Introduction

This document provides a brief outline of the public constructors and methods for the major classes in the objectdraw library for Grace.

Application

An object of type Application pops up an empty window. The user may add GUI components to this window. (See details in appendix ??). An object of type Application is normally created by writing an object (or class) that inherits from the following class which pops up a window with dimensions given by dimensions' and has title given by initialTitle

```plaintext
class applicationTitle(initialTitle: String)
    size (dimensions': Point)
```

This type contains the following simple methods. Other methods will be described in Appendix ??

```plaintext// A standalone window which contains other components.
type Application = Container & type {

    // The title of the application window.
    windowTitle --> String

    // Set the title of the application window.
    windowTitle:= (value: String) --> Done

    // width of the window
    width --> Number

    // height of the window
    height --> Number

    // Create window with listeners for mouse enter and exit actions
    startApplication --> Done

    // Close the window
    stopApplication --> Done
}
```

GraphicApplication

An object of type GraphicApplication pops up a window with a canvas installed for drawing on. It is also capable of responding to mouse events.
3.1 Creating a GraphicApplication

Objects that extend this class will pop up a window with a canvas for drawing when the `startGraphics` method is requested.

```
// Create a graphic application with a canvas that is has size given by dimensions
class graphicApplicationSize( dimensions: Point ) -> GraphicApplication
```

The class provides methods for accessing the canvas as well as providing methods for mouse-event handling.

```
// canvas holds graphic objects on screen
canvas -> DrawingCanvas

// Respond to a mouse click (press and release) in the canvas at the given point.
onMouseClick (mouse: Point) -> Done

// Respond to a mouse press in the canvas at the given point.
onMousePress (mouse: Point) -> Done

// Respond to a mouse release in the canvas at the given point.
onMouseRelease (mouse: Point) -> Done

// Respond to a mouse move in the canvas at the given point.
onMouseMove (mouse: Point) -> Done

// Respond to a mouse drag (move during press) in the canvas at the given point.
onMouseDrag (mouse: Point) -> Done

// Respond to a mouse entering the canvas at the given point.
onMouseEnter (mouse: Point) -> Done

// Respond to a mouse exiting the canvas at the given point.
onMouseExit (mouse: Point) -> Done

// must be invoked to create window and its contents as well as prepare the window to handle mouse events
startGraphics -> Done
```

The mouse event-handling methods (onMouseClick, onMousePress, etc.) have a default behavior of doing nothing. The programmer should override any of these methods that are relevant, and provide them with the desired behavior in the subobject or subclass.

4 Animator

The animation library provides the following type definition and corresponding method implementations.

4.1 Creating an Animator object

```
// class to generate animator of objects with default pause of pauseTimeMS' milliseconds
class animator.pausing(pauseTimeMS') -> Animator {
```
4.2  Types used in methods of type Animator

The following type definitions are used in the methods of Animator:

// type of a block that takes no parameters and completes an action
type Procedure = {apply -> Done}

// type of a block that takes no parameters and returns a boolean
type BoolBlock = {apply -> Boolean}

// type of objects that can be animated
type NumberBlock = {apply -> Number}

4.3  Methods available in objects of type Animator

The animation library supplies the following methods. If the library is promoted using a statement of the form `import "animation" as animator` then the methods can be invoked by statements like

```
animator.while {...} pausing (...) do { ... }
```

type Animator = {
// Repeatedly execute block while condition is true
while (condition: BoolBlock) pausing (pauseTime: Number) do (block: Block) -> Done

// Repeatedly execute block while condition is true, pausing pauseTime between iterations
// when condition fails, execute endBlock.
while (condition: BoolBlock) pausing (pauseTime: Number) do (block: Block)
  finally(endBlock: Block) -> Done

// Repeatedly execute block while condition is true
// pausing variable amount of time (obtained by evaluating timeBlock) between iterations
// when condition fails, execute endBlock.
while (condition: BoolBlock) pauseVarying (timeBlock: NumberBlock) do (block: Block) -> Done

// Repeatedly execute block while condition is true
for[[T]] (range': Sequence[[T]]) pausing (pauseTime: Number) do (block: Block[[T,Done]]) -> Done

// Repeatedly execute block while condition is true
// when condition fails, execute endBlock.
for[[T]] (range': Sequence[[T]]) pausing (pauseTime: Number) do (block: Block[[T,Done]])
  finally (endBlock: Block) -> Done

}

5  DrawingCanvas objects

Canvases are for drawing on. All graphic items are displayed on a canvas.

