CSC 308 Lecture Notes Week 5

Details of Requirements Model Derivation and Refinement

I. Administrative matters.

I. Administrative matters.

A. Modeling for Milestone 4.

I. Administrative matters.

A. Modeling for Milestone 4.

1. See M4 example for guide of how much to do.

I. Administrative matters.

A. Modeling for Milestone 4.

1. See M4 example for guide of how much to do.

   a. Each team member must commit at least six model classes.

I. Administrative matters.

A. Modeling for Milestone 4.

1. See M4 example for guide of how much to do.

   a. Each team member must commit at least six model classes.

   b. Classes can be in one or more `.java` files.

I. Administrative matters.

A. Modeling for Milestone 4.

1. See M4 example for guide of how much to do.

   a. Each team member must commit at least six model classes.

   b. Classes can be in one or more `.java` files.

   c. Team coordination for major shared objects and package structure.

Milestone 4, cont’d

2. Create package sub-directories under `specification` directory.

Milestone 4, cont’d

2. Create package sub-directories under `specification` directory.

3. Put `.java` files in appropriate package dirs.

Milestone 4, cont’d

2. Create package sub-directories under `specification` directory.

3. Put `.java` files in appropriate package dirs.

4. ***The files must compile with `javac`.***

Milestone 4, cont'd

2. Create package sub-directories under `specification` directory.

3. Put `.java` files in appropriate package dirs.

4. ***The files must compile with `javac`.***

5. ***Javadoc must be generated with `javadoc`.***

Milestone 4, cont’d

B. Remember, 1st round of inspection testing.

Milestone 4, cont’d

B. Remember, 1st round of inspection testing.

1. Review procedure in the SOP Vol. 2.

Milestone 4, cont'd

B. Remember, 1st round of inspection testing.

1. Review procedure in the SOP Vol. 2.

2. Decide as team time of pre-testing check-in, so librarian can release by 11:59PM Fri.

II. Guidelines for modularizing a model.

II. Guidelines for modularizing a model.

A. To *modularize* means subdivide into independent units.

II. Guidelines for modularizing a model.

A. To *modularize* means subdivide into independent units.

B. Dictionary definition --

> *"... an independent unit that can be used to construct a more complex structure".*

Modularization, cont’d

C. In Java, modules defined as `packages`.

Modularization, cont’d

C. In Java, modules defined as `packages`.

D. Good heuristic uses large-grain UI structure.

Modularization, cont'd

C. In Java, modules defined as `packages`.

D. Good heuristic uses large-grain UI structure.

1. Each menu in a menu-based UI is a module.

Modularization, cont'd

C. In Java, modules defined as `packages`.

D. Good heuristic uses large-grain UI structure.

1. Each menu in a menu-based UI is a module.

2. Similarly, top-level UI toolbars can be considered modules.

Modularization, cont’d

E. Given these heuristics, packaging structure of Calendar Tool can look like this:

Modularization, cont’d

E. Given these heuristics, packaging structure of Calendar Tool can look like this:

```
package file;
package edit;
package schedule;
package view;
package admin;
package options;
```

Modularization, cont’d

F. Within each package are appropriate classes.

Modularization, cont'd

F. Within each package are appropriate classes.

1. For Cal Tool focus is `schedule` and `view`.

Modularization, cont'd

F. Within each package are appropriate classes.

1. For Cal Tool focus is `schedule` and `view`.

2. Packaging structure is easy to view in `javadoc` form.

Modularization, cont'd

F. Within each package are appropriate classes.

1. For Cal Tool focus is `schedule` and `view`.

2. Packaging structure is easy to view in `javadoc` form.

3. Class-level `Javadoc` required for Milestone 4.

III. Summary of core steps of modeling

See online lecture notes for details

IV. Specific modeling guidelines.

IV. Specific modeling guidelines.

A. Object and operation naming.

IV. Specific modeling guidelines.

A. Object and operation naming.

1. Derive names directly from requirements.

IV. Specific modeling guidelines.

A. Object and operation naming.

1. Derive names directly from requirements.

2. Title bar contains object name

IV. Specific modeling guidelines.

A. Object and operation naming.

1. Derive names directly from requirements.
2. Title bar contains object name
3. Dialog labels are component names.

Specific guidelines, cont’d

4. Menu item or button name is op name.

Specific guidelines, cont’d

4. Menu item or button name is op name.

5. Punctuation must be removed

Specific guidelines, cont’d

4. Menu item or button name is op name.

5. Punctuation must be removed
   - otherwise retain full spelling

Specific guidelines, cont’d

4. Menu item or button name is op name.

5. Punctuation must be removed
   - otherwise retain full spelling
   - lower case first letter for Java convention

Specific guidelines, cont’d

B. Inheritance.

Specific guidelines, cont'd

B. Inheritance.

1. Derive as perceptible by user.

Specific guidelines, cont'd

B. Inheritance.

1. Derive as perceptible by user.

2. Not for representational efficiency.

Specific guidelines, cont’d

B. Inheritance.

1. Derive as perceptible by user.

2. Not for representational efficiency.

3. Prime directive of modeling
   -- ***"If the user perceives it, model it".***

V. Details of object derivation.

A. Clearly defined UI =>
straightforward object derivation.

B. Table from last week summarizes.

Java Type	**Common Interface Form**
int	string editor, slider, dial
double	same as integer
String	string editor, combo box
boolean	string editor, on/off button
data field	box containing other types
eunm	radio buttons; fixed-length list
Collection or List	variable-length list

Method	push button or menu item

VI. Details of operation derivation.

VI. Details of operation derivation.

A. The "..." suffix in a menu

VI. Details of operation derivation.

A. The "..." suffix in a menu

1. Input dialog, OK button.

VI. Details of operation derivation.

A. The "..." suffix in a menu

1. Input dialog, `OK` button.

a. Only one op to model.

