The BlueJ Code Pad

This document describes the BlueJ code pad component.

The BlueJ code pad allows quick and easy evaluation of arbitrary snippets of Java code (expressions and statements). Thus, the code pad can be used to investigate details of Java semantics and to illustrate and experiment with Java syntax.

1 Showing the code pad

Summary: To start using the code pad, select Show Code Pad *from the* View *menu.*

The code pad is not shown by default. To show it, use the *Show Code Pad* item from the *View* menu. The main window will now include the code pad interface at the lower right, next to the object bench (Figure 1). Both the horizontal and vertical boundaries of the code pad and object bench can be adjusted to change their sizes.

The code pad area can now be used to enter expressions or statements. On pressing *Enter*, each line will be evaluated and a result may be displayed.

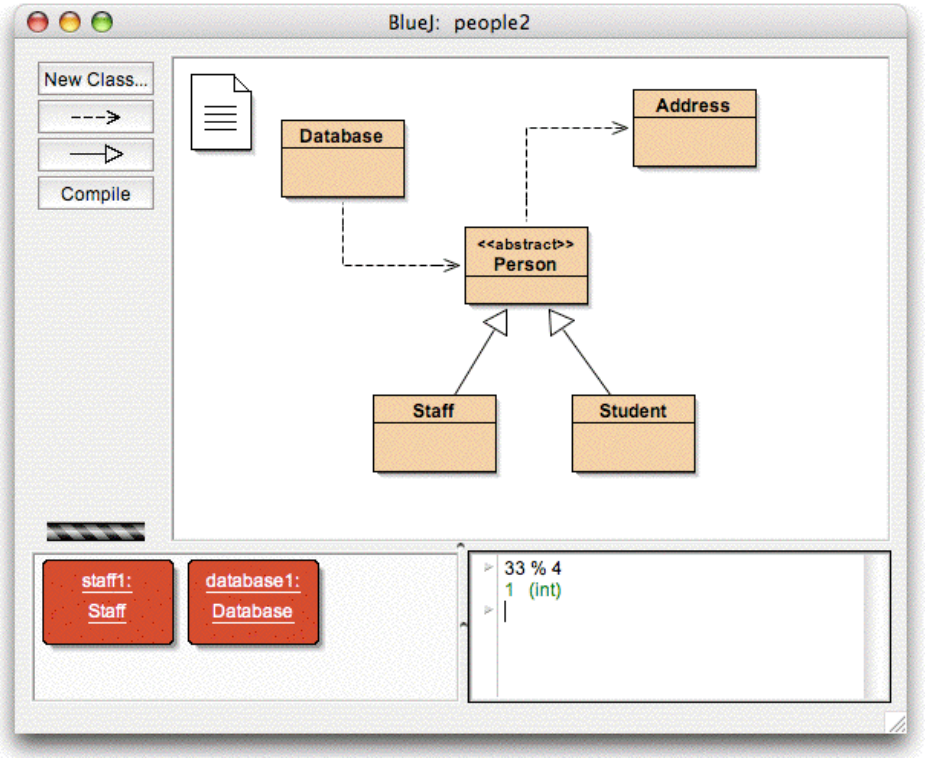

Figure 1: The main window with code pad shown

2 Simple expression evaluation

Summary: To evaluate Java expressions, just type them into the code pad.

The code pad can be used to evaluate simple expressions. Try entering, for example:

```
4 + 45
"hello".length()
```

```
Math.max(33, 4)
(int) 33.7
javax.swing.JOptionPane.showInputDialog(null, "Name:")
```

Expressions can refer to standard Java values and objects, as well as classes from the current project. The code pad will display the result value, followed by its type (in parenthesis), or an error message if the expression is incorrect.

You can also use the objects you have on the object bench. Try the following: place an object of class student onto the object bench (using the class popup menu as described earlier). Name it *student1*.

In the code pad, you can now type

```
student1.getName()
```

Similarly, you can refer to all available methods from your project classes.

3 Receiving objects

Summary: To transfer objects from the code pad to the object bench, drag the small object icon.

Some expression results are objects, rather than simple values. In this case, the result is shown as <*object reference*>, followed by the type of the object, and a small object icon is painted next to the result line (Figure 2).

