4 Interactions

The **Interactions** feature in jGRASP allows the user to enter Java statements and expressions and then execute/evaluate them immediately. This feature is not meant to be a replacement for the traditional edit-compile-run cycle, but rather a convenient way to experiment with Java statements and expressions. The Interactions feature is relevant for beginning as well as advanced users who are programming in Java. The feature was introduced briefly in *Getting Started* and *Getting Started with Objects*. In this tutorial, we provide a more complete description with detailed examples. If you are not familiar with the basic features of jGRASP (e.g., compiling, running, and debugging), you are encouraged to read *Getting Started*.

Objectives – When you have completed this tutorial, you should be able to use Interactions with the Object Workbench, Debugger, and Viewers in jGRASP. You should be able to declare primitive variables, assign values, and use them in expressions. You should be able declare reference variables, create instances of objects, invoke methods on the objects, and use the reference variables in expressions. You should be able use interactions containing variables from the workbench and debugger. You should be able to copy interactions and paste them to a CSD window as source code.

The details of these objectives are captured in the hyperlinked topics listed below.

4.1 Starting Interactions

jGRASP

Let's begin by starting jGRASP and then closing any files that had been left open from the previous session. We need to select the Interactions tab in the lower window of the jGRASP desktop as shown in Figure 4-1. As indicated in the figure, an Interactions session can be terminated by clicking the **End** button, the window can be cleared by clicking he **Clear** button, and the window containing the Interactions tab can be resized by dragging on the partitions or by clicking on the up/down arrows at the top or left end of each of the partitions. Many users find it helpful to switch between a full-width message pane across the bottom of the desktop and a full-height tab pane on the left. The button (⊟ or ⊞) in the lower left corner of the desktop provides convenient way to change the layout of the partitions.

The blue triangle in the Interactions window indicates where we should enter our first interaction. Click in the window to gain focus, and we are ready to begin.

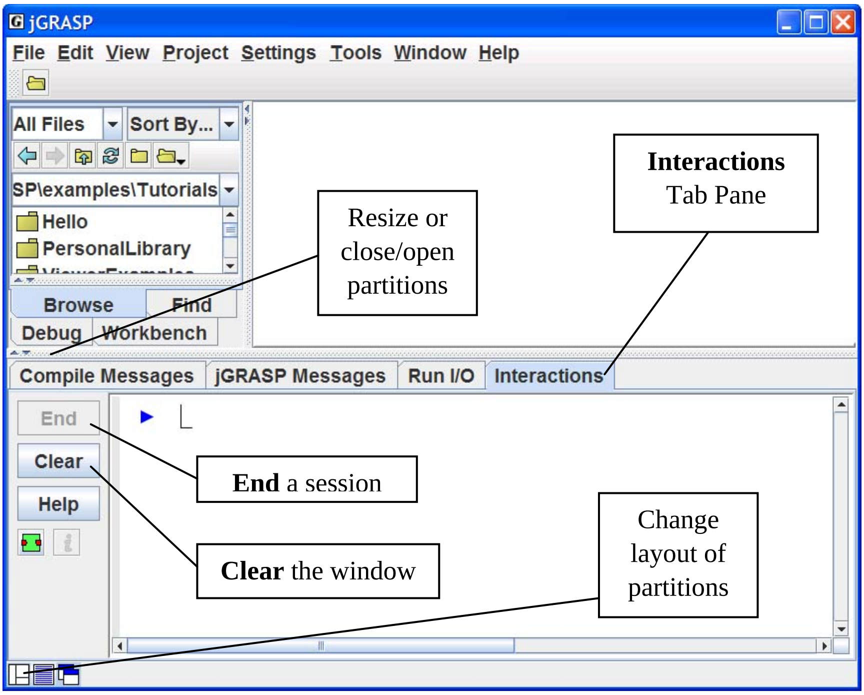


Figure 4-1. The jGRASP Virtual Desktop

4.2 Interactions with Primitives

Our first interactions will explore Java primitive types. Let's begin by declaring an integer variable `i` and assigning it an initial value of 10. After entering the following statement, press ENTER.

```
int i = 10;
```

As soon as you press ENTER, the Interactions session will be started, and the variable `i` should appear on the Workbench.