VI. Details of operation derivation.

A. The "..." suffix in a menu

1. Input dialog, OK button.

a. Only one op to model.

b. Name derived from menu item.

VI. Details of operation derivation.

A. The "..." suffix in a menu

1. Input dialog, OK button.

   a. Only one op to model.

   b. Name derived from menu item.

   c. OK button itself *not* a separate op.

Operation derivation, cont’d

d. Three-phase GUI sequence for one op:

Operation derivation, cont’d

d. Three-phase GUI sequence for one op:

   i. Select it in menu.

Operation derivation, cont’d

d. Three-phase GUI sequence for one op:

   i. Select it in menu.

   ii. Fill in input dialog.

Operation derivation, cont’d

d. Three-phase GUI sequence for one op:

   i. Select it in menu.

   ii. Fill in input dialog.

   iii. Confirm or cancel.

Op derivation, cont’d

2. An alternative use of "..." leads to a multi-operation dialog.

Op derivation, cont’d

2. An alternative use of "..." leads to a multi-operation dialog.

    a. One op for each button or sub-menu item.

Op derivation, cont’d

2. An alternative use of "..." leads to a multi-operation dialog.

   a. One op for each button or sub-menu item.

   b. Menu itself derives module, not op.

Op derivation, cont’d

B. No "..." in menu item means

Op derivation, cont’d

B. No "..." in menu item means

1. there are no inputs, or

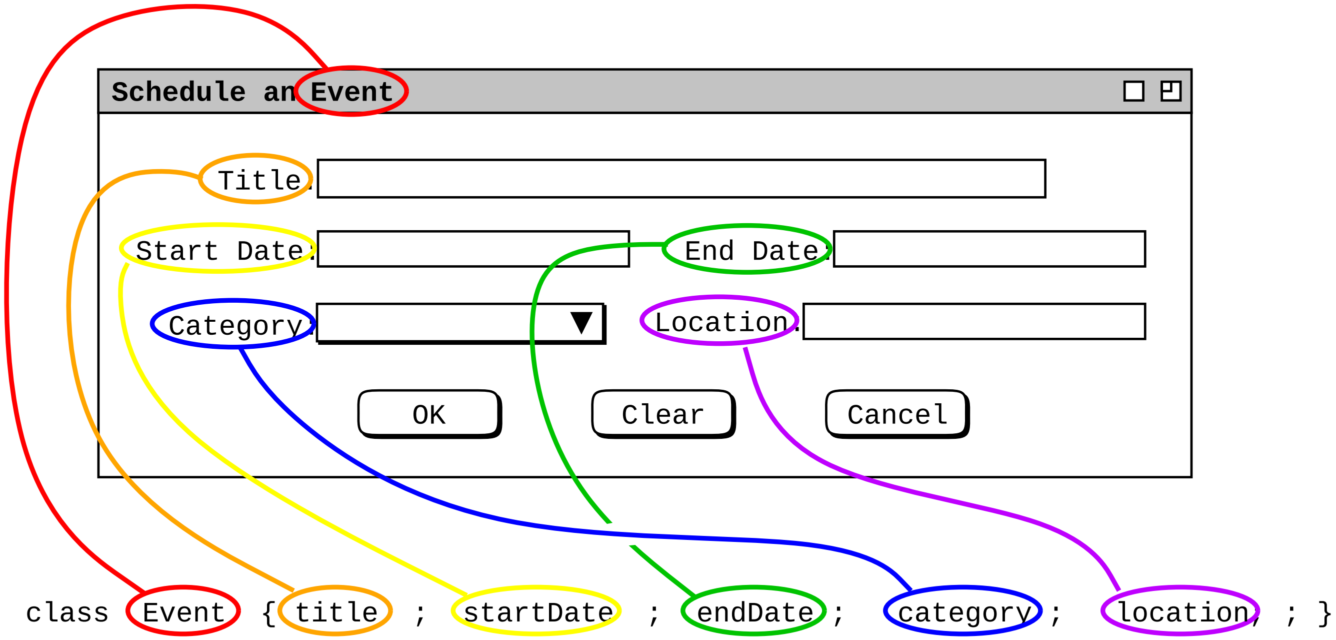
Op derivation, cont’d

B. No "..." in menu item means

1. there are no inputs, or
2. input(s) are default values from surrounding environment.

