

# The Watson Labs



## User Manual



## Table Of Contents

- 0. Introduction To Watson
  - 0.1 Overview
  - 0.2 The Watson Labs
  - 0.3 History of Watson
  - 0.4 Acknowledgments
- 1. General Watson Features
  - 1.1 Activity Page
  - 1.2 Watson Title Bar
- 2. Spreadsheet Lab
  - 2.1 Cell Table
  - 2.2 Tool bar
- 3. Database Lab
  - 3.1 Relation Editor
  - 3.2 Contents of Relation
  - 3.3 List Of Relations
- 4. Data Structures Lab
  - 4.1 Console Window
  - 4.2 Memory Usage Grid
  - 4.3 Temporary Variables
  - 4.4 Data Object
  - 4.5 Datasets
  - 4.6 Data Structures Overview
- 5. Graphics Lab
  - 5.1 Variable Declarations
  - 5.2 Program Code
  - 5.3 Drawing Window
  - 5.4 Code Execution Controls



- 6. JavaScript Lab
  - 6.1 Code Window
  - 6.2 Command Bar
  - 6.3 Program Output Window
  - 6.4 Variables Window
  
- 7. Object-Oriented Lab
  - 7.1 Class Diagrams
  - 7.2 Code Viewer
  - 7.3 Object Diagrams
  - 7.4 Lab Controls Panel
  
- 8. Assembly Lab
  - 8.1 Assembly Code
  - 8.2 Memory
  - 8.3 Registers
  - 8.4 Commands
  - 8.5 Program Counter And Instruction Register
  - 8.6 Execution Controls
  - 8.7 Flags
  
- 9. Digital Logic Lab
  - 9.1 Circuit Design Tools
  - 9.2 Circuit Editor
  - 9.3 Component Tools
  
- Appendix A: Glossary
  
- Appendix B: System Requirements
  
- Appendix C: Credits



## Chapter 0: Introduction to Watson

This chapter provides an overview of the Watson project and its suite of labs. A brief history of the project, along with acknowledgments to the many people who have contributed their time and talents to Watson are also included in this chapter.

### 0.1 Overview

Watson is a collection of eight interactive laboratories designed to introduce freshmen to many of the key concepts that form the core of computer science.

The first two Watson labs provide exposure to two common applications: Spreadsheets and Databases. The Data Structures lab focuses on the organization of data at the program level. Next, fundamental programming concepts are introduced through a sequence of three labs: a simple Graphics programming lab, a JavaScript-based programming lab, and an object-oriented programming lab. Machine organization and the basics of computer hardware design are explored in the final two labs: Assembly Language and Digital Logic.

All Watson labs support a single common interface that minimizes keyboard input -- allowing students to concentrate on problem solving tasks, rather than spending their time trying to master the complexities of multiple user interfaces.

The Watson labs are written in Java, so they can run in most modern web browsers (requires installation of Sun's Java Runtime Environment -- available free of charge from <http://www.java.com>). The labs can also be run from our auto-loading CD on Windows, Linux, and Mac platforms.

Extensive documentation is provided to support the Watson labs, including this user's manual and a complete set of interactive tutorials. For more information, or to get your copy of the labs, visit <http://watson.latech.edu>



## 0.2 The Watson Labs

The current incarnation of Watson includes eight labs. While each of these labs incorporates a common user interface, the individual labs address subject matter ranging from common applications to digital logic.

A brief description of each lab follows:

- Spreadsheet – Manipulate numeric data and equations to perform calculations and solve "what if" type problems.
- Database – Retrieve information from a relational database using the select, project, and join operators.
- Data Structures – Learn about the behavior of common data structures such as stacks, queues, lists, and trees.
- Graphics Programming – Create graphical images and animations using a simple graphics programming language.
- JavaScript Programming – Expand your knowledge of programming with this JavaScript-based lab.
- Object-Oriented Programming – Explore the world of objects, classes, methods, inheritance, and message passing.
- Assembly Language Programming – Write programs at the lowest levels and understand their operation in terms of 1's and 0's.
- Digital Logic – Build digital circuits capable of functions such as comparing and adding binary numbers.

## 0.3 History of Watson

Watson development has taken place in three distinct phases: the first from 1993 through 1995, the second from 1996 to 1998, and the third phase



from 2002 to 2004.

## Phase I (1993-1995)

The first phase of Watson involved the conceptualization, design, and initial implementation of nine lab environments. During this time, Watson was support by a \$100K grant from the National Science Foundation.

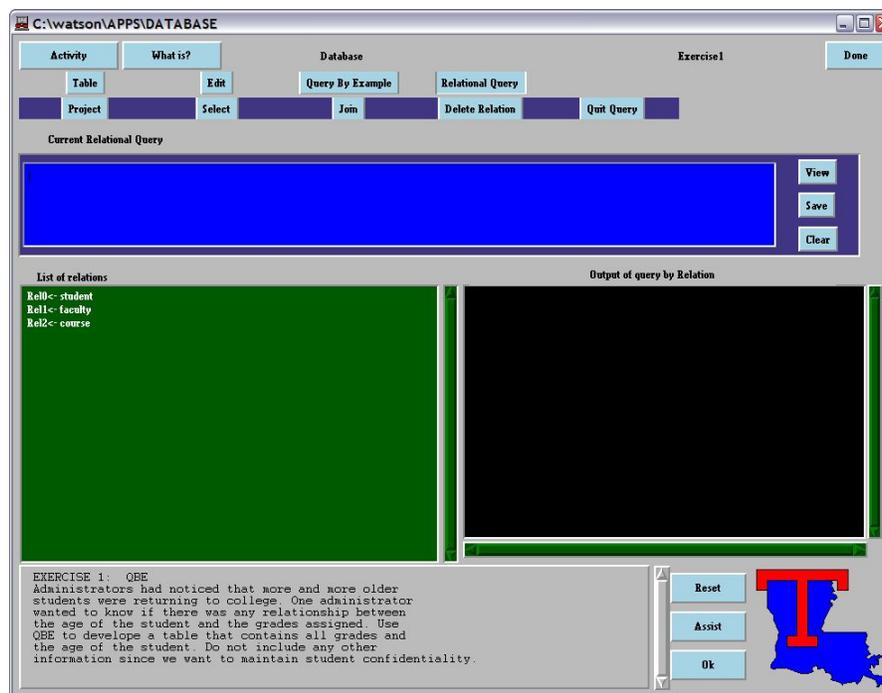


Figure 0.1: A Watson Lab: Phase I (1993 – 1995)

Watson's core concepts of a unified graphical user interface, cross platform compatibility, breadth of concept coverage through the use of a suite of independent labs, and depth of coverage within individual labs by careful attention to the pedagogical content of individual lab activities were established early on.

The original suite of labs included: a simple spreadsheet, a relational database, a data structures lab, a graphics-based programming



environment, an imperative programming environment, a functional programming environment, a Finite State Automata lab, an assembly language and machine architecture lab, and a digital logic lab. These labs were implemented in C and used the SUIT (Simple User Interface Toolkit) libraries for cross platform compatibility.

While these prototypes established the viability of the Watson approach to introductory computer science education, they were not robust enough to be used outside of a closed laboratory environment with extensive student assistance provided by instructors and lab monitors. In fact, the Project Director's candid assessment of the original implementation of Watson would be "buggy as hell". This should not be surprising, given the goals of the project, its groundbreaking nature, and limited funding. Additionally the interface provided by the labs were not very attractive, nor intuitive.

## **Phase II (1996-1998)**

By the mid-1990's it became apparent that Watson needed a total rewrite in order to become a viable teaching tool. After several years of continuous classroom use, resulting in feedback from hundreds of students, the Project Director had a very clear idea of what worked and what didn't. Furthermore, the emergence of Java and explosion of web technologies offered the hope of a stable cross-platform environment for Watson.

Given that NSF funding had expired at this point, the Project Director, set about the bold task of designing and implementing Watson in Java. Since very few Java-based tools or libraries existed at the time (combined with the fact that Java itself was rapidly evolving), the Project Director began by building up a "Watson Java Toolkit" of common components, such as choice selectors, numeric entry pads, and even low-level primitives such as "Watson Buttons". Once these were in place he redesigned the graphics lab based on student feedback from the Phase I version. He then coded the lab and thoroughly debugged it. The graphics lab thus served as a "template" for all Watson Phase II labs.

Given this template students were offered independent study classes in which they would "re-implement" a Phase I lab using the toolkit and graphics



lab as a guide. Some of the labs such as spreadsheet, database, assembly, and digital logic incorporated few pedagogical changes in their port to Java. Other labs, such as data structures, graphics, and imperative programming were completely rethought. Still other labs, such as the functional programming lab and finite state automata lab, were abandoned.

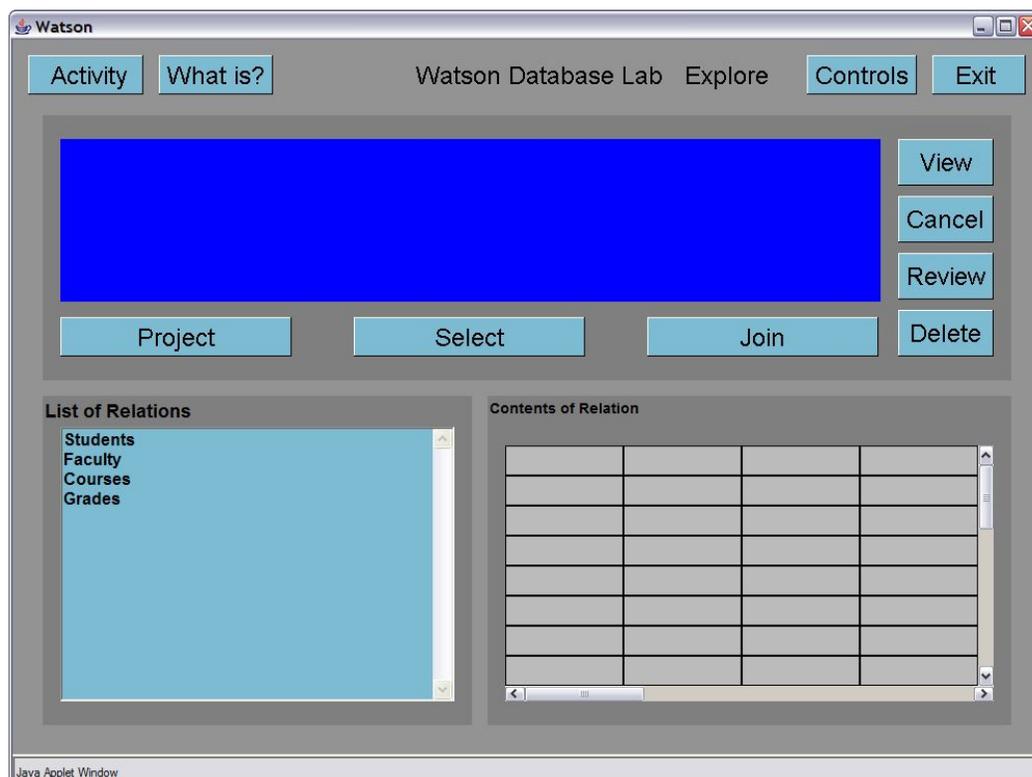


Figure 0.2: A Watson Lab: Phase II

The result of Phase II was a collection of stable, albeit somewhat plain-looking labs, that could be accessed via web browsers. These labs were used successfully in Louisiana Tech's CSC 100 courses for a number of years.

By 1998, the project had reached the point where national dissemination appeared imminent. In addition to Watson, its companion textbook (drafts of which have been student tested since the mid-1990's) was nearing completion. Despite Watson's promise, the Project Director had to



reluctantly shelve Watson for several years. The continued lack of external funding for Watson, combined with allure of the dot com boom, enticed the Project Director into a two-year sabbatical in industry where we became the Chief Technology Officer for OneNetNow.com. After OneNetNow was sold to Earthlink in 2001, the Project Director elected to return to academia rather than accept a permanent position with EarthLink.

### **Phase III (2002-2004)**

Upon return to academia in the Fall of 2001, four things became apparent (1) Since the labs had been written in the very early days of Java, they were no longer stable on modern Java environments, (2) The look of the labs had become outdated since they were not built with Swing components, (3) The pedagogical content embodied in the Imperative lab was no longer as appropriate as it had been in the mid-1990's, and (4) The fact that Watson contained zero coverage of object-oriented programming concepts had become a serious omission.

Additionally, the uncompleted textbook could not simply be finished, as sections of it needed to be rewritten to reflect the changes in the field from the mid-late 1990's to the early 2000's. Furthermore, since the textbook and the labs are interrelated, any changes to the labs required similar changes to the text.

The Project Director began restarting Watson development in 2002. He recruited a graduate student, Cliff Lemoine, to help maintain the Phase II labs and to design and develop an object-oriented programming lab. In the Summer of 2002, he made some inroads into updating the textbook. During the Spring of 2003, he applied to the National Science foundation for funding to fully launch Phase III. Though the proposal reviews were generally strong, the NSF ultimately declined the application(Oh well... Watson is not so easily killed).

So, in the tradition of Watson, the Project Director embarked on Phase III. He recruited a bright team of undergraduates and with nothing but Independent Study credit in his pocket and pushed forward with Phase III.



The Phase III team was able to build on a Swing-based Watson platform implemented in 1999 by a former graduate student, Don Garrett, together with a nearly complete implementation of the graphics lab Don wrote. The Phase III team made rapid progress re-implementing several of the labs. From December 2003 through February 2004, database, assembly, and digital logic were ported. Data structures, and JavaScript were begun. From March 2004 to May 2004, the spreadsheet lab was successfully ported. During this time, the data structures rewrite was completed, and the initial implementation of the object-oriented lab was finished.

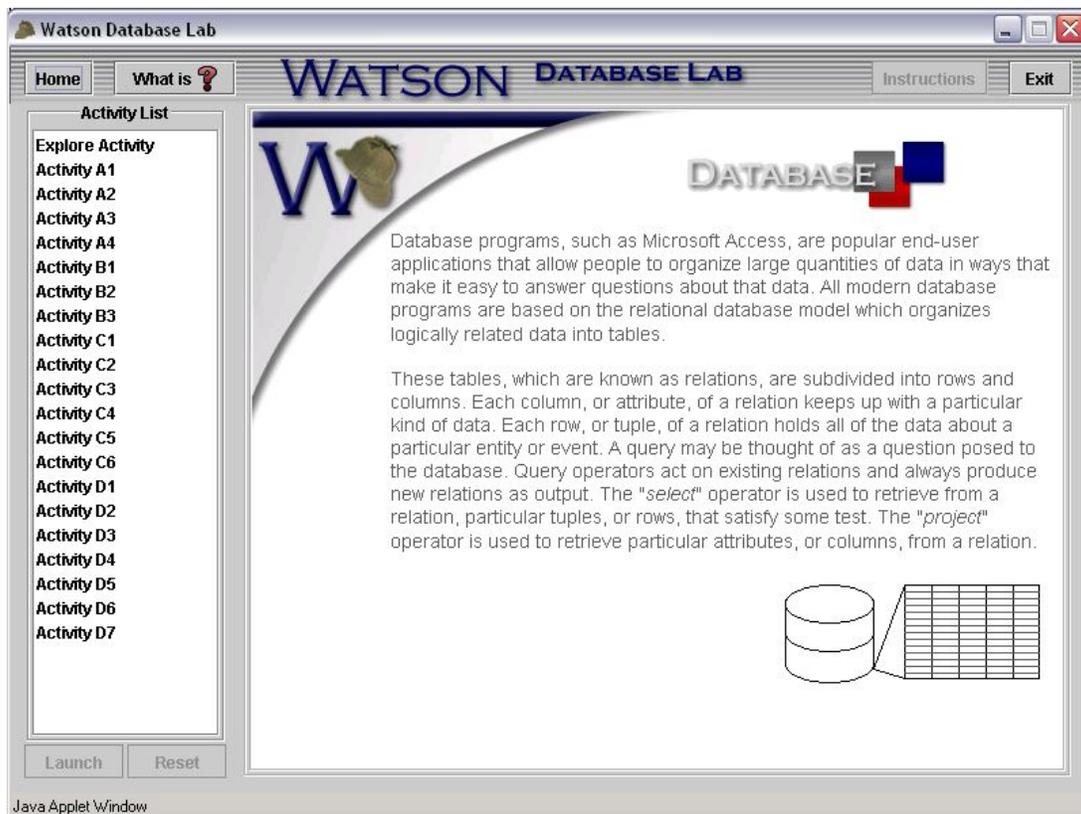


Figure 0.3: A Watson Lab: Phase III

As of May 2004 the only lab that is incomplete is the JavaScript lab. Unfortunately, from a pedagogical standpoint, this is one of the most important Watson labs.



During the spring of 2004, the Project Director had the good fortune of teaching CSC 404 "Senior Capstone". In the spirit of "use every resource at your disposal", the Project Director selected Watson as the topic for Senior Capstone. In addition to providing desperately needed manpower for the Watson project, this choice enabled the class to gain near-real-world experience with a large project on the back end of the software design process. The CSC 404 class devoted tremendous effort to testing, documentation, and code maintenance. They tested the Phase III software with actual students and incorporated UI changes based on the student's feedback. Furthermore the class refined the graphical "look" of the labs and incorporated this look into the web and printed documentation they developed.

As of May 2004, the Project Director is once again in the process of applying to the National Science Foundation for funds. These funds are needed to complete development of the JavaScript lab, provide critically needed support for ongoing software maintenance, and to help finish the companion textbook.

## 0.4 Acknowledgments

The number of faculty members and students who have contributed to the Watson Project over the years is truly amazing. Below, I have attempted to provide a comprehensive list of those who have made Watson possible.

The first person I would like to thank is Barry Kurtz. Watson would not exist without him. Barry came up with the original idea for a suite of labs to support a breadth-first introduction to computing – and then proceeded to convince me of both the value and feasibility of this approach. Barry was also the driving force behind convincing the National Science Foundation to initially fund the project.

During the first phase of the Watson project, Don Garrett and Alex Ramos were key members of the team. Don and Alex implemented the library of shared code that was central to the original C / SUIT lab prototypes. Other individuals who contributed to the development of this phase of Watson



include: Pat Bronson, Lee Falta, Michael Gaudet, Dr. Gary Klein, Josh Kleinpeter, Dr. Barry Kurtz, Jeff Matocha, Unmesh S. Mayekar, Dr. Mike O'Neal, Alex Ramos, Dr. Louis Roemer, Vinay Shivaiah, Sameer Singh, Charlie Stear, and Mark Williamson.

During the second phase of Watson development, Josh Kleinpeter was my "right hand man". He was the first person to assist me with use of the labs in the classroom setting. Plus, he did an excellent job keeping the C / SUIT prototypes running well beyond the point at which they should have been retired.

The developers for the second phase of Watson include:

- Chris Dickenson, Gene Rogers, Chris Plock, and Dave Muse, Imperative
- Mike O'Neal, Graphics and Watson-Java Toolkit
- Chris Plock, Josh Kleinpeter, and Dave Muse, Database
- Heather Richards, Digital Logic
- Jeff Rugg, Spreadsheets and Data Structures
- Ben VanWagner, Assembly
- Kasoum Gioul, portions of the Spreadsheet code
- Jon J. Walker, mouse-oriented editor in Graphics

Jayson Calloway, Cai Hong, and Byren Vaughn helped prototype a number of these Java-based labs.