Now enter the code to declare a variable x of type double:

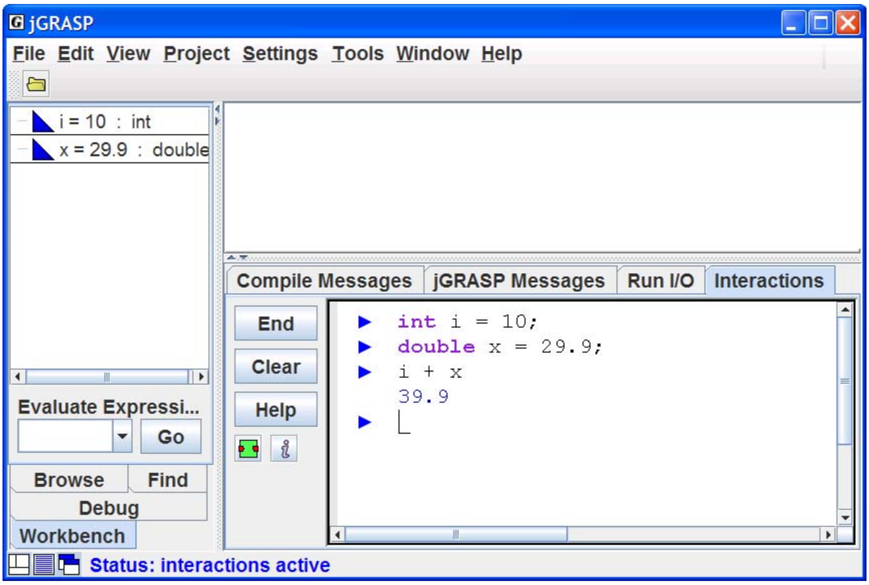
```
double x = 29.9;
```

After pressing ENTER, `x` should appear on the Workbench.

Now let's enter an expression that uses the two variables `i` and `x`. Note that an expression does not end with a semi-colon (;)

```
i + x
```

As soon as ENTER is pressed the expression will be evaluated, and we should see 29.9 displayed below the expression. Figure 4-2 shows the desktop after the interactions above have been entered.



```
i = 10 : int
x = 29.9 : double
```

```
▶ int i = 10;
▶ double x = 29.9;
▶ i + x
  39.9
▶
```

Status: interactions active

Figure 4-2. Our first interactions

Now let's try a few more interactions that use `i` and `x`.

```
        i = 10;

        i = i + 10;

        x = x + 3.5;
```

As you enter each of these, be sure to observe the changes to the variables on the Workbench.

Errors – If a statement contains an error, a message similar to a compiler error message will be displayed.

Repeating a statement – To find a statement you have already entered, press the UP and DOWN arrow keys to scroll through the previous statements (history) one by one until you reach the statement. Then use the LEFT and RIGHT arrow keys or mouse to move around within the statement in order to make the desired changes. Press ENTER to execute the statement again.

Splitting a statement over two lines – When you want to continue a statement on the next line, you can delay execution by pressing Shift-ENTER rather than ENTER. For example, you would need to press Shift-ENTER after the first line below and ENTER after the second line.

```
  System.out.println                 Shift-ENTER
   ("i = " + i + " and x = " + x);         ENTER
```

