Kasper Fisker
Michael Kölling

The BlueJ Environment
Reference manual
Version 1.1 for BlueJ Version 1.3.5

The BlueJ Environment

Reference Manual

Version 1.1

for BlueJ Version 1.3.5

Kasper Fisker
Michael Kölling
Mærsk Mc-Kinney Moller Institute
University of Southern Denmark

How Do I ... ? – Introduction

About This Document

This is the "How Do I..." reference manual for the BlueJ Application Development Environment. This manual tries to answer questions about the BlueJ environment. These questions are assumed to be of the form "How do I do this-or-that?" You can replace the *do this-or-that* part with any of the section headings, and hopefully construct a question that is both close to what you wanted to ask and answered in this manual. Of course, you can also just browse through this manual, maybe after you have used the environment for a while, and possibly find out one thing or another that you did not know about BlueJ. There are lots of tips and tricks that are not essential (you get along without them) but which are handy, neat and can speed up your work. Many of these are described in this document.

Related Documents

This document is intended as a reference manual to look up specific tasks when working with BlueJ. If you are interested in an introduction to the BlueJ system, refer to the BlueJ Tutorial.

Report An Error

If you find any errors in this document, please contact us at bluej-support@bluej.org

Using The Context Menu

A common element in graphical user interfaces is the context menu. It is related to an object in the user interface and displays a menu located next to that object. The menu items are often actions that are relevant to the related object, at this particular time and state.

On MacOs the context menu is invoked by holding down the CTRL key while clicking the mouse on a object.

On Windows the right mouse button is clicket on an object.

1   Projects

1.1   Create a new project

menu: Project/New Project...   *shortcut:* —

To create a new project, start BlueJ. Select New... from the Project menu. A file selection dialog will open. Use the file selection dialog to enter a name and location for the project.

Click Ok, and the new project is created and opened.

BlueJ will create a directory for your project, so you should be a bit careful with the name: characters that your file system cannot cope with should not be included in the project name. Also, class paths will be set automatically for your project, and Java has trouble with some characters in the class path. Generally, it is best to stick to alphanumerical characters for project names.

1.2   Open a project

menu: Project/Open Project...   *shortcut:* Ctrl-o

There are two ways to open a project: from the command line of a shell (before Blue is started) type

```
bluej <project-name>
```

to start BlueJ and automatically open the project <project-name>.

If BlueJ is already running, use the Open Project... command from the menu. A file selection dialog will open. Use the file selection dialog to select a project to open. Click Ok, and the project is opened.

1.3   Find out what a project does

menu: *shortcut:* —   *context menu:* Open

A project is described in the project description. The project description is shown as a note icon close to the top left corner of the main screen:

To read the project description, double click on the note icon.

1.4   Copy a project

To copy a project, you first Open a project. This is the original you want to copy. Then you select Save As... from the project menu to save the project under a different name. The Save As... will open a file selection dialog. Use the file selection dialog to enter a new name and location for the project.

1.5   Compile a project

menu: Tools/Compile   *shortcut:* Ctrl-k   *toolbar button:* Compile

To compile a project, select Compile from the menu or toolbar. This function will make an analysis of the current project, check which classes need recompilation, check dependencies, and then compile all classes that need compilation in the appropriate order.

This function will always try to compile all uncompiled classes. If errors are detected, each class with one or more errors is opened and the first error is displayed. To compile one or more specific classes without compiling them all, see Compile A Class.

1.6   Document a project

To document a project properly, you should at least do the following: Write a project description, comment all classes and comment all methods.

Class comments and method comments are both entered in the class's source code. Open a class to enter these comments.

1.7   Execute a project

To execute a project, you first have to find out which method you want to execute. Some projects have a hierarchical structure with one top-level class. The top-level class can have one or more start methods. Some projects have a more open structure in which you call various functions from different classes to perform the various tasks the project offers. If you do not know which class or which method you should use have a look at the project description note. It should be described there. The project description note is further explained in Find out what a project does. As a rule-of-thumb, top-level classes are usually placed in the project window near the top-left corner.

Once you know what you want to execute, you may want to Execute a static method, or you may need to Create an object to call one of its methods. Once you have created that object, you can Call a method.

1.8 Print the package window

menu: Project/Print...   *shortcut:* Ctrl-p

To print a project, select Print... from the project menu. A dialog appears that looks like this:
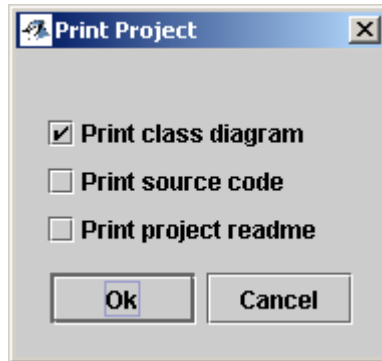
Select one or more of the aspects of the project for printing. After selecting Ok, you will see your standard print dialog.

1.9 Create (executable) Jar files

You can create jar files directly from BlueJ. Jar files include all files of a given project in a single file. This presents a convenient way to transfer a project from one location to another, for example if you want to send it as an attachment in an email. You can also make jar files executable, effectively creating an application that can be started by double-clicking.

In BlueJ, you can create a Jar file by choosing the Export function from the Project menu. You will see a dialogue similar to the one shown here.

To create a Jar file, choose "Store as jar file". If you want the jar file to be executable (so that, for example, a double-click in Windows will start it) select

the "main class" from the popup menu. The main class must contain a standard main method which will be executed to start the application.

With the last checkbox, you can choose to include the source into the jar file. If you are creating a jar file as an executable application, you may choose not to include the source. If you, on the other hand, create the file for the pupose of submitting your project to your teacher for marking, you would probably want to include the sources.

Once you click Continue, you will be prompted for a file name for the jar file. That's all - you're done!

2   Classes

2.1   Create a new class

menu: Edit/New Class...   *shortcut:* Ctrl-n   *toolbar button:* New Class...

To create a new class in a project, select the New Class... menu item or toolbar button. This is only possible while you have a project open.

A dialog will appear to let you enter a name for the new class:
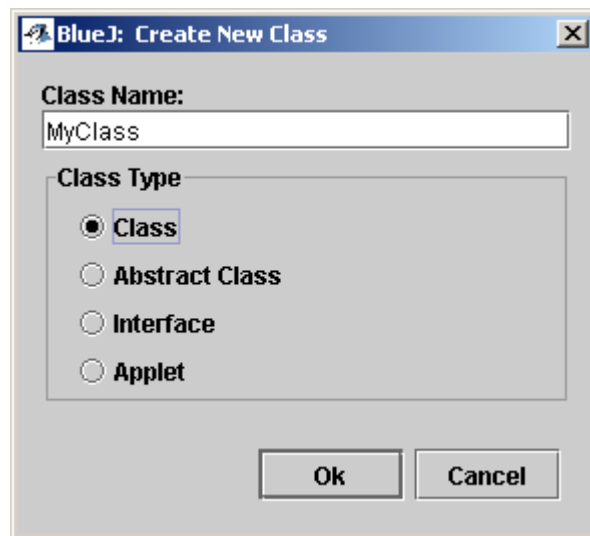
You have to enter a name, and choose one of the options "Class", "Abstract Class", "Interface" or "Applet".

The name must be a valid Java identifier. In short, it must be a word that consists only of letters, digits and the underscore (_). No spaces or other characters are allowed. The first character may not be a digit.

"Class", "Abstract Class", "Interface" or "Applet" options only determine what kind of skeleton is used for the initial source of the class. It is not a final choice: you could change your mind later, and replace the complete source code of, say, a class with code for an interface. The environment would recognise that the class is now an interface and treat it accordingly. Of course, this works the other way around as well.

2.2   Remove a class

menu: Edit/Remove   *context menu:* Remove

To remove a class, select it and then choose Remove Class from the Edit menu. You will be asked to confirm, since removing a class is not reversible. Once a class has been removed, it is deleted in the file system and cannot be restored.

If more than one class is selected, all the classes in the selection will be removed. See <u>Select more than one class or package in the class diagram</u>

2.3 Rename a class

To rename a class, first <u>Open a class</u>, then replace the name of the class in the class header (the first line of the source) with the new name you want it to have. As soon as you save or close the class, the new name will also be shown in the class icon in the class diagram.

2.4 Compile a class

<u>main window:</u>
menu: Tools/Compile Selected   *shortcut:* Shift-Ctrl-k   *context menu*: Compile

<u>editor:</u>
menu: Tools/Compile   *shortcut:* Ctrl-k   *toolbar button:* Compile

There are two ways to compile a particular class: from the main window or from the editor.

In the main window, select the class(es) and then select Compile Selected from the Tools menu.

If the class is open, select Compile from the Tools menu or the toolbar in the editor.

Both of these functions do the same thing: they will first analyse the dependencies of the selected class and check whether a class that this one depends on is uncompiled. Classes that this one depends on are classes used and the parent class (if any). Those classes are compiled first, if necessary, and then the selected class is compiled. Thus, it is not necessary to explicitly compile used classes first. If an error is found it will be highlighted and an error message is displayed (the class will be opened if it is not open already). If no error is found, the class will be marked as compiled in both the editor and the main window.

It is not necessary to save classes before compilation. They will be implicitly saved before compilation takes place if they were changed since the last save.

