```java
package rolodex;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/****
 *
 * Class OKAddButtonListener defines the action listener that is is attached to
 * the OK button in the add card dialog.  The design of this class is an
 * example of referential transparency in design.  In the context of an
 * object-oriented design, referential transparency basically means that all
 * data referenced in a class are stored locally in that class, rather than
 * being inherited from a parent class or obtained from a lexically enclosing
 * outer class.  In the specific case of OKAddButtonListener, referential
 * transparency means that the class has direct reference to Model and View
 * objects that provide the methods called when the user press the "OK" button.
 *                                                                          <p>
 * On the plus side, this form of referential transparency increases the
 * cohesiveness of a class, in that it is very easy to find the definition of
 * all data referred to in any class method.  Referenced data are either method
 * parameters, local method variables, or class variables.  There is no need to
 * search up an inheritance hierarchy or a class nesting hierarchy to find a
 * variable definition.
 *                                                                          <p>
 * On the minus side, referential transparency may increase communication
 * coupling, since data not constructed locally must be sent in to the class
 * constructor.  For example, in the case of OKAddButtonListener, references to
 * the Rolodex, RolodexMenuUI, and AddCardDialog must be sent to the
 * constructor in order for the local data members to be properly initialized.
 *
 */

public class OKAddButtonListener implements ActionListener {

    /**
     * Construct this with the given rolodex model and parent dialog view and
     * grandparent rolodex view.  Access to the model is for calling its add
     * function.  Access to the parent dialog is for gathering data to be sent
     * to Rolodex.add.  Access to the grandparent view is for reporting input
     * error messages.  See the specs of actionPerformed below for further
     * discussion.
     */
    public OKAddButtonListener(Rolodex r, AddCardDialog acd,
            RolodexMenuUI rmui) {
        this.r = r;
        this.acd = acd;
        this.rmui = rmui;
    }

    /**
     * Respond to a press of the OK button by calling Rolodex.add.  The card
     * data sent as the parameter to add are gathered from the JTextFields for
     * the card fields.  These text fields are contained in the parent
     * AddCardDialog.
     */
    public void actionPerformed(ActionEvent e) {

        try {
            r.add(new Card(
                new Name(acd.getName()),        // Card name as a String
                new Id(acd.getId()),            // Card id as an Integer
                new Age(acd.getAge()),          // Card age as an Integer
                new Sex(acd.getSex()),          // Sex as an enum literal
                new Address(acd.getAddr())      // Address as a String
            ));
        }
        catch (AddInputErrors aie) {
            /*
             * Display zero or more error messages, as appropriate.
             */
            rmui.showErrorDialog(aie.getErrorList());
        }

        /*
         * Notify the Rolodex observers if it has changed.  Per the specs of
         * the Java Observable class, notifyObservers only performs the
         * notification if the observed object has actually changed.
         */
        r.notifyObservers();

    }

    /** The companion model */
    protected Rolodex r;

    /** The parent view */
    protected AddCardDialog acd;

    /** The grandparent view */
    protected RolodexMenuUI rmui;

}
```