CSC 307 Lecture Notes Weeks 6

The Program Design Process
High-Level Design Patterns
GUI Design in Java Swing

Milestones 5-6

Milestones 5-6

1. Due Mon 9 November

Milestones 5-6

1. Due Mon 9 November
2. Fully finished requirements

Milestones 5-6

1. Due Mon 9 November
2. Fully finished requirements
3. Model/view design

Milestones 5-6

1. Due Mon 9 November
2. Fully finished requirements
3. Model/view design
4. Initial model/view implementation

Milestones 5-6

1. Due Mon 9 November
2. Fully finished requirements
3. Model/view design
4. Initial model/view implementation
5. Model/view communication for 3 model methods

Milestones 5-6

1. Due Mon 9 November
2. Fully finished requirements
3. Model/view design
4. Initial model/view implementation
5. Model/view communication for 3 model methods
6. Pre/post for 3 model methods

Milestones 5-6 Immediate Action Item

Decide by Wed 28 October if your team wants to use a GUI toolkit other than Java Swing

I. Major goals of the design process

I. Major goals of the design process

A. Adhere to the specification

I. Major goals of the design process

A. Adhere to the specification

1. Any deviation in a SCO

I. Major goals of the design process

A. Adhere to the specification

1. Any deviation in a SCO

2. The spec + SCOs form binding *contract*

I. Major goals of the design process

A. Adhere to the specification

1. Any deviation in a SCO
2. The spec + SCOs form binding *contract*
3. No changes without consulting customer

Goals of design, cont'd

B. Achieve design quality goals:

Goals of design, cont'd

B. Achieve design quality goals:

1. *Traceability*

Goals of design, cont'd

B. Achieve design quality goals:

1. *Traceability*
2. *Modularity*

Goals of design, cont'd

B. Achieve design quality goals:

1. *Traceability*
2. *Modularity*
3. *Portability*

Goals of design, cont'd

B. Achieve design quality goals:

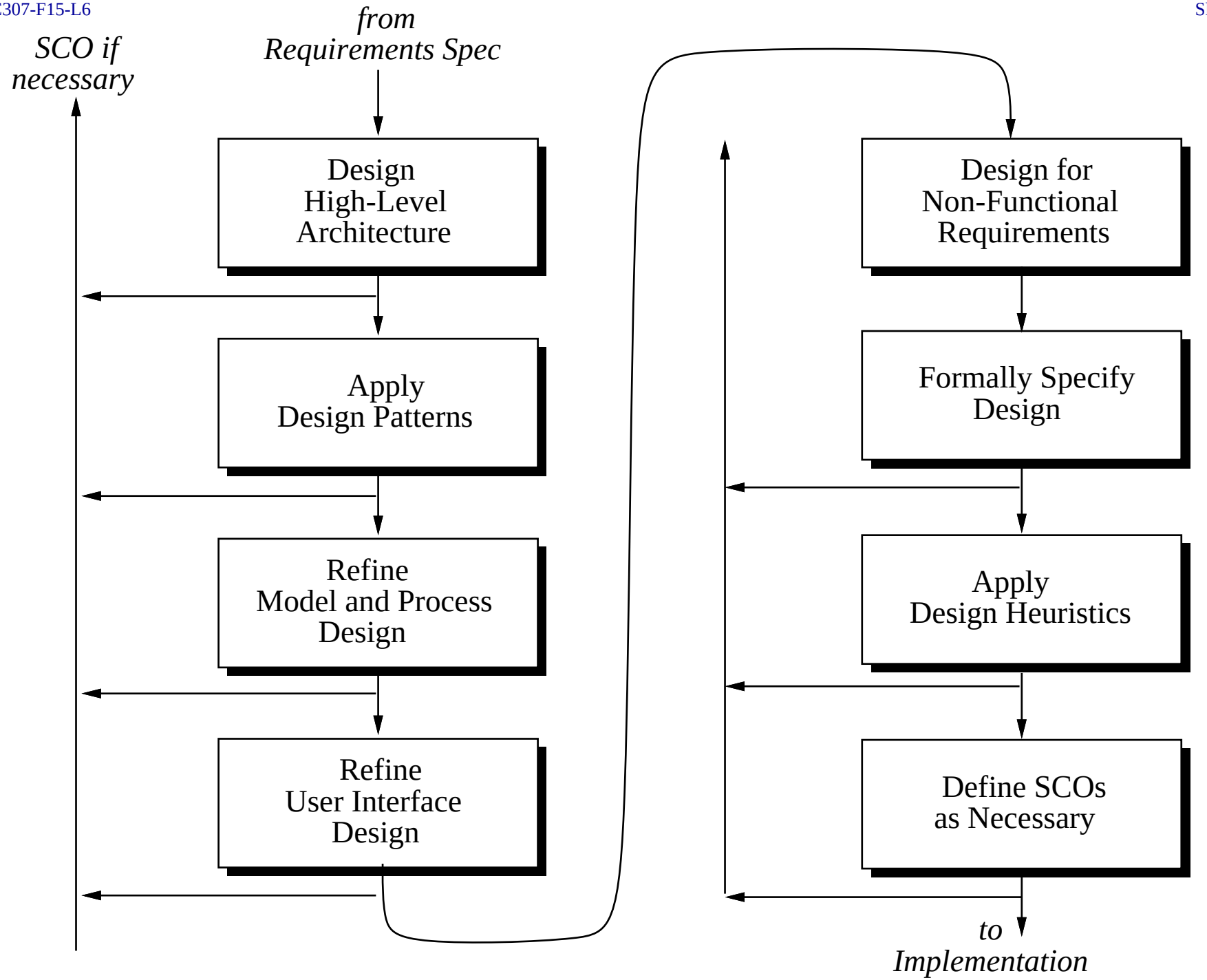
1. *Traceability*
2. *Modularity*
3. *Portability*
4. *Maintainability*

Goals of design, cont'd

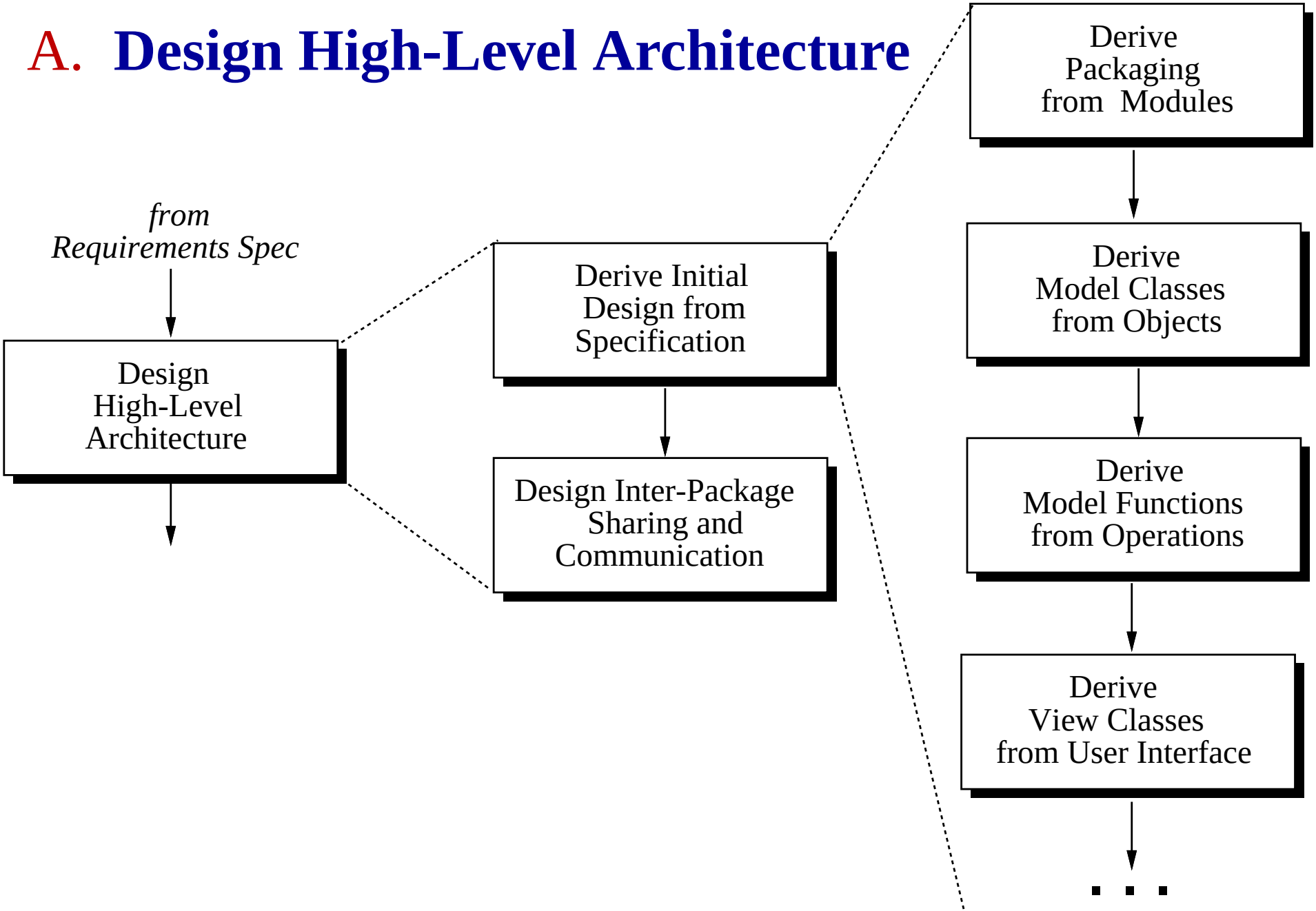
B. Achieve design quality goals:

1. *Traceability*
2. *Modularity*
3. *Portability*
4. *Maintainability*
5. *Reusability*

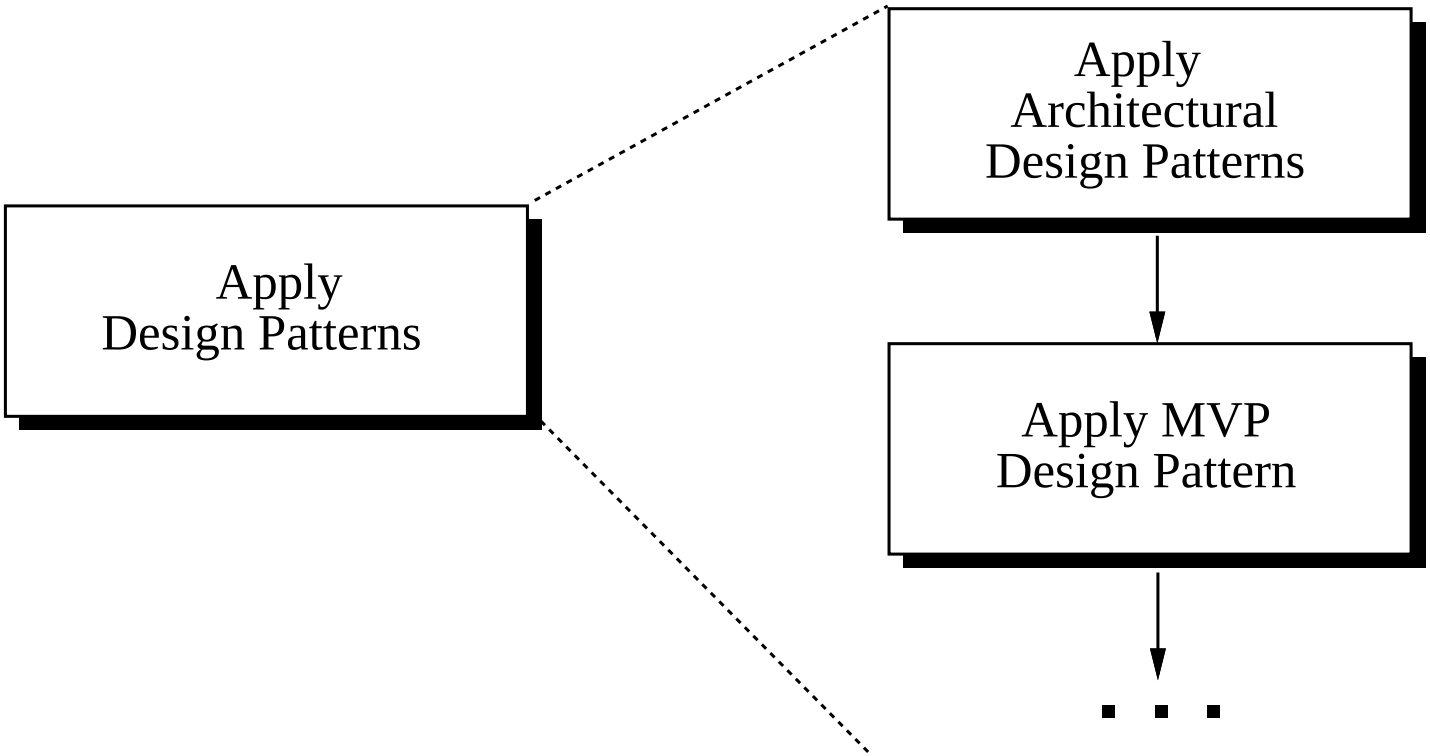
II. 307 design process


from
Requirements Spec
SCO if
necessary
Design
High-Level
Architecture
Apply
Design Patterns
Refine
Model and Process
Design
Refine
User Interface
Design
Design for
Non-Functional
Requirements
Formally Specify
Design
Apply
Design Heuristics
Define SCOs
as Necessary
to
Implementation