5.1  Constructing a DrawingCanvas object

Canvases are created when an object is created from class graphicApplication and may be requested from such an object via a request to canvas.
5.2 Methods available in objects of type `DrawingCanvas`

The following are the most useful methods available on `DrawingCanvas`:

- // clear the canvas
  `clear` -> Done

- // return the current dimensions of the canvas
  `width` -> Number
  `height` -> Number
  `size` -> Point

6 Graphic objects

6.1 Methods available for all Graphic objects

- // location of object on screen
  `x` -> Number
  `y` -> Number
  `location` -> Point

- // Add this object to canvas c
  `addToCanvas (c: DrawingCanvas)` -> Done

- // Remove this object from its canvas
  `removeFromCanvas` -> Done

- // Is this object visible on the screen?
  `isVisible` -> Boolean

- // Determine if object is shown on screen
  `visible := (_: Boolean)` -> Done

- // move this object to newLocn
  `moveTo (newLocn: Point)` -> Done

- // move this object dx to the right and dy down
  `moveBy (dx: Number, dy: Number)` -> Done

- // Does this object contain locn
  `contains (locn: Point)` -> Boolean

- // Does other overlap with this object
  `overlaps (other: Graphic2D)` -> Boolean

- // set the color of this object to c
  `color := (c: Color)` -> Done

- // return the color of this object
  `color` -> Color

- // Send this object up one layer on the screen
  `sendForward` -> Done

- // send this object down one layer on the screen
7 GRAPHIC2D OBJECTS

sendBackward -> Done

// send this object to the top layer on the screen
sendToFront -> Done

// send this object to the bottom layer on the screen
sendToBack -> Done

// Return a string representation of the object
asString -> String

7 Graphic2D Objects

7.1 Additional methods available for Graphic2D objects

Graphic2D objects have all of the methods of Graphic plus the following:

// dimensions of object
width -> Number
height -> Number
size -> Point

// Change dimensions of object
size := (dimensions: Point) -> Done
width := (width: Number) -> Done
height := (height: Number) -> Done

7.2 Classes generating Graphic2D objects

// class to generate framed rectangle at (x',y') with size dimensions' created on canvas'
class framedRectAt (location': Point) size (dimensions': Point)
on (canvas': DrawingCanvas) -> Graphic2D

// class to generate filled rectangle at (x',y') with size dimensions' created on canvas'
class filledRectAt (location': Point) size (dimensions': Point).
on (canvas': DrawingCanvas) -> Graphic2D

// class to generate framed oval at (x',y') with size dimensions created on canvas'
class framedOvalAt (location': Point) size (dimensions': Point)
on (canvas': DrawingCanvas) -> Graphic2D

// class to generate filled oval at (x',y') with size dimensions; created on canvas'
class filledOvalAt (location': Point) size (dimensions': Point)
on (canvas': DrawingCanvas) -> Graphic2D

// class to generate framed arc at (x',y') with size dimensions'
// from startAngle radians to endAngle radians created on canvas'
// Note that angle 0 is in direction of positive x axis and increases in
// angles go clockwise.
class framedArcAt (location': Point) size (dimensions': Point)
  from (startAngle: Number) to (endAngle: Number) on (canvas': DrawingCanvas) -> Graphic2D

// class to generate filled arc at (x',y') with size dimensions'
// from startAngle degrees to endAngle degrees created on canvas'
// Note that angle 0 is in direction of positive x axis and increases in
8  Line Objects

8.1  Additional methods available for Line objects

Line objects have all of the methods of Graphic plus the following:

// return the start and end of line
start → Point
end → Point

// set start and end of line
start:= (start': Point) → Done
end:= (end': Point) → Done
setEndPoints (start': Point, end': Point) → Done

8.2  Class generating a Line object

// Create a line from start' to end' on canvas'
class lineFrom (start': Point) to (end': Point) on (canvas': DrawingCanvas) → Line

9  Text Objects

9.1  Additional methods available for Text items

Text objects have all of the methods of Graphic plus the following:

// return the contents displayed in the item
contents → String

// reset the contents displayed to be s
contents:= (s: String) → Done

// return width of text item
width → Number

// return size of the font used to display the contents
fontSize → Number

// Set the size of the font used to display the contents
fontSize:= (size: Number) → Done

9.2  Class generating a Text object

// class to generate text at location' on canvas' initially showing contents'
class textAt (location': Point) with (contents': String) on (canvas': DrawingCanvas)
   → Text
10 Colors

10.1 Type Color

Elements of type Color represent (immutable) colors that can be associated with geometric objects.

```javascript
 type Color = {
   red -> Number // The red component of the color.
   green -> Number // The green component of the color.
   blue -> Number // The blue component of the color.
 }
```

10.1.1 An object that can create Colors

The object colorGen contains a class for generating colors, some predefined colors, and a method for generating random colors.

Elements of type Color can be created using the following class, where the red, green, and blue components are numbers between 0 and 255

```javascript
 class colorGen.r(r')g(g')b(b') -> Color
```

A random color will be generated when colorGen.random is evaluated.

10.1.2 Predefined colors

The following colors are predefined in objectdraw and may be used in programs: colorGen.white, colorGen.black, colorGen.green, colorGen.red, colorGen.gray, colorGen.blue, colorGen.cyan, colorGen.magenta, colorGen.yellow, and colorGen.neutral.

11 Sounds

11.1 Type Audio

Elements of type Audio represent sounds that can be played.

```javascript
 type Audio = {
   // An audio file that can be played
   source -> String // The source URL of the audio file.
   source:= (value: String) -> Done // Play the audio.
   play -> Done // Pause playing the audio.
   pause -> Done // does the audio loop back to the start?
   isLooping -> Boolean // determine whether the audio will loop
   looping:= (value: Boolean) -> Done // whether the audio has finished
   isEnded -> Boolean
 }
```

11.1.1 A class generating sounds

The class audioUrl generates objects of type Audio.

```javascript
 class audioUrl (source: String) -> Audio
   // Creates an audio file from source, which is the web URL of a .wav file
```