My "right hand man" for the third phase of Watson has been Cliff Lemoine. Cliff was invaluable in helping me to restart Watson development after a three-year lull resulting from my spending time in industry. Cliff helped tutor students in my CSC 100 classes, maintained the Phase II labs, and developed the object-oriented lab for Phase III.



A special heart felt THANK YOU also goes out to my good friend and former student Don Garrett who, in addition to being a primary developer in Phase I of Watson, returned to Louisiana Tech to pursue a Master's degree and in the process developed the Java Swing framework for Watson Phase III. Don's years in industry developing object-oriented Java-based applications provided him with the background to lay a solid foundation for the current set of labs. Without Don's efforts, Watson Phase III would not exist.

The primary developers for Watson Phase III include:

- Brandon George, Spreadsheet
- Jim Patterson, Database
- Chris Martin, Data Structures
- Don Garrett, Graphics and Overall Watson Framework
- James Phillips, JavaScript
- Cliff Lemoine, Object-Oriented
- Clay Zuvich, Assembly
- Anthony Stonaker, Digital Logic

In addition to these developers, I'd like to thank the entire spring 2004 CSC 404 "Senior Capstone" class. This class did a spectacular job improving the look and feel of the Phase III labs, including developing all of the graphics, the user's manual, brochure, CD case, flash-based web site, flash tutorials, and multi-platform (Windows, Mac, Linux) Watson CD. Additional efforts included: testing, bug tracking, bug fixes, and numerous improvements to the UI based on feedback collected from actual users. While all of those who worked so hard on this project deserve a BIG thank you, special recognition should be give to Jim Patterson (the Project Director), and Chris Martin and Louis Landry (who were jointly responsible for the "look" of Watson Phase III).

To each and every one of you, "THANK YOU!"

Dr. Mike O'Neal  
Director, Project Watson

## Chapter 1: General Watson Features

Every Watson Lab is equipped with an activity page. Also, each lab has a common tool bar, referred to as the Watson Title Bar.

### 1.1 Activity Page

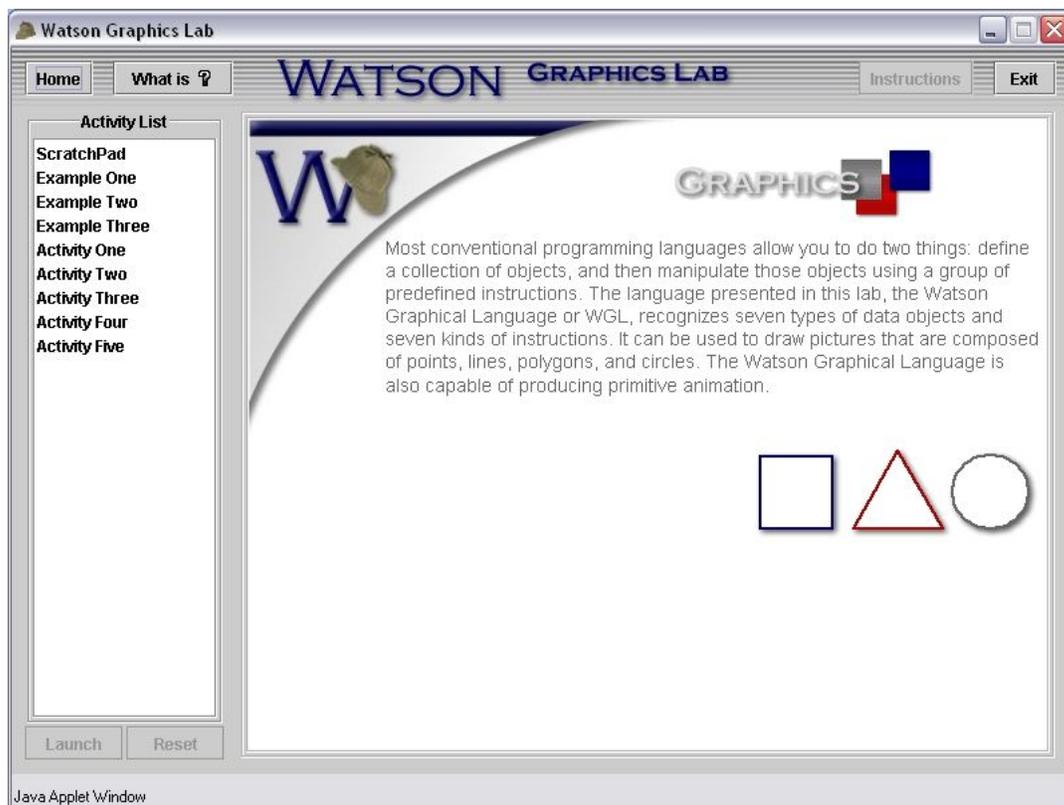


Figure 1.1: Activity page.

This is the first page a user will see when they load any Watson lab. From this screen, the user can choose any activity and click "Launch" to load that particular activity, including the "Scratch Pad". A summary of the activity is displayed on the right.

## 1.2 Watson Title Bar



Figure 1.2: Watson Title Bar.

This tool bar allows the user to choose a new activity, discover the function of a particular area of the lab, reveal the activity instructions, or exit the lab. This is the first page a user will see when a Watson lab is first started.

- **Home Button**  
This button returns the user to the activities screen described above.
- **What is? Button**  
This button can be used to discover the function of each of the components of the lab. Click this button followed by another button you want a description of.
- **Instructions Button**  
This button will show the Activity Description Pane if it is hidden. See the next section for a description of the Activity Description Pane.
- **Exit Button**  
This button exits the current lab.

## 1.3 Activity Description Pane

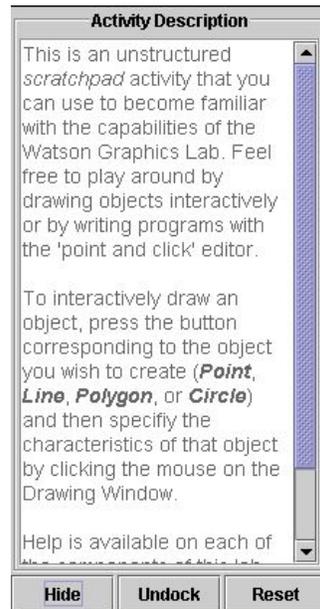


Figure 1.2: Docked Activity Description Pane.

This pane shows you the activity instructions. It also allows the user to hide the Activity Description Pane, view the Activity Description in a separate window, and reset the current activity.

- **Hide Button**  
This button will remove the Activity Description Pane from the screen.
- **Dock Button**  
This button will bring the Activity Description Pane to its original docked state.
- **Reset Button**  
This button will reset the current activity.

- **Undock Button**

This button will detach the Activity Description Pane from the main Watson window. The Undock Button will change to the Dock Button.

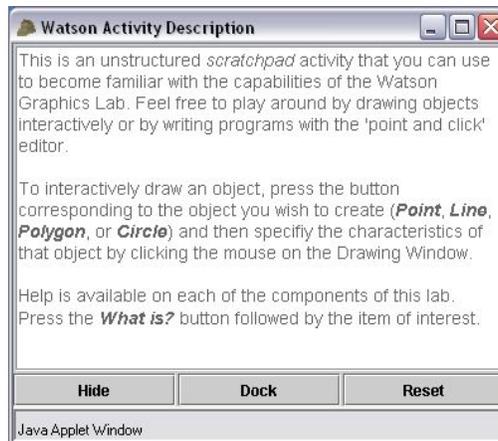


Figure 1.3: Undocked Activity Description Pane.

## Chapter 2: Spreadsheet Lab

### Concept Overview

Spreadsheets are one of the most popular end user applications in existence today. They are general purpose modeling tools that allow end-users to represent a problem as a collection of data items and the formulas that express the relationships between those items. Spreadsheets consist of a grid of horizontal and vertical cells referenced by a column letter and row number.

The Watson Spreadsheet laboratory provides a simple spreadsheet that you can use to become familiar with most of the basic concepts that are common to these types of applications.

A spreadsheet is a grid that organizes data in a matrix of rows and columns and processes many numbers in a small amount of time. Spreadsheets make it easy to display information, and people can insert formulas to work with data. Each intersection of a row and column forms a cell which holds one piece of information. The cells are linked together by formulas. If you make a change in one cell, the spreadsheet uses the power of the computer to automatically recalculate and alter every other cell linked to that original cell. For business or for home, an electronic spreadsheet is an excellent tool for manipulating numbers. It executes tedious calculations without error and much faster than can be done by hand.


## Spreadsheet Lab Layout

The Watson Spreadsheet Lab consists of three main areas. The first area is the Watson Title Bar, which is common to all the Watson Labs. The next area is the Cell Table, which holds the cell contents for the activities. To the right is the Tool Bar area that displays the options available for cell manipulation.

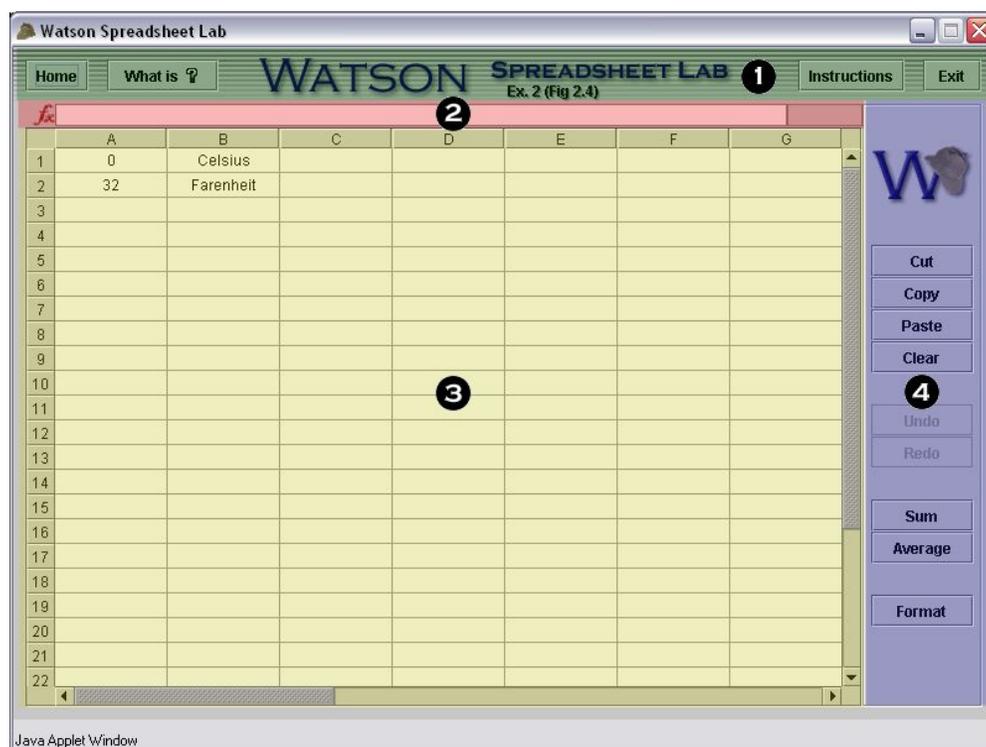


Figure 2.1: The Interface of the Watson Spreadsheet Lab.

- 1 Watson Title Bar**  
This is a standard feature on all Watson Labs.
- 2 Equation Editor**  
This allows equations to be entered into the table.
- 3 Cell Table**  
All data is entered and displayed in this section.

#### 4 Tool Bar

This area has buttons to do certain tasks in the spreadsheet.

### 2.1 Equation Editor



Figure 2.2: The equation editor.

The equation editor displays the information being entered into the current cell. A formula can be entered that references other cells by beginning input with an equal sign(=). The formula can consist of any combination of addition, subtraction, multiplication, division, summation, and average. Normal mathematical rules for order of operations apply with parenthesis. For the functions to work correctly, use the format OP(Begin:End). For example, SUM(A1:A7).

Remember cells can be referenced absolutely and relatively. Absolute references will not change when an equation is moved, but relative references are based upon the location of the cell that the equation is in. To make a reference absolute, simply put a \$ in front of it. Note that a column can be absolute without the row being absolute and vice-versa.

### 2.2 Cell Table

	A	B	C	
1	0	Grades	50	
2	32.0	Fahrenheit	100	
3			75	
4				
5		Average	75.0	
6				
7				
8				
9				

Figure 2.2: The cell table.

This area contains cells labeled with numbers denoting rows and letters denoting columns. To select a cell, simply click on the cell. Dragging the mouse will select multiple cells. The selected cells are outlined in red.

## 2.3 Tool Bar



Figure 2.3: The tool bar.

- **Cut Button**

This button removes the selection from the active document and places it on the clipboard.

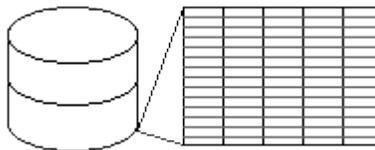
- **Copy Button**  
This button copies the selection to the clipboard.
- **Paste Button**  
This button inserts the contents of the clipboard at the insertion point, replacing any selection.
- **Clear Button**  
This button removes all cell contents.
- **Undo Button**  
This button reverses the last command or action performed.
- **Redo Button**  
This button reverses the Undo command.
- **Sum Button**  
This button adds the contents of selected cells. Also typing SUM into the equation editor will sum the specified cells.
- **Average Button**  
This button averages the contents of a selected set of cells. Also, typing AVG into the equation editor will average the specified cells.
- **Format Button**  
This button adjusts the number of decimal places that are displayed in a cell containing numeric data or the result of a formula.

## Chapter 3: Database Lab

### Concept Overview

Database programs, such as Microsoft Access, are popular end-user applications that allow people to organize large quantities of data in ways that make it easy to answer questions about that data. All modern database programs are based on the relational database model which organizes logically related data into tables.

These tables, which are known as relations, are subdivided into rows and columns. Each column, or attribute, of a relation keeps up with a particular kind of data. Each row, or tuple, of a relation holds all of the data about a particular entity or event. A query may be thought of as a question posed to the database. Query operators act on existing relations and always produce new relations as output. The "*select*" operator is used to retrieve data from a relation, particular tuples, or rows, that satisfy some test. The "*project*" operator is used to retrieve particular attributes, or columns, from a relation.



## Database Lab Layout:

The Watson Database Lab consists of four main areas. The first area is the Watson Title Bar, which is common to all the Watson Labs. The next area is the Relation Editor, which holds the current relation. Below that is the Contents of Relations area that displays the contents. Finally there is the List of Relations which is a list of relations available to the user.



Figure 3.1: The interface of the Watson Database Lab.

- 1 Watson Title Bar**  
This is a standard feature on all Watson Labs.
- 2 Relation Editor**  
Area used for building new queries to generate intermediate relations.

- 3 **Contents of Relation**  
This area contains the data of the current working relation.
- 4 **List of Relations**  
This area contains relations currently usable by the editor.

### 3.1 Relation Editor



Figure 3.2: The relation editor.

This area of the screen is used to construct relational queries. You are able to edit a query by clicking one of the three buttons at the bottom Project, Select, or Join. A generic query will appear in the editor pane depending on whether you clicked Project, Select, or Join you clicked.

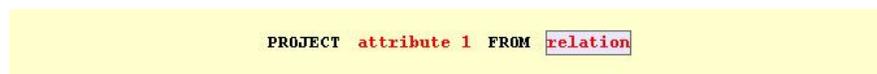


Figure 3.3: The editor pane with a generic Project query

Click on the red text in order to assign a specific value to that portion of the query. The first field that must be completed is the RELATION field. Expressions that have a variable number of fields, such as Project, will automatically expand as they are completed to provide as many fields as necessary to express the query. Each time that you click text that has been highlighted in red, a dialog box will pop up, prompting the user to choose what to change that text to.

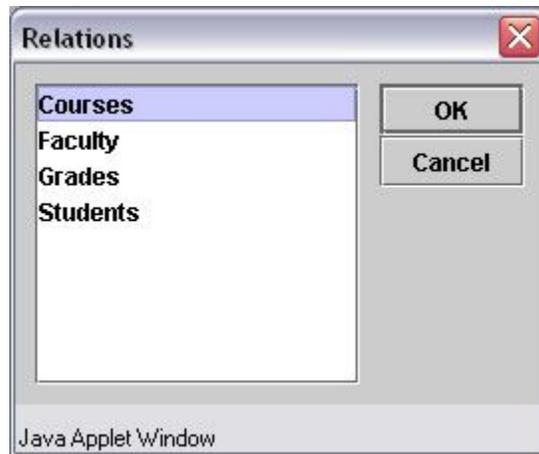


Figure 3.4: An example of a dialog that pops up upon clicking the red relation text

Select your choice of data with which you want to replace the red text, and click OK. This will update the generic query with the text that you have selected.

```
PROJECT CREDITS , attribute 2 FROM Courses
```

Figure 3.5: An example project query

Once you have updated all the necessary red keywords to match the relation you want to create, click OK to save your relation to the 'List of Relations' area.



Figure 3.6: The example Project query in the List Of Relations

## Relation Editor Components

- **Editor Pane**  
This area with a yellow background is used to view and modify queries.
- **Project Button**  
This button will allow you to create a new relation (table) by Projecting attributes (columns) from an existing relation.
- **Select Button**  
This button will allow you to create a new relation (table) by Selecting tuples (rows) from an existing relation where entries in a particular column meet a specified condition
- **Join Button**  
This button will allow you to create a new relation (table) by Joining two existing relations over a common field.
- **OK Button**  
This button will display the contents of the relation that is defined by the current query. It will also give the relation a name and save it in the 'List of Relations' so that it may be reviewed at a later time.
- **Clear Button**  
This button can be used to cancel your work on the current query. It clears only the current query. None of the relations in the 'List of Relations' is affected. This button could come in handy if you discover that you have made a mistake expressing the current query and wish to start it over.

### 3.2 Contents of Relation

**Contents of Relation: Students**

SNAME	AGE	MAJOR	ID	SEX	ADDRESS	CITY	STATE		
Anderson B.	18	CS	55555501	M	101 Rocket Way	Atlantis	CA		
Barnes D.	17	MATH	55555502	F	1402 Elf Lane	Ruston	LA		
Bronson P.	26	MATH	55555503	M	1 Web Master	Ruston	LA		
Brooks D.	18	CS	55555504	F	900 Baird Street	Dallas	TX		
Garrett D.	20	PSY	55555505	M	BGB Consulting	Dallas	TX		
Howard M.	21	CS	55555506	M	5 Scarborough	Dallas	TX		
Huey B.	18	CS	55555507	F	1 Historic Place	Jackson	MS		
Kleinpeter J.	24	CS	55555508	M	69 Watson Lane	Ruston	LA		
Kizer D.	19	CS	55555509	F	40 Animas Way	Hammond	LA		

Figure 3.3: The contents of relation portion, showing the students relation.

This table is used to display the contents of a relation. To view an existing relation, even those relations which you have created, select it from the 'List of Relations'. To view the results of a query, press the OK button after completing construction of the query.

### 3.3 List of Relations



Figure 3.4: The list of relations area, showing the four default relations.

This is a list of currently available relations. New relations can be added by constructing queries and clicking OK in the Relation Editor.

### **List of Relations Components:**

- **Delete Button**

This button can be used to delete a user-defined relation from the 'List of Relations'. Built-in relations (such as: Students, Faculty, Courses, and Grades) may not be deleted.

