Using the Objectdraw dialect in Grace

Kim Bruce

1 Introduction

The purpose of this document is to explain how to use the *objectdraw* library in Grace[1]. We assume the reader is familiar with Grace. While not necessary, it will be helpful if the user is already familiar with the objectdraw library in Java. A brief description is available at http://eventfuljava.cs.williams.edu/library.html. More detailed information is available in the text Java: An Eventful Approach [2]. The author is currently writing a revised version of that text, **Programming with Grace**, that is based on Grace and uses the Grace version of the objectdraw library. It is available at http://www.cs.pomona.edu/~kim/ProgrammingWithGrace/bookmain.pdf. In particular the appendix of the text presents a summary of objectdraw commands.

2 Running Grace Programs on the Web

To run Grace programs, go to http://web.cecs.pdx.edu/~grace/minigrace/exp/ using the Chrome web browser. (Up-to-date versions of Firefox will likely work as well, but Safari will not.). Be sure that your pop-up blocker is turned off if you are going to use the objectdraw library. Otherwise you will get an internal error in objectdraw referring to the "dom".

When you go to the Grace Editor web site the window will be divided into three panels. On the left will be the file pane with a a list of open files, the top right is the editor pane and will show the file you are currently working on, while the bottom right is the output pane, which will show error messages and program output.

To start a program, click on the "New" button in the Files pane (also indicated by a circled plus sign) and give your program a name.

If the "hello world" program is not already showing, type

```
print "hello world!"
```

in the editor pane. You can then compile and execute your program by clicking on the "Run" button at the bottom of the editor pane. The output should show up in the editor pane.

A command line version of the Grace compiler is available as well. Go to https://github.com/gracelang for details on installing this compiler.

3 Overview of Objectdraw dialect

The objectdraw library allows users to create applications using simple and intuitive graphics primitives.

The user creates graphic objects including rectangles, ovals, lines, and text to label them on a DrawingCanvas in a window. The main program will be defined in an object that is of type GraphicApplication. The program is written by inheriting from the class graphicApplication. Figures 1

```
dialect "objectdraw"

// Simple program that draws a filled rectangle when the mouse is pressed.
// If the user drags the mouse then the rectangle will move with it
object {
    inherits graphicApplication.size (400, 400)

    // set title of window displaying graphics (not in all browsers)
    windowTitle := "Simple Objectdraw Demo"

    text.at (20 @ 20) with ("Press mouse and drag") on (canvas)

    // item to be moved
    var cloth

    // When the user presses mouse, create new rectangle at that point
    // to be moved by dragging mouse
    method onMousePress (mousePoint) {
        cloth := filledRect.at (mousePoint) size (100, 100) on (canvas)
        cloth.color := red
    }

    // When mouse is dragged, rectangle follows it
    method onMouseDrag (mousePoint) {
        cloth.moveTo (mousePoint)
    }

    // required to pop up window and start graphics
    startGraphics
}
```

Figure 1: Program using objectdraw to drag rectangles in a window – type info omitted

and 2 provide two versions of a simple program using the objectdraw library. The first one omits all type information, while the second includes all static type information. The two versions provide identical behavior.

When an object inheriting from `graphicApplication` is initialized, a window with the title "Simple Objectdraw Demo" will be created. The instructions will be displayed in the upper left corner. (The origin of the coordinates is the upper left corner of the window, with the x coordinate increasing to the right and the y coordinate increasing as you move down.) Each time the user presses the mouse button down and then drags the mouse, a red rectangle will be created and will follow the mouse until the mouse button is released. The following is a brief explanation of the code in the program.

Every object that corresponds to a graphics program should inherit from objectdraw's graphicApplication class, providing the width and height of the window.

