CSC 309 Lecture Notes Week 2

General Design Principles

High-Level Design Patterns

Examples of Design Derivation

I. Milestone and other Java examples:

A. www:.../309/exmaples/milestone2

B. www:.../309/exmaples/misc-java

II. What is design?

A. *Abstraction* of implementation.

1. Abstraction means *things get left out*.

2. Simply put,
   design leaves out *method code bodies*.

3. This is an over simplification.

4. There are several levels of design.

What is design, cont'd

B. Levels of design abstraction.

1. **Packaging Design**

   a. Largest modular units.

   b. Pkg names, descriptions, communication.

   c. Separate applications and servers.

What is design, cont’d

2. **Abstract Class Design**

   a. Classes added to packages.

   b. Class names, descriptions; no contents.

What is design, cont’d

3. **Mid-Level Class Design**
   1. Add methods and data fields to classes.
   2. Method and field names, descriptions; no method signatures or concrete data reps.
   3. Define class inheritance.

What is design, cont’d

4. **Detailed Class Design**
   a. Add full input/output signatures.
   b. Select concrete data representations.

What is design, cont’d

5. **Functional Design**

    a. Add pre- and postconditions to methods.

    b. Define control flow among methods.

What is design, cont’d

C. At any level, apply suitable patterns.

D. We’ll start with two patterns:

1. "Model/View/Process".

2. "Information Processing Tool".

III. What is a design pattern?

A. From architect Christopher Alexander:

> "Each pattern describes a problem which occurs over and over again ... "

B. The same applies to software.

C. For software, notation is design diagrams, code templates, step-by-step descriptions.

IV. The MVP pattern

A. Separate core processing from GUI and underlying support.

1. ***Model*** is directly traceable to abstract spec.

2. ***View*** is concrete GUI.

3. ***Process*** is underlying support.

MVP, cont'd

B. Correspondence between models and companion views.

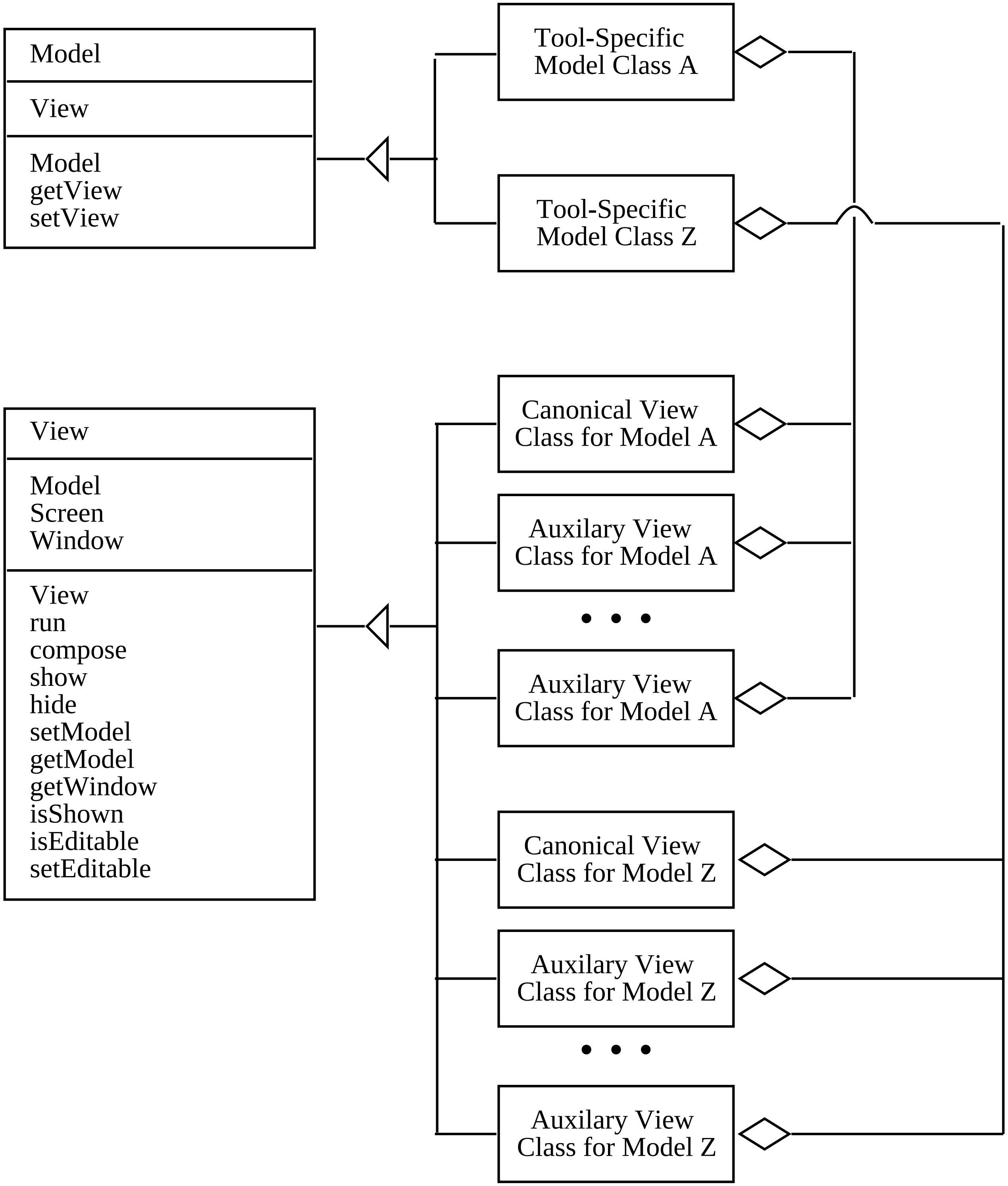
1. There is typically a *canonical* view.

2. May also be additional views.

3. Both model and view classes are directly traceable to requirements spec.

MVP, cont'd

C. General diagram of MVP …


Model
View
Model
getView
setView
Tool-Specific
Model Class A
Tool-Specific
Model Class Z
View
Model
Screen
Window
View
run
compose
show
hide
setModel
getModel
getWindow
isShown
isEditable
setEditable
Canonical View
Class for Model A
Auxilary View
Class for Model A
• • •
Auxilary View
Class for Model A
Canonical View
Class for Model Z
Auxilary View
Class for Model Z
• • •
Auxilary View
Class for Model Z

MVP, cont'd

1. Figure shows data members and methods for abstract Model and View classes.

2. Defined in 309 Java class library.

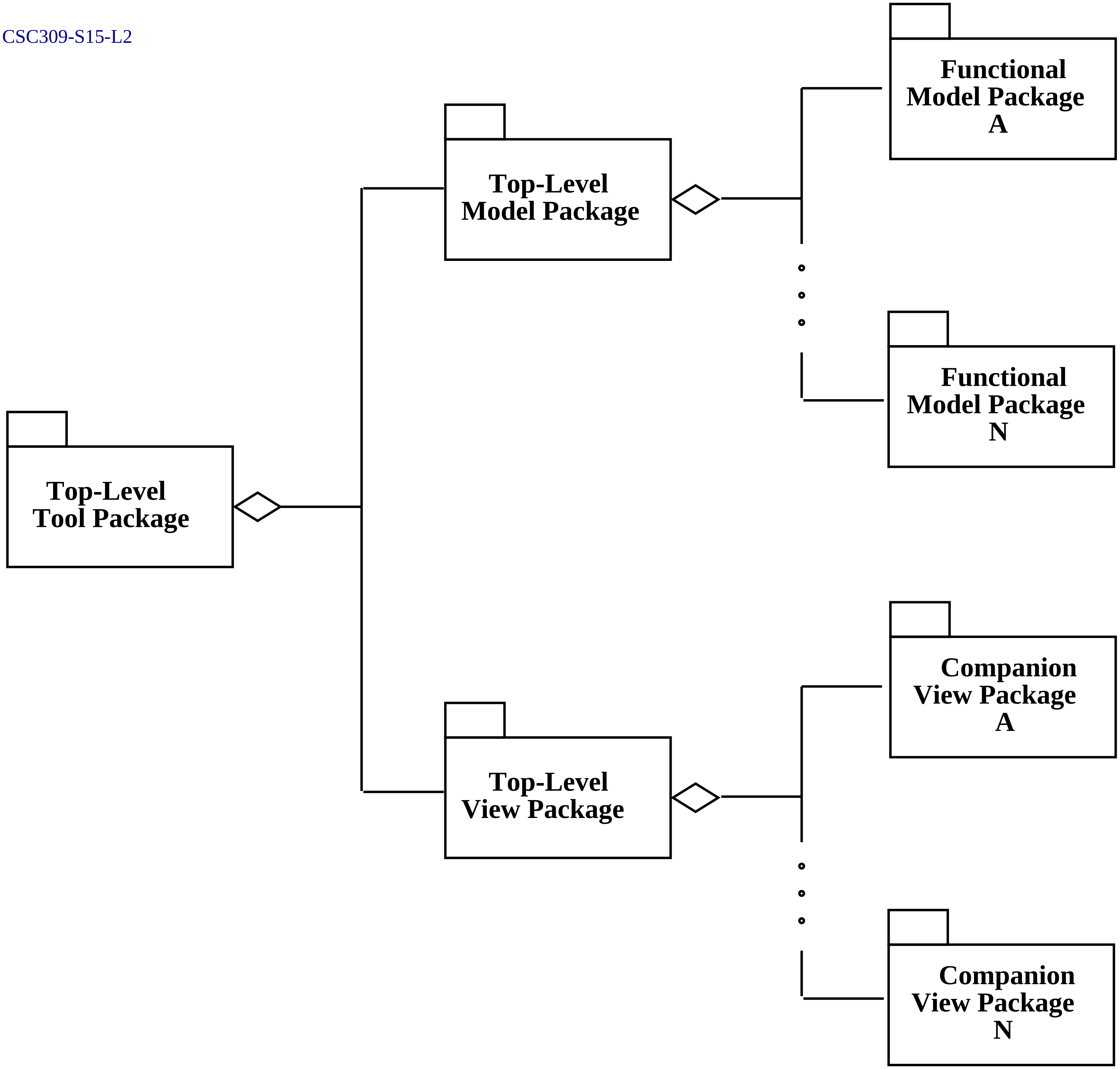
V. "Info Processing Tool" pattern

A. Used to layout high-level packaging.

B. In conjunction with MVP.

C. Applies to 309 applications specifically.

D. Major functional groupings consist of a pair of model/view packages.



Top-Level Tool Package

Top-Level Model Package

Functional Model Package A

Functional Model Package N

Top-Level View Package

Companion View Package A

Companion View Package N

E. IPT pattern info sources:

1. *The organization of the top-level GUI*
2. *The organization of the end-user requirements scenarios*
3. *Modular organization of the abstract Java/UML model*

VI. Calendar Tool example from 308.

A. Similar in size, scope to 308/309 projects.

B. We’ll continue this quarter with design and implementation.

VII. **Applying Info-Processing-Tool pattern**

A. Eight modules in Cal Tool spec:

1. File -- file processing

2. Edit -- general editing

3. Schedule -- item scheduling

4. View -- viewing calendars

Applying patterns, cont’d

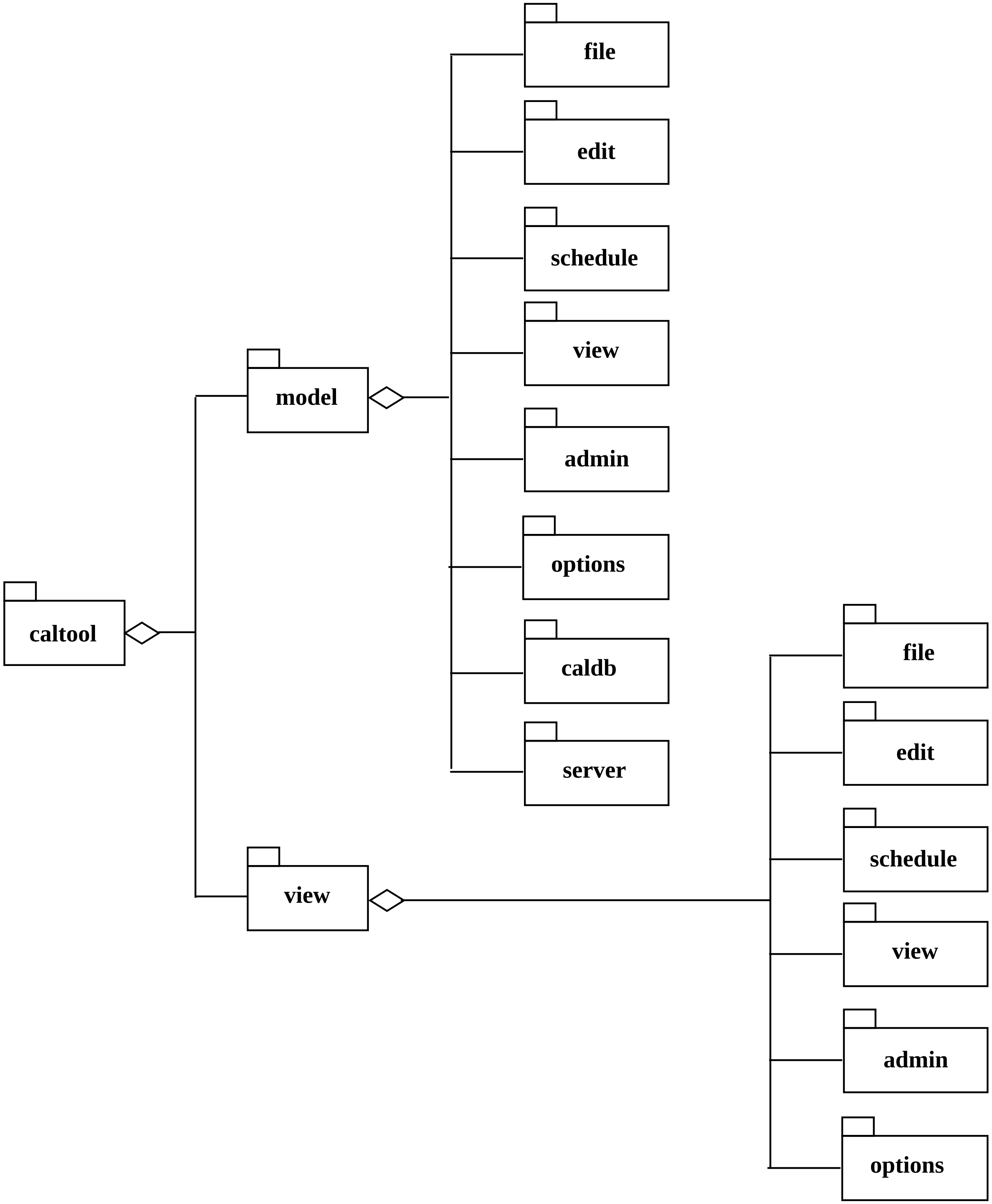
5. Admin -- managing databases
6. Options -- managing options
7. CalDB -- underlying calendar database
8. Server -- host server and communication

Applying patterns, cont’d

B. From derivation pattern, we get eight model packages.

C. Applying IPT, we add a top-level tool package, and companion view packages.

D. Diagram:


file
edit
schedule
view
model
admin
options
caltool
caldb
server
view
file
edit
schedule
view
admin
options

E. Note no companion UIs for CalDB and Server pkgs.

F. Derived top-level tool class in

```
implementation/source/
    java/caltool/model/
        CalendarTool.java:
```

CalendarTool.java

```
package caltool.model;

import caltool.view.*;
import caltool.model.file.*;
import caltool.model.edit.*;
import caltool.model.schedule.*;
import caltool.model.view.*;
import caltool.model.admin.*;
import caltool.model.options.*;
import caltool.model.help.*;
import caltool.model.caldb.*;
import mvp.Model;
```