## Chapter 4: Data Structures Lab

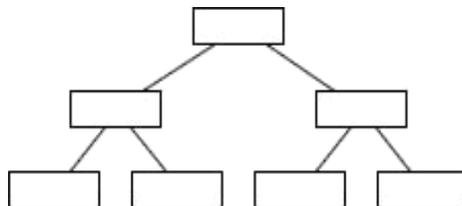
### Concept Overview

Data structures are a fundamental part of programming algorithms. Even though data structures are not considered part of an algorithm, many algorithms use standard data structures to produce their intended results. There are also algorithms that will modify these standard data structures to get efficient performance on a specific problem. In short a good understanding of basic data structures is essential for programming more advanced algorithms to solve more advanced problems in computer science.

The Watson Data Structures Lab is an introduction to the operation of basic data structures. The data structures introduced here are:

- stack
- queue
- array
- list
- tree

These are fundamental data structures in programming. The lab will demonstrate what they are and how they operate in a program.



## Data Structures Lab Layout

The Watson Data Structures Lab is an introduction to the operation of basic data structures. Due to differences in data structures, the layout of many activities are different. The main areas are console window, memory grid, temporary variable, data object, and the dataset.

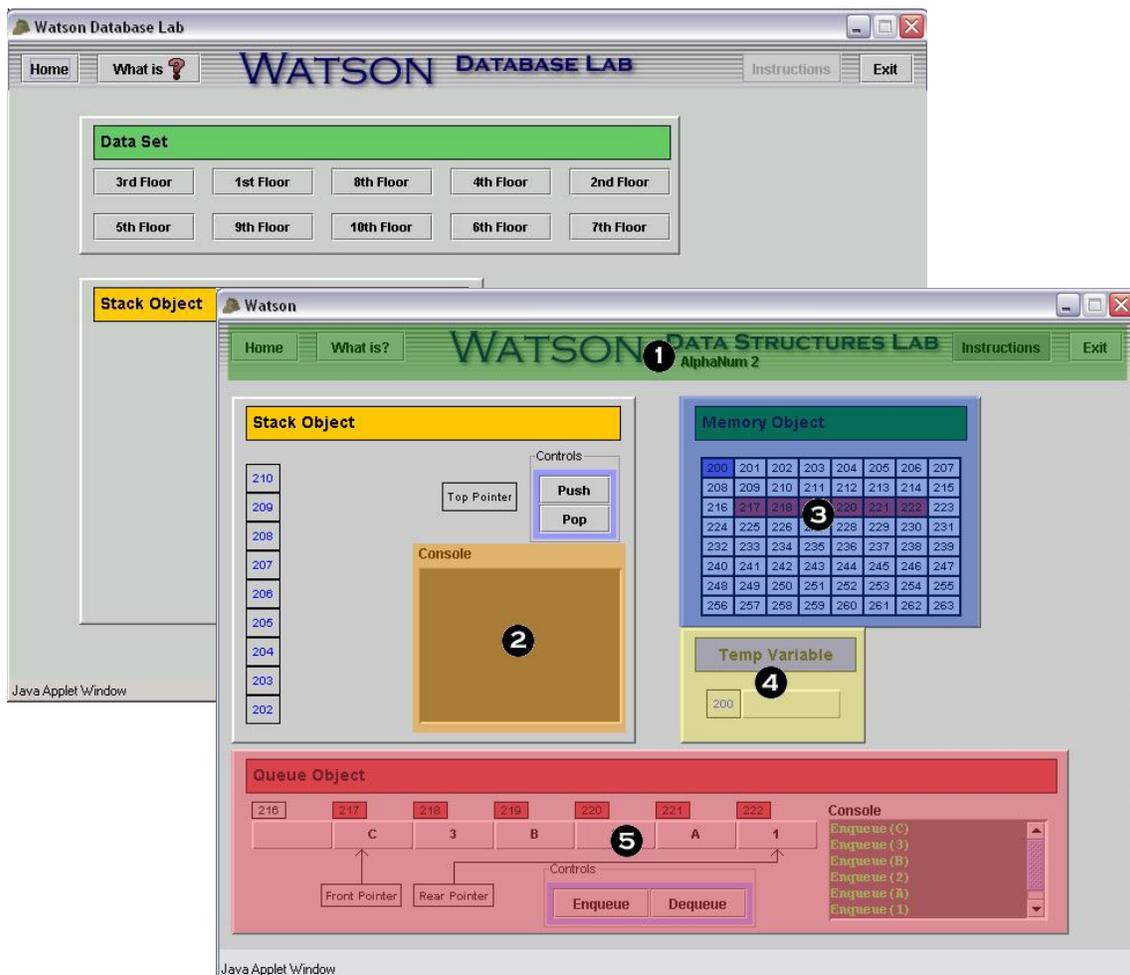


Figure 4.1: The main areas of the Data Structures Lab

### 1 Watson Title Bar

This is a standard feature on all Watson Labs.

- ② **Console window**  
This window shows the previous commands issued.
- ③ **Memory usage grid**  
The grid represents the memory in a computer.
- ④ **Temporary variables**  
This displays what is currently in the temporary variable.
- ⑤ **Data Object**  
Representation of data structure is displayed here.
- ⑥ **Dataset**  
All available input data is displayed here.

## 4.1 Console Window

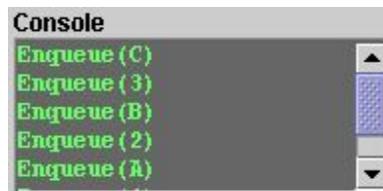
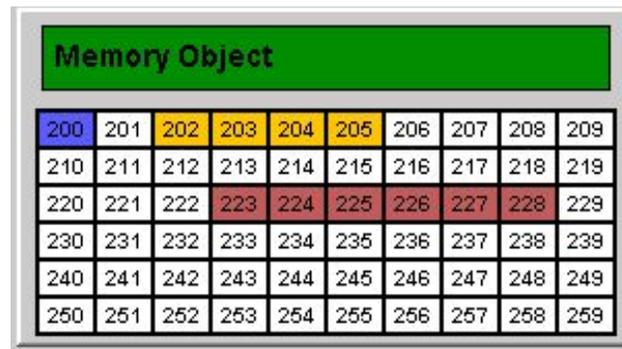


Figure 4.2: The console Window.

The console window displays a history of operations. It displays the order in which data was pushed to or popped from a stack or enqueued in a queue. The console displays the operation and the data, such as **Enqueue(C)**.

## 4.2 Memory Usage Grid



A grid titled "Memory Object" showing memory addresses from 200 to 259. The grid is 10 columns wide and 6 rows high. The first row (200-209) has columns 200, 202, 203, 204, and 205 highlighted in yellow. The second row (210-219) is white. The third row (220-229) has columns 223, 224, 225, 226, 227, and 228 highlighted in red. The fourth row (230-239) is white. The fifth row (240-249) is white. The sixth row (250-259) is white.

Memory Object									
200	201	202	203	204	205	206	207	208	209
210	211	212	213	214	215	216	217	218	219
220	221	222	223	224	225	226	227	228	229
230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249
250	251	252	253	254	255	256	257	258	259

Figure 4.3: The memory usage grid.

This area displays what memory data structures use. The different data structures have different colors. Also, different data structures handle memory in different ways. The temporary variable will be locked in one location. The stack will grow and shrink but the base of the stack will always stay where it was allocated in memory. The queue will move through memory like a snake.

## 4.3 Temporary Variables



Figure 4.4: The temporary variable display.

The temporary variable is a storage location for temporary data. Imagine you have two computers on two desks. Now if you want to swap the computers, then you will need a temporary third location to store one computer while the other is moved. Then the computer in the temporary location can be moved to the empty desk. This is exactly how the temporary variable works, but instead of computers, it stores data.

## 4.4 Data Object



Figure 4.5: The data object.

The data object displays a graphical representation of a particular data structure. In this example, it displays a queue. The elements of the queue are labeled in the boxes. There are also front and rear pointers that point to the front and rear of the queue. The console window is in this area and shows the processes which were applied to the queue.

## 4.5 Datasets



Figure 4.6: The dataset display.

The data set is where all possible entries are located. These data items can be selected by clicking on them. Usually this will move the data item to the temporary variable location to be placed into a data structure.

## 4.6 Data Structures Overview

The different data structures in this lab are the array, the stack, the queue, the linked list, and the tree. This will be a brief overview of how these data

structures are organized, set up, and work.

An array may be thought of as a collection of variables, placed side by side, that can be accessed by specifying the position number (or index) of the variables. For example, we can speak of the first element of the array, `Array[1]`, the third element, `Array[3]`, or the tenth element, `Array[10]`. Elements in an array are accessed by an index number. The index number is an offset from the beginning of the array. Basically, an array is a large number of regular variables in a line that are accessed by the index number.

A stack is like a stack of plates. Data can only be added or removed from the top of the stack. When data is added to the top of the stack, we say it is being pushed on the stack. When data is removed from the top of the stack, we say it is being popped off the stack. This scheme is called Last In First Out(LIFO).

A queue is similar to a waiting line. The first data item in is the first data item out. Enqueuing involves adding data to the back of the queue. Dequeuing involves removing data from the front of the queue. This scheme is called First In First Out(FIFO).

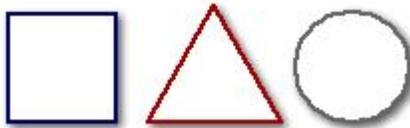
A linked list is a list of data items. A data item in a linked list will point to the next item in the list. This is useful because data items can be added to a linked list as they are needed and removed when they are not needed. As opposed to an array, which must allocate all its memory when it is declared, a linked list is dynamic and can grow and shrink as required. The disadvantage of a linked list is accessing elements in the list takes longer than accessing elements in an array.

A tree is a collection of nodes that follow certain rules in its formation. The root node of a tree is at the top and does not have a parent. The root node points to other nodes which are called its children. Their parent is the root node. A node may point to other nodes, but the node it points to must not have another node pointing to it, thus each node has exactly one parent except the root node. Trees are very useful in organized searching techniques.

## Chapter 5: Graphics Lab

### Graphics Lab Concept

Most conventional programming languages allow you to do two things: define a collection of objects, and then manipulate those objects using a group of predefined instructions. The language presented in this lab, the Watson Graphical Language or WGL, recognizes seven types of data objects and seven kinds of instructions. It can be used to draw pictures that are composed of points, lines, polygons, and circles. The Watson Graphical Language is also capable of producing primitive animation.



## Graphics Lab Layout

The Watson Graphics Lab consists of five main areas. The first area is the Watson Title Bar, which is common to all the Watson Labs. The next area is the Variable Declarations, which is used to define and display the variables referenced by the program statements. Below that is the Program Code area, which allows for the creation, display, and editing of program statements. Next there is the Drawing Window which displays the graphics defined by the Program Code. Below that are the Program Execution Controls which allow the user to either run a program in its entirety or step through it line by line.



Figure 5.1: The interface of the Watson Graphics Lab.

### 1 Watson Title Bar

This is a standard feature on all Watson Labs.

**2 Variable Declarations**

This area is used for the creation of variables.

**3 Program Code**

This area contains the statements used in drawing the graphics; in addition, this is where variables are assigned values.

**4 Drawing Window**

This area contains the drawing defined by the code statements.

**5 Code Execution Controls**

This area contains the controls that run the program or step through it.

## 5.1 Variable Declarations

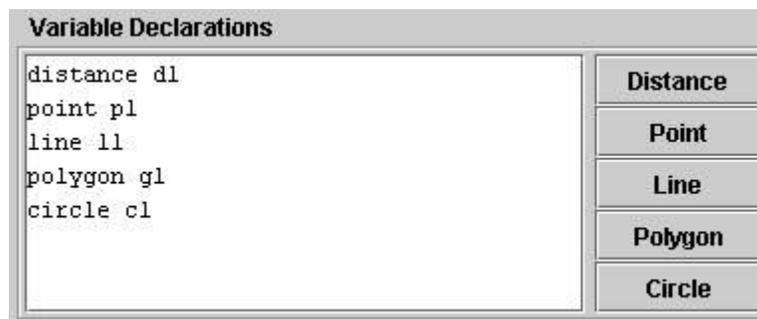


Figure 5.2: Variable Declarations.

- **Declarations Pane**

This area is used in conjunction with the Data Type buttons to develop Watson Graphical Language programs. It displays all the variables being used in the program.

- **Distance Button**

This button defines a variable of type 'Distance'. Note that unlike the other types, a 'Distance' cannot be defined interactively in the Drawing Window, since distances can not be drawn.

- **Point Button**

This button defines a variable of type 'Point'. After pressing this button, you can interactively draw a point by clicking in the Drawing Window at the position you would like the point to be located.

- **Line Button**

This button defines a variable of type 'Line'. After pressing this button, you can interactively draw a line by clicking in the Drawing Window; once at the location where you would like the line to begin and again where you would like it to end.

- **Polygon Button**

This button defines a variable of type 'Polygon'. Polygons are closed multi-sided figures that may be defined by a sequence of points, where the first and last point in the sequence must be the same. Polygons can be drawn interactively in this lab by clicking on the Drawing Window, once for each point in the polygon. To close the polygon, click on the initial point a second time.

- **Circle Button**

This button defines a variable of type 'Circle'. Circles consist of a center point and radius distance. They can be drawn interactively by first clicking on the center point and then where the rim of the circle should be drawn.

## 5.2 Program Code

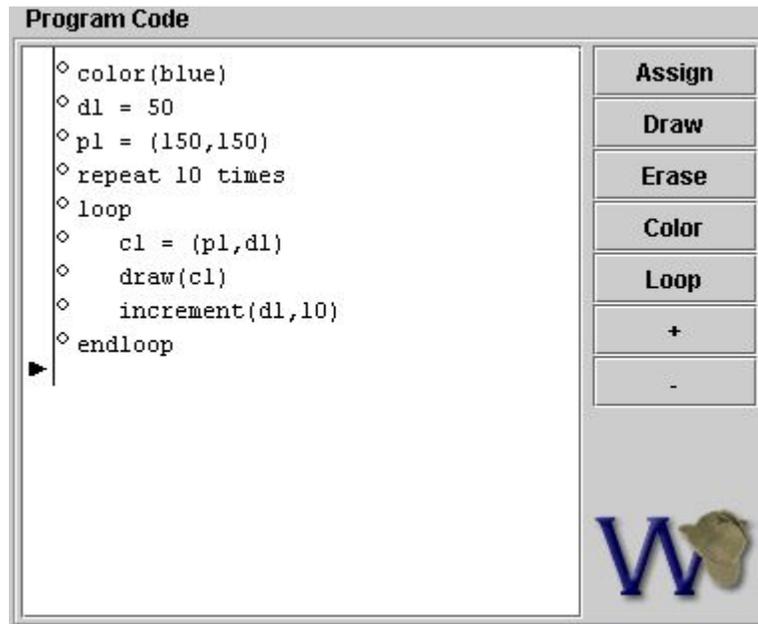


Figure 5.3: The Program Code Window.

- **Program Code Pane**

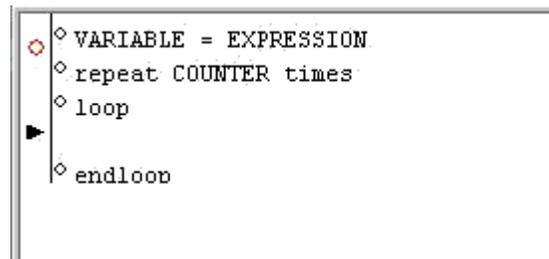
This area with a white background is used to view/edit statements and assign values to variables. See the code example below.

- **Remove Statements**

To remove statements from the Code Window, click once on the small circle located to the left of the statement or statement block. The code to be deleted will highlight red.

- **Insert Statements**

To insert statements into the Code Window above the current cursor position (small black arrow), click to the left of the vertical line at the position where the new code is to be placed. In the figure below, clicking on the position of the small red circle will place the cursor at that position.



```
◇ VARIABLE = EXPRESSION
◇ repeat COUNTER times
◇ loop
◇ endloop
```

Figure 5.4: Insert Statements

- **Assign Button**

This button inserts into the program a statement of the form:

**VARIABLE := EXPRESSION**

This is used to assign a value to a particular variable. Once a variable has been created by clicking the appropriate button, click on VARIABLE, choose the variable, then click OK to continue. Depending on which type of variable being used, there will be different fields that require values. These consist of X, Y, and RADIUS values. Once one of these values is clicked on, either a constant or a distance can be assigned for the value, constants being defined using the keypad. The distance is useful for loops as it can be incremented.

- **Draw Button**

This button inserts into the program a statement of the form:

**draw (OBJECT)**

This is used to display an object of type Point, Line, Circle, or Polygon in the Drawing Window. The object must be fully defined before it can be drawn. Once OBJECT is clicked on, a defined object can be selected, followed by OK to continue.

- **Erase Button**

This button inserts into the program a statement of the form:

**erase (OBJECT)**

This is used to erase an object of type Point, Line, Circle, or Polygon from the Drawing Window. Once OBJECT is clicked on, a defined object can be selected, followed by OK to continue.

- **Color Button**

This button inserts into the program a statement of the form:

**color ( COLOR\_NAME )**

This is used to specify a drawing color. All subsequent draw commands will use the specified color unless it is changed by another color command. Note that the default drawing color is red. Once COLOR\_NAME is clicked on, one of seven colors can be chosen, followed by OK to continue.

- **Loop Button**

This button inserts a loop structure into the program. Loops are used to repeat a group of statements a fixed number of times. They are useful for drawing a series of similar objects and for creating simple animations. The main feature of a loop is the COUNTER. Once this is clicked on, a value can be assigned. This value represents the number of iterations the loop will take.

- **+ Button**

This button inserts into the program a statement of the form:

**increment ( DISTANCE, AMOUNT )**

This is used to increase the value of a distance variable by some constant amount. Once DISTANCE is clicked on, a distance variable can be chosen, followed by OK to continue. Then click AMOUNT, followed by a constant from the keypad, and finally OK to continue.

- **- Button**

This button inserts into the program a statement of the form:

**decrement ( DISTANCE, AMOUNT )**

This is used to decrease the value of a distance variable by some constant amount. Once DISTANCE is clicked on, a distance variable

can be chosen, followed by OK to continue. Then click AMOUNT, followed by a constant from the keypad, and finally OK to continue.

### Example of using code to draw objects

In this example we will use code to draw a circle. Circles consist of a center point and radius distance. Located in the Variables Declarations pane of the Watson Graphics Lab, the Circle button lets you create a circle variable. Click the Circle button to continue.

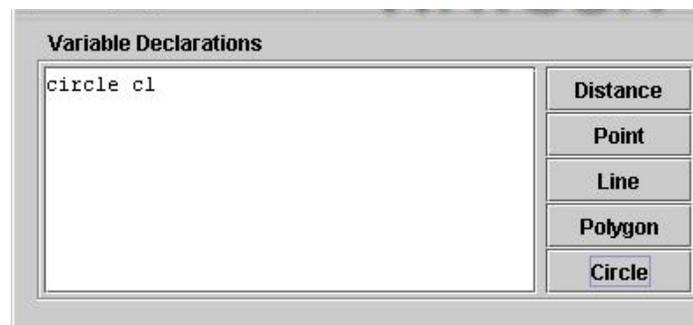


Figure 5.5: Variable of circle defined.