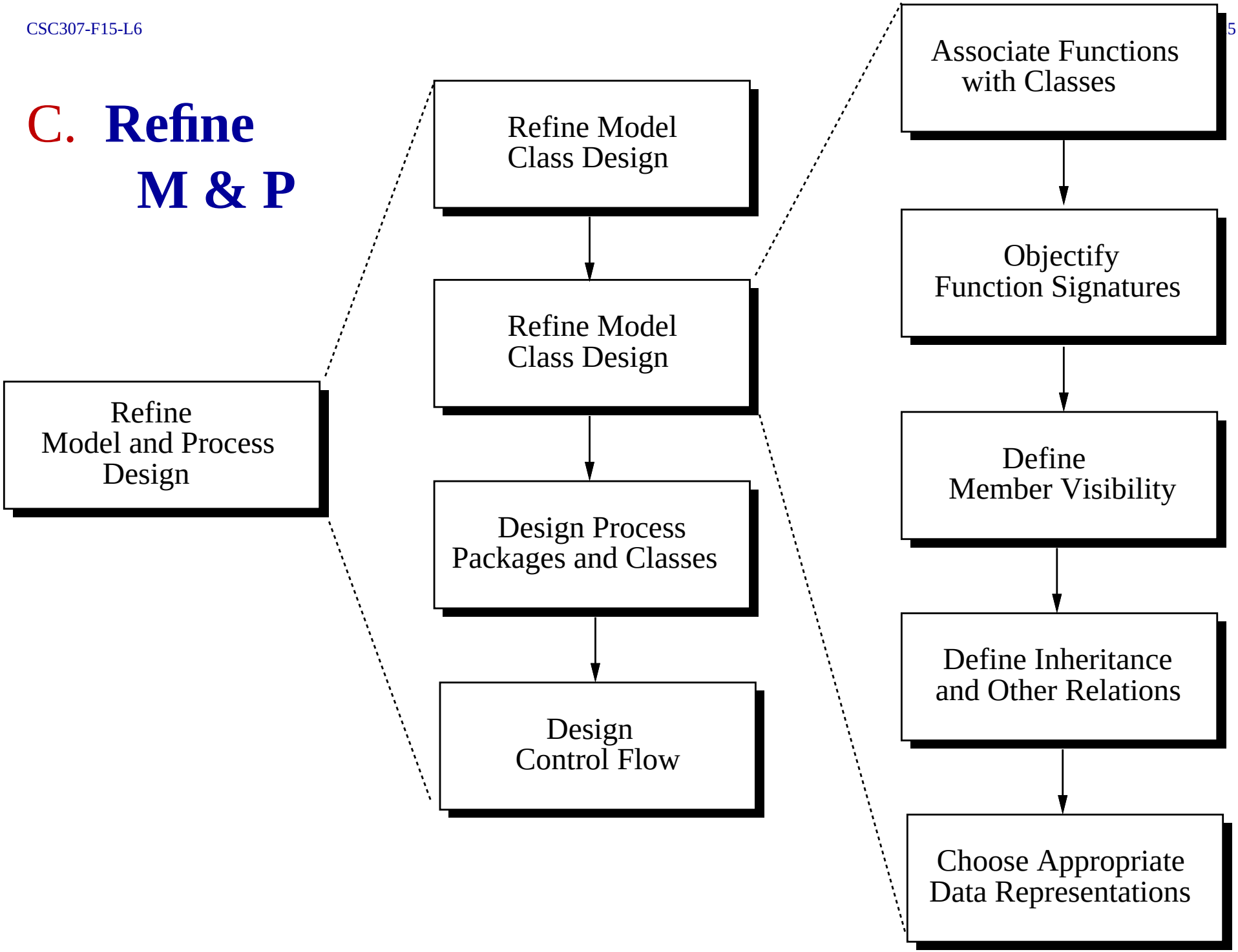
A. Design High-Level Architecture



B. Apply Design Patterns



C. Refine M & P


Refine Model and Process Design
Refine Model Class Design
Refine Model Class Design
Design Process Packages and Classes
Design Control Flow
Associate Functions with Classes
Objectify Function Signatures
Define Member Visibility
Define Inheritance and Other Relations
Choose Appropriate Data Representations

Design process, cont'd

D. Refine UI Design

Design process, cont'd

D. Refine UI Design

1. The fourth step.

Design process, cont'd

D. Refine UI Design

1. The fourth step.
2. Relies heavily on libraries.

Design process, cont'd

D. Refine UI Design

1. The fourth step.
2. Relies heavily on libraries.
3. Commonly-used interface elements and layouts.

Design process, cont'd

D. Refine UI Design

1. The fourth step.
2. Relies heavily on libraries.
3. Commonly-used interface elements and layouts.
4. Model classes must be refined.

Design process, cont'd

D. Refine UI Design

1. The fourth step.
2. Relies heavily on libraries.
3. Commonly-used interface elements and layouts.
4. Model classes must be refined.
5. Particularly useful is "Observer/Observable".

Design process, cont'd

E. Design for Non-Functional Requirements

Design process, cont'd

E. Design for Non-Functional Requirements

1. Any non-functionals not yet incorporated.

Design process, cont'd

E. Design for Non-Functional Requirements

1. Any non-functionals not yet incorporated.

2. Ensure system-related non-functionals are fully addressed.

Design process, cont'd

F. Formally Specify Design

Design process, cont'd

F. Formally Specify Design

1. As detailed program design established.

Design process, cont'd

F. Formally Specify Design

1. As detailed program design established.

2. Precise def of function signatures and pre/post.

Design process, cont'd

F. Formally Specify Design

1. As detailed program design established.
2. Precise def of function signatures and pre/post.
3. Derived from pre/posts defined in ops.

Design process, cont'd

G. Apply Design Heuristics

Design process, cont'd

G. Apply Design Heuristics

1. Applied throughout the process.

Design process, cont'd

G. Apply Design Heuristics

1. Applied throughout the process.
2. Minimizing coupling.

Design process, cont'd

G. Apply Design Heuristics

1. Applied throughout the process.
2. Minimizing coupling.
3. Maximizing cohesion.

Design process, cont'd

G. Apply Design Heuristics

1. Applied throughout the process.
2. Minimizing coupling.
3. Maximizing cohesion.
4. Other heuristics, such as controlling size.

Design process, cont'd

H. Define SCOs and Iterate Back

Design process, cont'd

H. Define SCOs and Iterate Back

1. Aspects of requirements spec may need to be modified or enhanced.

Design process, cont'd

H. Define SCOs and Iterate Back

1. Aspects of requirements spec may need to be modified or enhanced.

2. Designer defines a *specification change order*.

Design process, cont'd

H. Define SCOs and Iterate Back

1. Aspects of requirements spec may need to be modified or enhanced.

2. Designer defines a *specification change order*.

3. In keeping with our "traditional" process.

III. Comments on the 307 Design Process

III. Comments on the 307 Design Process

A. Employs a number of design methodologies.

III. Comments on the 307 Design Process

A. Employs a number of design methodologies.

B. Works for software with substantial HCI.

III. Comments on the 307 Design Process

A. Employs a number of design methodologies.

B. Works for software with substantial HCI.

C. Also for other software, with adjustments.

III. Comments on the 307 Design Process

A. Employs a number of design methodologies.

B. Works for software with substantial HCI.

C. Also for other software, with adjustments.

D. Less applicable to realtime, embedded software.

IV. Languages of specification & implementation.

IV. Languages of specification & implementation.

A. Specification language may differ from the programming language.

IV. **Languages of specification & implementation.**

A. Specification language may differ from the programming language.

B. If so, spec-to-imple transistion is harder.

IV. Languages of specification & implementation.

A. Specification language may differ from the programming language.

B. If so, spec-to-imple transistion is harder.

C. Not the case in 307 this year.

V. What is design?

V. What is design?

A. *Abstraction* of implementation.

V. What is design?

A. *Abstraction* of implementation.

1. Abstraction means *things get left out*.

V. What is design?

A. *Abstraction* of implementation.

1. Abstraction means *things get left out*.

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

V. **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.

V. 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.

What is design, cont'd

B. Levels of design abstraction.

1. **Packaging Design**

What is design, cont'd

B. Levels of design abstraction.

1. **Packaging Design**

a. Largest modular units.

What is design, cont'd

B. Levels of design abstraction.

1. **Packaging Design**

   a. Largest modular units.

   b. Pkg names, descriptions, communication.

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

What is design, cont'd

2. **Abstract Class Design**

   a. Classes added to packages.

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

What is design, cont'd

3. Mid-Level Class Design

a. Add methods and data fields to classes.

What is design, cont'd

3. **Mid-Level Class Design**

   a. Add methods and data fields to classes.

   b. Method and field names, descriptions; no method signatures or concrete data reps.

What is design, cont'd

3. Mid-Level Class Design

a. Add methods and data fields to classes.

b. Method and field names, descriptions; no method signatures or concrete data reps.

c. Define class inheritance.

What is design, cont'd

4. Detailed Class Design

What is design, cont'd

4. Detailed Class Design

a. Add full input/output signatures.

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

What is design, cont'd

5. **Functional Design**

    a. Add pre- and postconditions to methods.

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.

What is design, cont'd

C. At any level, apply suitable patterns.

D. We'll start with two patterns:

What is design, cont'd

C. At any level, apply suitable patterns.

D. We'll start with two patterns:

1. "Model/View/*Whatever*"

What is design, cont'd

C. At any level, apply suitable patterns.

D. We'll start with two patterns:

1. "Model/View/*Whatever*"
2. "Information Processing Tool"

VI. What is a design pattern?

VI. What is a design pattern?

A. From architect Christopher Alexander:

VI. What is a design pattern?

A. From architect Christopher Alexander:

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

VI. 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.

VI. 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.

VII. The MVP pattern

VII. The MVP pattern

A. Separate core processing from GUI and underlying support.

VII. The MVP pattern

A. Separate core processing from GUI and underlying support.

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

VII. 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.

VII. 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 views.

MVP, cont'd

B. Correspondence between models and views.

1. There is typically a *canonical* view.

MVP, cont'd

B. Correspondence between models and views.

1. There is typically a *canonical* view.

2. May also be additional views.

MVP, cont'd

B. Correspondence between models and views.

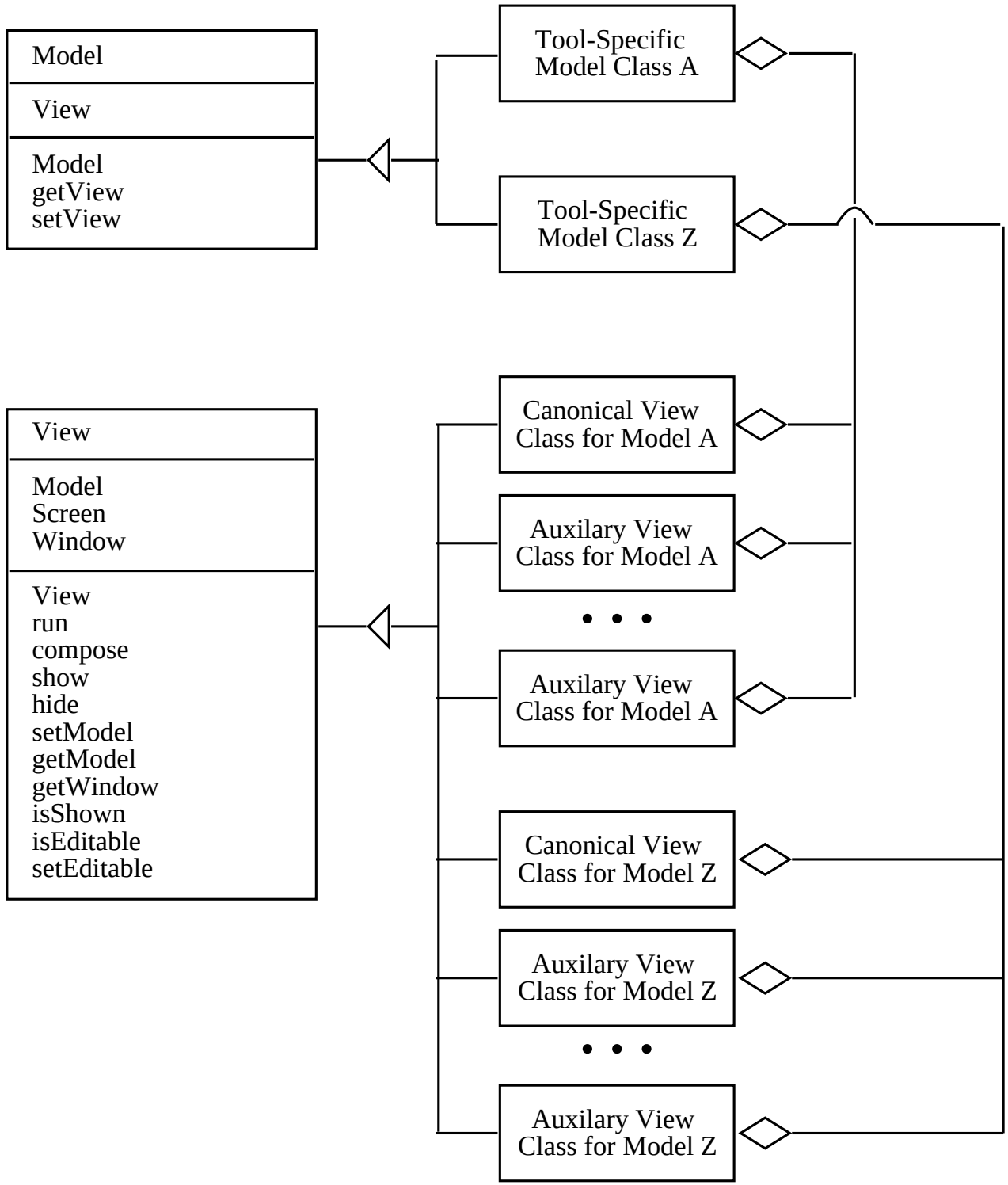
1. There is typically a *canonical* view.