CalTool.java, cont’d

```
/****
 *
 * Class CalendarTool is ...
 *
 * @author Gene Fisher (gfisher@...
 * @version 13apr15
 *
 */
```

CalTool.java, cont'd

```
public class CalendarTool extends Model {

       . . .

  /** File-handling model class */
  protected File file;

       . . .

  /** Calendar database model class */
  protected CalendarDB caldb;

}
```

VIII. Excerpts from Cal Tool spec and derived design.

A. Packaging & class design for Calendar Tool.

B. Refined from 308 model and prototype.

C. Major refinement is packages, per IPT and Model/View patterns.

D. Between mid-level & detailed class design.

IX. Selected Calendar Tool java files.

A. caltool/model/caldb/package.html

```
<html>
<body>
Major model database classes ...
</html>
</body>
```

B. **CalendarDB.java**

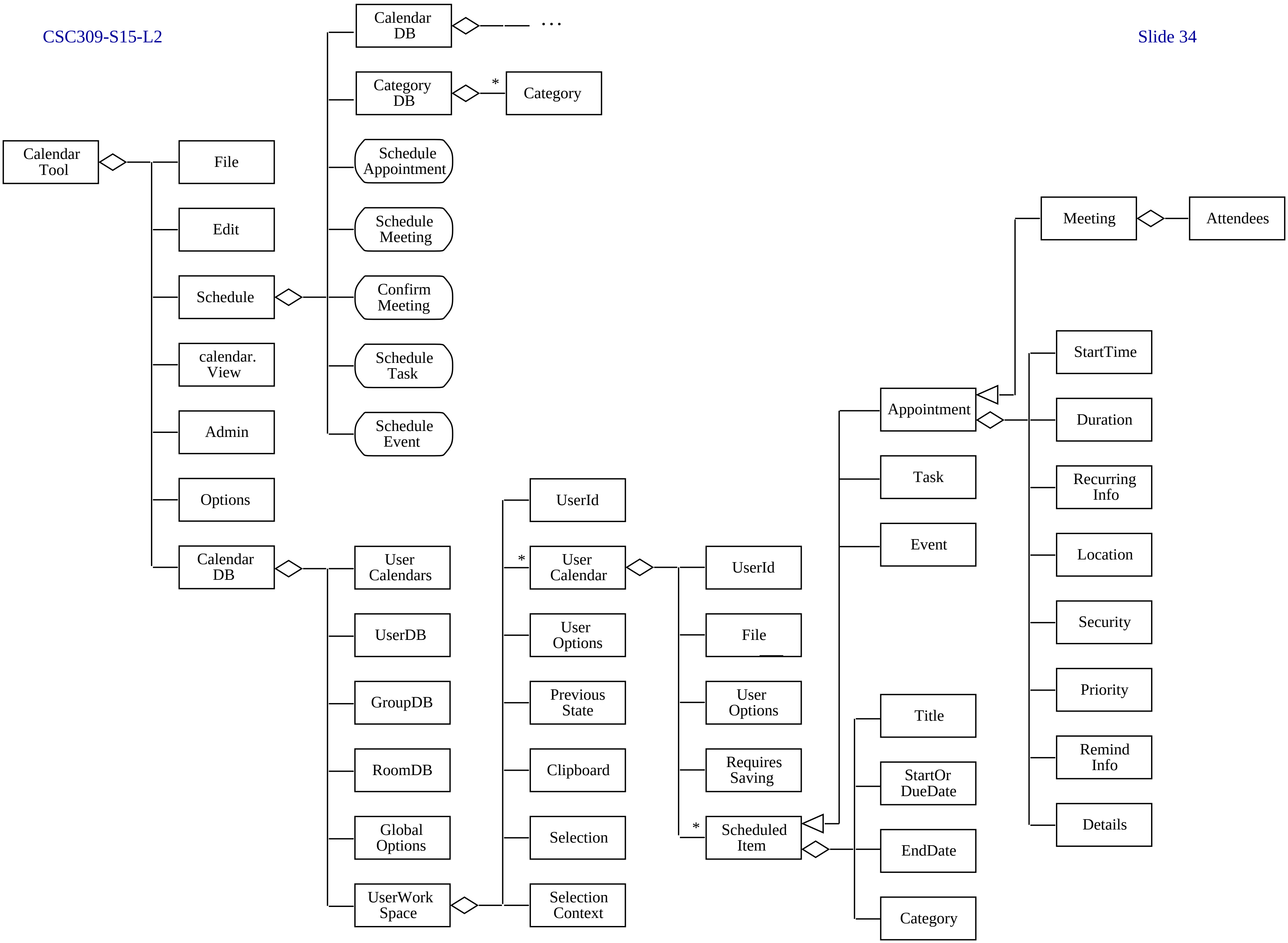
```
package caltool.caldb;

import caltool.schedule.*;
import caltool.admin.*;
import caltool.options.*;
import mvp.*;
import java.util.Collection;
```

```
/****
 *
 * The CalendarDB is the top-level data
 * repository for the Calendar Tool. ...
```

See code file links at end of online notes.

We’ll look at some during lecture.

C. Class diagram for Milestone 2ish design:



Calendar Tool

File

Edit

Schedule

calendar. View

Admin

Options

Calendar DB

Calendar DB

. . .

Category DB

*

Category

Schedule Appointment

Schedule Meeting

Confirm Meeting

Schedule Task

Schedule Event

User Calendars

UserDB

GroupDB

RoomDB

Global Options

UserWork Space

UserId

*

User Calendar

User Options

Previous State

Clipboard

Selection

Selection Context

UserId

File

User Options

Requires Saving

*

Scheduled Item

Appointment

Task

Event

Title

StartOr DueDate

EndDate

Category

Meeting

Attendees

StartTime

Duration

Recurring Info

Location

Security

Priority

Remind Info

Details

X. Observations on derivations in Notes 2

A. `CalendarDB.java`

1. Traceability to spec is quite direct.

2. This is a managerial class.

3. Class comment derived from spec descrip.

4. Derived data fields trace to spec object.

B. `Schedule.java`

1. Another managerial class.

2. Doesn’t appear in abstract model.

3. Used to help model/view communication.

C. `ScheduledItem.java`

1. Traces directly to abstract object.

2. Object components are class data fields.

D. **`Appointment.java`**,
`Meeting.java`,
`Task.java`,
`Event.java`

1. Also trace to spec objects.

2. Inheritance relationships retained.

E. `Date.java`

1. Straightforward translation of spec object.

2. Likely replaced with Java lib class.

XI. What if abstract spec is incomplete or inconsistent?

A. See 308 week 4-5 notes.

B. When spec is incomplete or inconsistent, derive design directly from requirements.

XII. Intro to Java library classes.

A. Get into the "library habit".

B. Key packages in JFC lib:

1. *java.lang*
2. *java.util*
3. *java.io*
4. *javax.swing*
5. *java.awt*

Library classes, cont’d

C. Packages have data structure and GUI classes you’ll use in 309.

1. Summarized UML in Week 4 in notes.

2. Concrete examples in milestone and misc-java directories.

3. This week focus on intro to GUI classes.

XIII. Package `javax.swing`

A. Selected Classes:

- `Box`
- `JButton`
- `JComboBox`
- `JLabel`
- `JList`
- `JMenu`
- `JMenuBar`
- `JMenuItem`
- `JTabbedPane`
- `JTextArea`
- `JTextField`

`javax.swing`, cont'd

B. Selected subpackages:

- `swing.colorchooser`
- `swing.filechooser`
- `swing.table`
- `swing.text.html.parser`
- `swing.tree`
- `swing.undo`

XIV. Package `java.awt`

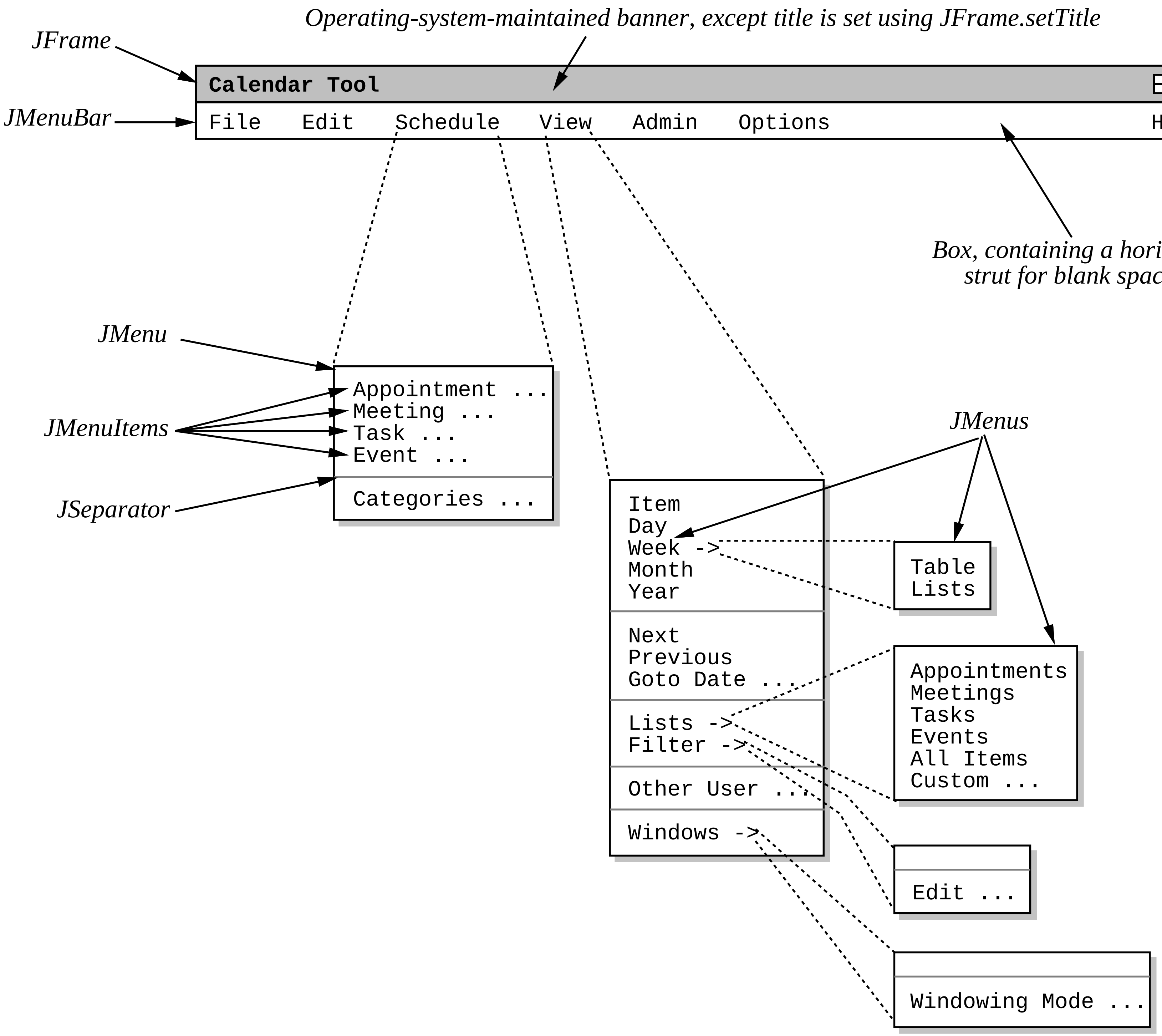
Lower-level support for `swing`.

- `Color`
- `Component`
- `Event`
- `Graphics2D`
- `GridLayout`
- `Image`

XV. Designing GUIs with Swing.

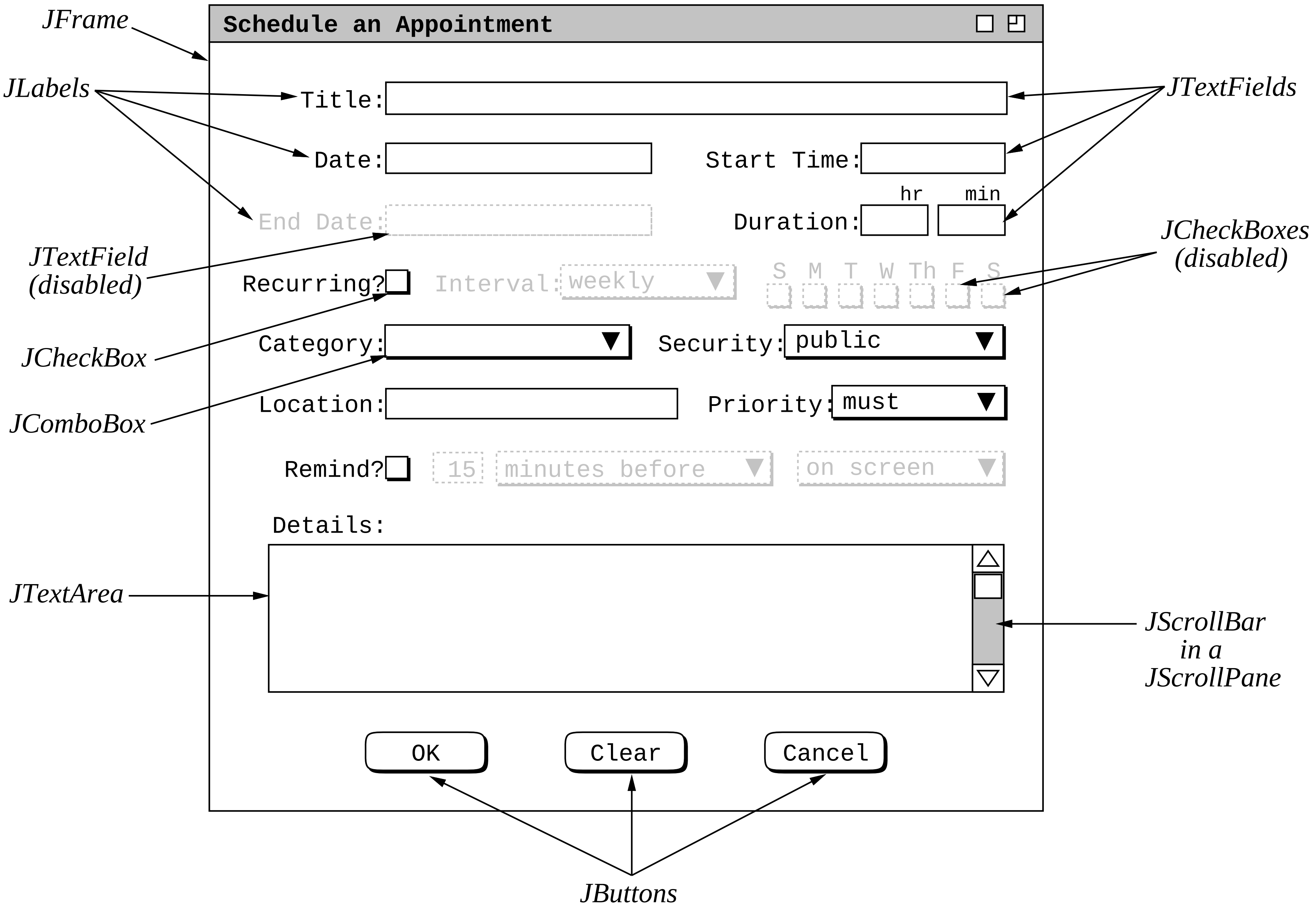
A. Above list used widely in 309.