2.5 Edit a class

To edit the source of a class, you first have to <u>Open a class</u>. If the class is in *interface view*, you have to switch to the *implementation view* (see <u>View the implementation of a class</u>). Then the class is ready to be edited.

2.6 Open a class

<u>main window:</u>
context menu: Open Editor

"Open a class" is the expression we commonly use to shorten the more precise "Open an editor to show the source of a class". You open a class if you

- want to see the interface of the class
- want to see the implementation
- want to edit the class (change the interface or implementation)

You can also open a class by double clicking the class icon.

Depending on which view you used when you last looked at the class, you might see it in *implementation view* or *interface view*. If you have not opened the class in this session, it will initially appear in implementation view. The drop-down box in the editor toolbar indicates which view is currently shown:
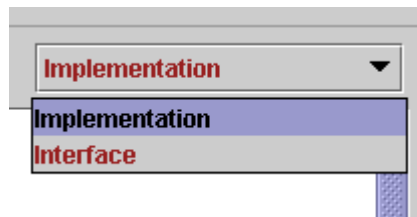
2.7 View the interface of a class

To see the interface of a class, you have to Open a class. If the class display is in implementation mode, select the Interface option in the drop-down box in the editor toolbar.

Showing the interface is only possible if the class has been compiled.

2.8 View the implementation of a class

To see the implementation of a class, you have to Open a class. If the class display is in interface mode, select the Implementation option in the drop-down box in the editor toolbar.

2.9 Create Objects from Available Library Classes

You can create objects of library classes that are not shown in the class diagram (such as java.lang.String or java.awt.Point) by selecting Tools/Use Library Class... . In the resulting dialog, type in (or select from the popup menu) the full class name of the class you want to instantiate. Hit enter, and you will see a list of all constructors and static methods. Select a constructor and click Ok.

You can, in the same way, call static methods of library classes.

3   Project editing

3.1   Create and remove classes

To find out how to create or remove a class from the project, see Create a new class and Remove a class in the "Classes" section of this manual.

3.2   Layout a project

You can change the layout of the project (the icons in the main window) without changing its functionality. You should attempt to create a layout that reflects as closely as possible the logical application structure.

To change the layout, you can Move a class icon on the screen. Move all the icons until the layout appears the way you want it. You cannot move arrows yourself. All arrows are computed and drawn automatically.

You can also Resize a class icon.

3.3   Move a class icon

To move an icon on the screen, you drag it with the (left) mouse button. Dragging means: you click in the icon and keep holding the mouse button down while you move the mouse around. Release the mouse button to drop the icon.

3.4   Resize a class icon

To resize a class icon, you click in the lower right corner of the icon and drag the corner. This corner is separated from the rest of the icon to mark the area in the icon used for resizing.

Resize handle
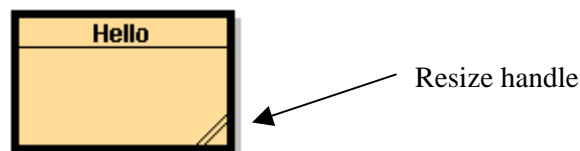
3.5   Move an arrow

You cannot move an arrow explicitly. If you do not like the layout of your arrows, all you can do is to move your classes around. All arrows will be redrawn automatically.

3.6 Add a "uses" arrow

menu: Edit/New Uses Arrow   *toolbar button:* `--->`

To add a "uses" arrow, select the New Uses Arrow item from the menu or toolbar, select the client class , then select the server class. You can also click the client class and then drag to the server class.
(That is: click and hold the mouse button in the client class, move the mouse pointer to the server class while still holding the button down, and release the button in the server class.)

Alternatively, you can add the used class to the source of the client class. The project diagram will be updated as soon as the class is saved or closed.

3.7 Add an "inheritance" arrow

menu: Edit/New Inheritance Arrow   *toolbar button:* `---▷`
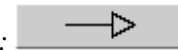
To add an inheritance arrow, select the New Inheritance Arrow item from the menu or toolbar, select the subclass, then select the superclass. You can also click the subclass and then drag to the superclass.
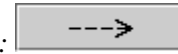(That is: click and hold the mouse button in the subclass, move the mouse pointer to the superclass while still holding the button down, and release the button in the superclass.)

Alternatively, you can insert the superclass to the class header in the source of the subclass. The project diagram will be updated as soon as the class is saved or closed.

3.8 Remove a "uses" or "inheritance" arrow

menu: Edit/Remove   *context menu*: Remove

To remove an arrow, select it and choose Remove from the Edit menu. This will remove the existing arrow.

Alternatively, you can remove the relationship represented by the arrow from the source. The project diagram will be updated as soon as the class is saved or closed.

3.9 Write a project description

The project description is entered in the *project description note*. The project description note is shown on the main screen as an icon:

You can double-click the note to open it, and then write the description of the project into it. It should at least contain a statement describing what the project does, who wrote it, what it is intended for, and how a user starts it (what classes/methods to use to invoke certain functionality). The project description note provides, by default, a structure to enter some of these details. It is a good idea to use this structure, since it makes it easier to find information.

4 Editing source code

4.1 View the source code of a class

How to view the source of a class is described in <u>Open a class</u> and <u>View the implementation of a class</u> in the section about classes.

4.2 Enter a new method

<u>editor:</u>

menu: Edit/Insert Method   *shortcut:* Ctrl-m

To enter a new method, type the method in the editor at the location where the method should appear.

Selecting Edit/Insert method inserts a skeleton for the new method. The skeleton will be indented to the level that the cursor was indented to when invoking this function.

You can configure the method skeleton (See <u>Use different template for new methods</u>) that is used when inserting a new method.

The new method will appear as an option in the class's popup menu.

4.3 Find errors in the source

To check the source of a class for error, you just <u>Compile a class</u>. Compilation will highlight the first error in the source and display the error message in the information area in the editor window.

Compilation only shows one error at a time. To see other errors, fix the first one and compile again.

If you need help to understand the error message, click the icon. BlueJ will then try to explain the error message in straight forward terms.

4.4 Find out what the editor can do

<u>editor:</u>

menu: Options/Key Bindings

Getting to know your editor is a very valuable thing that can greatly increase your work efficiency (and it's a lot more fun too!). Moe (the editor in the BlueJ system) has two mechanisms to find out what functions are supported and what each function does. The key bindings dialog (select Key Bindings from the

Options menu) offers a full list of editor functions. It also shows a brief explanation and the key combinations that call the function.

The following is an example of a key bindings dialog:
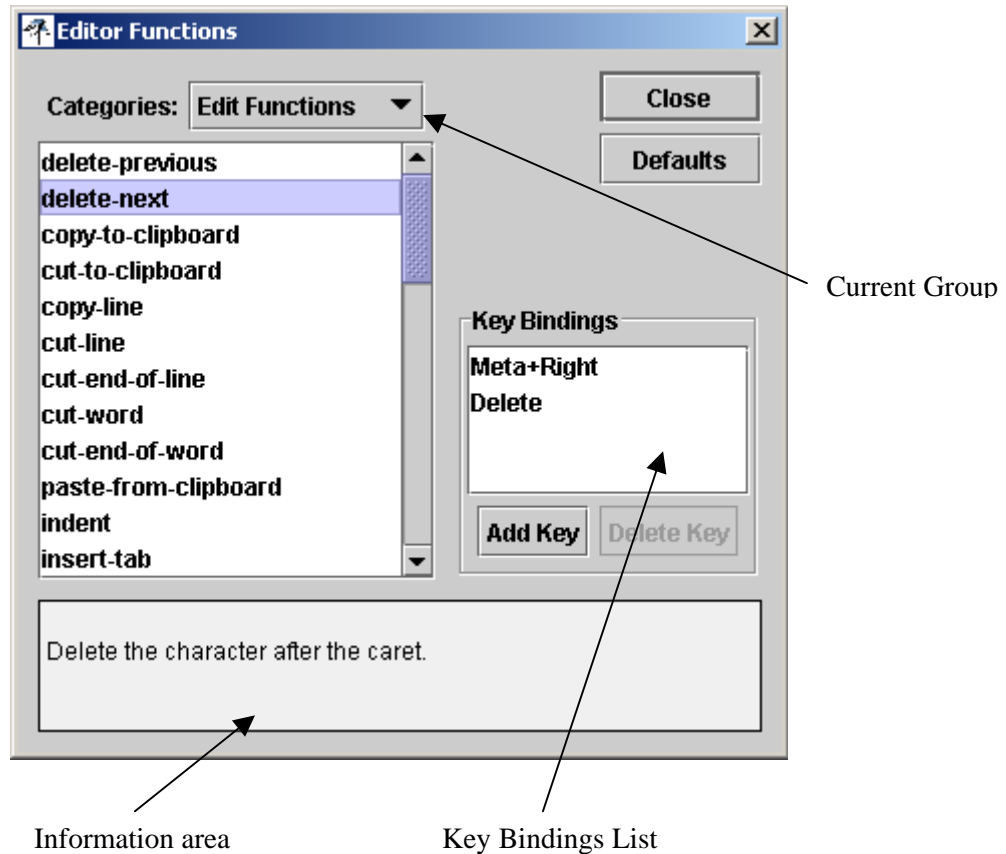
The functions are organised in groups. Choose a function group first to see a list of functions in that group. The information area then shows a description of the function and the key bindings list shows what key combinations call the function.

