Programming Language:

Espresso!

a small cup of Java

PLT CS 4115
Prof. Stephen Edwards
December 18, 2003

Joya Zuber (jaz49), Philip Coakley (pwc35),
Aaron Bauman (adb54), Erin Adelman (eca41)

Table of Contents

Chapter 1

White Paper

1. Introduction and Overview:

The *Espresso!* programming language facilitates the creation of Java applets, providing an introduction to the basics of Java syntax and semantics. As computing saturates many aspects of our lives today, programming aptitude is becoming a significant part of what it means to be computer literate. With *Espresso!* inexperienced programmers and children can learn to quickly create powerful applets. This small cup of java offers a positive and practical start to a student's computer education.

2. Background:

Learning to build graphical java applications without prior programming experience is like running a marathon without training. *Espresso!* was created in order to achieve results without treading through the overhead associated with a robust Java education. Realizing that it is possible (as well as easy and fun) to use the computer as a development tool is an important step in the training of the novice programmer.

Apple's Hypercard serves as a simple interface to create dynamic multimedia presentations. Hypercard uses the easy to understand metaphor of cards and stacks to aid the user in quick comprehension of the program's functionality. The intuitive nature of Hypercard has made it a successful educational tool. In addition to the card and stack metaphor, a drag and drop interface allows novice users to realize graphical applications. However, this design choice does not forward the developer's programming prowess.

A forerunner in the cross-platform, programming, and object-oriented paradigm, Java is a necessary component of modern computer science education. In order to learn Java, one needs to become comfortable with the use of a vast API and complicated structure. This can be intimidating to the beginning programmer.

Espresso! is a fusion of the pedagogical nature of Hypercard and the power of Java.

3. Simple:

Simplicity is a defining feature of *Espresso!* For the beginning programmer, *Espresso!* affords the functionality of Java without the tedious and confusing overhead involved in creating a Java applet from scratch. The easy-to-use primitives with pre-programmed defaults provide quick production of graphical interfaces. For example, note the simplicity of a basic HelloWorld program in *Espresso!* :

```
card helloWorld {
       Textbox myText;
       myText.setText("Hello World!");
}
```

If, as above, no other parameters are specified, *Espresso!* will center this black text on a white card, and add the card to the a default stack. A typical Java program would take approximately 10 lines to create the resulting *Espresso!* applet below:

Espresso! simplifies algorithmic tasks such as for loops. The code below will print out a series of statements "number 1" "number 2" etc. to the center of the applet:

```
card someText {
       number i = 0;
       While(i <10) {
              print("number "+i);
              i = i + 1;
       }
}
```

5. Object-Oriented:

The card and stack analogy lends itself to an inherently object-oriented design. By grouping graphical and textual elements into card objects, *Espresso!* familiarizes students with the basics of object-oriented design in an intuitive and logical manner.

6. Portable:

Espresso! code compiles to create Java applets, exploiting the renowned portability of Java. *Espresso!* is as transportable as any standard Java application, and viewable in all Java-enabled web browsers.

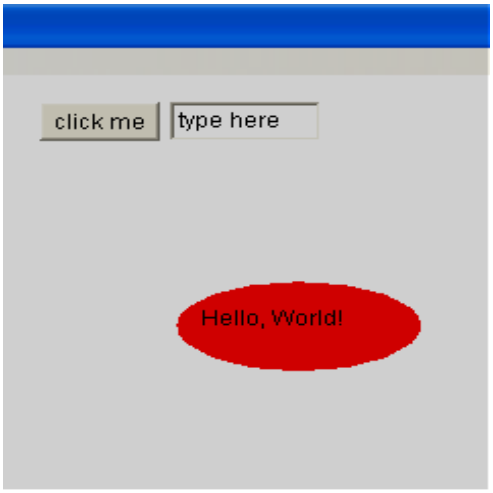
8. Educational:

A goal of *Espresso!* is to provide a spring board to the world of Java programming. Its instructional intentions place education at the heart of the language. *Espresso!* makes a great first language for any child or inexperienced programmer!

Chapter 2

Tutorial

So you want to write an *Espresso!* program? This section will get you acquainted with some of the basics of the language by walking you through a simple but snazzy example. We're going to have you draw an oval on the screen that initially says "Hello, World!" You'll also put a TextBox and a Button above the oval, so that when you type something different in the box and click the button, the text in the oval will change too. This is what we're aiming for:



Step 1: Declare a new card. Evey *Espresso!* program is encapsulated within what is known as a "card". The first line of any program you write should tell the *Espresso!* translator that you are about to declare a card, and what the card's name will be. The code that follows should then be encapsulated within a set of curly braces. We'll call our example snazzyHelloWorld, so that declaration should look just like this:

```
card snazzyHelloWorld {
```

Step 2: Declare your objects. You can think of an Object as any 'thing' that you want to get drawn on the screen. In this case, it shouldn't be hard to convince yourself that you are going to need an Oval, a TextBox, and a Button. The tricky Object in this particular example is the text that gets drawn in the middle of the oval. If you want text to appear on something that doesn't inherently *need* text, then you must declare a String object for the text to appear as an entity by itself. For example, a button inherently *needs* text to tell you what it is there for; so you don't need a String Object for the "click me" text. A TextBox *needs* text because its sole purpose is to deal with text input; so the "type here" you see above does not need its own String object. An Oval, however, will not always have text drawn in the middle of it; and so you must declare a String Object to coexist with the oval.
Every Object you declare must have a name associated with it, to differentiate it from other objects in your card. *Espresso!* will try to recognize any word starting with a capital letter as the **type** of an object, and any word starting with a lowercase letter as a **name**. Finally, you should know that ***every*** line of code you write in espresso must end with a semi-colon. Our declarations will look like this:

```
Button changeTextButton;
```

```
TextBox newTextBox;
Oval justAnOval;
String myTextObject;
```

Step 3: Assign Initial Values. Now that we have declared Objects to be drawn on the screen, we need to assign some attributes to those Objects. In our case, we want the Button to say "click me", the TextBox to start off saying "type here", and the String to start off saying "Hello, World!" In *Espresso!*, you use a what is known as a **method** to make this sort of assignment. Every Object that can handle some sort of text has a method called "setText". You can use the setText method on any object you have already declared. You must call your objects by their **name**. The syntax should be clear from this example:

```
changeTextButton.setText("click me");
newTextBox.setText("type here");
myTextObject.setText("Hello, World!");
```

Step 4: Associate an action with your Button. In our example, we want the button – when pressed – to grab the text from the TextBox (newTextBox) and shove it into the String (myTextObject). We achieve this by using another **method** that retrieves the text from any TextBox. The action(s) we want to associate with our button must be stated within curly braces. The code within the curly braces will be executed only when the button is pressed, and it will execute *every single time* the button is pressed. It works just like this:

```
changeTextButton action {
        myTextObject.setText( newTextBox.getText() );
}
```

Step 5: Tell the Oval and String where they're supposed to go. *Espresso!* will handle placement of all Button and TextBox Objects, but need you to tell it where to put the more artistic elements such as Oval and String. Again, you achieve this by using a method to set the X and Y coordinates. Think of the screen like a graph; the origin is (0,0) in the upper-left-hand corner; the X coordinates increase as you go right; the Y coordinates increase as you go down. A standard *Espresso!* card is 300 x 300. *Espresso!* requires you to set each X and Y coordinate independently. We'll also slip in a little trick here – you can change the color of any object by simply using the "setColor" method.

```
justAnOval.setX(150);
justAnOval.setY(150);
justAnOval.setHeight(50);
justAnOval.setWidth(120);
justAnOval.setColor(red);
myTextObject.setX(160);
myTextObject.setY(150);
```

Step 6: The Finishing Touches. You've now written all the code necessary to implement our snazzyHelloWorld program! To finish it off, you need to:

- Close the program with a curly brace '}'. This closing brace gets paired with the opening brace you typed next to the card declaration at the very top of the program. *Espresso!* needs this to know that you are done writing the code.
- Save your program as '<*somefilename*>.esp'

- Make sure ‘javac’ can be found in your CLASSPATH, and type the following at the UNIX command prompt:

  $ esp <*somefilename*>.esp
- *Espresso!* will automatically translate your program to a Java Applet, compile it, and place the whole program in your ~/public_html directory as a world-readable website. You can then see the program you created by pointing any Java-Enabled Browser to your web address, at the subdirectory “/<*somefilename*>/”

This tutorial was intended to acquaint you with some of the basic functionality of the *Espresso!* language. Once you’re comfortable with the code presented here, please explore the more comprehensive Language Reference Manual to learn about some of the more advanced features possible with *Espresso!*

Chapter 3

Language Reference Manual

3.1 Introduction

This manual describes the *Espresso!* language. *Espresso!* programs must be written on a UNIX platform.

3.2 Program Structure

An espresso program consists of a 'card':

```
card myCard {
        statement;
        statement;
        …
        statement;
}
```

A card can be defined as a window or layout of an applet. Each card will consist of a group of statements.

Cards are an abstraction to help the programmer visualize the concept of an applet. Only one card may be created per program. in a window at a time, and are stored as a stack. Statements are described in the proceding sections.

3.3 Lexical Conventions

3.3.1 Tokens

There are classes of tokens: identifiers, keywords, string literals, operators, and other separators. Spaces, tabs, newlines and comments separate tokens but are otherwise ignored.

3.3.2 Comments

The characters /* indicate the start of a comment, which terminates with the characters */. Nesting of comments is not supported, nor is commenting within other tokens.

3.3.3 Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter. Any successive character may be either a letter (uppercase or lowercase), digit (0..9), or the underscore '_' character.

Identifiers are case sensitive, for instance "ABC" is different from "abc". An identifier can be arbitrarily long. *Espresso!* places no explicit limit on identifier length, however they cannot exceed the available memory of the machine where the program is being developed.

3.3.4 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
card              while       if          object      color   action
else              break       number      string      print   setBackground
disableLayout     pause
```

3.3.5 Numbers

A number consists of a string of digits with an optional sign '-' and an optional decimal point '.'. All built-in mathematical operations performed on numbers assume base-10 format.

3.3.6 Strings

A string is a sequence of characters enclosed by double quotes: "*string*". A double quote inside the string is indicated by two consecutive double quotes. The second double quote is ignored in the final token.

3.3.7 Colors

A color is a selective string that denotes an *Espresso!* color. Supported colors include black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow. Any combination of these may be used to create other colors.

3.3.8 Other tokens

- Spatial delimiters:

( ) { } [ ] ; : , .

- Mathematical operators:

+ - * / %

- Boolean operators:

> < ! == >= <= !=

- Assignment operator:

=

3.3.9 Types

This language has minimal types to ensure simplicity and ease of use while resembling Java type syntax. Memory does not need to be explicitly allocated for typed variables and they can be created at anytime throughout the program. Implemented types are as follows:

- number : 64 bit IEEE floating format
- string : arbitrarily long array of Unicode characters
- object : see table 3.3.1 for object types and descriptions. (These can be changed by editing the runtime library files).
- method : see table 3.3.1 for object methods. (These can be changed by editing the runtime library files.)
- color: see section 3.3.7 for supported colors.

Object	Description	Method		
		Name	Return Type	Parameter Type
TextBox	A box where strings are displayed and/or modified.	textSet	void	string
		getText	string	void
		setWidth*	void	number
		getWidth*	number	void
		setHeight*	void	number
		getHeight*	number	void
		setX*	void	number
		getX*	number	void
		setY*	void	number
		getY*	number	void
CheckBox	A check box	setWidth	void	number
		getWidth	number	void
		setHeight	void	number
		setX*	void	number
		getX*	number	void
		setY*	void	number
		getY*	number	void
		getHeight*	void	number
Button	a button complete with ActionListener Capabilities	setText	void	string
		getText	string	void
		setX*	void	number
		getX*	number	void
		setY*	void	number
		getY*	number	void
		setWidth*	void	number
		getWidth*	number	void
		setHeight*	void	number
		getHeight*	void	number
		action	void	void
Image	an image from a given	setSource	void	string

	URL	getSource	string	void
		setX*	void	number
		getX*	number	void
		setY*	void	number
		getY*	number	void
Oval	an oval shape	setColor	void	color
		setX*	void	number
		getX*	number	void
		setY*	void	number
		getY*	number	void
		setWidth	void	number
		getWidth	number	void
		setHeight	void	number
		getHeight	void	number
Rectangle	a rectangle shape	setColor	void	color
		setX*	void	number
		getX*	number	void
		setY*	void	number
		getY*	number	void
		setWidth	void	number
		getWidth	number	void
		setHeight	void	Number
		getHeight	void	Number
String	a string to be printed	setText	void	string
		setX	void	number
		setY	void	number

Figure 3.3.1: Object Types and Descriptions

* denotes disable layout unique functions (see section 3.5.9)

3.4 Expressions

3.4.1 Primary Expressions

Primary expressions consist of identifiers, numbers, colors, strings, and objects; boolean operators, mathematical operators, and the assignment operator. Boolean operators can be used on identifiers, numbers, strings, and objects. Mathematical operators can be used on numbers and identifiers that represent numbers. The assignment operator can be used to associate an identifier with a number, string or object.

Expressions are used in assignments to create and modify variables and objects. They are used in loops as test cases for further actions and method calls.

3.4.2 Identifiers

An identifier or name can refer to a variety of things. Variables, colors, and objects are labeled with identifiers. Any identifier is limited to the scope in which it is defined, following Java scooping conventions.

Identifiers are strings beginning with an ALPHA character (letters a-z, A-Z, or _) and containing an unspecified length string of alpha-numeric characters and the underscore character.

An identifier cannot have the same spelling as a keyword or Boolean literal. Identifiers are case sensitive; two identifiers are the same only if they have the same string of ASCII values.

3.4.3 Spatial Delimiters

The special delimiters in Espresso are fairly intuitive. See details and descriptions in table 3.4.1 below.

Delimiter(s)	Name(s)	Description/Usage
( )	Parentheses	Used around parameter lists in a method call or object declaration.
{ }	Braces	Used in algorithmic loops or to facilitate scoping.
[ ]	Brackets	Used to in array declarations or assignments
;	Semicolon	Must be at the end of each statement in the card
,	Comma	Used to separate parameters in a parameter list.
.	Dot	Used to indicate decimal number notation (ex. `5.7`) and in modify object statements to denote parameters of an object (ex. `MyButton.setColor`)

Table 3.4.1: Spatial Delimiters Description and Usage

3.4.4 Boolean Operators

Boolean expressions are composed of identifiers, numbers or strings. Boolean operators evaluate to 0 or 1 (for details see table 3.4.2 below).

The values that are compared by the Boolean operators must be numbers or strings, or variables that represent a numerical or string value.

When evaluating string comparisons strings with a higher alphabetical precedence evaluate higher. For example, the character ‘A’ has a higher value than ‘Z’. Uppercase letters take a higher precedence than lowercase letters. Symbols take the lowest precedence and are ranked in order of their ASCII values.

Symbol	Operator Name	Usage	Description
>	Greater than	A > B	Returns 1 A is greater than B, and zero otherwise.
<	Less than	A < B	Returns 1 if A is less than B, and zero otherwise.
>=	Greater than or equal to	A >= B	Returns 1 A is greater than B or if their values are equivalent, and zero otherwise.

<=	Less than or equal to	A <= B	Returns 1 A is less than B or if their values are equivalent, and zero otherwise.
==	Equal to	A == B	Returns 1 only if A and B are equivalent. Returns zero otherwise.
!=	Not equal to	A != B	Returns 1 only if A and B are not equivalent. Returns zero otherwise.

Table 3.4.2: Boolean Operators and Usage (A and B above are identifiers representing literals, numbers, or strings)

3.4.5 Mathematical Operators

Espresso supports five basic mathematical operations, in order of precedence: modular arithmetic, multiplication, division, addition, and subtraction. When used, the mathematical operators and their arguments must be separated by spaces

Mathematical Operators are found in expressions that contain numbers and are formed from ASCII characters. The grammar ensures that the precedence levels will be maintained. Balanced parentheses may be used to explicitly state the order in which an expression must be evaluated. The most internal statements with parentheses will be evaluated first.

Typically the result of a mathematical operation is assigned to a variable or used in a Boolean expression. See table 3.4.3 below for operator details.

Operator	**Description**	**Usage**	**Precedence Level (1 being highest)**
%	Modulo	A % B	1
*	Multiplication	A * B	2
/	Division	A / B	2
+	Addition	A + B	3
-	Subtraction	A – B	3

Table 3.4.3: Mathematical Operators and Usage (A and B above are identifiers representing numbers)

3.4.6 Logical Operators

Logical operators take Boolean expressions as operators. There are three logical operators : **NOT, AND,** and **OR**, denoted '!' , "&&"; and "||" respectively.

NOT has the highest precedence among the operators, and **OR** has the lowest. Logical operators are used in expressions only in loops (if, do, while) and not in assignment statements. Expressions involving logical operators evaluate to 0 or 1.

Expressions involving logical operators are greedy. For example, if the first Boolean expression A in (A && B) evaluates to false, the value of B will not be considered. Likewise if A evaluates to true in (A || B), the value of B will not be considered. !A will evaluate to 1 if A is 0, and will evaluate to 0 if A is not equal to 0.

3.4.7 Assignment operator

The assignment operator, '=' is used to assign values or other variables to identifiers.

3.5 Statements

Statements are composed of keywords, expressions and delimiters. They create the functionality of the language. Statements in *Espresso!* consist of while and do loops, if statements, break statements, declarations, assignments, method calls, print statements, modify object statements, and paint statements. Nesting of loops is allowed and encouraged in the language. Scoping within all statements is global in *Espresso!*. In this section, values in <> represent required fields, and values in bold represent required keywords and punctuation.

3.5.1 While Statement

The while statement, denoted while_stmt, must be of the form:

```
while ( <expression> )
{
  <statement>;
}
```

This statement is constructed like a standard while loop in Java. While the expression in the parentheses evaluates to true or 1, the statement in the braces will be executed.

3.5.2 If Statement

The if statement, denoted if_stmt, must be of the form:

