Warmup Exercises: Data types, Casts, Strings, Java API

Objectives:
• Understand effects of binary data representation and the Java data types
• Introduce automatic conversions, casts, and the Math class
• Explore the char type and Unicode representation
• Learn about some and String methods
• Learn how to explore the Java API

Exercise 1: Use the jGrasp Interactions pane to find the values

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 / 3</td>
<td></td>
</tr>
<tr>
<td>4.0 / 3</td>
<td></td>
</tr>
<tr>
<td>1.0 / 200</td>
<td></td>
</tr>
<tr>
<td>1.0 / 2000</td>
<td></td>
</tr>
<tr>
<td>1 + 1 == 3</td>
<td></td>
</tr>
<tr>
<td>0.7 + 0.7 == 1.4</td>
<td></td>
</tr>
<tr>
<td>0.7 * 0.7 == .49</td>
<td></td>
</tr>
<tr>
<td>0.7 * 0.7 - .49</td>
<td></td>
</tr>
</tbody>
</table>

Reminders:
• Open jGrasp and click on the Interactions tab (lower part of window).
• Type any expression to get its value; type variable names to get their values.
• Watch the Workbench tab on the top/left part of the window; it lists your variables and their values.
• You can type Java statements such as variable declarations, assignment statements, conditionals, loops and even import directives!

Exercise 2: jGrasp Interactions: casts and the Math class

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(double) 4 / 3</td>
<td></td>
</tr>
<tr>
<td>(double) (4 / 3)</td>
<td></td>
</tr>
<tr>
<td>(int) (0.7 * 0.7 * 100)</td>
<td></td>
</tr>
<tr>
<td>Math.round(0.7 * 0.7 * 100)</td>
<td></td>
</tr>
<tr>
<td>Math.abs(0.7 * 0.7 - .49)</td>
<td></td>
</tr>
<tr>
<td>Math.abs(0.7 * 0.7 - .49) &lt; 0.000000001</td>
<td></td>
</tr>
<tr>
<td>double phi = Math.PI / 3</td>
<td>phi</td>
</tr>
<tr>
<td>Math.cos(phi)</td>
<td></td>
</tr>
<tr>
<td>Math.sin(phi)</td>
<td></td>
</tr>
<tr>
<td>Integer.MAX_VALUE</td>
<td></td>
</tr>
<tr>
<td>Long.MAX_VALUE</td>
<td></td>
</tr>
<tr>
<td>Double.MAX_VALUE</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 3: jGrasp Interactions: char and Unicode

Note:
- If nothing displays, type the name of the variable again to get its value.
- If you get an error, make a note and try to figure out why the expression or statement causes an error.

```java
char letter = 'G';

String oneLetter = "G"

letter == oneLetter

(int) letter

letter ++

(int) letter

letter ++

letter = letter + 1

oneLetter = oneLetter + 1

oneLetter ++

(char) ('A' + 1)

(char) ('A' + 15)

(char) ('A' + 35)

(char) ('A' + 36)

letter = 'A'
```
Exercise 4: jGrasp Interactions: String methods and more conversions

String word = "evolve"
word.length() ______ word.charAt(1) ____ word.charAt(0) ____
word.toUpperCase() _______ word __________
word.replace("e","E") _______ word __________
word.replaceAll("e","E") _______ word __________
String mutation = word.replaceAll("e","")
               mutation __________ word __________
word.substring(3) __________
word.substring(2,4) __________

String answer = "Yes"
answer == "yes" ______ answer == "Yes" ______
answer.equals("yes") ______ answer.equalsIgnoreCase("yes") ______
answer.charAt(0) == "y" ______ answer.charAt(0) == 'y' ______
answer.charAt(0) == 'y' || 'Y' ______
answer.charAt(0) == 'y' || answer.charAt(0) == 'Y' ______

int n = 0;
while (n < word.length())
{
    System.out.print(word.charAt(n) + "*" );
    n++;
}
Exercise 5: Exploring the Java API

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

```java
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

```java
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 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: 

Villanova University  CSC 1051  www.csc.villanova.edu/~map/1051  Dr. Papalaskari
Summary - What have you learned?

- Understand effects of binary data representation and the Java data types
- Explore automatic conversions, casts, and the Math class
- Understand the difference between String and char
- Explore the char type and Unicode representation
- Learn about some and String methods
- 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 starts with a lowercase letter and is directly to the left of the open parenthesis
- All methods require parentheses for their parameters
- 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

Lab 5 preparation:
1. Complete all warmup exercises in this handout.
2. Consider the following code snippet:

```java
String word = "evolve";
int n = 0;

while (n < word.length())
{
    System.out.print(line.charAt(n) + "*");
    n++;
}
```

Modify it so that it prints
- each letter in a separate line
- with a $ in front of it and
- three asterisks after. So the output should be:

```plaintext
$e***
$v***
$0***
$s1***
$e***
```

3. Be sure to test your code so that it produces the desired output. You can do this in the interactions pane or write a complete Java program to test it.
4. In the submission box on blackboard, copy/paste the code snippet that produces the desired output as described above.
5. You do NOT need to submit anything else or attach any files for this preparation, but bring this exercise worksheet to check with your partner in class.

Partner name:______________________________

Signature:__________________________

Villanova University     CSC 1051     www.csc.villanova.edu/~map/1051     Dr. Papalaskari