4.5   Find Out What A Function Key Does

If you quickly want to check what function any key combination invokes in Moe, type Ctrl-D and the key in question. If, for instance, you wonder what the function key F5 might do, type [Ctrl-D] [F5]. Ctrl-D calls the function describe-key. This function reads the next keypress and displays the name of the function that is called by this key in the information area.

4.6   Change key bindings

In the dialog Editor Functions <u>Find out what the editor can do</u> (select Key Bindings from the Options menu) you can specify which key should invoke a

specific editor function. Select the function you want in the list to the left then select the Add Key button. BlueJ will now read the next key or key combination you press and bind it to the selected function.

4.7 Make Stand-out comments

Sometimes it is useful to make a location in your source code stand out. For example, if you are a teacher, and you want to give students a half-implemented class and mark the places where they should enter code, you want them to find those places easily. Or you want to leave notes for yourself marking sections of unfinished work.

You can use stand-out comments for this. Stand out comments start with the symbols /*# (that is: a normal comment symbol followed by a hash sign), for example:

```
public int getTotal()
{
    /*# insert your code here */
}
```

The only difference between normal comments and stand-out comments is that they are displayed in a different colour (pink) in BlueJ's editor. The actual colour can be changed by editing the file 'moe.defs'.

4.8 Use the auto-indent int the text editor

Many people like auto-indentation in their text editor. The BlueJ editor, simple as it is, can do something like this.

Note that the auto-indentation that it will do is not a Java specific, sytax directed indentation - it simply indents every line as the line above.

So, here is how:

The BlueJ editor provides a list of editor functions with flexible key bindings. Select the "Key Bindings..." item from the "Options" menu to see the list.

Have a look at the functions in the "Edit Functions" category. You will see that there are, amongst others, the following functions and key bindings:

KEY	FUNCTION
Tab	Insert-tab
Enter	Insert-break
Shift-Tag	Indent
Shift-Enter	Insert-Break and indent

Using indent and insert-break-and-indent instead of insert-tab and insert-break basically provides auto-indentation functionality. That means that using Shift-Enter at the end of the line and Shift-Tab to indent will give you auto-indentation.

You can now change your key bindings to make this the default behaviour. Use the Key Bindings dialogue to change the bindings so that Enter is bound to

insert-break-and-indent and Tab is bound to indent. You can bind insert-tab and insert-break to Shift-Enter and Shift-Tab.

This should give you simple auto-indent behaviour.

5   Execution

5.1   Create an object

To create an object of a class, right click the class and select a constructor for the class. The constructors (if more than one) will be in the top part of the popup menu and have the word new in front of them.

An object creation dialog will appear. This dialog may look like this:
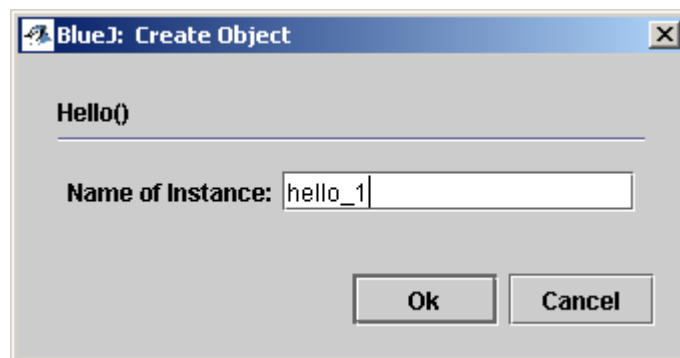
In the name field, you have to enter a name for the object to be created. The name has to be a valid Java identifier.

If the constructor has parameters, the dialog will also include fields to enter the parameters. These fields are needed for the call of the constructor and are identical to the fields in a normal method call dialog. See Call a method for details.

Objects can only be created from compiled classes or applets.

Once the object is created, it will appear on the object bench (the area close to the bottom of the main window) as a red box. The object icon shows its name and class. For example:

5.2   Call a method

To call any method, you must first Create an object. Once you have the object on the object bench, clicking it with the right mouse button will show a pop-up menu which lists all methods of the object:

Select the method you wish to call.

If the method has parameters, you will have to Enter parameters.

After clicking OK, the method will execute, and function results (if any) will be displayed in a separate window.

5.3 Enter parameters

If you Call a method (including a constructor) and that method has parameters, then a parameter dialog will appear to let you enter parameters.

Enter the parameters in the field between the parenthesis or click into the history list to re-use a previously typed parameter list. All parameters are written in exactly the same way they would be written as part of a method call statement in the source code of a class. Strings, for example, have to be written with quotes.

5.4 Execute a static method

When calling a static method you don't need an object. Right click the class and select the method in the popup menu.

If the method has parameters, you will have to Enter parameters.

After clicking OK, the method will execute, and function results (if any) will be displayed in a separate window.

5.5 Execute a "main" method

A main method is the starting point of execution of a class. When the java runtime environment is asked to execute a class, it will search for a main method.

The main method must be declared *static* and *final*. It also has to have *void* as the return type. The main method takes an array of Strings as its only parameter.

BlueJ, in contrast to the java runtime environment, doesn't treat the main method any different than any other static method. It can be called as described in Execute a static method. The Parameter for main is an array of string, this is written as a comma seperated list of strings enclosed in curled brackets. Remember that strings is in quotes. An example:

{"some string","some other string"}

5.6 Execute an object that was returned by a method

If a method call returns an object that you want to examine or use further, you can select that object in the function result dialog and then select the Get button.
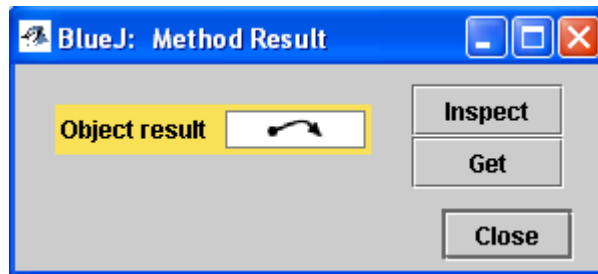
Selecting Get will place that object onto the object bench where you can examine or call it like any of your existing objects.

5.7 Use an object from the object bench as a parameter

You can use an object that lies on the object bench as a parameter to a method call. To do this is simple: While you Enter parameters for the method simply click on the object you want to pass in. This will insert the object's name into the parameter list. (You could also manually type the object's name.)

5.8 Stop the execution of a blueJ program

menu: -   *shortcut:* Ctrl+Shift+R

To interrupt a running BlueJ program, you can Open the debugger window and click on the Halt button or you can right click on the work indicator: and select "Reset Machine"

5.9 Evaluate a single expression

menu: Tools/Evaluate Expression...   *shortcut:* Ctrl-e

To evaluate a short piece of code you can open the "Evaluate Expression" window.
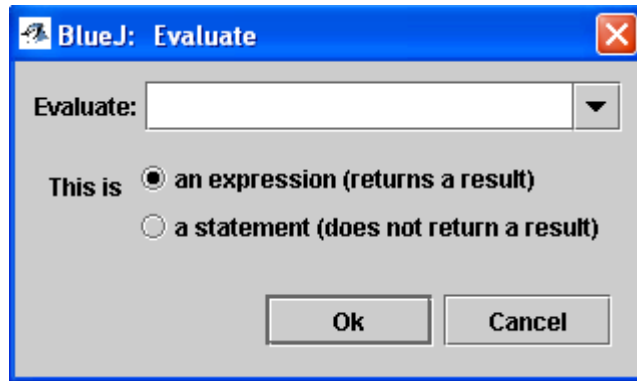
Type the piece of code you want to evaluate in the text field, decide whether the code is an expression or a statement.

An expression is a piece of code that evaluates to either a boolean value, a numerical value or an object. The result will be displayed in an inspector window. Expressions doesn't end with, or contain, a semicolon.

Examples of expressions that evaluate to a boolean value is

"**2 > 5**", "**2+3 == 3**"

or

"**student1.getName.equals("Peter")**"

where student1 is a hypothetical object on the objectbench.

Examples of expressions that evaluates to numerical values is

"**5*5+1**"

or

"**student1.getYearOfBirth**() "

Examples of expressions that evaluate to an object is

"**new String("test")**"

or

"**new Staff("Douglas Adams", 1952, "Room 42")**"

where Staff is a class available from the project or from library.

A statement is a piece of code that doesn't return a result in it self. This doesn't mean that it has no effect. It can write to the terminal or change values in objects on the objectbench. Statements can contain semicolons, so more than one statement can be evaluated at the time.

Examples of statements could be

"**System.out.println("To the terminal")**"

or

"**int a = 5; System.out.println("the integer 'a' has the value " + a)**".

If "student1" is an object on the objectbench the following statement reads, writes and reports a value from that object.

"**String name = student1.getName();**

System.out.println(name);

student1.setName("Peter");

System.out.println(student1.getName());"

6   Debugging

6.1   Inspect an object

The most essential part of debugging is object inspection. Object inspection allows you to look inside an object and see the current values of its instance variables.

You can inspect an object by selecting the Inspect item from the pop-up object menu or by double clicking the object.