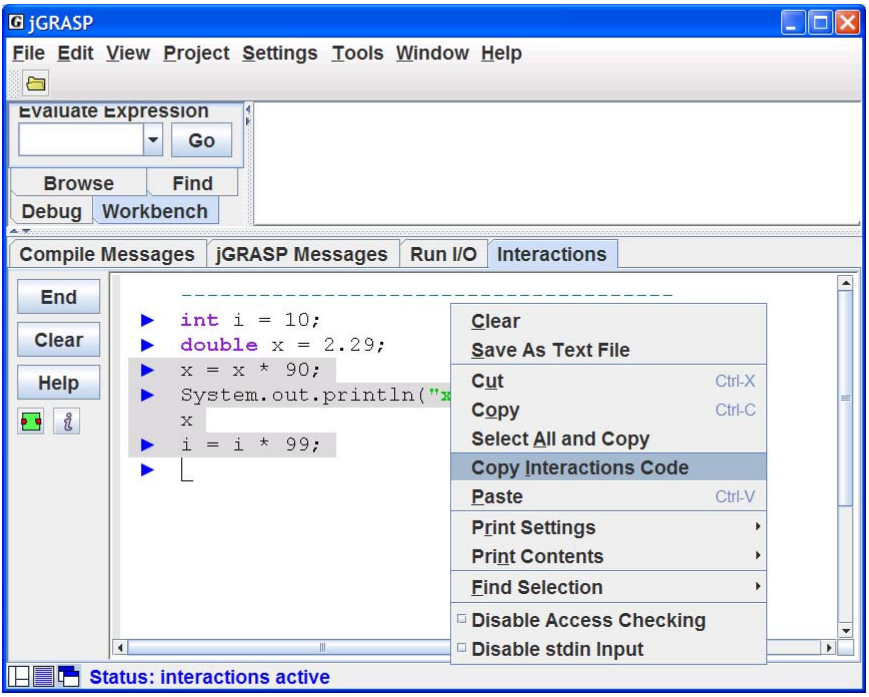
If you simply press ENTER at the end of the first line, Interactions will attempt to execute the incomplete statement and you will get an error message. Below is the result you should see after the statements above are entered with delayed execution.

```
    System.out.println
    ("i = " + i + " and x = " + x);
    i = 10 and x = 22.89
```

Compound statements – When entering statements such as `if`, `if-else`, `while`, `for`, and block statements `{}`, execution is delayed until the "normal" end of the statement is reached. To enter the following while statement on two lines, you can press ENTER at the end of the each line (i.e., there is no need to press SHIFT-ENTER after the first line).

```
    while (i > 0)
    i = i - 1;
```

Copying Interactions – After you have entered one or more statements in Interactions, you may find it useful to copy and then paste them back into Interactions in order to execute them again or perhaps paste them into a CSD window to make them part of a program. To copy statements, first use the mouse to select the range of statements. Next, right-click the mouse to bring up the context menu and then select "Copy Interactions Code" as shown in Figure 4-3. When you do the "paste", it will not include the "x" that was output when `System.out.println("x");` was executed.



```
-------------------------------------
▶ int i = 10;
▶ double x = 2.29;
▶ x = x * 90;
▶ System.out.println("x
  x
▶ i = i * 99;
▶
```

Figure 4-3. Selecting and copying interactions

Viewers – Now let's take a quick side trip and open viewers on `i` and `x` to explore their details. The easiest way to open a viewer on a variable is to simply drag it from the Workbench (i.e., left-click on the item and while holding down on the button, "drag" the item and release the mouse anywhere). Alternatively, you can open a viewer by right-clicking on the item and then selecting "View by Name."

Figures 4-4 and 4-5 show viewers for each of `i` and `x`. Note that **Viewer** is set to **Basic**. This is similar to the view in the Workbench.

