Jeliot Program Animation System
User's Guide

Version 3.2

Niko Myller, Andrés Moreno García, Moti Ben-Ari, Erkki Sutinen

March 3, 2004

1 Introduction

Jeliot is a program animation system intended for teaching introductory programming. Programs are animated *automatically*, requiring no modifications or annotations on the part of the instructor or student. While this limits the flexibility of the animation, Jeliot is extremely simple to use so that it is easily accepted by true novices, as well as by their teachers who do not have to invest in learning how to prepare animations. See Ben-Bassat Levy et al. (2003) for a description of the use of Jeliot in the classroom.

Jeliot is written in Java for portability and animates program that are written in Java.

1.1 History

The first member of the Jeliot family of program animation systems was called Eliot; it was developed by Erkki Sutinen and Jorma Tarhio and their students at the University of Helsinki. Eliot was written in C and used the Polka animation library. Later a version was written in Java and the name was changed to Jeliot. (This version is now called Jeliot I to distinguish it from later versions.)

Jeliot I was a flexible system: the user could choose the variables to be animated, the graphical form of the animated elements and multiple views of the same program. It proved too difficult for teaching novices, so a new version called Jeliot 2000 was developed at the Weizmann Institute of Science by Pekka Uronen under the supervision of Moti Ben-Ari. A pedagogical experiment carried out by Ronit Ben-Bassat Levy proved the effectiveness of Jeliot in teaching introductory programming.

Jeliot 2000 only implemented a very small subset of the Java language. The current version Jeliot 3 is a re-implementation designed to significantly extend the range of Java features that are animated, in particular, to cover object-oriented programming. The design and implementation was carried out at the University of Joensuu by Niko Myller and André Moreno-García, under the supervision of Moti Ben-Ari and Erkki Sutinen.

For an extensive discussion of the history of Jeliot, see Ben-Ari et al. (2002).

1.2 Release notes

Version 3.2 contains documentation and supports inheritance, supports: inheritance of classes and `super()` method calls at the beginning of a constructor.

Version 3.1 supports objects creation and object method calls. Line numbering was added for source code editor and viewer. Supports: User made classes (inheritance is not yet supported), constructor calls, object method calls, object field accesses.

Version 3.01 is a maintenance release. Bugs of the initial release were fixed. Support for `switch` statement was added.

Version 3.0 is the initial public release. Supports: Values of type `String` and all primitive types, one-dimensional arrays with primitive types or strings as their component type, expressions including all unary and binary operations except `instanceof`, all the control statements (`if`, `while`, etc.) except `switch` statement and conditional expression (`exp?exp1:exp2`), method invocations, including recursive invocations.

1.3 Installation and execution

The student distribution is in a `zip` file called `jeliotN.zip` where `N` is a version number; it contains the executable `jar` file and several directories: `examples` contains sample programs, `docs` contains documentation files and `images` contains the graphics images that are used in the display. The file `jeliotNsrc.zip` contains the source code, both that of Jeliot itself and that of the DynamicJava package used in the system. The internal documentation of Jeliot including instructions on building the system is contained in a separate document `jeliot-internal.pdf`.

You must have Java (SDK or JRE) Version 1.4 installed in order to run Jeliot. Open the distribution `jeliotN.zip` into the directory `\jeliot`. To run Jeliot, execute `java -jar jeliot.jar`. You can also create a shortcut to the `jar` file and use the icon supplied. There is also a batch files `jeliot.bat` which runs the above command.

2 Using Jeliot

2.1 The User Interface

The Jeliot user interface and display is shown in Figure 1.

The screen is divided into a source frame on the left and an animation frame on the right. Commands may be issued from the menus or from two sets of buttons: one at the top of the screen for file and edit operations, and one at the bottom for animation control. All commands have mnemonics and shortcuts for quicker keyboard control. The following discussion will use the button name for each command, unless it appears only in a menu, in which case, the menu selection will be used.

The file commands are New, Open and Save, and the edit commands are Cut, Copy, Paste and Edit/Select All. The other general commands are Help/Help, Help/About and File/Exit.

