CSC 309 Lecture Notes Week 3

More on Model/View Design

Design for Independent, Incremental Testing

Refining Model Design Using Java Library

Bi-Weekly Reports --

Please submit by this eve.

Recap of Milestone 2 Deliverables:

Recap of Milestone 2 Deliverables:

1. well-organized package dirs

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views
5. operational top-level UI

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views
5. operational top-level UI
6. two classes, six model/view method pairs per member

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views
5. operational top-level UI
6. two classes, six model/view method pairs per member
7. javadoc for all classes

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views
5. operational top-level UI
6. two classes, six model/view method pairs per member
7. javadoc for all classes
8. @author tags for all java files

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views
5. operational top-level UI
6. two classes, six model/view method pairs per member
7. javadoc for all classes
8. @author tags for all java files
9. m2-duties.html file (more detailed work breakdown)

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views
5. operational top-level UI
6. two classes, six model/view method pairs per member
7. javadoc for all classes
8. @author tags for all java files
9. m2-duties.html file (more detailed work breakdown)
10. updated work breakdown

Recap of Milestone 2 Deliverables:

1. well-organized package dirs
2. overview.html file for project
3. package.html files for each package
4. java files for models and views
5. operational top-level UI
6. two classes, six model/view method pairs per member
7. javadoc for all classes
8. @author tags for all java files
9. m2-duties.html file (more detailed work breakdown)
10. updated work breakdown
11. HOW-TO-RUN.html file

I. Model/View vs Model/View/Controller

A. Controller is ***mediator*** between model,view.

B. Without controller, model/view communication can be ***direct***, i.e., unmediated.

C. Browser-based apps most particularly need such mediation. (***Why?***)

II. The Model/View/Process Variant of Model/View/*Whatever*

A. Orignated long ago in ***Smalltalk***

B. **Model/View** same as MVC

C. **Process** component:

1. support model
2. has no view
3. encapsulates low-level processing, e.g., communication, databases

III. Details of `mvp` abstract classes

A. Model classes inherit from `mvp.Model`.

B. Similarly, view classes inherit `mvp.View`.

C. Java class defs in the notes.

D. Online at 309/lib

Details of mvp package, cont’d

1. Comments provide design rationale.
2. Methods have pre and postconditions.
3. You’re welcome, but not required to use `mvp`.
4. Code’s in `309/lib/source/java/mvp`.
5. Jar file’s in `309/lib/csc309libs.jar`.

IV. Method-call backtraces.

A. Illustrate invocation in event-based design.

1. Shows order of method calls.

2. Generated using `jdb`.

Backtraces, cont’d

B. Setup `File New` menu item

```
[1] FileMenu.addNewItem (FileMenu.java:111)
[2] FileMenu.compose (FileMenu.java:64)
[3] FileUI.compose (FileUI.java:35)
[4] CalendarToolUI.composeMenuBar
      (CalendarToolUI.java:186)
[5] CalendarToolUI.compose
      (CalendarToolUI.java:114)
[6] main (CalendarTool.java:114)
```

C. OK button in SCheduleEventDialog

```
[1] OKScheduleEventButtonListener
      (OKScheduleEventButtonListener.java:32)
[2] ScheduleEventDialog.composeButtonRow
      (ScheduleEventDialog.java:251)
[3] ScheduleEventDialog.compose
      (ScheduleEventDialog.java:96)
[4] ScheduleUI.compose (ScheduleUI.java:56)
[5] CalendarToolUI.composeMenuBar
      (CalendarToolUI.java:188)
[6] CalendarToolUI.compose
      (CalendarToolUI.java:114)
[7] main (CalendarTool.java:114)
```

D. Press File New menu item.

```
[1] caltool.file.File.fileNew (File.java:36)
[2] caltool.file_ui.FileMenu$1.
      actionPerformed(FileMenu.java:117)
[3] javax.swing.AbstractButton.
      fireActionPerformed
          (AbstractButton.java:1,819)
    ...
[10] java.awt.Component.processMouseEvent
        (Component.java:5,166)
    ...
[22] java.awt.EventQueue.dispatchEvent
        (EventQueue.java:456)
    ...
[27] java.awt.EventDispatchThread.run
        (EventDispatchThread.java:100)
```

E. Press OK in SCheduleEventDialog

```
[1] caltool.schedule.Schedule.
      scheduleEvent (Schedule.java:93)
[2] caltool.schedule_ui.
      OKScheduleEventButtonListener.actionPerformed
        (OKScheduleEventButtonListener.java:50)
[3] javax.swing.AbstractButton.
      fireActionPerformed
          (AbstractButton.java:1,819)
    ...
[25] java.awt.EventDispatchThread.run
        (EventDispatchThread.java:100)
```

F. Press `View Lists Appointments`.

```
[1] caltool.view.Lists.viewAppointmentsList
      (Lists.java:60)
[2] caltool.view_ui.AppointmentsListDisplay
      .update (AppointmentsListDisplay.java:79)
[3] caltool.view_ui.ViewMenu$11.actionPerformed
      (ViewMenu.java:263)
[4] javax.swing.AbstractButton. ...
    ...

[28] java.awt.EventDispatchThread.run
        (EventDispatchThread.java:100)
```

V. "Canned" model data.

A. For initial testing of model/view design.

1. In beginning, can be entirely "canned".

2. Get concrete examples from requirements.

Canned model data, cont'd

B. Delivered to view using methods that will ultimately produce real data.

1. E.g., an iterator method.

2. Or generated by temporary testing method.

Canned model data, cont'd

C. Examples in code from Week 3 notes.

1. Iterator methods in `MonthlyAgenda` deliver to `MonthlyAgendaDisplay`.

2. In `Lists` model class, there is `generateSampleList()` method.

VI. Designing for independently testable pkgs.

A. Team members can test independently.

B. Provide "canned" test data.

1. For pkgs not yet implemented.

2. Also handy when imple’d package breaks.

Independently testable pkgs, cont’d

C. Individualized `main` methods.

1. Can be in model classes.

2. Will evolve to formal testing classes.

Independently testable pkgs, cont’d

D. Testing `main`s do this:

1. Construct model class(es) to be tested.
2. Construct, compose companion view(s).
3. Construct canned test data.
4. Show the top-level view(s).

Independently testable pkgs, cont'd

E. Independently-testable designs allow *incremental* development.

Question:

How many packages and classes in the standard Java library?

Answer:

- *In Java 8:*
  - *217 packages*
  - *4240 classes*
- *In Java 7 it was 209 and 4205*