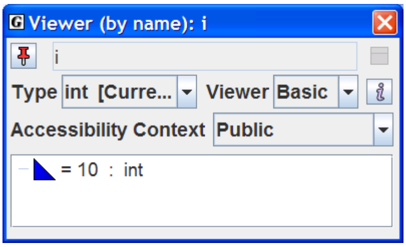


Figure 4-4. Viewer (Basic) of `i`

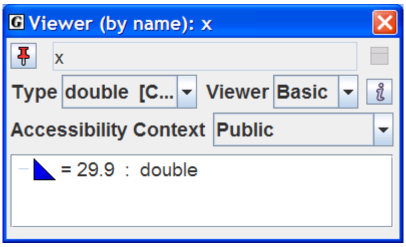
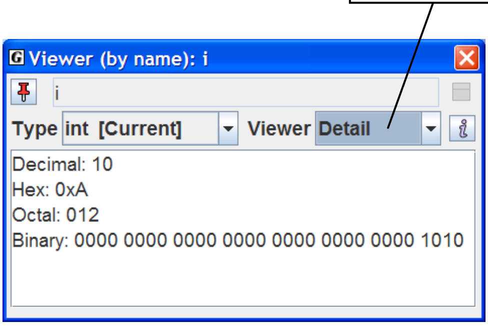


Figure 4-5. Viewer (Basic) of `x`

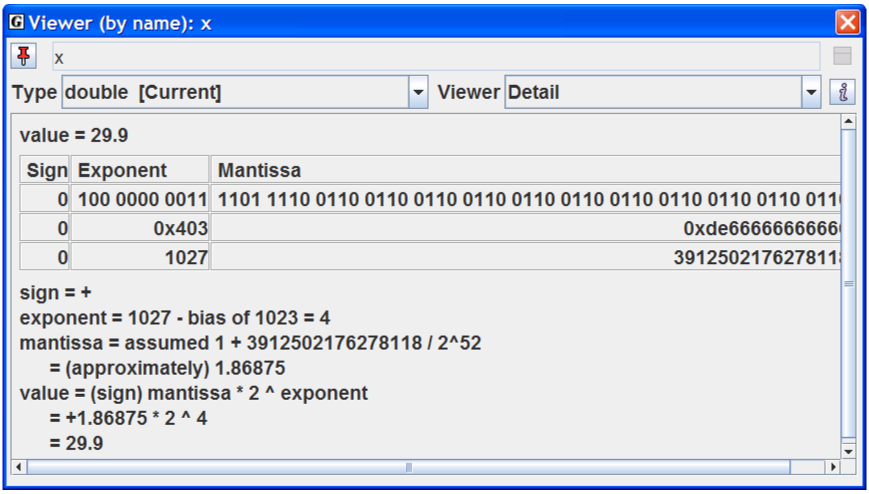
Using the drop-down menu on the viewer, we can change the setting for **Viewer** from **Basic** to **Detail**. Figure 4-6 shows the **Detail** view for `i` with its value in decimal, hexadecimal, octal, and binary. If you change **Viewer** to **Detail** in the viewer for x, you will see the IEEE floating point representation (sign, exponent, and mantissa) for its value as well as the details for how the computation was done. See Figure 4-7.

Change **Viewer** from **Basic** to **Detail**



```
Decimal: 10
Hex: 0xA
Octal: 012
Binary: 0000 0000 0000 0000 0000 0000 0000 1010
```

Figure 4-6. Viewer (Detail) of `i`



```
value = 29.9
```

Sign	Exponent	Mantissa
0	100 0000 0011	1101 1110 0110 0110 0110 0110 0110 0110 0110 0110 0110 0110 011
0	0x403	0xde6666666666
0	1027	391250217627811

```
sign = +
exponent = 1027 - bias of 1023 = 4
mantissa = assumed 1 + 3912502176278118 / 2^52
    = (approximately) 1.86875
value = (sign) mantissa * 2 ^ exponent
    = +1.86875 * 2 ^ 4
    = 29.9
```

Figure 4-7. Viewer (Detail) of x

Exploring the increment operator – Many beginning programmers find the increment and decrement operators confusing. We finish up this section by taking a look at the two forms of the increment operator. Let's enter the following expressions and observe the result returned in Interactions versus the result shown on the Workbench or in the viewer.

```
++i
i++
```

The difference between these two expressions is significant. If you do not see a difference at first, enter each expression again (use UP arrow) and carefully observe the result in Interactions and the result on the Workbench. If you still do not see the difference, see the explanation below.