```
if ( <expression> ) {
      <statement>
}
else ( expression ) {
      statement
} (0 or more times)
```

The statements, expressions, and delimiters in italics are optional. The if part of the statement is required. There may be zero or one "else" phrases of the statement. The braces after the if and else clauses are optional if only one statement follows the expression.

3.5.3 Break Statement

The break statement, of the form: **`break;`** will break out of the current while or if statement.

3.5.4 Declaration

A declaration assigns a value or set of values to an identifier of a certain type. Espresso is strongly typed in that an identifier declared as one type cannot be modified to represent a value of a different type. Declarations cannot be performed in an action block. There are several forms of acceptable declarations:

- Number declaration:

```
number <identifier> =  <number or identifier> ;
```

- string declaration:

```
string <identifier> = <string or identifier> ;
```

- object declaration:

```
<object> <identifier> ;
```

- color declaration:

```
color <identifier> = <color from table 3.3.2> (+ <color from table 3.3.2>)*;
```

3.5.5 Assignment

Assignments are in one of the following forms:

```
<identifier> = <mdas operator> ;
```

The first form is associated with a single declaration of an number, string, or color. An mdas operator is an element of the grammar that involves one or more identifiers, colors, numbers, and strings that may include the mathematical operators. mdas stands for the end of the standard mathematical precedence acronym PEMDAS because only the operators multiply, divide, add, subtract and mod may be involved. Assigments are used to modify or replace the value denoted by a previously defined identifier, and the operator must be of the correct type for assignment.

3.5.6 Object Modifications/Accessors

Object assignments work differently than number, string, and color assignments. They are in the following form:

```
<identifier>.<modify_method>( <parameter> );
```

An object modify method must be a valid method for the given identifier, and the parameter must be of a valid type. If object assignments have a return type, they are object accessor methods. In this case the form will be as follows:

```
<identifier1> = <identifier2>.<accessor_method>( <parameter> );
```

anad identifier1 must be of the same type as the return type of the accessor functions. For a list of the object types and their valid methods, return types, and parameter types, see table 3.3.1.

3.5.7 Print Statement

The print() function takes one or more parameters and prints them one by one to the center of the applet screen. The parameter type must be a string. It is in the following form:

```
print <one or more strings> ;
```

3.5.8 The Set Background Statement

The **setBackground** statement sets or resets the background of the current card. It takes one color argument to which the background is to be set in the following form:

```
setBackground ( <color> ) ;
```

3.5.9 Disable Layout Statement

The disable layout statement is used to override the default Layout Manager (flow layout). If this statement is used, it must be the first statement in the card. It is in the following form:

```
disableLayout ;
```

If the disable layout statement is not used, the default flow layout of an applet will be implemented. If it is used, the user has full control over the positions of the elements of the applet. When disable layout is used, the user can use the disable layout unique functions (see table 3.3.1), and if these methods are not set all elements default to the (0,0) position of the applet.

3.5.10 Pause Statement

The pause statement will "pause" (in Java sleep) for the given number of milliseconds. The form of the statement is as follows:

```
pause( <number> ) ;
```

3.5.11 Action Statement

This statement is used in event handling. An action statement can be performed on any object that has the action option. A list of which objects have this option is contained in table 3.3.1. The form is as follows

```
<identifier> action {
   <statement>*
}
```

When this statement is used, the code inside the action braces will be placed in the action block of the final applet and will act as the event handling procedure on the object represented by the given identifier.

Chapter 4

Project Plan

4.1 Planning, specification, development and testing

Following the completion of *Espresso!*'s white paper, the *Espresso!* development team began the planning and specification process. Our team's first task was to outline our overall goals in a concrete way. This was done by developing a set of JAVA applets, which encompassed the desired functionality of *Espresso!* These applets provided a template which we worked to dynamically replicate.

We spent several days outlining the various components of the interpreter's structure, deciding how these parts would work together, and delegating responsibility for each module. *Espresso!*'s architecture yielded an obvious division of labor between the grammar, parser, and tree walker and the translator backend, which includes the run-time library and class definitions and the actual JAVA code generation.

The completion of *Espresso!*'s language reference manual helped in delineating the various tasks. While each team member worked with a slightly different piece of the puzzle, open communication and discussion kept everyone abreast of our overall progress.

Our work model was incremental development, whereby we would start with a reduced problem and work towards a solution, building on each small step. This process went hand in hand with testing, which took much of the burden off of the integration phase. System testing and debugging was performed by the whole team. We wrote applets and tested them on eachothers code to find bugs.

4.2 Project Timeline and Log

September 23	White Paper submitted
3rd and 4th week of September	Writing of goal applets
3rd and 4th week of October	Architecture outline and development
October 28	Language reference manual submitted
2nd week of November	Antlr grammar defined
3rd week of November	Antlr parser
4th week of November	AST walker
1st week of December	Backend
2nd week of December	Integration and testing; grammar, parser, and walker finalized

December 10	Project Presentation
2nd and 3rd week of December	Backend finalized; Testing and documentation
December 18	Final report

4.3 Team Responsibilities

Erin	Front end, Back end, testing, and documentation
Aaron	Back end, testing, and documentation
Phil	Team leader, Back end, testing, and documentation
Joya	Front end, Back end testing, and documentation

Please see Appendix B for detailed breakdown of responsibilities.

4.4 Programming Style Guide

4.4.1 General Guidelines

- All code will be clear and tested througout writing. After initial implementation of new code it must be tested with all other classes.
- All design choices will be explained to and discussed with the other group members before implementation to get their opinions and facilitate interoperability.
- Each person will save backups on his or her laptop before a change is made.
- All group members are in the UNIX group Espresso and will store updated copies in a "current" directory with files chgroup Espresso and chmod 770.
- All test applets will reside in the current/Test directory.

4.4.2 Antlr Coding Style

- Antlr source files should use the ".g" postfix.

- If a rule is short, it will be written all on one line.
- For longer rules with multiple choices, each choice is on its own line.
- Token names are in upper case, and variables are in lower case.
- Comments are made inline with the "//" characters.

4.4.3 Java Coding Style

- For java source files, use ".java" postfix. For java byte code, use ".class" postfix
- The java header includes import statements as well as block comments outlining the purpose of the class and the author
- Method names should be descriptive, and include javadoc-style comments
- Indentation should be used for loops, if statements, methods, and expressions which do not fit on a single line
- Use get/set methods to access data within a class

4.5 Development Environment

All code was developed independently using applications like Sun ONE studio, emacs, and pico. Antlr version 2.7.2 was used for Antlr code. For integration and testing, we used CUNIX servers (JDK 1.4.2) as well as Internet Explorer 5.2 and Safari 1.0.

Chapter 5

Architectural Design

5.1 Overview

Four main components exist in the *Espresso!* Architecture: The lexer/parser, the walker, the backend API, and the java Runtime Library. An *Espresso!* .esp file passes through the lexer/parser to ensure there are no syntax errors, and an AST is created. The backend API and walker work together to propagate the AST, checking for semantic errors. EspressoToJava.java, part of the backend API, and the walker interoperate to output user-defined code into the java applet template. The output .java file is then compiled and run. The propagation through the Espresso Architecture is encapsulated in the esp script which takes an input file and possible output file as parameters, performs all compilations and executes the resulting applet.

5.2 The *Espresso!* lexer/parser:

The code for the *Espresso!* lexer and parser is located in grammar.g . The grammar attempts to preserve java syntax whenever possible to facilitate a swift learning curve for developers of all levels. ANTLR will catch syntactical errors. The intuitive flow of the grammar was chosen to server the primary purposes of *Espresso!*. As a tool for advanced developers to quickly create applets, it is necessary that the flow resembles that of java. As an educational tool, removing the confusing elements of java syntax is key to ensure simplicity. The lexer/parser was implemented by Joya Zuber.

5.3 The *Espresso!* walker:

The *Espresso!* walker interoperates with the backend API. Error checking is performed directly in the walker using object types created by the API suite, the Espresso Symbol table and the Espresso Object Type table. The walker prints appropriate code to the generated java file. The walker was implemented by Joya Zuber and Erin Adelman.

5.4 The *Espresso!* backend API:

The backend API contains classes for the *Espresso!* data types EspressoNumber, EspressoString, EspressoColor, and EspressoObject. EspressoNumber represents a double in java. EspressoString represents a String in java. EspressoColor is a unique data type that contains information about colors which will be later translated to java. The EspressoObject class creates a generic EspressoObject which can be any of the AWT constructs described in the runtime library. The API also contains the EspressoToJava class which is primarily responsible for final code generation. This class is used in the walker and directs the user generated code to the appropriate block of the applet template. The EspressoSymbolTable.java file in the API emulates the symbol table used by the walker for sematic analysis. At the instantiation of the symbol table, the supported colors of *Espresso!* are loaded in. As the walker encounters new declarations, it will store the identifiers with the appropriate typenames in the symbol table so it can be used later in assignment verification. The EspressoObjectTypeTable.java file contains information based on the runtime library description file concerning valid object types, their corresponding methods, parameter types, and return types. It is referenced by the walker in object

declaration, object modification, and object action attempts by the *Espresso!* writer. The backend API was a group effort. See Appendix B for the authors of each file.

5.5 The *Espresso!* Runtime Library:

One of the most unique elements of *Espresso!* is its complicated yet elegant implementation of a runtime library. It became clear in the initial phases of development that not all resolutions should be performed at compile time. The library is also a vital element of the scalability of *Espresso!*. The runtime library consists of a library description file and a suite of java classes. Below is an example of a construct entry in the library description file:

```
TextBox::EspressoTextBoxRT(this) {
void textSet::setText( string )
string getText( void )
void setSize( number )
number getSize( void )
void action ( void )
}
```

Hence the format is as follows:

```
<Espresso Object Name>::<RTL Object Name>(<Java parameter list>) {
<Espresso return type> <Espresso/Java method name> (<Espresso parameter type>)
/* possible mapping of method names to support unique Espresso methods */
<Espresso return type> <Espresso method name>::<Java method name>
                                                    (<Espresso parameter type>)
}
```

The format of the description file is crucial for the scalability of *Espresso!*. The EspressoObjectTypeTable reads this runtime library description file, and generates errors used by the walker in semantic analysis. The runtime library also includes Java wrapper classes for each construct as detailed in the description file. These classes are actually used by the output file at runtime as objects are declared, modified, and manipulated. An experienced Java developer can easily add new RT classes and the appropriate entry to the RTL description library, and the *Espresso!* translator will incorporate them into the language. The Runtime library was implemented by Philip Coakley and Aaron Bauman.

Chapter 6

Test Plan

6.1 Representative Source Code:

Included below are three representative programs including the .esp source, the generated java code, and a screen shot of the applet.

6.2 Cool *Espresso!* Color Applet

The Color applet is in a sense a color calculator. The user can push the buttons in the applet to add more of the specified color to the background, and can reset. This applet exhibits the unique color functionality of *Espresso!*.We felt color was an important area in which to ensure robust functionality. Previous versions of this applet with less functionality were created to test all elements of *Espresso!*'s color capabilities.

6.2.1 ColorApplet.esp

```
card colorCalculator{

        color curr=black;
        number w=45;
        number spacer=200;
        spacer=16;

        Button redB;
        Button blueB;
        Button greenB;
        Button whiteB;
        Button blackB;
        Button resetB;

        pause 100;

        redB.setText("red");
        redB.setX(spacer);
        redB.setWidth(w);
        redB.setY(20);

        blueB.setText("blue");
        blueB.setX(w+2*spacer);
        blueB.setWidth(w);
        blueB.setY(20);

        greenB.setText("green");
        greenB.setX(2*w+3*spacer);
        greenB.setWidth(w);
        greenB.setY(20);

        whiteB.setText("white");
        whiteB.setX(spacer);
        whiteB.setWidth(w);
        whiteB.setY(50);

        blackB.setText("black");
        blackB.setX(w+2*spacer);
        blackB.setWidth(w);
        blackB.setY(50);

        resetB.setText("reset");
        resetB.setX(2*w+3*spacer);
        resetB.setWidth(w);
```

```
        resetB.setY(50);

        Rectangle swatch;

        swatch.setWidth(150);
        swatch.setHeight(70);
        swatch.setX(25);
        swatch.setY(100);
        swatch.setColor(curr);
        setBackground(white);

        redB action{

                curr=curr+red;
                swatch.setColor(curr);
        }

        blueB action{

                curr=curr+blue;
                swatch.setColor(curr);
        }

        greenB action{

                curr=curr+green;
                swatch.setColor(curr);
        }

        blackB action{

                curr=curr+black;
                swatch.setColor(curr);
        }

        whiteB action{

                curr=curr+white;
                swatch.setColor(curr);
        }

        resetB action{

                curr=white;
                swatch.setColor(curr);
        }
}
```

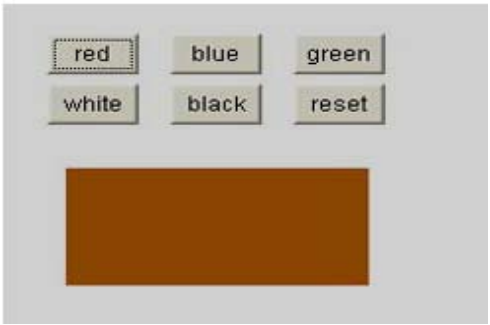
6.2.2 espresso.java (output of ColorApplet.esp)

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.net.*;
import java.util.*;
import RTL.*;


public class espresso extends Applet implements Runnable, ActionListener {


    Graphics graphicsBuffer = null;
    Image imageBuffer = null;
    Thread conductor = null;
    LinkedList listOfEspressoShapes = null;
    public boolean __always_false=false;

    //all declarations specified by programmer
    EspressoColorRT curr;
    double w;
    double spacer;
    EspressoButtonRT redB;
    EspressoButtonRT blueB;
    EspressoButtonRT greenB;
    EspressoButtonRT whiteB;
```

```
    EspressoButtonRT blackB;
    EspressoButtonRT resetB;
    EspressoRectRT swatch;


    public void init () {

        listOfEspressoShapes = new LinkedList();
        curr= new EspressoColorRT(new Color(0,0,0), 1);
        w=45;

        spacer=200;

        redB = new EspressoButtonRT(this);
        blueB = new EspressoButtonRT(this);
        greenB = new EspressoButtonRT(this);
        whiteB = new EspressoButtonRT(this);
        blackB = new EspressoButtonRT(this);
        resetB = new EspressoButtonRT(this);
        swatch = new EspressoRectRT(listOfEspressoShapes, this);
    } // end of init()



    public synchronized void paint ( Graphics g ) {

        g = getGraphics();
        imageBuffer = createImage(getWidth(), getHeight());
        graphicsBuffer = imageBuffer.getGraphics();
        graphicsBuffer.setColor(getBackground());
        graphicsBuffer.fillRect(0, 0, getWidth(), getHeight());
        for(int __i__ = 0; __i__ < listOfEspressoShapes.size(); __i__++) {
            ((EspressoShapeRT)listOfEspressoShapes.get(__i__)).draw(graphicsBuffer);    }
        g.drawImage(imageBuffer, 0, 0, this);
    } // end of paint




    public void actionPerformed ( ActionEvent evt ) {
       if ( evt.getSource() == redB.getElement() ) {
        curr=curr.calcColor(new Color(255,0,0), 1);
        swatch.setColor(curr.getColor());
       repaint();
}      if ( evt.getSource() == blueB.getElement() ) {
        curr=curr.calcColor(new Color(0,0,255), 1);
        swatch.setColor(curr.getColor());
       repaint();
}      if ( evt.getSource() == greenB.getElement() ) {
        curr=curr.calcColor(new Color(0,255,0), 1);
        swatch.setColor(curr.getColor());
       repaint();
}      if ( evt.getSource() == blackB.getElement() ) {
        curr=curr.calcColor(new Color(0,0,0), 1);
        swatch.setColor(curr.getColor());
       repaint();
}      if ( evt.getSource() == whiteB.getElement() ) {
        curr=curr.calcColor(new Color(255,255,255), 1);
        swatch.setColor(curr.getColor());
       repaint();
}      if ( evt.getSource() == resetB.getElement() ) {
        curr= new EspressoColorRT(new Color(255,255,255), 0);
        swatch.setColor(curr.getColor());
       repaint();
}    } // end of actionPerformed


    public void start() {
        if(conductor == null ) {
            conductor = new Thread(this, "App");
            conductor.start();
        }
    } // end of start
```

```
    public void stop() {
    } // end of stop


    public void run () {

        spacer=16;

        redB.setText("red");
        redB.setX(spacer);
        redB.setWidth(w);
        redB.setY(20);
        blueB.setText("blue");
        blueB.setX((w + (2 * spacer)));
        blueB.setWidth(w);
        blueB.setY(20);
        greenB.setText("green");
        greenB.setX(((2 * w) + (3 * spacer)));
        greenB.setWidth(w);
        greenB.setY(20);
        whiteB.setText("white");
        whiteB.setX(spacer);
        whiteB.setWidth(w);
        whiteB.setY(50);
        blackB.setText("black");
        blackB.setX((w + (2 * spacer)));
        blackB.setWidth(w);
        blackB.setY(50);
        resetB.setText("reset");
        resetB.setX(((2 * w) + (3 * spacer)));
        resetB.setWidth(w);
        resetB.setY(50);
        swatch.setWidth(150);
        swatch.setHeight(70);
        swatch.setX(25);
        swatch.setY(100);
        swatch.setColor(curr.getColor());
        setBackground(new Color(255,255,255));

    } // end of run
} // end of espresso
```

6.2.3 index.html



6.3 *Espresso!* Calculator Applet

This applet emulates a simple calculator. We chose to use this for testing because it tests the mathematical capabilities of the language as well as layout management and object manipulation. Initially, we tested each of these capabilities individually, but found that a calculator was an excellent (and useful!) way to combine them. The calculator is an example of an applet where “disableLayout;” was used. This affords the user complete control over the location of the objects in the applet.