To see the pop-up menu, click the object icon on the object bench with the right mouse button. A menu will pop up that looks similar to this:
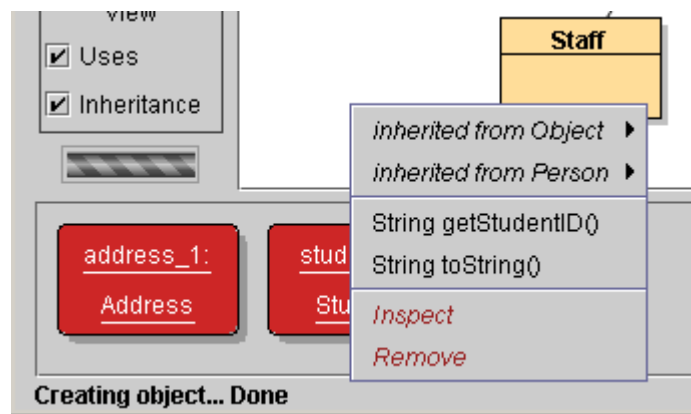
Under the method calls there are two special items. The second from the bottom is Inspect. Select this to open the object. A dialog will be displayed that shows all instance variables, their types and values.
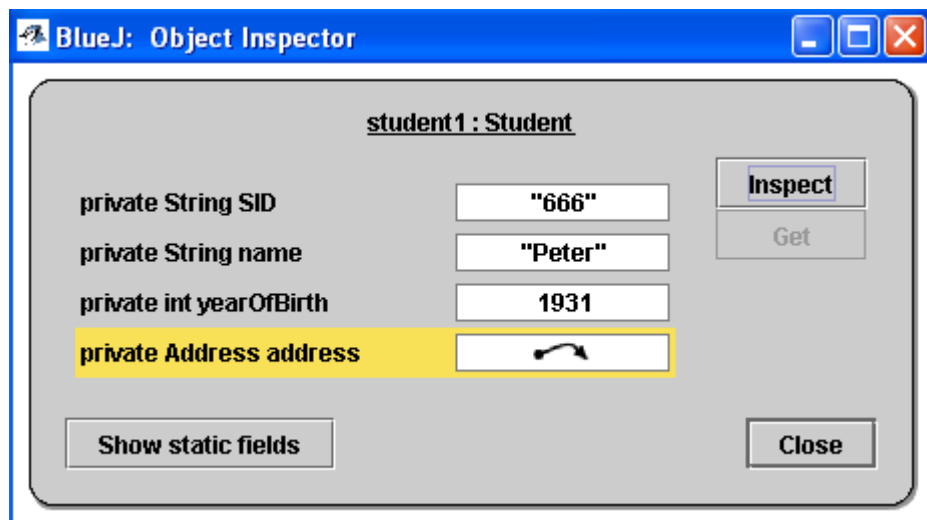
If the value of a variable is itself a complex object, it is only shown as ⬐. If you are interested in the details of such an object, you can inspect that object in turn by selecting it and then clicking the inspect button.

The static fields can be inspected by clicking the Show static fields button.

In this way a whole data structure may be examined. If your structure is, for instance, a linked list, you can inspect the objects referred to by the *next* field until this field is *nulll*.
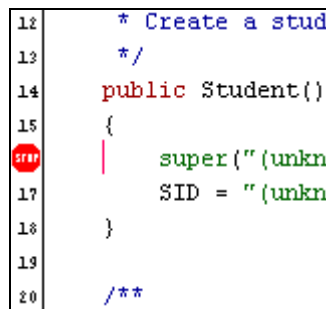
6.2   Set a breakpoint

editor:

menu: Tools/Set BreakPoint   *shortcut:* Ctrl-b

There are two ways to set a breakpoint: You can select Set/Clear Breakpoint from the Tools menu. This will set a breakpoint in the current line (the line the cursor is in).

Or you can click in the tag bar (the area left of the source lines where break points are displayed). Clicking in that area will set a breakpoint at the selected position.

A breakpoint can only be set if the class is compiled. For a compiled class, the tag bar appears white and accepts attempts to set break points. For a class that has not been compiled the tag bar is grey and does not allow breakpoints to be set.

6.3   Remove a breakpoint

editor:

menu: Tools/Clear BreakPoint   *shortcut:* Ctrl-b

There are two ways to clear a breakpoint: You can select Set/Clear Breakpoint from the Tools menu. This will clear the breakpoint in the current line (the line the cursor is in). If there is no breakpoint in the current line, a breakpoint will be set.

Or you can click directly onto the breakpoint displayed in the tag bar (the area left of the source lines where break points are displayed). This will remove the breakpoint.

6.4 Step through my code

To single step through your code, you must first <u>Set a breakpoint</u>. Once execution reaches the breakpoint, the machine will be interrupted, the source code currently executed will be shown and the debugger window will be opened automatically. For example:
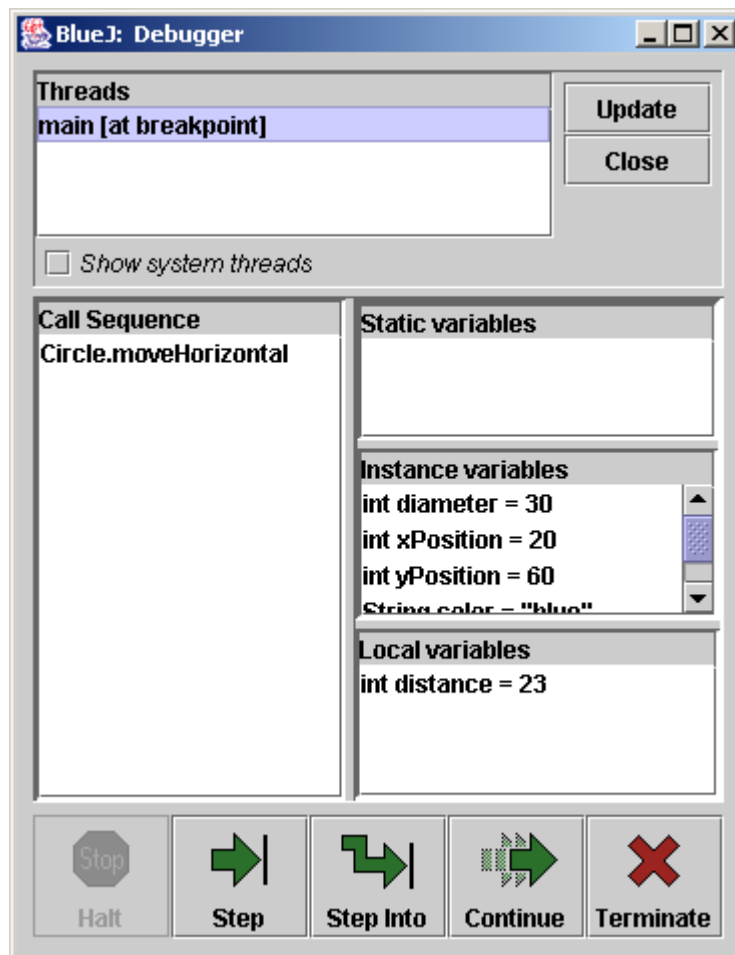
You can then use the Step or Step Into button to step through your code. The current position in the code will be indicated by an arrow in the tag bar. The arrow points to the instruction that will be executed next – that instruction has not yet been executed.

The Step and Step Into buttons differ only if the next statement is a method call: Step will execute the whole method and stop again at the next instruction after

the method. Step Into will go into the method and stop at the first instruction within the method.

Continue may be used to continue execution normally (until execution finishes or the next breakpoint is reached). Terminate may be used to terminate the execution.

6.5 Inspect variable values in my program

If you want to inspect the value of instance variables between interactive method calls, see Inspect an object.

If you want to see the value of local variables or instance variables at a particular moment of the execution, Set a breakpoint at the place in the code where you want to inspect the variable. Once execution reaches the breakpoint and the debugger window is shown.
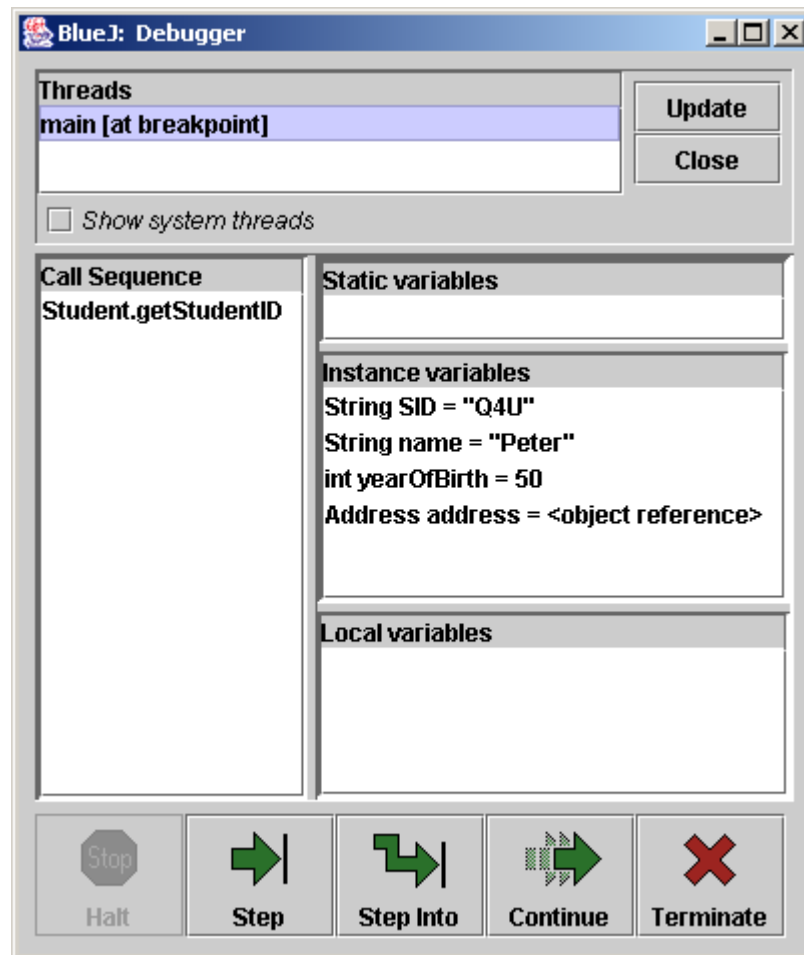
You can inspect any object listed in the variable list by double clicking it, just as for interactive objects on the object bench, as described in Inspect an object.

