import java.util.*;
import java.io.Reader;

/**
 * This class supplies a bare minimum model implementation,
 * basically allowing views to be added, removed, and
 * notified of messages, errors, and other model information.
 *
 * @author Owen Astrachan
 */

public abstract class AbstractModel implements IModel
{
    private ArrayList myViews;

    public AbstractModel()
    {
        myViews = new ArrayList();
    }

    public void addView(IView view)
    {
        myViews.add(view);
    }

    public void removeView(IView view)
    {
        myViews.remove(view);
    }

    public void notifyViews(Collection elements)
    {
        for(int k=0; k < myViews.size(); k++)
        {
            IView view = (IView) myViews.get(k);
            view.notifyElements(elements);
        }
    }

    public void showViewsError(String s)
    {
        for(int k=0; k <myViews.size(); k++)
        {
            IView view = (IView) myViews.get(k);
            view.showError(s);
        }
    }

    public void messageViews(String s)
    {
        for(int k=0; k <myViews.size(); k++)
        {
            IView view = (IView) myViews.get(k);
            view.showMessage(s);
        }
    }

    public void initialize(Reader r)
    {
        initialize(new Scanner(r));
    }
}
import java.util.Collection;
import java.io.Reader;

/**
 * The interface for typical models used in Compsci100.
 * The model is typically initialized to start a process,
 * e.g., a game or a simulation. The user then interacts
 * with the model via the model’s process method which most
 * likely results in the model notifying all views that
 * a change in the model has occurred.
 *<p>
 * To set up a model/view typically the client/main code
 * looks like this:
 *<p>
 * <PRE>
 * IModel model = new ApplicationModel();
 * IView view = new GuiViewForApplication();
 * model.addView(view);
 * </PRE>
 * In this architecture the control is not part of the model,
 * but is typically part of the view (e.g., GUI control)
 * and other client code.
 *<p>
 * @author Owen Astrachan
 *<p>
 */

class IModel {
    /**
     * Initialize the model using a reader as the source
     * of the model’s initial data.
     * @param r is the Reader used to initialize.
     */
    public void initialize(Reader r);

    /**
     * Initialize the model using a scanner as the source
     * of the model’s initial data.
     * @param s is the Scanner used to initialize
     */
    public void initialize(Scanner s);

    /**
     * Add a view to the model, views are updated
     * based on application-specific basis.
     * @param view is added to this model’s views
     * @see IView
     */
    public void addView(IView view);

    /**
     * Remove a view from this model, it may be an error
     * to remove a view not previously added to the model.
     * @param view is removed from this model’s views
     */
    public void removeView(IView view);

    /**
     * Process an Object in some model-specific way. Typically
     * the user might interact with the model, making a change
     * via the object. Views would be notified if the model
     * changes, for example.
     * @param o is the Object processed by this model
     */
    public void process(Object o);

    /**
     * Notify this model’s views of the relevant model-state
     * passed to views in the collection
     * @param elements represents views to be displayed
     * @see IView
     */
    public void notifyViews(Collection elements);

    /**
     * Send a message to each of this model’s views.
     * @param s is the message sent to all views
     */
    public void messageViews(String s);

    /**
     * Send an error-message to this model’s views.
     * @param s is the error message displayed in each view
     */
    public void showViewsError(String s);
}
import java.util.Collection;

public interface IView
{
    /**
     * Display an information message in this view.
     * By convention this should not require
     * user−interaction, i.e., no modal dialog.
     * @param s is the message displayed
     */
    public void showMessage(String s);

    /**
     * Display an error message in this view. For GUI views
     * this could pop up an error dialog.
     * @param s is the error message displayed
     */
    public void showError(String s);

    /**
     * Display all elements of the Collection
     * in some way. Typically the toString method
     * of each element could be displayed, but application
     * specific views can display as warranted.
     * @param elements is the collection to be displayed
     */
    public void update(Collection elements);
}
import java.awt.event.ActionEvent;
import javax.swing.AbstractAction;
import javax.swing.JMenu;

/** *
 * @author Owen Astrachan
 * @version Sep 12, 2004
 */
public class JottoViewer extends SimpleViewer {

/** *
 * @param title
 * @param prompt
 */
public JottoViewer(String title, String prompt) {
    super(title, prompt);
}

protected JMenu makeFileMenu() {
    JMenu fileMenu = super.makeFileMenu();
    fileMenu.add(new AbstractAction("New game") {
        public void actionPerformed(ActionEvent ev) {
            JottoModel jm = (JottoModel) myModel;
            jm.newGame();
        }
    });
    return fileMenu;
}
}
import java.awt.event.ActionEvent;
import javax.swing.AbstractAction;
import javax.swing.JMenu;

/**
 * @author Owen Astrachan
 * @version Sep 12, 2004
 */
public class JottoViewer extends SimpleViewer {

/**
 * @param title
 * @param prompt
 */
public JottoViewer(String title, String prompt) {
    super(title, prompt);
}

protected JMenu makeFileMenu() {
    JMenu fileMenu = super.makeFileMenu();
    fileMenu.add(new AbstractAction("New game"){
        public void actionPerformed(ActionEvent ev){
            JottoModel jm = (JottoModel) myModel;
            jm.newGame();
        }
    });
    return fileMenu;
}
}