CSP 125 Lab 4: Using Objects

Introduction

This lab has one goal and that is to give you practice reading and writing programs with objects. To do this, the programming this week will involve using the EzWindows library written by the authors of the course textbook. The EzWindows library allows us to create programs that have windows of their own in which graphics (e.g., rectangles) can be drawn.

Along with the programming in this lab, another programming tool is introduced. The new programming tool is called **`make`**. Make is used to automate the compilation (i.e., making) of programs.

1. Preparation

Before you arrive at your assigned lab section for the week, you should have done the previous labs, and read the textbook. Of particular interest for this week's lab is section 3.9. Of course, much of the textbook material that precedes that section is relevant.

2. Simple: the first EzWindows program

☐ Begin by **creating a directory** for this week's lab work and change to that directory.

☐ **get the files used for this lab** The files can be copied from `~bouvier/pub/lab04` The files are also available from (depending on your section) the lab webpage or WebCT. The files needed are:

```
simple.cpp
simpleblink.cpp
Makefile
```

☐ **verify that you successfully copied the files** using the `ls` command.

About `make`

As mentioned in the introduction, the make utility automates the compilation of programs. Make only does what it is told to do. The normal way to give make those instructions is with a makefile. The makefile we are using is named `Makefile`. It is just a text file, so you could look at it, edit it, whatever. But, your lab will go better if you don't change it (at least not yet). When you run make you will see the steps used to compile the program. Any error messages will appear on the screen just as before.

To run make you can type `make` by itself. In this case, `make` will compile the default target program. For lab this week, the default target is `simple`.

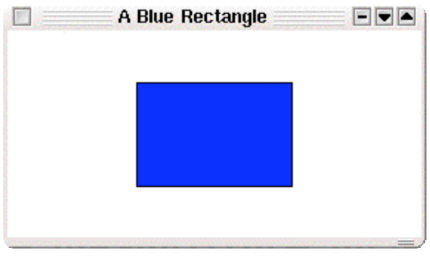
To run make you can type `make` with a target name. For lab this week, the target names are:

```
simple       - (default) compile simple.cpp to make simple
simpleblink  - compile simpleblink.cpp to make simpleblink
house        - compile (non-existent) house.cpp to house
clean        - clean up (deletes non-source code files)
```

☐ **compile the `simple.cpp` program** using make. All you have to do is type `make`

☐ **verify that you successfully made simple** (use `ls`)

☐ **run simple** All you have to do is type `simple`
When the program runs a window opens with the title "A Blue Rectangle". Within that window is drawn a blue rectangle.



The code for the simple program appears below.

Most of the 'interesting' stuff of the simple program happens in the first four lines of the code in the `ApiMain` function and explained within the boxes.

```
// file: simple.cpp
// date: spring 2005
// CSP125 Lab04 (week of 31Jan05)
//
// this program is a simple EzWindows application

#include <iostream>

#include "rect.h"

int ApiMain() {

  SimpleWindow W("A Blue Rectangle", 8, 4);
  W.Open();
  RectangleShape R(W, 4.0, 2.0, Blue, 3, 2);
  R.Draw();

  cout << "type Ctrl-C to remove the display and exit" << endl;
  char AnyChar;
  cin >> AnyChar;
  W.Close();

  return 0;
}
```

W is a **SimpleWindow** object created with the title and size indicated.

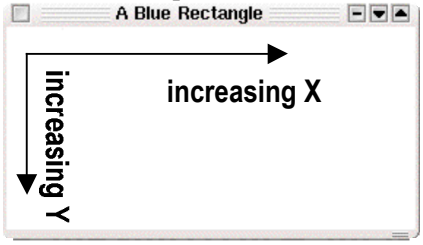
W object is told to Open.

R is a **RectangleShape** object created in the W window with the position, color, and size indicated.

R object is told to Draw itself

Other points to be aware of:

- **the include file used** – the program `#includes "rect.h"` which is where the SimpleWindow and RectangleShape classes are defined.
- **the name of the function** – this program has a function called ApiMain instead of the "main" function used in all previous labs. This is a requirement of the EzWindows library.
- **EzWindows coordinate system** – the EzWindows library specifies a coordinate system as illustrated at the right



- the **behavior of the program** – this program doesn't re-draw the content of the window if the window is re-sized, covered and then exposed, etc. It also doesn't respond to the close button on the window.

2. a closer look at class RectangleShape

At the right is the UML diagram for the RectangleShape class. The three parts of the UML diagram represent three aspects of the class. The top box is the class name. The middle box lists the state information (i.e., values, data members, or attributes) for each object of this class. The bottom box lists the actions (a.k.a., methods, set of operations) for objects of the class.

class RectangleShape
- float width; - float height; - color Color; - bool Border; - Position Center; - SimpleWindow& window;
+ RectangleShape(....) + SetSize(float, float) + SetPosition(float, float) + SetColor(color); + SetBorder(); + ClearBorder(); + Draw(); + Erase();

Some of the values of RectangleShape class are obvious. For example, the first three values (i.e., width, height, and color) are all self-explanatory. The last three are not as obvious.