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.

MVP, cont'd

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

2. Defined in 307 Java class library.

VIII. "Info Processing Tool" pattern

VIII. "Info Processing Tool" pattern

A. Used to layout high-level packaging.

VIII. "Info Processing Tool" pattern

A. Used to layout high-level packaging.

B. In conjunction with MVP.

VIII. "Info Processing Tool" pattern

A. Used to layout high-level packaging.

B. In conjunction with MVP.

C. Applies to 307 applications specifically.

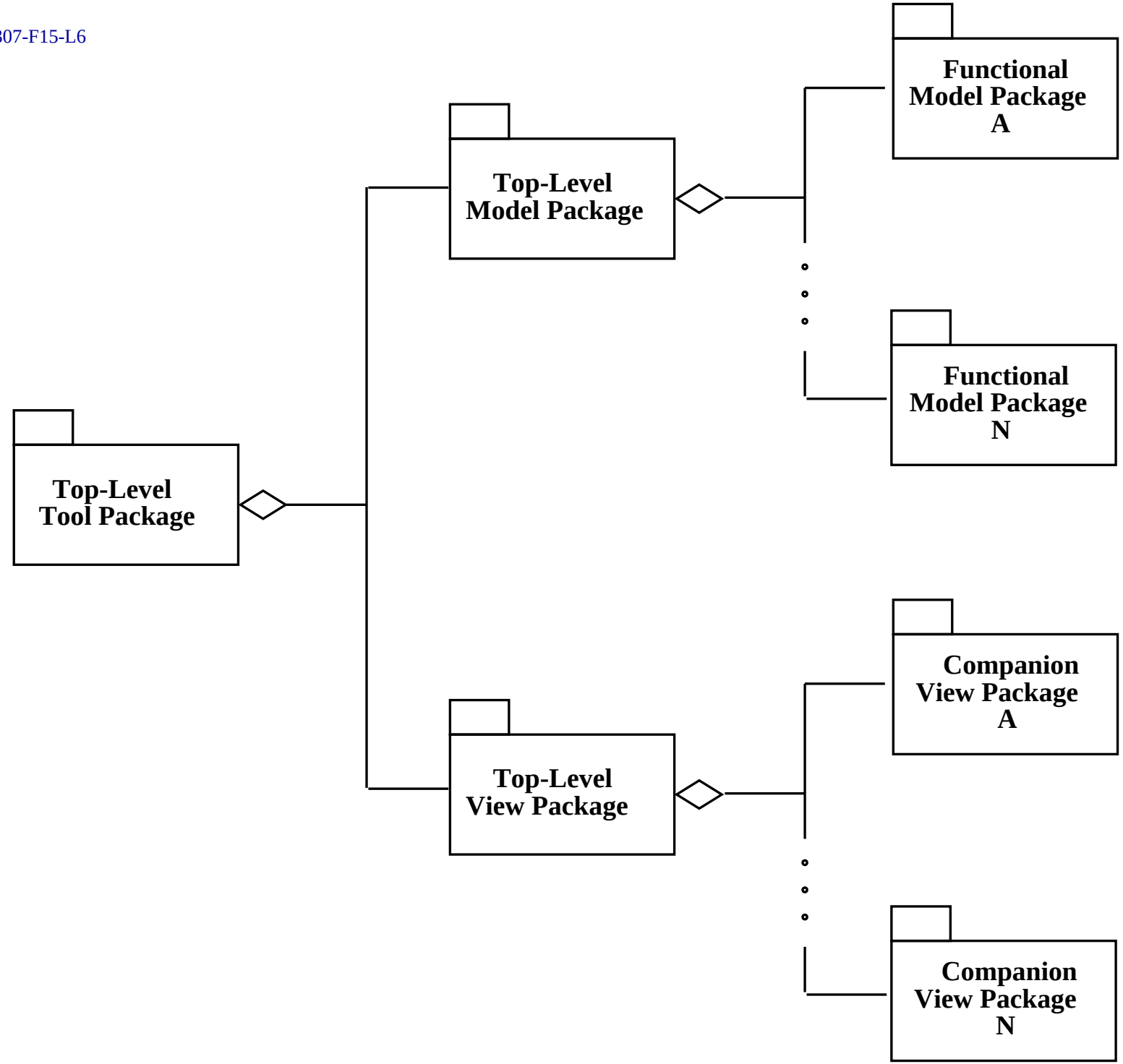
VIII. "Info Processing Tool" pattern

A. Used to layout high-level packaging.

B. In conjunction with MVP.

C. Applies to 307 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:

E. IPT pattern info sources:

1. *The organization of the top-level GUI*

E. IPT pattern info sources:

1. *The organization of the top-level GUI*

2. *The organization of the requirements scenarios*

E. IPT pattern info sources:

1. *The organization of the top-level GUI*
2. *The organization of the requirements scenarios*
3. *Package organization of the Java model*

IX. Calendar Tool example from 307.

IX. Calendar Tool example from 307.

A. Milestone 6 example shows what's expected.

IX. Calendar Tool example from 307.

A. Milestone 6 example shows what's expected.

B. See in particular the executable jar.

X. Applying Info-Processing-Tool pattern

X. Applying Info-Processing-Tool pattern

A. Eight modules in Cal Tool spec:

X. Applying Info-Processing-Tool pattern

A. Eight modules in Cal Tool spec:

1. File -- file processing

X. Applying Info-Processing-Tool pattern

A. Eight modules in Cal Tool spec:

1. File -- file processing

2. Edit -- general editing

X. Applying Info-Processing-Tool pattern

A. Eight modules in Cal Tool spec:

1. File -- file processing
2. Edit -- general editing
3. Schedule -- item scheduling

X. 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

Applying patterns, cont'd

5. Admin -- managing databases

6. Options -- managing options

Applying patterns, cont'd

5. Admin -- managing databases

6. Options -- managing options

7. CalDB -- underlying calendar database

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. Six model packages from main GUI, two more packages from admin scneario narrative.

Applying patterns, cont'd

B. Six model packages from main GUI, two more packages from admin scneario narrative.

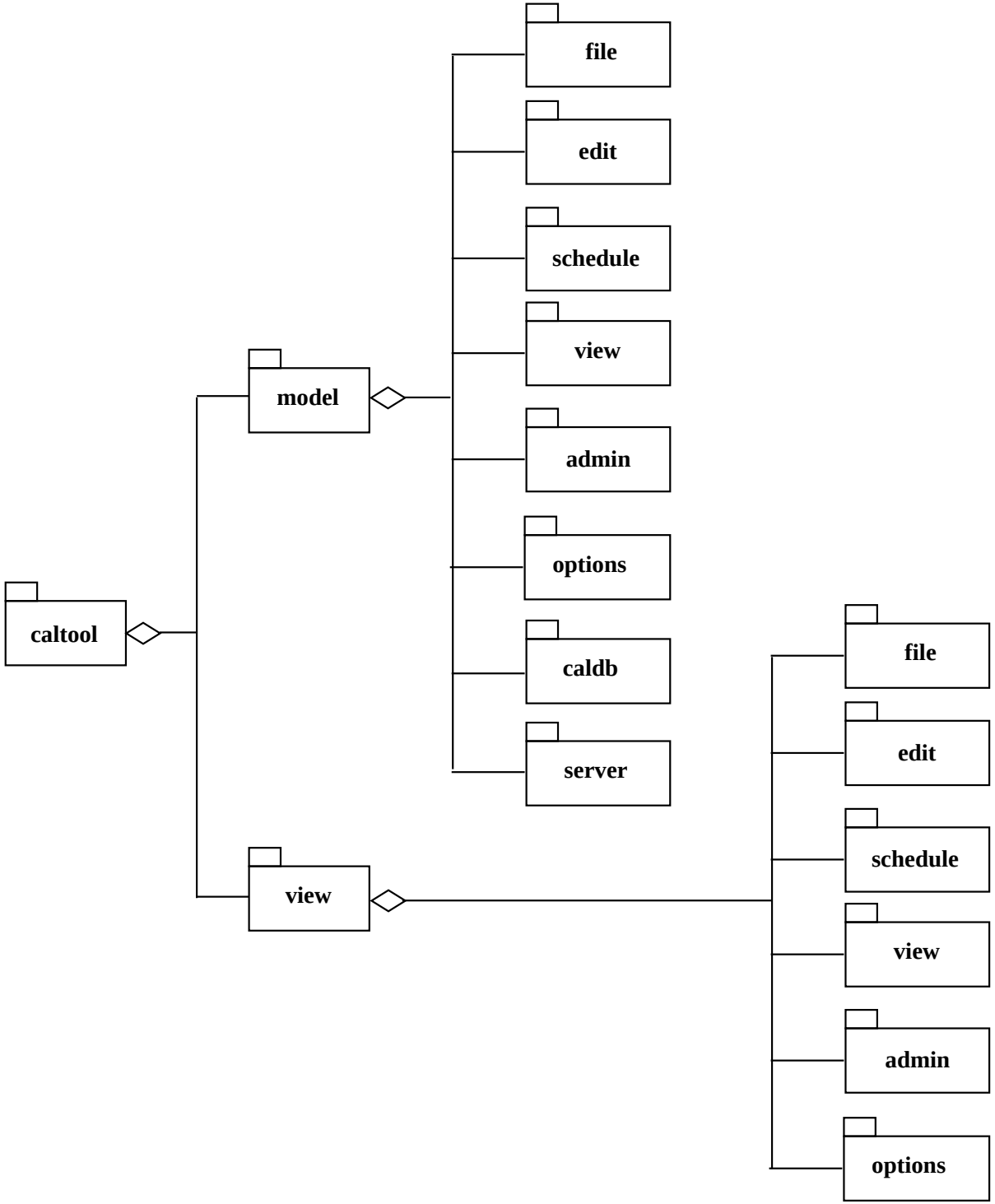
C. Applying IPT, we add a top-level tool package, and companion view packages.

Applying patterns, cont'd

B. Six model packages from main GUI, two more packages from admin scneario narrative.

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
file
edit
view
schedule
view
admin
options

Applying patterns, cont'd

E. No companion UIs for CalDB and Server pkgs.

Applying patterns, cont'd

E. 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;
```

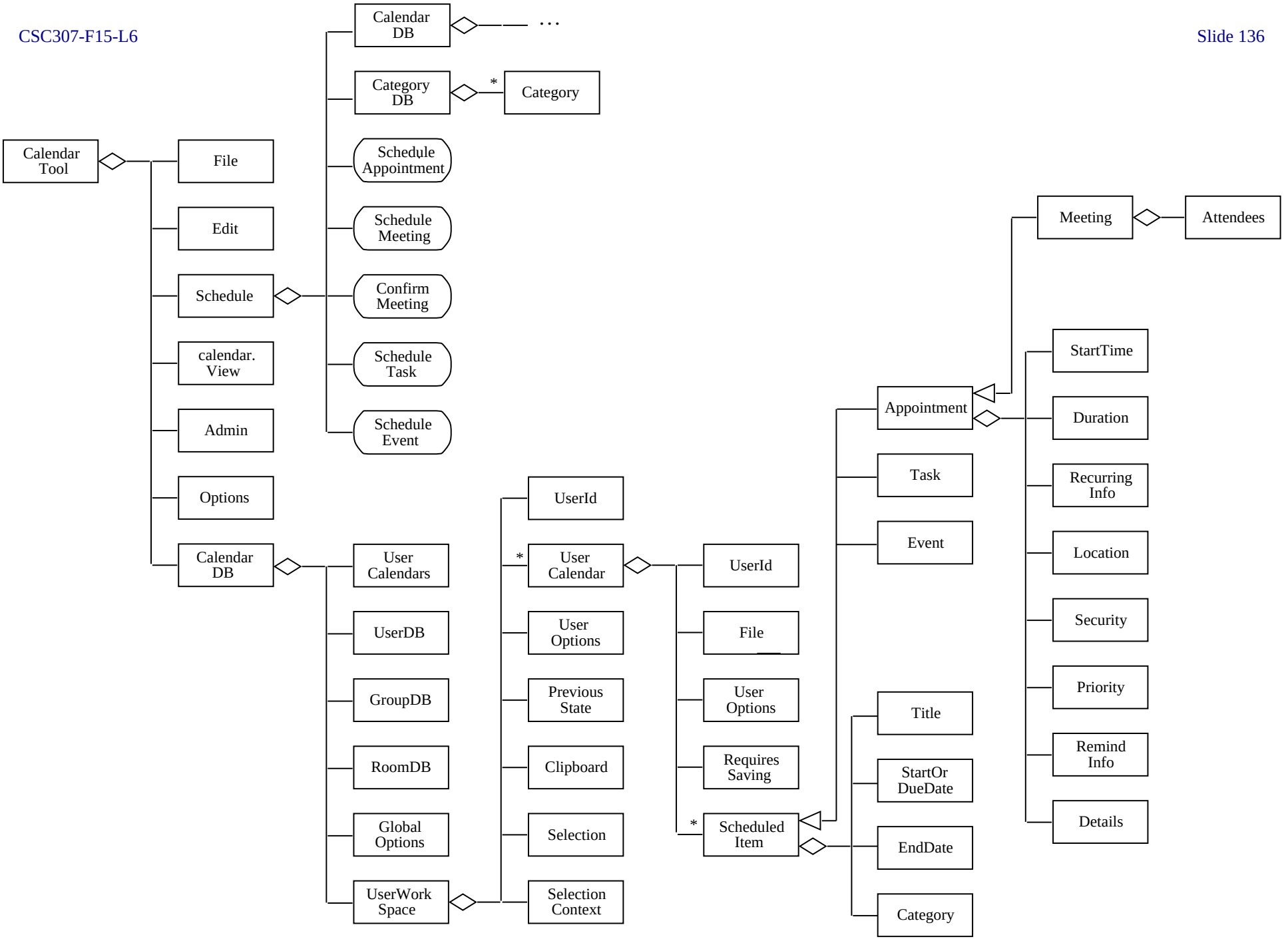
CalTool.java, cont'd

```
/****
 *
 * Class CalendarTool is ...
 *
 * @author Gene Fisher (gfisher@...
 * @version 9nov15
 *
 */
```

CalTool.java, cont'd

```
public class CalendarTool extends Model {

    . . .