  In Java 6 it was 203 and 3793

VII. Java library for model and process data.

A. Key packages:

1. *java.lang*

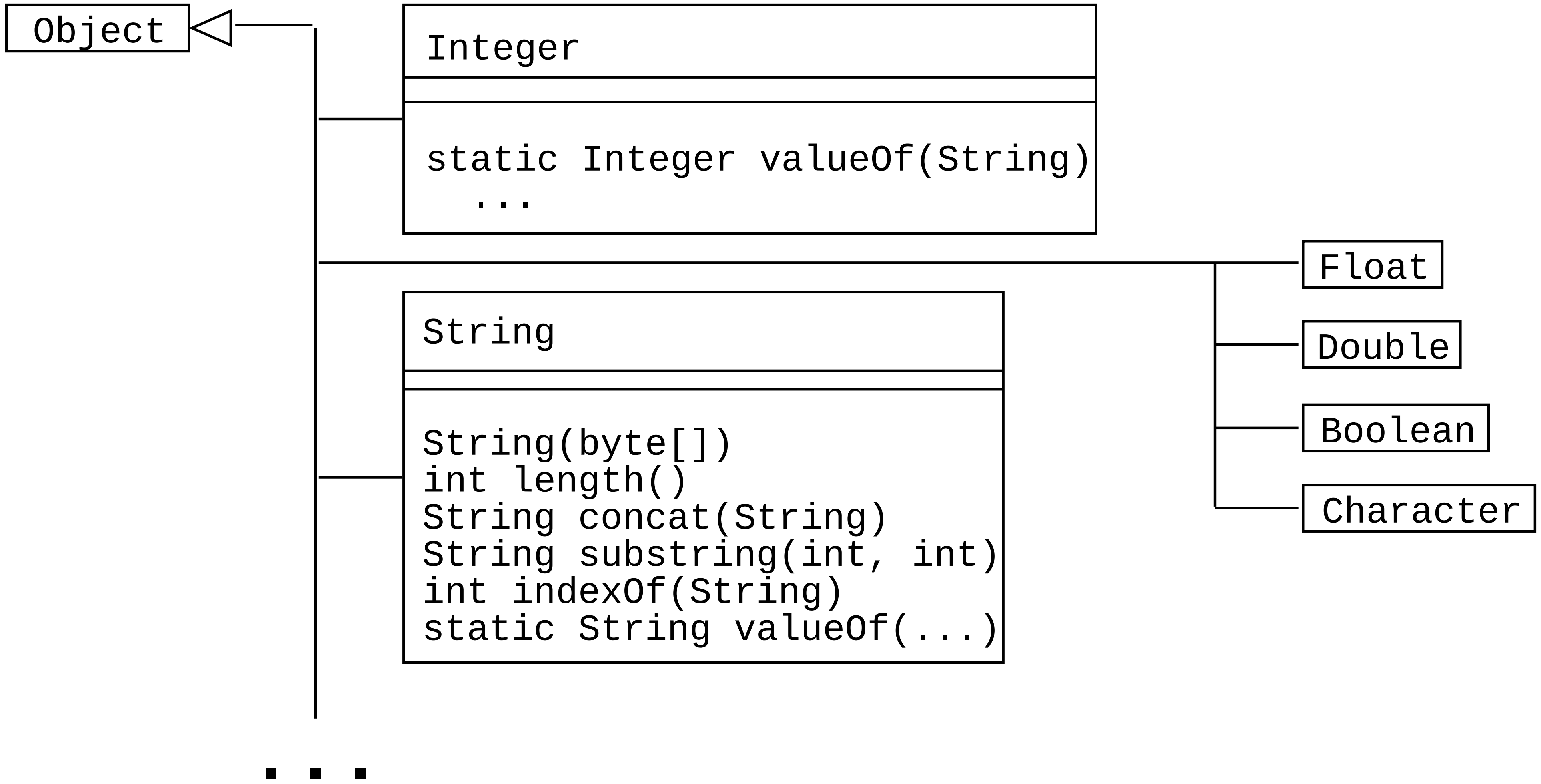
2. *java.util*

3. *java.io*

B. Central to work in 309.

C. Summarized in UML diagrams.

D. Package **java.lang**



Object

Integer

```
static Integer valueOf(String)
  ...
```

String

```
String(byte[])
int length()
String concat(String)
String substring(int, int)
int indexOf(String)
static String valueOf(...)
```

Float

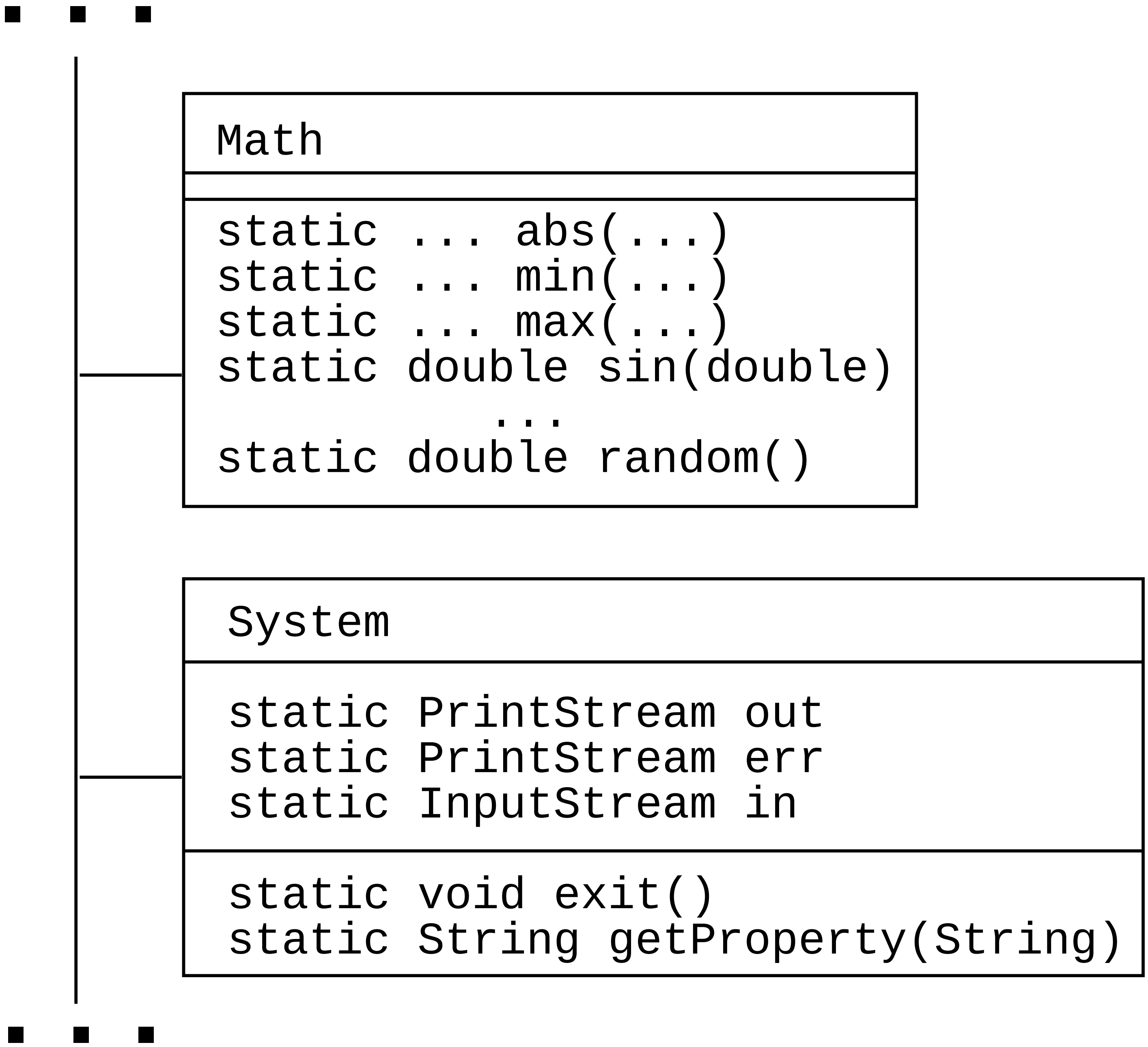
Double

Boolean

Character

. . .

