```java
package rolodex;

import java.util.*;
import java.io.*;
import javax.swing.*;
import mvp.*;

/****
 *
 * Class File is the model class for the rolodex file handling.  It contains
 * functions for all of the operations defined on the Rolodex File menu.
 *
 */

public class File extends Model implements Observer {

    /**
     * Construct this with the given companion view and the parent RolodexTool
     * model.  The RolodexTool is provided for access to its newRolodex
     * operation as well as access to the Rolodex model.  Access to the sibling
     * model is necessary to observe it for changes.  This is done to keep
     * track of when the Rolodex changes so the offer to save can be made to
     * the user when appropriate.
     */

    public File(View v, RolodexTool rt) {

        /*
         * Call the parent constructor.
         */
        super(v);

        /*
         * Set the parent and sibling models.
         */
        this.rt = rt;
        r = rt.getRolodex();

        /*
         * Set the rolodex change status initially to false.
         */
        rolodexHasChanged = false;

        /*
         * Make this an observer of the sibling Rolodex model object.
         */
        r.addObserver(this);

    }

    /**
     * Create a new empty rolodex.  If the current Rolodex object has changed
     * since the last save, then offer to save it before creating the new empty
     * one.
     */
    public void newFile() {

        boolean saveStatus = true;      // Return value from OfferToSave

        if (rolodexHasChanged) {
            saveStatus = offerToSave();
        }

        /*
         * If the file was dealt with properly (or didn't require dealing
         * with), proceed to create new rolodex.  Otherwise just leave without
         * doing anything.
         */

        if (saveStatus) {
            rt.newRolodex();
            rolodexHasChanged = false;
        }

    }

    /**
     * Initiate the opening of a new rolodex.  If the current Rolodex object
     * has changed since the last save, then offer to save it before opening.
     * Then get the name of the file from the user and proceed with the actual
     * opening.
     */
    public void open() {

        boolean saveStatus;             // Return value from OfferToSave

        if (rolodexHasChanged) {
            saveStatus = offerToSave();

            if (saveStatus) {
                System.out.println("Djwanna save?");
            }
        }

    }

    /**
     * Open an existing rolodex file of the given file name.  Create a rolodex
     * containing the data of the opened file.  If the current Rolodex object
     * has changed since the last save, then offer to save it before opening
     * the new one.  If the given file name is not a legal rolodex or not
     * openable, throw an exception.
     */
    public void open(Name n) {
        System.out.println("In File.open");
    }

    /**
     * Save the contents of the current canvas in the filespace.
     */
    public void save() {
        System.out.println("In File.save");
```

```java
    }

    /**
     * Save this in the filespace under the given name.
     */
    public void saveAs(Name n) {
        System.out.println("In File.saveAs");
    }

    /**
     * Print this in textual format.
     */
    public void print() {
        System.out.println("In File.print");
    }

    /**
     * Quit the rolodex system.
     */
    public void exit() {
        System.out.println("In File.exit -- bye.");
        System.exit(0);
    }

    /**
     * Respond to a change in the Rolodex model object by setting the local
     * flag that indicates the rolodex has changed since the last save.
     */
    public void update(Observable o, Object arg) {
        System.out.println("In File.update");
        rolodexHasChanged = true;
    }

    /**
     * Offer to save the current rolodex.  Return what comes back from the
     * offer-to-save dialog in the view, which is true if the user selects
     * 'Save' or 'Discard' or false if the user selects 'Cancel'.
     */
    protected boolean offerToSave() {

        // Quick and dirty view stub.
        JOptionPane jop = new JOptionPane();
        Object[] options = { "Save", "Discard", "Cancel" };
        return jop.showOptionDialog(null, "Do you want to save?",
            "Offer to Save", JOptionPane.PLAIN_MESSAGE,
                JOptionPane.YES_NO_CANCEL_OPTION,
                    null, options, options[0]) > 0;
    }

    /** Data representation of this is simply a Java File.  Since only one
        Rolodex can be open at a time, only a single file is necessary. */
    java.io.File file;

    /** Flag indicating that the Rolodex object has changed since the last
        save. */
    boolean rolodexHasChanged;

    /** Parent model. */
    RolodexTool rt;

    /** Sibling model. */
    Rolodex r;

}
```