B. Figure 2 shows a typical menubar.


Operating-system-maintained banner, except title is set using JFrame.setTitle
JFrame
JMenuBar
Calendar Tool
File
Edit
Schedule
View
Admin
Options
Help
Box, containing a horizontal strut for blank spacing
JMenu
JMenuItems
JSeparator
Appointment ...
Meeting ...
Task ...
Event ...
Categories ...
JMenus
Item
Day
Week ->
Month
Year
Next
Previous
Goto Date ...
Lists ->
Filter ->
Other User ...
Windows ->
Table
Lists
Appointments
Meetings
Tasks
Events
All Items
Custom ...
Edit ...
Windowing Mode ...

Swing, cont’d

C. Figure 3 shows typical dialog.



Schedule an Appointment

Title:

Date:

Start Time:

End Date:

Duration: hr min

Recurring? □ Interval: weekly ▼ S M T W Th F S □ □ □ □ □ □ □

Category: ▼

Security: public ▼

Location:

Priority: must ▼

Remind? □ 15 minutes before ▼ on screen ▼

Details:

JFrame

JLabels

JTextFields

JCheckBoxes (disabled)

JTextField (disabled)

JCheckBox

JComboBox

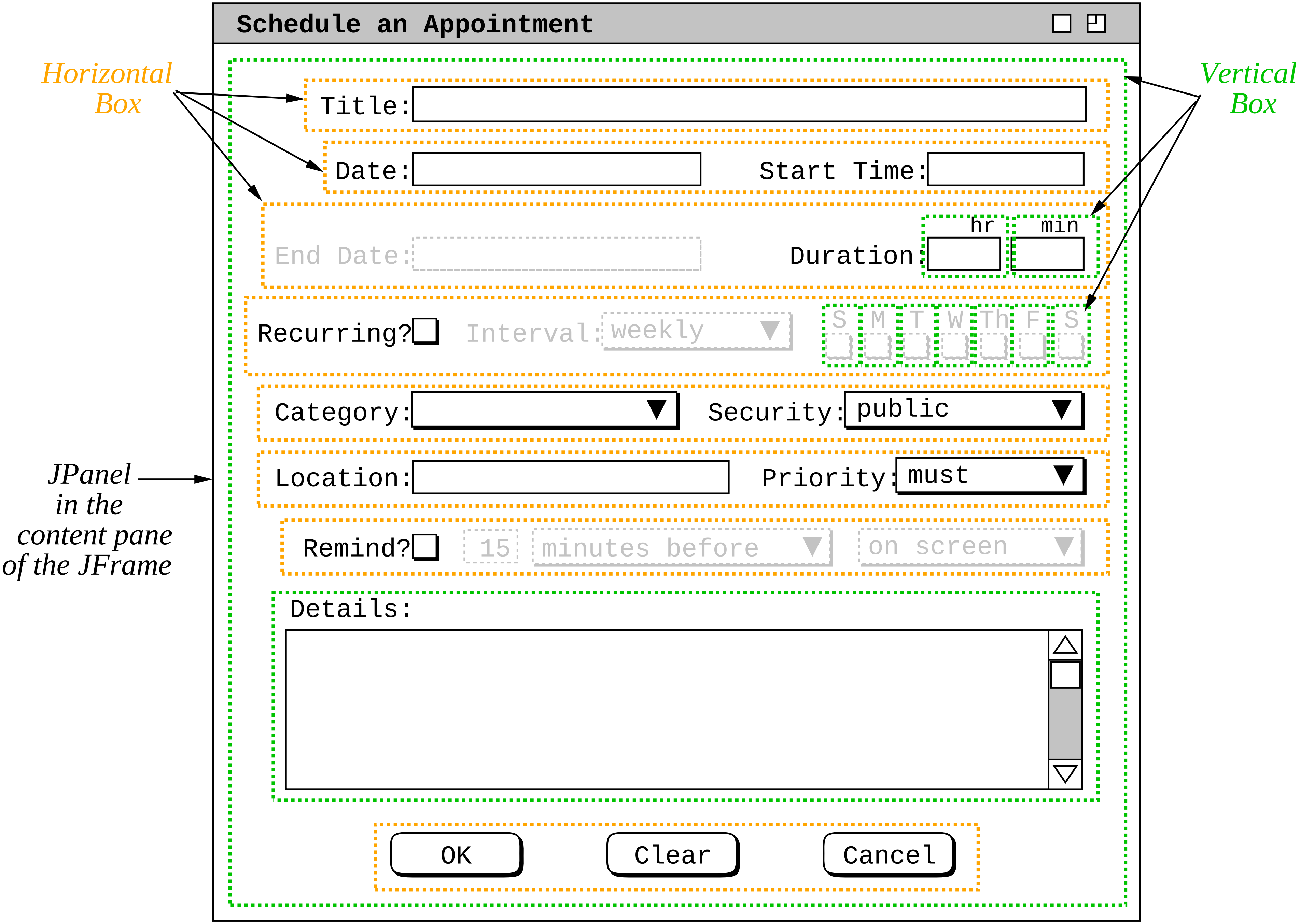
JTextArea

JScrollBar in a JScrollPane

JButtons

Swing, cont’d

D. Figure 4 dialog layout with Boxes



Schedule an Appointment

Title:

Date: Start Time:

End Date: Duration: hr min

Recurring? □ Interval: weekly ▼ S M T W Th F S

Category: ▼ Security: public ▼

Location: Priority: must ▼

Remind? □ 15 minutes before ▼ on screen ▼

Details:

Horizontal Box

Vertical Box

JPanel in the content pane of the JFrame

XVI. View class naming conventions.

A. Standard name suffixes.

B. For classes that inherit from `mvp.View`.

C. Suffixes indicate general usage

View class naming, cont'd

Suffix	Example
`UI`	`ScheduleUI`
`Dialog`	`ScheduleAppointmentDialog`
`Editor`	`CategoriesEditor`
`Display`	`MonthlyAgendaDisplay`
`ButtonListener`	`OKScheduleAppointmentButtonListener`
`Panel`	`SchedulingOptionsPanel`

XVII. **Coordination of Model, View classes.**

A. Parallel decomposition.

1. `Model` and `View` classes at top of inheritance hierarchy.

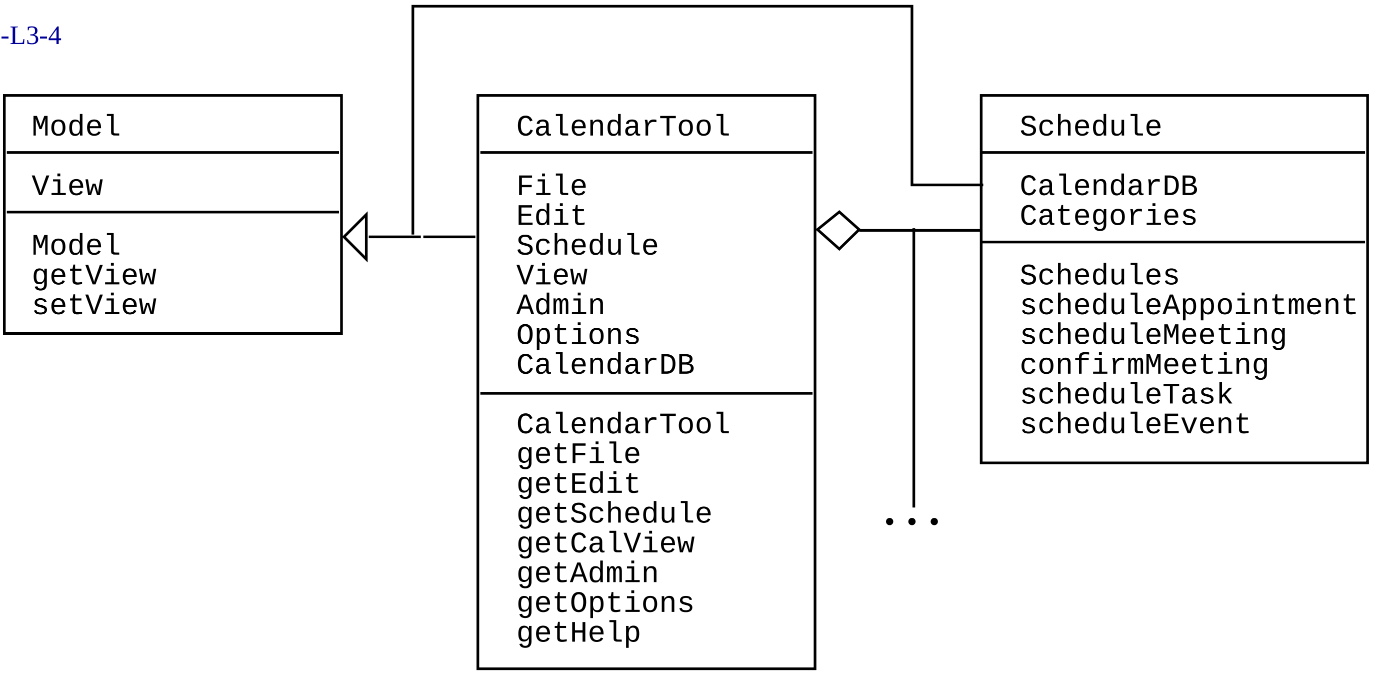
2. Tool-specific model and view classes inherit from these.

M/V Coordination, cont’d

B. High-level class decomposition follows *functional hierarchy*.

C. Most important that functional hierarchy *makes sense.*

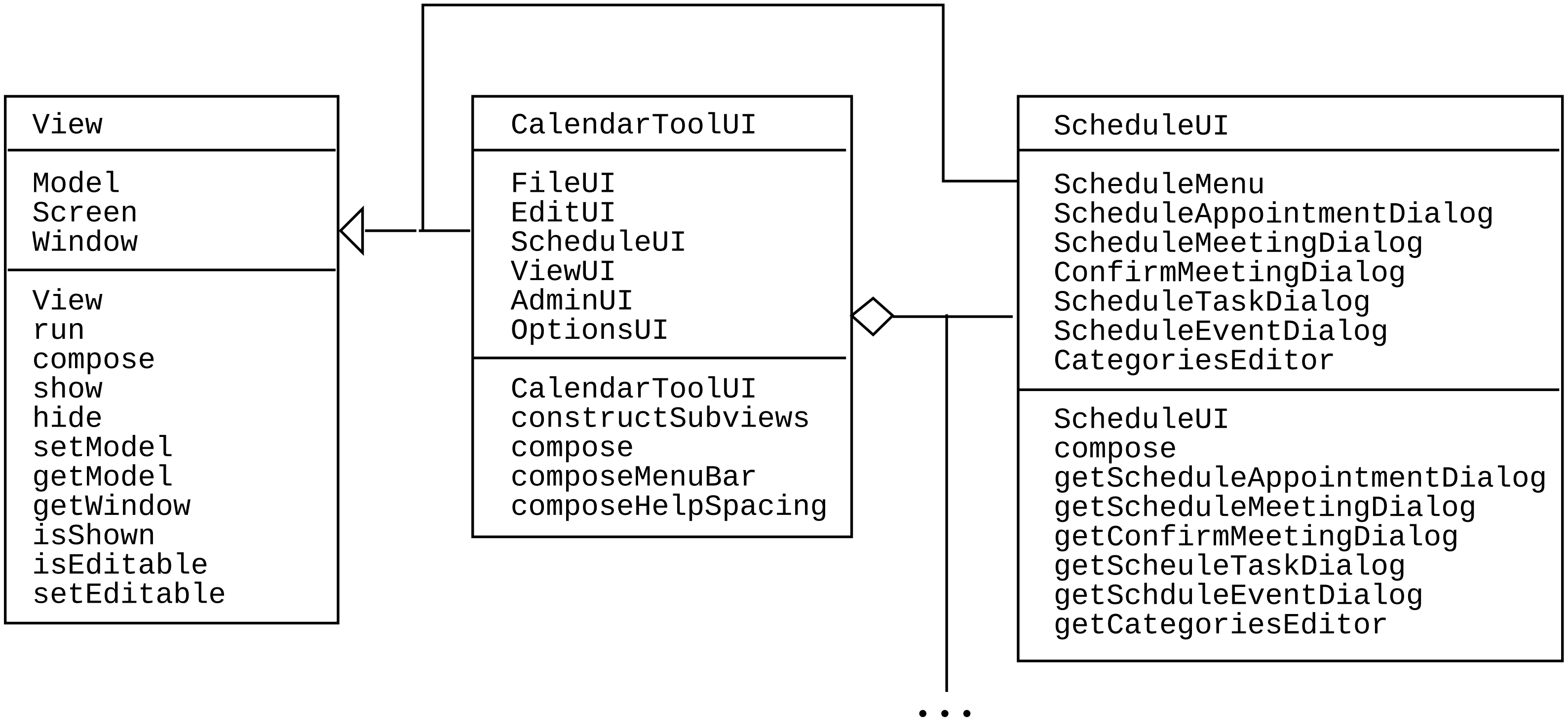
XVIII. Example of high-level Model/View class diagram.



Model
View
Model
getView
setView

CalendarTool
File
Edit
Schedule
View
Admin
Options
CalendarDB
CalendarTool
getFile
getEdit
getSchedule
getCalView
getAdmin
getOptions
getHelp

Schedule
CalendarDB
Categories
Schedules
scheduleAppointment
scheduleMeeting
confirmMeeting
scheduleTask
scheduleEvent



View
Model
Screen
Window
View
run
compose
show
hide
setModel
getModel
getWindow
isShown
isEditable
setEditable

CalendarToolUI
FileUI
EditUI
ScheduleUI
ViewUI
AdminUI
OptionsUI
CalendarToolUI
constructSubviews
compose
composeMenuBar
composeHelpSpacing

ScheduleUI
ScheduleMenu
ScheduleAppointmentDialog
ScheduleMeetingDialog
ConfirmMeetingDialog
ScheduleTaskDialog
ScheduleEventDialog
CategoriesEditor
ScheduleUI
compose
getScheduleAppointmentDialog
getScheduleMeetingDialog
getConfirmMeetingDialog
getScheuleTaskDialog
getSchduleEventDialog
getCategoriesEditor

M/V class diagram, cont’d

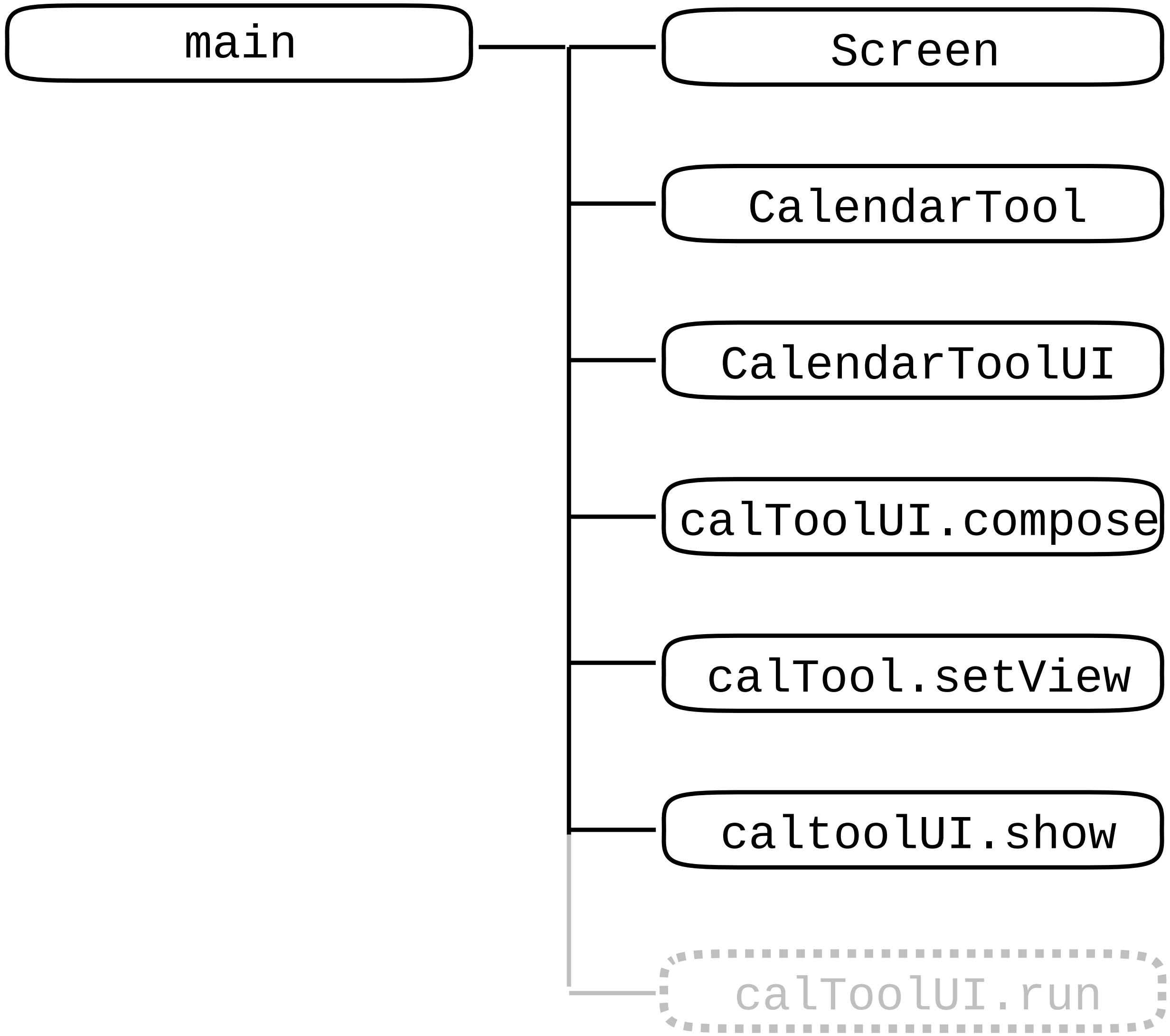
A. `Model` and `View` are the root of the inheritance hierarchy.