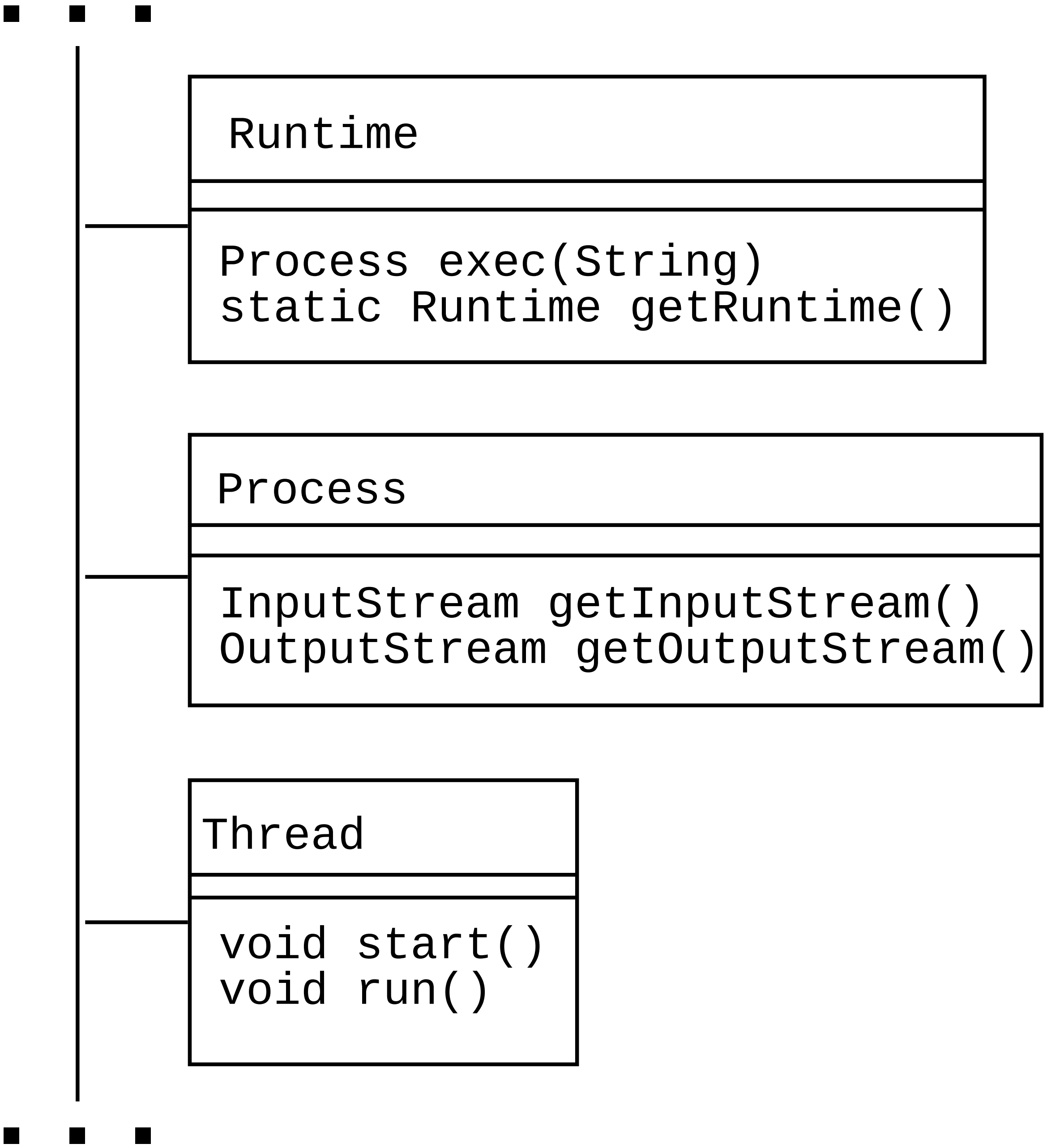
java.lang, cont’d



```
Math

static ... abs(...)
static ... min(...)
static ... max(...)
static double sin(double)
         ...
static double random()
```

```
System

static PrintStream out
static PrintStream err
static InputStream in

static void exit()
static String getProperty(String)
```

java.lang, cont’d



Runtime

Process exec(String)
static Runtime getRuntime()

Process

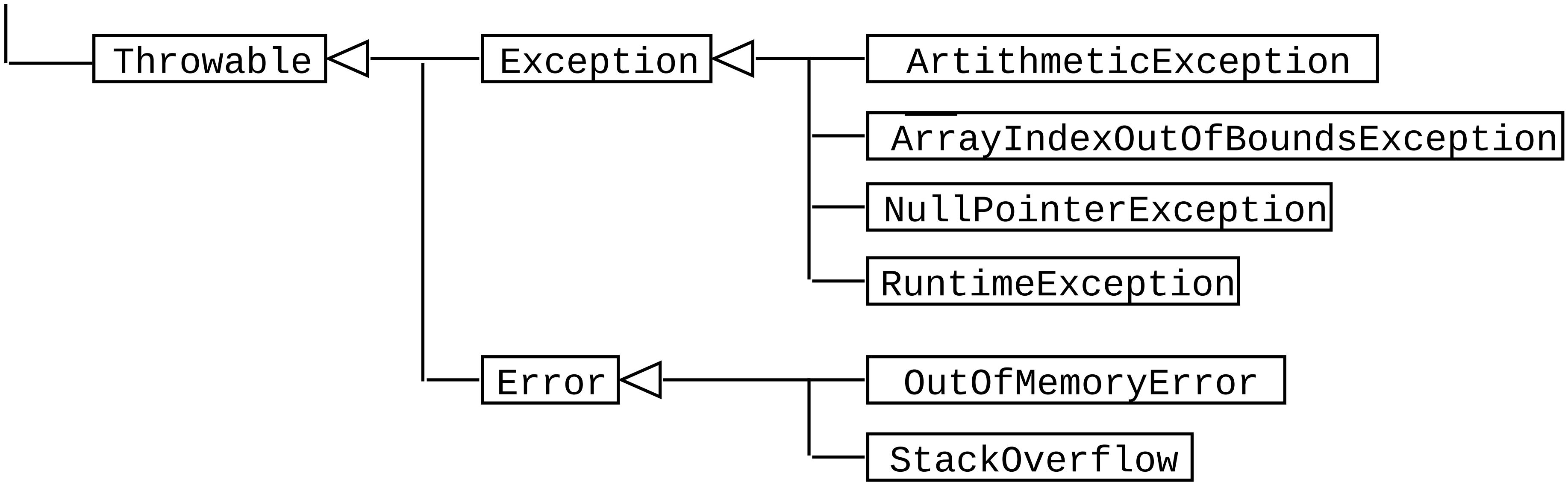
InputStream getInputStream()
OutputStream getOutputStream()

Thread

void start()
void run()