You must define the center point and assign the radius distance to the variable. Click the Assign button to continue. You must choose the variable to assign values to. Click VARIABLE to continue. This is where you choose the variable to assign values to. Choose c1 to continue. Click OK to continue.

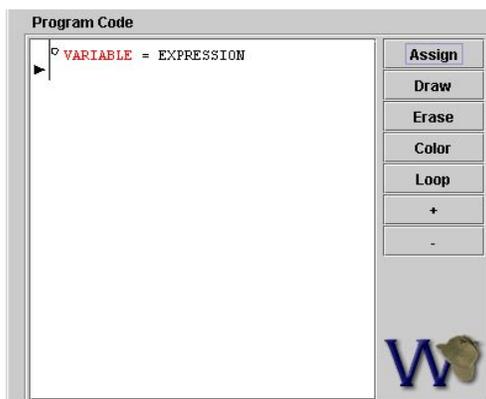


Figure 5.6: Click to select variable.



Figure 5.7: Select c1 for the circle.

You need to define the X-axis value of the center point. Click X to continue. Once you have entered a value, you can move on. Click OK to continue. You need to define the Y-axis value of the center point. Click Y to continue. Once you have entered a value, you can move on. Click OK to continue.

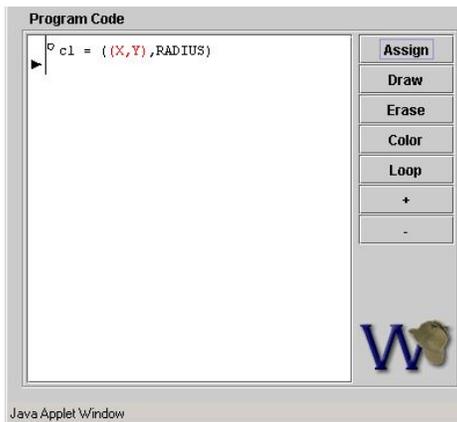


Figure 5.8: X and Y coordinates.

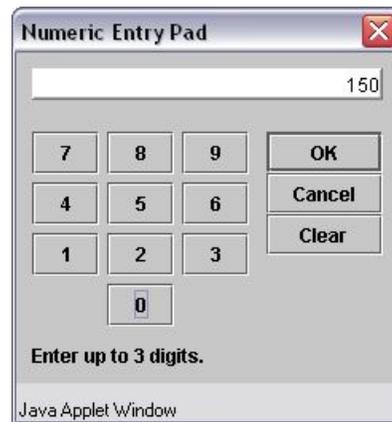


Figure 5.9: Numeric entry pad.

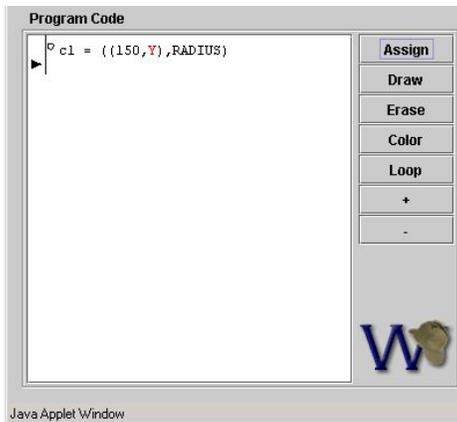


Figure 5.10: X is entered.

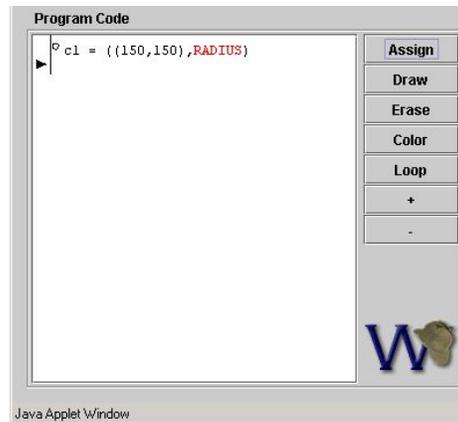


Figure 5.11: Radius must be entered.

You must choose a radius value for your circle. Click RADIUS to continue. Once you have entered a value, you can move on. Click OK to continue. Now that you have the values for the circle, you can draw it. Click the Draw Button to continue. You need to tell the draw function which object to draw. Click OBJECT to continue. Choose c1 to continue. Click OK to continue.

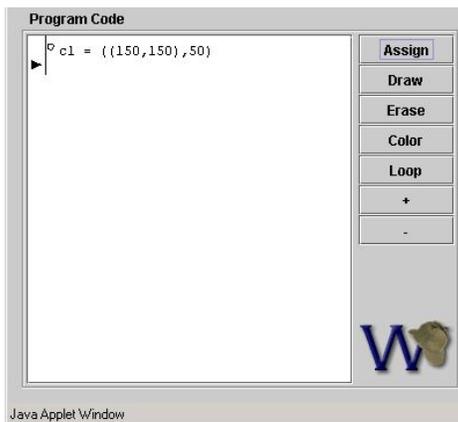


Figure 5.13: Statement for circle.

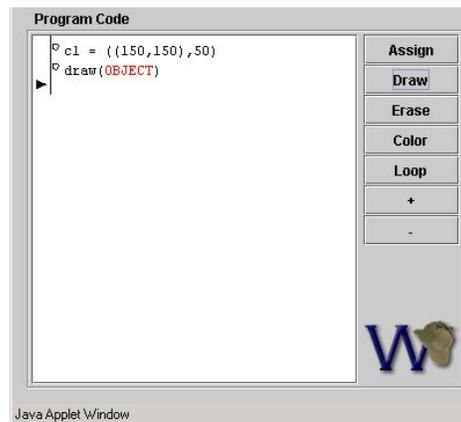
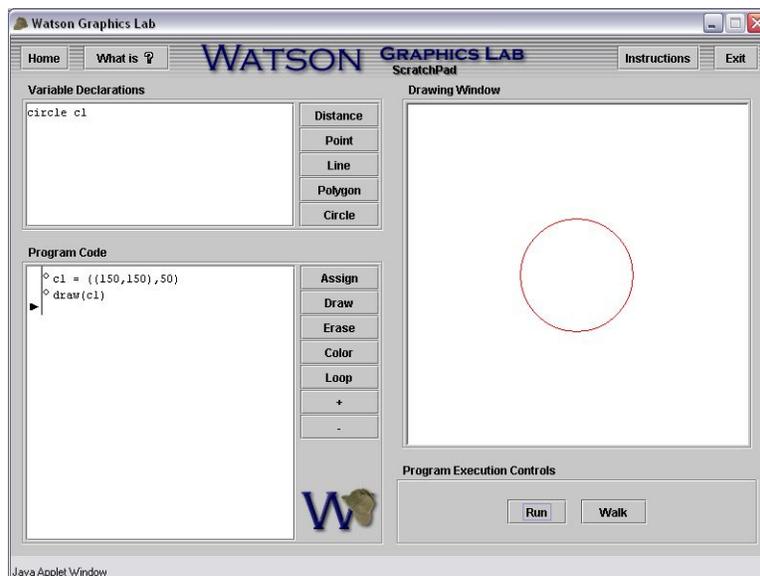


Figure 5.14: Object must be selected.

Click the Run Button to draw the circle.



*Figure 5.16: Program after it is ran.*

## 5.3 Drawing Window

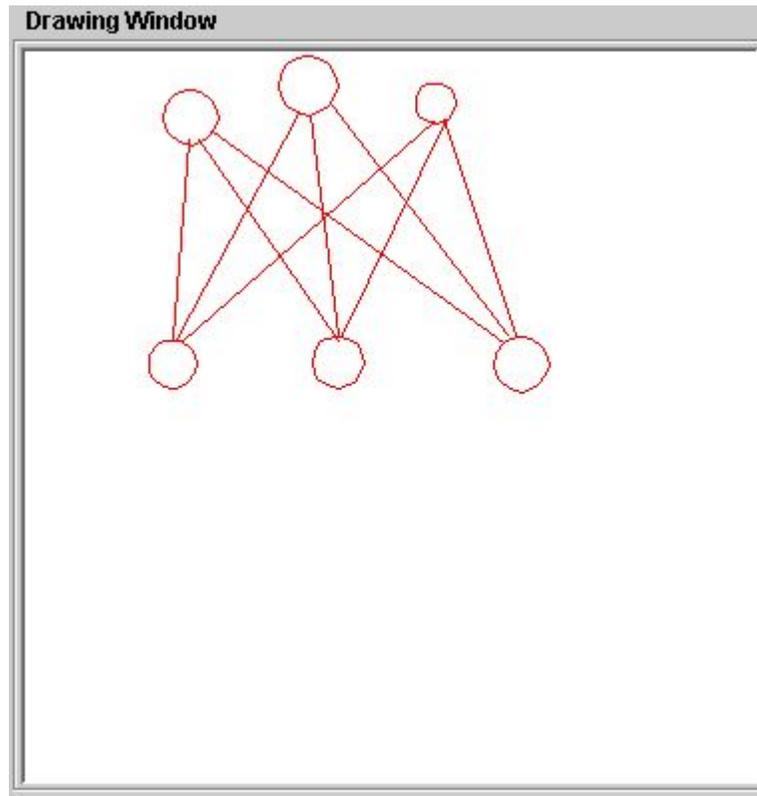


Figure 5.17: The Drawing Window.

This is the window where the code in the Code Window is translated into graphics and displayed. This window also allows the user to create images interactively by selecting the object to be drawn (using the buttons located in the Variable Declarations area) and then clicking on the location of the objects in the drawing window (See section 5.3.1 below). Note that when the mouse is passed over this window, the coordinates are shown in the top right. The coordinates are referenced with 0,0 being the bottom left corner. This feature allows for precise placing of graphics.

### Interactive Drawing Example

You can draw an object interactively by selecting the object type and clicking in the Drawing Window, once for each point of the object. In this example we will draw a triangle.

First, we must select the object type that we want to draw, in this case a polygon. Click Polygon to continue.

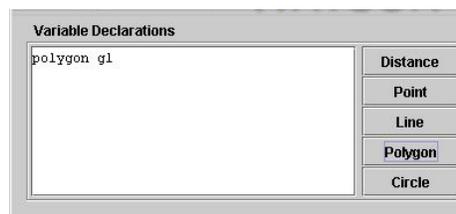


Figure 5.18: The polygon defined.

You draw a polygon by clicking once for each point in the polygon. The last point must also be the position of the first. Click anywhere in the Drawing Window to continue.



Figure 5.19: The first line.

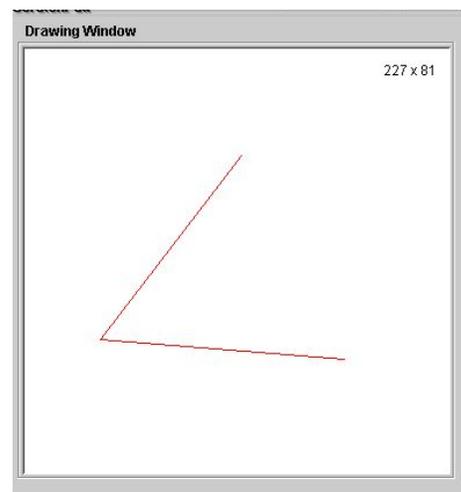


Figure 5.20: The second line.

You can then click anywhere for the next point to be drawn, in this example we are going to draw a triangle, so click anywhere in the Drawing Window to continue. The last point must be in the same area as the first. Click the first point to continue.

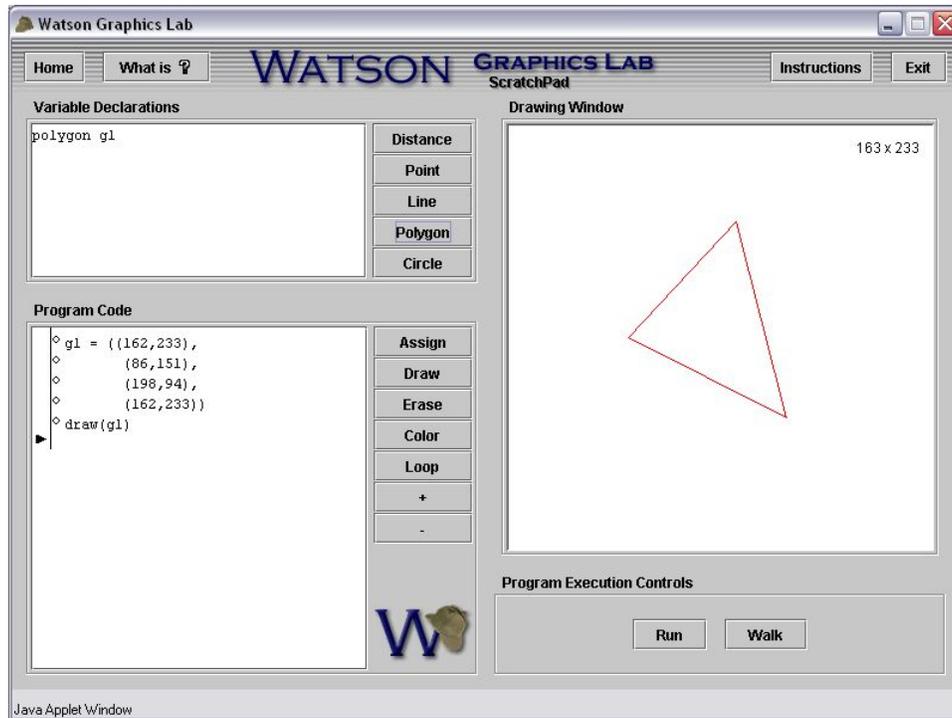


Figure 5.21: The triangle with the code produced.

## 5.4 Code Execution Controls



Figure 5.22: Code Execution Controls.

- **Run Button**

This button will execute the program shown on the left side of the screen and display its output in the Drawing Window.

- **Walk Button**

This button will place the lab in a 'walk mode' which allows the user to step through the program one line at a time. Each click of this button will progress through one line of code.

## Chapter 6: JavaScript Lab

### JavaScript Lab Concept

The JavaScript Lab continues the introduction to programming provided by the Graphics Lab. This lab is based on an augmented subset of the JavaScript programming language. While one goal of this lab is to enable the user to write simple programs in the widely available JavaScript programming language, the focus is on fundamental concepts, such as sequence, selection, repetition, and functions, rather than on particular syntactic features. Thus, the lab enforces concepts such as strong typing that are usually absent from JavaScript and includes several Watson-specific Input / Output instructions. The point-and-click syntax-directed program editor used in the Graphics Lab is also included in the JavaScript Lab -- freeing users to concentrate on program design rather than being forced to track down syntax errors.

```
1 <HTML>
2 <BODY BGCOLOR="#000064"
3 <SCRIPT LANGUAGE="JavaS
4 document.writeln("<H1>Hello W
5 // end script hiding -->
6 </SCRIPT>
7 </BODY>
8 </HTML>
```



## JavaScript Lab Layout

The Watson JavaScript Lab consists of five main areas. The first area is the Watson Title Bar, which is common to all the Watson Labs. The next area is the Code Window, which allows for the creation, display, and editing of program statements. The next area is the Command Bar, which contains all of the controls for generating code in the Code Window. Next there is the Program Output window which displays any output defined in the Code Window. Below that is the Variables window, used to display information about the variables used in the program.

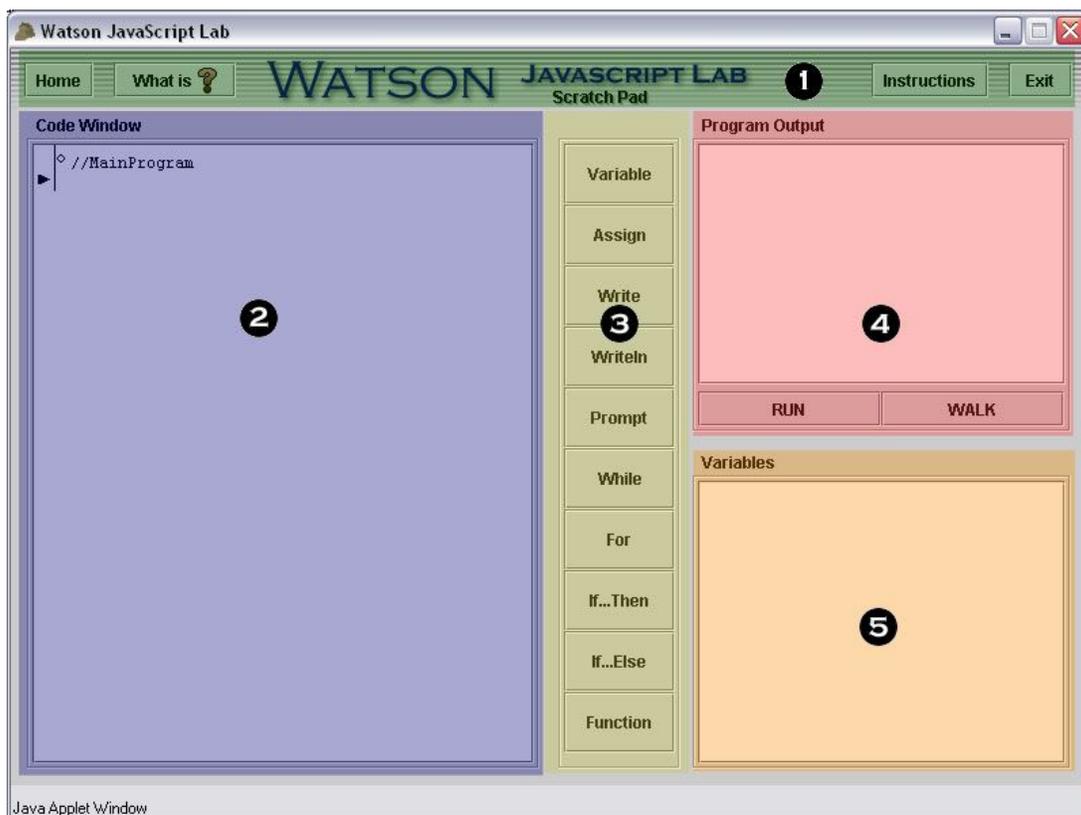


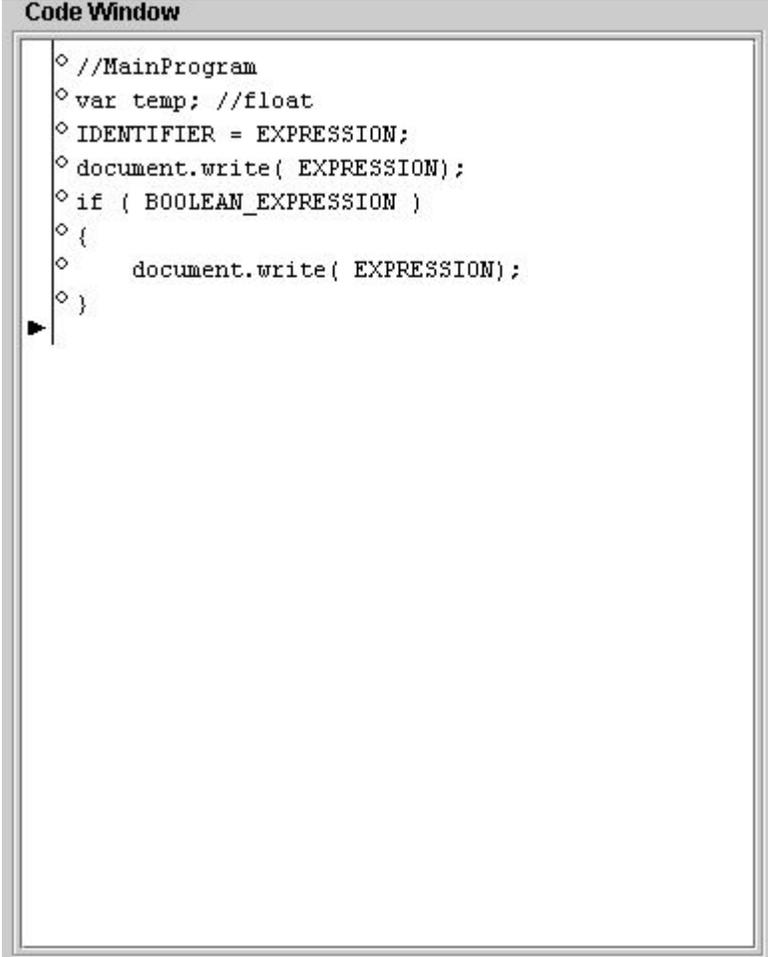
Figure 6.1: The interface of the Watson JavaScript Lab.