`++i` and `i++`
`++i` : the ++ before the `i` causes `i` to be incremented by 1 and the new value to be used in the expression. Thus, the value in Interactions will match the value on the Workbench.
`i++` : the ++ after the `i` causes the current value of `i` to be used in the expression and then `i` is incremented by 1. Thus, the value in Interactions will be the old value, and the value on the Workbench will be the new incremented value.

4.3 Interactions with Reference Types

Now let's enter statements in Interactions that involve reference types and instances of objects and primitive types. We begin by entering a statement that declares a reference `s1` of type String and assigns a String literal to it.

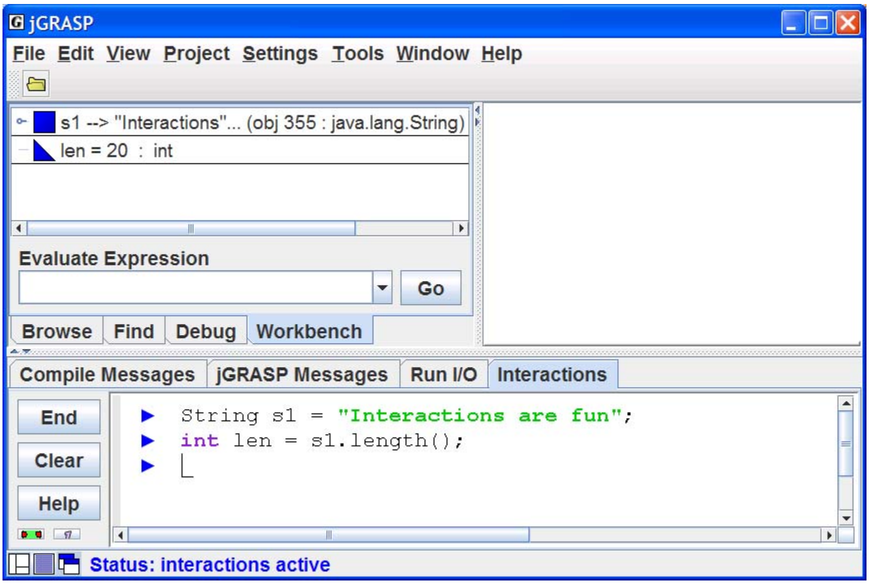
```
String s1 = "Interactions are fun";
```

After ENTER is pressed, you should see an instance of String called s1 on the Workbench.

Now let's enter a statement that declares an integer variable `len` and sets its value by invoking the length() method on `s1`.

```
int len = s1.length();
```

After ENTER is pressed, you should see `len` on the Workbench with a value of 20 as shown in Figure 4-9. Notice the difference in the notation used for the reference variable 🟦 `s1` versus the primitive variable ◣ `len`. We see that `s1` is "pointing to" an object of type String whereas `len` is an int whose value is simply "equal to" 20. This notation is intended to visually remind us that the underlying representations of primitive and reference variables are quite different.


jGRASP
File Edit View Project Settings Tools Window Help
Evaluate Expression
Go
Browse Find Debug Workbench
Compile Messages jGRASP Messages Run I/O Interactions
End
Clear
Help

```
s1 --> "Interactions"... (obj 355 : java.lang.String)
len = 20  :  int
```

```
▶ String s1 = "Interactions are fun";
▶ int len = s1.length();
▶
```

Status: interactions active

Figure 4-9. Interactions with results on the Workbench

Import statements in interactions work in the same way they do in a Java program. For example, to create an instance of the Scanner class, we could enter the following import statement at some point during the Interactions session prior to entering a statement that references the Scanner. Suppose we want to use the Scanner class to input a double and assign it to the variable y. Let's enter the four statements below.