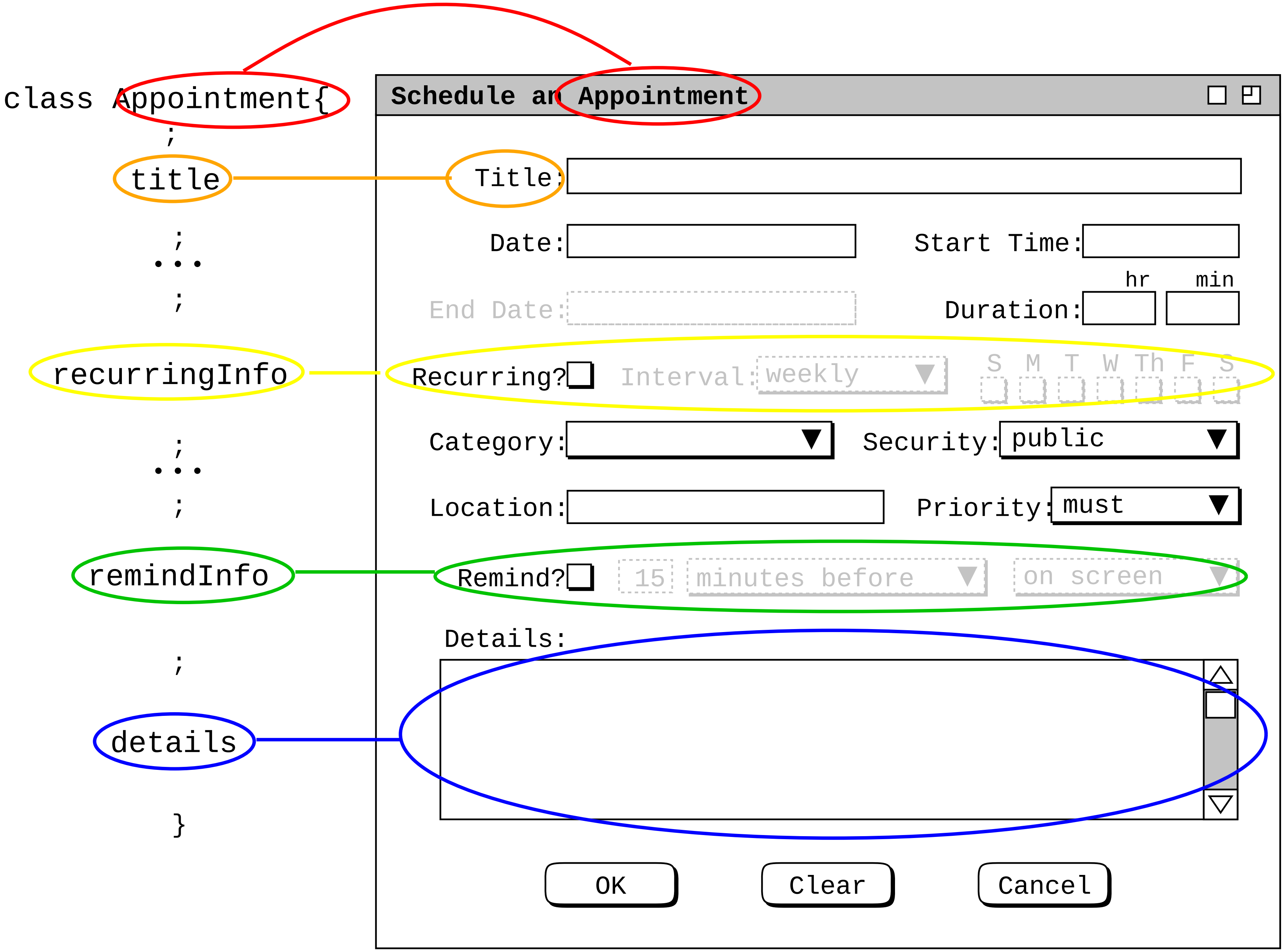
VII. Example -- scheduling.

A. Event (the simplest form of scheduled item).



```
class Event { title ; startDate ; endDate ; category ; location ; }
```

B. Appointment



```
class Appointment{
   ;
  title

   ;
  ...
   ;

recurringInfo

   ;
  ...
   ;

 remindInfo

   ;

  details

   }
```