### 1 Watson Title Bar

This is a standard feature on all Watson Labs.

- 2 Code Window**  
This area shows the code as it is being designed by the user. It also allows the user to define values for variables, create loops and functions, etc.
- 3 Command Bar**  
This area contains all of the controls used in the Lab.
- 4 Program Output**  
This area shows the output of the program in the Code Window.
- 5 Variables**  
This area is used to display variables used by the Code Window programs. The variable name, type, and scope level are displayed.

## 6.1 Code Window



```
Code Window
◇ //MainProgram
◇ var temp; //float
◇ IDENTIFIER = EXPRESSION;
◇ document.write( EXPRESSION);
◇ if ( BOOLEAN_EXPRESSION )
◇ {
◇     document.write( EXPRESSION);
◇ }
▶
```

Figure 6.2: Code Window

- **Program Code Pane**  
This area with a white background is used to view/edit statements and assign values to variables.
- **Remove Statements**  
To remove statements from the Program Code window, click once on the small circle located to the left of the statement or statement block. The code to be deleted will highlight red.

- **Insert Statements**

To insert statements into the Program Code window above the current cursor position (small black arrow), click to the left of the vertical line at the position where the new code is to be placed. In the figure below, clicking on the position of the small red circle will place the cursor at that position.

```
◇ //MainProgram
◇ var temp; //float
◇ IDENTIFIER = EXPRESSION;
◇ document.write( EXPRESSION);
◇ if ( BOOLEAN_EXPRESSION )
◇ {
◇     document.write( EXPRESSION);
◇ }
▶
```

*Figure 6.3: Insert Statements*

## 6.2 Command Bar



Figure 6.4: The Command Bar.

- **Variable Button**

This button will create a new variable. The user must choose the name and type of the variable. The name can be chosen by clicking on 'id', then typing in a string for the name. The type is chosen by clicking on TYPES, then select one of the types from the menu, followed by OK.

- **Assign Button**

This button inserts into the program a statement of the form:

**VARIABLE = EXPRESSION;**

This is used to assign a value to a particular variable. Once a variable has been created by clicking the appropriate button, click on VARIABLE, choose the variable, then click OK to continue. Depending on which type of variable being used, there will be different fields that require values.

- **Write Button**

This button inserts into the program a statement of the form:

**document.write( EXPRESSION );**

This button will output the value of the **EXPRESSION** to the Program Output window. It **will not** create a new line after the write.

- **WriteLn Button**

This button inserts into the program a statement of the form:

**document.writeln( EXPRESSION );**

This button will output the value of the **EXPRESSION** to the Program Output window. Unlike the **Write** function, it **will** create a new line after the write.

- **Prompt Button**

This button inserts into the program a statement of the form:

**IDENTIFIER = prompt( EXPRESSION, EXPRESSION );**

This button prompts the user and waits for input. The input is then assigned to the variable and substitutes for IDENTIFIER.

- **While Button**

This button inserts into the program a statement of the form:

**while ( BOOLEAN\_EXPRESSION ) { }**

This button will insert a **WHILE** loop in the program. The user must supply the test condition for the loop to operate with.

- **For Button**

This button inserts into the program a statement of the form:

**for ( ID = EXPR; ID < EXPR; ID = ID + 1 ) { }**

This button will create a **FOR** loop in the program. The user must supply the initial condition of the loop, the stopping condition, and the iteration command for the loop to operate with.

- **If Button**

This button inserts into the program a statement of the form:

**if( BOOLEAN\_EXPRESSION ) { }**

This button will create an **IF** statement in the program. The user must supply the test condition for the statement.

- **If...Else Button**

This button inserts into the program a statement of the form:

**if( BOOLEAN\_EXPRESSION ) {} else {}**

This button will create an **IF...Else** statement in the program. The user must supply both the test conditions for the statement.

- **Function Button**

This button will allow the user to both define functions and to call them. A function must be defined before it can be called. Once the user clicks this button, they can either declare a function or call a function. If they declare a function, they must click 'id' and enter a string to identify the function. If they call a function, they must choose the name of the function to call.

## 6.3 Program Output Window

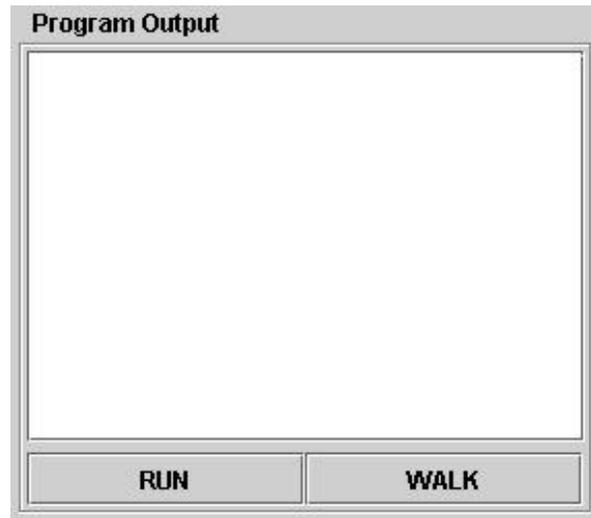
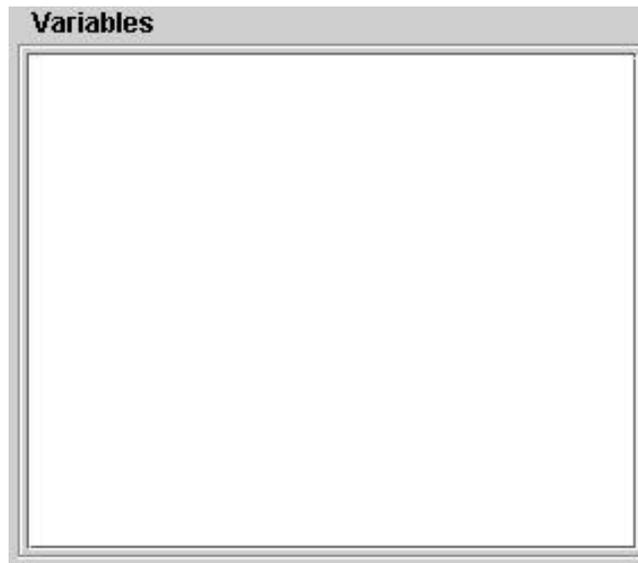


Figure 6.5: The Program Output Window.

This is the window where the output from the code in the Code Window is shown. This window also allows the user to run and walk through the program.

- **Run Button**  
This button will execute the program shown on the left side of the screen and display its output in the Output Window.
- **Walk Button**  
This button will step through the program one line at a time and the user can observe the values of the variable as they change.

## 6.4 Variables Window



*Figure 6.6: The Variables Window.*

This window is used to display variables used by the Code Window programs. The variable name, type, and scope level are displayed.

## Chapter 7: Object-Oriented Lab<sup>1</sup>

### Concept Overview

Computer Science graduates today can expect some extreme demands to be placed upon them when entering the workplace and an ability to adapt to changes in programming tools and concepts is key to continued advancement and employment. A solid foundation in one of the current tools available, object-oriented programming, could aide a graduate in easing the transition between job required skills. Unfortunately, the formation of a solid basis often falls short due to inadequate teaching and complex explanations of core object-oriented concepts.

The Watson Object-Oriented lab remedies this problem by providing explanations of object-oriented material utilizing: (a) an objects-first approach; (b) graphical representations of complex and abstract terms, including objects, classes, message passing, information hiding, and inheritance; (c) a classical “drill and test” method to reinforce knowledge. The lab targets introductory level computer science majors and provides a simple interface to interact with object-oriented features such as instantiated objects, classes, and source code editing. The Watson Object-Oriented lab will provide a unique and simple solution to forming a basic understanding of object-oriented concepts.

---

<sup>1</sup> This chapter is largely from Cliff Lemoine.

## Object-Oriented Lab Layout

The Watson Object-Oriented Lab consists of five main areas. The first area, common to all labs, is the Watson Title Bar. Beneath the Watson bar on the left is the second area, Class Diagrams, where a graphical representation of classes will appear. The third area, the Code Viewer, is a display area for showing the code that the class contains. Fourth, there is an area for Object Diagrams where each instance of a class will appear beneath its class diagram. Lastly, the Lab Controls Panel consists of a group of buttons from which you can access the labs structured activities and end-of-activity questions.

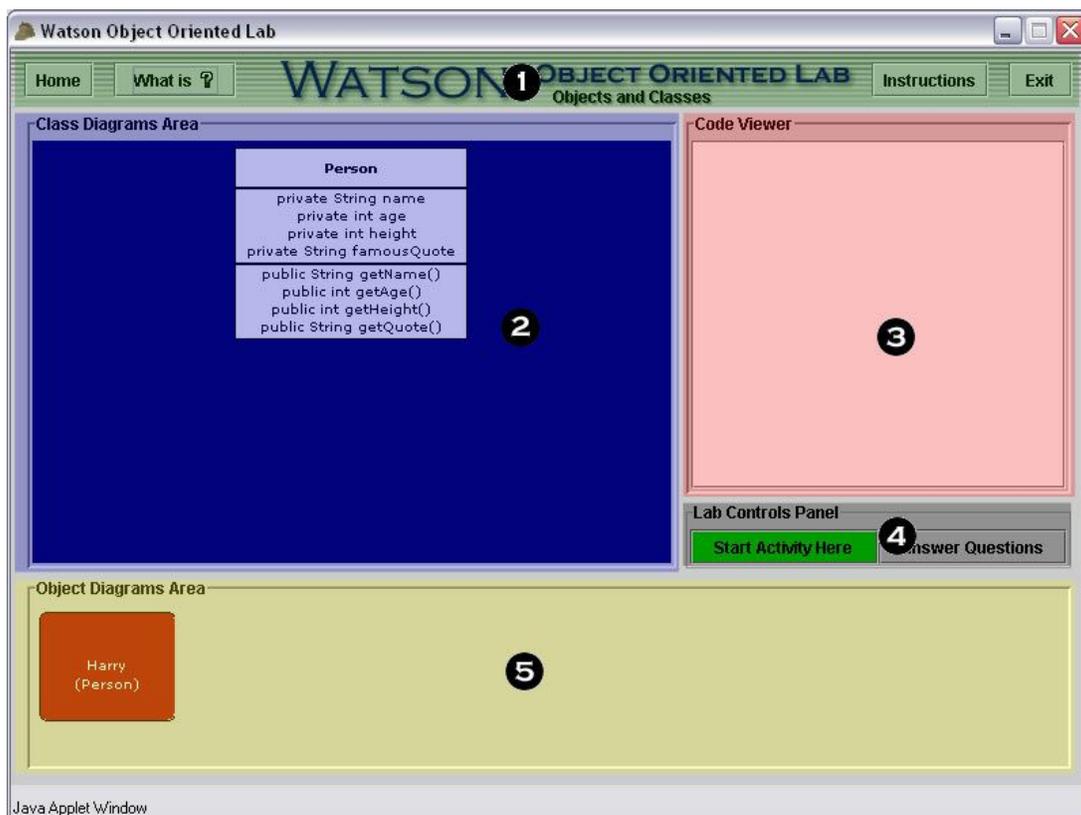
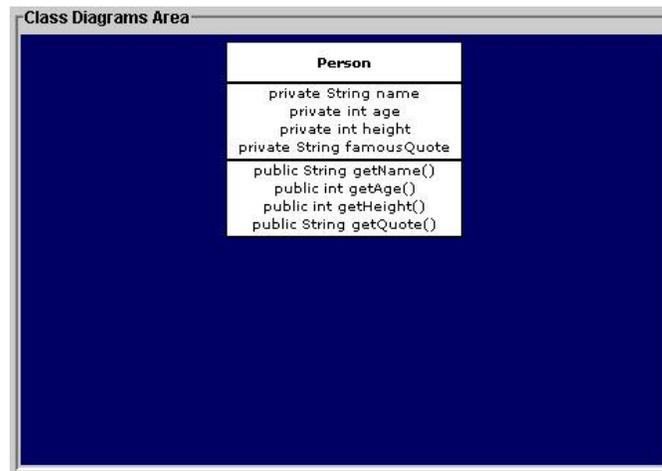


Figure 7.01: Object-Oriented Lab Interface

- 1 Watson Title Bar**  
This is a standard feature on all Watson Labs.
- 2 Class Diagrams**  
This area presents a graphical representation of the classes that have been created, along with methods and attributes associated with that class.
- 3 Code Viewer**  
The code for each class will be displayed in this window.
- 4 Object Diagrams**  
This area contains diagrams of instances of classes that appear in the Class Diagrams area.
- 5 Lab Controls Panel**  
This area contains the two buttons you will use to begin an activity and answer the questions necessary to move on to the next section: "Start Activity Here" and "Answer Questions".

## 7.1 Class Diagrams



*Figure 7.02: Class Diagrams Area, shown displaying an example class named Person.*

This area is a display window that allows the user to see what classes have been created. For each class that has been created, a class diagram will be present in the Class Diagrams Area. The Class Diagram is the most important visual component of the Watson Object-Oriented Programming lab because it presents the user with a graphical representation for a core abstract concept. Clicking each Class Diagram provides users with menu options for operations in the lab. The Class Diagram object is modeled after the UML standard diagram for representing classes. The diagram is split into three sections: the top section holds the class name (and any super-class name specification), the middle section holds the list of class data members (or attributes), and the bottom section holds the list of class methods. An example of a functional Class Diagram is shown in Figure 7.02 above. In addition, if the class inherits the attributes and methods of another class there will be a double colon after the class name, followed by the name of the class it is inheriting from (also called it's superclass).

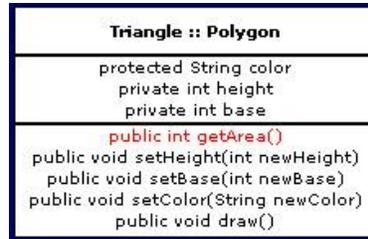


Figure 7.03: The triangle class, which is a subclass of Polygon

Clicking each section of the Class Diagram will present you with a different menu. Clicking on the class name will present you with the Class Diagram Menu. Clicking on the section of the class's box that contains attributes will bring up the Class Attribute Menu. Lastly, clicking any of the methods will bring up the Class Method Menu. These three menus contain all the actions necessary for creating classes, modifying classes, deleting classes, and viewing the code for all of the classes in the labs.

When you click anywhere inside the Class Diagrams Area (excluding an actual Class Diagram), a menu of possible actions is provided. The three basic options provided from this menu, shown in Figure 7.04, are Create a New Class, View Class Code, and Remove a Class Diagram.

### 7.1.1 Class Diagrams Area Menu

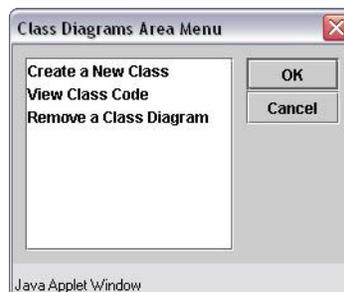


Figure 7.04: The Class Diagrams Area Menu

This menu will allow you to create a new class, view class code, or remove an existing class from the diagram area.

### Creating a New Class

To create a new class, click anywhere in the class diagram area (in blue), except for the class diagram itself. You will be presented with a menu. Choose Create a New Class from this menu and click OK.

This option allows the user to create a new Class Diagram for use in the activity. Upon selecting this option, the user is presented with a New Class Dialog, shown in Figure 7.05, which allows the user to name the new class, as well as set a class to extend from (used in the Inheritance activity primarily).



Figure 7.05: New Class Dialog

Upon the user's clicking OK, several naming convention checks are performed on the user-entered name. Once these checks pass, a new Class Diagram is placed in the Class Diagrams Area.

### Viewing Class Code

To create a new class, click anywhere in the class diagram area (in blue), except for the class diagram itself. You will be presented with a menu. Choose View Class Code from this menu and click OK.

This option provides the user with a mechanism for viewing the generated source code from a Class Diagram in the Code Viewer. This option appears in practically all of the mouse click menus and the functionality of displaying code in the Code Viewer is a default operation for practically all other operations performed in the lab. For example, when a user edits an attribute or method, the code for the updated Class Diagram is automatically displayed without having to ask the user to explicitly view the class code.

### Removing a Class Diagram

To remove a class, click anywhere in the class diagram area (in blue), except for the class diagram itself. You will be presented with a menu. Choose Remove a Class Diagram from this menu and click OK.

This option allows the user to remove a user-created Class Diagram from the Class Diagrams Area. The sole limitation on this operation is that lab-defined Class Diagrams (the diagrams that are loaded by default for each activity) are not to be removed. This limitation will prevent a user from restarting an activity after accidentally removing a diagram needed to complete the tutorial set.

Upon removal of the Class Diagram, any Object Diagrams (and associated Nested Object Diagrams) created from this class will also be removed from the Object Diagrams Area to prevent the users from running an instance of a class no longer present in the lab environment.

If you choose to either view the class code, or remove a class diagram, and there are multiple classes in the Class diagram area, a dialog box will appear, prompting you to choose which class to complete the selected action with.

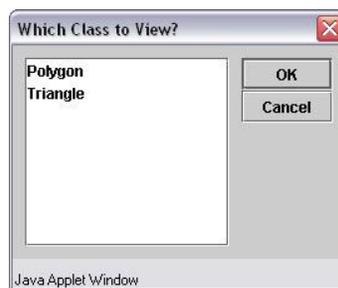


Figure 7.06: An example dialog for choosing which class' code to display

### 7.1.2 Class Diagram Menu

The Class Diagram Menu, shown in figure 7.07, provides the users with the basic operations associated with interacting with a class as a whole. These

operations include class instantiation (Create an Instance), addition of class data members and methods (Add a New Attribute, Add a New Method), viewing of class source code (View Class Code), and the deletion of a class from the lab environment (Remove this Class Diagram). The modification of class attributes and methods are not listed in this menu; instead, class attribute and method modification operations are listed in separate menus associated with mouse click events on the actual attributes and methods.

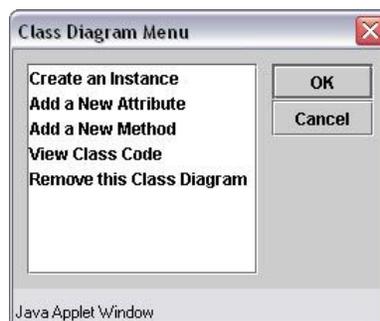


Figure 7.07: the Class Diagram Menu

### Creating An Instance

Click on the first option, titled "Create an Instance". A dialog box prompting for the name of the new object to be created will appear.