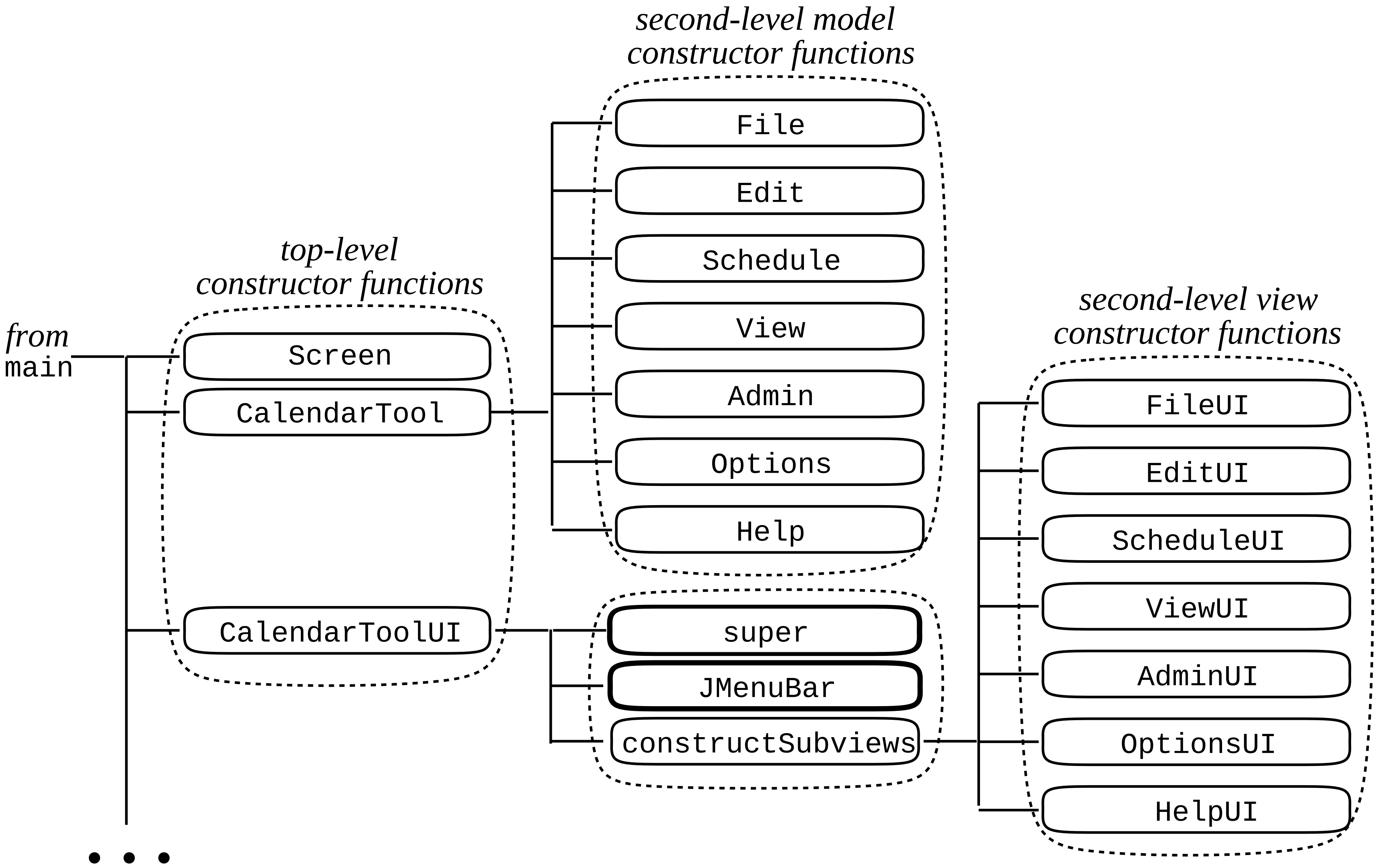
B. Inheriting from these are top-level model and view classes.

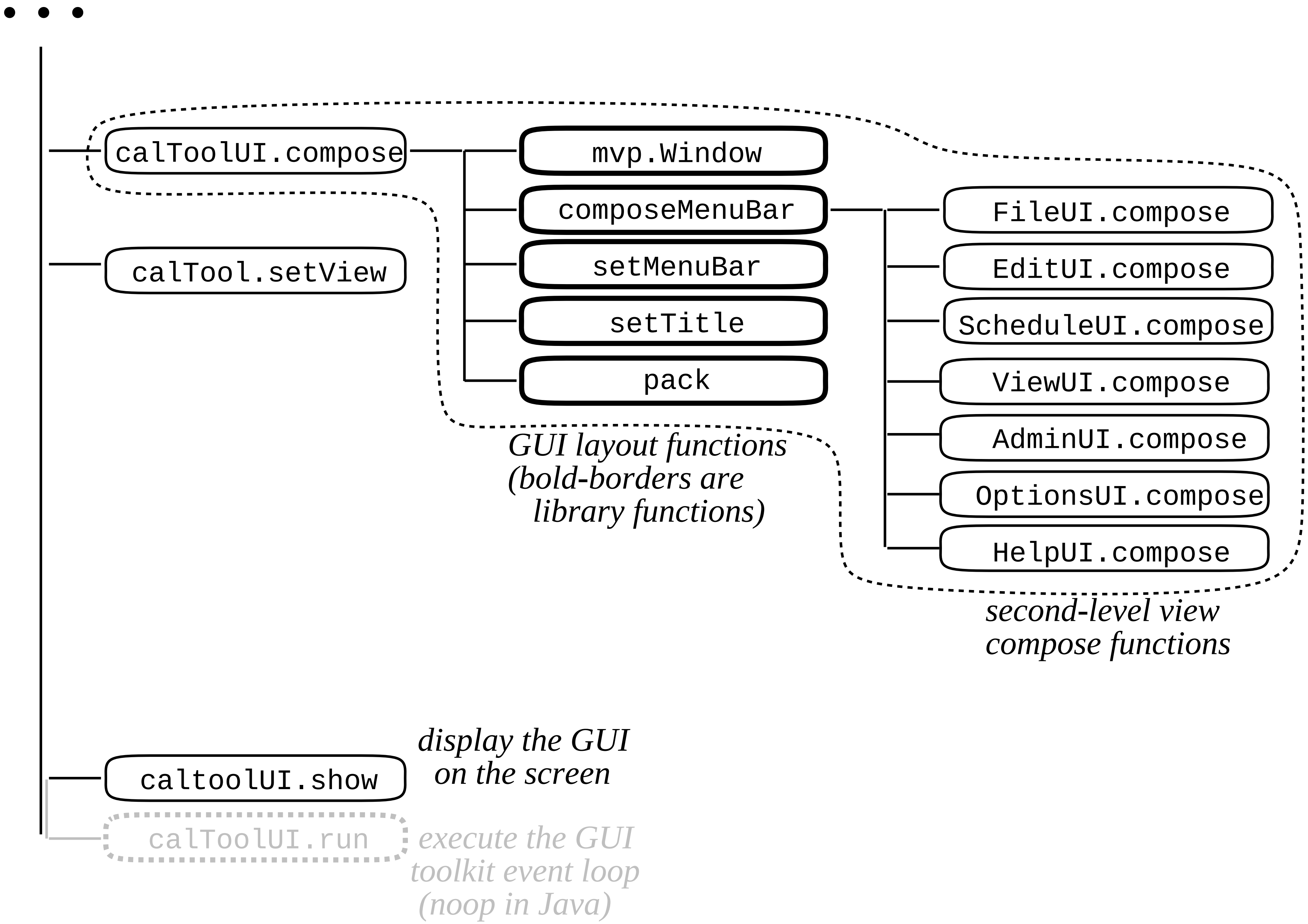
C. Below top-level are *submodels* and *subviews*.

D. These also inherit from `Model` and `View`.

XIX. High-level M/V function diagram.




second-level model
constructor functions
top-level
constructor functions
second-level view
constructor functions
from
main
Screen
CalendarTool
CalendarToolUI
File
Edit
Schedule
View
Admin
Options
Help
super
JMenuBar
constructSubviews
FileUI
EditUI
ScheduleUI
ViewUI
AdminUI
OptionsUI
HelpUI
• • •



• • •

calToolUI.compose

mvp.Window

composeMenuBar

setMenuBar

setTitle

pack

FileUI.compose

EditUI.compose

ScheduleUI.compose

ViewUI.compose

AdminUI.compose

OptionsUI.compose

HelpUI.compose

GUI layout functions (bold-borders are library functions)

second-level view compose functions

calTool.setView

caltoolUI.show

display the GUI on the screen

calToolUI.run

execute the GUI toolkit event loop (noop in Java)

High-level function diagram, cont’d

A. First three calls are top-level constructors.

1. `Screen` initializes GUI.
2. `CalendarTool` constructs models.
3. `CalendarMenuUI` constructs views.

Function diagram, cont’d

B. `CalendarToolUI.compose` performs UI layout.

1. Subfunctions layout various UI pieces.

2. Functions with bold borders are in Java and 309 libraries.

Function diagram, cont’d

C. `CalendarTool.setView` sets model to point to view.

1. Model and view mutually refer.
2. Model constructed first, View constructor passed a Model reference.
3. Then Model.setView is called.
4. Enables two-way communication.

Function diagram, cont'd

D. `View.show` inserts the view's main window into UI screen.

E. Depending on GUI toolkit, call to `View.run` may be necessary.

1. In Java, `run` is no-op.
2. In other toolkits, `run` starts GUI event handling loop.

High-level function diagram, cont’d

F. Once event loop is started, all program control assumed by toolkit.

1. In Java, event loop is separate thread.

2. Event loop calls application methods that *listen* for events.

XX. Overview of Event-Based Design

A. Event loop takes over at end of `Main`.

1. A separate thread -- `java.awt.EventDispatchThread`

2. `MainThread` terminated.

3. Application methods invoked via *events*.

Overview, cont'd

B. Event-based processing in all GUI toolkits.

1. Details vary widely.

2. Each has an *event model*.

3. What's the same is main program looses control, methods invoked through events.

XXI. Designing event-based programs

A. Two important aspects:

1. Setting up event handlers.

2. Handling the events.

Designing event-based, cont'd

B. Event handlers respond to events.

1. Events are user actions.

2. In Java, we set up an *`EventListener`*.

3. Typical case is an *`ActionListener`* for a menu item or button.

Designing event-based, cont’d

C. Event handler invokes an application method.

1. Event-invoked methods are "*call-backs*".

2. In Java, call-backs invoked by *`actionPerformed`* method of *`EventListener`*.

Designing event-based, cont'd

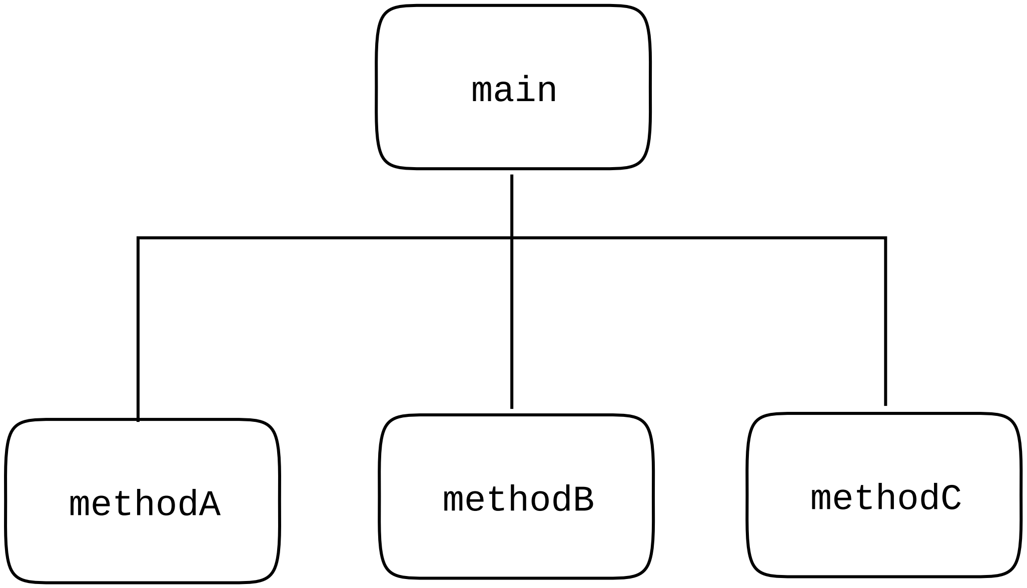
3. *`actionPerformed`* is specialized for each listener.