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 ▼

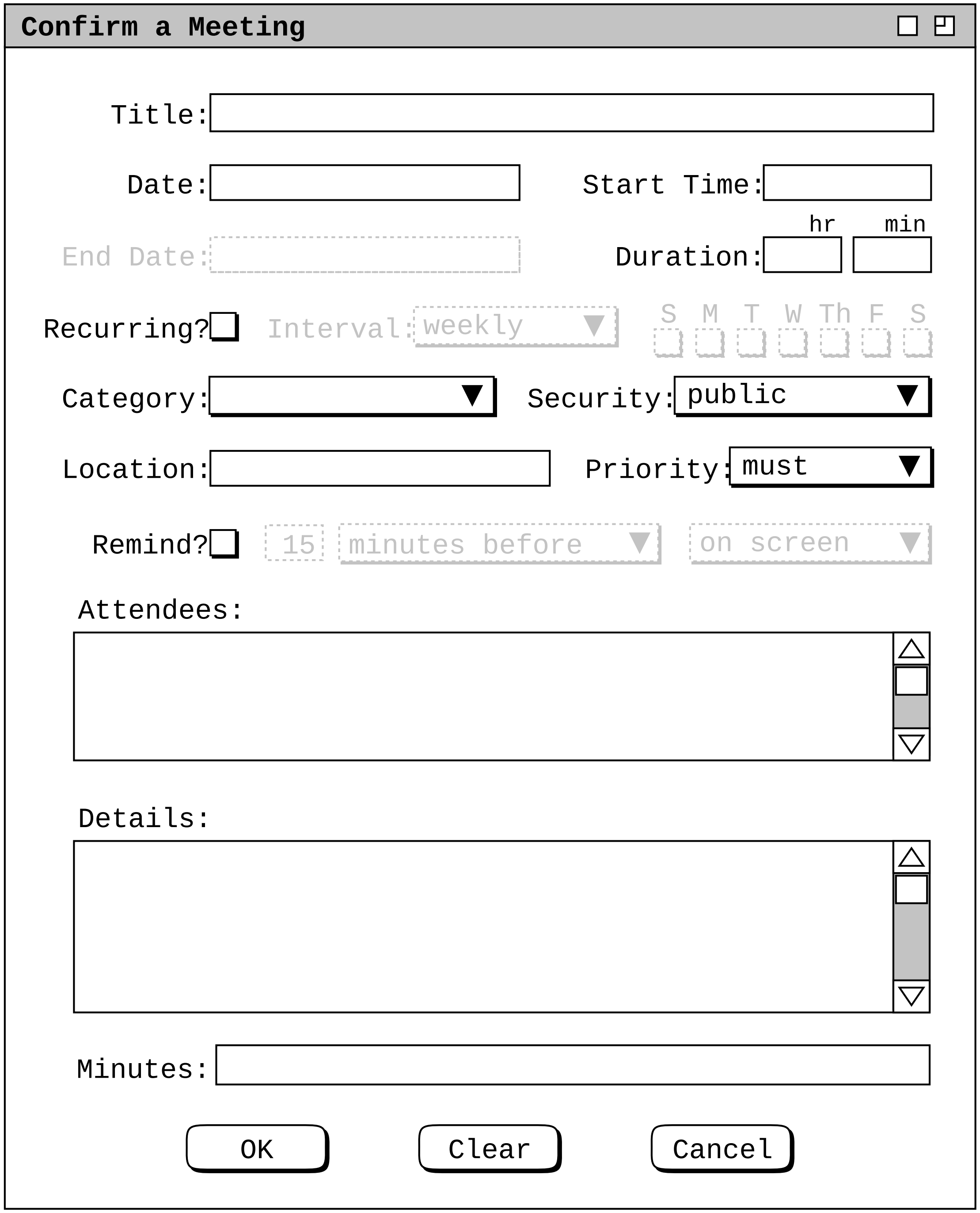
Location: Priority: must ▼

Remind? □ 15 minutes before ▼ on screen ▼

Details:

```
class Appointment {
    String title;
    Date startDate;
    Date endDate;
    Time startTime;
    Duration duration;
    RecurringInfo recurringInfo;
    Category category;
    Location location;
    AppointmentSecurity meetingSecurity;
    AppointmentPriority priority;
    RemindInfo remindInfo;
    Text Details;
}
```

C. Meeting



Confirm a Meeting

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 ▼

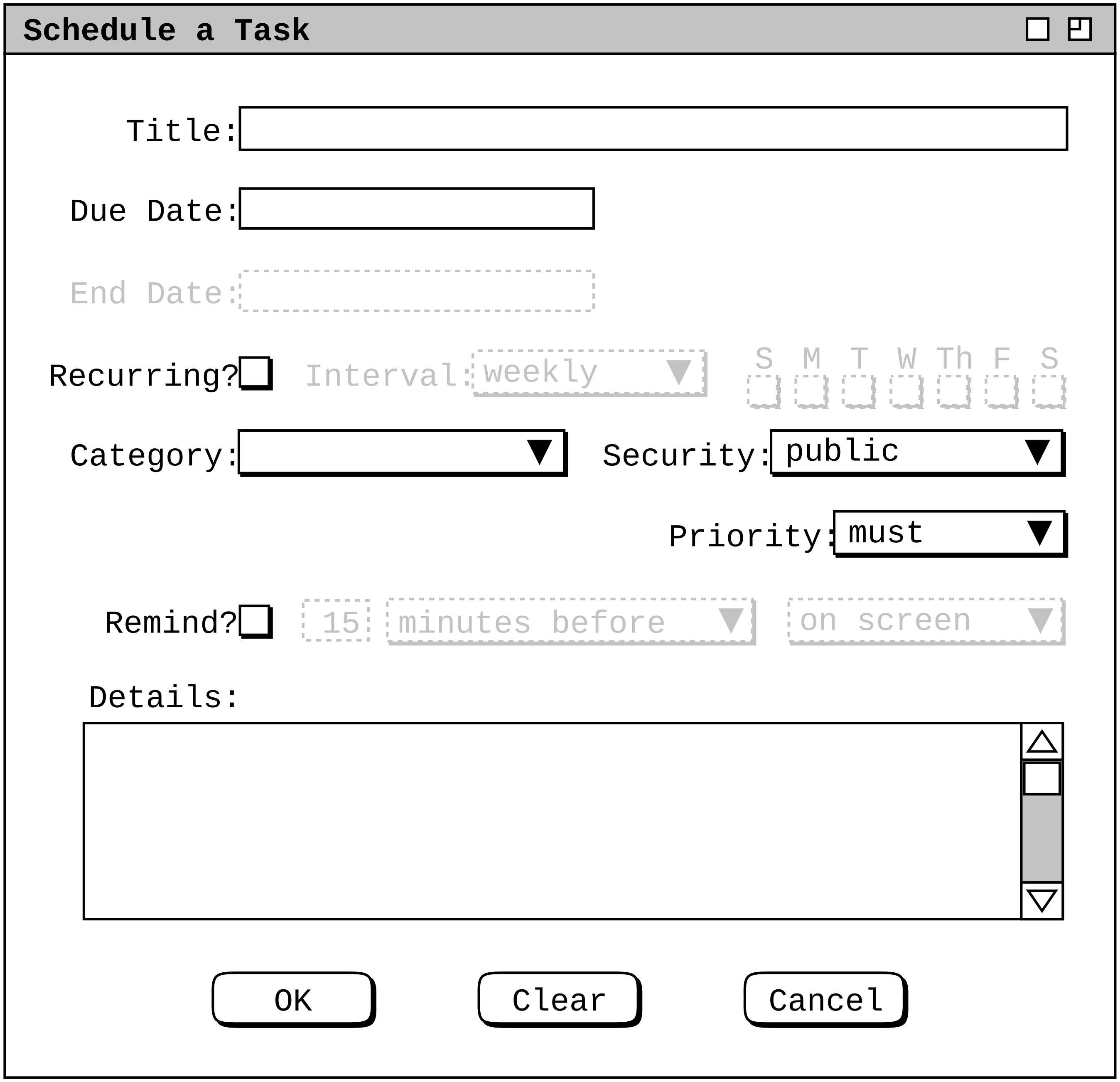
Attendees:

Details:

Minutes:

```
class Meeting {
    String title;
    Date startDate;
    Date endDate;
    Time startTime;
    Duration duration;
    RecurringInfo recurringInfo;
    Category category;
    Location location;
    AppointmentSecurity meetingSecurity;
    AppointmentPriority priority;
    RemindInfo remindInfo;
    Attendees attendees;
    Text Details;
    Text Minutes;
}
```

D. Task



Schedule a Task

Title:

Due Date:

End Date:

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

Category: Security: public

Priority: must

Remind? □ 15 minutes before on screen

Details:

```
class Task {
    String title;
    Data dueDate;
    Date endDate;
    Category category;
    Security security;
    int Priority;
    RemindInfo remindInfo;
    Text details;
    boolean carryOverFlag;
    boolean completedFlag;
}
```

Scheduling example, cont’d

1. Note `CompletedFlag` that’s not in dialog.

Scheduling example, cont'd

1. Note `CompletedFlag` that's not in dialog.

2. Neither is `carryOverFlag`.

Scheduling example, cont’d

1. Note `CompletedFlag` that’s not in dialog.

2. Neither is `carryOverFlag`.

3. There’s to-do item in M6 scenario about this.

Scheduling example, cont'd

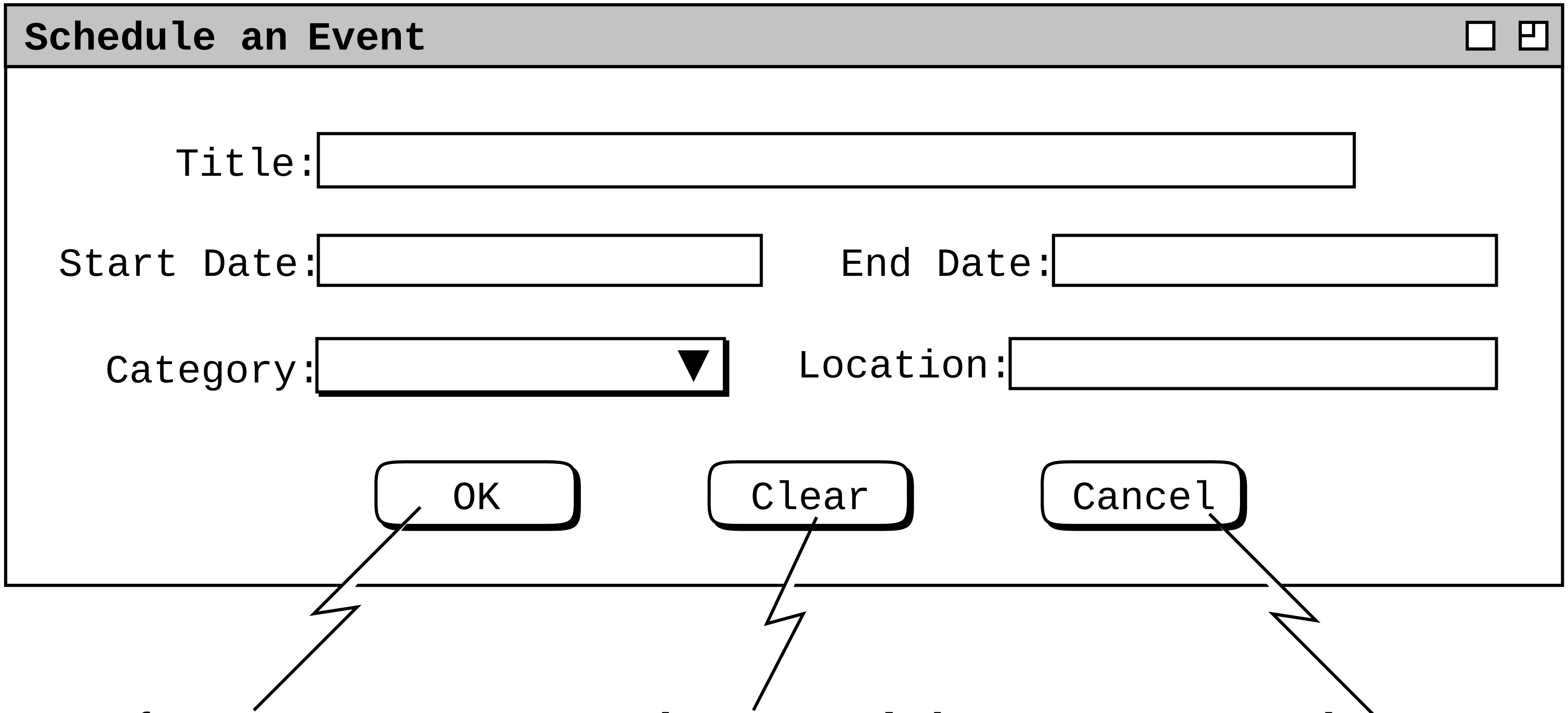
1. Note `CompletedFlag` that's not in dialog.

2. Neither is `carryOverFlag`.

3. There's to-do item in M6 scenario about this.

4. Example of model being ahead of scenarios.

E. Deriving `scheduleEvent` operation



Confirms operation scheduleEvent. (There is no operation named "ok".)

Clears input dialog. (GUI only; there is no operation named "clear".)