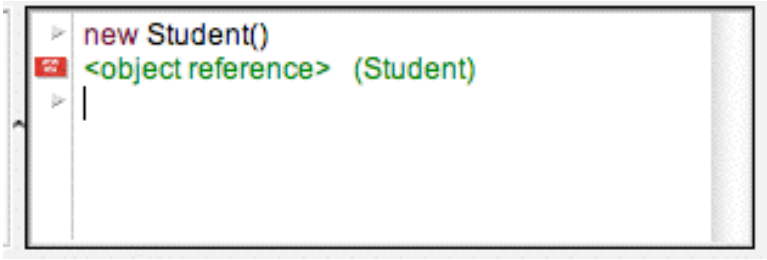

Figure 2: An object as a result of a code pad expression

If the result is a string, the string value will be displayed as the result, but you will also see the small object icon (since strings are objects).

Some expressions you could try to create objects are

```
new Student()
"marmelade".substring(3,8)
new java.util.Random()
"hello" + "world"
```

The small object icon can now be used to continue working with the resulting object. You can point to the icon and drag it onto the object bench (Figure 3). This will place the object onto the bench, where it will be available for further calls to its methods, either via its popup menu or via the code pad.

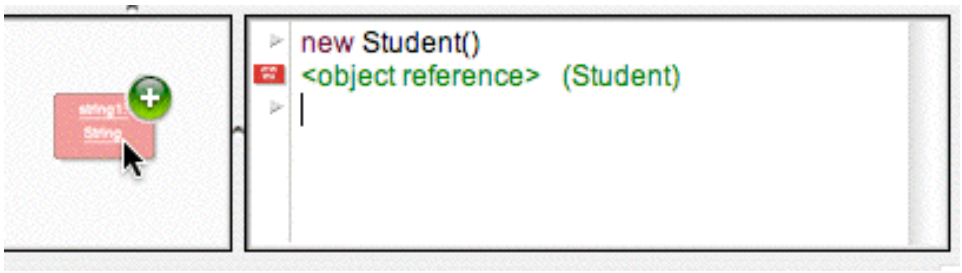

Figure 3: Dragging the object to the object bench

4 Inspecting objects

Summary: To inspect result objects in the code pad, double-click the small object icon.

If you want to inspect an object that was returned as a result from a code pad expression, you can do this without placing it onto the object bench: you can just double-click the object's icon to open the usual object inspector.

5 Executing statements

Summary: Statements that are typed into the code pad are executed.

You can also use the code pad to execute statements (that is: Java instructions that do not return a value). Try these, for example:

```
System.out.println("Gurkensalat");
System.out.println(new java.util.Random().nextInt(10));
```

Statements are correctly evaluated and executed with or without semicolons at the end.

6 Multi-line statements and sequences of statements

Summary: Use shift-Enter *at the end of a line to enter multi-line statements.*

You can enter sequences of statements or statements spanning multiple lines by using *shift-Enter* at the end of the input line. Using *shift-enter* will move the cursor to the start of the next line, but not (yet) execute the input. At the end of the last input line type *Enter* to evaluate all lines together. Try, for example, a *for* loop:

```
for (int i=0; i<5; i++) {
   System.out.println("number: " + i);
}
```

7 Working with variables

Summary: Local variables can be used in single, multi-line statements. The names of objects on the object bench serve as instance fields.

Variables – instance fields and local variables – can be used in the code pad in restricted ways.

You can declare local variables in the code pad, but this is only useful as part of multi-line statement sequences, since variables are discarded between separate inputs. For

example: You can enter the following block as a single, multi-line input, and it will work as expected:

```
int sum;
sum = 0;
for (int i=0; i<100; i++) {
   sum += i;
}
System.out.println("The sum is: " + sum);
```

Entering the same sequence as separate input statements, however, will fail, since the local variable *sum* is not remembered between inputs.

You can think of the input you type as the text within a method body. All code that is legal to write in the body of a Java method is also legal in the code pad. However, every text input you type forms the part of a *different* method, so you cannot refer from one input line to a variable declared in another.

You can think of variables on the object bench as instance fields. You cannot define any new instance fields from within a method body (or from with the code pad), but you can refer to the instance fields and make calls to the objects held in them.

You can create a new instance field by dragging an object from the code pad to the object bench.

8 Command history

Summary: Use up-arrow *and* down-arrow *keys to make use of the input history.*

The code pad keeps a history of your previously used inputs. Using the *up* or *down* arrow keys, you can easily recall previous input lines, which can be edited before being reused.