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

    . . .

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

}
```

XI. Class diagram for derived Calendar design:


Calendar DB
...
Category DB
*
Category
Calendar Tool
File
Edit
Schedule
calendar. View
Admin
Options
Calendar DB
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
Meeting
Attendees
Appointment
Task
Event
StartTime
Duration
Recurring Info
Location
Security
Priority
Remind Info
Details
Title
StartOr DueDate
EndDate
Category

Class diagram Calendar design, cont'd

See detailed notes for a bit more disucssion.

XII. Observations about Milestone 6 Design

XII. Observations about Milestone 6 Design

A. `CalendarDB.java`

XII. Observations about Milestone 6 Design

A. `CalendarDB.java`

1. Traceability to spec is quite direct.

XII. Observations about Milestone 6 Design

A. `CalendarDB.java`

1. Traceability to spec is quite direct.

2. This is a "managerial" class.

XII. Observations about Milestone 6 Design

A. `CalendarDB.java`

1. Traceability to spec is quite direct.
2. This is a "managerial" class.
3. Class comment derived from spec descrip.

XII. Observations about Milestone 6 Design

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`

B. `Schedule.java`

1. Another managerial class.

B. `Schedule.java`

1. Another managerial class.

2. Doesn’t appear in abstract model.

C. `ScheduledItem.java`

C. `ScheduledItem.java`

1. Traces directly to abstract object.

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`

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

1. Also trace to spec objects.

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

1. Also trace to spec objects.

2. Inheritance relationships retained.

E. Date.java

E. `Date.java`

1. Straightforward translation of spec object.

E. `Date.java`

1. Straightforward translation of spec object.

2. Likely replaced with Java lib class.

XIII. Program GUIs in Java Swing

XIII. Program GUIs in Java Swing

A. Swing is default for 307; your team may choose a different Java-based GUI library.

XIII. **Program GUIs in Java Swing**

A. Swing is default for 307; your team may choose a different Java-based GUI library.

B. Library package is `javax.swing`.

XIII. **Program GUIs in Java Swing**

A. Swing is default for 307; your team may choose a different Java-based GUI library.

B. Library package is `javax.swing`.

C. Key Swing classes:

`javax.swing`, cont'd

`javax.swing`, cont'd

- Box

javax.swing, cont'd

- Box
- JButton

javax.swing, cont'd

- Box
- JButton
- JComboBox