Figure 7.08: the constructor dialog

Once you enter a name for the object and click OK, an instance of this object will be created, and an Object Diagram representing that instance will appear in the Object Diagrams Area.

### Adding a New Attribute

To add an attribute to a class, go to the Class Diagram Menu and select "Add

a New Attribute". This will bring up a new window, shown in figure 7.09, where you can specify the name, the data type, and the access type of the new object.

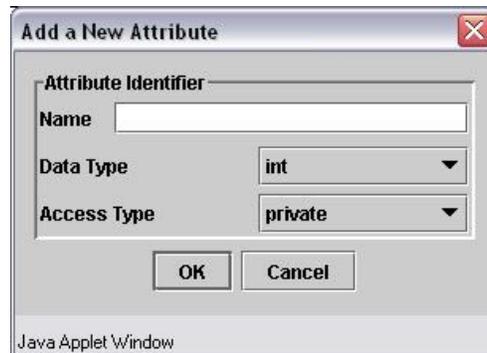


Figure 7.09: The Add a New Attribute Window

You are able to select from three predefined data types: int (integer), Boolean (either true or false), or String (group of text characters). In addition to these predefined types, you are able to specify that your attribute is a type of class. Any classes that are listed in the Class Diagram Area will be available as a data type. In choosing an access type, you may choose public, private, or protected.

Inheritance poses a special problem for the addition of a new attribute to a Class Diagram. If the access type of this new attribute is public or protected and at least one sub-class of this Class Diagram is defined in the lab environment, then this new attribute must be propagated down to the sub-class Class Diagram. This problem is fixed easily by adding the new attribute to the sub-class Class Diagram(s) after it is added to the current Class Diagram.

#### Adding a New Method

To add a new method to a class, click on the name of the class, and choose "Add a New Method" from the menu that appears. A new window, shown in figure 7.10, will pop up allowing you to set the specifics of the method that you wish to create.



Figure 7.10: the Add a New Method Window

The same options are present here for specifying name, return type, and access type, as were present in the add an attribute window. In addition to this, however, you have three new sections: Method Parameters, Local Variables, and Statements. Method parameters can be added if you are planning to pass any data to your method via a variable. Local variables can be created if you are going to store data inside your method. Both of the previous sections, Method Parameters and Local Variables, will prompt you for name and data type for your variable.

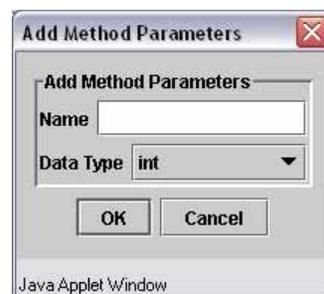


Figure 7.11: Add Method Parameters Dialog

As with creating a new attribute, you are able to specify a name, and data type for your parameters and variables. See the section titled "Adding a New Attribute" for more information on data types and naming.

The right-hand side of the dialog contains the graphical editor for simple statements inside of a method, called the OOP Editor (labeled Statements). The OOP Editor provides users with eight simple Java statement stubs which they can interact with to complete with specific information. It also handles some good programming practices by limiting data types to appropriate terms, by checking for incomplete placeholders, and semantic type-checking for the return type of the method. These incomplete placeholders (such as those labeled Identifier and Expression in figure 7.11) can be replaced with meaningful data by clicking on them, and choosing the appropriate data.

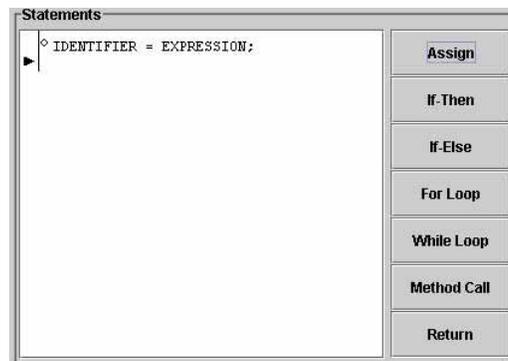


Figure 7.12: An example statement with incomplete placeholders

If you lose your place in an activity, or forget what exactly you are supposed to be doing with a given method, you may click on the View tutorial dialog button to review the tutorial

### Viewing Class Code

To view a class's code, click on the name of the class, and when prompted with a menu, choose the option labeled "View Class Code". This will display the complete code for this class in the Code Viewer.

### Removing a Class Diagram

To remove a class diagram, click on the name of the class you wish to remove. You will be prompted with a list of options. Choose the option labeled "Remove this Class Diagram" and click OK.

### 7.1.3 Class Attribute Menu

When a user clicks on a class attribute, the Class Attribute Menu, shown in figure 7.13, displays the three actions that can be performed on that specific attribute: Edit this Attribute, Remove this Attribute, and View Class Code. The View Class Code option has the same functionality as previously described in the Class Diagrams Area menu option, so it will not be described here.

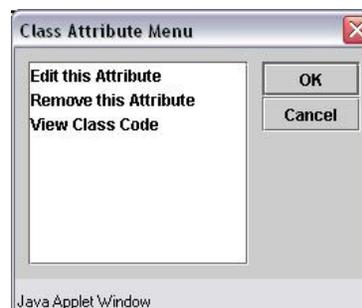


Figure 7.13: Class Attribute Menu

#### Editing an Attribute

To edit an attribute, click on its name in the Class Diagram and select Edit This Attribute from the menu that appears. This option allows the user to edit the definition of a specific class attribute, namely the one that received the mouse click. Editing is performed by an Attribute Dialog in Edit mode, in which all of the editable fields are set to the current attribute component values. The user is allowed to make the changes and click OK to apply them.

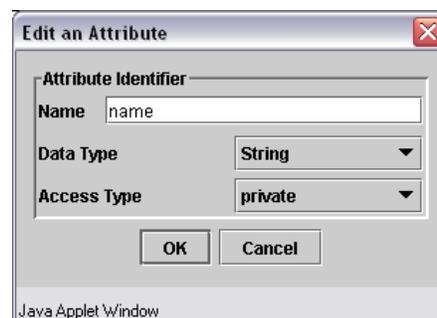


Figure 7.14: Edit an Attribute Menu, shown editing the name attribute of the person class

Like the addition of a new attribute, inheritance poses special problems for the editing of a class attribute. First, if this attribute is inherited in a sub-class, the changes must be propagated down to the sub-class. Changes in the access modifier pose the most difficult roadblock since access changes can change whether or not an attribute can be inherited. The Watson OOP Lab handles all of these special cases for the propagation of inherited attribute changes. The second problem is if this attribute is one that *is* inherited (i.e., this is a sub-class attribute), then users are not allowed to edit the attribute. The lab displays an error message explaining that editing of inherited attributes must be done in the Class Diagram in which the attribute was originally defined.

#### Removing an Attribute

To Remove an attribute click on an attribute name and select the option titled Remove This Attribute. This option allows the user to remove the current attribute from the Class Diagram. The same inheritance problems persist and the lab handles the removal of this attribute from any sub-classes, if they inherited this attribute.

#### **7.1.4 Class Method Menu**

When a user clicks on a class method, the Class Method Menu displays the three actions that can be performed on that specific method: Edit this Method, View Class Code, and Remove this Method. As before, the View Class Code option will not be described here.

#### Editing a Method

To edit a method, click on a method name in the method section of the Class Diagram. This will present you with a dialog. Choose the option titled Edit This Method.

This option allows the user to edit the definition of a method by using a Method Dialog in Edit mode (shown in figure 7.15). Just as with the Attribute Dialog in Edit mode, all of the fields are set to the current values of the method in the Class Diagram. Changes are registered back to the Class Diagram similarly and propagation of the changed method to sub-classes is performed accordingly.

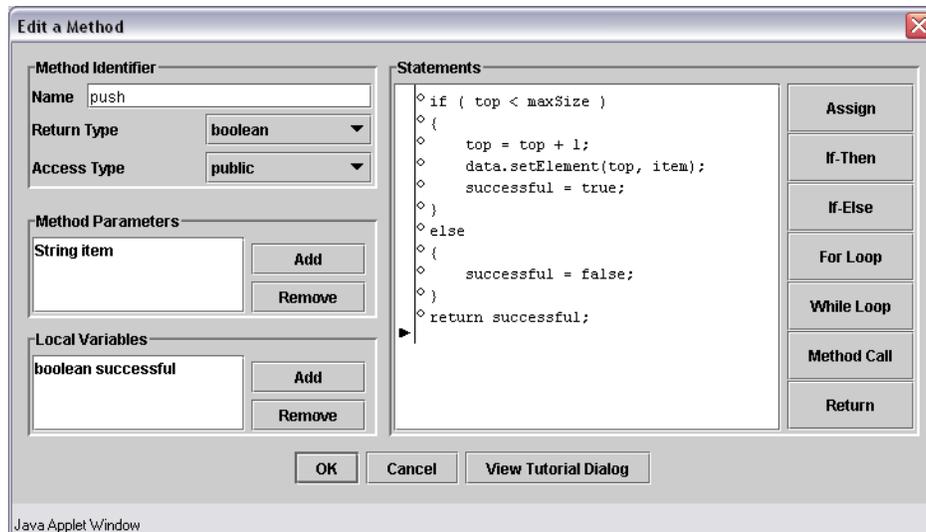


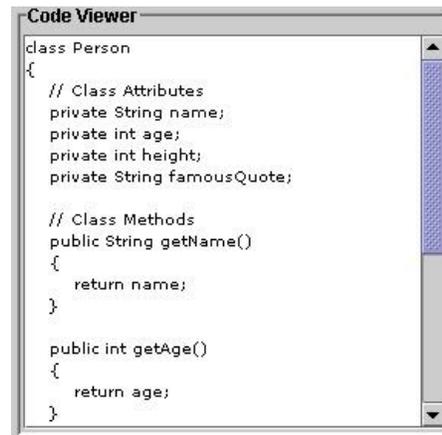
Figure 7.15: Edit Method Dialog

### Removing a Method

To remove a method, click on a method name in the method section of the Class Diagram. This will present you with a dialog. Choose the option titled Remove This Method.

This option allows the user to remove the current method from the Class Diagram. If the method is an inherited one, an error message is displayed saying that this method must be removed in the Class Diagram in which it was defined; otherwise, the method can be removed successfully. Also, if there are any sub-classes, the removal will be propagated down to reflect the removal in the current Class Diagram.

## 7.2 Code Viewer



```
Code Viewer
class Person
{
// Class Attributes
private String name;
private int age;
private int height;
private String famousQuote;

// Class Methods
public String getName()
{
return name;
}

public int getAge()
{
return age;
}
```

Figure 7.16: Code Viewer, showing code from the Person class

The Code Viewer, shown in Figure 13, displays the generated Java source code for a Class Diagram specified by the user. The Code Viewer is updated on several operations within the lab - particularly, any time a user requests a interactive option from a menu based in a Class Diagram. Also, the Code Viewer is updated any time the user Runs an Instance of an object.

This code is not editable from this window. It is only meant to show you what the complete class' code will look like.

## 7.3 Object Diagrams



Figure 7.17: Object Diagrams area, shown an example object

The Object Diagrams Area, shown in Figure 7.17, contains the instantiated objects used in the current exercise. Instances of a class (objects) are visu-

ally represented by an Object Diagram and these diagrams are added from left to right. The Object Diagram helps users visualize an instance of a class. The diagram also loosely follows the UML standard notation, and provides users with a mouse click menu of options to perform on that specific diagram. An Object Diagram displays the object name and the class that it instantiated from to allow users to see a distinction between instances.



*Figure 7.18: Object Diagram of Harry, an object of the class Person*

Like the Class Diagrams Area, the Object Diagrams Area provides the user with a menu of possible actions in the lab to perform by clicking anywhere in the area, or on the object. The menu that appears when you click anywhere in the Object Diagrams Area, except on the actual object diagram itself, is called the Object Diagrams Area Menu. In addition to this menu, another menu pops up after the user clicks directly on the Object Diagram. This menu is called the Object Diagrams Menu, and displays the options available in association with a single Object Diagram.

In addition to the regular Object Diagram there is another type of Diagram for Objects, the Nested Object Diagram. The Nested Object Diagram is similar to a regular Object Diagram, except that this type of diagram is meant to illustrate class attributes of non-basic data types. Because these non-basic data types are really objects themselves and are instantiated at the same time as the primary object, this special diagram represents these non-basic data types. A Nested Object Diagram does not implement a mouse listener interface, so users cannot modify or interact with the diagram. Along with displaying the object name and data type, the Nested Object Diagram also displays which Object Diagram it is associated with so that multiple instances of a particular class can be differentiated. The figure below illustrates a Nested Object Diagram alongside its parent Object Diagram.

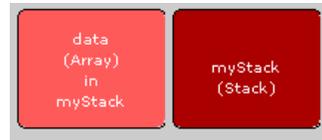


Figure 7.19: Nested Object Diagram Example

### 7.3.1 Object Diagrams Area Menu

The base menu for the Object Diagrams Area, shown in Figure 7.18, allows the user just two options: Create an Instance and Remove an Object Diagram. Both of these functionalities and user interaction constraints will be discussed in the section entitled “Object Diagrams Area Menu”.



Figure 7.20: Object Diagrams Area Menu

#### Creating an Instance

To create an instance, click anywhere inside the Object Diagrams Area and choose Create an Instance from the menu that appears.

This option allows users to create a new Object Diagram from a Class Diagram by using a Constructor Dialog. Similar to the New Class Dialog, a Constructor Dialog (Figure 7.08) allows users to enter a name for the Object Diagram, which is verified against the naming conventions upon clicking the OK button. Once the name passes the naming checks, a new Object Diagram is created (along with any Nested Object Diagrams) and placed into the Object Diagrams Area container.

If there are multiple classes present, just like with previous operations, the user will be prompted to choose which class an instance will be created for.

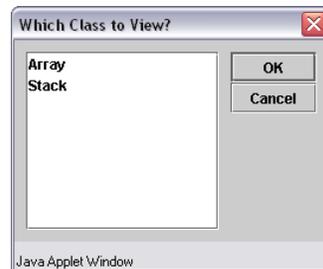


Figure 7.21: Dialog prompting for which class to instantiate

### Removing an Object Diagram

To remove an object diagram, click anywhere inside the Object Diagrams Area and choose Remove an Object Diagram from the menu that appears.

This option, similar to the Remove a Class Diagram operation, allows users to remove an Object Diagram that is currently in the Object Diagrams Area. Any Nested Object Diagrams associated with this Object Diagram will also be removed.

### 7.3.2 Object Diagrams Menu

Along with the basic information, an Object Diagram provides a menu of options consisting of three choices: Run this Instance, View Class Code, and Remove this Instance. The View Class Code and Remove this Instance options perform as described in earlier sections and will not be discussed here. The following figure is of the Object Diagram Menu that is displayed upon a mouse click on an Object Diagram.

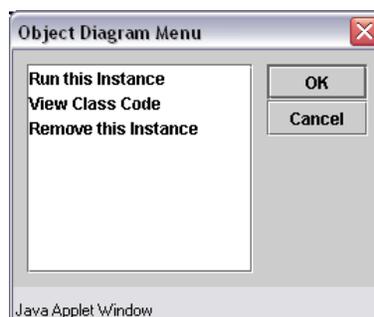


Figure 7.22: Object Diagram Menu

### Running an Instance

To run an instance of an object, click on that object, and select Run this Instance from the menu.

This option allows the user to execute the runnable object associated with this particular Object Diagram. If this Object Diagram represents an object instantiated from a Class Diagram that is not tied to an activity or a specific activity task, an error message will be displayed informing the user that no runnable object exists for this particular Object Diagram.

## 7.4 Lab Controls Panel



*Figure 7.23: Lab Controls Panel*

The Lab Controls Panel houses the two control buttons for accessing Tutorial Dialogs and the End of Activity Question set. When an activity is first loaded, the Tutorial Dialog button is highlighted and has the label of "Start Activity Here" for students to easily begin the scripted activities. Once the first Tutorial Dialog box is displayed and dismissed, the label changes back to "Tutorial Dialog" for the remainder of the activity.

### **Lab Controls Panel Components:**

- **Start Activity Here**

Clicking this button will present the user with a tutorial dialog, shown in figure 7.24, displaying the steps for the current activity. You may click on this button at any time during the activity.

The Tutorial Dialog windows are frequently the most visible component of the Watson Object-Oriented Programming lab in that they guide users through the various activities by providing informational texts and a set of small tasks to perform. Each activity consists of between

four and eight tutorial dialogs, each with one to three separate sub-tasks. These subtasks can range from adding or removing components from a Class Diagram, to creation and execution of a instance of a class, to the editing of class methods to handle specific code statements. Once users attempt to complete tasks through specific dialogs, the lab environment performs a completion check to determine if the task was actually accomplished. Upon completion of the task, the current Tutorial Dialog reappears in front of the user with a message display stating "Task Completed!" and the Next button enabled to allow advancement to the following Tutorial Dialog. A more detailed description of the Tutorial Task engine will be included in the following section.

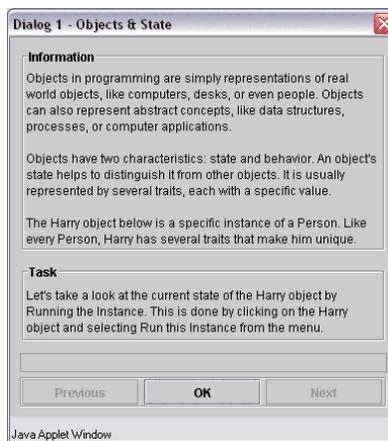


Figure 7.24: Example Activity Dialog

- **Answer Question Button**

Once all of the Tutorial Dialogs for an activity have been completed, the user is prompted to answer the End of Activity Questions, which are shown via a set of three Question Dialogs. Each question is displayed at the top of the dialog with the multiple choice answer set being presented as a set of three radio buttons with text assigned to them. Once the user clicks the Submit Answer button, the question is checked for correctness. If the user answered correctly, the answer is highlighted in green; however, an incorrect answer will highlight the correct answer in

red. If users do not answer all three questions correctly, they are asked to repeat the question set until a 100% correct answer rate is achieved.

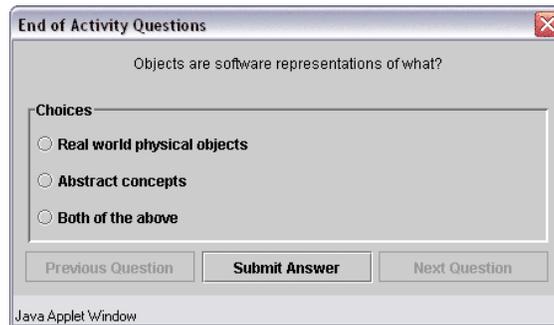


Figure 7.25: Example Question Dialog

## Chapter 8: Assembly Lab

### Concept Overview

Assembly language programming is one of the lowest levels of programming a machine. In assembly language, the programmer has a limited set of instructions to work with for the specific machine that he is programming. This is different from other high level languages which translates code into assembly which in turn is translated into binary. The programmer has more control over the computer programming in assembly than he would in other high level languages. Assembly programs also have a higher probability to be more efficient when done correctly, but this comes at the cost of the code is very difficult to understand and it is difficult to maintain.