6.6 Find out about the call sequence at a breakpoint

Once you have [Set a breakpoint](#) and the machine execution stops, you can find out about the sequence of method calls that brought you to this particular piece of code by looking at the debugger display. **INSPECT VARIABLE VALUES IN MY PROGRAM (6.5)**. The left half of the window shows the method call sequence (currently open method calls). It presents a kind of stack display:
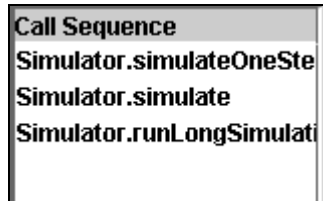
The method at the bottom of the list is the one that was interactively called, with the call sequence progressing upwards in the list towards the current method at the top.

Local and instance variables in intermediate methods can be inspected by selecting the method in this call sequence list.

6.7 Open the debugger window

menu: View/Show Debugger   *shortcut:* Ctrl-d

The debugger window opens automatically when the machine hits a breakpoint.

If you want to open or close the debugger window manually, you can do this by selecting the Show Debugger toggle in the view menu.

Alternatively, you can right click on the work indicator: and select "Show Debugger".

7   The terminal window

7.1   Show/hide the terminal window

menu: View – Show Terminal   *shortcut:* Ctrl-t

To show or hide the Text Terminal toggle the Show Terminal item in the View menu. The text terminal is automatically popped up (shown if it was hidden or de-iconified if it was iconified) if output is written to it or input is expected.

7.2   Clear the screen of the text terminal

terminal window:

menu: Options – Clear   *shortcut:* Ctrl-k

To clear the text terminal, select Clear from the Option menu in the terminal window.

The terminal can also be cleared from within a Java application by printing a formfeed character (unicode 000C).
For example, like this:

```
private void clearScreen()
{
   System.out.print('\u000C');
}
```

7.3   Save the program output to a file

terminal window:

menu: Options – Save to file…   *shortcut:* Ctrl-s

The contents of the text terminal can be saved to a file. This may be useful to document the output of a program. To save the terminal text, select Save to File… from the terminal's Options menu.

7.4   Keep all output

terminal window:

menu: Options – Unlimited buffering   *shortcut:* —

By default, the terminal buffers only the last 48 lines of output. If you want to keep more of the output (for example to save it to a file later), switch on the Unlimited buffering option from the terminal's Options menu.

With unlimited buffering, all output is kept in the terminal. This option can make the output relatively slow if the application produces a lot of text.

7.5   Record which methods is called

terminal window:

menu: Options – Record method calls   *shortcut: —*

With Record method calls switched on, the terminal window will output name and parameters of each method call. To switch it on or off select Record method calls in the terminal's Option menu.

7.6   Clear the terminal window after each method call

terminal window:

menu: Options – Clear screen at method call   *shortcut: —*

The terminal can be made to automatically clear the screen between method calls. Select the Clear screen at method call option from the Option menu.

Switching on both Record method calls and Clear screen at method call, will result in the method call being displayed as a sort of header with the output from the call beneath it. This can then be saved to disk or put in the clipboard.

8 Configuration

8.1 Change BlueJ settings using bluej.defs

There are two ways to change configuration settings in BlueJ: User options are changed from within BlueJ. Use the preferences dialog to change those settings.

Administrator options are changed by editing the configuration file `bluej.defs`. The configuration file is at `<BLUEJ_HOME>/lib/bluej.defs` (where `<BLUEJ_HOME>` is the directory where BlueJ was installed).

The configuration file contains a list of properties in the format

        `property-name = value`

The property names should not be changed. The values may be changed to alter the configuration. It is a good idea to make a backup of this file before you edit it.

All properties in the `bluej.defs` file apply to all users on the system. There is also a user specific configuration file for each user. It is stored in

    `<USER_HOME>/.bluej/bluej.properties`   (Unix)

or

    `<USER_HOME>\bluej\bluej.properties`   (Windows)

The `<USER_HOME>` directory may vary on single user systems. Search for a file named "bluej.properties" if you are not sure where it is on your system.

Each of the properties in system configuration file (`bluej.defs`) can be specified in the user configuration file. Settings in the user configuration file override the settings in the system configuration file.

The individual properties are described in the next few sections of this manual.

8.2 Change BlueJ settings using invokation parameters

BlueJ accepts invokation parameters in the format

        `-Dproperty-name=value`

The propery-names and their values are the ones from the bluej.defs file (See Change BlueJ settings using bluej.defs) which is described in the next few sections of this manual.

To change the interface language of BlueJ to german, BlueJ can be invoked like this:

    java -jar bluej.jar -Dbluej.language=german   (Unix)

or

    bluej.exe -Dbluej.language=german   (Windows)

8.3 Find bluej.defs on my MacOS X

Change BlueJ settings explains how various BlueJ settings can be configured: you can edit the 'bluej.defs' file. On systems other than MacOS this was found in the 'lib' directory. On MacOS, there is no 'lib' directory - so where's the file?

Mac users can edit this file, too. To find it, Ctrl-click the BlueJ application and select '**Show Package Contents**'. Then navigate your way down the folders **Contents/Resources/Java**. Here, you will see the file '**bluej.defs**' (and all other BlueJ configuration files, including moe.defs and the language directories). The files can be edited with any text editor that saves plain ASCII text.

8.4 Use the preferences dialog

main window:
menu: Tools – Preferences…   *shortcut: —*

editor:
menu: Options – Preferences…   *shortcut: —*

To set user preferences from within BlueJ, choose Preferences… from the menu.

The preferences you can specify here are stored in the user configuration file. (See Change BlueJ settings for more information about this file.) User preferences, if changed, will override the settings in the system configuration file.

The settings that can be changed using the preferences dialog include the editor font size, use of syntax highlighting and the Java documentation.

8.5 Add additional help menu items

Users can add their own menu items to the help menu. Each menu item, when selected, will open a URL in a web browser.

New items are listed in the `bluej.help.items` property in the form

`bluej.help.items=<tag_1> <tag_2>`

For every tag, there should be two additional properties:

`bluej.help.<tag_1>.label=<menu_label>`

`bluej.help.<tag_1>.url=<url to open>`

The label will apppear in the menu, the url will be opened in the browser.

8.6 Change my home directory

This is also defined by Java, and if that's fine for you, don't specify this property. This property, if specified, will override Java's user.home property. The property is:

`bluej.userHome=<home_dir>`

8.7 Use a local copy of the documentation

Three menu items in the BlueJ help menu display the BlueJ tutorial, the BlueJ reference manual and the Java standard class libraries. Invoking these functions opens a web browser which, by default, displays documents on a remote web site. You can change the URL to be opened to point to a local copy of these documents. This can improve efficiency and allow you, if installed on the local machine, to use the documentation while being offline. The BlueJ tutorial and reference manual can be downloaded from the BlueJ web site; the Java documentation can be downloaded from Sun at java.sun.com.

To change the URLs, edit the values for the following properties:

```
bluej.url.tutorial
bluej.url.reference
bluej.url.javaStdLib
```

If your web browser is not opened by these functions, see Configure the web browser.

8.8 Configure the documentation generation

Specify the command used for generating documentation by editing the property

```
doctool.command=javadoc
```

by default the javadoc command located in the JDK directory that was used to launch BlueJ is used.

Change the options given to the doctool command by editing the propety

```
doctool.options=-author  -version  -nodepricated  -
package
```

For instance, if you want private methods included in the documentation, change "-package" in the options to "-private". Consult the documentation for javadoc at java.sun.com for more options.

To change the directory name within the project directory where the documantation is stored, edit the property

```
doctool.outputdir=doc
```

If "linkToStandardLib" is true, we will try to use the URL specified in the "bluej.url.javaStdLib" property, to create links. If that URL is not accessible, documentation generation will fail. Therefore, if you want to work offline, set "linkToStandardLib" to false (you can also do that from within BlueJ in the Preferences dialog).