javax.swing, cont'd

- Box
- JButton
- JComboBox
- JLabel

javax.swing, cont'd

- Box
- JButton
- JComboBox
- JLabel
- JList

javax.swing, cont'd

- Box
- JButton
- JComboBox
- JLabel
- JList
- JMenu

javax.swing, cont'd

- Box
- JButton
- JComboBox
- JLabel
- JList
- JMenu
- JMenuBar

javax.swing, cont'd

- Box
- JButton
- JComboBox
- JLabel
- JList
- JMenu
- JMenuBar
- JMenuItem

javax.swing, cont'd

- Box
- JButton
- JComboBox
- JLabel
- JList
- JMenu
- JMenuBar
- JMenuItem
- JTabbedPane

javax.swing, cont'd

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

javax.swing, cont'd

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

`javax.swing`, cont'd

D. Selected subpackages:

`javax.swing`, cont'd

D. Selected subpackages:

- `swing.colorchooser`

`javax.swing`, cont'd

D. Selected subpackages:

- `swing.colorchooser`
- `swing.filechooser`

`javax.swing`, cont'd

D. Selected subpackages:

- `swing.colorchooser`
- `swing.filechooser`
- `swing.table`

`javax.swing`, cont'd

D. Selected subpackages:

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

`javax.swing`, cont'd

D. Selected subpackages:

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

`javax.swing`, cont'd

D. 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`.

XIV. Package **`java.awt`**

-- lower-level support for `swing`.

- `Color`

XIV. Package **`java.awt`**

-- lower-level support for `swing`.

- `Color`
- `Component`

XIV. Package **`java.awt`**

-- lower-level support for `swing`.

- `Color`
- `Component`
- `Event`

XIV. Package **`java.awt`**

-- lower-level support for `swing`.

- `Color`
- `Component`
- `Event`
- `Graphics2D`

XIV. Package **`java.awt`**

-- lower-level support for `swing`.

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

XIV. Package **`java.awt`**

-- lower-level support for `swing`.

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

XV. Designing GUIs with Swing.

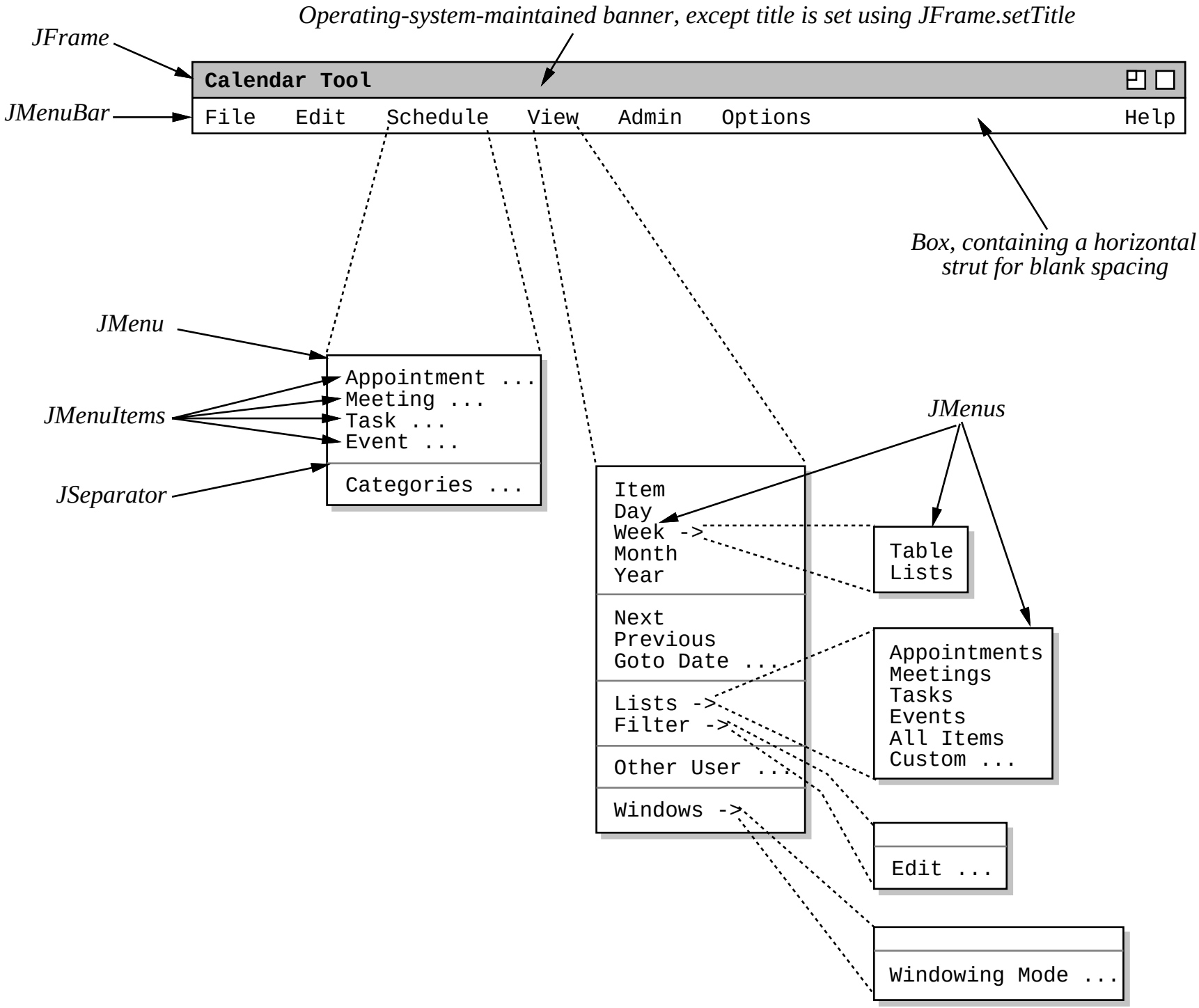
XV. Designing GUIs with Swing.

A. Above list used widely in 307.

XV. Designing GUIs with Swing.

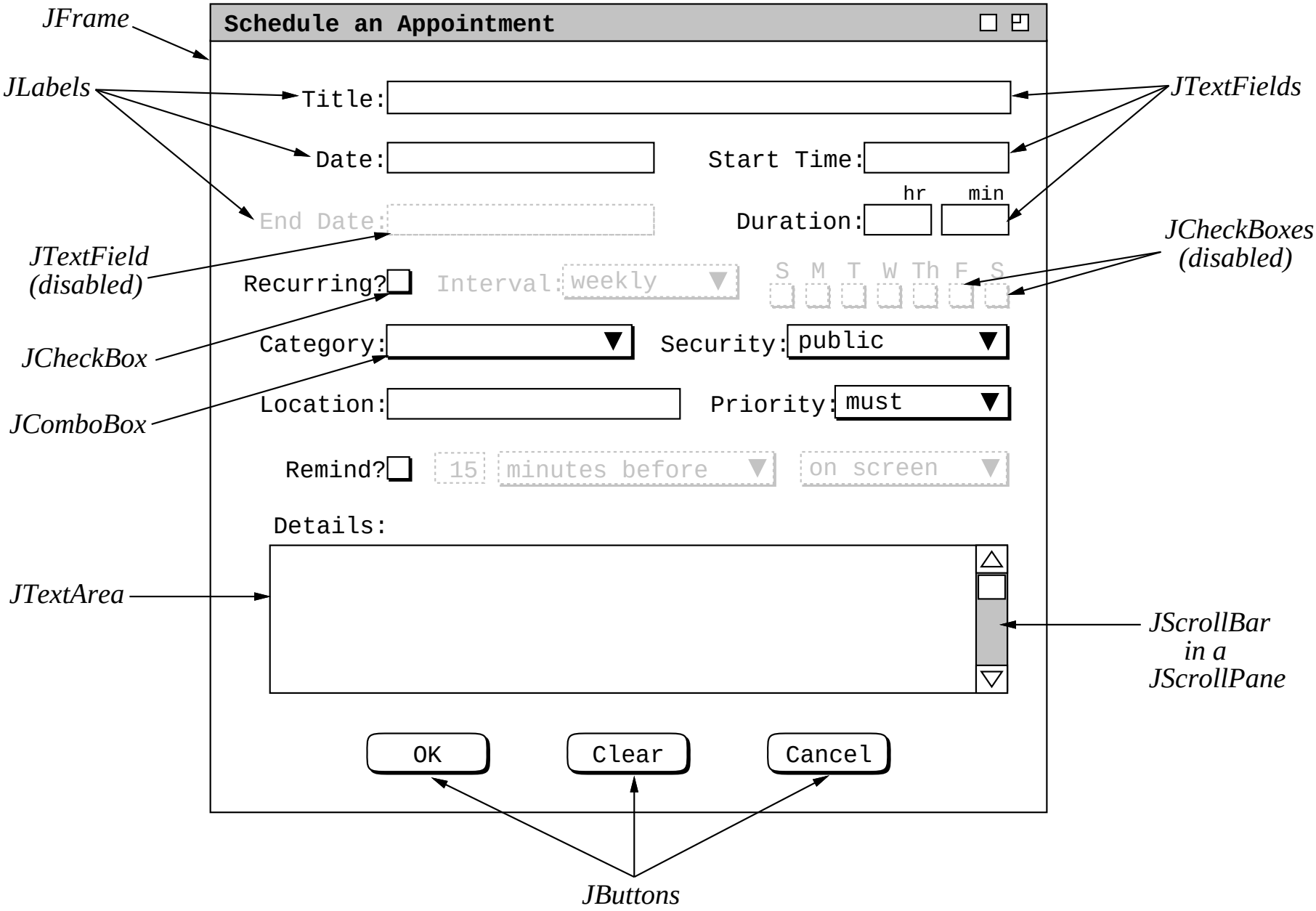
A. Above list used widely in 307.

B. Figure 7 shows a typical menubar.


Operating-system-maintained banner, except title is set using JFrame.setTitle
JFrame
Calendar Tool
JMenuBar
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 ...

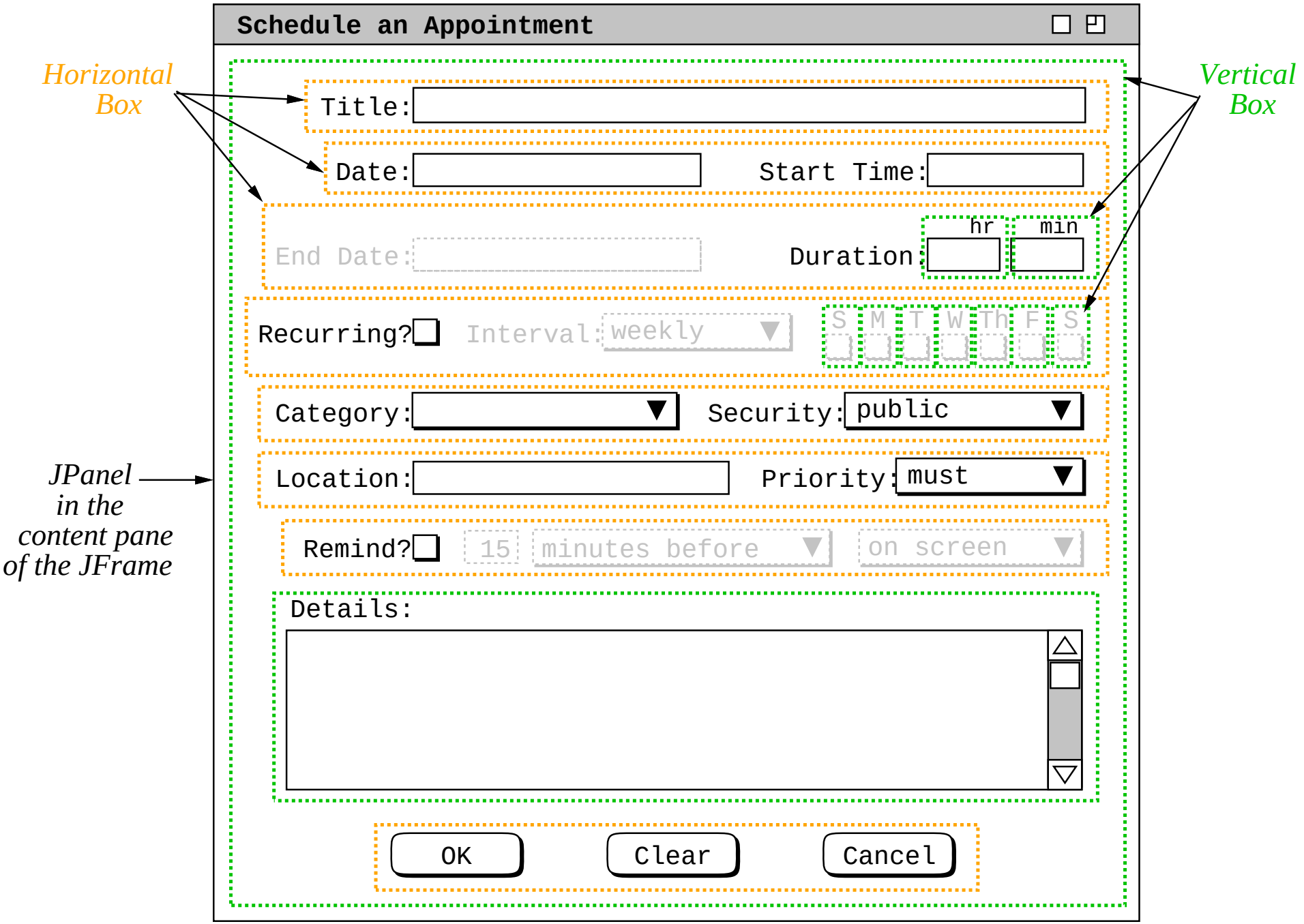
Swing, cont'd

C. Figure 8 shows typical dialog.


JFrame
JLabels
JTextFields
JTextField (disabled)
JCheckBoxes (disabled)
JCheckBox
JComboBox
JTextArea
JScrollBar in a JScrollPane