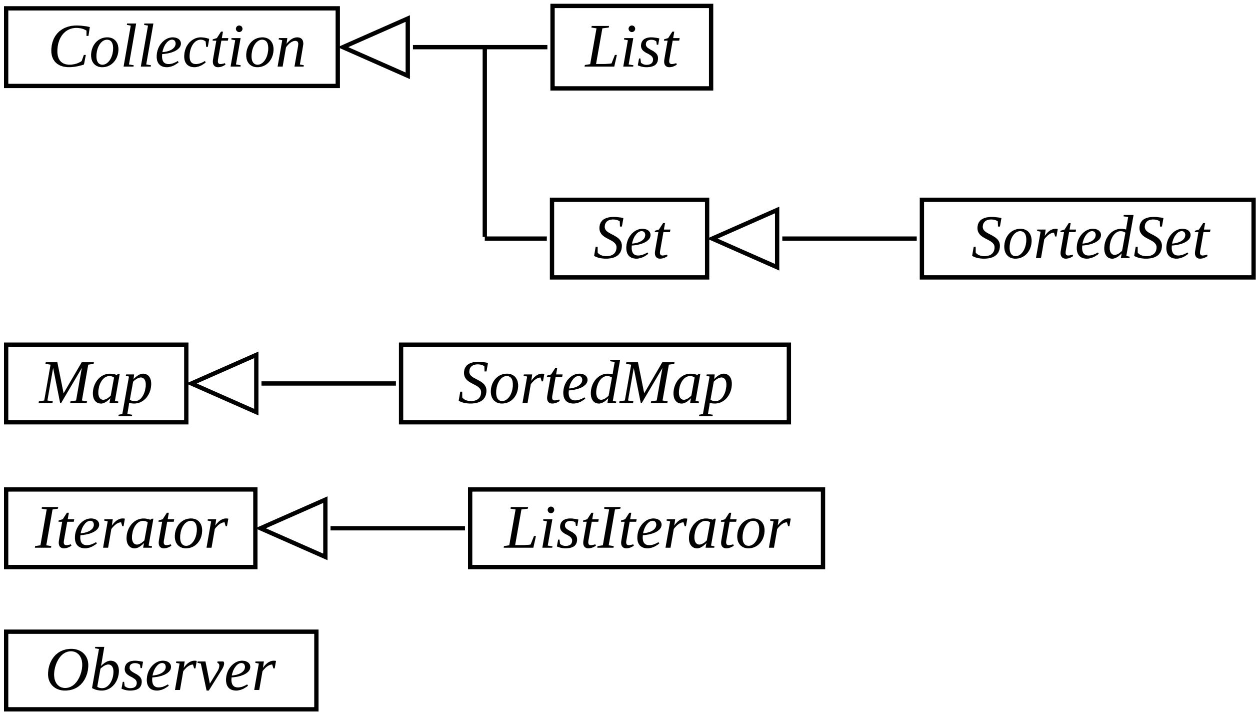
java.lang, cont’d

...

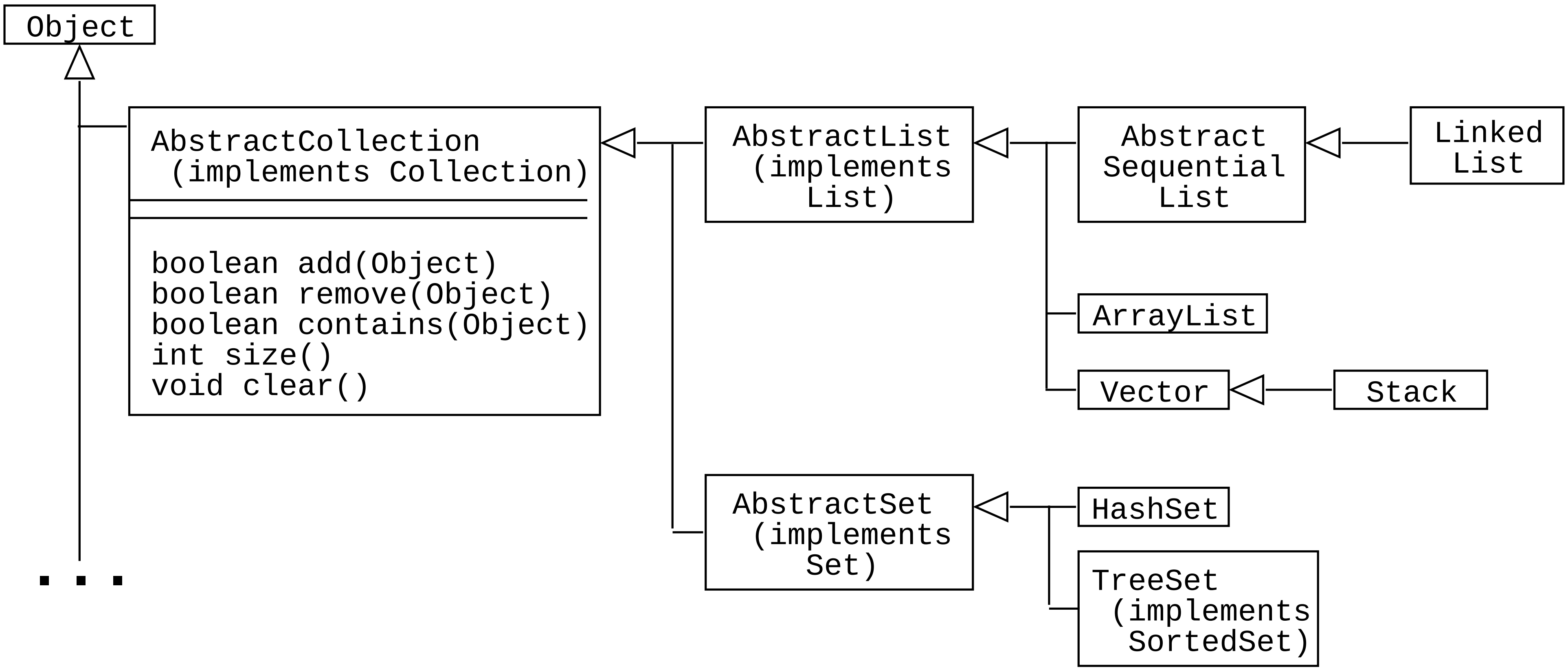


E. Package **`java.util`**

Interfaces:



java.util, cont’d



Object

AbstractCollection
(implements Collection)

boolean add(Object)
boolean remove(Object)
boolean contains(Object)
int size()
void clear()

AbstractList
(implements
List)

Abstract
Sequential
List

Linked
List

ArrayList

Vector

Stack

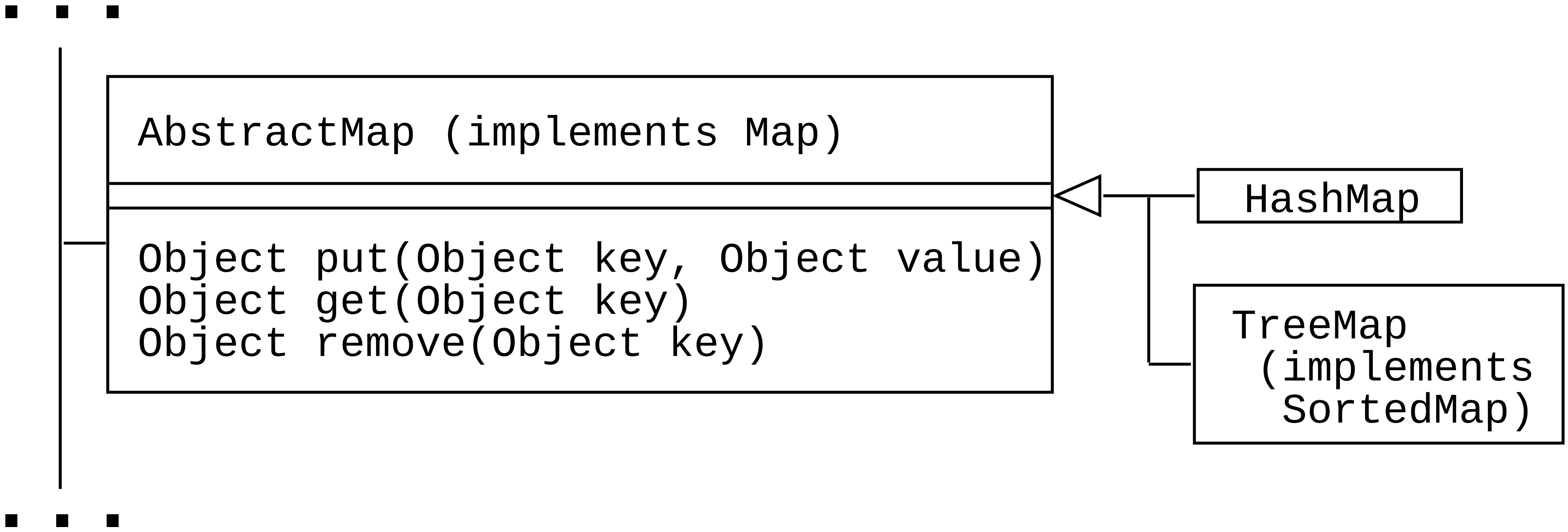
AbstractSet
(implements
Set)

HashSet

TreeSet
(implements
SortedSet)

. . .

java.util, cont’d



. . .