6.3.1 Calcamajig.esp

```
card calcLikeARealMan {

        number currVal = -1;
        number currOp = 0;

        Button button1;
        Button button2;
        Button button3;
        Button button4;
        Button button5;
        Button button6;
        Button button7;
        Button button8;
        Button button9;
        Button button0;
        Button buttonPlus;
        Button buttonMinus;
        Button buttonMult;
        Button buttonDiv;
        Button buttonClear;
        TextBox buttonLCD;

        setBackground(white);

        button0.setText("0");
        button1.setText("1");
        button2.setText("2");
        button3.setText("3");
        button4.setText("4");
        button5.setText("5");
        button6.setText("6");
        button7.setText("7");
        button8.setText("8");
        button9.setText("9");
        buttonPlus.setText("+");
        buttonMinus.setText("-");
        buttonMult.setText("*");
        buttonDiv.setText("/");
        buttonClear.setText("C");
        buttonLCD.setText("0");

        button1.setX(10);
         button1.setY(40);
        button2.setX(60);
        button2.setY(40);
         button3.setX(110);
        button3.setY(40);
        button4.setX(10);
        button4.setY(80);
        button5.setX(60);
        button5.setY(80);
        button6.setX(110);
        button6.setY(80);
        button7.setX(10);
        button7.setY(120);
        button8.setX(60);
        button8.setY(120);
        button9.setX(110);
        button9.setY(120);
        button0.setX(10);
        button0.setY(160);
        buttonClear.setX(110);
        buttonClear.setY(160);
        buttonPlus.setX(160);
        buttonPlus.setY(40);
        buttonMinus.setX(160);
        buttonMinus.setY(80);
        buttonMult.setX(160);
        buttonMult.setY(120);
        buttonDiv.setX(160);
        buttonDiv.setY(160);

         button1.setWidth(50);
         button2.setWidth(50);
```

```
button3.setWidth(50);
button4.setWidth(50);
button5.setWidth(50);
button6.setWidth(50);
button7.setWidth(50);
button8.setWidth(50);
button9.setWidth(50);
button0.setWidth(50);
buttonClear.setWidth(50);
buttonPlus.setWidth(30);
buttonMinus.setWidth(30);
buttonMult.setWidth(30);
buttonDiv.setWidth(30);

button1.setHeight(30);
button2.setHeight(30);
button3.setHeight(30);
button4.setHeight(30);
button5.setHeight(30);
button6.setHeight(30);
button7.setHeight(30);
button8.setHeight(30);
button9.setHeight(30);
button0.setHeight(30);
buttonClear.setHeight(30);
buttonPlus.setHeight(30);
buttonMinus.setHeight(30);
buttonMult.setHeight(30);
buttonDiv.setHeight(30);

buttonLCD.setX(10);
buttonLCD.setY(5);
buttonLCD.setWidth(180);
buttonLCD.setHeight(30);

button0 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 0; }
                  else if(currOp == 2) {
                        currVal = currVal - 0; }
                  else if(currOp == 3) {
                        currVal = currVal * 0; }
                  else if(currOp == 4) {
                        currVal = currVal / 0; }

                currOp = 0;
        }
        else {
                currVal = 0;
        }

buttonLCD.setText(""+currVal);
}

button1 action{
         if(currOp != 0) {
                 if(currOp == 1) {
                         currVal = currVal + 1; }
                 else if(currOp == 2) {
                         currVal = currVal - 1; }
                 else if(currOp == 3) {
                         currVal = currVal * 1; }
                 else {  /* currOp == 4 */
                         currVal = currVal / 1; }
                 currOp = 0;
         }

         else {
                 currVal = 1;
         }
buttonLCD.setText(""+currVal);
}

button2 action{
         if(currOp != 0) {
```

```
                if(currOp == 1) {
                        currVal = currVal + 2; }
                else if(currOp == 2) {
                        currVal = currVal - 2; }
                else if(currOp == 3) {
                        currVal = currVal * 2; }
                else {  /* currOp == 4 */
                        currVal = currVal / 2; }
                currOp = 0;
        }
        else {
                currVal = 2;
        }
buttonLCD.setText(""+currVal);
}

button3 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 3; }
                else if(currOp == 2) {
                        currVal = currVal - 3; }
                else if(currOp == 3) {
                        currVal = currVal * 3; }
                else {  /* currOp == "/" */
                        currVal = currVal / 3; }
                currOp = 0;
        }
        else {
                currVal = 3;
        }
buttonLCD.setText(""+currVal);
}

button4 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 4; }
                else if(currOp == 2) {
                        currVal = currVal - 4; }
                else if(currOp == 3) {
                        currVal = currVal * 4; }
                else {  /* currOp == 4 */
                        currVal = currVal / 4; }
                currOp = 0;
        }
        else {
                currVal = 4;
        }
buttonLCD.setText(""+currVal);
}

button5 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 5; }
                else if(currOp == 2) {
                        currVal = currVal - 5; }
                else if(currOp == 3) {
                        currVal = currVal * 5; }
                else {  /* currOp == 4 */
                        currVal = currVal / 5; }
                currOp = 0;
        }
        else {
                currVal = 5;
        }
buttonLCD.setText(""+currVal);
}

button6 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 6; }
                else if(currOp == 2) {
                        currVal = currVal - 6; }
```

```
                else if(currOp == 3) {
                        currVal = currVal * 6; }
                else {  /* currOp == 4 */
                        currVal = currVal / 6; }
                currOp = 0;
        }
        else {
                currVal = 6;
        }
buttonLCD.setText(""+currVal);
}

button7 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 7; }
                else if(currOp == 2) {
                        currVal = currVal - 7; }
                else if(currOp == 3) {
                        currVal = currVal * 7; }
                else {  /* currOp == 4 */
                        currVal = currVal / 7; }
                currOp = 0;
        }
       else {
        currVal = 7;
       }
buttonLCD.setText(""+currVal);
}

button8 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 8; }
                else if(currOp == 2) {
                        currVal = currVal - 8; }
                else if(currOp == 3) {
                        currVal = currVal * 8; }
                else {  /* currOp == 4 */
                        currVal = currVal / 8; }
                currOp = 0;
        }
        else {
                currVal = 8;
        }
buttonLCD.setText(""+currVal);
}

button9 action{
        if(currOp != 0) {
                if(currOp == 1) {
                        currVal = currVal + 9; }
                else if(currOp == 2) {
                        currVal = currVal - 9; }
                else if(currOp == 3) {
                        currVal = currVal * 9; }
                else {  /* currOp == 4 */
                        currVal = currVal / 9; }
                currOp = 0;
        }
        else {
                currVal = 9;
        }
buttonLCD.setText(""+currVal);
}

buttonPlus action{
       currOp = 1; }

buttonMinus action{
        currOp = 2; }

buttonMult action{
        currOp = 3; }

buttonDiv action{
```

```
                currOp =4; }

        buttonClear action{
                currVal = 0;
                currOp = 0;
                buttonLCD.setText(""+currVal);
        }
}
```

6.3.2 Calcimajig.java (Calcamajig.esp output code)

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.net.*;
import java.util.*;
import RTL.*;


public class calcimajig extends Applet implements Runnable, ActionListener {


    Graphics graphicsBuffer = null;
    Image imageBuffer = null;
    Thread conductor = null;
    LinkedList listOfEspressoShapes = null;
    public boolean __always_false=false;

    //all declarations specified by programmer
    double currVal;
    double currOp;
    EspressoButtonRT button1;
    EspressoButtonRT button2;
    EspressoButtonRT button3;
    EspressoButtonRT button4;
    EspressoButtonRT button5;
    EspressoButtonRT button6;
    EspressoButtonRT button7;
    EspressoButtonRT button8;
    EspressoButtonRT button9;
    EspressoButtonRT button0;
    EspressoButtonRT buttonPlus;
    EspressoButtonRT buttonMinus;
    EspressoButtonRT buttonMult;
    EspressoButtonRT buttonDiv;
    EspressoButtonRT buttonClear;
    EspressoTextBoxRT buttonLCD;


    public void init () {

        listOfEspressoShapes = new LinkedList();
        currVal=(-1);

        currOp=(0);

        button1 = new EspressoButtonRT(this);
        button2 = new EspressoButtonRT(this);
        button3 = new EspressoButtonRT(this);
        button4 = new EspressoButtonRT(this);
        button5 = new EspressoButtonRT(this);
        button6 = new EspressoButtonRT(this);
        button7 = new EspressoButtonRT(this);
        button8 = new EspressoButtonRT(this);
        button9 = new EspressoButtonRT(this);
        button0 = new EspressoButtonRT(this);
        buttonPlus = new EspressoButtonRT(this);
        buttonMinus = new EspressoButtonRT(this);
        buttonMult = new EspressoButtonRT(this);
        buttonDiv = new EspressoButtonRT(this);
        buttonClear = new EspressoButtonRT(this);
        buttonLCD = new EspressoTextBoxRT(this);
```

```
    } // end of init()



    public synchronized void paint ( Graphics g ) {

        g = getGraphics();
        imageBuffer = createImage(getWidth(), getHeight());
        graphicsBuffer = imageBuffer.getGraphics();
        graphicsBuffer.setColor(getBackground());
        graphicsBuffer.fillRect(0, 0, getWidth(), getHeight());
        for(int __i__ = 0; __i__ < listOfEspressoShapes.size(); __i__++) {
            ((EspressoShapeRT)listOfEspressoShapes.get(__i__)).draw(graphicsBuffer);    }
        g.drawImage(imageBuffer, 0, 0, this);
    } // end of paint




    public void actionPerformed ( ActionEvent evt ) {
       if ( evt.getSource() == button0.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (0)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (0)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (0)));

        }/* end if */

        else {

        if(((currOp) == (4))) {

        (currVal)=(((currVal) / (0)));

        }/* end if */

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(0);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
       repaint();
}      if ( evt.getSource() == button1.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {
```

```
        (currVal)=(((currVal) + (1)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (1)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (1)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (1)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(1);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button2.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (2)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (2)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (2)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (2)));

        }/*end of else */

        }/*end of else */

        }/*end of else */
```

```
        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(2);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button3.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (3)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (3)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (3)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (3)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(3);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button4.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (4)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (4)));

        }/* end if */
```

```
        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (4)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (4)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(4);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button5.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (5)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (5)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (5)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (5)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(5);

        }/*end of else */
```

```
        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button6.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (6)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (6)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (6)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (6)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(6);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button7.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (7)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (7)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (7)));

        }/* end if */

        else {
```

```
        (currVal)=(((currVal) / (7)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(7);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button8.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (8)));

        }/* end if */

        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (8)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (8)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (8)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(8);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == button9.getElement() ) {
        if(((currOp) != (0))) {

        if(((currOp) == (1))) {

        (currVal)=(((currVal) + (9)));

        }/* end if */
```

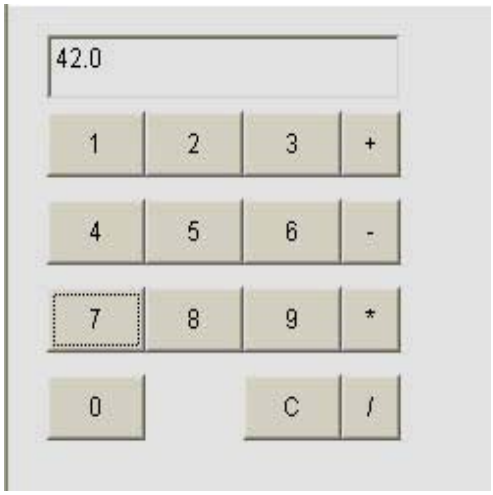
```
        else {

        if(((currOp) == (2))) {

        (currVal)=(((currVal) - (9)));

        }/* end if */

        else {

        if(((currOp) == (3))) {

        (currVal)=(((currVal) * (9)));

        }/* end if */

        else {

        (currVal)=(((currVal) / (9)));

        }/*end of else */

        }/*end of else */

        }/*end of else */

        (currOp)=(0);

        }/* end if */

        else {

        (currVal)=(9);

        }/*end of else */

        buttonLCD.setText(""+(currVal));
        repaint();
}       if ( evt.getSource() == buttonPlus.getElement() ) {
        (currOp)=(1);

        repaint();
}       if ( evt.getSource() == buttonMinus.getElement() ) {
        (currOp)=(2);

        repaint();
}       if ( evt.getSource() == buttonMult.getElement() ) {
        (currOp)=(3);

        repaint();
}       if ( evt.getSource() == buttonDiv.getElement() ) {
        (currOp)=(4);

        repaint();
}       if ( evt.getSource() == buttonClear.getElement() ) {
        (currVal)=(0);

        (currOp)=(0);

        buttonLCD.setText(""+(currVal));
        repaint();
}     } // end of actionPerformed


    public void start() {
        if(conductor == null ) {
            conductor = new Thread(this, "App");
            conductor.start();
        }
    } // end of start



    public void stop() {
    } // end of stop
```

```
public void run () {

    setBackground(new Color(255,255,255));

    button0.setText("0");
    button1.setText("1");
    button2.setText("2");
    button3.setText("3");
    button4.setText("4");
    button5.setText("5");
    button6.setText("6");
    button7.setText("7");
    button8.setText("8");
    button9.setText("9");
    buttonPlus.setText("+");
    buttonMinus.setText("-");
    buttonMult.setText("*");
    buttonDiv.setText("/");
    buttonClear.setText("C");
    buttonLCD.setText("0");
    button1.setX((10));
    button1.setY((40));
    button2.setX((60));
    button2.setY((40));
    button3.setX((110));
    button3.setY((40));
    button4.setX((10));
    button4.setY((80));
    button5.setX((60));
    button5.setY((80));
    button6.setX((110));
    button6.setY((80));
    button7.setX((10));
    button7.setY((120));
    button8.setX((60));
    button8.setY((120));
    button9.setX((110));
    button9.setY((120));
    button0.setX((10));
    button0.setY((160));
    buttonClear.setX((110));
    buttonClear.setY((160));
    buttonPlus.setX((160));
    buttonPlus.setY((40));
    buttonMinus.setX((160));
    buttonMinus.setY((80));
    buttonMult.setX((160));
    buttonMult.setY((120));
    buttonDiv.setX((160));
    buttonDiv.setY((160));
    button1.setWidth((50));
    button2.setWidth((50));
    button3.setWidth((50));
    button4.setWidth((50));
    button5.setWidth((50));
    button6.setWidth((50));
    button7.setWidth((50));
    button8.setWidth((50));
    button9.setWidth((50));
    button0.setWidth((50));
    buttonClear.setWidth((50));
    buttonPlus.setWidth((30));
    buttonMinus.setWidth((30));
    buttonMult.setWidth((30));
    buttonDiv.setWidth((30));
    button1.setHeight((30));
    button2.setHeight((30));
    button3.setHeight((30));
    button4.setHeight((30));
    button5.setHeight((30));
    button6.setHeight((30));
    button7.setHeight((30));
    button8.setHeight((30));
    button9.setHeight((30));
```

```
        button0.setHeight((30));
        buttonClear.setHeight((30));
        buttonPlus.setHeight((30));
        buttonMinus.setHeight((30));
        buttonMult.setHeight((30));
        buttonDiv.setHeight((30));
        buttonLCD.setX((10));
        buttonLCD.setY((5));
        buttonLCD.setWidth((180));
        buttonLCD.setHeight((30));
    } // end of run
} // end of calcimajig
```

6.3.3 index.html

screenshot:



6.4 HelloWorld! Applet

This applet is the applet shown in the presentation. It implements the standard “hello, world!” program with a twist! The user can input the text for the baby to say. We chose to use this applet because it combines the use of 5 different kinds of objects in *Espresso!* (buttons, text boxes, ovals, rectangles, and images). It was an excellent tool in testing the functionality of the runtime library because of the user interaction. We tested the portions of the applet individually (started with a static hello world and implemented the button and text box seperately), and then combined the elements.

6.4.1 HelloWorld.esp

```
card helloWorld {

        Button sayit;
        TextBox tosay;
        Image baby;
        Oval oval;
        Rectangle rect;
        String speech;
        string say="hello";

        number yoffset = 20;
        setBackground(white);
        speech.setText("wassup stephen");
        sayit.setText("click me");
        tosay.setText("type here");
        baby.setSource("http://www.columbia.edu/~adb54/stoned.jpg");
        oval.setColor(red);
        rect.setColor(red);
```

```
        baby.setY(yoffset);
        oval.setX(148);
        oval.setY(91 + yoffset);
        oval.setHeight(53);
        oval.setWidth(120);
        rect.setX(157);
        rect.setY(120 + yoffset);
        rect.setWidth(20);
        rect.setHeight(20);
        speech.setX(160);
        speech.setY(117 + yoffset);
        print speech;

        sayit action{
               say=tosay.getText();
               speech.setText(say);
        }
}
```

6.4.2 espresso.java (output of HelloWorld.esp)

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.net.*;
import java.util.*;
import RTL.*;


public class espresso extends Applet implements Runnable, ActionListener {


    Graphics graphicsBuffer = null;
    Image imageBuffer = null;
    Thread conductor = null;
    LinkedList listOfEspressoShapes = null;
    public boolean __always_false=false;

    //all declarations specified by programmer
    EspressoButtonRT sayit;
    EspressoTextBoxRT tosay;
    EspressoImageRT baby;
    EspressoOvalRT oval;
    EspressoRectRT rect;
    EspressoStringRT speech;
    String say;
    double yoffset;


    public void init () {

        listOfEspressoShapes = new LinkedList();
        sayit = new EspressoButtonRT(this);
        tosay = new EspressoTextBoxRT(this);
        baby = new EspressoImageRT(this, listOfEspressoShapes);
        oval = new EspressoOvalRT(listOfEspressoShapes, this);
        rect = new EspressoRectRT(listOfEspressoShapes, this);
        speech = new EspressoStringRT(listOfEspressoShapes, this);
        say="hello";

        yoffset=(20);

        listOfEspressoShapes.add(speech);
    } // end of init()



    public synchronized void paint ( Graphics g ) {

        g = getGraphics();
        imageBuffer = createImage(getWidth(), getHeight());
        graphicsBuffer = imageBuffer.getGraphics();
        graphicsBuffer.setColor(getBackground());
        graphicsBuffer.fillRect(0, 0, getWidth(), getHeight());
        for(int __i__ = 0; __i__ < listOfEspressoShapes.size(); __i__++) {
            ((EspressoShapeRT)listOfEspressoShapes.get(__i__)).draw(graphicsBuffer);    }
```

```
        g.drawImage(imageBuffer, 0, 0, this);
    } // end of paint



    public void actionPerformed ( ActionEvent evt ) {
        if ( evt.getSource() == sayit.getElement() ) {
         say=tosay.getText();;
         speech.setText(say);
        repaint();
 }    } // end of actionPerformed


    public void start() {
        if(conductor == null ) {
            conductor = new Thread(this, "App");
            conductor.start();
        }
    } // end of start



    public void stop() {
    } // end of stop



    public void run () {

        setBackground(new Color(255,255,255));

        speech.setText("wassup stephen");
        sayit.setText("click me");
        tosay.setText("type here");
        baby.setSource("http://www.columbia.edu/~adb54/stoned.jpg");
        oval.setColor(new Color(255,0,0));
        rect.setColor(new Color(255,0,0));
        baby.setY((yoffset));
        oval.setX((148));
        oval.setY((((91) + (yoffset))));
        oval.setHeight((53));
        oval.setWidth((120));
        rect.setX((157));
        rect.setY((((120) + (yoffset))));
        rect.setWidth((20));
        rect.setHeight((20));
        speech.setX((160));
        speech.setY((((117) + (yoffset))));
    } // end of run
} // end of espresso
```

6.4.3 index.html

screenshot:

Chapter 7

What We Learned

Joya:

The most important thing I learned was a new appreciation of group projects. It was great working with other people because the work could be split up based on personal interests/skills, which greatly increased productivity. I am still amazed at what we were able to accomplish and regret that we didn't have more time to tweak our language exactly how we wanted it. Also, I learned that ANTLR is a powerful but poorly documented tool. I had a lot of difficulty finding a tutorial that was at the appropriate level for this project. Everything I found was either greatly simplified, or way beyond the scope of *Espresso!*

Erin:

This was the first time I had worked in a group programming project of this magnitude. I realized what a good working experience it can be to cooperate like this. Our group had great communication skills and I realized that being able to determine and clearly explain the front end of each person's contributions to the others is an invaluable skill in projects such as this. I leaned how to formulate my ideas and coding choices in a way that people rather than computers would understand. For many of the problems we encountered, several different opinions about the best way to solve them would arise. I learned more about each person's programming style and understood ways to improve my own coding skills. I am now much more comfortable with the elements of a Java applet (although I don't really need to as I could just use *Espresso!* as a template ☺) I had a refresher in RCS, UNIX groups creation and use, and shell scripting as well!