JButtons
Schedule an Appointment
Title:
Date:
Start Time:
hr
min
End Date:
Duration:
Recurring?
Interval:
weekly
S M T W Th F S
Category:
Security:
public
Location:
Priority:
must
Remind?
15
minutes before
on screen
Details:
OK
Clear
Cancel

Swing, cont'd

D. Figure 9 dialog layout with Boxes


Schedule an Appointment
Horizontal Box
Vertical Box
JPanel in the content pane of the JFrame
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:
OK
Clear
Cancel

XVI. View class naming conventions.

XVI. View class naming conventions.

A. Standard name suffixes.

XVI. View class naming conventions.

A. Standard name suffixes.

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

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 and View classes.

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

A. Parallel decomposition.

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

A. Parallel decomposition.

1. `Model`, `View` classes top inheritance hierarchy.

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

A. Parallel decomposition.

1. `Model`, `View` classes top inheritance hierarchy.

2. Tool-specific model, view classes extend these.

M/V Coordination, cont'd

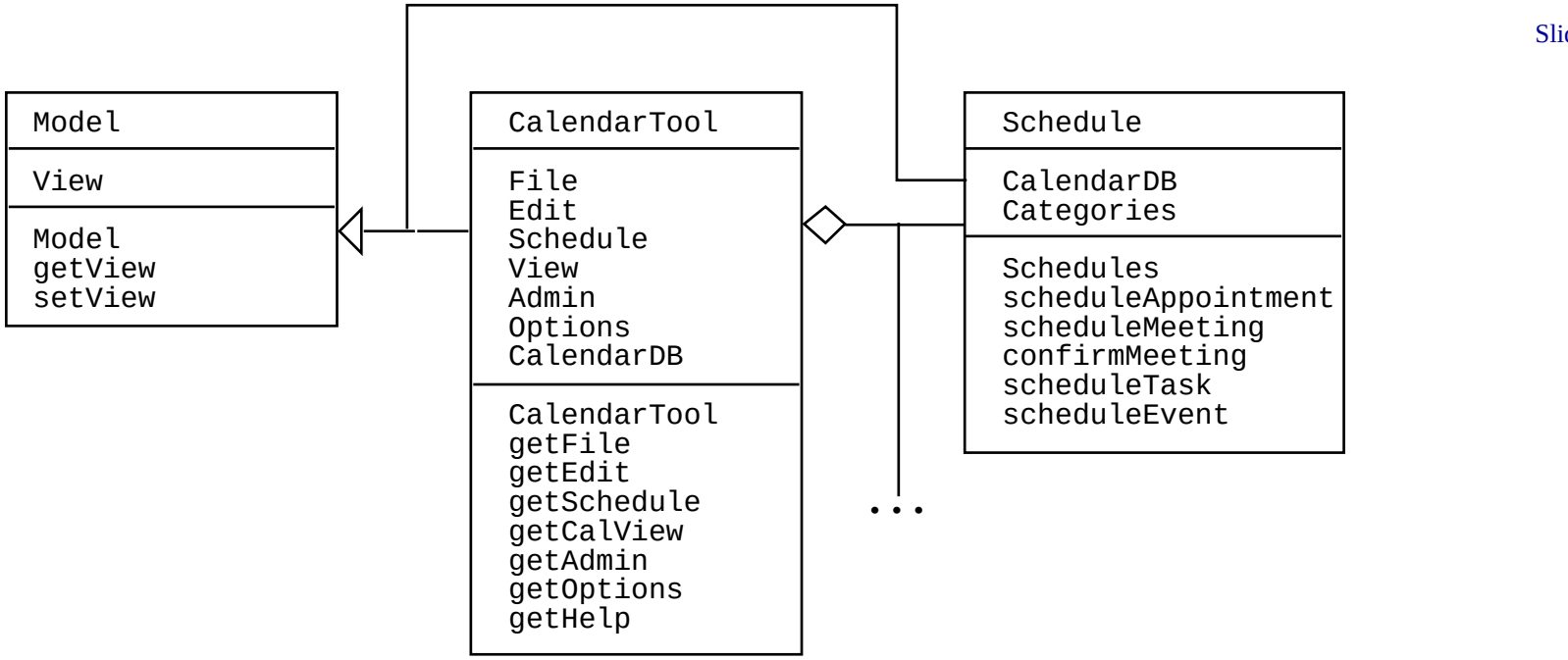
B. High-level class decomposition follows *functional hierarchy* of the requirements.

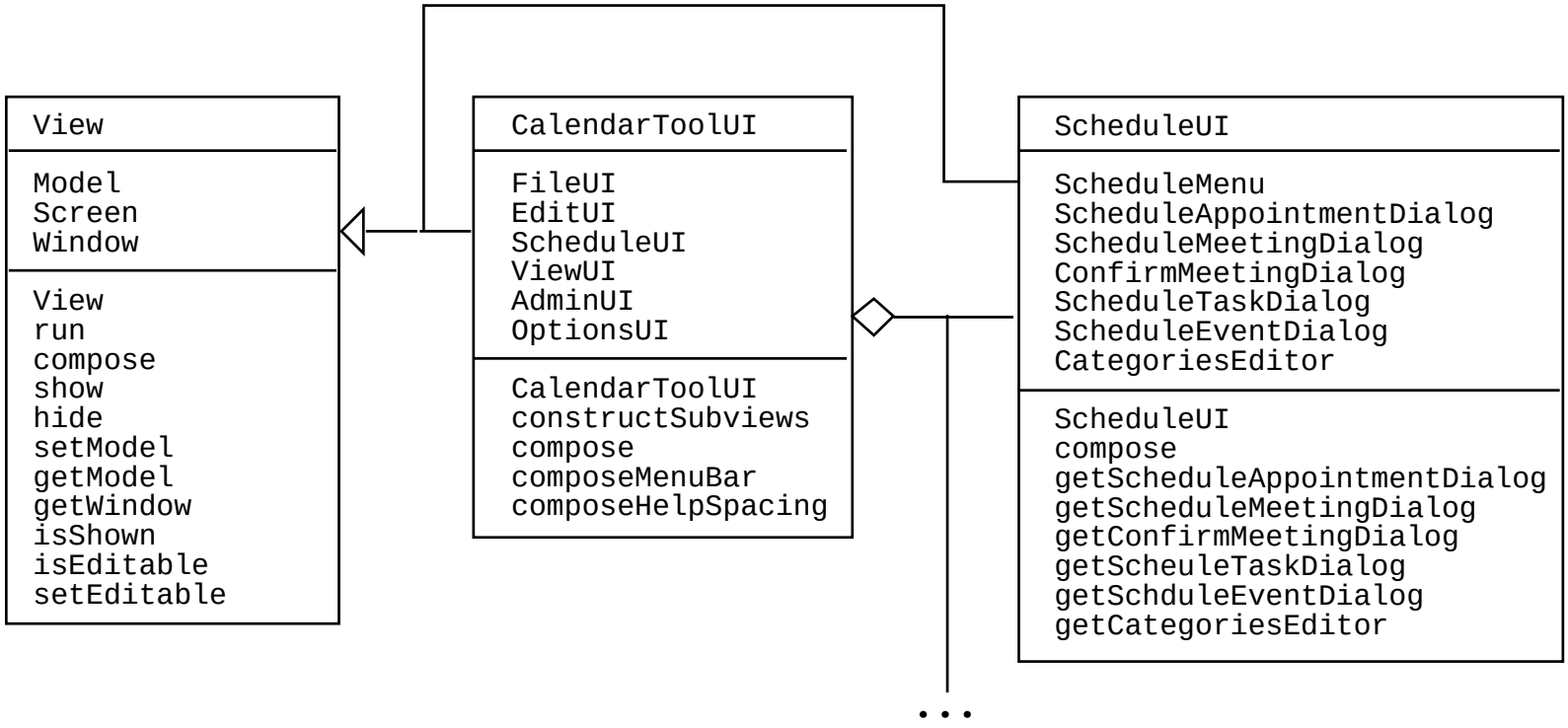
M/V Coordination, cont'd

B. High-level class decomposition follows *functional hierarchy* of the requirements.

C. What's important is that the functional hierarchy ***makes sense***, both in requirements and design.

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`,`View` at root of the inheritance hierarchy.

M/V class diagram, cont'd

A. `Model`,`View` at root of the inheritance hierarchy.

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

M/V class diagram, cont'd

A. `Model`,`View` at root of the inheritance hierarchy.

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

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

M/V class diagram, cont'd

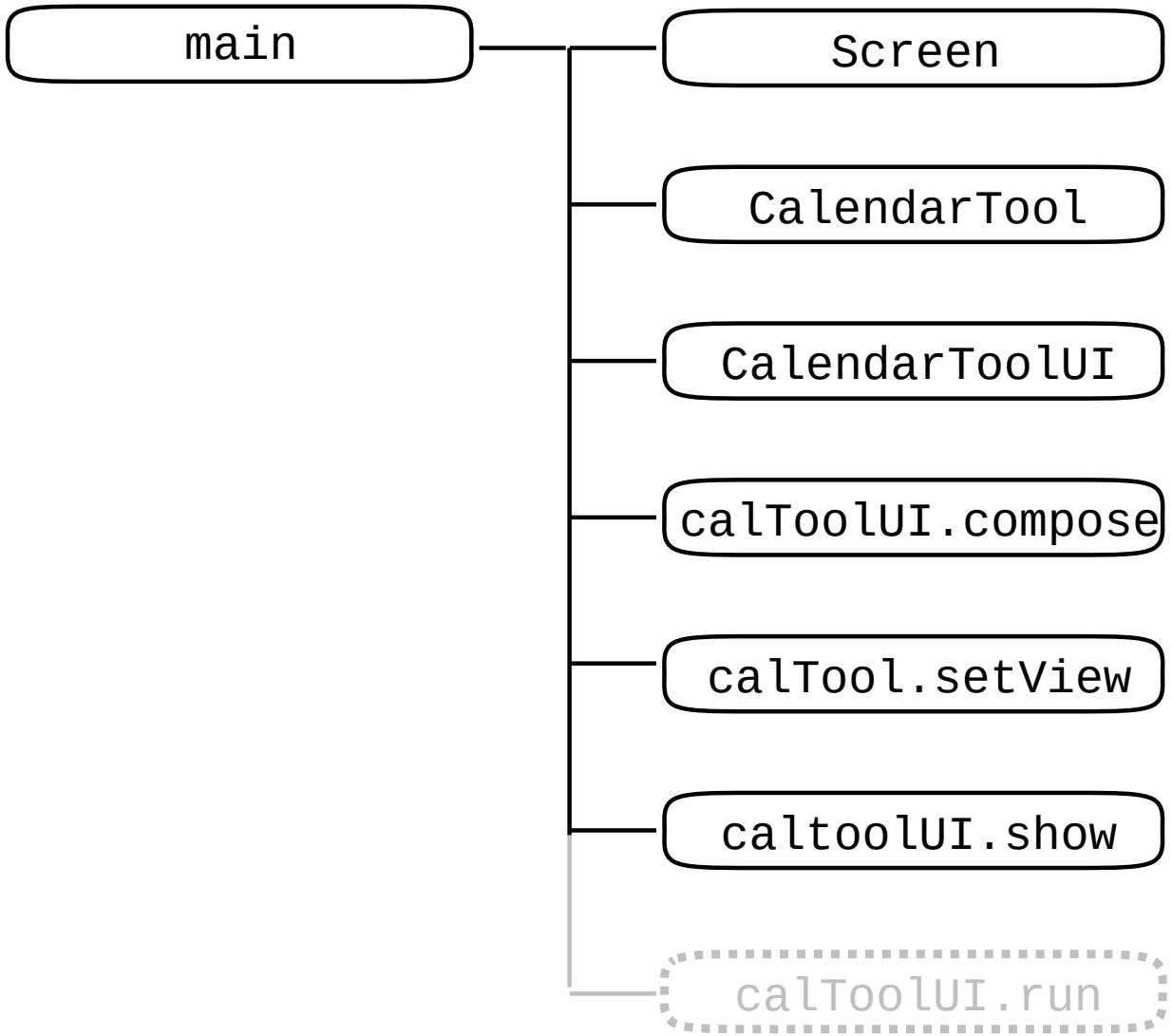
A. `Model`,`View` at root of the inheritance hierarchy.

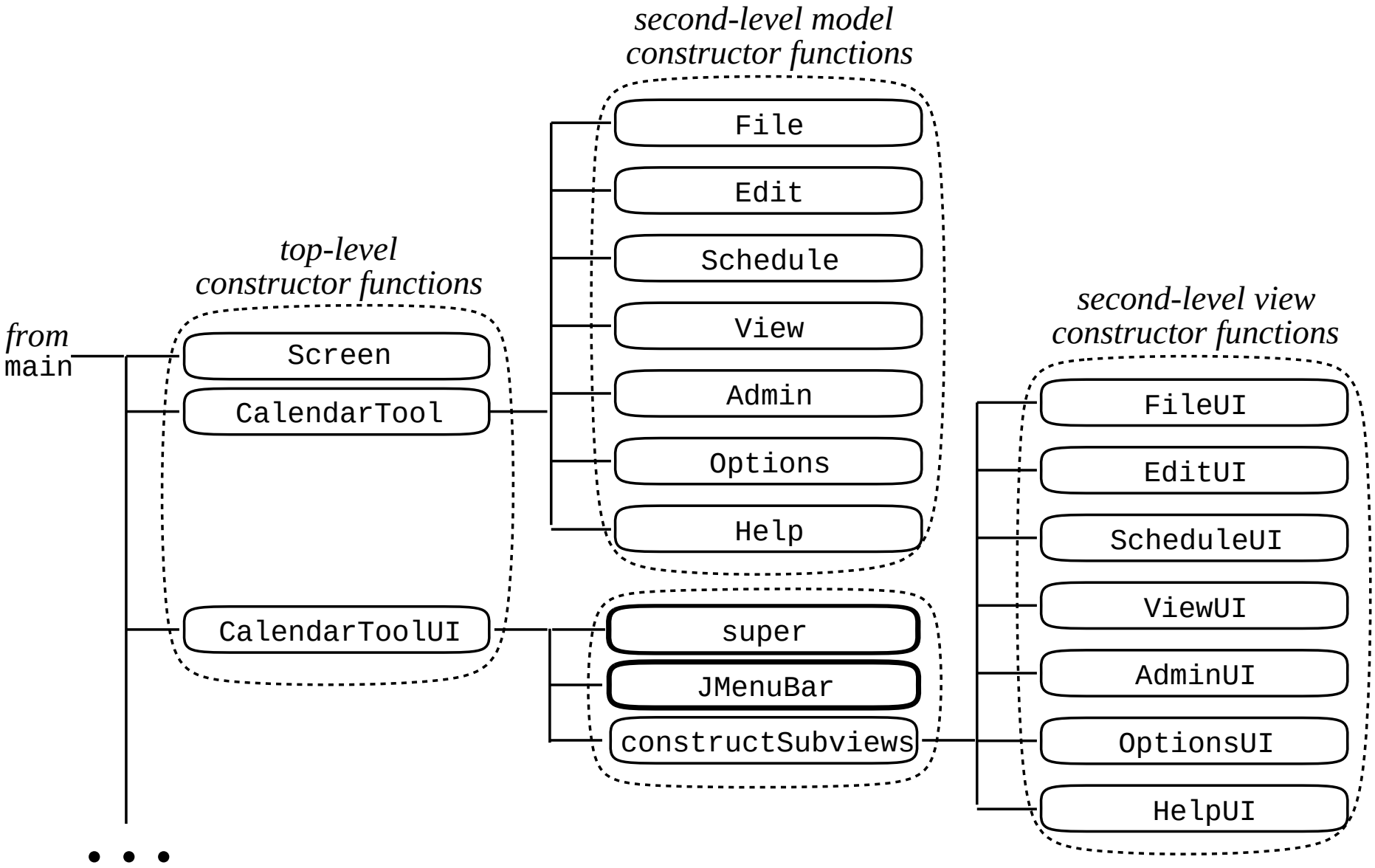
B. Inheriting from these are top-level model and view classes.

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

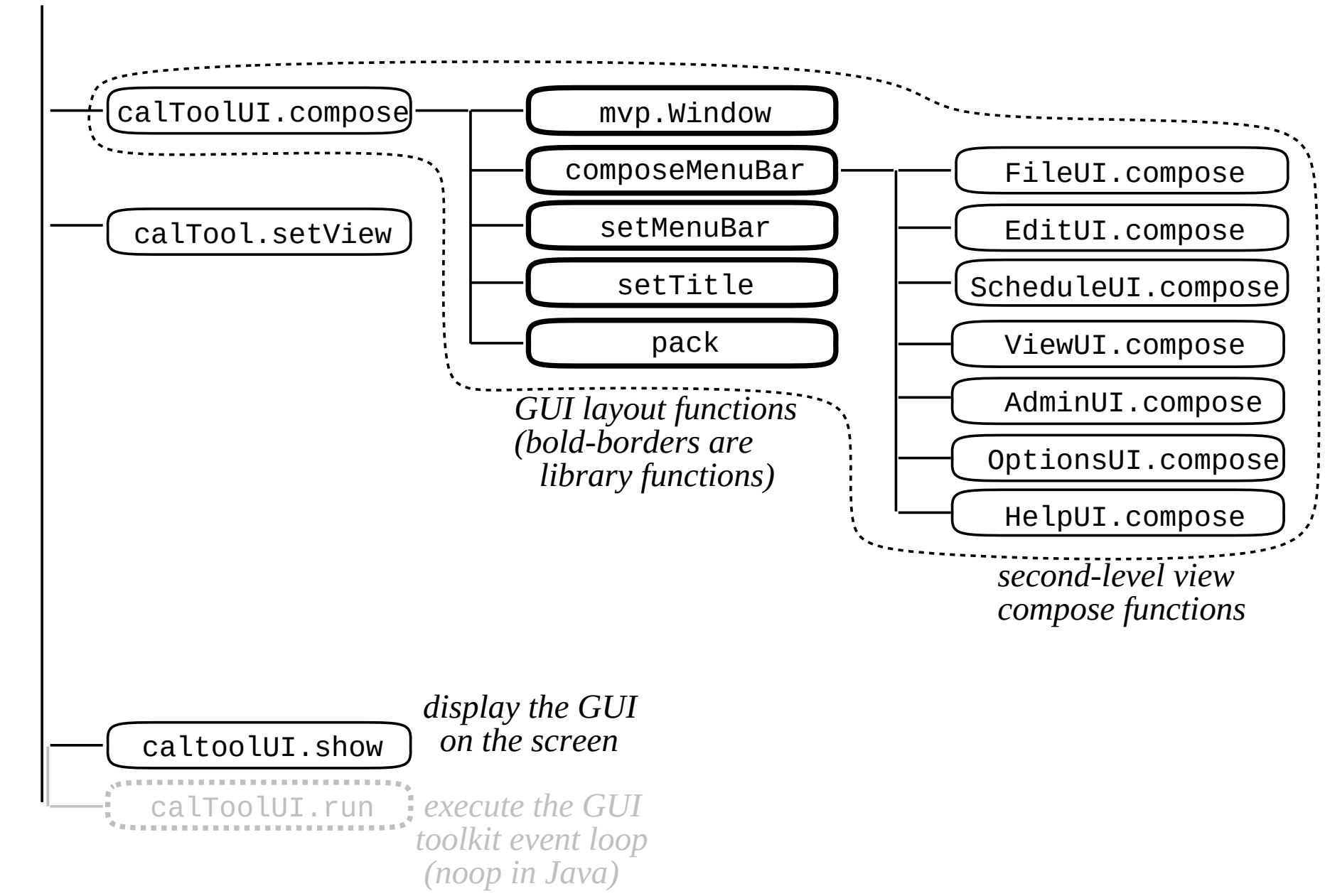
D. These also inherit from `Model` and `View`.

XIX. High-level Model/View 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
calTool.setView
GUI layout functions
(bold-borders are
library functions)
second-level view
compose functions
