```
/****
 *
 * Class View is an abstract parent class for view classes in an MVP design.
 * See Fisher SE lecture notes for further discussion of the MVP design
 * methodology.
 *
 */

#ifndef ViewIncluded
#define ViewIncluded

#include "std-macros.h"
#include "model.h"
#include "iv-support.h"


class View {
 public:
    View(Screen* s = NULL, Model* m = NULL);
    /*
     * Construct a view with the given Screen and Model.  Initialize other data
     * members to NULL or false, as appropriate.  The given screen is that in
     * which the view will insert itself to be phyiscally displayed.  The model
     * is the companion model in an MVP design.
     *
     * pre: s != NULL;
     *
     * post: (this'->s == s) and (this'->m == m) and
     *       (this'->w == NULL) and (this'->shown == false) and
     *       (this'->editable == false);
     */

    virtual ~View();
    /*
     * Shallow distruct this.  I.e., do not delete the screen or model.
     *
     * post: freed(this');
     */

    virtual void Run();
    /*
     * Run this by entering the screen's event loop, which will in turn display
     * the window of this and any interactors within the window.  In a
     * non-modal GUI, only the topmost view will be run.  In a modal GUI, Run
     * is specialized in View subclasses to contain a modal event loop that
     * locks out events from other view components until the event loop has
     * been completed, e.g., by the user selecting 'OK' or 'Cancel'.
     *
     * pre: not s->IsRunning();
     *
     * post: s'->IsRunning() and
     *       forall (Interator* i | i in w->GetComponents())
     *           i->IsDisplayed();
     */

    virtual void Compose();
    /*
     * Construct the physical interface components of this, including the
     * physical display window.  Note that Compose does not physically display
     * this, it only constructs its components so it may subsequently be
     * displayed with the Run and Show functions.
     *
     * post: ;   // I.e., Compose is intended to be specialized in subclasses.
     */

    virtual void Show();
    /*
     * Insert the window of this into the screen, thereby phyiscally displaying
     * it.  Upon insertion, the window will be the frontmost of all other
     * screen windows.  If this is already displayed, Show has the effect of
     * moving its window to the front of all other windows.
     *
     * To move the window to some position other than the front, extract the
     * window with this->GetWindow and maninpulate it with functions available
     * in the Window class, q.v. in ./window.h.
     *
     * The screen position of the window will be selected by the underlying
     * window manager.  To show the window of this at a specific screen
     * position, use the overloaded version of Show specified below.
     *
     * pre: ; // Note that the window may or may not be currently displayed
     *
     * post: this'->s->IsDisplayed(this->w) and
     *       this'->s->IsFrontmost(this->w);
     */

    virtual void Show(Coord x, Coord y);
    /*
     * Same specs as Show(), except the window is shown at the given x,y
     * coordinate.  The x,y coordinate system is in units of screen pixels,
     * with the 0,0 origin in the lower left corner of the screen.  The given
     * x,y coordinates refer to the lower left corder of the window.
     */

    virtual void Hide();
    /*
     * Remove the window of this from the screen, thereby phyiscally
     * undisplaying it.
     *
     * pre: ; // Note that the window may or may not be currently displayed
     *
     * post: not this'->s->IsDisplayed(this->w);
     */

    virtual Model* GetModel();
    /*
     * Return the model of this.
     *
     * post: return == m;
     */

    virtual void SetModel(Model* m);
```

```
    /*
     * Set the model of this to the given model, if the model is not already
     * set.  This SetModel function is used if this must be constructed before
     * its companion model is constructed, and therefore the model will not be
     * available to pass to the constructor.
     *
     * pre: this->m == NULL;
     *
     * post: this’->m == m;
     */

    virtual Window* GetWindow();
    /*
     * Return the window of this.
     *
     * post: return == w;
     */

    virtual bool IsShown();
    /*
     * Return true if this is currently displayed on the screen.
     *
     * post: return == shown;
     */

    virtual bool IsEditable();
    /*
     * Return true if this is currently editable.  The editability of a view
     * dictates whether the user is allowed to enter values in the view’s
     * components that are normally editable, e.g., string or text editors.
     *
     * post: return == editable;
     */

    virtual void SetEditable(bool editable);
    /*
     * Set the editability of this to the given boolean value.
     *
     * post: this’->editable == editable;
     */


 protected:
    Model* m;                   // The unique compnion model for this
    Screen* s;                  // The physical display screen for this
                                // s==NULL means this is not a top-level
                                // interactor, i.e., it’s a component of
                                // another interactor
    Window* w;                  // The phyiscal UI window for this
    bool shown;                 // True if the window is displayed
    bool editable;              // True if this is editable

    View(View&);                // Standard protected copy constructor
    View& operator=(View&);     // Standard protected op =
};

#endif
```