Phil:

I've taken several classes at Columbia that require you to work in a group just so the TA's don't have as many programs to grade. Once we started work on this project though, I realized right away that there was absolutely no way it could get done within the semester unless we all put our heads together. The initial design process was definitely the hardest part to get through. The biggest conceptual issues involved realizing the between a compiler and a translator, and deciding exactly which pieces of the puzzle needed to be solved at translation-time and which things we needed to leave for javac to resolve on its own. Each member of our group had a very different coding and learning style, and this absolutely worked in our favor when problems and design issues came up. Picking group members that you could live with is a really good idea, because believe me you will be living with them.

Aaron:

This was one of my first large software projects, and I learned a great deal in working as a team member. The most interesting part of the process to me was working together to achieve common goals. While I was assigned certain parts of the project to work on, I wanted to keep track of what was going on with the other pieces. This required explanation and periodic briefings from the other team members. When coding with other people, it's fun to see how programming styles and methods vary. One of the best parts about working in a group is that I find that I don't run into so many of the small problems that plague solo projects. If I was unsure about something, I wouldn't hesitate to ask one of my teammates. Having several people also comes in handy when debugging. A fresh set of eyes on your code can bring to light issues which were hidden to you. Finally, although the task seemed daunting at first, and I did not know where to begin, breaking the project down into bite sized pieces helped to clarify each of our roles.

Appendix A

Source Code

A.1 esp

```
#!/bin/sh
#
# A simple script for compiling .esp programs
#
# User must specify the .esp file and an
# optional output directory name.
#
# Author:  Joya Zuber, Erin Adelman

if [$2 = $null]
then
java Main $1 espresso
mkdir espresso/RTL
cp RTL/*.class espresso/RTL
javac espresso/espresso.java
mkdir $HOME/public_html/espresso
mv espresso/* $HOME/public_html/espresso
chmod -R 755 $HOME/public_html/espresso

#rm -Rf espresso
else
java Main $1 $2
mkdir $2/RTL
cp RTL/*.class $2/RTL
javac $2/$2.java
mkdir $HOME/public_html/$2
mv $2/* $HOME/public_html/$2
chmod -R 755 $HOME/public_html/$2
#rm -Rf $2
fi
```

A.2 grammar.g

```
/*
 * grammar.g : the lexer and the parser for Espresso!
 *
 * uses ANTLR grammar
 *
 * @author Joya Zuber - jaz49@columbia.edu
 *
 * @version 1
 */

class L extends Lexer;

options {

    k = 2; //look ahead is two characters (longest operator is ==)
    charVocabulary = '\3'..'\377'; //forces antlr to be all
                                   //unicode characters useful for ~
    testLiterals = false; //don't automatically test literals for keywords
    exportVocab = Espresso; //name the vocabulary 'Espresso'
}

//reports to the terminal when there is a lex error
{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

protected //can only be used by other rules in the lexer

ALPHA   :  'a'..'z' | 'A'..'Z' | '_'; // alpha charaters

protected
LOWER   :  'a'..'z'; //lower case characters

protected
UPPER   :  'A'..'Z'; //upper case characters

protected
DIGIT   :  '0'..'9'; //digits


//skips all spaces tabs
WS      :  (' ' | '\t' )+
           { $setType(Token.SKIP); }
        ;

//finds a newline character and advances the line count in the lexer
NL      :  '\n'

            //newline() increments the parsers count
            //and is used for error messages
```

```
            { $setType(Token.SKIP); newline(); }
        ;

//all identifiers.  tests literals so that it can check for reserved words
ID options { testLiterals = true; }
        :  LOWER (ALPHA | DIGIT  )*
        ;

//espresso only supports simple signed decimal numbers
NUMBER  :  (DIGIT)+ (DOT (DIGIT)+)?
        ;

//espresso object must start with an uppercase letter
//and be followed by 0 or more alpha characters
OBJECT  :  UPPER (ALPHA)*
        ;

/* Based on the comments used in the lexer for Esterel.  Allows only
   java style multiple line comments.
*/
COMMENT
        :  "/*"
           ( options { greedy=false; }:(NL)
        |  ~('\r'|'\n') )* "*/"
           { $setType(Token.SKIP); }
        ;

/* From the lexer for Esterel.  This treats two double quotes
   as a literal double quote, discarding one in the final
   string token.  The double quotes surrounding the actual
   string are also discarded in the final string.
*/
STRING  :  '"'!
           (  ~('"' | '\n' |'\r')
            |  ('"'!'"')
           )*
          '"'!
        ;

//delimiters
LPAREN      :  '(';
RPAREN      :  ')';
LBRACE      :  '{';
RBRACE      :  '}';
LBRK        :  '[';
RBRK        :  ']';

SEMI        :  ';';
COMMA       :  ',';

ASSGN       :  '=';
DOT         :  '.'
            ;

//Mathematical operators
PLUS     :  '+';

MINUS    :  '-'
            ( (DIGIT)+ (DOT (DIGIT)+)? { $setType(NUMBER); } )?
         ;
```

```
MULT     :  '*';

DIV      :  '/';

//Boolean operators:
GT       :  '>';
GTEQ     :  ">=";

LT       :  '<';
LTEQ     :  "<=";

OR       :  "||";
AND      :  "&&";

EQ       :  "==";
NOT      :  '!';
NOTEQ    :  "!=";


class P extends Parser;

options {
    k = 2; //looks ahead two tokens
    buildAST = true;
    exportVocab = Espresso; //call its vocabulary 'Espresso'
}

//tokens in espresso
tokens {
    CARD;
    ASSIGNMENT;
    OBJ_ASSGN;
    NUM;
    STR;
    OBJ;
    CONCAT;
  }

//reports to the terminal when there is a lex error
{
    int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

/* This rule dictates that a program will consist only of
   a card definition.  A program will look look like this:

   card name_of_card {
      statemenS;
   }
*/
card
        : "card"! ID! LBRACE!
          (disable_stmt)?(statement)*
```

```
          RBRACE!
          { #card=#([CARD,"CARD"],card); }
        ;

/* this rule dictates that a statement will consist of
   the following definitions below.
*/
statement
        :  while_stmt   //while loop
        |  if_stmt      //if/else statement
        |  declaration  //creates number, string, color, or object
        |  assignment   //modifies value of created number, string, or color
        |  obj_modify   //modifies value of created object
        |  print_stmt   //prints(built-in function)
        |  set_backgrnd //sets the background color of the applet
        |  pause        //pauses for a given amount of milliseconds
        |  action_stmt  //creates an action listener for an object
        ;

/* a while statement will loop while the expression
   enclosed in parentheses after the 'while' remains
   true.  The group of zero or more statements will
   be evaluated on each loop.
*/
while_stmt
        :
           "while"^ LPAREN! (bool) RPAREN!
           LBRACE! (statement | break_stmt)* RBRACE!
        ;

/* standard if/else statement
*/
if_stmt
        :  "if"^ LPAREN! (bool) RPAREN! if_block
           (options {greedy = true;}: "else"  else_block)?
        ;

/* the if block of the if/else statement.
   Braces are not necessary if only one statment
   follows the if keyword.
*/
if_block
        :  LBRACE! (options {greedy = true;}: (statement)(statement)* RBRACE!)
        |  statement
        ;

/* else block of the if/esle statement
   0 or 1 else statements can follow each
   if statement. Braces are not necessary if only
   one statement follows the else keyword.
*/
else_block
        :  LBRACE! (options {greedy = true;}: (statement)(statement)* RBRACE!)
        |  statement
        ;

/* a declaration creates a number, string, color, or object.
   Number, string, and colors declarations include an assignment
   to a default value as specified by the user.
*/
declaration
```

```
        :  "number"! ID ASSGN! (mdas_op | internal_obj_modify) SEMI!
           { #declaration=#([NUM,"NUM"],declaration); }
        |  "string"! ID ASSGN! (STRING | ID | concat|internal_obj_modify) SEMI!
           { #declaration=#([STR,"STR"],declaration); }
        |  "color"^ ID ASSGN! (mdas_op | internal_obj_modify) SEMI!
        |  OBJECT ID SEMI!
           { #declaration=#([OBJ,"OBJ"],declaration); }
        ;

/* an assignment changes the value of a number, string, or color.
*/
assignment
        :  ID ASSGN! (((mdas_op | CONCAT | STRING) SEMI!) | obj_modify)
           { #assignment=#([ASSIGNMENT,"ASSGINMENT"],assignment); }
        ;

/* an object modification changes the value of an object attribute
   using a method defined in the runtime library.
*/
obj_modify
        :  ID DOT^ ID LPAREN! (param)? RPAREN! SEMI!
        ;

/* a parameter is used in object modifications
*/
param
        :  (mdas_op | STRING | concat | internal_obj_modify)
        ;

/* a parameter that is itself an object modification
*/
internal_obj_modify
        :  ID DOT^ ID LPAREN! (param)? RPAREN!
        ;

/* internal print statement prints a string to the applet screen
*/
print_stmt
        : "print"^ (STRING | concat | ID) SEMI!
        ;

/* standard break statement used only in while loops.
*/
break_stmt
        :  "break"^ SEMI!
        ;

/* disable statement disables the default layout manager (NULL)
   to afford the user layout manager capabilities in the applet.
*/
disable_stmt
        :  "disableLayout"^ SEMI!
        ;

/* set the background color of the applet.
*/
set_backgrnd
        :  "setBackground"^ LPAREN! mdas_op RPAREN! SEMI!
        ;

/* perform an action when an action capable object is
```

```
   triggered.
*/
action_stmt
        : ID "action"^ LBRACE! (statement)* RBRACE!
        ;

/* pause for a given amount of milliseconds.
*/
pause
        :  "pause"^ mdas_op SEMI!
        ;

/* a boolean value that is the result of a string
   comparison.
*/
bool    :  LPAREN! (strcmp|expression) RPAREN!
        |  (strcmp|expression)
        ;

/* string comparison format
*/
strcmp  :  (STRING | concat) EQ^ (STRING | concat)
        ;

/* a combination of identifiers or numbers, logic terms, and
   the || operator.
*/
expression
        :  logic_term ( OR^ logic_term)*
        ;

/* a combination of identifiers, numbers, logic factors,
   and the && operator
*/
logic_term
        :  logic_factor ( AND^ logic_factor)*
        ;

/* a boolean expression with an optional ! associated with it
*/
logic_factor
        :  (NOT^)? boolean_expr
        ;

/* a boolean expression (numbers or identifiers, mdas ops and the boolean
   comparators)
*/
boolean_expr
        :  mdas_op ( (GT^ | GTEQ^ | LT^ | LTEQ^ | EQ^ | NOTEQ^ ) mdas_op)?
        ;

/* the mdas of PEMDAS (multiplication, division, addition,
   and subtraction) a combination of numbers, identifiers,
   multiplication, division, addition, and subtraction
*/
mdas_op
        :  mdas_term ( ( PLUS^ | MINUS^ )  mdas_term )*
        ;
/* the md of PEMDAS (a combination of numbers and identifiers
   and the multiplication and division operators.
*/
```

```
mdas_term
        :  identifier ((MULT^ | DIV^) identifier)*
        ;

/* an identifier to be used in mathematical expressions
*/
identifier
        :  ID
        |  NUMBER
        ;

/* the concatinatoin of strings and identifiers.
*/
concat
        :  STRING PLUS! (STRING | NUMBER | ID)
          { #concat=#([CONCAT,"CONCAT"],concat); }
        ;
```