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.

High-level function diagram, cont'd

A. First three calls are top-level constructors.

1. `Screen` initializes GUI.

High-level function diagram, cont'd

A. First three calls are top-level constructors.

1. `Screen` initializes GUI.

2. `CalendarTool` constructs models.

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.

Function diagram, cont'd

B. `CalendarToolUI.compose` performs UI layout.

1. Subfunctions layout various UI pieces.

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 307 libraries.

Function diagram, cont'd

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

Function diagram, cont'd

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

1. Model and view mutually refer.

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.

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.

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.

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.

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.

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 event handling loop.

High-level function diagram, cont'd

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

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.

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

XX. Overview of Event-Based Design

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

XX. Overview of Event-Based Design

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

1. In Swing, a Java separate thread -- `java.awt.EventDispatchThread`

XX. Overview of Event-Based Design

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

1. In Swing, a Java separate thread -- `java.awt.EventDispatchThread`

2. `MainThread` terminated.

XX. Overview of Event-Based Design

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

1. In Swing, a Java separate thread -- `java.awt.EventDispatchThread`

2. `MainThread` terminated.

3. Application methods invoked via *events*.

Overview, cont'd

B. Event-based processing is in all GUI toolkits.

Overview, cont'd

B. Event-based processing is in all GUI toolkits.

1. Details vary widely.

Overview, cont'd

B. Event-based processing is in all GUI toolkits.

1. Details vary widely.

2. Each has an *event model*.

Overview, cont'd

B. Event-based processing is 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

XXI. Designing event-based programs

A. Two important aspects:

XXI. Designing event-based programs

A. Two important aspects:

1. Setting up event handlers.