After that you can declare whatever objects are needed for your application (in this case only a variable cloth representing red squares to be dragged), as well as include initialization code. Here the initialization code sets the window title and then displays a text item with instructions to the user at coordinates (20,20)

The method onMousePress is executed when the mouse button is depressed, while onMouseDrag is executed repeatedly while the mouse is dragged (with the button depressed).

```
dialect "objectdraw"

// Simple program that draws a filled rectangle when the mouse is pressed.
// If the user drags the mouse then the rectangle will move with it
object {
    inherits graphicApplication.size (400, 400)

    // set title of window displaying graphics (not in all browsers)
    windowTitle:= "Simple Objectdraw Demo"

    text.at (20 @ 20) with ("Press mouse and drag") on (canvas)

    // item to be moved
    var cloth: Graphic2D

    // When the user presses mouse, create new rectangle at that point
    // to be moved by dragging mouse
    method onMousePress (mousePoint: Point) -> Done {
        cloth := filledRect.at (mousePoint) size (100, 100) on (canvas)
        cloth.color := red
    }

    // When mouse is dragged, rectangle follows it
    method onMouseDrag (mousePoint: Point) -> Done{
        cloth.moveTo (mousePoint)
    }

    // required to pop up window and start graphics
    startGraphics
}
```

Figure 2: Program using objectdraw to drag rectangles in a window – static type info included

In this program, when the mouse is depressed a new red square is created at the location of the mouse. When the mouse is dragged the last red square created is moved to the new location of the mouse (continuously during the dragging).

The command startGraphics must be executed at the end of the object definition in order to create the window and start the graphics.

4 Graphics Classes

While the library has a number of types and classes, we will only describe here the ones likely to be of interest to end users of the objectdraw library. The classes will generate both framed (outlined) and filled (solid) rectangles, ellipses, and arcs as well as straight lines, text items, and images from the web that can be put on the canvas.

The type and the associated classes for generating graphical objects are provided in the Objectdraw Quick Reference document at
http://www.cs.pomona.edu/~kim/GraceStuff/ObjectdrawGraceQuickReference.pdf.

Seven classes generate two dimensional graphic objects, which all have type Graphic2D. The objectdraw library also supports lines and text items to be displayed on the canvas. Graphics items may be in any color. All of these, including information on colors, are available in the quick reference document.

5 Responding to Mouse Events

The objectdraw library is designed to be used in interactive programs. Thus it provides methods that will be called by the system to respond to mouse events. Methods to respond to mouse presses, releases, clicks, moves, drags, and entering or leaving the window are defined in class graphicApplication, though their default behavior is to do nothing. They should be overridden as desired in any object the inherits from graphicsApplication.

Here is a sample method that prints the location of the mouse when the mouse button is pressed:

```
method onMousePress (pt: Point) -> Done {
    print "mouse was pressed at {pt}"
}
```

The methods to respond to these events are all listed in the quick reference document. The appropriate methods will be called by the system each time the corresponding mouse event occurs. The parameter will be supplied by the system and corresponds to the location of the mouse when the event occurred. Users should be aware that a mouse click generates three events: a press event, a release event, and then a click event.

6 Animations

The minigrace compiler does not currently support true parallelism and concurrency (though we hope to add these in the near future). For now, animations are supported by creating timers to trigger actions. This is supported by the library `Animation.grace`. Meanwhile, readers are encouraged to look at the code in `Animation.grace` which supports higher level constructs based on timers. We suggest that you look at the appropriate chapter of the draft text, Programming with Grace, to see how to use these facilities.

7 Summary

The objectdraw library has been ported from Java to Grace. The types, classes, and methods in Grace are similar to those in Java, but take advantage of features in Grace like multi-part method names and methods that look like assignment statements.

Many sample programs using the objectdraw library in Grace are available in the text Programming with Grace at http://www.cs.pomona.edu/~kim/ProgrammingWithGrace/bookmain.pdf.

References

[1] A. P. Black, K. B. Bruce, M. Homer, J. Noble, A. Ruskin, and R. Yannow. Seeking Grace: A new object-oriented language for novices. In *ACM Conference on Computer Science Education (SIGCSE)*, 2013.

[2] K. Bruce, A. Danyluk, and T. Murtagh. *Java: An Eventful Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.