A.3 walker.g

```
/*
 * walker.g : the AST walker.
 *
 * @author Joya Zuber, Erin Adelman
 * @version 1.1
 *
 * The tree walker for Espresso
 *
 */


{
import java.io.*;
import java.util.*;
}

class EspressoWalker extends TreeParser;

options {
    importVocab = Espresso;
}

{
    //create necessary backend objects for Espresso! Translator
    static EspressoDataType null_data = new EspressoDataType("<NULL>");
    static EspressoSymbolTable sym_table = new EspressoSymbolTable();
    static EspressoObjectTypeTable obj_type_table = new EspressoObjectTypeTable();
    static EspressoToJava gen_code;
    static boolean par=false;

    public void setOutput(String out) {
        gen_code = new EspressoToJava(out, obj_type_table);
    }

    /**
     * sets a new filename for the runtime library config
     */
    public void setRTLCFG( String fname ) {
        obj_type_table.setRTLib( fname );
    }
}

/* method walks the AST checking for semantic errors
*/
expr returns [ EspressoDataType r ]
{
    EspressoDataType a, b, c;  //data type variables used in AST evaluation
    r = null_data;             //eventually the root of the tree
}

    /* CARD token and breaces denote the beginning
       and end of the program.
    */
    : #(CARD (a=pred:expr)* ) { AST next = pred.getNextSibling();
                                if(next==null) {
                                    gen_code.close();
                                    System.exit(0);
                                }
```

```
                            }


/* Mathematical operations return r as an instance
   of EspressoNumber if a and b are both EspressoNumbers.
   Otherwise an error will be thrown in the backend
   classes and the program will exit.
*/
| #(MINUS a=expr b=expr)     { r = a.minus(b); }
| #(PLUS a=expr b=expr)      { r = a.plus(b); }
| #(MULT a=expr b=expr)      { r = a.mult(b); }
| #(DIV a=expr b=expr)       { r = a.div(b); }


/* Boolean operations return r as an instance of
   EspressoBool if a and b are of type EspressoNumber.
   Otherwise an error will be thrown in the backend
   classes and the program will exit.
*/
| #(GT a=expr b=expr)        { r = a.gt(b); }
| #(GTEQ a=expr b=expr)      { r = a.gteq(b); }
| #(LT a=expr b=expr)        { r = a.lt(b); }
| #(LTEQ a=expr b=expr)      { r = a.lteq(b); }
| #(AND a=expr b=expr)       { r = a.and(b); }
| #(OR a=expr b=expr)        { r = a.or(b); }
| #(NOT a=expr)              { r = a.not(); }


/* These Boolean operations can be applied to
   EspressoNumbers or EspressoString and return
   an EspressoBool if a and b are of the correct type.
 */
| #(EQ a=expr b=expr)        { r = a.eq(b); }
| #(NOTEQ a=expr b=expr)     { r = a.noteq(b); }


/* The CONCAT token is used to concatenate two strings.
   r will be an EspressoString that is the concatenation of an
   EspressoString a and an EspressoDataType b that is either
   an EspressoString or an EspressoNumber. If a and b are not
   of the correct type, the backend throws an error and exits.
*/
| #(CONCAT a=expr b=expr)    {  r= a.concat(b); }


/* The internal print statment prints a string to the screen
   of the applet, defaulting to the middle of the screen.
*/
| #("print" a=expr)          {

        if(!(a instanceof EspressoString)) {
            System.out.println("Method print must take a string");
            System.exit(1);
        }

        gen_code.setStateInit();
        gen_code.printCode("listOfEspressoShapes.add"+
            "(new EspressoStringRT("+a.name+", this));"); r=a;
        gen_code.unsetStateInit();
    }
```

```
    /* The break statement can be used only in a while or if loop
       in Espresso. It functions exacly like a break statement
       in Java.
    */
    | "break"                   {   gen_code.printCode("break;\n"); }


    /* The disableLayout statement sets the layout manager so the
       user has complete control over the location of objects throughout
       the applet. If used, this statement must appear as the first
       statement written by the programmer.
    */
    | "disableLayout"           { gen_code.setStateInit();
                                  gen_code.printCode("setLayout(null);");
                                  gen_code.unsetStateInit();
                                }


    /* Standard while loop statement ensures that the condition
       is an instance of EspressoBool. Otherwise an error is
       thrown and the program exits.
    */
    | #("while" a=expr {
                            if( a instanceof EspressoBool )
                                gen_code.printCode("while(__always_false||"+a.name+")
{\n");

                            else {
                                System.err.println("while takes boolean as arg");
                                System.exit(1);
                            }
                        }

            (b=expr)*) {
                gen_code.printCode("} //end while\n");
        }


   /* Standard if/else statement. Walker ensures that the
      condition in the if statement is a boolean expression.
   */
   |#("if" a=expr {
                    /* a is a boolean expression, continue */
                 if( a instanceof EspressoBool ) {
                             gen_code.printCode("if("+a.name+") {\n");
                            }
                            else {
                                System.err.println("if takes boolean as arg");
                                System.exit(1);
                            }
                }
                /* 1 or more statements within the if {} brackets */
                (b=expr)+
                {
                    gen_code.printCode("}/* end if */\n");
                }
             /* 0 or 1 else statements follow end of if */
            ( "else" { gen_code.printCode("else {\n"); }

              /* 1 or more statements within the else{} brackets */
```

```
          (c=expr)+ { gen_code.printCode("}/*end of else */\n"); } )?)


/* Number Declaration associates an EspressoNumber
   with an identifier and stores it in the symbol
   table. If the argument is not an EspressoNumber
   an error is thrown and the program exits.
*/
| #(NUM nid:ID a=expr)       {

            if(a instanceof EspressoNumber)  {

                /* create a new EspressoNumber */
                r = new EspressoNumber(a.name);

                /* add identifier to symbol table */
                String rv = sym_table.add(nid.getText(), r);

                /* print appropriate code to generated file */
                if( !rv.equals(null) ) {

                    /* declare new number */
                    gen_code.setStateDecl();
                    gen_code.printCode(rv);
                    gen_code.unsetStateDecl();

                    /* initialize new number */
                    gen_code.setStateInit();
                    gen_code.printCode(nid.getText()+"="+a.name+";\n");
                    gen_code.unsetStateInit();
                }

           }

            /* argument is not of type Espresso Number, exit */
            else {
                System.err.println(a.name+": is not of type EspressoNumber\n");
                System.exit(1);
            }
    }

/* String Declaration associates an EspressoString
   with an identifier and stores it in the symbol
   table. If the argument is not an EspressoString
   an error is thrown and the program exits.
*/
| #(STR sid:ID a=expr)    {

        if(a instanceof EspressoString)  {

            /* create a new EspressoString */
            r = new EspressoString(a.name);

            /* add identifier to the symbol table */
            String rv = sym_table.add(sid.getText(), r);

            /* print appropriate code to generated file */
            if( !rv.equals(null) )  {
                System.out.println(r.typename());

                /* declare new string */
```

```
                    gen_code.setStateDecl();
                    gen_code.printCode(rv);
                    gen_code.unsetStateDecl();

                    /* initialize new string */
                    gen_code.setStateInit();
                    gen_code.printCode(sid.getText()+"="+a.name+";\n");
                    gen_code.unsetStateInit();
                   }
            }

            /* argument is not an EspressoString, exit */
            else {
                System.err.println(a.name+": is not of type EspressoString\n");
                System.exit(1);
            }
        }

    /* Color Declaration associates an EspressoColor
       with an identifier and stores it in the symbol
       table. If the argument is not an EspressoColor
       an error is thrown and the program exits.
    */
    | #("color" cid:ID a=expr)       {

                if(a instanceof EspressoColorPrim)  {

                    /* create a new EspressoColor */
                    r = new EspressoColor(a.getColor());

                    /* add identifier to the symbol table */
                    String rv = sym_table.add(cid.getText(), r);

                    /* print appropriate code to generated file */
                    if( !rv.equals(null) ) {

                        /* declare new color */
                        gen_code.setStateInit();
                        gen_code.printCode(cid.getText()+
                            "= new EspressoColorRT("+a.getColor()+", "
+a.getNumberColors()+");");
                        gen_code.unsetStateInit();

                        /* initialize color */
                        gen_code.setStateDecl();
                        gen_code.printCode(rv);
                        gen_code.unsetStateDecl();;
                    }

                }

                /* argument is not an EspressoColor, exit */
                else {
                    System.err.println(a.name+": is not of type EspressoColor\n");
                    System.exit(1);
                }
        }


    /* Object declaration associates an identifier with a valid
       object type. If the object type is invalid or the identifier
```

```
   already exists in the symbol table, an error is thrown and
   the program exits.
*/
| #(OBJ obj:OBJECT oid:ID) {

        /* check that obj is a valid instance of object */
        EspressoObjectType o = obj_type_table.check(obj.getText());

        /* create a new EspressoObject and add it to the symbol table */
        EspressoObject dt = new EspressoObject( oid.getText(), obj.getText() );
        String rv = sym_table.add(oid.getText(), dt);

        /* declare object in output file */
        gen_code.setStateDecl();
        gen_code.printCode(o.getDecl( oid.getText() ));
        gen_code.unsetStateDecl();

        /* initialize object with default values in output file */
        gen_code.setStateInit();
        gen_code.printCode(o.getInit(oid.getText()));
        gen_code.unsetStateInit();
  }

/* Assignment of number, string, or color changes
   The value of a data type associated with a given
   identifier.
*/
| #(ASSIGNMENT a=expr b=expr) {

        /* the arguments are Numbers, proceed with assignment */
        if((a instanceof EspressoNumber)
                && (b instanceof EspressoNumber)) {
            r=new EspressoNumber(a.name);
           gen_code.printCode(a.name +"="+b.name+";\n");
        }

        /* the arguments are strings, proceed with assignment */
        else if((a instanceof EspressoString)
                && (b instanceof EspressoString)) {
            r=new EspressoString(a.name);
            gen_code.printCode(a.name +"="+b.name+";");
        }

        /* the arguments are colors, proceed with assignment */
        else if((a instanceof EspressoColor)
                 && (b instanceof EspressoColor)) {
            r = new EspressoColor(a.name, b.color, b.numColors);
            gen_code.printCode(a.name +"="+b.name+";");
        }

        /* one argument is a color identifier, and one is a
           new color; proceed with assignment
        */
        else if((a instanceof EspressoColor)
                 && (b instanceof EspressoColorPrim)) {
            r = new EspressoColor(a.name, b.color, b.numColors);
            gen_code.printCode(a.name+
                "= new EspressoColorRT("+b.getColor()+", " + b.numColors +");");
        }

        /* types are incompatible. Print error message and exit */
```

```
        else {
            System.err.print("\nIncompatible types for assignment: ");
            System.err.print(a.name+" "+b.name+"\n");
            System.exit(1);
        }
    } /* end of ASSIGNMENT */



/* Object modification uses the dot operator to change the
   value of an element of the given object. This operator also
   acts as an accessor. The walker ensures that the id is a
   an object that can be modified with the input method name
   and parameter type. Else an error is thrown and the program
   exits.
*/
| #(DOT objnm:ID fnc:ID (a=expr {par=true;

                /* Check that the object is in the symbol table */
                r = sym_table.check(objnm.getText());

                /* object is not in the symbol table, print error and exit */
                if(!(r instanceof EspressoObject)) {
                    System.err.println(objnm.getText() + " is not an object.");
                    System.exit(1);
                }

                /* ensre that the object type is valid */
                EspressoObjectType o = obj_type_table.check(r.typename());

                /* modify the object */
                r=r.modify(fnc.getText(), a, o);

                /* this means it is of void return type */
                if(r.typename.equals("void"))
                    gen_code.printCode(r.name);
            })?
        ) {

        /* there is no parameter */
        if(!par) {

                /* check that the object is in the symbol table */
                r = sym_table.check(objnm.getText());

                /* object is not in the symbol table, exit */
                if(!(r instanceof EspressoObject)) {
                    System.err.println(objnm.getText() + " is not an object.");
                    System.exit(1);
                }

                /* ensure that the object type is valid */
                EspressoObjectType o = obj_type_table.check(r.typename);
                r=r.modify(fnc.getText(), new EspressoDataType("", "void"), o);

                /* this means it is of void return type */
                if(r.typename.equals("void")) {
                    gen_code.printCode(r.name);
                }
                par=false;
```

```
            }   /* end of !par */
        } /* end of DOT */

/* Set Background statement sets the background color of the canvas.
   Walker ensures that the color is valid, otherwise prints
   an error message and quits.
*/
| #("setBackground" a=expr ) {

        /* a is an identifier that is a color */
        if (a instanceof EspressoColor) {
            r = a;
            gen_code.printCode("setBackground("+a.name+".getColor());\n");
        }

        /* a is a color primitive (not id) */
        else if(a instanceof EspressoColorPrim) {
            r=new EspressoColor(a.name);
            gen_code.printCode("setBackground("+a.getColor()+");\n");
        }

        /* a is not a valid color, print error message and exit*/
        else {
                System.out.println(a.name+" is not a valid color");
                System.exit(1);
            }
} /* end of setBackground */

/* The action statement dictates the event handling routine when
   an event is performed. The walker ensures the object can have
   an action performed on it, and prints the appropriate code to
   the generated file.
*/
| #("action" aaid:ID {

    /* check thta the id is in the symbol table */
    r = sym_table.check(aaid.getText());

     /* check that the id is an object */
     if(!(r instanceof EspressoObject)) {
                    System.err.println(objnm.getText() + " is not an object");
                    System.exit(1);
                }

            /* check that an action can be performed on the object */
            EspressoObjectType o = obj_type_table.check(r.typename);
                r=r.modify("action", new EspressoDataType("", "void"), o);
            gen_code.setStateAction(aaid.getText());
     }
            (a = expr)*) {
            gen_code.unsetStateAction();

  }

/* the pause statement will sleep for a given amount of
   milliseconds.
*/
|#("pause" a=expr) {

    /* print appropriate code to generated file */
    if(a instanceof EspressoNumber ) {
```

```
            gen_code.printCode("try {\n");
            gen_code.printCode("    Thread.sleep((int)"+
                            a.name+");\n");
            gen_code.printCode("}\n");
            gen_code.printCode("catch(Exception e) {\n");
            gen_code.printCode("    System.err.println("+
                            "\"Exception, exiting\");\n");
          nt appropriate code to generated file */
    if(a instanceof EspressoNumber ) {
            gen_code.printCode("try {\n");
            gen_code.printCode("    Thread.sleep((int)"+
                            a.name+");\n");
            gen_code.printCode("}\n");
            gen_code.printCode("catch(Exception e) {\n");
            gen_code.printCode("    System.err.println("+
                            "\"Exception, exiting\");\n");
            gen_code.printCode("    System.exit(1);\n");
            gen_code.printCode("}\n");
        }

    /* the parameter is not an EspressoNumber */
    else {
            System.err.println(a.name+" is not of expected type EspressoNumber");
            System.exit(1);
    }
}

/* Identifier is a number, string, object, or color. Walker
   checks that the identifier is in the symbol table.
*/
| id:ID                    {  r=sym_table.check(id.getText());
                              if(r instanceof EspressoColorPrim)
                                  r=new EspressoColorPrim(id.getText());
                           }

/* create a new number (double in java) */
| num:NUMBER               {  r=new EspressoNumber(num.getText()); }

/* create a new string (String in java) */
| str:STRING               {  r=new EspressoString("\""+str.getText()+"\""); }
;
```

A.4 EspressoException.java

```
/**
 * Exception class: messages are generated in various classes
 *
 *
 * @author Erin Adelman - eca41@columbia.edu
 * @version 1.1
 */

class EspressoException extends RuntimeException {
    EspressoException( String msg ) {
        System.err.println( "Error: " + msg );
      System.exit(1);
    }
}
```

A.5 EspressoDataType.java

```
import java.awt.Color;

/**
 * The base data type class
 *
 * Error messages are generated here.
 *
 * @author Erin Adelman
 * @version 1.1
 */

public class EspressoDataType {

    public String name;
    public String typename;
    public Color color;
    public int numColors;

    /* EspressoDataType null parameter constructor */
    public EspressoDataType() {
      this.name = null;
      this.typename="unknown";
    }

    /* EspressoDataType string constructor */
    public EspressoDataType( String name ) {
      this.name = name;
      this.typename="unknown";
    }

    /* EspressoDataType constructor when both name
       and typename are known.
     */
    public EspressoDataType( String name, String typename ) {
      this.name = name;
      this.typename = typename;
    }

    /* EspressoDataType constructor for an EspressoColor */
    public EspressoDataType(String name, Color c, int nc) {
      this.name=name;
      this.typename="color";
      this.color=c;
      this.numColors=nc;
    }

    /* return the current typename */
    public String typename() {
      return typename;
    }

    /* set the name of the object */
    public void setName(String name) {
      this.name = name;
    }

    /* return an error with a message */
    public EspressoDataType error(String msg) {
```

```
  throw new EspressoException("illegal operation: " + msg
                        + "( <" +typename() +"> "
                        + (name != null ? name : "?")
                        + " )" );
}

/* return an error specific to an object with a message */
public EspressoDataType error(EspressoDataType a, String msg) {
  if( a == null)
      return error(msg);
  throw new EspressoException(
      "illegal operation: " + msg
        + "( <" + typename() + "> "
        + ( name != null ? name : "<?>" )
        + " and "
        + "<" + typename() + "> "
        + ( name != null ? name : "<?>" )
      + " )" );
}

/* assignment method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType assign(EspressoDataType b) {
  return error(b, "=");
}

/* minus method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType minus(EspressoDataType b) {
  return error(b, "-");
}

/* plus method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType plus(EspressoDataType b) {
  return error(b, "+");
}

/* multiply method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType mult( EspressoDataType b) {
  return error(b, "*");
}

/* divide method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType div( EspressoDataType b ) {
    return error(b, "/" );
}

/* mod method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType mod( EspressoDataType b ) {
    return error(b, "%" );
}
```

```
/* greater than method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType gt( EspressoDataType b ) {
    return error(b, ">" );
}

/* greater than or equal method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType gteq( EspressoDataType b ) {
    return error(b, ">=" );
}

/* less than method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType lt( EspressoDataType b ) {
    return error(b, "<" );
}

/* less than or equal method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType lteq( EspressoDataType b ) {
    return error(b, "<=" );
}

/* equals method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType eq( EspressoDataType b ) {
    return error(b, "==" );
}

/* not equal method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType noteq( EspressoDataType b ) {
    return error(b, "!=" );
}

/* and method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType and( EspressoDataType b ) {
    return error(b, "and" );
}

/* or method - default, a correctly
   instantiated object should never reach this
*/
public EspressoDataType or( EspressoDataType b ) {
    return error(b, "or" );
}

/* modify method - default, a correctly
   instantiated object should never reach this
*/
```

```
    public EspressoDataType modify( String methodName, EspressoDataType param,
EspressoObjectType ot ) {
        return error( param, methodName + " -can only be applied to an object" );
    }

    /* not method - default, a correctly
       instantiated object should never reach this
    */
    public EspressoDataType not() {
        return error( "not" );
    }
    /* concat method - default, a correctly
       instantiated object should never reach this
    */
    public EspressoDataType concat(EspressoDataType b) {
      return error(b, "concat");
    }

    /* getInit method - default, a correctly
       instantiated object should never reach this
    */
    public String  getInit() {
      return "error getInit()";
    }

    //this will never happen
    public String getColor() {
        return ("function getColor() cannot be applied.");
    }

    /* create an color object for the
       given color  method - default, a correctly
       instantiated object should never reach this
    */
    public Color getColorObj() {
      System.err.println("error");
      System.exit(1);
      return Color.black;
    }
    /* get number of colors mixed in the current
       color method - default, a correctly
       instantiated object should never reach this
    */
    public int getNumberColors() {
      System.err.println("error");
      System.exit(1);
      return -1;
    }
}/* end of EspressoDataType */
```

A.6 EspressoBool.java

```
/**
 * the wrapper class for boolean
 *
 *
 * @author Erin Adelman - eca41@columbia.edu
 * @version 1.1
 */
class EspressoBool extends EspressoDataType {

    /* constructor calls EspressoDataType constructor */
    EspressoBool( String name ) {
        super(name);
    }

    /* typename for storing in symbol table */
    public String typename() {
        return "bool";
    }

    /* and operator returns an EspressoBool */
    public EspressoDataType and( EspressoDataType b ) {
        if (!(b instanceof EspressoBool))
            b.error(b, "and");
        return new EspressoBool( "("+name+"&&"+b.name+")" );
    }

    /* or operator returns an EspressoBool */
    public EspressoDataType or( EspressoDataType b ) {
        if (!(b instanceof EspressoBool))
            b.error(b, "or");
        return new EspressoBool( "("+name+"||"+b.name+")" );
    }

    /* not operator returns an EspressoBool */
    public EspressoDataType not() {
        return new EspressoBool( "!("+name+")");
    }

    /* eq operator returns an EspressoBool */
    public EspressoDataType eq( EspressoDataType b ) {
        if ( b instanceof EspressoBool )
            return new EspressoBool("( "+name+" == "+b.name+")" );
        return error( b, "==" );
    }

    /* noteq operator returns an EspressoBool */
    public EspressoDataType noteq( EspressoDataType b ) {
            if ( b instanceof EspressoBool )
            return new EspressoBool("( "+name+" != "+b.name+")" );
        return error( b, "!=" );
    }
} /* end of EspressoBool */
```