8.9 Change the way applets are executed

If the appletViewer.command line is commented out (default), the viewer command is located in the JDK directory that was used to launch BlueJ. Edit the

```
appletViewer.command=appletviewer
```

property to specify another appletviewer.

8.10 Specify libraries that are to be used in all projects

Sometimes, you may want to make your own libraries generally available in the same style as the Java standard libraries. For example, you may have your own package called "simpleIO" that you want to use. Then you may want to be able to write

```
import simpleIO.*;
```

without the need to copy all the simpleIO classes into every project.

This can be done in two ways.

Using the dialog

Open the "Preferences" dialogue and select the "Libraries" tab. Then add the location where your classes are as a library path. Restart BlueJ.
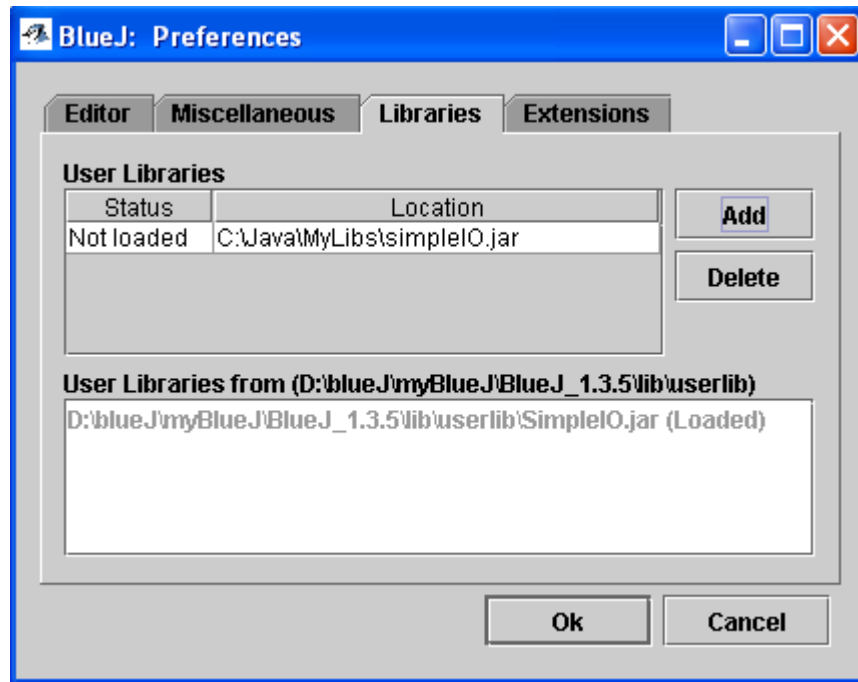
One small thing to look out for: if the classes are in a jar file, select the jar file itself as the library. If your classes are in a named package directory structure (for example in a directory named "simpleIO"), choose the directory that contains simpleIO (not the simpleIO directory itself) as the library!

Using userlib

Inside the lib directory of BlueJ installation there is a directory called "userlib". Any class libraries added as .jar or .zip archives to the userlib directory are automatically added to BlueJ's list of system libraries (called the classpath), for all projects to use. The added libraries will be visible in the "Libraries" pane of BlueJ's Preferences dialog (Tools – Preferences – Libraries) and can be imported in projects as if it was a Java standard library.

The classes made available by a library can also be use interactively with the "Use Library Class" function (See <u>Create Objects from Available Library Classes</u>) or the "Evaluate Expression" (See <u>Evaluate a single expression</u>)

8.11 Specify libraries that are to be used in a single project

Create a folder in the projectfolder and call it "+libs" (without the quotation marks) and put the jar files in that folder.

The classes made available by a library can also be use interactively with the "Use Library Class" function (See <u>Create Objects from Available Library Classes</u>) or the "Evaluate Expression" (See <u>Evaluate a single expression</u>)

8.12 Configure the web browser on non mac/win32 systems

The command used to start your web browser is defined through the properties

```
browserCmd1
browserCmd2
```

BlueJ will first use the first property (Cmd1). If that fails, it will try the second one (Cmd2). A dollar sign ($) in the value will be replaced by the URL to be opened. The default values for Unix are

```
browserCmd1=netscape -remote openURL($)
browserCmd2=netscape $
```

These commands first attempt to open the URL in an already running instance of Netscape and, if that fails, start a new Netscape browser. You can edit these commands if, for instance, Netscape is not in your path or you want to use a different browser.

On windows or mac systems, BlueJ will use the build-in facility to open the default browser.

8.13 Change fonts

Fonts are specified using the following properties:

```
bluej.fontsize              (font size for most interface components)
bluej.editor.fontsize       (font size for editor)
bluej.editor.font           (font face for editor)
bluej.fontsize.printText    (font size for printed source)
bluej.fontsize.printTitle   (font size for printout title)
bluej.fontsize.printInfo    (font size for print footer)
```

All font sizes are integer numbers.

The only font face that can be specified is the font used in the editor for showing class sources. Font faces are specified using a font family name followed by an optional "-bold". Examples are

```
bluej.editor.font=Monospaced
bluej.editor.font=Monospaced-bold
bluej.editor.font=SansSerif
bluej.editor.font=Arial-bold
```

The only fonts guaranteed to exist on all Java platforms are Monospaced, Serif, SansSerif and Symbol. It is generally preferable to use monospaced fonts for editing source code. Sources using a mix of TABs and spaces will not line up with other fonts.

8.14 Change the interface language

The interface language can be changed using the property

```
bluej.language
```

Supported language currently include English, German, Swedish, Afrikaans, Chinese, Czech, French, Italian, Japanese, Korean, Portuguese, Spanish.

Language specific files are stored in `<BLUEJ_HOME>/lib/<language>` (where `<BLUEJ_HOME>` is the directory where BlueJ was installed and `<language>` is a directory with the same name as the language). The language directories contain several text files with BlueJ interface texts.

If your preferred language is not included, it is easy to make a language definition yourself. This is how:

- Copy the directory of a language you understand to a new name representing your language (e.g. copy the directory "english" to "spanish").

- Open each file within the "spanish" directory with a text editor and translate the texts.

- Set the language property in the configuration file to

  ```
  bluej.language=spanish
  ```

If you make a translation for a new language for a new language, please send your translation to bluej@bluej.org - we will then include your language definition in the next release of BlueJ.

8.15 Use a default location for projects

The property

```
bluej.defaultProjectPath
```

can be used to specify the default location of BlueJ projects. The file selection dialog BlueJ uses to open projects will start in the directory specified in this property.

8.16 Switch syntax highlighting on/off

The property

```
bluej.syntaxHilighting
```

is a boolean property (its values are `true` or `false`) for switching syntax highlighting on or off. This property provides the default. Users can modify this setting using their preferences dialog.

8.17 Switch automatic indention on/off

The property

```
bluej.autoIndent
```

is a boolean property (its values are `true` or `false`) for switching autoindention on or off. This property provides the default. Users can modify this setting using their preferences dialog.

8.18 Switch linenumbers on/off

The property

```
bluej.displayLineNumbers
```

is a boolean property (its values are `true` or `false`) for switching the showing of linenumbers on or off. This property provides the default. Users can modify this setting using their preferences dialog.

8.19 Switch bracket matching on/off

The property

```
bluej.matchBrackets
```

is a boolean property (its values are `true` or `false`) for switching bracketmatching on or off. This property provides the default. Users can modify this setting using their preferences dialog.

8.20 Set the tab size

The property

```
bluej.tabsize
```

allow users to specify how many spaces a tab is to be substituted with. This property provides the default. Users can modify this setting using their preferences dialog.

8.21 Make BlueJ backup my sourcefiles

The property

```
bluej.makeBackup
```

is a boolean property (its values are `true` or `false`) for switching backup on or off. This property provides the default. Users can modify this setting using their preferences dialog. When backup is active, a backupfile is created in the same directory as the original each time the original is saved. The backupfile has the same name as the original with a '~' appended.

8.22 Change the compiler BlueJ uses

The property

```
bluej.compiler.type
```

sets the type of the compiler. Currently supported types are: internal, javac and jikes. The propery

```
bluej.compiler.executable
```

specifies the name of the executable to run as the compiler. If it is not specified then BlueJ defaults to the standard name of the specified compiler type (ie javac for type javac and jikes for type jikes).

8.23 Make BlueJ write debug output to a logfile instead of the console

The propery

```
debug
```

specify what BlueJ should do with debug output. When on, debug output goes to the console; when off it is written to a log file in the user's bluej settings directory. The propety

```
bluej.debugLog = debug.log
```

specify which filename should be used.

8.24 Stop BlueJ from optimizing my code

menu: Tools/Preferences…/Miscellaneous/Use optimization

Optimization is on by defalt. Deselect this option to have BlueJ produce unoptimized bytecode. This will ensure that debugging doesn't fails because of difficulties with optimized code.

8.25 Make BlueJ place the menubar at the top of the screen on my mac

The property

```
bluej.macos.screenmenubar
```

is a boolean property (its values are `true` or `false`) for placing the menubar at the top of the window or at the top of the screen. This property only applies to the line of mac os'. This property provides the default.

8.26 Change the default dimensions of the terminal window

The properties

```
bluej.terminal.height=22
bluej.terminal.width=80
```
allow users to set the default dimensions of the terminal window.