AbstractMap (implements Map)

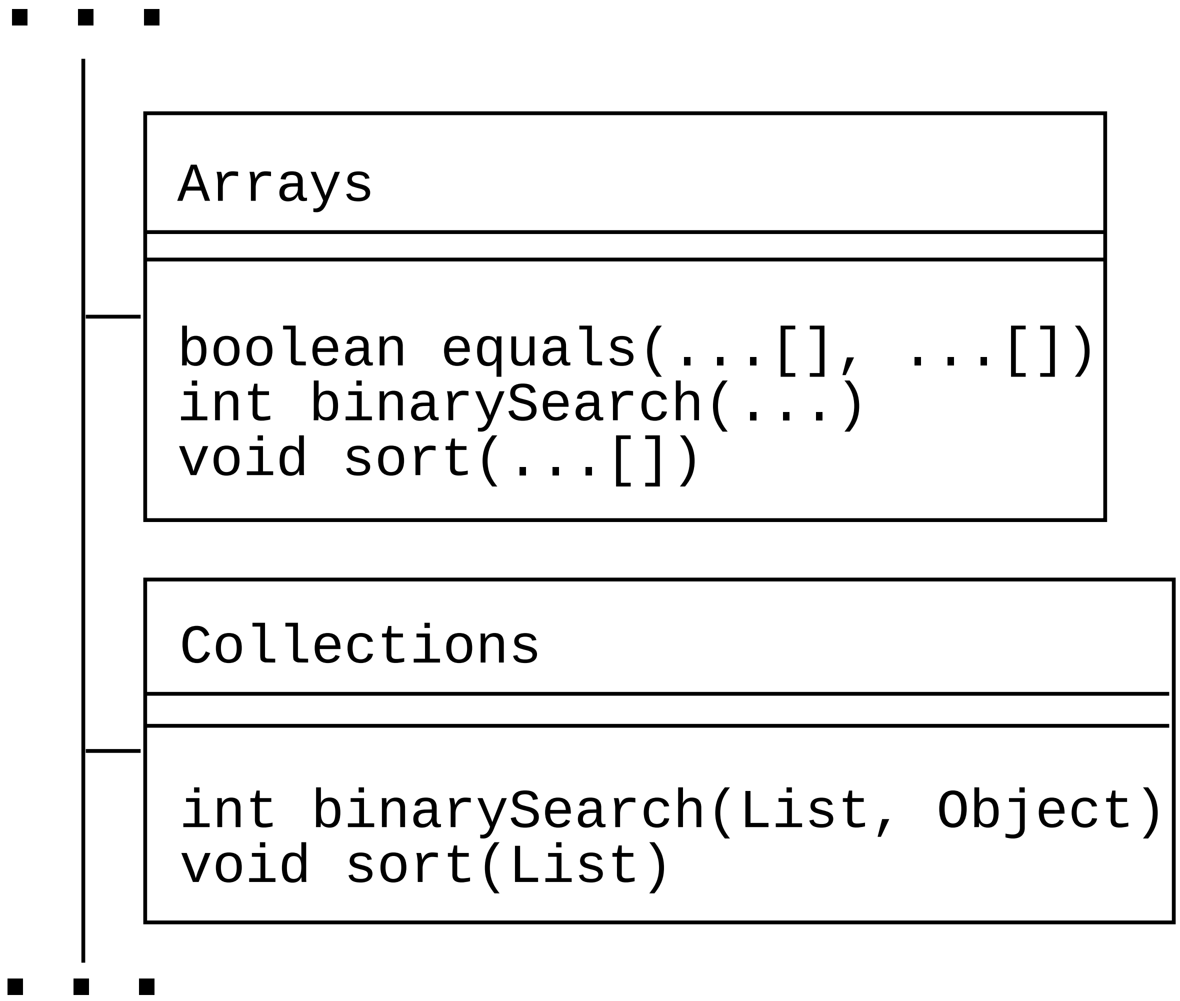
Object put(Object key, Object value)
Object get(Object key)
Object remove(Object key)

HashMap

TreeMap
(implements
SortedMap)

. . .

java.util, cont’d



. . .

Arrays

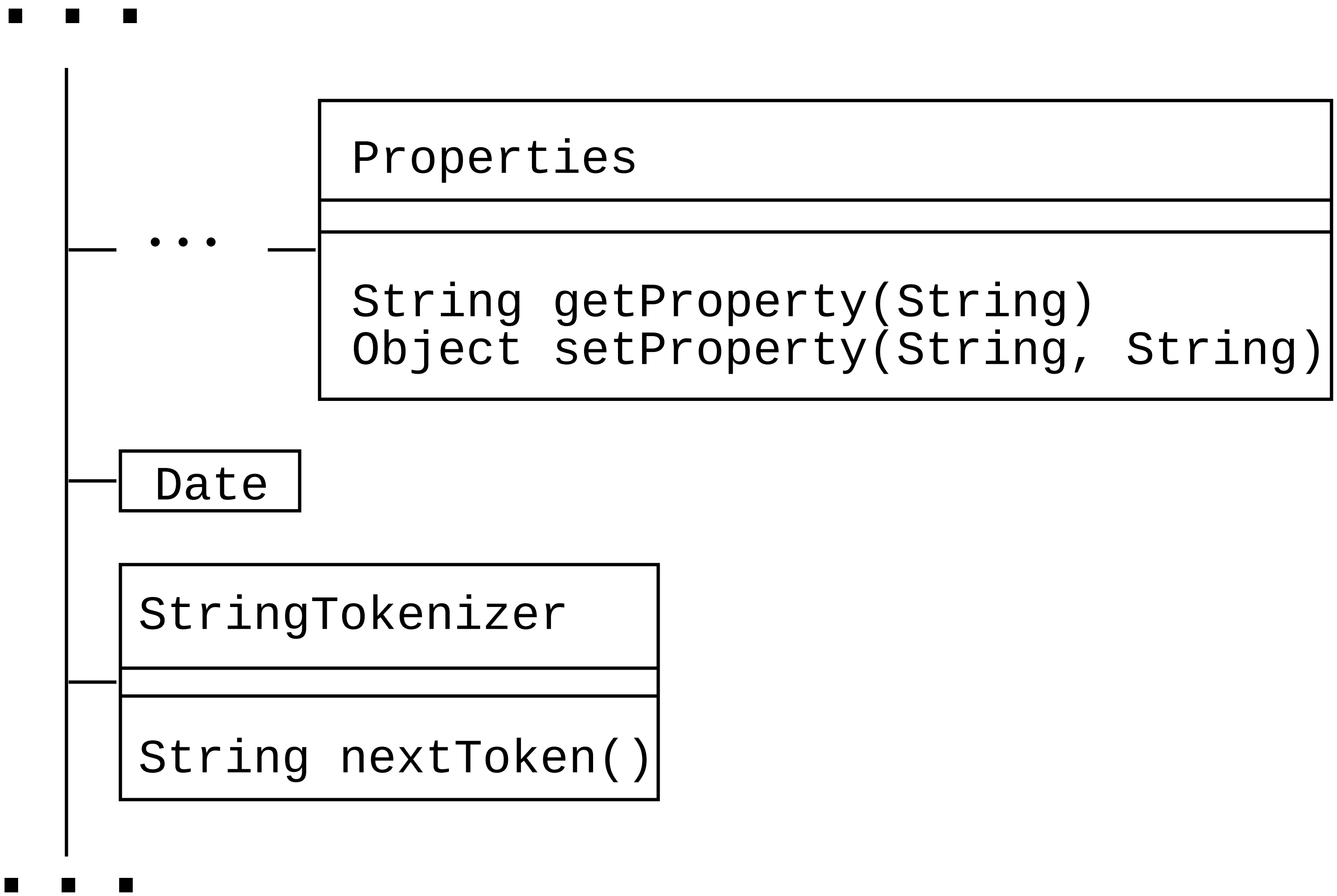
```
boolean equals(...[], ...[])
int binarySearch(...)
void sort(...[])
```