Cancels operation scheduleEvent (There is no operation named "cancel".)

```
class Calendar {
    . . .
    void ScheduleEvent(Event);
    . . .
}
```

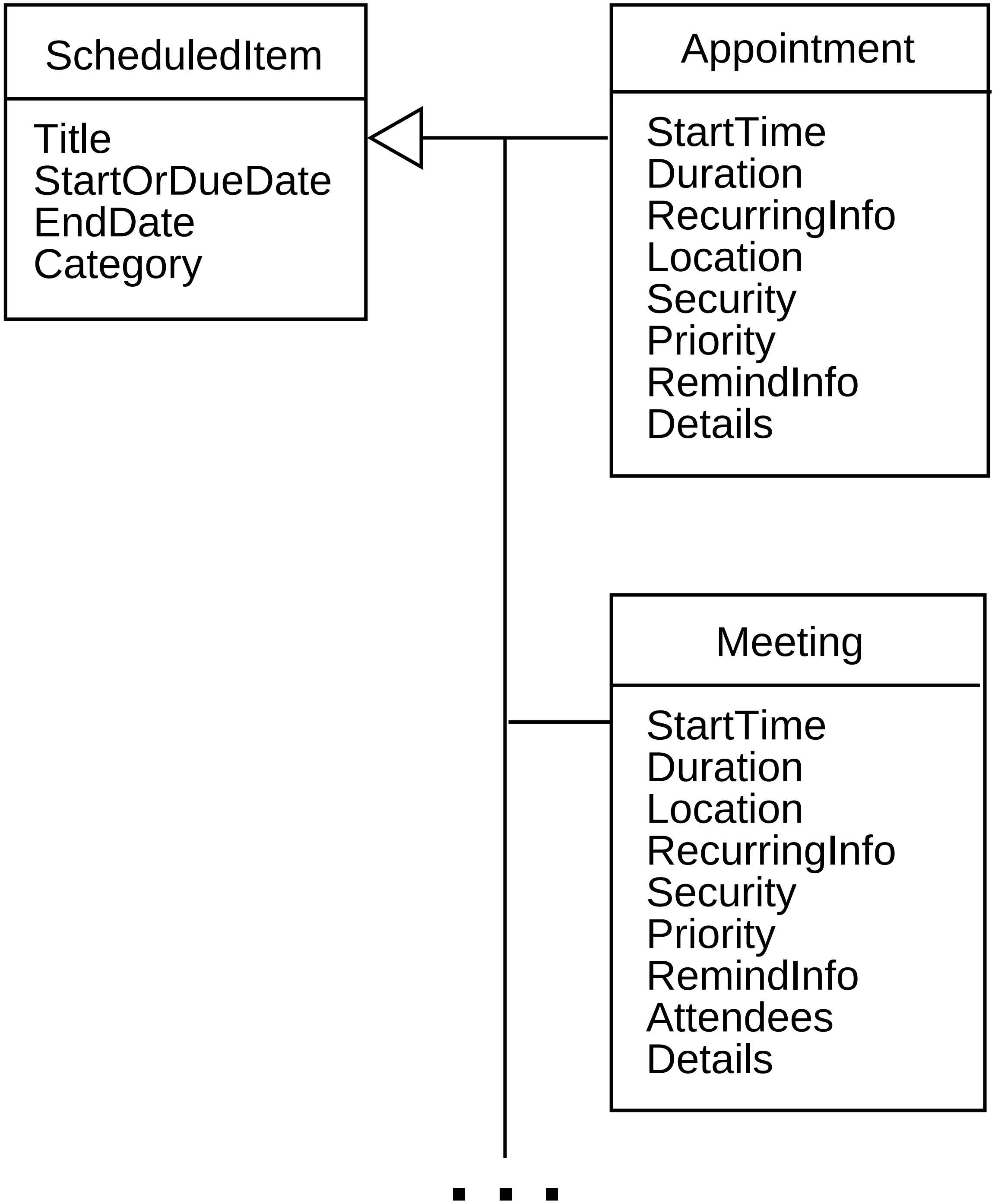
Scheduling example, cont’d

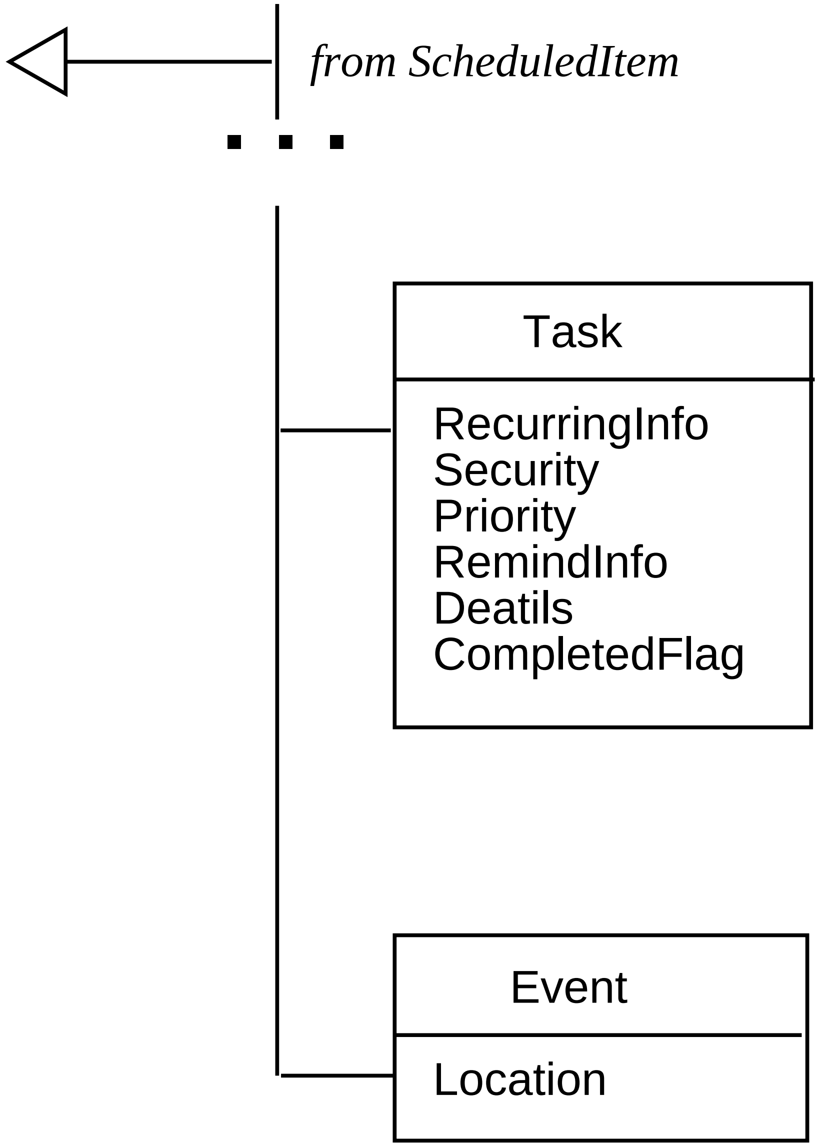
F. Refining inheritance.