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.

Designing event-based, cont'd

B. Event handlers respond to events.

1. Events are user actions.

Designing event-based, cont'd

B. Event handlers respond to events.

1. Events are user actions.

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

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.

Designing event-based, cont'd

C. Event handler invokes an application method.

1. Event-invoked methods are "*call-back*s".

Designing event-based, cont'd

C. Event handler invokes an application method.

1. Event-invoked methods are "*call-back*s".

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

Designing event-based, cont'd

3. `actionPerformed` specialized per listener.

Designing event-based, cont'd

3. *`actionPerformed`* specialized per listener.

4. Each specialized `actionPerformed` calls an appropriate model method.

XXII. Design diagram notation

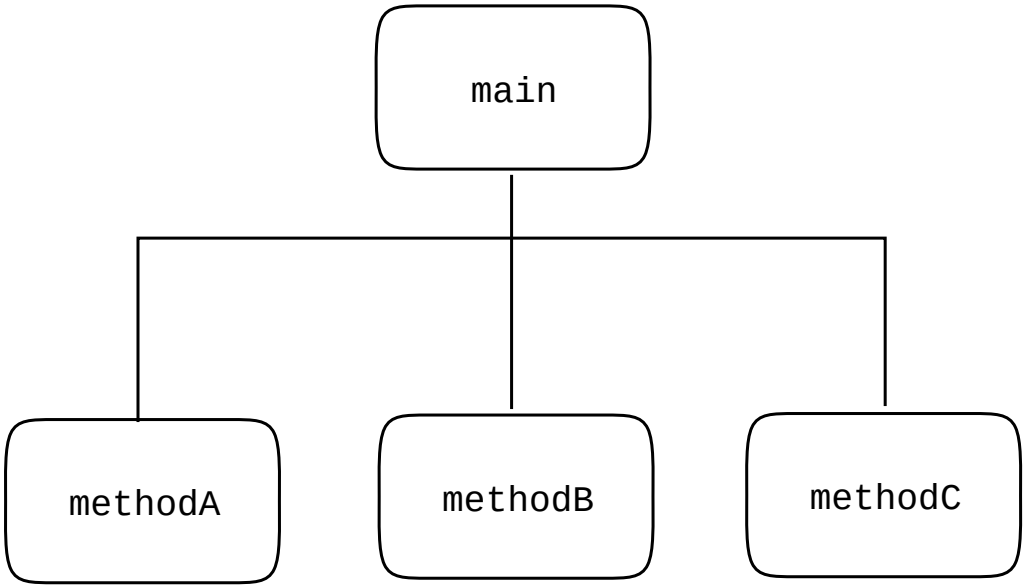
XXII. Design diagram notation

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

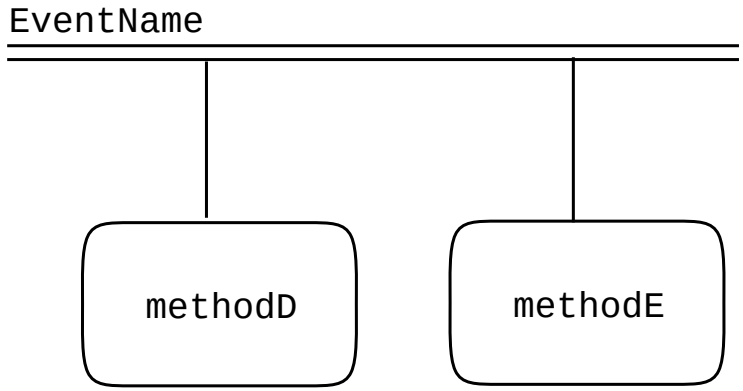
XXII. Design diagram notation

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

B. See Figure 12



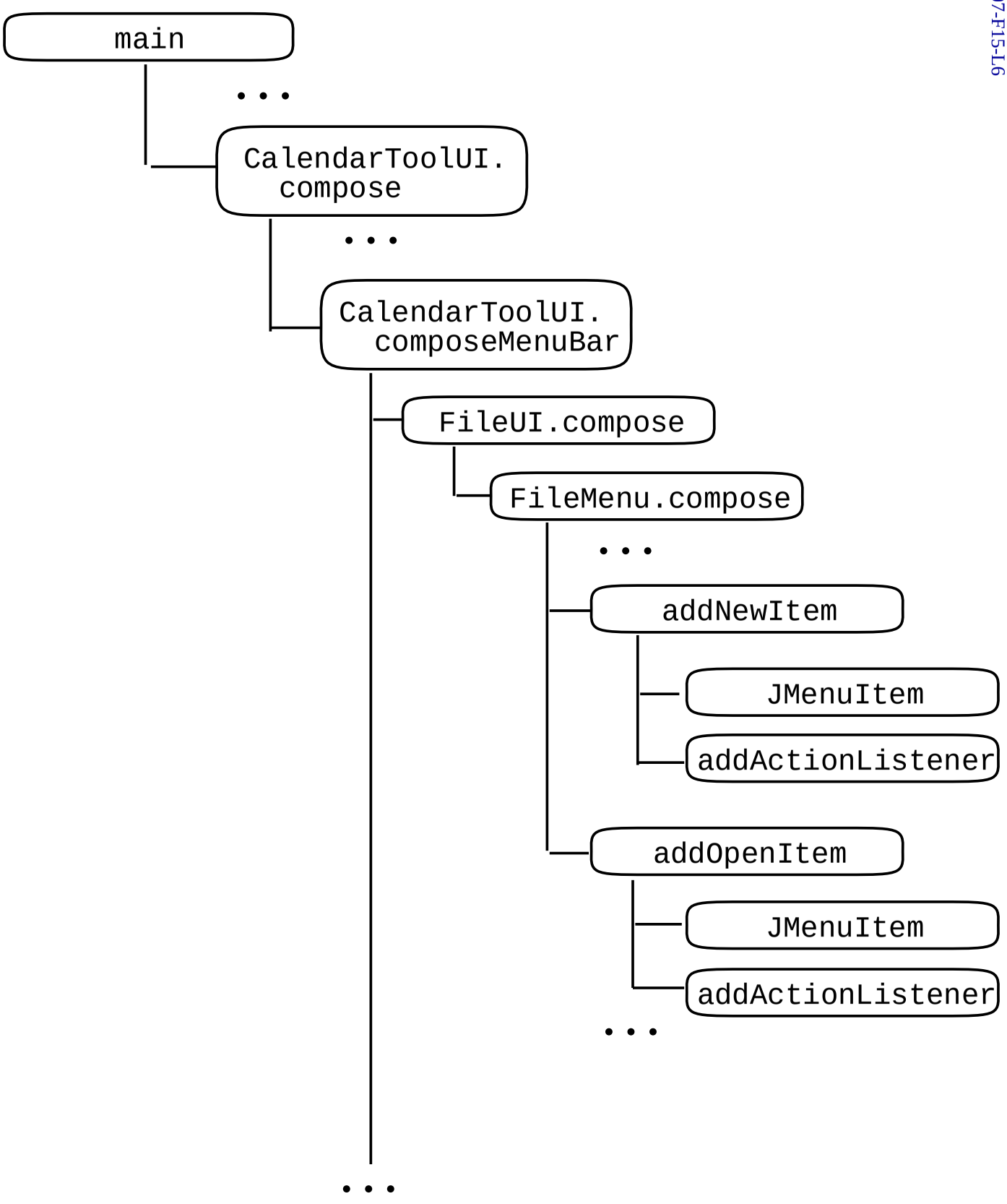
a. Normal method invocation



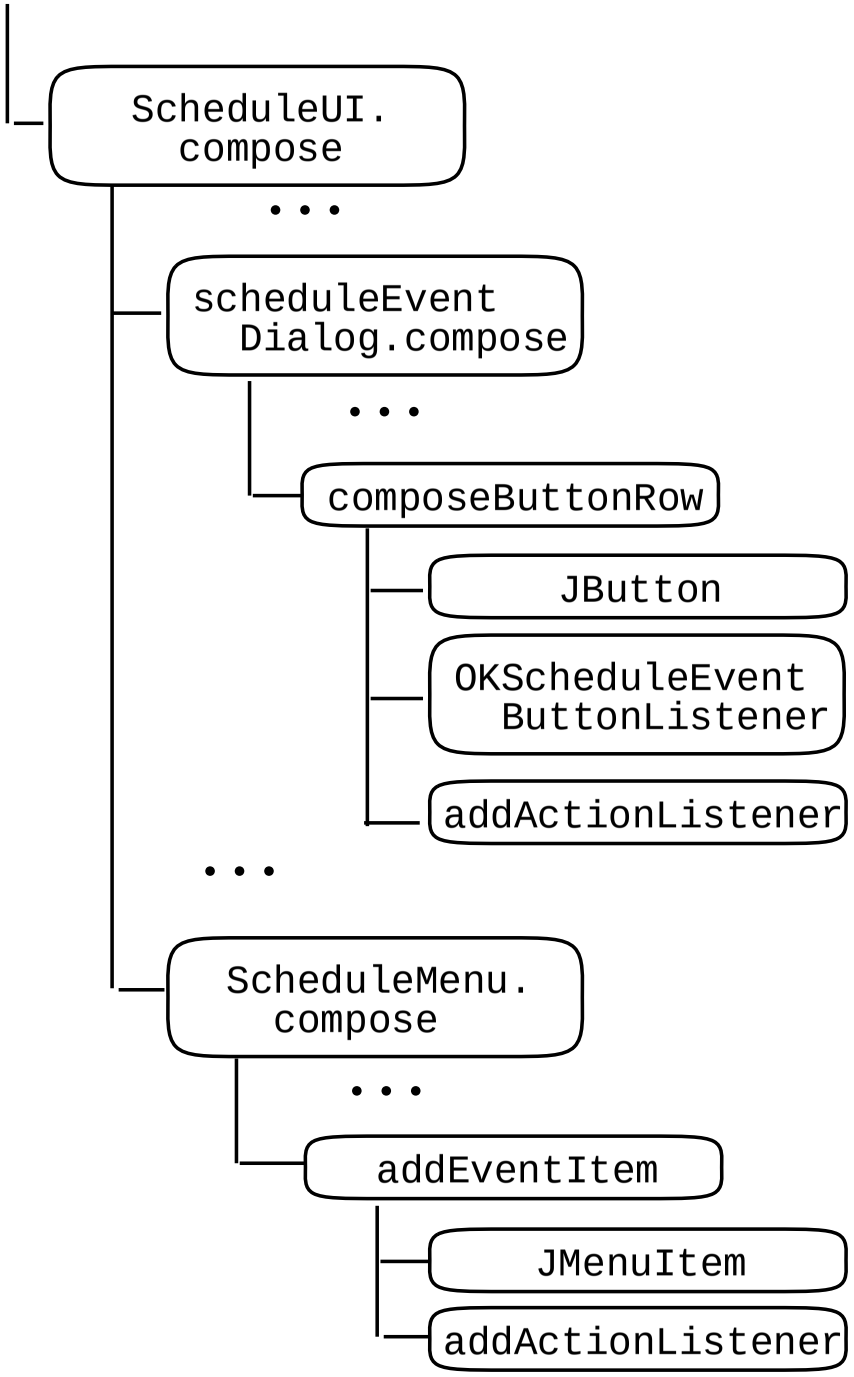
b. Event-based method invocation

XXIII. **Examples**

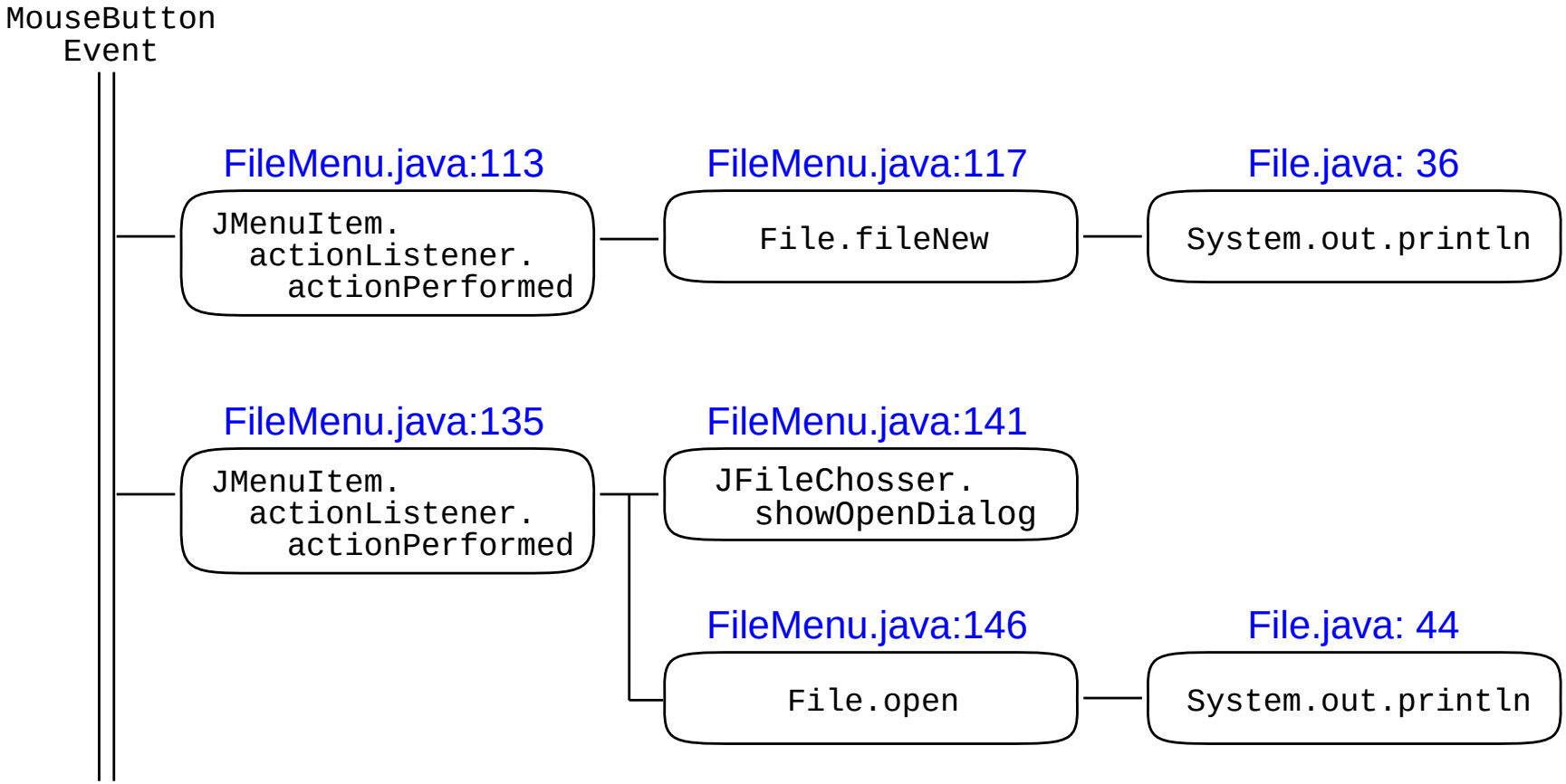
A. Figure 13 shows setting up event handlers.

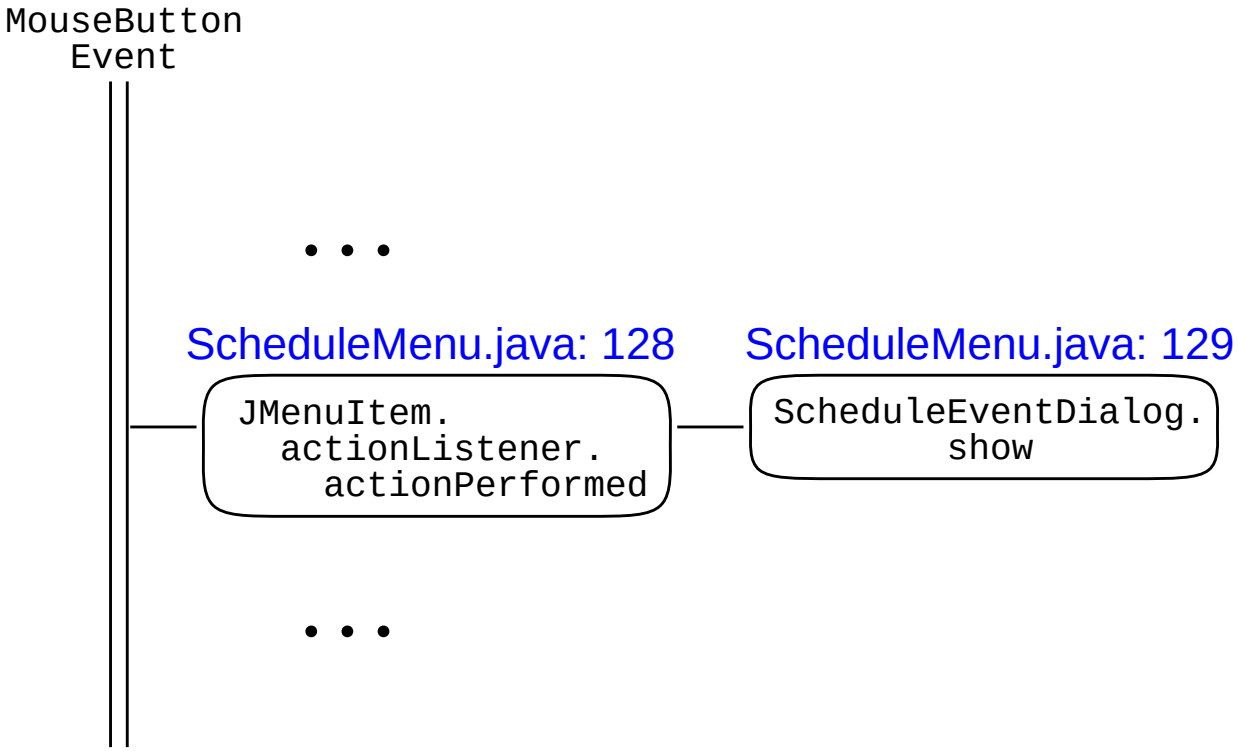

main
...
CalendarToolUI.
compose
...
CalendarToolUI.
composeMenuBar
FileUI.compose
FileMenu.compose
...
addNewItem
JMenuItem
addActionListener
addOpenItem
JMenuItem
addActionListener
...
...

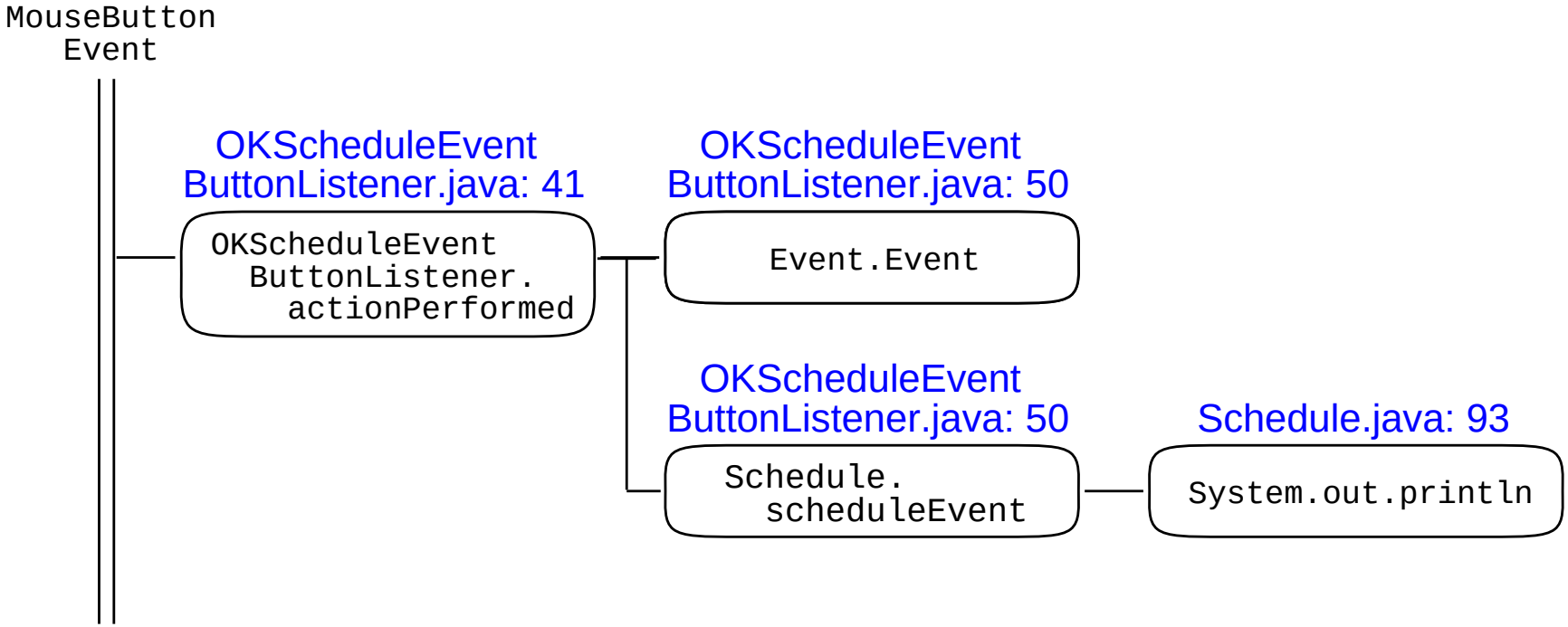
from CalendarToolUI.composeMenuBar

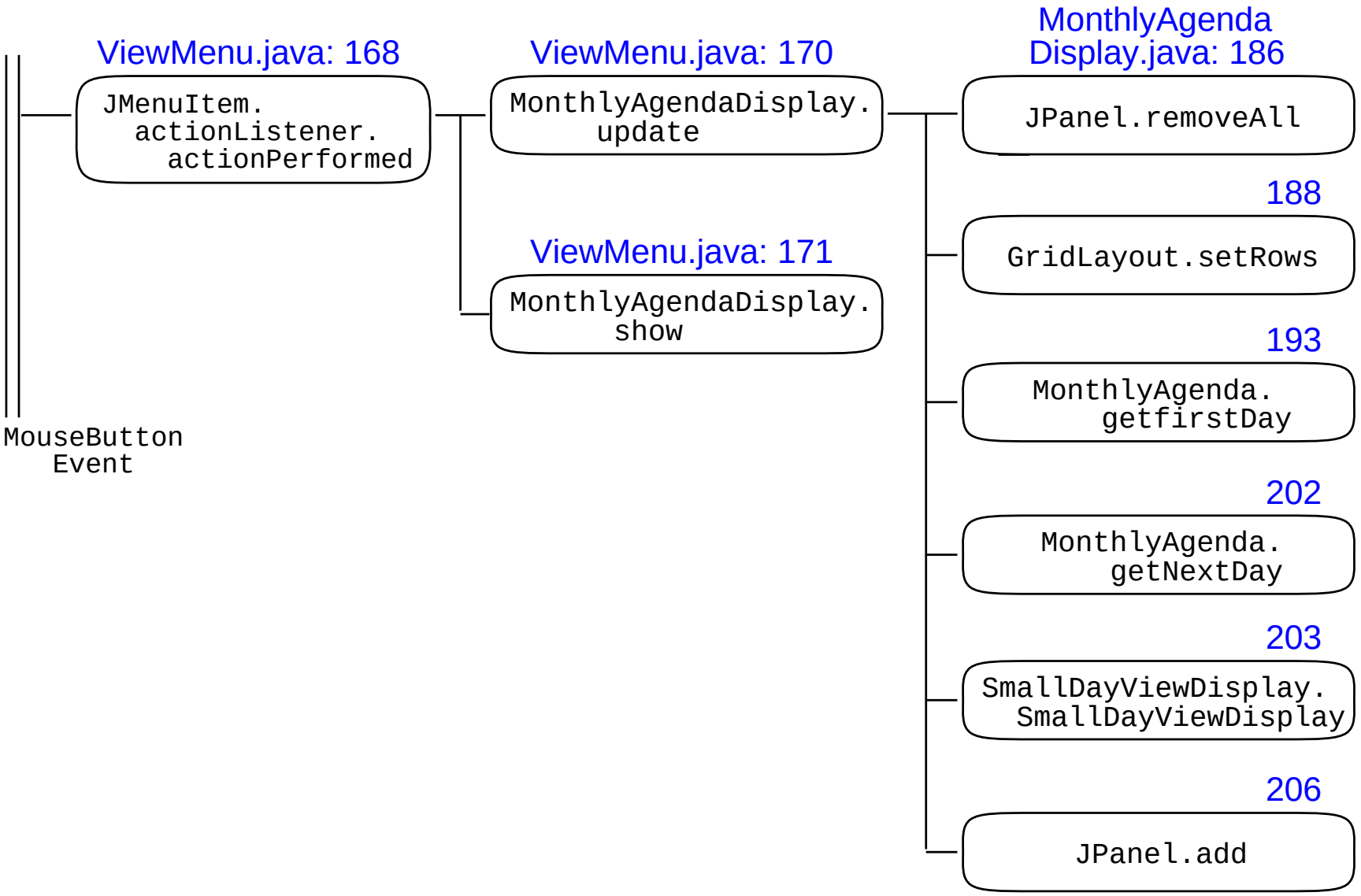


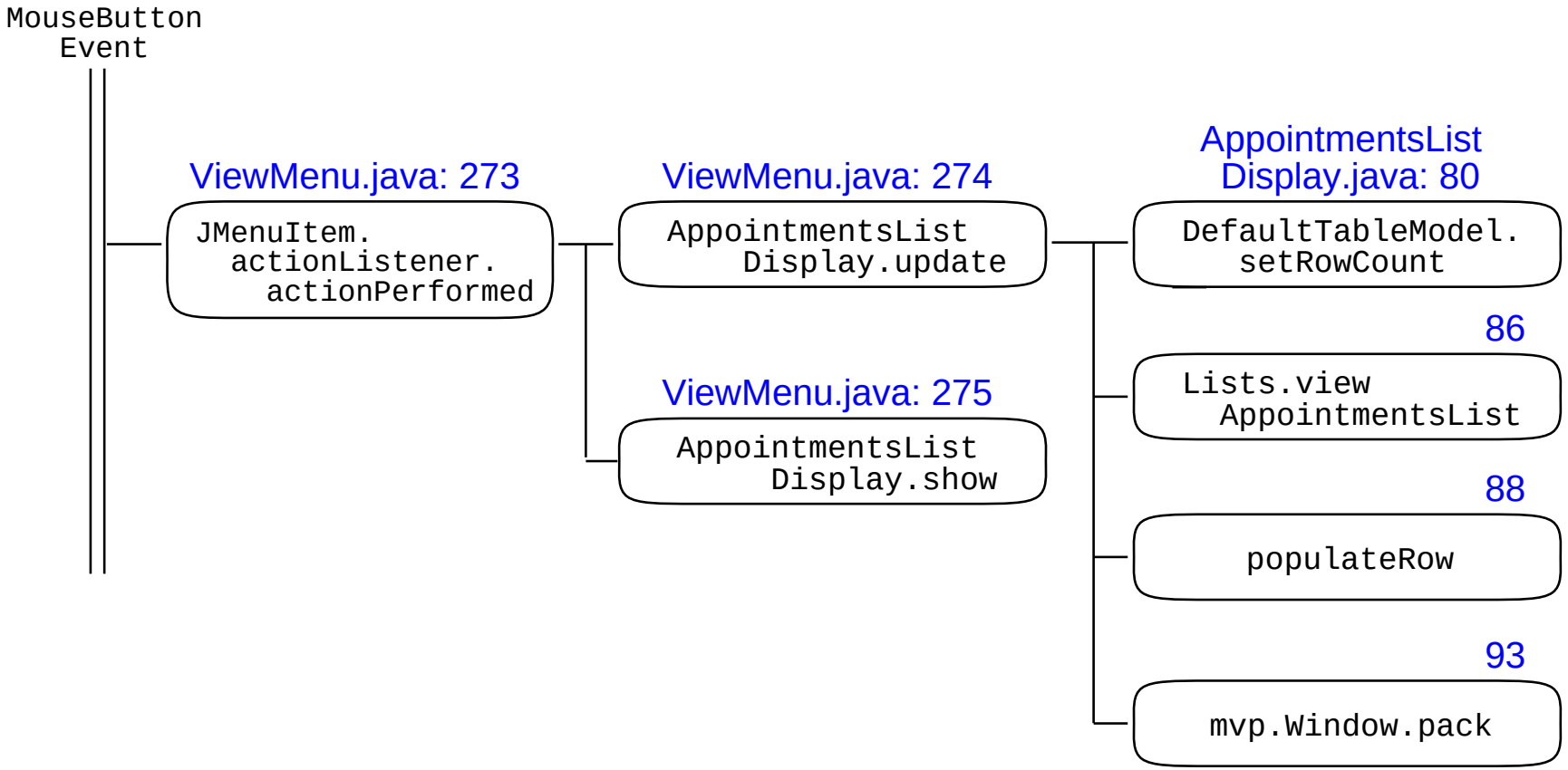
B. Figure 14 shows event-based invocation.


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


MouseButton
Event
ScheduleMenu.java: 128
ScheduleMenu.java: 129
JMenuItem.
actionListener.
actionPerformed
ScheduleEventDialog.
show


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


ViewMenu.java: 168
ViewMenu.java: 170
MonthlyAgenda Display.java: 186
JMenuItem. actionListener. actionPerformed
MonthlyAgendaDisplay. update
JPanel.removeAll
188
GridLayout.setRows
ViewMenu.java: 171
MonthlyAgendaDisplay. show
193
MonthlyAgenda. getfirstDay
202
MonthlyAgenda. getNextDay
203
SmallDayViewDisplay. SmallDayViewDisplay
206
JPanel.add
MouseButton Event


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