A.7 EspressoDouble

```
import java.io.PrintWriter;

/**
 * The wrapper class for double
 *
 * @author Erin Adelman
 * @version 1.1
 */
class EspressoNumber extends EspressoDataType {

    /* EspressoNumber constructor with no parameter
       calls superclass constructor.
    */
    public EspressoNumber() {
      super();
    }

    /* EspressoNumber constructor with one parameter
       calls superclass constructor.
    */
    public EspressoNumber(String name) {
      super("("+name+")");
    }

    /* return the typename of the object */
    public String typename() {
        return "number";
    }

    /* plus method returns java syntax to add
       two doubles.
    */
    public EspressoDataType plus( EspressoDataType b ) {
      if (!(b instanceof EspressoNumber ))
          b.error(b, "+");
      return new EspressoNumber("("+name + " + " + b.name+")");
    }

    /* minus method returns java syntax to subtract
       two doubles.
    */
    public EspressoDataType minus( EspressoDataType b ) {
      if (!(b instanceof EspressoNumber ))
          b.error(b, "-");
      return new EspressoNumber("("+name + " - " + b.name+")");
    }

     /* mult method returns java syntax to multiply
       two doubles.
    */
    public EspressoDataType mult( EspressoDataType b ) {
      if (!(b instanceof EspressoNumber ))
          b.error(b, "*");
      return new EspressoNumber("("+name + " * " + b.name+")");
    }

    /* div method returns java syntax to divide
       two doubles.
    */ public EspressoDataType div( EspressoDataType b ) {
```

```
      if (!(b instanceof EspressoNumber ))
          b.error(b, "/");
      return new EspressoNumber("("+name + " / " + b.name+")");
    }

    /* gt method returns an EspressoBool that determines
       if one number is greater than the other.
    */
    public EspressoDataType gt( EspressoDataType b ) {
      if (!((b instanceof EspressoNumber )||(b instanceof EspressoBool)))
          b.error(b, ">");
      return new EspressoBool("("+name + " > " + b.name+")");
     }

    /* gteq method returns an EspressoBool that determines
       if one number is greater than or equal to the other.
    */
    public EspressoDataType gteq( EspressoDataType b ) {
      if (!((b instanceof EspressoNumber )||(b instanceof EspressoBool)))
          b.error(b, ">=");
      return new EspressoBool("("+name + " >= " + b.name+")");
    }

    /* lt method returns an EspressoBool that determines
       if one number is less than the other.
    */
    public EspressoDataType lt( EspressoDataType b ) {
      if (!((b instanceof EspressoNumber )||(b instanceof EspressoBool)))
          b.error(b, "<");
      return new EspressoBool("("+name + " < " + b.name+")");
    }

    /* lteq method returns an EspressoBool that determines
       if one number is less than or equal to the other.
    */
    public EspressoDataType lteq( EspressoDataType b ) {
      if (!((b instanceof EspressoNumber )||(b instanceof EspressoBool)))
          b.error(b, "<=");
      return new EspressoBool("("+name + " <= " + b.name+")");
    }

    /* eq method returns an EspressoBool that determines
       if one number is equal to the other.
    */
    public EspressoDataType eq( EspressoDataType b ) {
      if (!((b instanceof EspressoNumber )||(b instanceof EspressoBool)))
          b.error(b, "==");
      return new EspressoBool("("+name + " == " + b.name+")");
    }

    /* noteq method returns an EspressoBool that determines
       if one number is not equal to the other.
    */
    public EspressoDataType noteq( EspressoDataType b ) {
      if (!((b instanceof EspressoNumber )||(b instanceof EspressoBool)))
          b.error(b, "!=");
      return new EspressoBool("("+name + " != " + b.name+")");
    }
}
```

A.8 EspressoString.java

```
import java.io.PrintWriter;

/**
 * the wrapper class for string
 *
 * @author Erin Adelman - eca41@columbia.edu
 * @version 1.1
 */
class EspressoString extends EspressoDataType {

    /* EspressoString constructor calls superclass
       constructor.
    */
    public EspressoString( String name) {
        super(name);
      typename="string";
    }

    /* return the typename of the object */
    public String typename() {
        return "string";
    }

    /* eq method determines if two strings are equal */
    public EspressoDataType eq( EspressoDataType b ) {
        if ( b instanceof EspressoString )
            return new EspressoBool("( ("+name+").equals("+b.name+"))" );
        return error( b, "==" );
    }

    /* noteq method determines if two strings are not equal */
    public EspressoDataType noteq( EspressoDataType b ) {
      if ( b instanceof EspressoBool )
          return new EspressoBool("(!("+name+").equals("+b.name+"))" );
        return error( b, "!=" );
    }

    /* concat method concatenates two strings or a string
       and an number.
    */
    public EspressoDataType concat( EspressoDataType b ) {
        if (b instanceof EspressoString)
          return new EspressoString(name+"+"+b.name);
      else if(b instanceof EspressoNumber)
          return new EspressoString(name+"+"+b.name);
      else {
          b.error(b, "concat" );
          return null;
      }
    }

}/* end of EspressoString */
```

A.9 EspressoColorPrim.java

```
import java.awt.Color;

/**
 * @author Joya Zuber
 * @version 1.1
 *
 * The wrapper class for color primitives.
 * Adds colors and finds their mixture value.
 */

public class EspressoColorPrim extends EspressoDataType {

    int numColors; /* number of colors mixed to make this color */
    Color currColor; /* current rgbs of this color */

    /* constructor takes in the name of a primitive color */
    public EspressoColorPrim(String name) {
      super(name);
      currColor=translate(name);
      numColors=1;
      name=getColor();
    }

    //return typename
    public String typename() {
      return "color_prim";
    }

    //returns current color as a string representation
    public String getColor() {
      return ("new Color("+
            currColor.getRed()+","
            +currColor.getGreen()+","+
            currColor.getBlue()+")");
    }

    //adds another color primitive to this one
    //finds the mixture based on the number of colors
    //already mixed
    public void addColor(Color c) {

      int r=c.getRed()+(currColor.getRed()*numColors);
      int g=c.getGreen()+(currColor.getGreen()*numColors);
      int b=c.getBlue()+(currColor.getBlue()*numColors);

      numColors++;
      r/=numColors;
      g/=numColors;
      b/=numColors;

            name=getColor();
    }

    //returns the number of colors mixed into this primitive
    public int getNumberColors() {
      return numColors;
    }

    //handles the adding of colors.  only allows adding
```

```
    //to other colorPrims
    public EspressoDataType plus(EspressoDataType b) {
      if (!(b instanceof EspressoColorPrim ))
          b.error(b, "+");
      addColor(translate(b.name));
      return this;
    }

    //translates a color name to its java object
    public Color translate(String c) {

      if(c.equals("black"))
          return Color.black;
      else if(c.equals("blue"))
          return Color.blue;
      else if(c.equals("cyan"))
          return Color.cyan;
      else if(c.equals("darkGray"))
          return Color.darkGray;
      else if(c.equals("gray"))
          return Color.gray;
      else if(c.equals("green"))
          return Color.green;
      else if(c.equals("lightGray"))
          return Color.lightGray;
      else if(c.equals("magenta"))
          return Color.magenta;
      else if(c.equals("orange"))
          return Color.orange;
      else if(c.equals("pink"))
          return Color.pink;
      else if(c.equals("red"))
          return Color.red;
      else if(c.equals("white"))
          return Color.white;
      else if(c.equals("yellow"))
          return Color.yellow;
      else
          return Color.orange;
    }
}
```

A.10 EspressoColor.java

```
import java.awt.Color;

/**
 * The wrapper class for color.  Supports adding of colors and
 * creating new colors.
 *
 * @author Joya Zuber
 * @version 1.1
 */

class EspressoColor extends EspressoDataType {

     int numColors;

    /* create a new color, default to white */
    public EspressoColor(String name) {
      super(name);
      color = Color.white;
      typename="color";
      numColors=1;
    }

    /* create a new color object with specified color */
    public EspressoColor(String name, Color c, int nc) {
      super(name, c, nc);
      color = c;
      typename="color";
      numColors=nc;
    }

    //returns typename
    public String typename() {
      return "color";
    }

    //adds a new color object to the current color
    public void addColor(String cname, int c) {
      numColors+=c;
      name=name+".calcColor("+cname+", "+c+")";
    }

    //handles the adding of colors
    public EspressoDataType plus(EspressoDataType b) {
      if (!(b instanceof EspressoColorPrim )
          &&!(b instanceof EspressoColor))
          b.error(b, "+");
      else
          addColor(b.getColor(), b.getNumberColors());
      return this;
    }
} /* end of color class */
```

A.11 EspressoObject.java

```
/**
 * an EspressoObject represents an instance of an Object in the AST at
 *   compile-time.
 *
 * an EspressoObject must be associated with an EspressoObjectType, a string
 *  for which must be retrieved from the obj_type_table at object declaration
 *  time.
 *
 * to determine if a method is legal for a particular EspressoObject:
 *  -retrieve the object type and call isValidMethod():
 *     getObjType().isValidMethod( String methodname )
 *
 *
 * version: 1.0 by Phil
 *
 * notes(1.0):
 *
 */

/**
 * @author phil
 */

public class EspressoObject extends EspressoDataType {

    /**
     * creates a new EspressoObject of type 'type' with ID name 'id'
     * @param type a String defining the type of object this is (must be
     *        registered on the EspressoObjectTypeTable)
     * @param id the ID's name
     */
    public EspressoObject( String id, String type ) {
      super(id, type);
     }

     /**
      * checks that a methodName is a valid method for this type of E-Object
      *
      * @param methodName the name of the method to apply
      * @param param the parameter, as an EDT, to apply to this obj
      * @param ot a reference to the EObjectType of this object (necessary
      *   to verify methods since we don't hash entire Objects on the symtable)
      * @return an EDT which is: -instanceof EspressoObject with type "void" if
      *   the requested method had void return type (should be outputted to
      *   Java file immediately.
      *  -instanceof anything else representative of the method's
      *    return type, with name=a string representative of this modify's
      *    action
      */
    public EspressoDataType modify( String methodName, EspressoDataType param,
EspressoObjectType ot ) {
        // get EspressoObjectMethod for the methodName in EspressoObjectType ot
        //    if it doesn't exist --> FAIL
        // check param.typename against EOMethod.paramtype
        //    different types --> FAIL
        // generate string s: id.methodName(param)
        // check EOmethod.returntype
        //  case void:  return a new EspressoObject with name=s
        //  case numer: return new EspressoNumber w/name=s
```

```
      //  case string: return new EspressoString w/name=s
      //  etc...

    String r=null;
    if(param.typename().equals("color_prim")) {
        param=new EspressoColor(param.getColor());
        r = name + "." + methodName + "(" + param.name + ");";
    }

      EspressoObjectMethod meth = ot.getMethod( methodName );
      if (meth == null) {
          System.err.println("Method ["+methodName+"] is not supported by"+
                  " class type ["+ot.espressoTypeName+"]");
          System.exit(1);
      }

      if (!meth.paramType.equals( param.typename() )) {
          System.err.println(name+"."+methodName+"("+param.name+
                  "): - class \""+this.typename()+"\"\n\texpected parameter "+
                  "of type ["+meth.paramType+"], got type ["
                  +param.typename()+"]");
          System.exit(1);
      }

      String s = name + "." + methodName + "(" + param.name + ");";

    if(param.typename().equals("color")) {
        if(null!=r)
          s=r;
        else
          s = name + "." + methodName + "(" + param.name + ".getColor());";
    }

      if (meth.returnType.equals("void"))
          return new EspressoObject(s, "void");
      if (meth.returnType.equals("number"))
          return new EspressoNumber(s);
    else if (meth.returnType.equals("string"))
          return new EspressoString(s);
    else if (meth.returnType.equals("color"))
          return new EspressoColor(s);
      else  /* otherwise case: returns a generic object */
          return new EspressoObject(s, meth.returnType);

      // if modify() returns instanceof EspressoObject with type "void",
      //   it should be print_code'd immediately.
      // else, it should only be returned as r
  }

}
```

A.12 EspressoRunTime.lib

```
# Runtime Library Definitions
# **************************
# Every class name and method may contain a mapping from Espresso!::Java
# If there is no mapping present, Espresso! will assume a 1-to-1 correlation.
# Every runtime Class and method must be declared in this file to be recognized
# by Espresso! during translation
#
# every class/method must be on a new line.
# blank lines are ignored.
#
# Java mapping of class declarations are used as constructors.
# No Espresso! constructors may take parameters, therefore Java constructors
# may only take parameters if they are hardcoded, i.e.:
#    Image::EspressoImageRT(this) {
#      ...
#    }
# will always initialize an Image as "id = new EspressoImageRT(this)"
# This particular construct is useful for the image class, as an Image
# in an Applet requires a reference to the parent Applet.
#
# Return types and parameter types must always match between Espresso!<->Java
# Example method mapping:
#    boolean isChecked::getState (void)
# will cause isChecked() in Espresso! to translate to getState() in Java
#
# Opening brace must immediately follow class name (on same line)
# Closing brace must be on line by itself.
#

TextBox::EspressoTextBoxRT(this) {
void setText( string )
string getText( void )
void setWidth( number )
void setHeight( number )
void setX( number )
void setY( number )
number getWidth( void )
number getHeight( void )
number getX( void )
number getY( void )
void action ( void )
}

Button::EspressoButtonRT(this){
void setText( string )
string getText( void )
void setWidth( number )
void setHeight( number )
void setX( number )
void setY( number )
number getWidth( void )
number getHeight( void )
number getX( void )
number getY( void )
void action ( void )
}
```

```
CheckBox::EspressoChkBoxRT(this){
void setText( string )
string getText( void )
boolean getState( void )
void setWidth( number )
void setHeight( number )
void setX( number )
void setY( number )
number getWidth( void )
number getHeight( void )
number getX( void )
number getY( void )
void action ( void )
}

Image::EspressoImageRT(this, listOfEspressoShapes){
void setSource( string )
string getSource( void )
void setX( number )
number getX( void )
void setY( number )
number getY( void )
void draw( Graphics )
}

Oval::EspressoOvalRT(listOfEspressoShapes, this){
void setColor( color )
void setX( number )
number getX( void )
void setY( number )
number getY( void )
void setHeight( number )
number getHeight( void )
number getWidth( void )
void setWidth( number )
void draw( Graphics )
}

Rectangle::EspressoRectRT(listOfEspressoShapes, this){
void setColor( color )
void setX( number )
number getX( void )
void setY( number )
number getY( void )
void setWidth( number )
number getWidth( void )
void setHeight( number )
number getHeight( void )
void draw( Graphics )
}

String::EspressoStringRT(listOfEspressoShapes, this){
void setColor( color )
void setText( string )
string getText( void )
void setX( number )
number getX( void )
void setY( number )
number getY( void )
void draw( Graphics )
}
```

A.13 EspressoSymbolTable.java

```
/**
 * EspressoSymbolTable handles all functions relating to the symbol table
 *
 * notes from phil-
 *
 * last updated by Phil 12-10-03:
 *  added colors statically symbol table
 *  updated comments
 */
import java.util.*;

public class EspressoSymbolTable {

    private Hashtable symTable;

    /**
     * constructs a new symbol table, adds all statically defined ID's
     *  (currently just colors)
     * note that adding an ID to the table here does not generate a declaration
     * in the translated code.  declarations for all static ID's must also
     * be included in EspressoToJava's DECLARE and INIT block headers.
     */
    public EspressoSymbolTable() {
      symTable = new Hashtable();
      // load colors:
        // every color is represented by an EspressoNumber.
        // there are 13 of them, all statically defined:
        symTable.put("black", "color_prim");     /* = 0  */
        symTable.put("blue", "color_prim");      /* = 1  */
        symTable.put("cyan", "color_prim");      /* = 2  */
        symTable.put("darkGray", "color_prim");  /* = 3  */
        symTable.put("gray", "color_prim");      /* = 4  */
        symTable.put("green", "color_prim");     /* = 5  */
        symTable.put("lightGray", "color_prim"); /* = 6  */
        symTable.put("magenta", "color_prim");   /* = 7  */
        symTable.put("orange", "color_prim");    /* = 8  */
        symTable.put("pink", "color_prim");      /* = 9  */
        symTable.put("red", "color_prim");       /* = 10 */
        symTable.put("white", "color_prim");     /* = 11 */
        symTable.put("yellow", "color_prim");    /* = 12 */
        // actual value assignment is done statically by EspressoToJava class.
      // object type loading (RTL definitions) done in EspressoTypeTable.
    }

    /**
     * adds the requested ID to the symbol table, logs it as an instance
     * of whatever type of EspressoDataType is passed.
     * @param id the ID to add to the symbol table
     * @param x the EDT representative of this ID's type
     * @return a String representing the declaration of this object in Java
     */
    public String add(String id, EspressoDataType x) {

      if (symTable.containsKey(id)) {

          System.err.println(id+": duplicate declaration\n");
          System.exit(1);
      }
```

```
      else {
          symTable.put(id, x.typename());

          if(x.typename().equals("number"))
             return ("double " + id + ";");

          else if(x.typename().equals("string"))
            return("String " + id + ";");

          else if(x.typename().equals("color"))
            return("EspressoColorRT " + id + ";");
      }

      return null;
    }

    /**
     * checks if there is an ID on the symbol table with name "key"
     * if so, returns the proper reference to an instance of EspressoDataType
     * @param key the ID to check the symbol table for
     * @return an EspressoDataType referring to the requested ID
     */
    public EspressoDataType check(String key) {

      EspressoDataType r = new EspressoDataType(null);

      if   (!symTable.containsKey(key)) {
          System.err.println(key+": undefined identifer\n");
          System.exit(1);
      }

      String typename = (String)symTable.get(key);
      if(typename.equals("number"))
          r = new EspressoNumber(key);
      else if(typename.equals("string"))
          r=new EspressoString(key);
      else if(typename.equals("color"))
          r=new EspressoColor(key);
      else if(typename.equals("color_prim"))
          r=new EspressoColorPrim(key);
        else {
            r=new EspressoObject(key, typename);
}
        /* -phil says...-
         * if an ID is on the symbol table, we must have put it there.
         * if the type doesn't resolve to a keyword, it's got to be an Object.
         * so i took these lines out:
         *
      else {
          System.err.println(key+": unknown identifier\n");
          System.exit(1);
      }
      */
      return r;
    }
}
```