Assembly programming offers a "1-to-1" mapping of assembly instructions to machine instructions. Due to this "1-to-1" mapping, assembly programming is closely tied to the capabilities of the machines (CPUs) being programmed. This lab gives an overview to the "low-level" Assembly programming language using the Watson Virtual Machine (WVM). The WVM has the following specifications:

- 16 32-bit registers
- 256 32-bit memory locations
- CPU with 16 machine level instructions
- Data bus that links the CPU and the memory

The WVM is a simple yet powerful tool to help you understand what is happening at the low levels of a computer. With only 16 instructions, you can perform a wide variety of programming functions.

```
1101101010101101011000011101010001100110100110110101101101010101010101000011101010001100110100110110100011010111100110111011010011011010010011010  
0110110100110101011010111010110000111010100011001101101101010011010110000111010100010111001111100110011011010010100110110101100110110100101001  
10110100110101001100110100110110100111011010101100110110100101001101101010100110101010001000110011010011011010011010001111001101101001010011011010
```

## Assembly Lab Layout

The Watson Assembly Lab consists of the following main sections. The first is the assembly code section. This section displays the current assembly level code. To the left of this is memory which shows what is currently in the program's memory. Farther to the left are the general registers. Below the Assembly Code window are the command buttons. The program counter, instruction register, and flags are located to the left of this. Finally, at the bottom, there are the execution controls that allow the user to control the program and the flags that display if they are set or not.



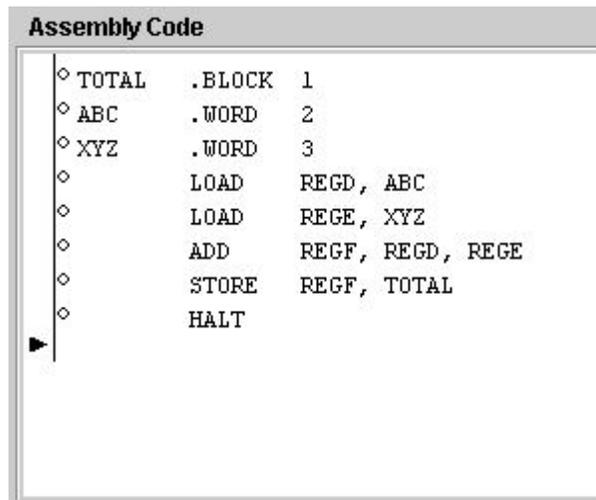
Figure 8.1: The interface for the Watson Assembly Lab.

### 1 Watson Title Bar

This is a standard feature on all Watson Labs.

- 2 Assembly Code**  
This area is used to display the current assembly code.
- 3 Memory**  
This shows the current memory of the machine.
- 4 Registers**  
The values of the registers are displayed here.
- 5 Commands**  
These buttons input commands into the assembly program.
- 6 Program Counter and Instruction Register**  
These are the values of the program counter and the instruction register.
- 7 Execution Controls**  
These buttons control the machine's execution and display.
- 8 Flags**  
The flags are shown to be set or not set.

## 8.1 Assembly Code



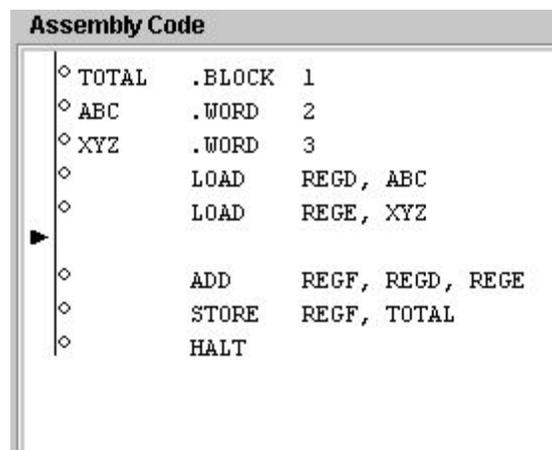
```
Assembly Code
◇ TOTAL .BLOCK 1
◇ ABC .WORD 2
◇ XYZ .WORD 3
◇
◇ LOAD REGD, ABC
◇
◇ LOAD REGE, XYZ
◇
◇ ADD REGF, REGD, REGE
◇
◇ STORE REGF, TOTAL
◇
◇ HALT
▶
```

Figure 8.2: The Assembly Code Editor.

This is the area where the assembly code is displayed. The assembly commands are able to control what a computer does. Each command has a specific numerical value that is stored in memory. When the machine reads that number, it knows what operation to execute. The corresponding number for each command can be seen in the memory section.

### Navigating the SLED Editor

The SLED editor displays the code for the assembly program being worked on. Usually, all that is needed to create a program is to click on the command buttons and then select their options by clicking on them in the editor. Sometimes a user may want to move around in the program. To do this, click on the left side of the editor above the target line. Notice that the large triangle is now pointing to the line. To delete a line, click on the circle to the left and confirm the deletion in the dialog box.



The screenshot shows a window titled "Assembly Code" with a list of assembly instructions. Each line is preceded by a diamond-shaped cursor. A large black triangle points to the line containing the instruction "LOAD REGE, XYZ".

```
◇ TOTAL .BLOCK 1
◇ ABC .WORD 2
◇ XYZ .WORD 3
◇ LOAD REGD, ABC
◇ LOAD REGE, XYZ
◇ ADD REGF, REGD, REGE
◇ STORE REGF, TOTAL
◇ HALT
```

Figure 8.3: Moving the cursor in SLED.

### Creating A Simple Program

Let's create a simple program to add two numbers. Click on the .Word button to make a new variable. Click on label and type in ABC. Click on const and select decimal. Type in 123. Next, repeat this to make DEF. Click on const and select hexadecimal. Type in AB.

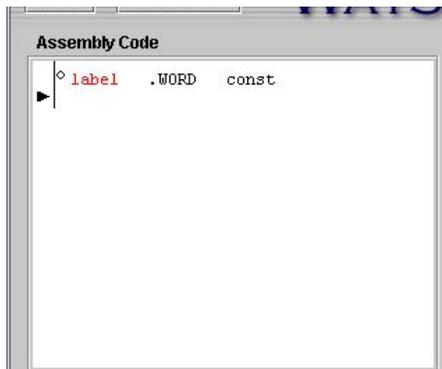


Figure 8.4:

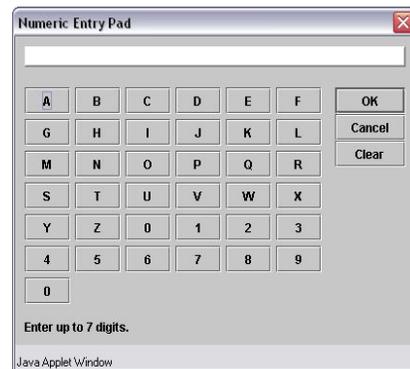


Figure 8.5:

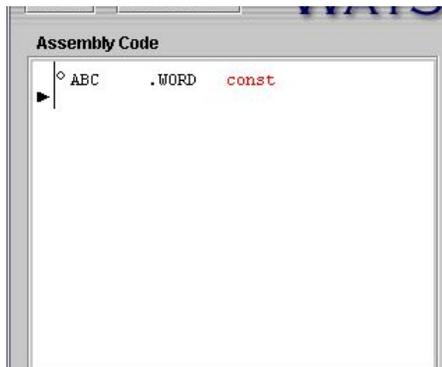


Figure 8.6:

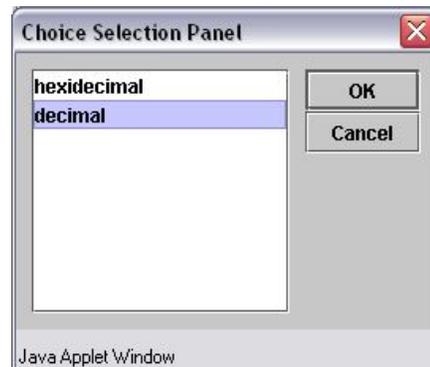


Figure 8.7:

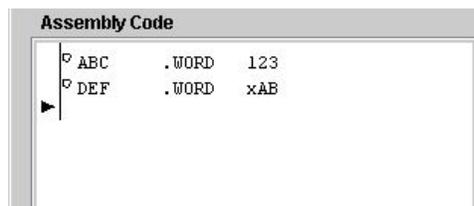


Figure 8.8:

Click on Load and select register A and load into it ABC. Click on Load again

and select register B and load into it DEF.

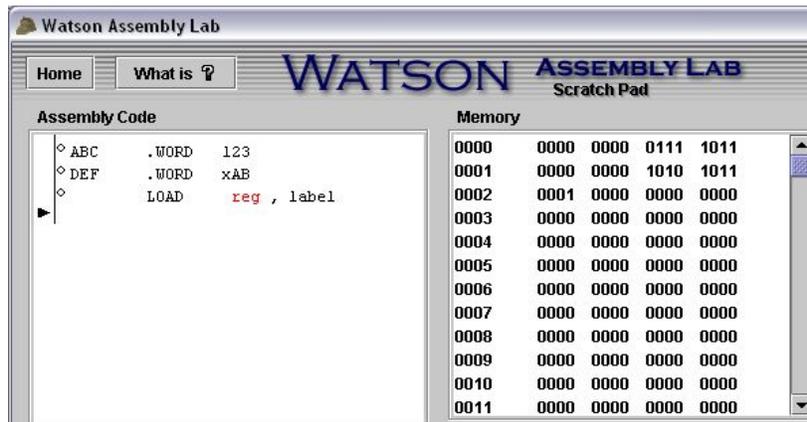


Figure 8.9:

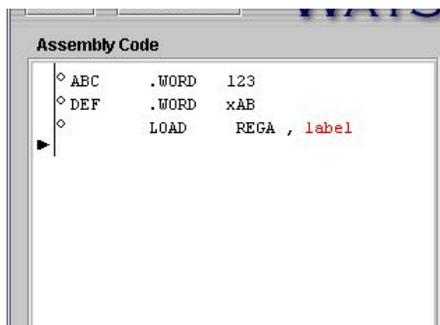


Figure 8.10:



Figure 8.11:

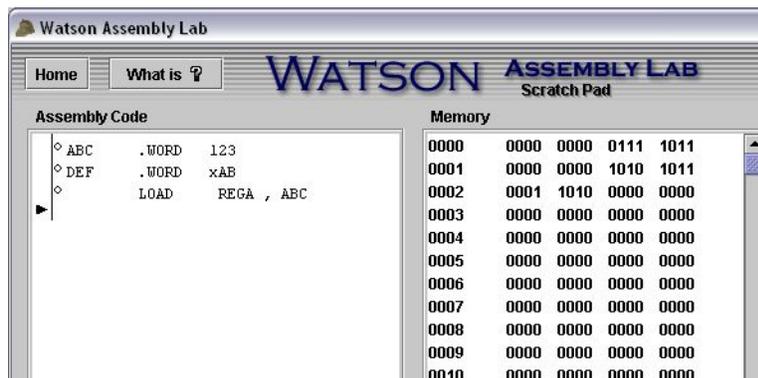


Figure 8.12:

Click on Add and select register C to store it, and add registers A and B together. Click on Halt to finish. Click Base 2/16 to make it easier to read.

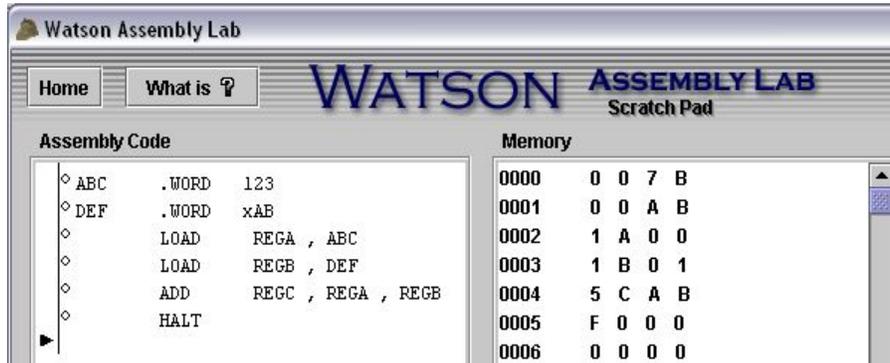


Figure 8.13:

Click run and notice how the contents of register A and B have been added and stored in register C.

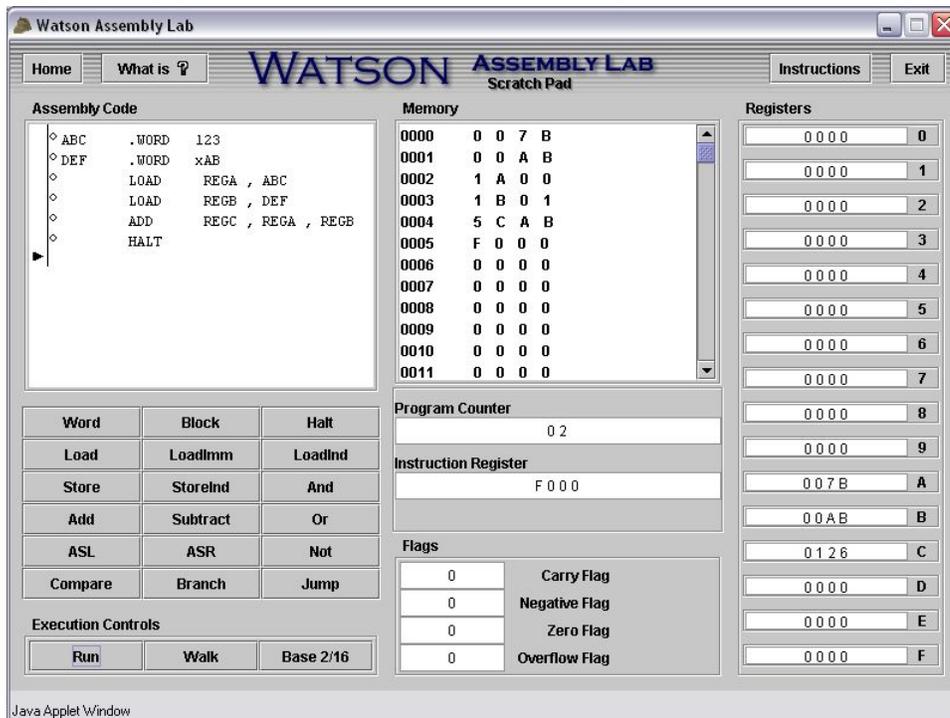
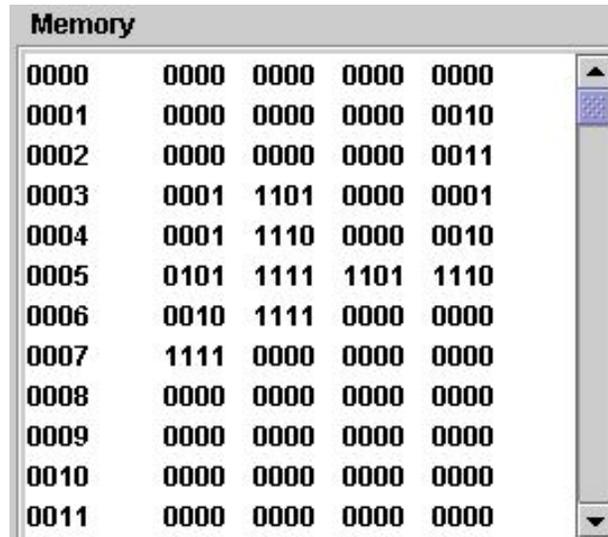


Figure 8.14:

## 8.2 Memory



Address	Content
0000	0000 0000 0000 0000
0001	0000 0000 0000 0010
0002	0000 0000 0000 0011
0003	0001 1101 0000 0001
0004	0001 1110 0000 0010
0005	0101 1111 1101 1110
0006	0010 1111 0000 0000
0007	1111 0000 0000 0000
0008	0000 0000 0000 0000
0009	0000 0000 0000 0000
0010	0000 0000 0000 0000
0011	0000 0000 0000 0000

Figure 8.15: The contents of memory.

This is the memory where the program runs. This contains the program code and other information the program needs in order to execute. Each memory location is 16 bits. These locations can be addressed with 4 bits.

## 8.3 Registers

Registers	
0000 0000 0000 0000	0
0000 0000 0000 0001	1
0000 0000 0000 0000	2
0000 0000 0000 0000	3
0000 0000 0000 0000	4
0000 0000 0000 0000	5
0000 0000 0000 0000	6
0000 0000 0000 0000	7
0000 0000 0000 0000	8
0000 0000 0000 0000	9
0000 0000 0000 0101	A
0000 0000 0000 0010	B
0000 0000 0000 1010	C
0000 0000 0000 0000	D
0000 0000 0000 0000	E
0000 0000 0000 0000	F

Figure 8.16: The general registers.

These are the machine's registers. The values for the registers change only when new values are loaded into them. To load a value into a specific register just insert the command to load the value and specify the register it is to be loaded into and the value to be loaded.

## 8.4 Commands

Word	Block	Halt
Load	LoadImm	LoadInd
Store	StoreInd	And
Add	Subtract	Or
ASL	ASR	Not
Compare	Branch	Jump

Figure 8.17: The command buttons.

These buttons are used to program the assembly code. By pressing the button, the lab will input the code for that command into the assembly code window. Each button represents a different command.

- **Word Button**

Allows a variable to be added to the program. The name 'label' can be replaced by a valid variable name. The initial value can also be set.

**LABEL .WORD CONST**

- **Block Button**

This allocates a block of memory to be used for data instead of instructions. The size of the block is specified by the number of words.

**LABEL .BLOCK NUM**

- **Halt Button**

Stops the current programs execution.

**HALT**

- **Load Button**

Loads the labeled value into the specified register.

**LOAD REG, LABEL**

- **LoadImm Button**

The register is loaded with the value specified in CONST. For unsigned numbers CONST must be between 0 and 255, inclusive.

**LOADIMM REG, CONST**

- **LoadInd Button**

The first register is loaded with the value stored in the memory location whose address is held in the second register.

**LOADIND REG, REG**

- **Store Button**

The value held in the first register is stored in the memory location whose address is held in the second register.

**STORE REG, LABEL**

- **StoreInd Button**

Stores the value in another register.

**STOREIND REG, REG**

- **And Button**

Performs a bitwise AND of the 2<sup>nd</sup> and 3<sup>rd</sup> registers and stores the result in the 1<sup>st</sup> register.

**AND REG, REG, REG**

- **Add Button**

Adds the values in the 2<sup>nd</sup> and 3<sup>rd</sup> registers; the result is stored in the 1<sup>st</sup> register.

**ADD REG, REG, REG**

- **Subtract Button**

Subtracts the value in the 3<sup>rd</sup> register from the 2<sup>nd</sup> register; the result is stored in the 1<sup>st</sup> register.

**SUBTRACT REG, REG, REG**

- **Or Button**

Performs a bitwise OR of the 2<sup>nd</sup> and 3<sup>rd</sup> registers and stores the result in the 1<sup>st</sup> register.

**OR REG, REG, REG**

- **ASL Button** – Arithmetic Shift Left

A copy is placed in the 1<sup>st</sup> register and the shifted value is placed in the 2<sup>nd</sup> register. The left shift inserts 0's on the right.

**ASL REG, REG, BITS**

- **ASR Button** – Arithmetic Shift Right

A copy is placed in the 1<sup>st</sup> register and the shifted value is placed in the 2<sup>nd</sup> register. The right shift inserts 0's on the left.

