O-O Windowing Decomposition

The Assignment -1

The goodie is to build an object-oriented model or decomposition of a windowing system whose requirements were described thusly:

Build a modular, object-oriented decomposition of a nice windowing system. When appropriate, the modules should be abstract data types or classes.

The Assignment -2

For each module, specify whether it is a type or object definition and give all needed operations.

Do NOT consider data structures or flow of control. This is intended to be an exercize in data abstraction and object-oriented thinking.

The Assignment -3

Your windowing system should be in the form of a collection of routines invokable by any application that wants to use windowing as its communication with the user.

The Assignment -4

Your windows should be

- **rectangular**
- **creatable to any size and position**
- **resizeable**
- **movable**
- **closable to be an icon**
- **openable from being an icon**
- **scrollable in all four directions**
- **active only when pointed at by mouse**

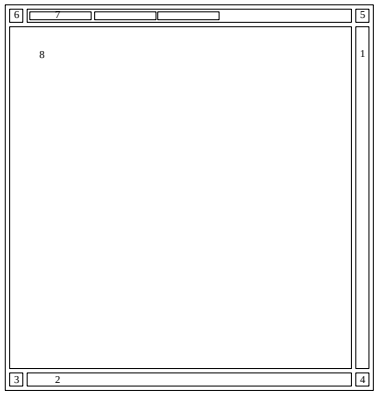
The Assignment -5

Any issue that is not covered herein, you may resolve it any way you please and build your abstraction accordingly. However, note that the ONLY issues that you are to decide are semantics and NOT implementation.

For hints you might want to look at MS-Windows, Macintosh windows, SunView, NeWS, X-windows, etc.

The next slide shows a picture of a typical window as I see it.

Diagram of Window and its Parts



Numbers refer to labels of attributes


```
class (...) window
/*  attribute           type */

    lowerLeftCorner     orderedPair
    upperRightCorner    orderedPair

    isIcon              boolean
    isAscii             boolean
    isActive            boolean
```

```
vertScrollBar       scrollBar(vert)  /* 1 */
horizScrollBar      scrollBar(horiz) /* 2 */
iconifier           pushButton       /* 3 */
resizer             draggingButton   /* 4 */
mover               draggingButton   /* 5 */
closer              pushButton       /* 6 */
menues              listOf(menu)     /* 7 */

contents            picture          /* 8 */
visiblePortionLLC   orderedPair
visiblePortionURC   orderedPair

cursorPosition      orderedPair
```



```
/*  procedures */

    create(lowerLeftCorner,upperRightCorner:
           orderdPair)
    close
    refresh /* invokes refresh of components */
    iconify
    deiconify
    scrollHorizontally(percentage:real)
    scrollVertically(percentage:real)
    resize(newLowerRightCorner:orderedPair)
    move(newUpperRightCorner:orderedPair)
    makeGhost
```

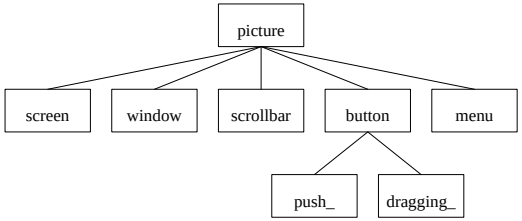


```
moveCursorIntoWindow(Position:orderedPair)
moveCursor(newPosition:orderedPair)

makeActive
makeInactive
```



```
    makeContentsAsciiTerminal
        /* after doing this, all the usual terminal
        functions are available and window can
        be target of stdout */

        /* Why is this a procedure and NOT a
        subclass? */

    bind
        /* for process to bind to window as an
        output device */
    getContents /* of bound window */
end window;
```

Class Inheritance Hierarchy



class picture

/* attributes and operations for defining a single screen-displayable picture */

/* this will be the parent class of all classes whose objects have a pictorial representation */

/* among the attributes of a picture are its dimensions so that once its location is determined, the exact screen coverage can be calculated */

/* among the operations are: */

picture procedure compose(picture p1,p2, orderedPair locationP1,locationP2);
/* compose p1 in front of p2 at indicated locations into a single picture */

/* if p1 and p2 do not overlap in their locations, then the order is irrelevant */

/* so that new pictures can be built by combining others. */ end picture;

class (picture) screen ...;
class (picture) window ...;
class (picture) scrollbar ...;
class (picture) button ...;
class (picture) menu ...;
class (button) pushButton ...;
class (button) draggingButton ...;

/* Now any application object that wishes to build a pictorial user interface needs only to bind to a particular window, as one binds to a file for output. Once a window is bound, an operation can be used to get to the picture object that is its contents, and then the picture operations can be used to update this picture (the window contents) to be whatever is desired. */

/* in Main program of application: */

/* Simulation of Electric Circuit Diagrams */

```
window outputDevice
picture windowContents

outputDevice := bind ...
windowContents := outputDevice.getContents
```


/* Usually the window contents picture is updated by composed pictures built out of application object pictures such as might be generated from the following classes: */

```
class (picture) circuitDiagram ...;
/* contains all the circuitElements for making
up one circuit */

class (picture) circuitElement ...;
/* contains all properties that ALL circuit
elements have independent of their particular
function */
```


/* a particular circuit element is a subclass of circuitElement which makes it also a subsubclass of a picture */

```
class (circuitElement) wire ...;
class (circuitElement) transistor ...;
```

/* Each object is responsible for doing its own behavior in a simulation and updating its own picture to reflect its new state at anytime the state changes. */

/* Each object is responsible for inserting itself into the circuit diagram and connecting itself to its neighbors AND updating its own picture to reflect this connectivity. */


```
class screen /* actually window_manager */
/*  attribute           type */

    contents            listOf(window)
    inFrontOf           setOfPairOf(window)
        /* (w1,w2) in inFrontOf if w1 is in front of w2 */

    cursorLocation      orderedPair  /* OR
                        window /* window
                            containing cursor */
```

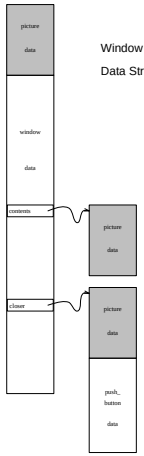


```
/*  procedures */

    push(p1:picture)
    rotate(p1,p2:picture)
    refresh
        /* refresh of any object invokes refresh
        of its component pictures */

/*  etc... */
end screen;
```


The next slide shows a diagram of the data structure of a window object.

picture is a super class of window and of push_button.

Therefore, you see picture data at the bottoms of the window object and of the contents and closer components of the window object; the types of the components are picture and push_button respectively.



Window Object
Data Structure