 ______ number + 4 ______

Integer.MAX_VALUE ______

Long.MAX_VALUE ______

Double.MAX_VALUE ______

Double.parseDouble(noSpaces) ______

Double.toString(phi) ______

Double.toString(phi).substring(0,5) ______

int n = 0;
while (n < line.length())
{
    System.out.print(line.charAt(n) + "*");
    n++;
}

Output:
For classes that are not in java.lang, we need to issue import directives:

    import java.util.Random;
    import java.text.NumberFormat;

Let’s try using the Random and NumberFormat classes. For the Random class be sure to enter each expression repeatedly and note the values generated.

NumberFormat money = NumberFormat.getCurrencyInstance()
NumberFormat percent = NumberFormat.getPercentInstance()

double amount = 0.83;

amount ______________________
    money.format(amount)      ________
    percent.format(amount)    ________

Random rand = new Random()

rand.nextInt(4) _______ _______ // repeat a few times
    ... _______ _______ _______ _______ _______

Range of values for  rand.nextInt(4) ________________________________

rand.nextFloat() _______ _______ _______ _______ _______ _______ ...  

Range of values for  rand.nextFloat() ________________________________

Similarly, experiment with the following expressions and note the range of values for each one:

Range of values for  rand.nextInt(6) + 1 ________________________________

Range of values for  rand.nextInt(50) + 100 ________________________________

Range of values for  rand.nextInt(10) - 5 ________________________________

Formulate expressions to produce each of the following ranges. Be sure to test each one to make sure you are getting values in the range.

<table>
<thead>
<tr>
<th>Range</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 12</td>
<td>________________________</td>
</tr>
<tr>
<td>1 to 20</td>
<td>________________________</td>
</tr>
<tr>
<td>15 to 20</td>
<td>________________________</td>
</tr>
<tr>
<td>-10 to 0</td>
<td>________________________</td>
</tr>
</tbody>
</table>
Part B. Java programs using String methods

1. Write a Java program that asks your first name and last name and then prints a greeting using your initials.
   For example, an interaction might look like this:

   Please enter your first name: Grace
   Please enter your last name: Hopper
   Great meeting you, G.H., have a nice day.

2. Write a Java program that asks your name and then prints it out one letter per line, ALL CAPS.
   For example, an interaction might look like this:

   Please enter your name: Grace
   Hello...
   G
   R
   A
   C
   E

   Hint: First, create an all-caps version of the string by using the String method toUpperCase(). Use code similar to one of the examples in part A

3. Write a Java program that asks your name and then prints it backwards.
   For example, an interaction might look like this:

   Please enter your name: Grace
   Hello ecarG

4. Write a Java program that counts the number of vowels in some text.
   For example, an interaction might look like this:

   Please enter some text:
   It is never too late to have a happy childhood.
   This text contains 16 vowels.

   Hint: Set up an extra counter to count the vowels, initially zero. Loop through the string as in previous exercises (forward or backward, it does not matter), but for each character instead of printing it, check to see if it is a vowel, i.e., whether it equals ‘a’ or it equals ‘e’, etc. – if so, increment your vowel counter. In order to avoid having to count both upper/lowercase vowels, you can use the all-caps version of the string to count vowels.
Lab 6 Comments

Comments on this lab, please:

What was the most valuable thing you learned in this lab?

What did you like best about this lab?

Was there any particular problem?

Do you have any suggestions for improving this lab as an effective learning experience?