8.27 Change the colours

The properties

```
colour.background=208,212,208
colour.graph.background=255,255,255
colour.text.bg=255,255,255
colour.text.fg=0,0,0

colour.arrow.uses=0,0,0
colour.arrow.implements=0,0,0
colour.arrow.extends=0,0,0

colour.target.border=0,0,0
colour.target.bg.compiling=200,150,100
colour.target.shadow=192,192,192
colour.target.stripes=152,152,152

colour.class.bg.default=255,221,153
colour.class.bg.abstract=255,221,153
colour.class.bg.interface=255,221,153
colour.class.bg.unittest=160,65,13
colour.package.bg.default=180,130,44


# object bench
colour.wrapper.bg=205,38,38
colour.wrapper.shadow=152,152,152

colour.menu.environOp=152,32,32
```

```
allow users to set the default colours of BlueJ.
```

8.28 Modify class templates

When creating new source files, BlueJ provides a default template for the source text. These templates are stored in `<BLUEJ_HOME>/lib/<language>/templates/newclass`

where `<BLUEJ_HOME>` is the directory where BlueJ was installed and <language> is the directory with the name of the intended language.

The templates and their file names are

- template for standard Java class (`stdclass.tmpl`)
- template for interface (`interface.tmpl`)
- template for abstract classes (`abstract.tmpl`)
- template for Applet source (`applet.tmpl`)
- template for JApplet source (`japplet.tmpl`)
- template for applet web page (`html.tmpl`)
- template for project note (`readme.tmpl`)
- text added to project note when exporting (`readmeexp.tmpl`)

The file "`shell.tmpl`" is for internal use and should not be modified.

If you want to use a different default text for sources, you can modify these files. If you want to use different templates on a per-user-basis, different files can be created, and the template properties can be modified in the user configuration file to point to these files. The template properties are named:

    template.*

Change the directory for templates

The propery

    bluej.classTemplatePath

is a path that direct BlueJ to the template directory.

Note: if the path contains backslashes, they must be written as double-backslashes (see example in bluej.defs).

There is a tip for teachers who work with labs of machines. What if you like the templates to change in the middle of the semester? Maybe you want different text in the template, or maybe you want additional templates altogether. Often, universities do not allow installations to be changed during term.

To do this, you can configure you initial installation to use a shared template directory. This is done by setting the **bluej.templatePath** property to point to a shared location, and templates will be read from this directory. For example:

    bluej.templatePath = F:\\shared\\bluej\\templates

If you edit the templates, or create new template files in this directory at any time, these templates will be available at the next start of BlueJ.

8.29 Add new class templates

When creating a new class a list of templates is presented to choose from. To add a template, create a new file in the directory

    <bluej>/lib/<language>/templates/newclass

with a ".tmpl" suffix (for example mainClass.tmpl) Names for applets, interfaces or abstract classes should start with "applet", "interface" or "abstract", respectively. Everything else will be treated as a standard class.

The variables $CLASSNAME and $PKGLINE will be substituted for the name of the class and the appropriate package statement, respectively.

The label with which the class is denoted in the "New Class" dialog can then be set by adding the property

```
pkgmgr.newClass.<template-name>=<label>
```

to the file <bluej>/lib/<language>/label

If a new class called "mainClass.tmpl" has been created in the template directory, the property

```
pkgmgr.newClass.mainClass=Class with main
```

will make the "New Class" dialog list the new class as "Class with main"

The order in which the templates is listed in the "New Class" dialog is can be set by the property

```
bluej.classTemplates=stdclass abstract interface applet
```

in bluej.defs. Each item in the list is a filename for a template to be found in the template directory. To add a new template see <u>Add new class templates</u>

8.30 Use different template for new methods

When creating new methods in classes, BlueJ provides a default template for the source text. This template is stored in
<BLUEJ_HOME>/lib/<language>/templates/method.tmpl
where `<BLUEJ_HOME>` is the directory where BlueJ was installed and <language> is the directory with the name of the intended language.

If you want to use a different default text for source, you can modify this file.

9   Unit testing

9.1   Enable the unit testing functionality

menu: Tools/Preferences…/Show unit testing tools

The explicit testing support in BlueJ is initially disabled. To use the testing tools select `Tools - Preferences...` and select the checkbox labelled `Show testing tools`.

9.2   Create test classes

context menu: Create Test Class

Right click a class and select the `Create Test Class` item from the popup menu. The test class is automatically named by adding a "Test" suffix to the name of the reference class.

9.3   Create test methods

context menu: Create Test Method…

Start by selecting `Create Test Method...` from the context menu of the test class.
After selecting this function, the user is prompted for a name for this test. Test names always start with the prefix "test" - if the chosen name does not start with "test" this will be automatically added. Thus, typing "testName" or "name" will both result in creating a test method called "testName".
After typing the name and clicking OK, all interaction will be recorded as part of this test. The 'recording' indicator is on, and the buttons to end or cancel this test recording are enabled as shown below.

recording StudentTest.testName()
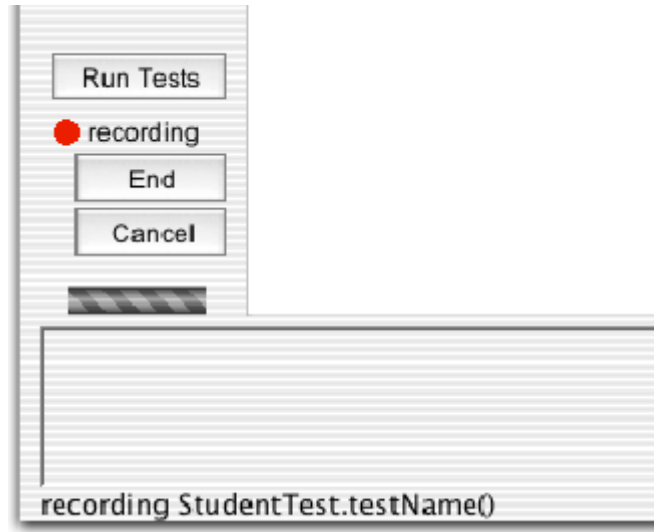
To create a test, do the following:

- Create an object of the class you want to test.

- Call a method that sets the relevant data. You can call as many methods as you like.

- Call a method that returns a variable you now expect to have a curtain value.

You will now see the result dialog. While you are recording tests, the result dialog includes a part that lets you specify assertions on the result (see below). Use this assertion facility to specify the expected outcome of the test.
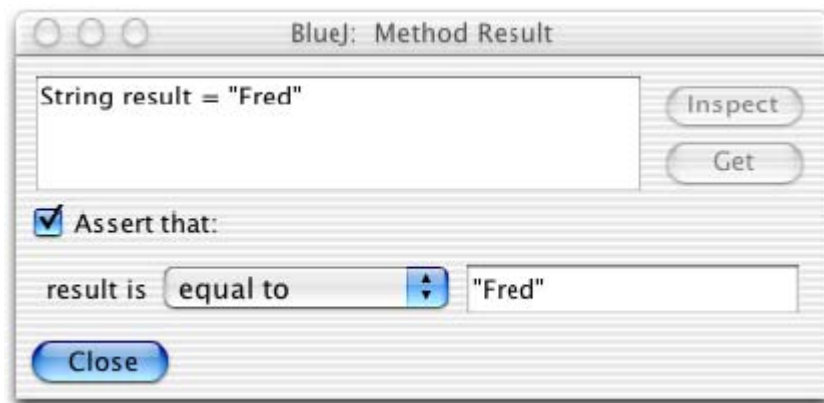
Several different kinds of assertions are available from the popup menu, including tests for equality, null, and not null.

This concludes the test case, so you can now click `'End'` under the test recording indicator to end the recording of the test.

Ending the test results in a test method being added to the test class. This test method is then available for running.

You can use the recording `'Cancel'` button to end the recording and discarding it.

9.4 Run tests

Run all tests by clicking the `Run Tests` button.

Individual tests are run by selecting them from the test class' context menu.

When a test is run individually, one of two things will happen: if the test is successful (the assertions in the test hold) a brief note indicating success is shown in the project window's status bar at the bottom of the window. If the test fails (an assertion fails or any other problem occurs) a test result window is displayed presenting details about the test run. See Interpret test results.

If all tests are run, the test result window is always displayed to show the outcome of the tests.