**ASR REG, REG, BITS**

- **Not Button**

The complement of the 2<sup>nd</sup> register is stored in the 1<sup>st</sup> register.

**NOT REG, REG**

- **Compare Button**

The 2<sup>nd</sup> register is subtracted from the 1<sup>st</sup> register and the flags are set based on this operation.

**COMPARE REG, REG**

- **Branch Button**

The program counter will change based upon a condition.

**BRANCH COND, ADDRESS**

The following conditions are recognized.

- LT – less than
- LE – less than or equal to
- GT – greater than
- GE – greater than or equal to
- EQ – equal
- NE – not equal

- **Jump Button**

Changes the program counter to the indicated value. Commands beginning at the new value will start to be executed.

**JUMP ADDRESS**

## 8.5 Program Counter And Instruction Register

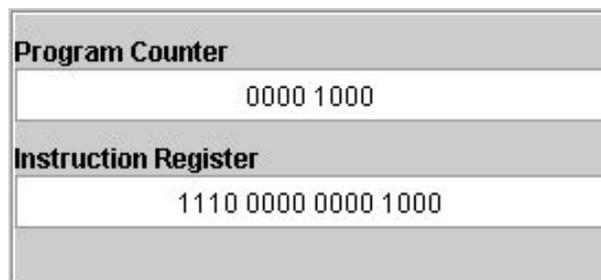


Figure 8.18: The program counter.

The program counter stores the address of the next instruction to be executed. By default, after an instruction is executed, the instruction register fetches the instruction at the location of the program counter and the program counter is incremented by one. A value can be loaded into the program counter and new instructions will come from that location in memory. The instruction register stores the current instruction at the memory location referenced by the program counter.

## 8.6 Execution Controls



Figure 8.19: The execution commands

- **Run Button**

This button runs the program.

- **Walk Button**  
This walks through program one command at a time.
- **Base 2/16 Button**  
Changes display from binary to hex and vice-versa.

## 8.7 Flags

Flags	
0	Carry Flag
1	Negative Flag
0	Zero Flag
0	Overflow Flag

Figure 8.20: The flags.

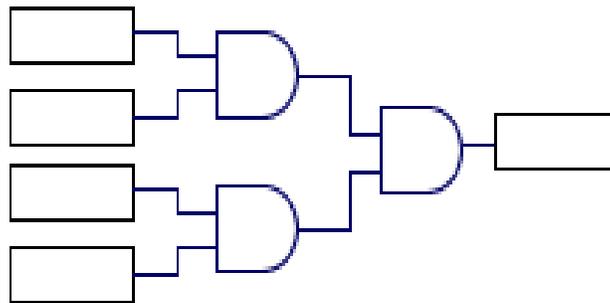
The flags are boolean values; they can be either true or false. When a flag is set to 0 it is false, and when a flag is set to 1 it is true.

- **Carry Flag**  
This flag shows if there is a carry in the machine.
- **Negative Flag**  
This flag shows if the last operation produced a negative number.
- **Zero Flag**  
This flag shows if the last operation resulted in a zero.
- **Overflow Flag**  
This flag shows if the last operation resulted in an overflow. This happens when the result of an operation is too large.

## Chapter 9: Watson Digital Logic Lab

### Lab Concept Overview

This lab examines how computer hardware can be constructed. While the details necessary to build a complete computer - even one as simple as the Watson Virtual Machine - are beyond the scope of this lab, this lab illustrates how many of the basic components of a computer, such as the addition unit and comparison circuitry, can be constructed. Surprisingly, as we will see, digital computers are based on just three simple logic circuits, or "gates" as they are called. These gates, and, or, and not, can be interconnected in various ways to form all of the major functional components of a computer.



## Digital Logic Lab Layout:

The Watson Digital Logic Lab consists of four main areas. The first area is the Watson Title Bar, which is common to all the Watson Labs. The three Circuit Design Tools are just below the Title Bar, followed by the Circuit Editor. At the bottom of the pane are the four Component Tools.

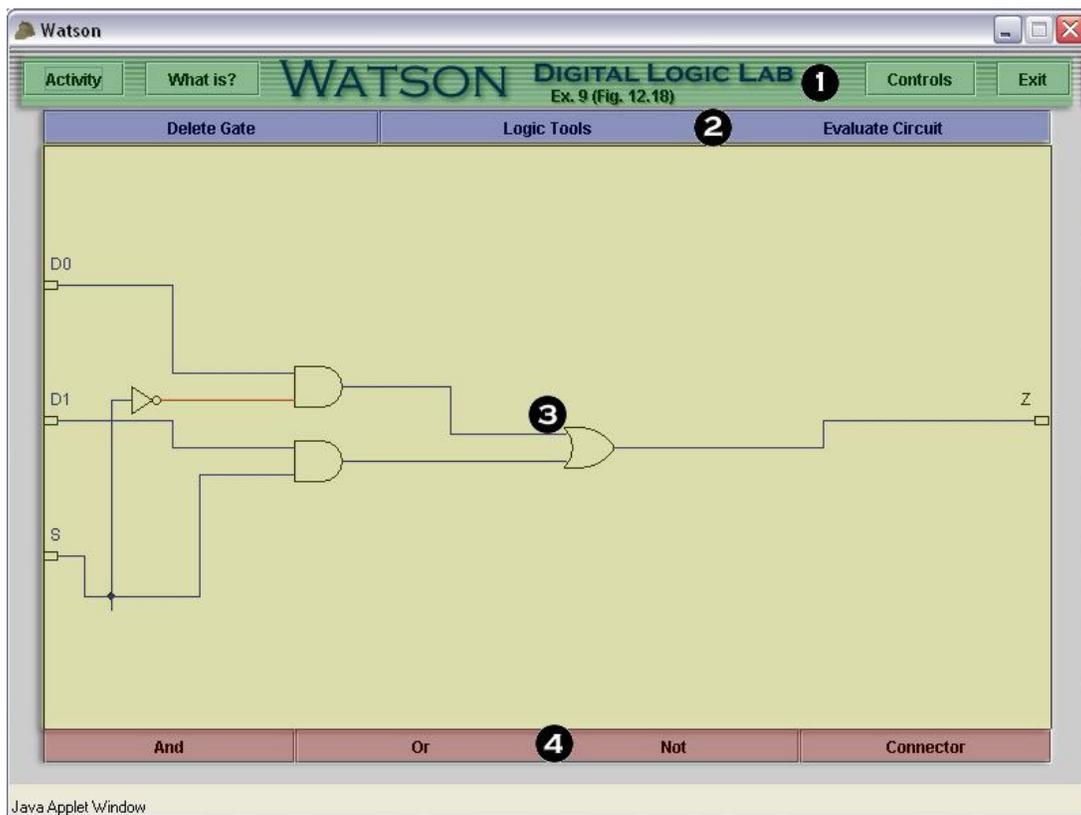


Figure 9.1: Full lab image will be here.

- 1 Watson Title Bar**  
 This is a standard feature on all Watson Labs.
- 2 Circuit Design Tools**  
 These buttons are used for circuit design tasks.

**3 Circuit Editor**

This area is where the circuit is constructed.

**4 Component Tools**

These buttons are used to create logic gates and connectors.

## 9.1 Circuit Design Tools



Figure 9.2: Circuit design tool buttons.

- **Delete Gate**

This allows a user to delete a gate or connector by selecting it. To delete multiple gates, the Delete Gate button must be selected multiple times.

- **Logic Tools**

This allows a user to select one of two tools from a menu. The Boolean Probe option allows a user to select one of the endpoints in the Circuit Editor and view the Boolean function which defines its value. The Truth Table option allows the user to view a truth table for the circuit present in the Circuit Editor.

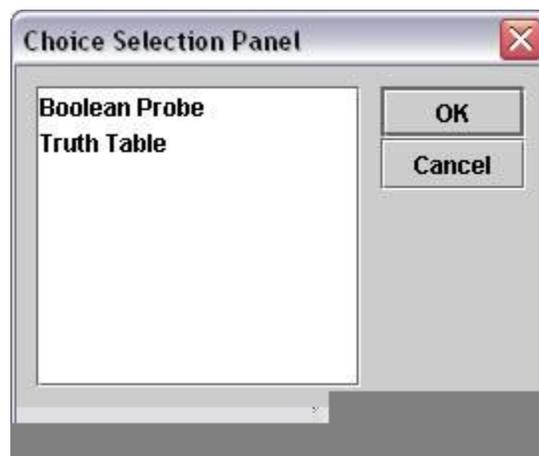


Figure 9.3: Logic Tools menu.

The Boolean probe allows the user to click any input, output, or gate and see the Boolean function that defines its value to that point. AND gates are signified by an asterisk (\*), OR gates are signified by a plus sign (+), and NOT gates are signified by a slash (/). For instance, if a user selected the top AND gate in Figure 9.1 with the Boolean probe, that user would see a dialog window with the equation  $(D0*/S)$ . D0 and S are the inputs. NOT S is signified by /S, and the asterisk signifies the AND gate that links the two inputs.



Figure 9.4: The result of a Boolean probe.

A truth table is a table detailing all possible input and output values for some circuit or Boolean equation. Figure 9.5 is a truth table for the NAND gate example in Example 1 of the lab. One can see from the truth table that when inputs A and B are set to 0 (low), output Z is set to 1 (high). The truth table arranges the inputs and outputs into columns with each row indicating a possible combination of values.

A screenshot of a dialog window titled "Truth Table". It contains a table with three columns: A, B, and Z. The table has four rows of data. Below the table is an "OK" button.

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

Figure 9.5: Truth table for a NAND gate.

- **Evaluate Circuit**

This performs a test to determine whether or not the circuit in the circuit editor meets the criteria for the current activity. If the circuit is correct, the user will receive a message stating that the circuit functions correctly. If the circuit is not correct, the user will receive a messages stating that the circuit does not meet the requirements for that activity and will also be directed to read the description of the problem again to ensure that the user understands those requirements. This function is not available in the Scratch Pad activity.

## 9.2 Circuit Editor

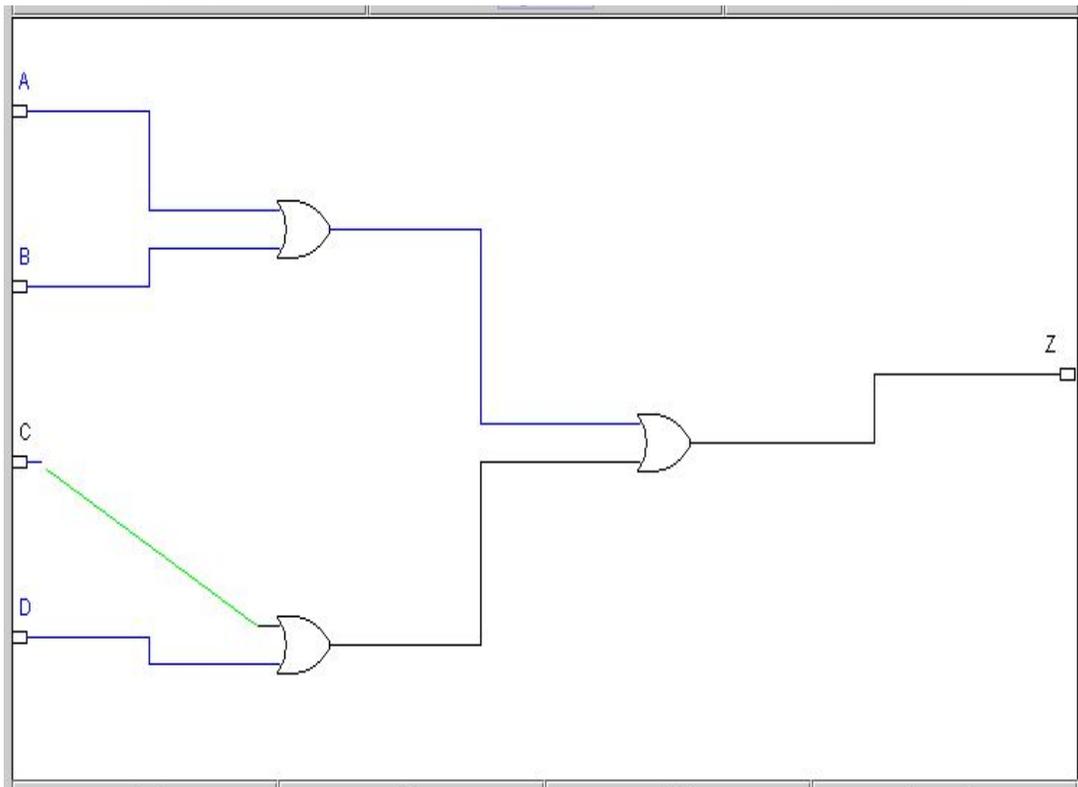


Figure 9.6: Image of circuit editor.

This area displays the circuit currently under construction and the endpoints to which it connects. The endpoints may be found on either side of the pane. Endpoints on the left side of the pane are the inputs of the circuit; endpoints on the right side of the pane are outputs.

To build a circuit, use the Component Tools to create gates and connectors, then draw lines between components, inputs, and outputs by holding down the left mouse button near the endpoints on the components and dragging the line to an endpoint on another component. The line will appear green when a connection can be made, blue or red when a valid connection exists, and black when no valid connection exists.

The circuit editor can also be used to manipulate the values in the currently available circuit. Clicking on an input will change the value of that input from high (colored red) to low (blue) or vice versa. The values will then propagate throughout the circuit according to its design. Consider an AND gate with inputs A and B and output W. If both A and B are toggled red (high) rather than blue (low), then output W will also show as red (high). Otherwise W will appear blue (low).

### 9.3 Component Tools



*Figure 9.7: Image of Component Tools*

- **And Button**  
This allows a user to place an AND gate anywhere in the Circuit Editor.
- **Or Button**  
This allows a user to place an OR gate anywhere in the Circuit Editor.
- **Not Button**  
This allows a user to place a NOT gate anywhere in the Circuit Editor.
- **Connector Button**  
This allows a user to place a connector anywhere in the Circuit Editor.

## Chapter 10: Glossary

Applet - A small program that runs in the context of a larger program on a client computer.

Arithmetic Shift - A shift in the binary representation of a number.

Array - A data structure that holds a fixed number of elements.

Assembly - A human-readable notation for the machine language that a specific computer architecture uses.

Assembly Program - A program written in assembly code.

Attribute - A data item within an object.

Base 2 - see Binary.

Base 16 - see Hexadecimal.

Binary (base-2) - A numeral system for representing numbers in which a base of two is used; that is, each digit in a binary numeral may have either of two different values. Typically, the symbols 0 and 1 are used to represent binary numbers.

Boolean Algebras - Algebraic structures which "capture the essence" of the logical operations AND, OR and NOT.

Class - Describes the rules by which objects behave; those objects, described by a particular class, are known as "instances" of said class.

Code - Any type of program that can be read by a human.

Computer Graphics - The field of synthesizing or augmenting imagery through digital means, for artistic, engineering, recreational or scientific purposes.

Computer Science - The study of computation and information processing, both in hardware and in software.

Console - A text output device for a computer system.

Constant - A fixed value in a program.

Data Set - A set of data that is available for input.

Data Structure - A way of storing data in a computer so that it can be used efficiently.

Database - An information set with a regular structure.

Dequeue - (used with queue data structures) An item is removed from the front of the queue, decreasing the queue size by one.

Enqueue- (used with queue data structures) An item is put at the end of a queue, increasing the queue size by one.

Execution - A computer running an instruction.

Finite State Machine (FSM) or Finite State Automaton (FSA) - An abstract machine that has only a finite, constant amount of memory. The internal states of the machine carry no further structure.

Function - A sequence of code which performs a specific task, as part of a larger program, and is grouped as one, or more, statement blocks.

Functional Programming - A programming paradigm that treats computation as the evaluation of mathematical functions.

Gate - see Logic Gate.

Hex - see Hexadecimal.

Hexadecimal - (often abbreviated hex) A base 16 numeral system, usually written using the symbols 0-9 and A-F.

**Imperative Programming** - A programming style that describes computation in terms of a program state and statements that change the program state.

**Instruction Register** - The part of a CPU's control unit that stores an instruction

**Internet** - The publicly available internationally interconnected system of computers (plus the information and services they provide to their users).

**Java Applet** - An applet written in the Java programming language; Java applets can run in a web browser, or in Sun's applet viewer, a stand alone tool to test applets.

**Java** - An object-oriented programming language created by Sun Microsystems.

**JavaScript** - A scripting language mainly used in web browsers.

**Join** - (used in database systems) Allows the user to join two or more relations to perform an action on the new result set.

**Linked List** - A data structure that consists of a sequence of nodes, each containing arbitrary data fields and one or two references ("links") pointing to the next and/or previous nodes. Linked lists permit insertion and removal of nodes at any point in the list in constant time, but do not allow random access.

**Lisp** - (which stands for "LIST Processing") A programming language oriented towards functional programming.

**Logic Gate** - An arrangement of electronically-controlled switches arranged to calculate operations in boolean algebra.

**Memory** - The parts of a digital computer that retain physical state (data) for some interval of time.

**Method** - Another name for an action, algorithm, function, or procedure;

more specifically, in object-oriented programming, it is an implementation of code responding to certain messages.

**Object** - A language supported mechanism for binding data tightly with methods that operate on that data.

**Pop** - (used with stack data structures) The top item is taken from the stack, decreasing stack size by one. In the case where there was no top item (i.e. the stack was empty), a stack underflow occurs.

**Program Counter** - A register in a computer processor which indicates where the computer is in its instructions.

**Project** - (used in database systems) Allows the user to select attributes from a relation.

**Push** - (used with stack data structures) An item is put on top of the stack, increasing the stack size by one. As stack size is usually limited, this may provoke a stack overflow if the maximum size is exceeded.

**Queue** - A First-In-First-Out (FIFO) data structure in which the first element in the queue will be the first one out.

**Register** - A small amount of very fast computer memory used to speed the execution of computer programs by providing quick access to commonly used values.

**Relational Database** - A collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.

**Scope** - Refers to where and when in a program the identifier can be referenced.

**Select** - (used in database systems) Allows the user to specify a query as a description of the desired result set.

Spreadsheet - A rectangular table (or grid) of information, often financial information.

Stack - A Last-in-First-Out (LIFO) data structure in which the last element in the stack will be the first one out.

Tree - A widely-used computer data structure that emulates a tree structure with a set of linked nodes. Each node has zero or more child nodes, which are below it in the tree (in computer science, unlike in nature, trees grow down, not up). The node of which a node is a child is called its parent node. A child has at most one parent; a node without a parent is called the root node (or root). Nodes with no children are called leaf nodes.

Truth Table - A kind of mathematical table used in logic to determine whether an expression is true or whether an argument is valid.

Variable – A way to store data that can be changed.

Web Browser - A software package that enables a user to display and interact with information over the Internet.

Definitions are from  
[http://en.wikipedia.org/wiki/Main\\_Page](http://en.wikipedia.org/wiki/Main_Page)  
[http://searchdatabase.techtarget.com/sDefinition/0,,sid13\\_gci212885,00.html](http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci212885,00.html)

Appendix B:

## **System Requirements For Watson Interactive Laboratories**

The Watson Interactive Laboratories use Java 1.4

Windows

Windows 9x or better.

Apple Mac

OS X.2 or better

Linux

???

## Appendix C: Credits