4. Each specialized `actionPerformed` calls an appropriate appl'n method.

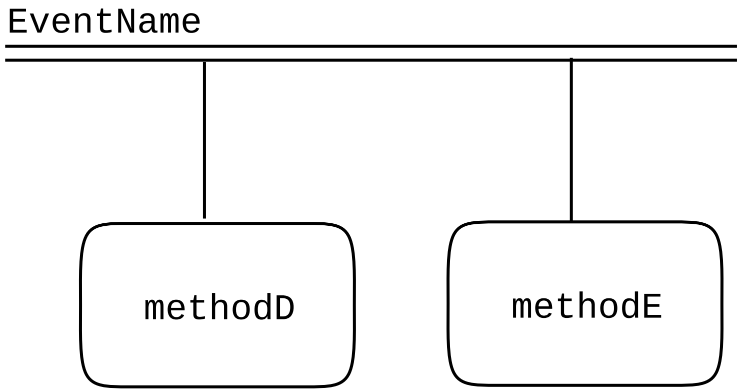
XXII. Design diagram notation

A. In function diagram, event-based invocation is shown with a double line.

B. See Figure 7



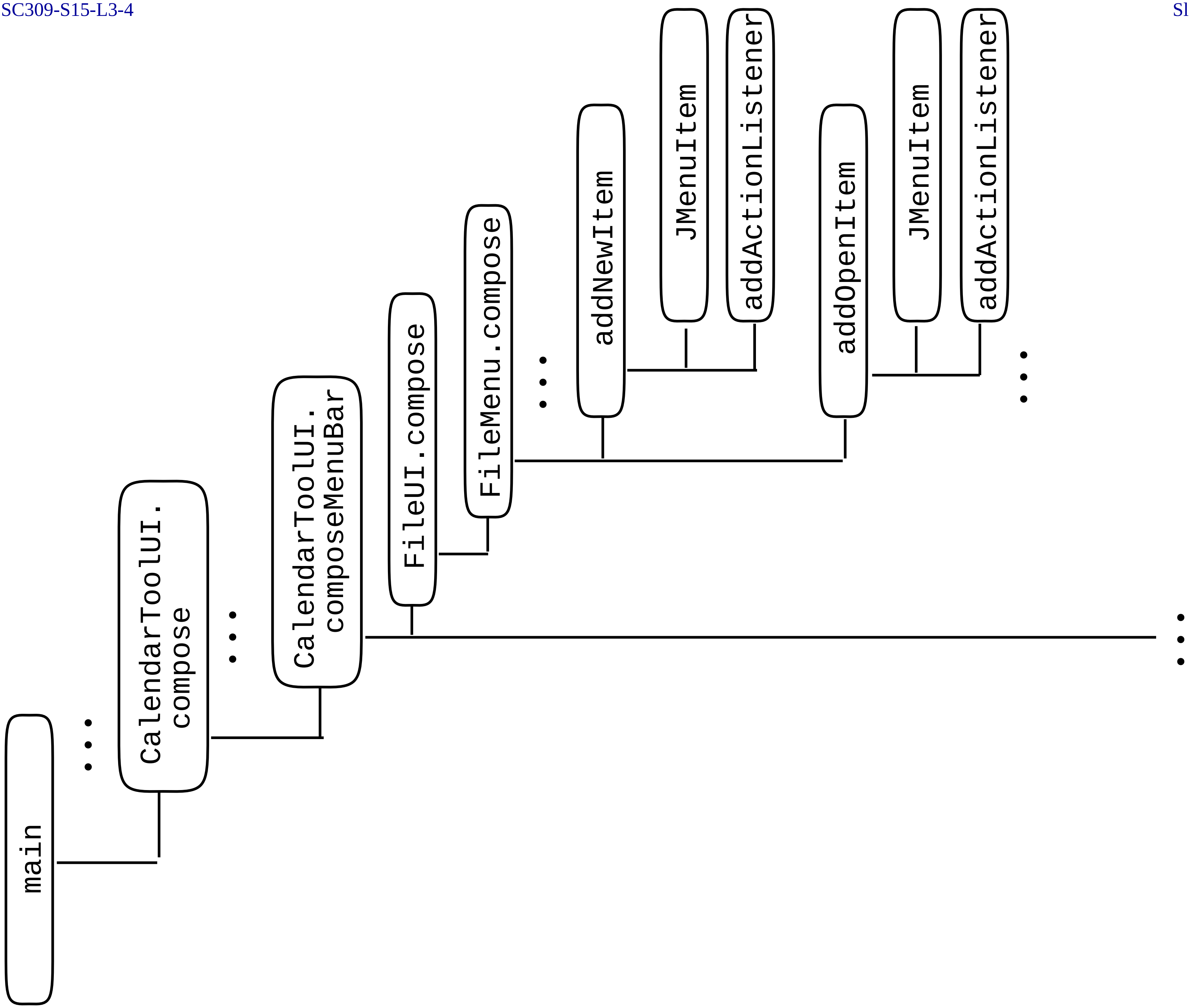
a. Normal method invocation

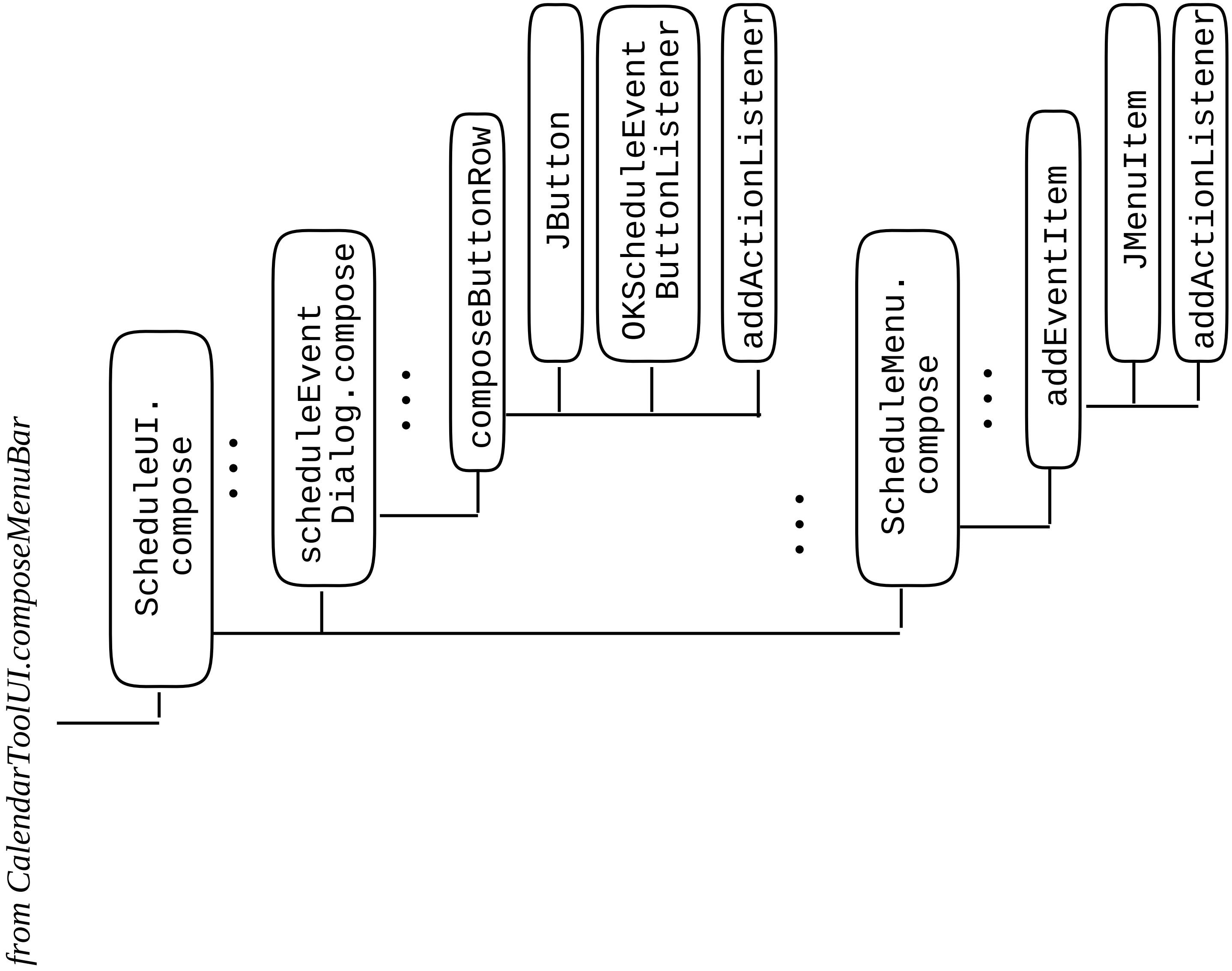


b. Event-based method invocation

XXIII. Examples

A. Figure 11 shows setting up event handlers.


main
CalendarToolUI.
compose
CalendarToolUI.
composeMenuBar
FileUI.compose
FileMenu.compose
addNewItem
JMenuItem
addActionListener
addOpenItem
JMenuItem
addActionListener



from CalendarToolUI.composeMenuBar

ScheduleUI.
compose

. . .

scheduleEvent
Dialog.compose

. . .

composeButtonRow

JButton

OKScheduleEvent
ButtonListener

addActionListener

. . .

ScheduleMenu.
compose

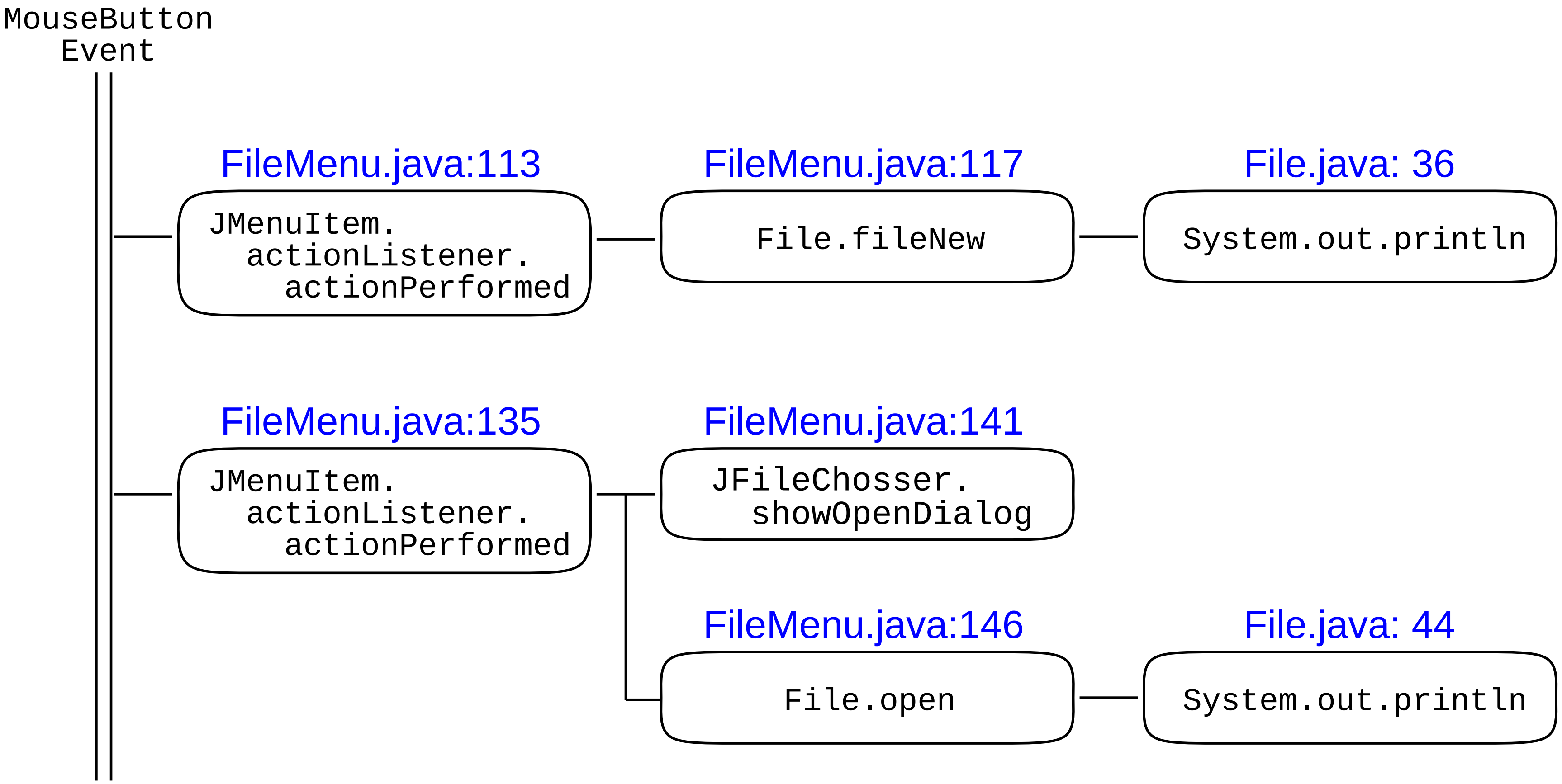
. . .

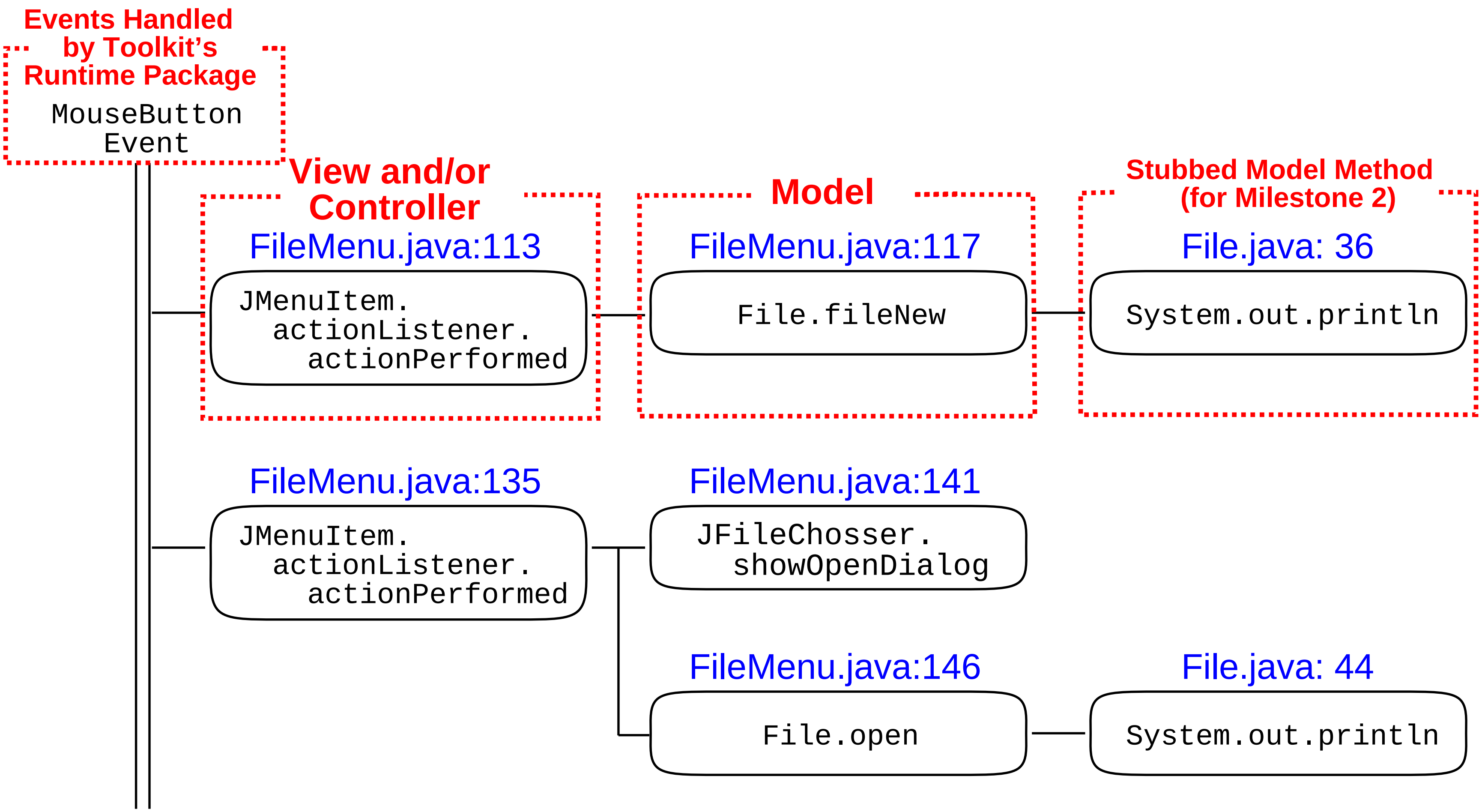
addEventItem

JMenuItem

addActionListener

B. Figure 9 shows event-based invocation.


MouseButton Event
FileMenu.java:113
JMenuItem. actionListener. actionPerformed
FileMenu.java:117
File.fileNew
File.java: 36
System.out.println
FileMenu.java:135
JMenuItem. actionListener. actionPerformed
FileMenu.java:141
JFileChosser. showOpenDialog
FileMenu.java:146
File.open
File.java: 44
System.out.println