Collections

```
int binarySearch(List, Object)
void sort(List)
```

. . .

java.util, cont’d



. . .

. . .

Properties

String getProperty(String)
Object setProperty(String, String)

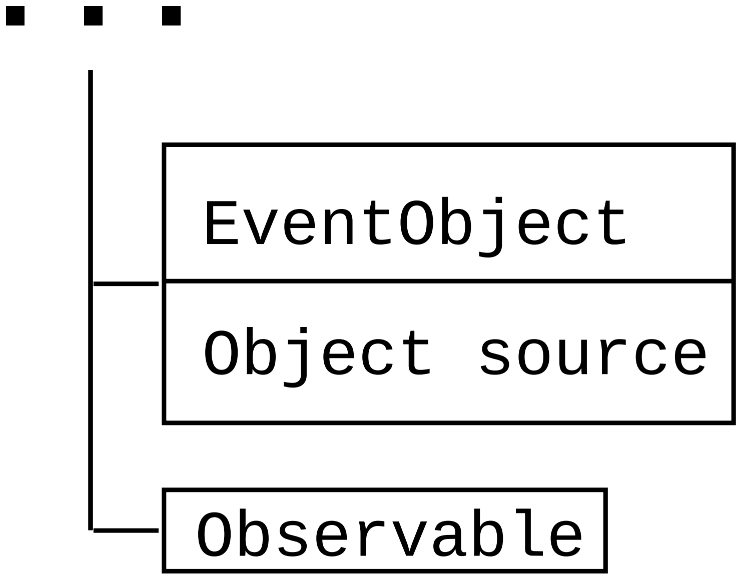
Date

StringTokenizer

String nextToken()

. . .

java.util, cont’d



. . .

