Overview of jGRASP and **the Tutorials**

jGRASP is a lightweight integrated development environment (IDE), created specifically to provide visualizations for improving the comprehensibility of software. jGRASP is implemented in Java, and thus, runs on all platforms with a Java Virtual Machine. jGRASP supports Java, C, C++, Objective-C, Ada, and VHDL, and it comes configured to work with several popular compilers to provide "point and click" compile and run functions. jGRASP is the latest IDE from the **GRASP** (**G**raphical **R**epresentations of **A**lgorithms, **S**tructures, and **P**rocesses) research group at Auburn University.

jGRASP currently provides for the automatic generation of three important software visualizations: (1) *Control Structure Diagrams* (Java, C, C++, Objective-C, Ada, and VHDL) for source code visualization, (2) *UML Class Diagrams* (Java) for architectural visualization, and (3) *Dynamic Viewers* (Java) which provide runtime views for primitives and objects including traditional data structures such as linked lists and binary trees. jGRASP also provides an innovative *Object Workbench*, *Debugger*, and *Interactions* which are tightly integrated with these visualizations. Each is briefly described below.

The **Control Structure Diagram (CSD)** is an algorithmic level diagram generated for Ada, C, C++, Objective-C, Java and VHDL. The CSD is intended to improve the comprehensibility of source code by clearly depicting control constructs, control paths, and the overall structure of each program unit. The CSD, designed to fit into the space that is normally taken by indentation in source code, is an alternative to flow charts and other graphical representations of algorithms. The CSD is a natural extension to architectural diagrams such as UML class diagrams.

The CSD window in jGRASP provides complete support for CSD generation as well as editing, compiling, running, and debugging programs. After editing the source code, regenerating a CSD is fast, efficient, and non-disruptive. The source code can be folded based on CSD structure (e.g., methods, loops, if statements, etc.), then unfolded level-by-level. Standard features for program editors such as syntax based coloring, cut, copy, paste, and find-and-replace are also provided.

The **UML Class Diagram** is currently generated for Java source code from all Java class files and jar files in the current project. Dependencies among the classes are depicted with arrows (edges) in the diagram. By selecting a class, its members can be displayed, and by selecting an arrow between two classes, the actual dependencies can be displayed. This diagram is a powerful tool for understanding a major element of object-oriented software - the dependencies among classes.

The **Dynamic Viewers** for objects and primitives provide visualizations as the user steps through a program in debug mode or invokes methods for an object on the workbench. Textbook-like *Presentation* views are available for instances of classes that represent traditional data structures. When a viewer is opened, a *structure identifier* attempts to automatically recognize linked lists, binary trees, hash tables, and array wrappers (lists, stacks, queues, etc.) during debugging or workbench use. When a positive identification is made, an appropriate *presentation* view of the object is displayed. The *structure identifier* is intended to work for user classes, including textbook examples, as well as the most commonly used classes in the Java Collections Framework (e.g., ArrayList, LinkedList, HashMap, and TreeMap). A future *Viewer API* will allow users to create custom dynamic views of their own classes.

The **Object Workbench**, in conjunction with the UML class diagram, CSD window, and Interactions, allows the user to create instances of classes and invoke their methods. After an object is placed on the Workbench, the user can open a viewer to observe changes resulting from the methods that are invoked. The Workbench paradigm has proven to be extremely useful for teaching and learning object-oriented concepts, especially for beginning students.

The **Integrated Debugger** works in conjunction with the CSD window, UML window, Object Workbench, and Interactions. The Debugger provides a seamless way for users to examine their programs step by step. The execution threads, call stack, and local variables are easily viewable during each step. The jGRASP debugger has been used extensively during lectures as a highly interactive medium for explaining programs.

The **Interactions** (new in jGRASP 1.8.7) feature allows users to enter most Java statements and expressions and then execute or evaluate them immediately. Interactions can be especially helpful when learning and experimenting with new elements in the Java language.

The ***jGRASP Tutorials*** provide best results when read while using jGRASP; however, they are sufficiently detailed to be read in a stand-alone fashion by a user who has experience with one or more other IDEs. The tutorials are quite suitable as supplemental assignments during a course. When working with jGRASP and the tutorials, students can use their own source code, or they can use the examples shown in the tutorials (..\jGRASP\examples\Tutorials\). Users should copy the examples folder to their own directories prior to modifying them. The Tutorials are listed below along with suggestions for their use.

1 Installing jGRASP – Most users will skip this tutorial. However, it does provide details on the installation process as well as instructions for changing default startup settings for jGRASP. This tutorial also describes how to set the system path and the Java classpath from within jGRASP.

2 Getting Started – This tutorial is a good starting place for those new to jGRASP. It introduces the process of creating and editing Java source files, then compiling and running programs. It also introduces interactions, the control structure diagram, and the debugger.

3 Getting Started with Objects – This tutorial is a good starting place for those interested in an *Objects First* approach to learning Java, but it assumes the reader will refer to the previous tutorial as needed. Projects, UML class diagrams, the Object Workbench, and Viewers are introduced.

The topics that are introduced in *Getting Started* and *Getting Started with Objects* are covered in more depth in the following seven tutorials. In most cases, these tutorials may be read as a topic becomes relevant to a user, rather than in the order indicated by their numbers.

4 Interactions – Although the Interactions feature is introduced in *Getting Started* and *Getting Started with Objects*, this tutorial provides examples for several common scenarios, including multi-line interactions and how to copy and paste interactions.

5 The Control Structure Diagram – This tutorial is perhaps best read as control structures such as the *if, if-else, switch, while, for,* and *do* statements are studied. However, for those already familiar with the common control structures of programming languages, the tutorial can be read at any time. The latter part contains some helpful hints on getting the most out of the CSD.

6 The Integrated Debugger – This tutorial can be done anytime. Students should be encouraged to begin using the debugger early on so that they can step through their programs, even if only to observe variables as their values change.

7 Projects – This tutorial discusses the concept of a project file (.gpj) in jGRASP which stores all information for a specific project. This includes the names (and paths) of each file in the project, the project settings, and the layout of the UML diagram. Some users may want to work in projects from the beginning while others want to deal with projects only when programs have multiple classes or files.

8 The UML Class Diagram – The focus of this tutorial is on generating a UML class diagram for a project and then using the diagram as a basis for creating instances for the workbench. This tutorial assumes the user understands the concept of a project and is able to create one (Tutorial 4).

9 The Workbench – This tutorial assumes the user is able to create a project (Tutorial 4) and work with UML class diagrams (Tutorial 5). The workbench provides an exciting way to approach object-oriented concepts and programming by allowing the user to create objects and invoke methods directly.

10 Viewers for Data Structures – This tutorial provides a more in-depth introduction to using Viewers with linked lists, binary trees, and other traditional data structures. Examples of *presentation* views are included for instances of non-JDK implementations for a linked list and binary tree as well as for instances of ArrayList, LinkedList, HashMap, and TreeMap.

For additional information and to download jGRASP, please visit our web site (http://www.jgrasp.org).