Scheduling example, cont’d

F. Refining inheritance.

1. Here’s the initial version:


ScheduledItem
Title
StartOrDueDate
EndDate
Category
Appointment
StartTime
Duration
RecurringInfo
Location
Security
Priority
RemindInfo
Details
Meeting
StartTime
Duration
Location
RecurringInfo
Security
Priority
RemindInfo
Attendees
Details
. . .

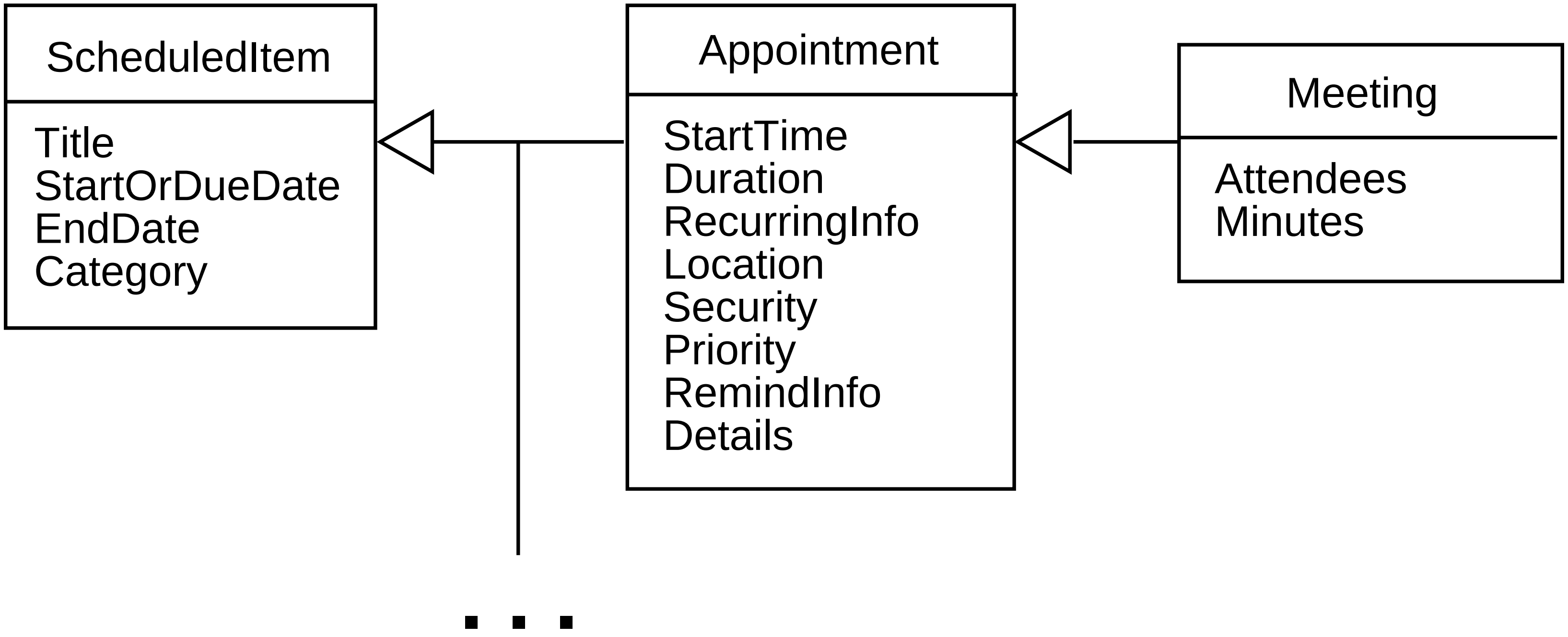


from ScheduledItem
. . .
Task
RecurringInfo
Security
Priority
RemindInfo
Deatils
CompletedFlag
Event
Location

```
class ScheduledItem {
  title; startOrDueDate; endDate;  category; }

class Appointment extends ScheduledItem {...}

class Meeting extends ScheduledItem {...}

class Task extends ScheduledItem {...}

class Event extends ScheduledItem {...}
```

Refining objects, cont'd

2. Second refinement pass.



```
classs ScheduledItem {
  title; startOrDueDate; endDate; category };

classs Appointment
  extends ScheduledItem  {...}

classs Meeting
  extends Appointment {...}
```

