Lab 4 Name:_____________________________ Checked:_______

Objectives:

- Learn how to test code snippets interactively.
- Learn about the Java API
- Practice using Random, Math, and String methods and assorted other methods from the Java API

Part A. Use jGrasp to test some code snippets

It is sometimes useful to try something out without writing a whole program. jGrasp has an **Interactions** tab that allows you to do that.

Open jGrasp and click on Interactions tab (lower part of window). Experiment typing a few of each of the following, noting results on this worksheet as you go along.

- arithmetic expressions
  - `4 + 3`
  - `4 / 3`
  - `(double) 4 / 3`
  - `(double) (4 / 3)`

- variable declarations, assignments
  - `int a = 1`
  - `int b = 2`
  - `int c = 3; // Note: semicolon is optional here`
  - `a = c`
  - `c = 5`

  You can type any expression to get its value; type variable names to get their values:

  - `a` __________
  - `b` __________
  - `c` __________

- Java expressions, involving Math and String classes. Try the following and note the results:
  - `double phi = Math.PI / 3` __________
  - `Math.sin(phi)` __________
  - `Math.cos(phi)` __________
  - `String name = "Grace"`
  - `name.length()` __________
  - `name.charAt(1)` __________

Tips:

- Watch the Workbench tab on the top/left part of the 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:
  - `import java.util.Random;`
  - `if (a > 0) ans = "yes";`

- `name.charAt(0)` ________
- `name.substring(2,4)` ________
- `name.toUpperCase()` ________

- Java statements:

  -
    ```
    if (a > b)
         System.out.println(a);
    else
         System.out.println(b);
    ```
    Output: _____

  -
    ```
    int n = 0;
    while (n < name.length())
    {
       System.out.println(name.charAt(n));
       n++;
    }
    ```
    Output:

- Import directives
  - `import java.util.Random;`
  - `import java.text.NumberFormat;`

- More Java expressions, using Random and NumberFormat classes. Try the following and note the results:
  - `Random rand = new Random()`
  - `rand.nextInt(4)` ________
  - `rand.nextInt(4)` ________ `// repeat a few times`

    `...`
  - `rand.nextFloat()` ________
  - `NumberFormat money = NumberFormat.getCurrencyInstance()`
  - `NumberFormat percent = NumberFormat.getPercentInstance()`
  - `double amount = 0.83;`
  - `money.format(amount)` ________
  - `percent.format(amount)` ________

- Notes about other things you tried:

____

Part B: Java programs using String methods

1. Write a Java program that asks your name and then prints it out one letter per line.

For example, an interaction might look like this:

```
Please enter your name: Grace
Hello...
G
r
a
c
e
```

2. 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
```

3. 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: You need to check each character in the String to find out if it is a vowel, i.e., whether it equals ‘a’ or it equals ‘e’, etc.

3. Modify the previous program to perform its I/O using JOptionPane.

You will need to import `javax.swing.JOptionPane`. Here is some code to get you started:

```
 String sentence; // the input sentence
 int num;         // the number of vowels
 sentence = JOptionPane.showInputDialog("Enter a sentence");
  // ...
  // code to calculate number of vowels
  // ...
 JOptionPane.showMessageDialog (null, "Your sentence has " + num + " vowels");
```

Part C: Java API exercise (homework)

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 programers 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 tell us the name of the method is **`close`** and the empty **`()`** indiciates 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 intialized **`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 assessible" so that we can use it. The return type follows and is a String. A method only ever retuns 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 browers'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 initilaized called `scan` and an integer declared and intialized call `num`, the nextInt() method could be invoked one of two ways:

```
      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 alphabetically 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 paramaters for the method: ___________________________________

Purpose of the method: ___________________________________

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

```
           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 paramaters 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( 3 - 10) );` would result in 7.

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

Method return type: ____________________

Required paramaters 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 parenthesis
- 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