A.14 EspressoObjectTypeTable.java

```
/**
 * this class evolved from EspressoSymbolTable
 *
 * stores classnames and legal methods{return types/parameter} lists
 *  for all functions in the runtime library.
 *
 * version: 1.0 by Phil
 *
 * notes (1.0):
 *  -unlike EspressoSymbolTable, this table hashes EspressoObjectTypes instead
 *   of just a String that identifies them.  Need to hash the entire object
 *   so we don't lose the method list
 */
import java.util.*;
import java.io.*;

/**** line 40 -- need to write code for getRTObjectTypeFileNames() ****/

/**
 * @author phil
 */

public class EspressoObjectTypeTable {

    // filename of the runtime library configuration file.
    // default is "EspressoRunTime.lib", set in constructor
    private  String rtLibFileName;
    private Hashtable typeTable;

    public EspressoObjectTypeTable() {

      typeTable = new Hashtable();

        rtLibFileName="EspressoRunTime.lib";

        setRTLib( rtLibFileName );
      //load colors
      //load object types
    }

    /**
     * returns a String[] with all RT Object Type filenames
     *
     **/

    public String[] getRTObjectTypeFileNames ()
    {
        LinkedList l = new LinkedList();
        Enumeration en = typeTable.elements();
        while (en.hasMoreElements()) {
            EspressoObjectType o = (EspressoObjectType)en.nextElement();
            l.add( o.javaTypeName );
        }

        String[] s = new String[ l.size() ];
        for (int i=0; i<s.length; i++)
            s[i] = ((String)l.get(i))+".class";
```

```
        return s;
    }

    /**
     * sets alternate runtime file library config file name
     * @param fname filename from which to read runtime lib cfg
     *              (default is 'EspressoRunTime.lib')
     */
    public void setRTLib( String fname ) {
        BufferedReader in =null;
        rtLibFileName = fname;

        System.err.print("Initializing RTL table ...");

        // clear whatever we've got now.  this line could be omitted if we
        //  decide to support loading multiple libraries.
        typeTable.clear();
        typeTable = new Hashtable();
        int lineNum =0;

        try {
            // parse file.
            in = new BufferedReader( new FileReader( rtLibFileName ));

            /**
             * file format-
             * ("EspressoClassName"("::JavaConstructorName()")? "{"
             *   ("returnType" "espressoMethodName"("::javaMethodName")?
"(paramType)")*
             * "}")*
             */
            String line = in.readLine();
            EspressoObjectType currentType = null;
            while (line != null) {

                // remove whitespace.
                lineNum++;
                line = line.trim();

                // discard comments and blank lines
              if (line.length() > 0)
                if ((line.charAt(0) != '#') && (line.charAt(0) != ' ')) {
                    if (line.charAt(0) == '}') {
                        // this line terminates a class type definition
                        if (currentType == null)
                            System.err.println("currentType NULL");
                        else if (currentType.espressoTypeName == null)
                            System.err.println("typename NULL");
                        System.err.flush();

                        typeTable.put( currentType.espressoTypeName, currentType );
                        currentType = null;
                    }
                    else {
                        int start=0;
                        int i=0;
                        String token=null;

                        // position: index 0, start of line

                        if (currentType==null) {
```

```
                            currentType = new EspressoObjectType();


                            // looking for delims ':','(',')','{'
                            while (i < line.length()) {
                                char delim = line.charAt(i);
                                if (line.charAt(i) == ':') {
                                    currentType.espressoTypeName =
line.substring(start, i);
                                    currentType.espressoTypeName =
currentType.espressoTypeName.trim();
                                    start = i+2; // skip the second colon
                                    i++;
                                } /*end case ':'*/
                                else if (line.charAt(i) == '(') {
                                    if (currentType.espressoTypeName == null)
                                        currentType.espressoTypeName =
line.substring(start, i);
                                    else
                                        currentType.javaTypeName =
line.substring(start, i);

                                    currentType.javaTypeName =
currentType.javaTypeName.trim();

                                    start = i;
                                } /*end case '('*/
                                else if (line.charAt(i) == ')') {

currentType.javaConstructorParams=line.substring(start, i+1);

currentType.javaConstructorParams=currentType.javaConstructorParams.trim();

                                    start=i+1;
                                } /*end case ')' */
                                else if (line.charAt(i) == '{') {
                                    // end of class type definition.
                                    // if there was no mapping, dupe entries.
                                    if (currentType.javaTypeName == null) {
                                        currentType.javaTypeName =
currentType.espressoTypeName;
                                    }
                                    // skip anything after the brace.
                                    i = line.length();
                                } /*end case '{' */

                                i++;
                            } /* end Classname-define while loop */

                        } /* end current classname type definition */
                        else {
                            // current class has already been defined.
                            // parse the method definition on this line.
                            EspressoObjectMethod currentMethod = new
EspressoObjectMethod();

                            while (i < line.length()) {
                                // looking for delim ' '
                                // the first space is the only one we care about
                                if ((line.charAt(i) == ' ') &&
(currentMethod.returnType == null)) {
```

```
                            currentMethod.returnType = line.substring(start, i);
                                currentMethod.returnType =
currentMethod.returnType.trim();
                            start = i;
                                } /* end case ' ' */
                                else if (line.charAt(i) == ':') {
                                    // the Espresso! part of a method name map
                                    currentMethod.espressoMethodName =
line.substring(start, i);
                                    currentMethod.espressoMethodName =
currentMethod.espressoMethodName.trim();
                                    start = i+2; //skip second ':'
                                    i++;
                                } /* end case ':' */
                                else if (line.charAt(i) == '(') {
                                    if (currentMethod.espressoMethodName == null)
                                        currentMethod.espressoMethodName =
line.substring(start, i);
                                    else
                                        currentMethod.javaMethodName =
line.substring(start, i);

                                    // (we trim the string below)

                                    start = i;
                                } /* end case '(' */
                                else if (line.charAt(i) == ')') {
                                    String s = line.substring(start+1, i);
                                    s = s.trim();
                                    currentMethod.paramType = s;

                                    // if there's no map, dupe name entry
                                    if (currentMethod.javaMethodName == null)
                                        currentMethod.javaMethodName =
currentMethod.espressoMethodName;

                                    currentMethod.javaMethodName =
currentMethod.javaMethodName.trim();
                                    currentMethod.espressoMethodName =
currentMethod.espressoMethodName.trim();

                                    currentType.addMethod( currentMethod );

                                    i = line.length();
                                    currentMethod = null;
                                } /* end case ')' */

                                i++;
                            } /* end method-define while loop */
                        } /* end current method definition */
                    } /* end parsing current line */

                } /* end parsing non-commented line */

                line = in.readLine();

            } /* end parsing current line */

            System.err.println(" successful.");
        } /* end parsing file */
```

```
        catch(Exception e) {
            System.err.println(" failed.");
            System.err.println("Error parsing runtime library definition file on line
"+
                                lineNum+":\n  "+e.getMessage()+
                               "\nAll runtime classes may not be recognized.");
            e.printStackTrace();
        }

        // try to close the file, squelch any errors
        try {
            in.close();
        }
        catch(Exception e) {}
    }

    /**
     * checks if the Object Type with name key is a valid object type.
     * @param key the Object Type to verify
     * @return an EspressoObjectType representative of key's type
     */
    public EspressoObjectType check(String key) {

      EspressoDataType r = new EspressoDataType(null);

      if   (!typeTable.containsKey(key)) {
          System.err.println(key+": unknown type name (not defined in RTL
definitions)\n");
          System.exit(1);
      }

        EspressoObjectType o = (EspressoObjectType)typeTable.get(key);
        return o;
    }

    public void print() {
        Enumeration e = typeTable.elements();
        while (e.hasMoreElements()) {
            EspressoObjectType ot = (EspressoObjectType)e.nextElement();
            ot.print();
        }
    }

    public static void main( String[] args ) {
        EspressoObjectTypeTable table = new EspressoObjectTypeTable();

        System.err.println(" back in main. ");
        table.print();

        String[] s = table.getRTObjectTypeFileNames();
        for (int i=0; i<s.length; i++)
            System.out.println(".RTL["+i+"] = "+s[i]);
    }
}
```

A.15 EspressoObjectType.java

```
/*
 * EspressoObjectType.java
 *
 * stores legal object types, mappings between espresso and java,
 * and hashtable of legal methods associated with each object.
 *
 * methods are hashed on the espressMethodName of the EspressoObjectMethod
 *
 * version: 1.0 (phil)
 *
 * notes (1.0)
 * -----------
 *
 */

/**
 *
 * @author  phil
 */
import java.util.*;

public class EspressoObjectType {
    public String espressoTypeName;      // typename in espresso (w/out params)
    public String javaTypeName;          // typename w/out parameters
    public String javaConstructorParams; // params for concatenation w/c'tor
                                         //    (usually just '()')
    public Hashtable legalMethods;       // table of EspressoObjectMethod's

    /** Creates a new instance of EspressoObjectType */
    public EspressoObjectType() {
        espressoTypeName = null;
        javaTypeName = null;
        javaConstructorParams = null;
        legalMethods = new Hashtable();
    }

    public void addMethod( EspressoObjectMethod m ) {
        legalMethods.put( m.espressoMethodName, m );
    }

    public boolean hasMethod( EspressoObjectMethod m ) {
        return (legalMethods.containsKey( m.espressoMethodName ));
    }

    public EspressoObjectMethod getMethod( String m ) {
      return (EspressoObjectMethod)legalMethods.get( m );
    }

    /**
     * generates a String for output to Java file to declare an E-Object with
     *  id passed as parameter
     * @param id the ID name to build a declaration for
     * @return String a java declaration (*not* an initialization)
     */
    public String getDecl( String id ) {
        return javaTypeName + " " + id + ";";
    }
```

```
    /**
     * generates a String for output to Java file to initialize the (already
     *  declared EspressoObject with id passed as param
     * @param id the ID name to build an initialization for
     * @return String a java initialization (*not* a declaration)
     */
    public String getInit( String id ) {
        return id + " = new " + javaTypeName + javaConstructorParams + ";";
    }

    /**
     * for debugging.
     * prints class name and all child methods to screen.
     */
    public void print() {
        System.err.println("*** Espresso! type: ["+espressoTypeName+"] ==
["+javaTypeName+"] in Java");
        System.err.println("     constructor parameters:
["+javaConstructorParams+"]");

        Enumeration en = legalMethods.elements();

        while (en.hasMoreElements()) {
            EspressoObjectMethod m = (EspressoObjectMethod)(en.nextElement());
            m.print();
        }
    }
}
```

A.16 EspressoObjectMethod.java

```
/*
 * EspressoObjectMethod.java
 *
 * an EspressoObjectMethod holds a method name, parameter type, and the return
 *  type for a legal method of an EspressoObjectType.
 *
 * one EspressoObjectMethod is constructed for every method defined in the
 *  RTL definition file.
 *
 * Every EspressoObjectType holds a hash table of legal EspressoObjectMethod's
 *  for that particular object, hashed on the methodName.
 *
 * version 1.0 - phil
 *
 * notes (1.0)
 * -----------
 * -only holds type for one parameter since Espresso! does not support
 *   parameter lists.
 *
 */

/**
 *
 * @author  phil
 */
public class EspressoObjectMethod {
    public String espressoMethodName;
    public String javaMethodName;
    public String returnType;
    public String paramType;

    /** Creates a new instance of EspressoObjectMethod */
    public EspressoObjectMethod() {
        espressoMethodName = null;
        javaMethodName = null;
        returnType = null;
        paramType = null;
    }

    public String getReturn() {
      return returnType;
    }

    /**
     * for debugging, prints this method to screen
     */
    public void print() {
       // System.err.println("     - ["+returnType+"] ["+espressoMethodName+"]
["+javaMethodName+"] ["+paramType+"]");
        System.err.println("    - "+returnType+"
"+espressoMethodName+"::"+javaMethodName+" ("+paramType+")");
       // System.err.println("       >returns:["+returnType+"]
>parameter:["+paramType+"]");

    }
}
```

A.17 EspressoToJava.java

```
import java.awt.*;
import java.io.*;
import java.util.*;
import java.lang.Runtime;

/**
 * notes from phil 12-10-03 (11:30pm):
 *  -line 70ish- should decide which runtime library classes to copy based on
 *    EspressoTypeTable -- E2J constructor should take a reference to that table
 *  -around line 215- shouldn't these lines go in the setStateAction() method
 *    instead?  this looks like it'll check the case for a particular action
 *    every time the printCode() method is called.  Instead, i think we should
 *    print the case header in setState() and the close brace in unsetState()?
 */

/**
 * Espresso to JAVA
 * sets up headers and footers for JAVA output file
 * including all common code
 * includes methods for walker to access when translating
 *
 * modified by phil 12-10:
 *  added color declaration and assignments to DECL and INIT
 *
 * @author aaron
 **/
public class EspressoToJava {

    //Create file writer here
    /**
     * array to store which state is active
     * only ONE state can be active at a time
     * 0 = Assignments (run) -- default state
     * 1 = Declarations (globals)
     * 2 = Inits (init)
     * 3 = Screen updates (paint)
     * 4 = Actions (actionPerformed)
     **/
    boolean[] activeState;

    // array to store the filewriters
    FileWriter[] writers;

    FileWriter declWriter;
    FileWriter initWriter;
    FileWriter assWriter;
    FileWriter paintWriter;
    FileWriter actionWriter;

    //the JAVA output file stem
    String file;
    String tab="    ";

    //extensions for the temporary buffers
    private static final String declExt = ".decl.tmp";
    private static final String initExt = ".init.tmp";
    private static final String assExt = ".ass.tmp";
```

```
private static final String paintExt = ".paint.tmp";
private static final String actionExt = ".action.tmp";

//base directory
private static String dirName;

//directory of runtime libraries
private static final String RTL = "RTL/";

/*************************************************************/
//runtime library pathnames
private static String[] rtFiles;

//frame rate of 10 frames/second is standard for human eye
private static final int FRAME_RATE = 100;

String actionID;

public EspressoToJava(String filename, EspressoObjectTypeTable EOTT) {
  file = filename;
  dirName = file + "/";
  File dir = new File(dirName);
  dir.mkdir();
  rtFiles = EOTT.getRTObjectTypeFileNames();
  init();
}

public void init() {

  try {
      declWriter = new FileWriter(dirName + file + declExt);
      initWriter = new FileWriter(dirName + file + initExt);
      assWriter = new FileWriter(dirName + file + assExt);
      paintWriter = new FileWriter(dirName + file + paintExt);
      actionWriter = new FileWriter(dirName + file + actionExt);
  } catch (IOException e) {
      System.err.println("EspressoToJava: could not create file: " + e);
      System.exit(1);
  }

  writers = new FileWriter[5];
  writers[1] = declWriter;
  writers[2] = initWriter;
  writers[0] = assWriter;
  writers[3] = paintWriter;
  writers[4] = actionWriter;

  activeState = new boolean[5];

  for(int i = 0; i < activeState.length; i++) {
      if(i == 0)
        activeState[i] = true;
      else
        activeState[i] = false;
  }

  try {

  ////////////////////BEGIN GENERIC APPLET HEADER///////////////
    /*  note that color names are not declared 'static final' because
     *  the Espresso walker sees color names as ordinary IDs, and therefore
```

```
         *  does not prohibit reassignment.  while they probably shouldn't
         *  be reassigned in practice, in case they are we don't want javac to
         *  complain about something Espresso doesn't see as an error */
        declWriter.write
            ( "import java.awt.*; \n" +
              "import java.awt.event.*; \n" +
              "import java.applet.*; \n" +
              "import java.net.*; \n" +
              "import java.util.*; \n" +
              "import RTL.*; \n" +
              "\n\n" +
              "public class " + file +
              " extends Applet implements Runnable, ActionListener { \n" +
              "\n\n" + tab +
              "Graphics graphicsBuffer = null; \n" + tab +
              "Image imageBuffer = null; \n" + tab +
              "Thread conductor = null; \n" + tab +
              "LinkedList listOfEspressoShapes = null; \n" + tab +
              "public boolean __always_false=false;\n\n"+tab+
              "//all declarations specified by programmer\n"
              );
        ////////////////////END HEADER//////////////////////////

        /////////////////BEGIN SECTION HEADERS///////////////
        initWriter.write
            ( "\n\n"+tab+"public void init () { \n\n"+tab+tab+
                "listOfEspressoShapes = new LinkedList(); \n"
              );

        assWriter.write
            ( "\n\n"+tab+"public void start() { \n"+tab+tab+
              "if(conductor == null ) { \n"+tab+tab+tab+
              "conductor = new Thread(this, \"App\");\n"+tab+tab+tab+
              "conductor.start(); \n"+tab+tab+
              "}\n"+tab+"} // end of start \n\n"+
              "\n\n"+tab+"public void stop() { \n" +tab+
              // "\t\tconductor.destroy();\n" +
              "} // end of stop \n\n" +
             "\n\n"+tab+"public void run () { \n\n"
              );

        paintWriter.write
            ( "\n\n"+tab+"public synchronized void paint ( Graphics g ) { \n\n"+
              tab+tab+"g = getGraphics();\n"+tab+tab+
              "imageBuffer = createImage(getWidth(), getHeight());\n"+tab+tab+
              "graphicsBuffer = imageBuffer.getGraphics();\n"+tab+tab+
              "graphicsBuffer.setColor(getBackground()); \n"+tab+tab+
              "graphicsBuffer.fillRect(0, 0, getWidth(), getHeight()); \n"+tab+tab+
              "for(int __i__ = 0; __i__ < listOfEspressoShapes.size(); __i__++) { \n"+
              tab+tab+tab+

"((EspressoShapeRT)listOfEspressoShapes.get(__i__)).draw(graphicsBuffer);"+tab+
              "}\n"
              );

        actionWriter.write
            ( "\n\n"+tab+"public void actionPerformed ( ActionEvent evt ) { \n"
              );
        } catch (IOException e) { System.err.println(e.getMessage()); }

    }
```

```
public void printCode(String code) {

  try {

      for(int i = 0; i < activeState.length; i++) {
        if(activeState[i] == true) {
            writers[i].write(tab+tab+ code + "\n");
        }
      }
  } catch (IOException e) { System.err.println(e.getMessage()); }
}

/**
 * close
 * print out section footers,
 * close filewriters,
 * compile files into one .java class file
 **/

public void close() {

  try{

  initWriter.write
      (
       tab+"} // end of init() \n\n"
       );

  assWriter.write
      (
       /*          " while(true) {\n" +
       " repaint(); \n" +
       "try { Thread.sleep(" +FRAME_RATE+ "); } \n" +
       "catch (InterruptedException e) { } \n" +
       " } // end of while \n" +
       */tab+"} // end of run \n" +
       "} // end of " + file
       );

  paintWriter.write
      (
       tab+tab+"g.drawImage(imageBuffer, 0, 0, this); \n" +tab+
       "} // end of paint \n\n "
       );

  actionWriter.write
      (
       tab+"} // end of actionPerformed"
       );

  for(int i = 0; i < writers.length; i++)
      writers[i].close();

  FileWriter fw = null;
  try{
      fw = new FileWriter(dirName + file + ".java");
  } catch (IOException e) {
      System.err.println("EspressoToJava error: could not create file "+
                      file + ".java");
```

```
    System.exit(1);
}

FileReader fr = null;
BufferedReader br = null;
try{
    fr = new FileReader(dirName + file + declExt);
}
catch (IOException e) {
    System.err.println("EspressoToJava error: could not open file");
    System.exit(1);
}
br = new BufferedReader(fr);

String line = br.readLine();
while(line != null) {
    fw.write(line + "\n");
    line = br.readLine();
}

try{
    fr = new FileReader(dirName + file + initExt);
}
catch (IOException e) {
    System.err.println("EspressoToJava error: could not open file");
    System.exit(1);
}
br = new BufferedReader(fr);
line = br.readLine();
while(line != null) {
    fw.write(line + "\n");
    line = br.readLine();
}

try{
    fr = new FileReader(dirName + file + paintExt);
}
catch (IOException e) {
    System.err.println("EspressoToJava error: could not open file");
    System.exit(1);
}
br = new BufferedReader(fr);

line = br.readLine();
while(line != null) {
    fw.write(line + "\n");
    line = br.readLine();
}

try{
    fr = new FileReader(dirName + file + actionExt);
}
catch (IOException e) {
    System.err.println("EspressoToJava error: could not open file");
    System.exit(1);
}
br = new BufferedReader(fr);

line = br.readLine();
while(line != null) {
    fw.write(line + "\n");
```

```
        line = br.readLine();
    }

    try{
        fr = new FileReader(dirName + file + assExt);
    }
    catch (IOException e) {
        System.err.println("EspressoToJava error: could not open file");
        System.exit(1);
    }
    br = new BufferedReader(fr);

    line = br.readLine();
    while(line != null) {
        fw.write(line+ "\n");
        line = br.readLine();
    }

    fw.close();
    File temp = new File(dirName + file + assExt);
    temp.delete();
    temp = new File(dirName + file + declExt);
    temp.delete();
    temp = new File(dirName + file + paintExt);
    temp.delete();
    temp = new File(dirName + file + actionExt);
    temp.delete();
    temp = new File(dirName + file + initExt);
    temp.delete();

    /* Create a .html file for the applet and add it to the directory */
    try{
        File htmlFile = new File(dirName + file + ".html");
        FileWriter fww = new FileWriter(htmlFile);
        fww.write
          ( "<html>\n " +
            "<title>" + file + "</title>\n" +
            "<body>\n" +
            "<applet code=" + file + ".class WIDTH=200 HEIGHT=200></applet>\n" +
            "<br><p>source:<a href=\""+file+".java\">"+
            file+".java</a>\n" +
            "</body></html>\n\n"
            );
        fww.close();
    } catch (IOException e) { System.err.println(e.getMessage()); }

    try{
        Runtime rt = Runtime.getRuntime();
        rt.exec("javac "+dirName+ file + ".java");
    }
    catch (Exception e) { System.err.println(e.getMessage()); }


    } catch (IOException e) { System.err.println(e.getMessage()); }


}

/**
 * set/unset state methods
 * the set methods must be called before anyone can print to file
```

```
     * unset methods must be called after printing is finished
     **/

    public void setStateDecl() {
      for(int i = 1; i < activeState.length; i++)
          if(activeState[i] == true && i != 1) {
            System.err.println("EspressoToJava error: trying to set two states at
once");
            System.exit(1);
          }
      activeState[0] = false;
      activeState[1] = true;
      tab="  ";
    }

    public void setStateInit() {
      for(int i = 1; i < activeState.length; i++)
          if(activeState[i] == true && i != 2) {
            System.err.println("EspressoToJava error: trying to set two states at
once");
            System.exit(1);
          }
      activeState[0] = false;
      activeState[2] = true;
    }

    public void setStatePaint() {
      for(int i = 1; i < activeState.length; i++)
          if(activeState[i] == true && i != 3) {
            System.err.println("EspressoToJava error: trying to set two states at
once");
            System.exit(1);
          }
      activeState[0] = false;
      activeState[3] = true;
    }

    public void setStateAction(String id) {
      for(int i = 1; i < activeState.length; i++)
          if(activeState[i] == true && i != 4) {
            System.err.println("EspressoToJava error: trying to set two states at
once");
            System.exit(1);
          }
      activeState[0] = false;
      activeState[4] = true;
      actionID = id;
      try {
      writers[4].write
          ( "\tif ( evt.getSource() == " + actionID + ".getElement() ) {\n" );
      } catch (IOException e) {System.err.println(e.getMessage()); }
    }

    public void unsetStateDecl() {
      activeState[1] = false;
      activeState[0] = true;
      tab="    ";
    }

    public void unsetStateInit() {
      activeState[2] = false;
```

```
      activeState[0] = true;
    }

    public void unsetStatePaint() {
      activeState[3] = false;
      activeState[0] = true;
    }

    public void unsetStateAction() {
      try {
      writers[4].write("\trepaint(); \n }");
      } catch (IOException e) {System.err.println(e.getMessage()); }
      activeState[4] = false;
      activeState[0] = true;
    }
}
```