EventObject

Object source

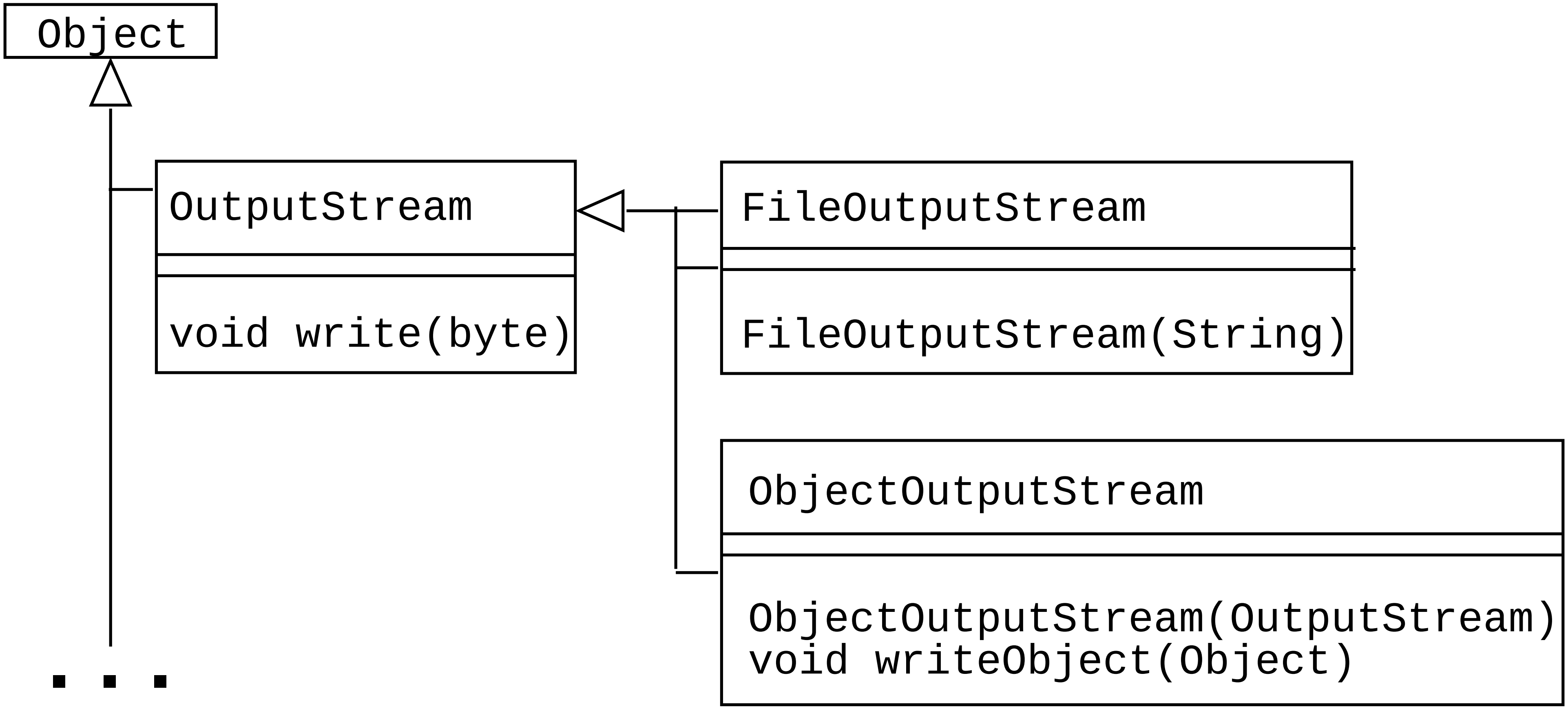
Observable

F. Package **`java.io`**

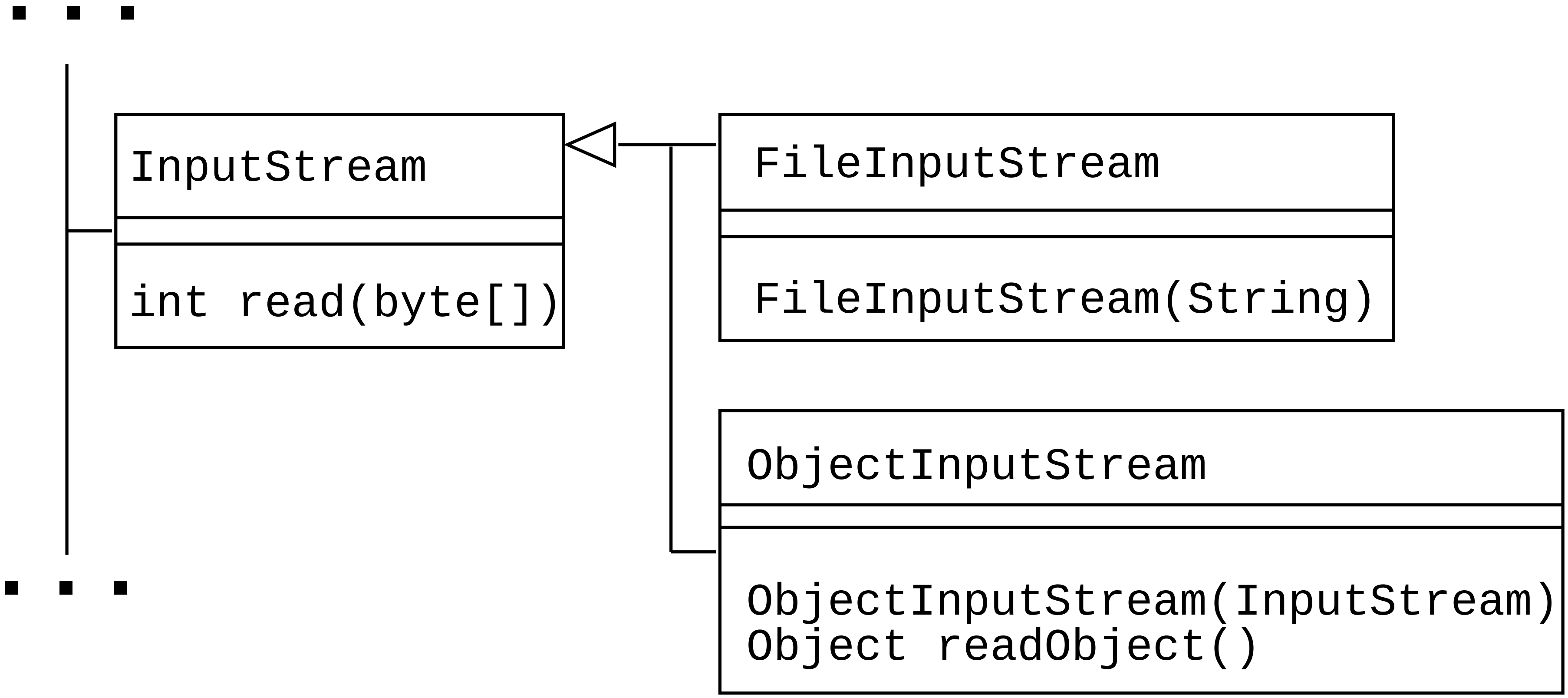
Interfaces: *Serializable*

• • •

java.io, cont’d



java.io, cont’d



. . .

InputStream

int read(byte[])

FileInputStream

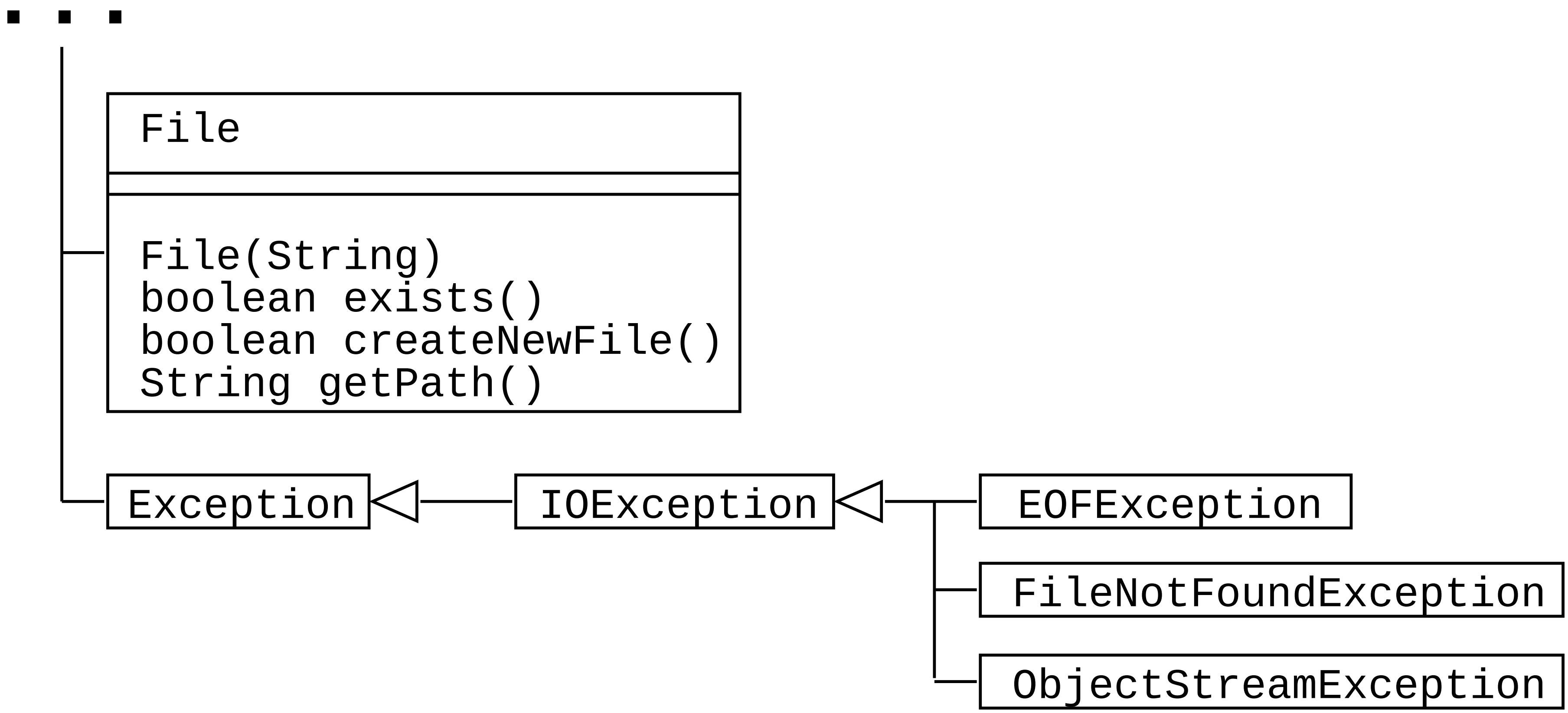
FileInputStream(String)

ObjectInputStream

ObjectInputStream(InputStream)
Object readObject()

. . .

java.io, cont’d



VIII. View data collection and validation.

A. When user enters data, View class collects in raw form.

B. E.g., `getText` extracts string from `JTextField`.

View data collection, cont'd

C. Once raw data are collected they are:

1. Converted by Model, from their raw form.
2. Validated by Model, based on preconditions to a model method.
3. Processed by Models as appropriate.