Events Handled by Toolkit's Runtime Package

MouseButton Event

View and/or Controller

FileMenu.java:113

JMenuItem. actionListener. actionPerformed

Model

FileMenu.java:117

File.fileNew

Stubbed Model Method (for Milestone 2)

File.java: 36

System.out.println

FileMenu.java:135

JMenuItem. actionListener. actionPerformed

FileMenu.java:141

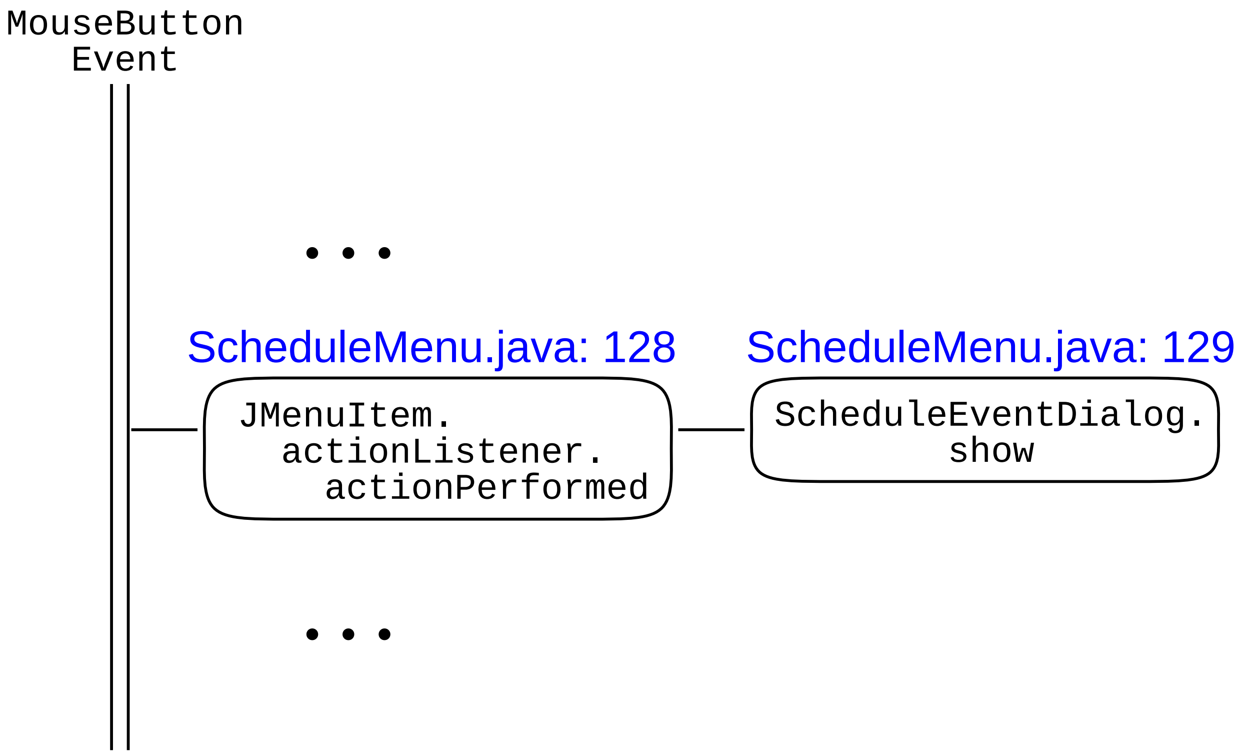
JFileChosser. showOpenDialog

FileMenu.java:146

File.open

File.java: 44

System.out.println



MouseButton Event

• • •

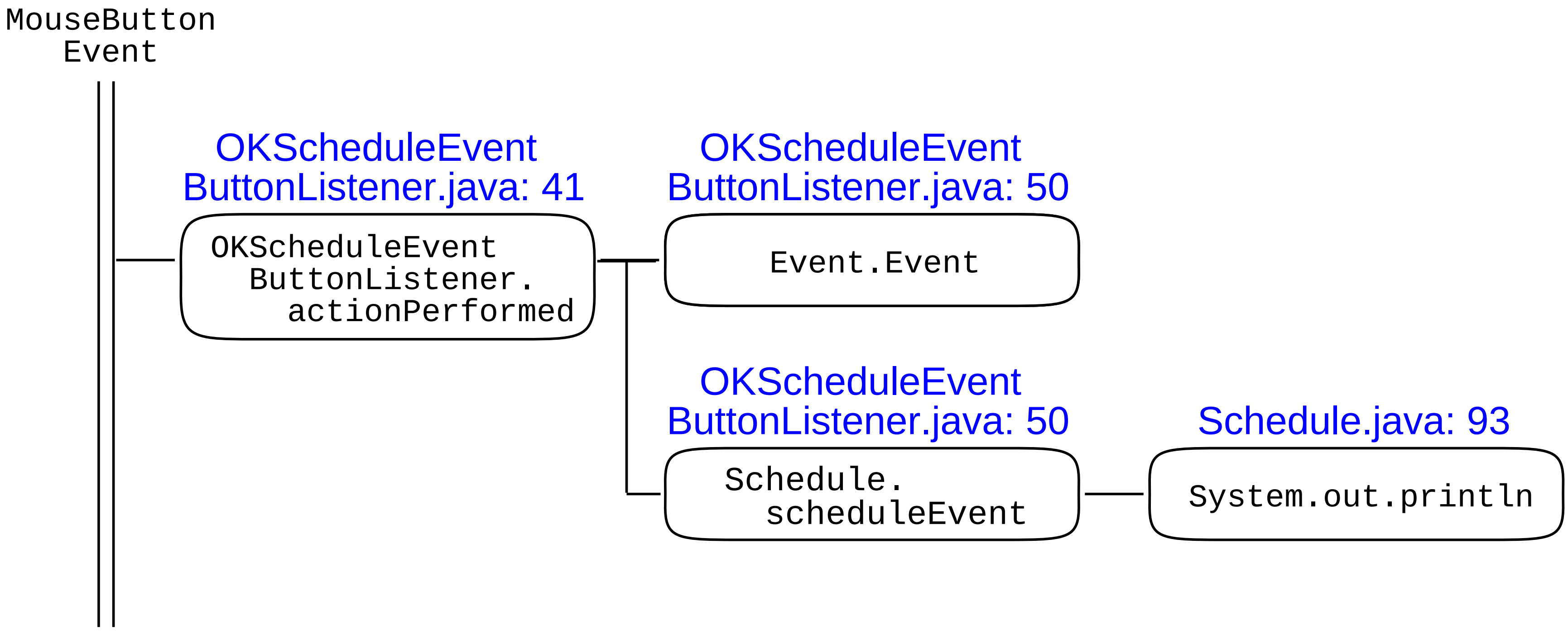
ScheduleMenu.java: 128

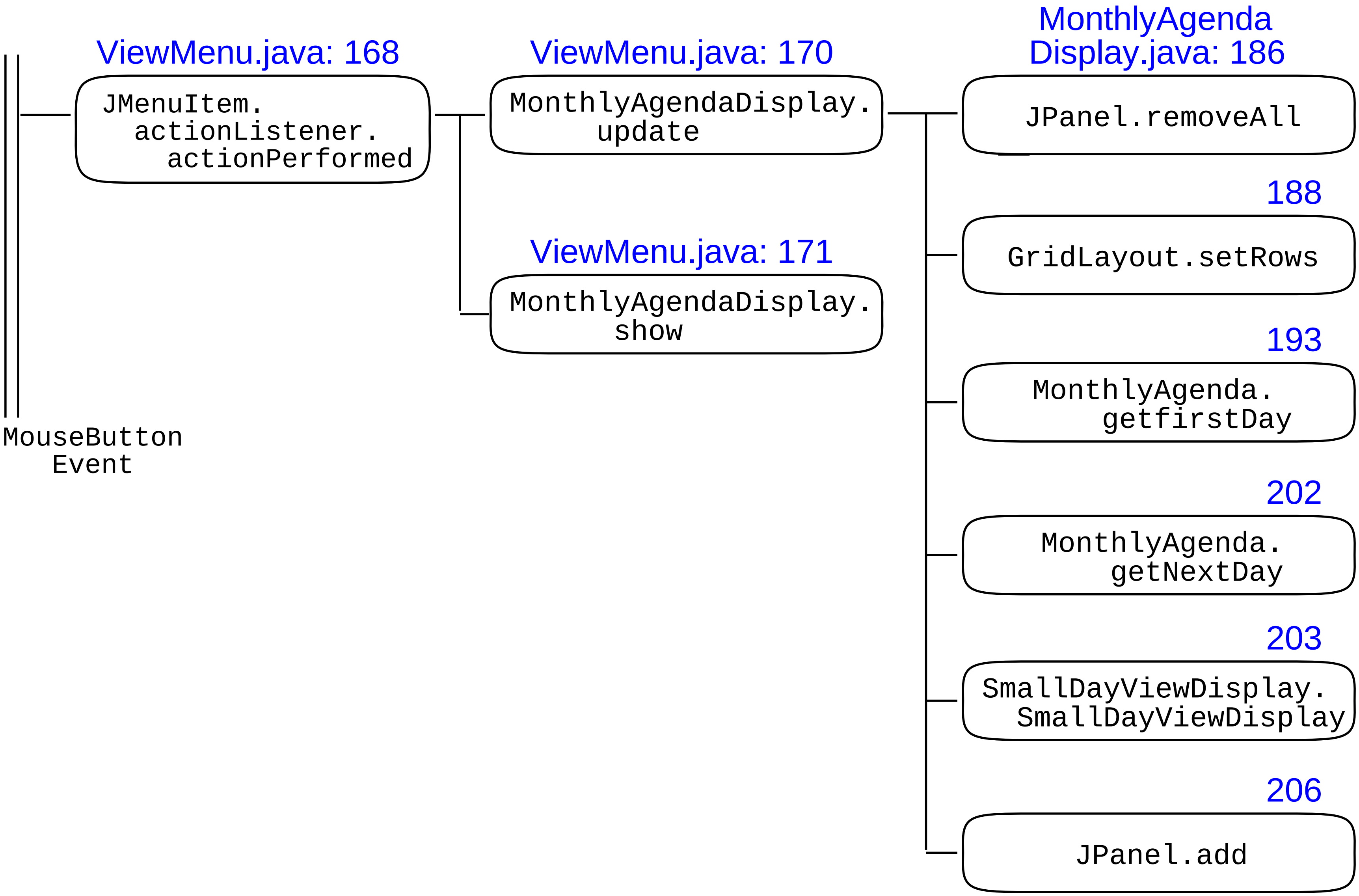
```
JMenuItem.
  actionListener.
    actionPerformed
```

ScheduleMenu.java: 129

```
ScheduleEventDialog.
        show
```

• • •


MouseButton
Event
OKScheduleEvent
ButtonListener.java: 41
OKScheduleEvent
ButtonListener.
actionPerformed
OKScheduleEvent
ButtonListener.java: 50
Event.Event
OKScheduleEvent
ButtonListener.java: 50
Schedule.
scheduleEvent
Schedule.java: 93
System.out.println