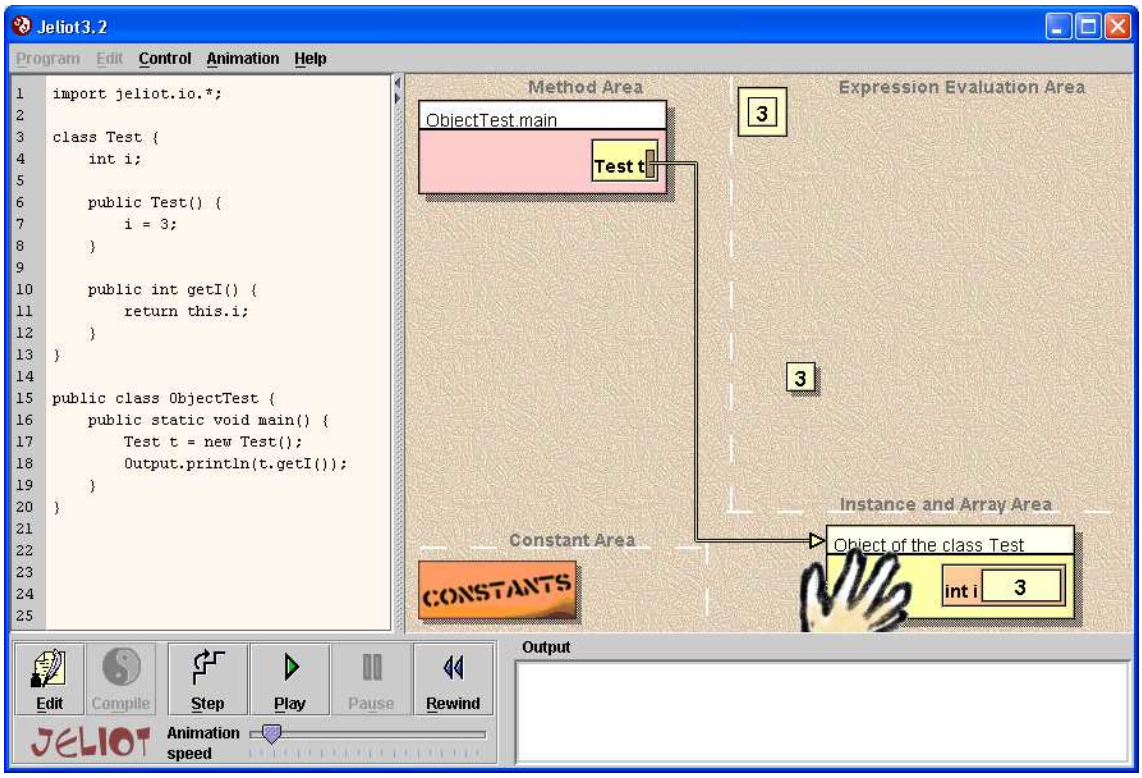

Figure 1: The Jeliot user interface and display

2.2 Editing and compiling

When you open or create a file, the source code will appear in the source frame. Select Compile to compile the file; if the compilation is successful, a curtain will open in the animation frame, and Jeliot will make the transition to Animation mode. If an error is found during the compilation, an error message is shown in the animation frame and the code section that (probably) contains the error is highlighed. You can return to Edit mode at any time by selecting Edit.

2.3 Animation control

The animation is controled by buttons at the bottom of the screen: Step, Play, Pause and Rewind. There is also a slider to control the speed of the animation. Animation may be step-by-step or continuous. Select Step for step-by-step animation. Select Play for continuous animation; you may stop it at any time by selecting Pause and resume the animation by selecting Play again. By checking the option Animation/Pause on message, continuous animation will be stopped whenever a control statement like an `if`-statement

is executed, and a message box describing the control action will popup. When you select OK, the animation continues.

2.4 Animation display

The animation is performed in the animation frame. In addition, there is an output frame at the bottom of the screen. The animation frame itself is divided into several areas:

- Activation frames for each method are cascaded in the upper left hand corner. These will include the name and value of each variable.
- For variables of primitive or `String` type, the value is displayed adjacent to the name. For variables of other types, the value of the variable is a reference to an object which will be displayed at the lower right hand corner of the animation frame. A `null` value is displayed with an electrical ground symbol.
- A *constant store* appears at the lower left hand corner; when an expression needs a literal, it is animated from the constant store.
- The upper right hand corner of the display is where expression evaluation (include method invocation) are animated. Values will be animated from the other areas and the evaluation of the expression will be carried out step-by-step. If a control statement is animated, messages will describe the outcome of the control action.

During the animation, single statements or blocks of statements will be highlighted in the source frame.

3 Java issues

There are two incompatibilities between Jeliot and standard Java.

- All classes must be in a single source file.
- For I/O, import the package `jeliot.io.*;` which provides the methods `void Output.println()`, `int Input.readInt()`, `double Input.readDouble()`, `char Input.readChar()`, `String Input.readString()`.

Jeliot uses DynamicJava (`http://koala.ilog.fr/djava/`) as a front-end and thus accepts almost all Java features that you would want to use for introductory programming, however, the implementation of the animation might not animate all features. Currently, the implementation includes:

- Values of type `String`, all primitive types and one-dimensional arrays.
- Expressions including all unary and binary operations except `instanceof`.
- All the control statements (`if`, `while`, etc.).
- Method invocation, including recursive invocation.
- Allocation of objects, constructors, invocation of methods on objects.

Not implemented:

- Static variables.
- Calls to `super(...)` method with parameters.
- Super field accesses.
- Arrays with components of reference type (except `String`)
- Two or more dimensional arrays.
- Conditional expressions (`exp?exp1:exp2`).
- Array initializers.
- Java 2 SDK API classes' methods cannot return object (except `String` type) or array types (e.g. `object.getClass()` that returns a `Class` instance).
- The used classes hashCode() -method has to return always a unique value.

4 References

Mordechai Ben-Ari, Niko Myller, Erkki Sutinen, and Jorma Tarhio. Perspectives on program animation with Jeliot. In *Software Visualization: International Seminar*, Lecture Notes in Computer Science 2269, 31-45, Dagstuhl Castle, Germany, 2002.

Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A. Uronen. The Jeliot 2000 program animation system. *Computers & Education*, 40(1), 1-15, 2003.