IX. Exception handling in data validation

A. There are different ways to perform input data validation in a model/view design.

B. Most, if not all, done by model.

Exception handling in data validation, cont’d

1. Jargon is: *"smart model, stupid view"*.

2. View does not know data semantics.

Exception handling in data validation, cont'd

3. View's in charge of displaying data, and interacting with user.

4. Model's in charge of storing data, managing access, manipulation, and validation.

Exception handling in data validation, cont'd

C. A useful way to handle validation is with exception handling

D. We'll now discuss this.

X. Quick review of exception handling.

A. Normally, method returns to caller.

B. Abnormally, method throws an exception.

Review of exception handling, cont’d

1. Excep’n exit is separate from normal return.

2. Return to nearest method that does catch.

3. In immediate caller, or higher.

4. Must be caught by active method.

Review of exception handling, cont’d

C. Different languages provide different styles.

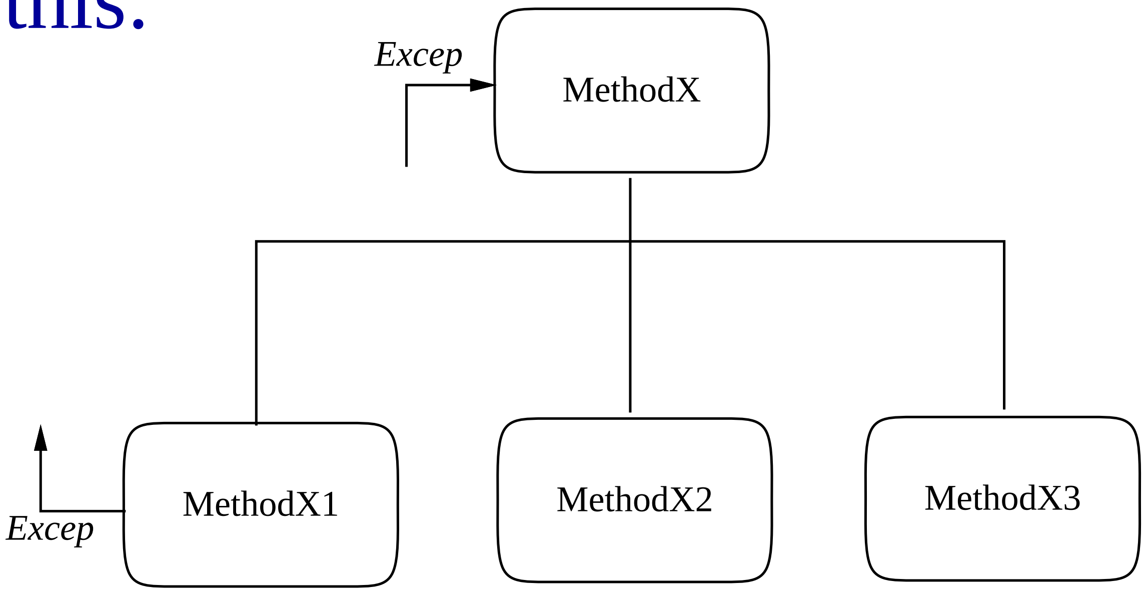
1. For design, there’s a graphical notation.

2. For implementation, there’s Java syntax.

XI. Design diagram notation

A. Shown with labeled arrows.

B. Like this:



Exception handling, cont’d

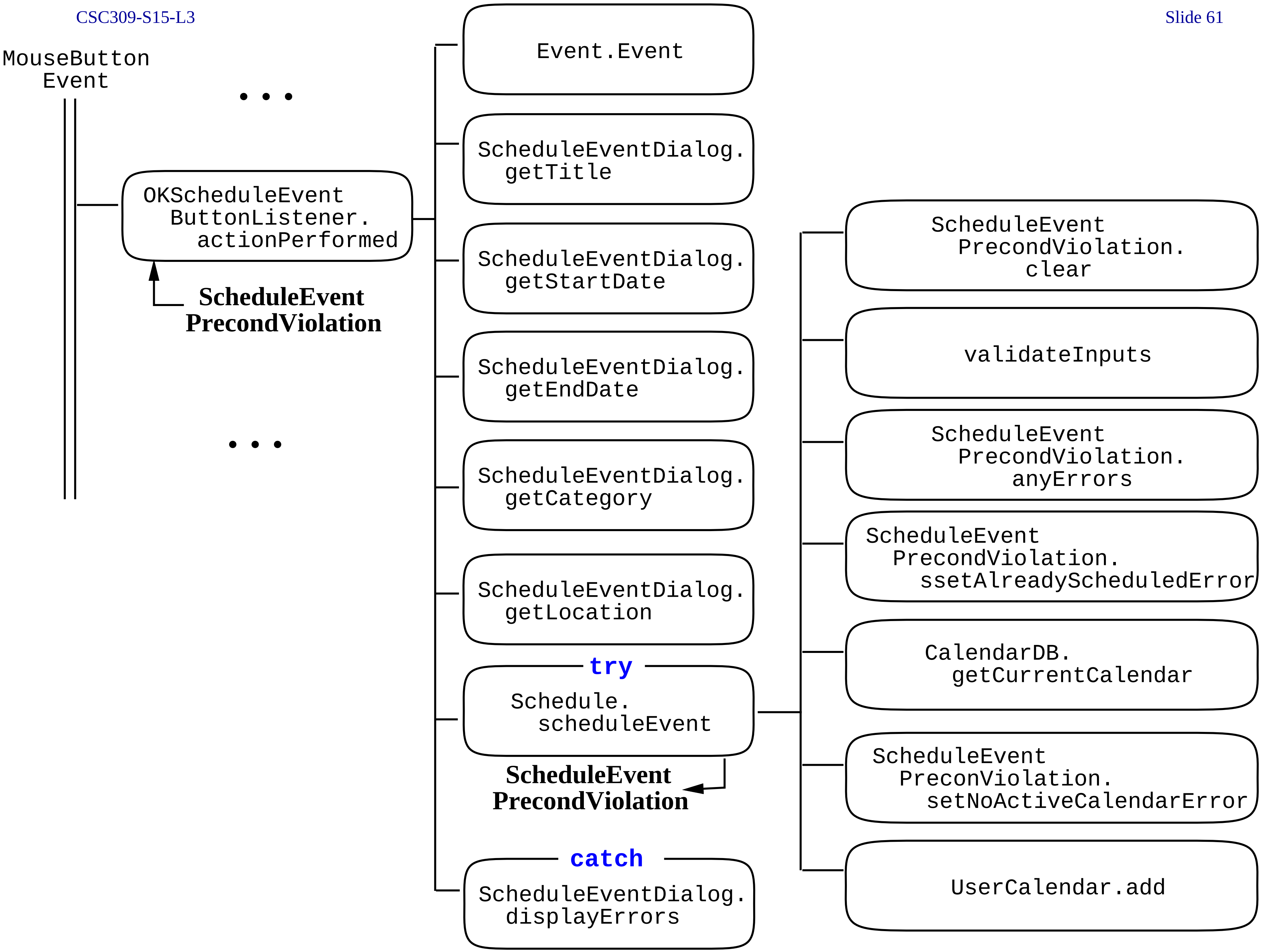
1. `MethodX` calls `X1`, `X2`, `X3`.

2. `X2` and `X3` return in normal way.

3. `X1` can return normal, or throw an exception caught by `MethodX`.

XII. Example Model-View comm’n

A. Next figure illustrates typical case.


MouseButton Event
. . .
OKScheduleEvent ButtonListener. actionPerformed
ScheduleEvent PrecondViolation
. . .
Event.Event
ScheduleEventDialog. getTitle
ScheduleEventDialog. getStartDate
ScheduleEventDialog. getEndDate
ScheduleEventDialog. getCategory
ScheduleEventDialog. getLocation
try
Schedule. scheduleEvent
ScheduleEvent PrecondViolation
catch
ScheduleEventDialog. displayErrors
ScheduleEvent PrecondViolation. clear
validateInputs
ScheduleEvent PrecondViolation. anyErrors
ScheduleEvent PrecondViolation. ssetAlreadyScheduledError
CalendarDB. getCurrentCalendar
ScheduleEvent PreconViolation. setNoActiveCalendarError
UserCalendar.add

Example, cont'd

B. Model throws to view (or controller).

C. Throw when input errors detected.

D. See code for

```
OKScheduleEventButtonListener.
    actionPerformed()
```

XIII. Use of Formal Specs in Data Validation

A. Preconds define precisely the data validation requirements.

B. Precond bool logic implemented directly.

C. Message content of `PrecondViolation` corresponds directly to precond clauses.