MouseButton
Event

ViewMenu.java: 168

JMenuItem.
actionListener.
actionPerformed

ViewMenu.java: 170

MonthlyAgendaDisplay.
update

ViewMenu.java: 171

MonthlyAgendaDisplay.
show

MonthlyAgenda
Display.java: 186

JPanel.removeAll

188

GridLayout.setRows

193

MonthlyAgenda.
getfirstDay

202

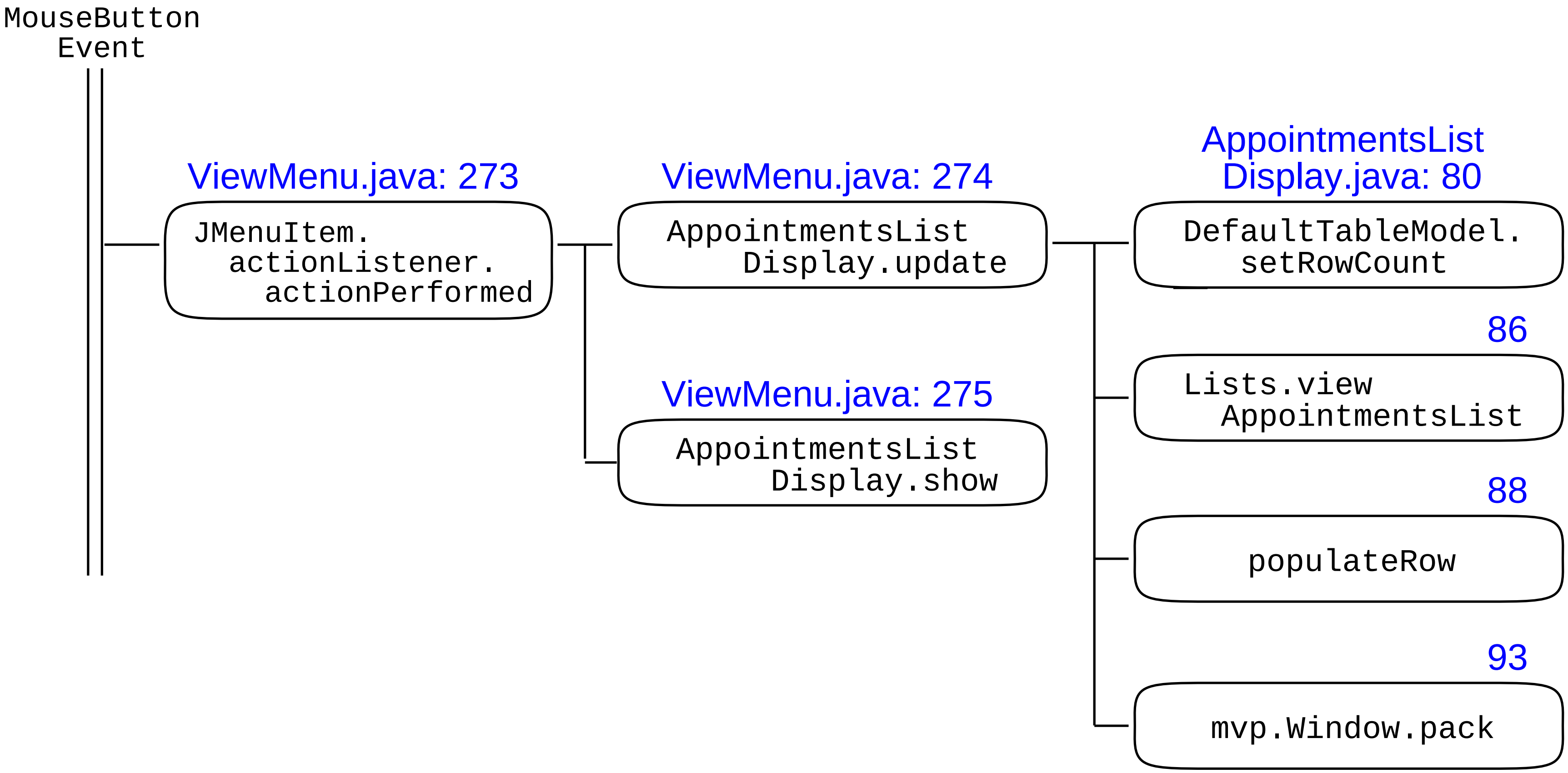
MonthlyAgenda.
getNextDay

203

SmallDayViewDisplay.
SmallDayViewDisplay

206

JPanel.add



MouseButton
Event

ViewMenu.java: 273

JMenuItem.
actionListener.
actionPerformed

ViewMenu.java: 274

AppointmentsList
Display.update

ViewMenu.java: 275

AppointmentsList
Display.show

AppointmentsList
Display.java: 80

DefaultTableModel.
setRowCount

86

Lists.view
AppointmentsList

88

populateRow

93

mvp.Window.pack

0. Schedule.java

```
package caltool.schedule;

import caltool.caldb.*;
import mvp.*;

/****
 *
 * Class Schedule is the top-level model
 * provides methods to schedule the four
 * contains a Categories data field, whi
 * scheduled item categories.
 *
 * @author Gene Fisher (gfisher@calpoly.
 *
 */
public class Schedule extends Model {

    /**
     * Construct this with the given com
     * model.  The CalendarDB is provide
     * store items in the current user c
     * empty error exceptions for each m
     *
     * pre: ;
     *
     * post: this.calDB == calDB && sche
     */
    public Schedule(View view, CalendarD
        super(view);
        this.calDB = calDB;
        scheduleEventPrecondViolation =
    }

    /*-*
     * Derived methods
     */

    /**
     * ScheduleAppointment adds the give
     * if an appointment of the same tim
     * scheduled.
     *
     * pre:
```

```
*
*       //
*       // The StartOrDueDate field
*       //
*       ((appt.start_or_due_date !=
*
*           &&
*
*       //
*       // If non-empty, the EndDat
*       //
*       ((appt.end_date != null) ||
*
*           &&
*
```

```
*       //
*       // The duration is between
*       //
*       ((appt.duration <= 1) && (a
*
*           &&
*
*       //
*       // If weekly recurring is s
*       // must be selected.
*       //
*       if appt.recurring.is_recurr
*       then appt.recurringInfo.det
*            appt.recurringInfo.det
*            appt.recurringInfo.det
*            appt.recurringInfo.det
*            appt.recurringInfo.det
*            appt.recurringInfo.det
*            appt.recurringInfo.det
*
*           &&
*
*       //
*       // No appointment or meetin
*       // and Title is in the curr
*       // CalendarDB.  The current
*       //
*       //     cdb.workspace.calend
*       //
*       // The index is 1 since, by
*        // maintained in most-recen
*        // being most recent and th
*        //
*        ! (exists (item in calDB.ge
*            (item.start_or_due_date
*            (item.duration.equals(a
*            (item.title.equals(appt
*
* post:
*
*        //
*        // Throw exceptions if prec
*        //
*        if (validateInputs(appt).an
*        then throw == scheduleAppoi
```

```
*
*           ||
*
*       if (alreadySchededuled(even
*       then throw == scheduleAppoi
*
*       ||
*
*       if (calDB.getCurrentCalenda
*       then throw == scheduleAppoi
*
*           ||
*
*      //
*      // If preconds met, a schedu
*      // and only if it is the new
*      // input calendar.
*      //
*      (forall (item in calDB’.getC
*          ((item == event) ||
*              (item in calDB.getCu
*
*          &&
*
*      (calDB’.getCurrentCalendar()
*          calDB.getCurrentCalendar
*
*          &&
*
*      (calDB’.getCurrentCalendar()
 *
 *          &&
 *
 *      (calDB’.getCurrentCalendar()
 *
 */
public void scheduleAppointment(Appo
    System.out.println("In Schedule.
}

/**
 * ScheduleMeeting adds a Meeting to
 * given MeetingRequest.  The work i
 * which determine a list of possibl
 * meeting scheduling options, and
 * meeting selected from the possibl
 */
public void scheduleMeeting(MeetingR
    System.out.println("In Schedule.
}

/**
 * Produce the list of possible meet
 * MeetingRequest.
 */
public PossibleMeetingTimes listMeet
    System.out.println("In schedule.
    return null;
}
```

```
/**
 * Set the meeting options in the Ca
 *
 */
public void setMeetingOptions(Meetin
    System.out.println("In schedule.
}

/**
 * ConfirmMeeting takes a CalendarDB
 * PossibleMeetingTimes, and a selec
 * new CalendarDB with the given req
 */
public void confirmMeeting(MeetingRe
        PossibleMeetingTimes possibl
    System.out.println("In Schedule.
}

/**
 * ScheduleTask adds the given Task
 * the same start date, title, and p
 */
public void scheduleTask(Task task)
    System.out.println("In Schedule.
}

/**
 * ScheduleEvent adds the given Even
 * of the same start date and title
 *
```

```
* pre:
*
*      //
*      // The Title field is at lea
*      //
*      ((event.title != null) && (e
*
*          &&
*
*      //
*      // The StartOrDueDate field
*      //
*      ((event.startOrDueDate != nu
*
*          &&
*
*      //
*      // If non-empty, the EndDate
*      //
*      ((event.endDate == null) ||
*
*          &&
*
*      //
*      // The current workspace is
*      //
*      (calDB.getCurrentCalendar()
*
*          &&
*
```

```
*      //
*      // No event of same StartDat
*      // calendar of the given Cal
*      //
*      ! (exists (item in calDB.get
*          (item.startOrDueDatevent
*          (item.title.equals(event
*
*  post:
*      //
*      // Throw exceptions if preco
*      //
*      if (validateInputs(event).an
*      then throw == scheduleEventP
*
```

```
*          ||
*
*      if (alreadySchededuled(event
*      then throw == scheduleEventP
*
*          ||
*
*      if (calDB.getCurrentCalendar
*      then throw == scheduleEventP
*
*          ||
*
*      //
*      // If preconds met, a schedu
*      // and only if it is the nev
```

```
 *         // input calendar.
 *         //
 *         (forall (item in calDB’.getC
 *             ((item == event) ||
 *                 (item in calDB.getCu
 *
 *             &&
 *
 *         (calDB’.getCurrentCalendar()
 *             calDB.getCurrentCalendar
 *
 *             &&
 *
 *         (calDB’.getCurrentCalendar()
 *
 *          &&
 *
 *      (calDB’.getCurrentCalendar()
 *
 */
public void scheduleEvent(Event even
        throws ScheduleEventPrecondV

    /*
     * Clear out the error fields in
     */
    scheduleEventPrecondViolation.cl

    /*
     * Throw a precond violation if
 * or end date.
 */
if (validateInputs(event).anyErr
    throw scheduleEventPrecondVi
}

/*
 * Throw a precond violation if
 * title is already scheduled.
 */
if (alreadyScheduled(event)) {
    scheduleEventPrecondViolatio
    throw scheduleEventPrecondVi
}
```

```
/*
 * Throw a precond violation if
 * Note that this condition will
 * through the view, since the '
 * whenever the there is no acti
 */
if (calDB.getCurrentCalendar() =
    scheduleEventPrecondViolatio
    throw scheduleEventPrecondVi
}

/*
 * If preconditions are met, add
 * active calendar.
 */
```

```
    calDB.getCurrentCalendar().add(e

}

/**
 * Change the given old appointment
 * current calendar.
 */
public void changeAppointment(Appoin
    System.out.println("In Schedule.
}

/**
 * Delete the given appointment from
 */
```

```
public void deleteAppointment(Appoin
    System.out.println("In Schedule.
}


/*-*
 * Access methods
 */

/**
 * Return the categories component.
 *
 * pre: ;
 * post: return == categories;
 */
```

```
public Categories getCategories() {
    return categories;
}

/**
 * Convert this to a printable strin
 * only converted shallow since no m
 * categories.  The deep string conv
 * since it’s the object to which th
 */
public String toString() {
    return
        "Categories: " + categories
        "caldDB.currentCalendar:0 +
        calDB.getCurrentCalendar() +
}

/*-*
 * Protected methods
 */

/**
 * Return true if there is an alread
 * any of the same dates as the give
 */
protected boolean alreadyScheduled(E

    /*
     * Implementation forthcoming.
     */
    return false;

    /*
     * The following won’t fully wor
     *
    return calDB.getCurrentCalendar(
        new ItemKey(e.startDate, nul
     *
     */

}

/**
 * Validate the <a href= Schedule.ht
 * </a> precondition.  Return the ap
 * scheduleEventPrecondViolation obj
 * ScheduleEventPrecondViolation.htm
 * for further details.
 */
protected ScheduleEventPrecondViolat

    if ((event.getTitle() == null) |
        scheduleEventPrecondViolatio
    }

    if (! event.getStartDate().isVal
        scheduleEventPrecondViolatio
    }

    if ((event.getEndDate() != null)
        scheduleEventPrecondViolatio
    }

    return scheduleEventPrecondViola

}


/*-*
 * Derived data fields
 */

/** Category list in which scheduled
protected Categories categories;
```

```
/*-*
 * Process data fields
 */

/** Calendar database that contains
 * items are stored */
protected CalendarDB calDB;

/** Precond violation exception obje
protected ScheduleAppointmentPrecond
    scheduleAppointmentPrecondViolat
protected ScheduleMeetingPrecondViol
protected ScheduleTaskPrecondViolati
protected ScheduleEventPrecondViolat
```

```
}
```

0. ScheduledItem.java

```
package caltool.schedule;

import caltool.caldb.*;
import mvp.Model;

/****
 *
 * A ScheduledItem is the generic defini
 * calendar.  The Title component is a b
 * for.  The StartOrDueDate and EndDate
 * scheduled.  The Category component is
 * color-coded categories.
 *
 * There are four specializations of Sch
 * Meeting, Event, and Task, q.q.v.
 *
 * @author Gene Fisher (gfisher@calpoly.
 * @version 6feb04
 *
 */

public abstract class ScheduledItem exte

    /**
     * Construct an empty scheduled item
 */
public ScheduledItem() {
    super();
}


/*-*
 * Access methods
 */
/**
 * Return the title
 */
public String getTitle() {
    return title;
}
```

```
/**
 * Return the .
 */
public Date getDate() {
    return startOrDueDate;
}

/**
 * Return the end date.
 */
public Date getEndDate() {
    return endDate;
}

/**
```

```
  * Return the category.
  */
public Category getCategory() {
    return category;
}


/*-*
  * Process methods
  */

/**
  * Return the unique lookup key for
  * each subclasss per the unique key
  * <a href=UserCalendar.html> UserC
```

```
 *
 */
public abstract ItemKey getKey();


/*-*
 * Derived data fields
 */

/** Brief description of the schedul
protected String title;

/** Date on which item is scheduled
    multi-purpose field of Scheduled
    an item is a Task and whether it
```

```
        For non-recurring appointments a
        the single date on which the ite
        recurring, StartOrDueDate is the
        non-recurring Task, StartOrDueDa
        If the task is recurring, StartO
     */
    protected Date startOrDueDate;

    /** Last date on which item is sched
        meetings, and tasks, the end dat
        item will recur.  In events, the
        multi-day event.  When the value
        StartOrDueDate component is inte
        item occurs.
     */
    protected Date endDate;

    /** Used to organize items into rela
    protected Category category;

}
```

0. **Appointment.java**

```
package caltool.schedule;

import caltool.caldb.*;
import mvp.*;
```

```
/****
 *
 * Class Appointment adds a number of co
 * The StartTime and Duration indicate w
 * it lasts.  The RecurringInfo defines
 * Location is where it is held.  The Se
 * appointment is scheduled.  Priority i
 * RemindInfo indicates if and how the u
 * Details are free form text describing
 *
 */

public class Appointment extends Schedul

    /**
 * Construct an empty appointment.
 */
public Appointment() {
    super();
}

/**
 * Construct an appointment with the
 * store the unique key for this app
 */
public Appointment(String title, Dat
       Time startTime, Duration dura
       Category category, String loc
       Priority priority, RemindInfo
```

```
this.title = title;
this.startOrDueDate = startOrDue
this.endDate = endDate;
this.startTime = startTime;
this.duration = duration;
this.recurringInfo = recurringIn
this.category = category;
this.location = location;
this.security = security;
this.priority = priority;
this.remindInfo = remindInfo;
this.details = details;

itemKey = new ItemKey(startOrDue
}


/*-*
 * Access methods.
 */

/**
 * Return the start date.
 */
public Date getStartDate() {
    return startOrDueDate;
}

/**
```

```
 * Return the end date.
 */
public Date getEndDate() {
    return endDate;
}

/**
 * Return the start time.
 */
public Time getStartTime() {
    return startTime;
}

/**
 * Return the duration
 */
public Duration getDuration() {
    return duration;
}

/**
 * Return the recurring info.
 */
public RecurringInfo getRecurringInf
    return recurringInfo;
}

/**
 * Return the location.
 */
```

```
public String getLocation() {
    return location;
}

/**
 * Return the security.
 */
public Security getSecurity() {
    return security;
}

/**
 * Return the priority.
 */
public Priority getPriority() {
        return priority;
    }

    /**
     * Return the remind info.
     */
    public RemindInfo getRemindInfo() {
        return remindInfo;
    }

    /**
     * Return the details.
     */
    public String getDetails() {
        return details;
```

```
}



/*-*
 * Process methods
 */

/**
 * Return the unique key for this, c
 * title.  Priority is unused at 0.
 * specialized in Meeting, since app
 * key formats.
 */
public ItemKey getKey() {
    return itemKey;
```

```
}



/*-*
 * Derived data fields, in addition
 */
/** Starting time of the appointment
protected Time startTime;

/** How long the appointment lasts *
protected Duration duration;

/** Defines if and how an appointmen
protected RecurringInfo recurringInf
```

```
/** Where the appointment is held */
protected String location;

/** Indicates who can see that the a
protected Security security;

/** How important the appointment is
protected Priority priority;

/** Indicates if and how user is rem
protected RemindInfo remindInfo;

/** Free-form text describing any sp
protected String details;
```

```
    /*-*
     * Process data field
     */

    /** The uniqe key for storing this i
    protected ItemKey itemKey;
}
```

0. **Meeting.java**

```
package caltool.schedule;

/****
 *
 * A Meeting adds an Attendees component
 * component reflects the fact that a me
 * person, whereas an appointment is for
 * involves checking more than one user
 * among all attendees.  The description
 * further details.
 *
 */

public class Meeting extends Appointment

    /**
     * Construct an empty meeting.
     */
    public Meeting() {
    }

    /*-*
     * Derived data field
     */
    protected Attendees attendees;

}
```