9.5 Interpret test results

The test result windows shows a summary of test runs and can display failure details.
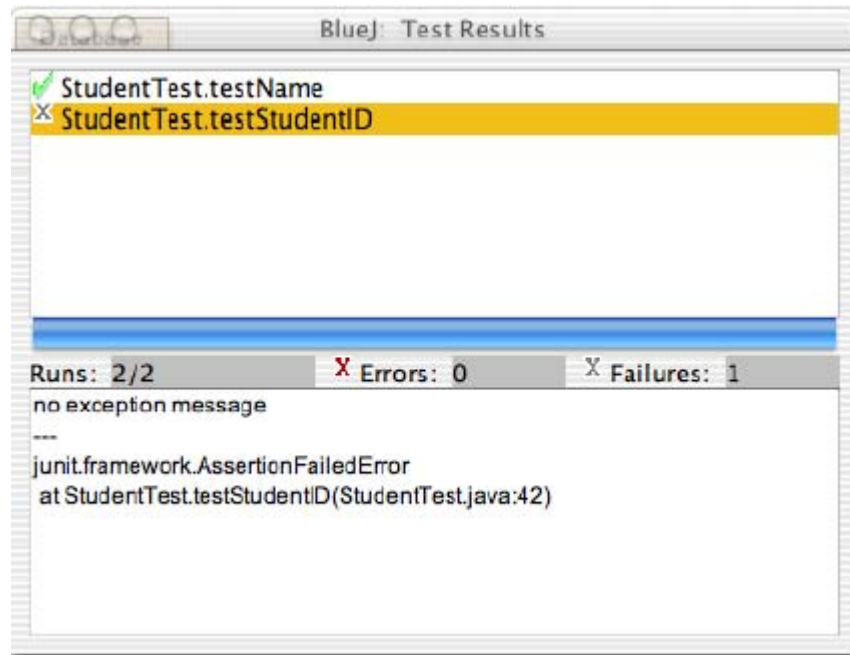
When tests have been executed, the Test Result window displays a summary of the testing outcomes (see above). The top pane of that window shows a list of all executed tests, tagged with an icon indicating their success or failure. A green tick mark indicates a successful test, a grey cross indicates a test failure and a red cross marks an error.

The number of executed tests, errors, and failures is also summarised in the middle section of the window.

A test has a failure (grey cross) if one of its assertions does not hold. For example, the test assertion may specify that a particular method result should not be null, but in this case it was.

A test has an error, if its execution resulted in any other kind of error, such as an unexpected exception being thrown.

For any unsuccessful test, details about its failure can be displayed by selecting the test in the list. The lower pane in the window then displays detail information about this failure or error.

The bar in the middle of the test window is the main summary of the test run: if it appears green, all is well - all tests have been successful. If it is red, there is a problem - at least one test has failed.

Note that on MacOS this bar does not change colour.

9.6   Create a testfixture

To create a fixture for a test class, create the desired objects on the object bench and then invoke the context menu of the test class which needs the fixture. Now select `Object Bench To Test Fixture` from the context menu. The objects in the testfixture will be created before, and removed after, the invokation of each test method.

9.7   Modify an existing testfixture

To add or remove a class from a testfixture, select `Test Fixture to Object Bench` from the context menu of the test class. Add or remove objects to the object bench as usual then select `Object Bench to Test Fixture` from the context menu.

9.8   Write test methods by hand

Select `Open Editor` from the test class' context menu. This will open the editor showing the code for the test class. You can now write test methods by hand. Each test method name must start with 'test', be public and have void as return type.

9.9   Write tests first

The eXtreme Programming methodology suggests that tests should be written *before* the implementation of any method. Using BlueJ's unit test integration this can be done in two different ways.

Firstly, tests can be written by hand, as explained in the previous section. Test writing then works the same way as in non-BlueJ implementations of JUnit.

Secondly, you can create method stubs in the reference class, returning dummy values for methods with non-void return types. Then create tests by using the interactive recording facility, and write assertions according to your expectations of the finished implementation.

9.10 Test multiple classes

To create a test class that isn't attached to a single reference class, click the New Class... button and select Unit Test. Unattached test classes can be used in the same way as other test classes.
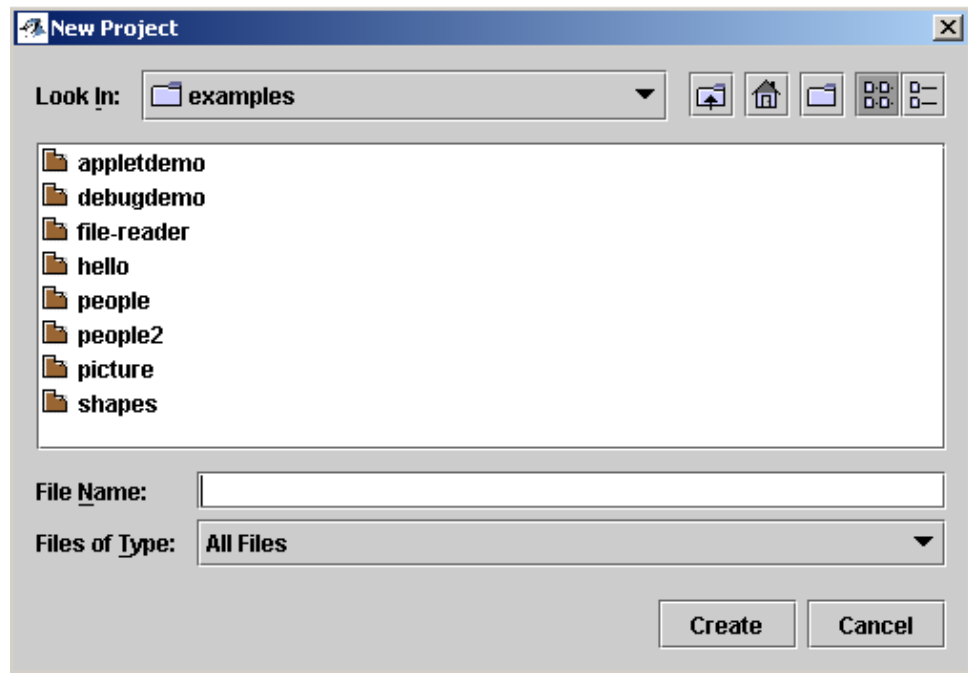
9.11 Test a GUI component

It is possible to fake a click of a button using this method.

```
public void fakeAction(Component c) {
    getToolkit().getSystemEventQueue().postEvent(
        new ActionEvent(c, ActionEvent.ACTION_PERFORMED, ""));
}
```

10 Miscellaneous

10.1 Use the file selection dialog

Use the directory list or the "Look In" dropdown to select a directory where you want to create the new BlueJ project. Type a name for the project in "File Name" field, and click Create.

Note that other BlueJ projects is listed with a different icon than ordinary folders.

Icons:

: Another BlueJ project

: Look in the folder above this one.

: Create new folder.

10.2 Make BlueJ find my file, picture or other resource.

Here is a common problem: You want to access a file on disk (maybe an image, maybe a text file, or anything else for that matter), and you need to specify the name. You write some code similar to this:

Image image = new ImageIcon("myImage.gif").getImage();

or

FileInputStream stream = new FileInputStream("myFile.txt");

Then you discover that it doesn't work. (Or, as some people wrote to us, it "works when I run it from xxx, but not from BlueJ".)

So why is that?

Most people put their image or text file into the project directory, and then want it to be found. The problem is that this depends on the system's "current directory". This is not a BlueJ-specific problem, but a general Java problem. Only, you may discover it for the first time when you use BlueJ.

The current directory is the directory you used when you start Java. When you use BlueJ, the current directory is the directory where you started BlueJ. If you started BlueJ from a DOS window, it is that current directory of that DOS window. If you started BlueJ, say, with a shortcut from the desktop, then it is the desktop.

This means that the above code fragments will probably not find the files when you run them in BlueJ. On the other hand, if you run Java from the DOS window, as below, it may work:

C:\> **cd myProject**
C:\myProject\> **java MyClass**

So is BlueJ wrong? No, because you should never assume that you know what the current directory is when your code runs. Consider the following way to start your Java class through the DOS window, which is an equally valid way to start your application:

C:\> **java myProject\MyClass**

This command will also start your class, but your code trying to load the file will fail, because now the current directory is different.

In other words: Writing code that relies on the current directory is usually a bad idea, and it is prone to break at any moment.

That leaves the question: what should I do instead? The answer is: Use the "getResource" method from java.lang.ClassLoader. This method will find a file anywhere in the current classpath. And the BlueJ project directory is always in the classpath, so it will always find a file there, no matter what the current directory is.

Here is a code example using this method to load an image:

```
public Image getImage(String fileName) {
  URL imageURL =
        getClass().getClassLoader().getResource(fileName);
  if(imageURL == null)
    return null;
  return new ImageIcon(imageURL).getImage();
}
```

Here is a code example reading a text file using getResource:

```
public InputStream openFile(String fileName)
  throws IOException
{
  URL url = getClass().getClassLoader().getResource(fileName);
  if(url == null)
    throw new IOException("File not found: " + fileName);
```

```
    return url.openStream();
}
```

The "getClass()" method is defined in class Object, so it is available in every object. Class URL is in the package java.net. Using this code, your program should work independently of how it was started.

10.3 Select more than one class or package in the class diagram

Click and drag a marquee over the classes that are to be selected. Unselected classes can be added to the selection by clicking them while holding down the CTRL or SHIFT key (on mac it's SHIF or META).

Clicking and dragging a marquee over an uselected set of classes while holding down CTRL or SHIFT key will add these classes to the existing selection.

Classes already in the selection can be taken out by clicking them while holding down CTRL or SHIFT.

The marquee will only add classes to the selection, not take them out.

Right clicking a class in a selection will limit the selection to that class.

11 Troubleshooting

11.1 Get the error messages produced by BlueJ

menu: Help/About BlueJ

Select About BlueJ in the Help menu. This will display a window that tells you which operating system your machine is running, which virtual machine BlueJ is running on, and where it is installed. Most importantly it will tell you the location of the BlueJ Debug log file. This file contains the above information and a log over error messages produced by BlueJ. In case you ever contact the BlueJ-Support team, make sure you attach this file to your email.