A.18 EspressoButtonRT.java

```
package RTL;

import java.awt.Button;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Component;
import java.awt.event.ActionListener;
import java.applet.Applet;

/**
 * Espresso Button type
 *
 * @author aaron
 **/

public class EspressoButtonRT extends Button{

    /**
     * constructor for Espresso! button
     **/

    Button element;
    ActionListener act;
    Applet app;
    static final int TEXT_XPAD = 12;
    static final int TEXT_YPAD = 8;
    int x = 0;
    int y = 0;
    int width = 50;
    int height = 10;


    public EspressoButtonRT(  Applet ap) {
      element = new Button();
      act = (ActionListener)ap;
      app = ap;
      element.addActionListener(act);
      element.setLabel("Espresso Button");
      app.add(element);
    }

    public void setWidth(double w) {
      width = (int)w;
      element.setSize( width, element.getHeight());
      element.repaint();
    }
    public void setHeight(double h) {
      height = (int)h;
      element.setSize( element.getWidth(), height);
      element.repaint();
    }
    public int getWidth( ) {
      return width;
    }
    public int getHeight( ) {
      return height;
    }
    public void setX(double X) {
```

```
      x = (int)X;
      element.setLocation(x, element.getY());
      element.repaint();
    }
    public void setY(double Y) {
      y = (int)Y;
      element.setLocation(element.getX(), y);
      element.repaint();
    }
    public int getX( ) {
      return x;
    }
    public int getY( ) {
      return y;
    }

    public void setText(String text) {
      element.setLabel(text);
      sizeToFit();
      element.repaint();
    }

    public String getText( ) {
      return element.getLabel();
    }

    public Button getElement() {
      return element;
    }

    public void action() {}

    public Dimension getPreferredSize() {
      FontMetrics fm = app.getFontMetrics(app.getFont());
      return new Dimension(
             fm.stringWidth(element.getLabel()) + TEXT_XPAD,
                 fm.getMaxAscent() +
                 fm.getMaxDescent() + TEXT_YPAD);
    }

    private void sizeToFit() {
      element.setSize(getPreferredSize());
      Component p = getParent();
      if (p != null) {
          p.invalidate();
          p.doLayout();
      }
    }
}
```

A.19 EspressoColorRT.java

```
package RTL;

import java.awt.Color;

/**
 * Espresso Color type
 *
 * @author aaron
 **/

public class EspressoColorRT {

    /**
     * constructor for Espresso! button
     **/
    public int numColors;
    public Color color;

    public EspressoColorRT(Color newC, int nc) {

        numColors=nc;
      color=newC;
    }

    public EspressoColorRT calcColor(Color c, int nc) {
      int r=(c.getRed()*nc)+(color.getRed()*numColors);
      int g=(c.getGreen()*nc)+(color.getGreen()*numColors);
      int b=(c.getBlue()*nc)+(color.getBlue()*numColors);

      numColors+=nc;
      r/=numColors;
      g/=numColors;
      b/=numColors;

      color=new Color(r,g,b);
      return new EspressoColorRT(color, numColors);
    }

    public Color getColor() {
      return color;
    }
}
```

A.20 EspressoRectRT.java

```
package RTL;

import java.awt.Color;
import java.awt.Graphics;
import java.util.LinkedList;
import java.applet.Applet;
/**
 * espressso Rectangle wrapper class
 *
 * @author aaron
 **/

public class EspressoRectRT implements EspressoShapeRT {

    int x;
    int y;
    int width;
    int height;
    Color color;
    Applet app;
    /**
     * create a new circle with default x,y (0,0) and color black
     * and default size 10x10
    **/

    public EspressoRectRT(LinkedList l, Applet a) {
        x = 0;
        y = 0;
        width = 10;
      height = 10;
        color = Color.black;
      l.add(this);
      app = a;
    }

    public void draw(Graphics g) {
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }


    public void setX(double X){
        x = (int)X;
        app.repaint();
    }

    public void setY(double Y){
        y = (int)Y;
        app.repaint();
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }

    public void setWidth(double w) {
```

```
        width = (int)w;
        app.repaint();
    }
    public int getWidth() {
        return width;
    }

    public void setHeight(double h){
      height = (int)h;
        app.repaint();
    }
    public int getHeight() {
      return height;
    }

    public void setColor(Color c) {
      color = c;
        app.repaint();
    }
}
```

A.20 EspressoShapeRT.java

```
package RTL;

import java.awt.Color;
import java.awt.Graphics;

/**
 * interface for espresso shapes
 *
 * @author aaron
 **/

public interface EspressoShapeRT  {

    public void draw(Graphics g);

}
```

A.22 EspressoStringRT.java

```
package RTL;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.FontMetrics;
import java.util.LinkedList;
import java.applet.Applet;

/**
 * String wrapper class
 *
 * @author aaron
 **/

public class EspressoStringRT implements EspressoShapeRT {

    int x;
    int y;
    String element;
    Color color;
    Applet app;
    /**
     * create a new string with default x,y (0,0) and color black
     **/

    public EspressoStringRT(LinkedList l, Applet a) {
      x = 100;
      y = 100;
      color = Color.black;
      element = "";
      l.add(this);
      app = a;
    }

    public EspressoStringRT(String s, Applet a) {
      app=a;
      FontMetrics fm = app.getFontMetrics(app.getFont());
      x = 100 - fm.stringWidth(s)/2;
      y = 100;
      element = s;
      color = Color.black;
      app = a;
      System.out.println("constructured");
    }

    public void draw(Graphics g) {
      g.setColor(color);
      g.drawString(element, x, y);
    }

    public void setX(double X){
      x = (int)X;
        app.repaint();
    }

    public void setY(double Y){
        y = (int)Y;
        app.repaint();
```

```
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }

    public String getText() {
      return element;
    }

    public void setText( String s ) {
      element = s;
        app.repaint();
    }

    public void setColor(Color c) {
      color=c;
        app.repaint();
    }
}
```

A.23 EspressoTextBoxRT.java

```
package RTL;

import java.awt.event.ActionListener;
import java.awt.TextField;
import java.applet.Applet;

/**
 * Text Box type
 *
 * @author aaron
 **/

public class EspressoTextBoxRT {


    TextField element;
    ActionListener act;
    Applet app;
    int x = 0;
    int y = 0;
    int width = 50;
    int height = 15;

    public EspressoTextBoxRT(Applet a) {
      element = new TextField( );
      act = (ActionListener)a;
      app = a;
      element.addActionListener(act);
      app.add(element);
    }

    public void setText(String t) {
      element.setText(t);
        element.repaint();
    }
    public String getText() {
        return element.getText();
    }

    public void setWidth(double w) {
        width = (int)w;
        element.setSize( width, element.getHeight());
        element.repaint();
    }
    public void setHeight(double h) {
        height = (int)h;
        element.setSize( element.getWidth(), height);
        element.repaint();
    }
    public int getWidth( ) {
        return width;
    }
    public int getHeight( ) {
        return height;
    }
    public void setX(double X) {
        x = (int)X;
        element.setLocation(x, element.getY());
```

```
        element.repaint();
    }
    public void setY(double Y) {
        y = (int)Y;
        element.setLocation(element.getX(), y);
        element.repaint();
    }
    public int getX( ) {
        return x;
    }
    public int getY( ) {
        return y;
    }

    public TextField getElement() {
        return element;
    }


    public void action() {}
}
```

A.24 EspressoCheckBoxRT.java

```
package RTL;

import java.awt.Checkbox;
import java.applet.Applet;

/**
 * Check box type
 *
 * @author aaron
 **/

public class EspressoChkBoxRT {

    Checkbox element;
    Applet app;
    int x = 0;
    int y = 0;
    int width = 80;
    int height = 10;

    public EspressoChkBoxRT( Applet ap) {
      element = new Checkbox();
      app = ap;
      app.add(element);
    }

    public void setText(String text) {
        element.setLabel(text);
      element.repaint();
    }

    public String getText(  ) {
        return element.getLabel();
    }

    public void setWidth(double w) {
        width = (int)w;
        element.setSize( width, element.getHeight());
        element.repaint();
    }
    public void setHeight(double h) {
        height = (int)h;
        element.setSize( element.getWidth(), height);
        element.repaint();
    }
    public int getWidth( ) {
        return width;
    }
    public int getHeight( ) {
        return height;
    }
    public void setX(double X) {
        x = (int)X;
        element.setLocation(x, element.getY());
        element.repaint();
    }
    public void setY(double Y) {
        y = (int)Y;
```

```
        element.setLocation(element.getX(), y);
        element.repaint();
    }
    public int getX( ) {
        return x;
    }
    public int getY( ) {
        return y;
    }

    public int getState( ) {
      return element.getState() ? 1 : 0;
    }

    public Checkbox getElement() {
      return element;
    }

    public void action() {}

}
```

A.25 EspressoImageRT.java

```
package RTL;

import java.net.URL;
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
import java.util.*;
/**
 * espresso image wrapper class
 *
 * @author aaron
 **/

public class EspressoImageRT implements EspressoShapeRT {

    int x;
    int y;
    String source;
    Image img;
    Applet applet;

    /**
     * Espresso Image constructor, setting default x, y to 0,0
     * @param app - the applet containing the image
     **/
    public EspressoImageRT(Applet app, LinkedList l) {
      applet = app;
      img = null;
      x = 0;
      y = 0;
      l.add(this);
    }

    public void setSource(String source) {
      try{
      img = applet.getImage(new URL(source));
      }
      catch (Exception e) { System.err.println("image not found: "+e); }
        applet.repaint();

    }

    public String getSource(){
        return source;
    }

    public void setX(double X){
      x = (int)X;
        applet.repaint();
    }
    public int getX() {
      return x;
    }

    public void setY(double Y){
      y = (int)Y;
        applet.repaint();
```

```
    }

    public int getY() {
      return y;
    }

    public void draw(Graphics g) {
      if(img!=null)
          g.drawImage(img, x, y, applet);
    }

}
```

A.26 EspressoOvalRT.java

```
package RTL;

import java.awt.Color;
import java.awt.Graphics;
import java.util.LinkedList;
import java.applet.Applet;

/**
 * Circle wrapper class
 *
 * @author aaron
 **/

public class EspressoOvalRT implements EspressoShapeRT {

    int x;
    int y;
    int height;
    int width;
    Color color;
    Applet app;

    /**
     * create a new circle with default x,y (0,0) and color black
     **/

    public EspressoOvalRT(LinkedList l, Applet ap) {
      x = 0;
      y = 0;
      width = 10;
      height = 10;
      color = Color.black;
      l.add(this);
      app = ap;
    }

    public void draw(Graphics g) {
      g.setColor(color);
      g.fillOval(x, y, width, height);
    }

    public void setX(double X){
      x = (int)X;
      app.repaint();
    }

    public void setY(double Y){
        y = (int)Y;
        app.repaint();
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
```

```
    public void setHeight(double h) {
        height = (int)h;
        app.repaint();
    }
    public int getHeight() {
        return height;
    }

    public void setWidth( double w) {
      width = (int)w;
        app.repaint();
    }
    public int getWidth( ) {
      return width;
    }

    public void setColor(Color c) {
      color=c;
        app.repaint();
    }
}
```

Appendix B

Who did What

B.1 Coding

A.1 esp	Joya, Erin
A.2 grammar.g	Joya
A.3 walker.g	Joya, Erin
A.4 EspressoException.java	Erin
A.5 EspressoDataType.java	Erin
A.6 EspressoBool.java	Erin
A.7 EspressoNumber.java	Erin
A.8 EspressoString.java	Erin
A.9 EspressoColorPrim.java	Joya
A.10 EspressoColor.java	Joya
A.11 EspressoObject	Phil
A.12 EspressoRunTime.lib	Phil, Aaron
A.13 EspressoSymbolTable.java	Joya, Erin, Phil
A.14 EspressoObjectTypeTable.java	Phil
A.15 EspressoObjectType.java	Phil
A.16 EspressoObjectMethod.java	Phil
A.17 EspressoToJava.java	Aaron, Phil
A.18 EspressoButtonRT.java	Aaron
A.19 EspressoColorRT.java	Aaron
A.20 EspressoRectRT.java	Aaron
A.21 EspressoShapeRT.java	Aaron
A.22 EspressoStringRT.java	Aaron
A.23 EspressoTexboxRT.java	Aaron
A.24 EspressoCheckboxRT.java	Aaron
A.25 EspressoImageRT.java	Aaron
A.26 EspressoOvalRT.java	Aaron

B.2 Documentation

Chapter 1	Joya, Erin, Phil, Aaron
Chapter 2	Phil
Chapter 3	Joya, Erin, Phil, Aaron (updated after review by Erin)
Chapter 4	Aaron
Chapter 5	Erin
Chapter 6	Erin
Chapter 7	Joya, Erin, Phil, Aaron
Compilation, editing	Erin, Phil

B.3 Sample Applets/Testing

ColorApplet.esp	Joya
Calcamajig.esp	Phil
MovingTarget.esp	Erin
Baby.esp	Aaron

Testing was a collaborative effort.