0. **Task.java**

```
package caltool.schedule;

import caltool.caldb.*;
```

```
/****
 *
 * Like an Appointment, a Task adds a nu
 * ScheduledItem.  A Task differs from a
 * Appointments have Duration and Locati
 * Appointments, the priority is either
 * priority is a positive integer indica
 * compared to other tasks.  (3) Tasks h
 * components; Appointments do not.
 *
 */

public class Task extends ScheduledItem

    /**
 * Construct an empty task.
 */
public Task() {
}

/**
 * Construct a task with the given f
 * unique key for this task.
 */
public Task(String title, Date start
        category, Time dueTime, Recu
        security, int priority, Remi
        boolean completedFlag, Date

    this.title = title;
```

```
            this.startOrDueDate = startOrDue
            this.endDate = endDate;
            this.category = category;
            this.dueTime = dueTime;
            this.recurringInfo = recurringIn
            this.security = security;
            this.priority = priority;
            this.remindInfo = remindInfo;
            this.details = details;
            this.completedFlag = completedFl
            this.completionDate = completion

            itemKey = new ItemKey(startOrDue
        }
```

```
/*-*
 * Process methods
 */

/**
 * Return the unique key for this, c
 * priority.  Duration is unsed.
 */
public ItemKey getKey() {
    return itemKey;
}

/*-*
 * Derived data fields
 */
```

```
/** Due time of the task */
protected Time dueTime;

/** Defines if and how an task recur
protected RecurringInfo recurringInf

/** Indicates who can see that the t
protected Security security;

/** Defines the relative priority of
protected int priority;

/** Indicates if and how user is rem
protected RemindInfo remindInfo;
```

```
/** Free-form text describing any sp
protected String details;

/** CompletedFlag is true if a Task
    system does not enforce any spec
    task’s CompletedFlag.  That is,
    Hence the meaning of the Complet
    particularly for recurring tasks
 */
protected boolean completedFlag;

/** CompletionDate is date on which
    not enforce any specific constra
    CompletionDate (other than it be
    of the CompletionDate value is u
```

```
        for recurring tasks.
     */
    protected Date completionDate;


    /*-*
     * Process data field
     */

    /** The uniqe key for storing this i
    protected ItemKey itemKey;

}
```

0. Event.java

```
package caltool.schedule;

import caltool.caldb.*;

/****
 *
 * An Event is the simplest type of Sche
 * to a ScheduledItem is simple security
 *
 * @author Gene Fisher (gfisher@calpoly.
 * @version 6feb04
 *
 */
```

```
public class Event extends ScheduledItem

    /**
     * Construct an empty event.
     */
    public Event() {
    }

    /**
     * Construct an event with the given
     * unique key for this event.
     */
    public Event(String title, Date star
            Category category, SimpleSec
```

```
        this.title = title;
        this.startOrDueDate = startOrDue
        this.endDate = endDate;
        this.category = category;
        this.security = security;

        itemKey = new ItemKey(startOrDue
    }


    /*-*
     * Access methods.
     */

    /**
```

```
 * Return the title.
 */
public String getTitle() {
    return title;
}

/**
 * Return the start date.
 */
public Date getStartDate() {
    return startOrDueDate;
}

/**
 * Return the end date
 */
public Date getEndDate() {
    return endDate;
}

/*-*
 * Process methods
 */

/**
 * Return the unique key for this, c
 * duration, and priority are unused
 */
public ItemKey getKey() {
    return itemKey;
```

```
}

/**
 * Convert this to a string.
 */
public String toString() {
    return
        "0 + "Title: " + title + "0
        "Start date: " + startOrDueD
        "End date: " + (endDate == n
        "Category: " + (category ==
            "0 +
        "Security: " + security + "0
}
```

```
/*-*
 * Derived data field
 */

/** Whether the event is public or p
protected SimpleSecurity security;


/*-*
 * Process data field
 */

/** The uniqe key for storing this i
protected ItemKey itemKey;
```

```
}
```

0. Date.java

```
package caltool.schedule;

import mvp.*;
import java.util.Calendar;
import java.text.*;

/****
 *
 * Class Date is the basic unit of calen
 * of the week, numeric date, month, and
 */

public class Date extends Model implemen

    /**
     * Construct an empty Date.
     */
    public Date() {
        day = null;
        number = 0;
        month = null;
        year = 0;
    }

    /**
 * Construct a date from the given s
 * the given string does not parse a
 * state representation is used inst
 * some users may want to delay the 
 * may not be interested in handling
 *
 * Use java.text.SimpleDateFormat an
 * This means that the first time th
 * format and jCalendar data fields 
 * and Calendar objects, resp.  Thes
 * subsequent Date constructions.
 */
/*@ requires true; ensures true; @*/
public Date(String dateString) {
```

```
constructJCalendarIfNecessary();

try {
    jCalendar.setTime(format.par
    day = convertJavaDay(jCalend
    number = jCalendar.get(Calen
    month = MonthName.values()[j
    year = jCalendar.get(Calenda
    jDate = jCalendar.getTime();
    valid = true;
}
catch (ParseException e) {
    valid = false;
}
```

```
}

/**
 * Construct a date from the given f
 * comments in the String-valued con
 */
public Date(DayName day, int number,

    constructJCalendarIfNecessary();

    this.day = day;
    this.number = number;
    this.month = month;
    this.year = year;
```

```
if (valid =
        (day != null) &&
        (month != null) &&
        (((month == MonthName.Ja
          (month == MonthName.Ma
          (month == MonthName.Ma
          (month == MonthName.Ju
          (month == MonthName.Au
          (month == MonthName.Oc
          (month == MonthName.De
              ((month == MonthNa
               (month == MonthNa
               (month == MonthNa
               (month == MonthNa
                    ((year % 4 ==
                           (number <
            ((year >= 1) && (year <=
        jCalendar.set(year - 1900, m
        jDate = jCalendar.getTime();
    }
}

/**
 * Construct the static java.util.fo
 * time the constructor has been cal
 */
protected void constructJCalendarIfN
    if (format == null) {
        format = (SimpleDateFormat)
            DateFormat.getDateInstan
        jCalendar = format.getCalend
    }

}

/**
 * Convert a java.util.Calendar.DAY_
 * caltool.schedule.DayName enum.  T
 * (necessarily) map the pseudo-enum
 * numeric sequence.  The last time
 */
protected DayName convertJavaDay(int
    switch (javaDayNum) {
        case Calendar.SUNDAY: return
        case Calendar.MONDAY: return
```

```
            case Calendar.TUESDAY: retur
            case Calendar.WEDNESDAY: ret
            case Calendar.THURSDAY: retu
            case Calendar.FRIDAY: return
            case Calendar.SATURDAY: retu
            default: return null;
        }
    }

    /**
     * Return true if this is a valid da
     */
    public boolean isValid() {
        return valid;
    }
```

```
/**
 * Return true if this is an empty d
 */
public boolean isEmpty() {
    return number == 0;
}

/**
 * Return the string representation 
 */
public String toString() {
    return day.toString().concat(" "
        concat(" ").concat(month.toS
            concat(Integer.toString(
}
```

```
/**
 * Define equality for this as compo
 */
public boolean equals(Object obj) {
    Date otherDate = (Date) obj;

    return
        day.equals(otherDate.day) &&
        number == otherDate.number &
        month.equals(otherDate.month
        year == otherDate.year;
}

/**
 * Define compareTo using java.util
 * dates is defined as follows: (1)
 * invalid.
 *
 */
public int compareTo(Object o) {
    Date otherDate = (Date) o;

    if ((! valid) && (! otherDate.va
        return 0;
    }
    if ((! valid) && (otherDate.vali
        return -1;
    }
    if ((valid) && (! otherDate.vali
        return 1;
        }

        /*
         * If both dates are valid, comp
         */
        return jDate.compareTo(otherDate
    }

    /**
     * Define the hash code for this as
     * code is used in turn by ItemKey.h
     */
    public int hashCode() {
        return day.hashCode() + number +
    }
```

```
/*-*
 * Derived data fields
 */

/** One of the seven standard days o
protected DayName day;

/** Numeric date in a month, between
protected int number;

/** One of the twelve months of the
protected MonthName month;

/** The four-digit year number. (Yes
    problem.)
 */
protected int year;

/** True if this is a valud date */
protected boolean valid;

/** The JFC SimpleDateFormat object
SimpleDateFormat format;

/** The JFC  object to use for date
Calendar jCalendar;

/** The java.util.Date value that re
    may be the only data rep of this
    data fields around as well.  At
```

```
            date rep is in this.compareTo. *
        protected java.util.Date jDate;

}
```

0. **DayName.java**

```
package caltool.schedule;

/****
 *
 * Class DayName is one of the seven sta
 *
 * @author Gene Fisher (gfisher@calpoly.
 * @version 25jan10
 *
 */

public enum DayName  {
    /** One of the seven days of the wee
    Sunday,

    /** One of the seven days of the wee
    Monday,

    /** One of the seven days of the wee
    Tuesday,

    /** One of the seven days of the wee
    Wednesday,

    /** One of the seven days of the wee
    Thursday,

    /** One of the seven days of the wee
    Friday,

    /** One of the seven days of the wee
    Saturday
}
```

0. **MonthName.java**

```
package caltool.schedule;

/****
 *
 * Class MonthName is one of the twelve
 *
 * @author Gene Fisher (gfisher@calpoly.
 * @version 15jan10
 *
 */

public enum MonthName {

    /** One of the twelve months of the
    January
```

```
/** One of the twelve months of the
February,

/** One of the twelve months of the
March,

/** One of the twelve months of the
April,

/** One of the twelve months of the
May,

/** One of the twelve months of the
June,
/** One of the twelve months of the
July,

/** One of the twelve months of the
August,

/** One of the twelve months of the
September,

/** One of the twelve months of the
October,

/** One of the twelve months of the
November,
    /** One of the twelve months of the
    December

}
```

0. **Duration.java**

```
package caltool.schedule;

import mvp.*;

/****
 *
 * Duration is the time length of a sche
 * miniumn duration value is 1 minute.
 *
 */

public class Duration extends Model {

    /**
     * Construct an empty duration value
     */
    public Duration() {
        hours = 0;
        minutes = 0;
    }

    /**
 * Construct a duration from the giv
 */
public Duration(int hours, int minut
    this.hours = hours;
    this.minutes = minutes;
}

/**
 * Return true if this is an empty d
 * minutes = 0.
 */
public boolean isEmpty() {
    return (hours == 0) && (minutes
}
```

```
/**
 * Return the string representation
 */
public String toString() {

    String hrString = (hours == 0) ?
        "" : Integer.toString(hours)
            (hours == 1) ? " hr " :

    String minString = (minutes > 0)
        Integer.toString(minutes).co

    return hrString.concat(minString

}
```

```
/**
 * Define equality for this as compo
 */
public boolean equals(Object obj) {
    Duration otherDuration = (Durati

    return
        hours == otherDuration.hours
        minutes == otherDuration.min
}

/**
 * Define the hash code for this as
 * code is used in turn by ItemKey.h
 */
```

```
public int hashCode() {
    return hours + minutes;
}



/*-*
 * Derived data fields
 */


/** Hour component of a duration val
int hours;

/** Minute component of a duration v
int minutes;
```

```
}
```

0. Time.java

```
package caltool.schedule;

import mvp.*;

/****
 *
 * A Time consists of an hour, minute, a
 * expressed using a 12-hour or 24-hour
 * an option by the user.  If the clock
 * is nil.
 *
 */

public class Time extends Model {

    /**
     * Construct an empty time value.
     */
    public Time() {
        hour = 0;
        minute = 0;
        amOrPm = null;
        valid = true;
        empty = true;
    }
```

```
/**
 * Construct a time from the given s
 * the given string does not parse t
 * state representation is used inst
 * some users may want to delay the 
 * may not be interested in handling
 */
public Time(String time) {
    /*
     * Constant stubbed implementati
     */
    hour = 12;
    minute = 0;
    amOrPm = null;
```

```
        valid = true;
        empty = false;
    }

    /**
     * Return true if his is an empty ti
     */
    public boolean isEmpty() {
        return empty;
    }

    /**
     * Return the string representation
     */
    public String toString() {
        return Integer.toString(hour).co
            concat((minute < 10) ? "0" :
                Integer.toString(minute)
                    concat((amOrPm != nu
                        " " + amOrPm.toS
    }

    /**
     * Define equality for this as compo
     */
    public boolean equals(Object obj) {
        Time otherTime = (Time) obj;

        return
            hour == otherTime.hour &&
        minute == otherTime.minute &
        amOrPm.equals(otherTime.amOr
}

/**
 * Define the hash code for this as
 * code is used in turn by ItemKey.h
 */
public int hashCode() {
    return hour + minute + amOrPm.ha
}


/*-*
 * Derived data fields
 */

/** The hour component of a time val
    on the clock style in use
 */
protected int hour;

/** The minute component of a time v
protected int minute;

/** Standard suffix used in 12-hour
protected AmOrPm amOrPm;


/* *
```

```
     * Process data fields
     */

    /** True if this is a valid time */
    boolean valid;

    /** True if this is an empty time, i
        amOrPm = "empty". */
    boolean empty;

}
```

0. **Security.java**

```
package caltool.schedule;

/****
 *
 * Security is one of four possible leve
 * or Private.  The levels specify the d
 * has to other users.  For an appointme
 * defined as all users other than the u
 * item appears.  For a meeting, "other
 * Attendees list of the meeting.
 *
 * Public security means other users can
 * information about the item.
 *
 * PublicTitle security means other user
* item but none of the other informatio
*
* Confidential security means other use
* unavailable for the time period of a
* about the scheduled item is visible.
* a specific time period, it is meaning
* not for tasks or events; tasks and ev
* components.
*
* Private security means other users se
* scheduled item, not even that the ite
* security hides a scheduled item from
* that a meeting may be scheduled at th
* It is up to the user to handle this s
* scheduled meeting.  Given the nature
```

```
 * to meetings.  I.e., only appointments
 *
 * @author Gene Fisher (gfisher@calpoly.
 * @version 15jan10
 *
 */

public enum Security {

    /** Public security means other user
        the information about the item.
    Public,

    /** PublicTitle security means other
        scheduled item but none of the o
```

```
PublicTitle,

/** Confidential security means othe
    unavailable for the time period
    information about the scheduled
    security applies to a specific t
    appointments and meetings, not f
    not have specific time component
Confidential,

/** Private security means other use
    scheduled item, not even that th
    security hides a scheduled item
    q.v., so that a meeting may be s
    appointment.  It is up to the us
```

```
        accepting or refusing the schedu
        private security, it does not ap
        appointments can have private se
    Private

}
```

0. **Priority.java**

```
package caltool.schedule;

/****
 *
 * Priority indicates whether an appoint
 * This information is used to indicate
 * appointment to the user.  The operati
 * ScheduleMeeting operation, where the
 * optional appointments as allowable ti
 *
 * @author Gene Fisher (gfisher@calpoly.
 * @version 15jan10
 *
 */

public enum Priority {


    /** Indicates a scheduled item is 'M
    Must
    /** Indicates a scheduled item is ’O
    Optional

}
```

0. Category.java

```
package caltool.schedule;

import mvp.*;

/****
 *
 * A Category has a Name and StandardCol
 * other categories.  Colored-coded cate
 * when viewing lists of scheduled items
 * used in filtered viewing.
 *
 */

public class Category extends Model {

    /**
     * Construct an empty category.
     */
    public Category() {
    }
```

```
/**
 * Construct a category of the given
 * asking the Catgories class for th
 * cateogories name.
 */
public Category(String name) {
    this.name = name;
    color = Categories.getColor(name


/**
 * Return the string representation
 * not do color.
 */
```

```
public String toString() {
    return name;
}


/*-*
 * Derived data fields
 */

/** Text name of the category */
String name;

/** Color name of the category */
StandardColor color;
```

```
}
```

0. **StandardColor.java**

```
package caltool.schedule;

/****
 *
 * A StandardColor is one of of a fixed set
 * coding a cateogry.  The possible valu
 * "Orange", "Yellow", "Green", "Blue",
 *
 * @author Gene Fisher (gfisher@calpoly.
 * @version 15jan10
 *
 */

public enum StandardColor {

    /** One of the built-in category col
    Black,

    /** One of the built-in category col
    Brown,

    /** One of the built-in category col
    Red,

    /** One of the built-in category col
    Orange,

    /** One of the built-in category col
    Yellow,

    /** One of the built-in category col
    Green,

    /** One of the built-in category col
    Blue,

    /** One of the built-in category col
    Purple

}
```