The fourth member of RectangleShape, `bool Border`, indicates if a border is drawn around the rectangle. By default, every RectangleShape object has a black border drawn around it. The border is 'turned-off' with the `ClearBorder()` method and 'turned-on' with the `SetBorder()` method.

The position of a RectangleShape object is determined by setting the position of the center of the RectangleShape.

More information about class RectangleShape appears on pages 902 and 903 of the textbook.

☐ **Add your name and date to the comments of the `simple.cpp` source code** file …

☐ **try some things** with the simple program

- comment out **`R.Draw();`** … save, compile, run …. what happens?
- change the size of the rectangle… save, compile, run
- change the size of the window… save, compile, run
- change the title of the window… save, compile, run
- try other colors (such as, Red, Yellow, Green, Magenta, Black, White)
- make multiple rectangle objects at one time

☐ **submit the edited simple program** using the submit technique for your instructor

3. Some other EzWindows classes

At the right are UML diagrams for three other EzWindows classes: SimpleWindow, TriangleShape, and Label.

class SimpleWindow
- float width; - float height; - string title; - SimpleWindow& window;
+ SimpleWindow (….) + Open(); + Close();

You may notice that the **TriangleShape class** is very similar to the RectangleShape class. Practically speaking, the only differences are the name of the class, the size is set by one value, and what it does (i.e., draw an equilateral triangle, not a rectangle). The triangle that is drawn will always be a equilateral triangle with the same orientation (these are limitation of the class in the EzWindows library).

The default size is one. That is, each side of the triangle will have length of 1. Use the SetSize() method to change the size of the triangle.

To use the TriangleShape class, you must `#include "triangle.h"`

class TriangleShape
- float width; - float height; - color Color; - bool Border; - Position Center; - SimpleWindow& window;
+ TriangleShape(….) + SetSize(float) + SetPosition(float, float) + SetColor(color); + SetBorder(); + ClearBorder(); + Draw(); + Erase();

Label class objects are used to draw text in a window. A Label object is created by specifying a window, a position (x, y), the label text, the text color, and the background color. The following code shows an example label of creating a Label object at position x=4, y=2 with the text "the label", text color is Yellow, and background color is Black. If you don't specify the colors, the default is Black text on a White background.

```
SimpleWindow W();
W.Open();
Label L(W, 4.0, 2.0, "the label", Yellow, Black);
L.Draw();
```

To use the Label class, you must `#include "label.h"`

class Label
- float width; - float height; - string label; - SimpleWindow& window;
+ Label(….) + SetSize(float); + Draw(); + Erase();

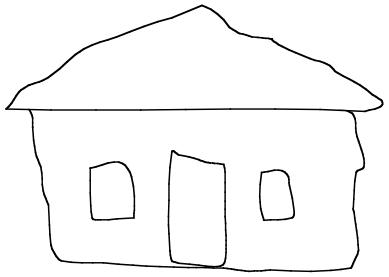
4. Building a house

☐ **copy the `simple.cpp` program to a file named `house.cpp`** …
type `cp simple.cpp house.cpp`

☐ **Add your name and date to the comments of the `house.cpp`** file …

☐ **modify `house.cpp`** to create the image of a house. You can create the house of the image below, or you can create your own house design. (Here is your chance to do 'extreme makeover: home edition'.) The final image of the house should have:

- a rectangle for the wall
- rectangles for windows, with borders
- a rectangle for the door, without borders
- title the window with your name
- more than one color
- a label
- a triangle roof



☐ **compile the `house.cpp` program** using make … type `make house`

☐ **run house** … type `house`

☐ **submit the final version of your `house` program** using the submit technique for your instructor

5. Completing the Lab

Before you leave, make sure you have submitted your programs. You should have submitted:

- `simple.cpp`
- `house.cpp`

6. Want to do more? : simple animation in EzWindows

The house program creates a single image and even though the individual rectangles are drawn one at a time, it all happens so fast that it appears all at once. If we had a way to slow the drawing down, we would be able to create an animation. Fortunately, there is an easy way to have the program pause for some number of seconds. That way is with the `sleep()` function.

☐ **compile the `simpleblink.cpp` program** using make … type `make simpleblink`

☐ **run simple** … type `simpleblink`

☐ **Add your name and date to the comments of the `simpleblink.cpp`** file …

☐ **modify `simpleblink.cpp`** to create an animation. For this program, you can do what you want. Here are some ideas:

- shapes slide across the window
- shapes shrink or grow in size
- shapes change color
- all of the above

☐ **submit the final version of your `simpleblink` program** using the submit technique for your instructor