Scheduling example, cont’d

G. Observations.

1. Inheritance derived bottom up.
2. "What the user thinks" is driving factor in model accuracy and correctness.

VIII. Other modeling examples.

VIII. Other modeling examples.

A. See online lecture notes week 5.

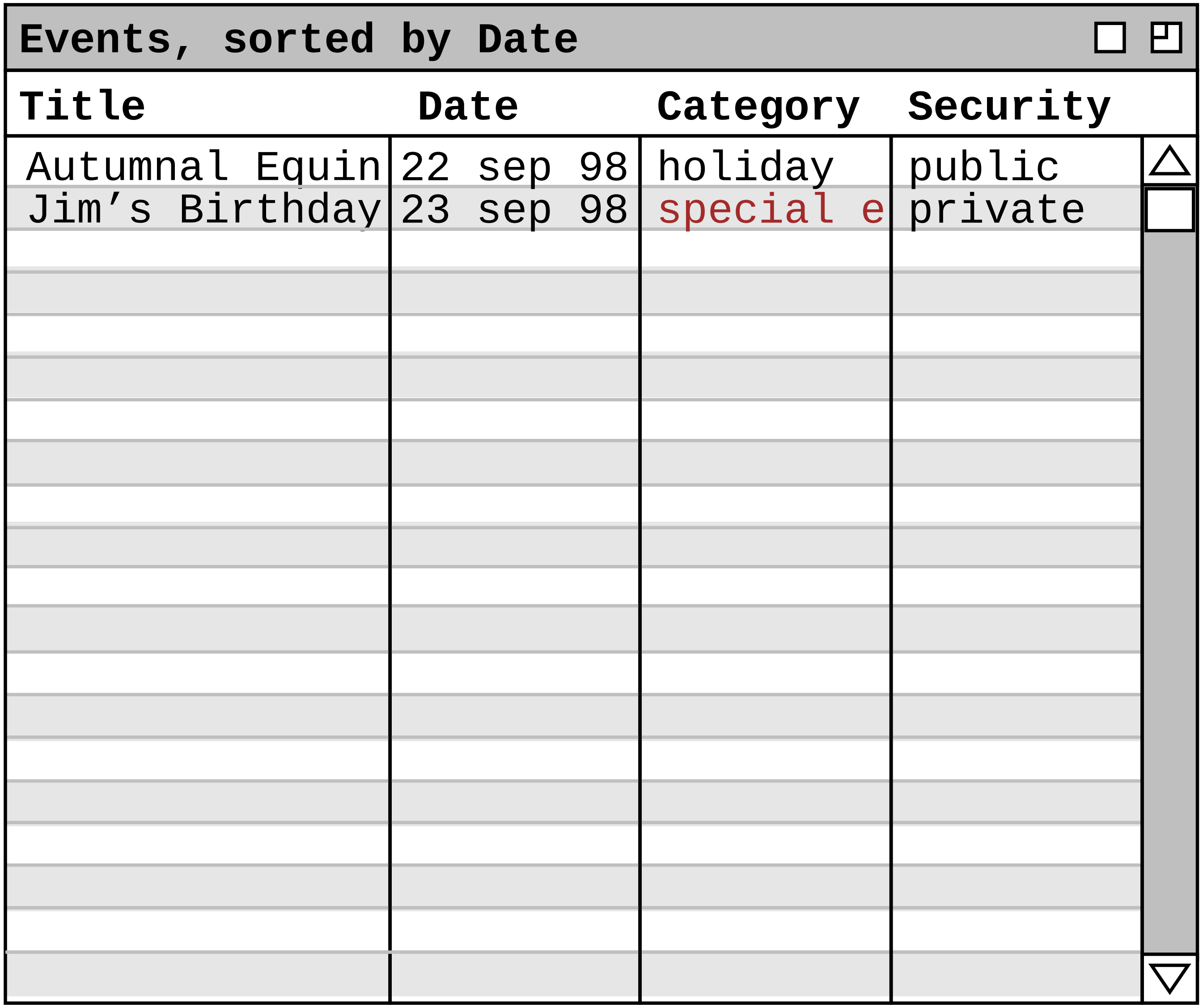
VIII. Other modeling examples.

A. See online lecture notes week 5.

B. See Milestones 4 and 6 examples.

C. A particularly common UI form --

a scrolling list with column headings:



Events, sorted by Date

Title	Date	Category	Security
Autumnal Equin	22 sep 98	holiday	public
Jim’s Birthday	23 sep 98	special e	private

modeled as:

```
class EventsByDate {
   Collection<EventListItem> items;
}

class EventListItem {
   String title;
   Date date;
   Category category;
   Security security;
}
```

IX. Summary observations

A. Goal is *abstract* model.

1. Certain details left out.

2. Much English verbiage left out.

3. Also concrete UI details.

Summary, cont’d

B. Beneficial feedback between requirements and spec phases.

1. English+pictures <==> JML+diagrams.

2. Consistency by deriving, refining, feeding back.

3. Feedback loop continues until user says done and spec passes JML checker.

X. Modeling the concrete GUI?

A. Are menus and windows objects?

1. CSC 308 answer is "no".

2. We define *abstract* model.

3. GUI is not an object.

Modeling GUI?, cont’d

B. We are specifying *direct manipulation* UIs.

C. UI structure provides modeling guidance.

Modeling GUI?, cont’d

D. Observations.

1. It’s not *wrong* to model the GUI.

2. We’re following a particular convention.

XI. Modeling the tool itself.

A. Is Calendar Tool an object?, an operation?, a module?

B. There are a variety of ways to model the overall system itself; here it is as an object:

Modeling the Tool, cont’d

```
/****
 *
 * Class CalendarTool ...
 *
 */
class CalendarTool {
    CalendarDB calendarDB;
    FileSpace fileSpace;
    SystemState systemState;
}
```

Modeling the Tool, cont’d

```
class CalendarDB { /* ... */ }

class FileSpace { /* ... */ }

class SystemState { /* ... */ }
```

Modeling the Tool, cont’d

C. We’ll discuss further in upcoming lectures.

XII. Mechanically checking a spec.

A. Run `javac`

```
> cd your-project/specification
> javac *.java
```

Mechanically checking, cont’d

B. Generate `javadoc`

```
> cd your-project/specification
> mkdir javadoc
> cd javadoc
> javadoc -private ../*.java
```