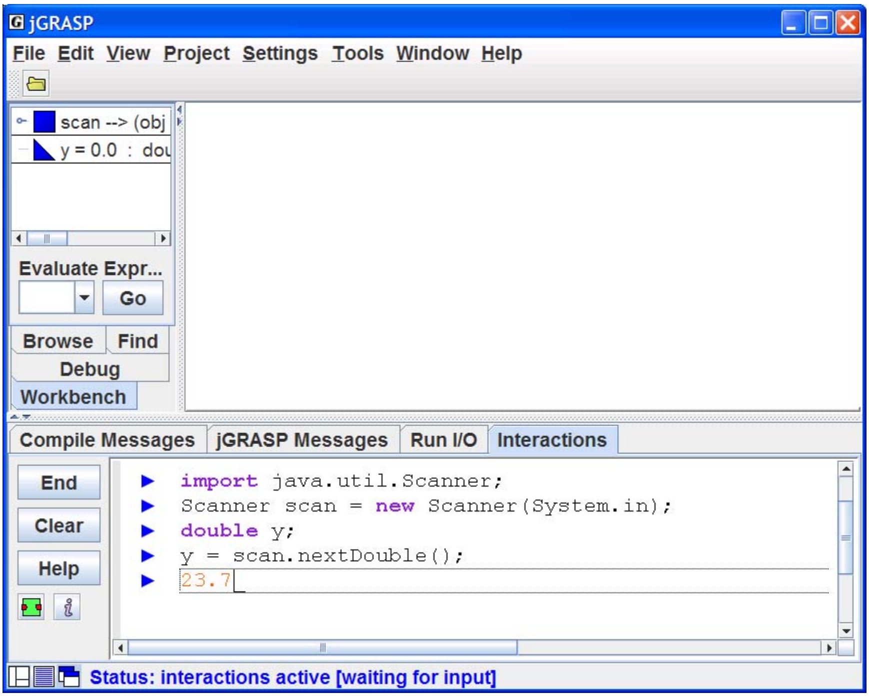
```
import java.util.Scanner;

Scanner scan = new Scanner(System.in);

double y;

y = scan.nextDouble()
```

When the last statement is entered and executed to read in a double, an input box is opened in Interactions to allow you to enter the value. Figure 4-10 shows the desktop after 23.7 has been entered in the input box but before ENTER has been pressed. When ENTER is pressed, the input box will disappear, and `y` will be updated on the Workbench.



```
▶ import java.util.Scanner;
▶ Scanner scan = new Scanner(System.in);
▶ double y;
▶ y = scan.nextDouble();
▶ 23.7
```

Status: interactions active [waiting for input]

Figure 4-10. Interactions to input and assign a double

4.4 Interactions with Your Own Classes

If you want to reference one or more of your own classes in Interactions, the classes need to be visible from Interactions. The easiest way to accomplish this is to open the file containing the class. If you start Interactions while the file has focus (assuming it has been compiled), this class as well as others in the same directory will be available to Interactions. Your file has focus if it is underlined in the Browse tab and/or on the window bar. The name of the file in focus will also be displayed in the title of the jGRASP desktop. If your class is not recognized in Interactions, click the END button and try it again, making sure your file has focus.

4.5 Working with Reference Types – Important Details

Performing interactions with reference types and instances of objects is similar to working with primitives. That is, after you enter a statement or expression, it is executed/evaluated when you press ENTER. The only significant difference is that while primitives are always available, you must ensure that any class which you are referencing is available to Interactions.

If you want to reference one of your own classes that you have opened in a CSD window, you should start Interactions after the file as been opened and while it has focus. Your file has focus if it is underlined in Browse tab and on the window bar, and it is displayed in the title of the jGRASP desktop. If Interactions does not recognize your class, click the END button and try it again, making sure your file has focus. When you start Interactions, all classes in the same directory as the file with focus will also be available to Interactions.

4.6 Interactions with the Debugger

When variables are declared in Interactions they are placed on the Workbench as seen in the examples above. You can also interact with variables in the Debug tab. When you run your program in debug mode and the program stops at a breakpoint, the Debug tab will contain the variables that have been declared and initialized. You can enter statements and expressions in Interactions that use these variables. That is, the variables in the Debug tab are available to Interactions. You may find this useful in debugging. For example, to find the length of the 10,000th element in an array of Strings named *stringArray*, you could simply enter stringArray[10000].length() in Interactions.