CSC 307 Lecture Notes Week 7

Design for Independent, Incremental Testing

Refining Model Design Using Java Library

Some Key Design Patterns for 307 Projects

I. Designing for independently testable pkgs.

I. Designing for independently testable pkgs.

A. Team members can test independently.

I. Designing for independently testable pkgs.

A. Team members can test independently.

B. Provide independent test data.

I. Designing for independently testable pkgs.

A. Team members can test independently.

B. Provide independent test data.

1. For pkgs not yet implemented.

I. Designing for independently testable pkgs.

A. Team members can test independently.

B. Provide independent 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.

Independently testable pkgs, cont’d

C. Individualized `main` methods.

1. Can be in model classes.

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 `mains` do this:

Independently testable pkgs, cont’d

D. Testing `main`s do this:

1. Construct model class(es) to be tested.

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).

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.

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.

NOTE:

We’re now moving on to topics beyond Milestone 6 ...

II. Java library for model and process data.

II. Java library for model and process data.

Going shopping for all the stuff
you implemented yourself in CSC 1xx.

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*

A. Key packages:

A. Key packages:

1. *java.lang*

A. Key packages:

1. *java.lang*

2. *java.util*

A. Key packages:

1. *java.lang*
2. *java.util*
3. *java.io*

A. Key packages:

1. *java.lang*
2. *java.util*
3. *java.io*

B. Central to work in 307.

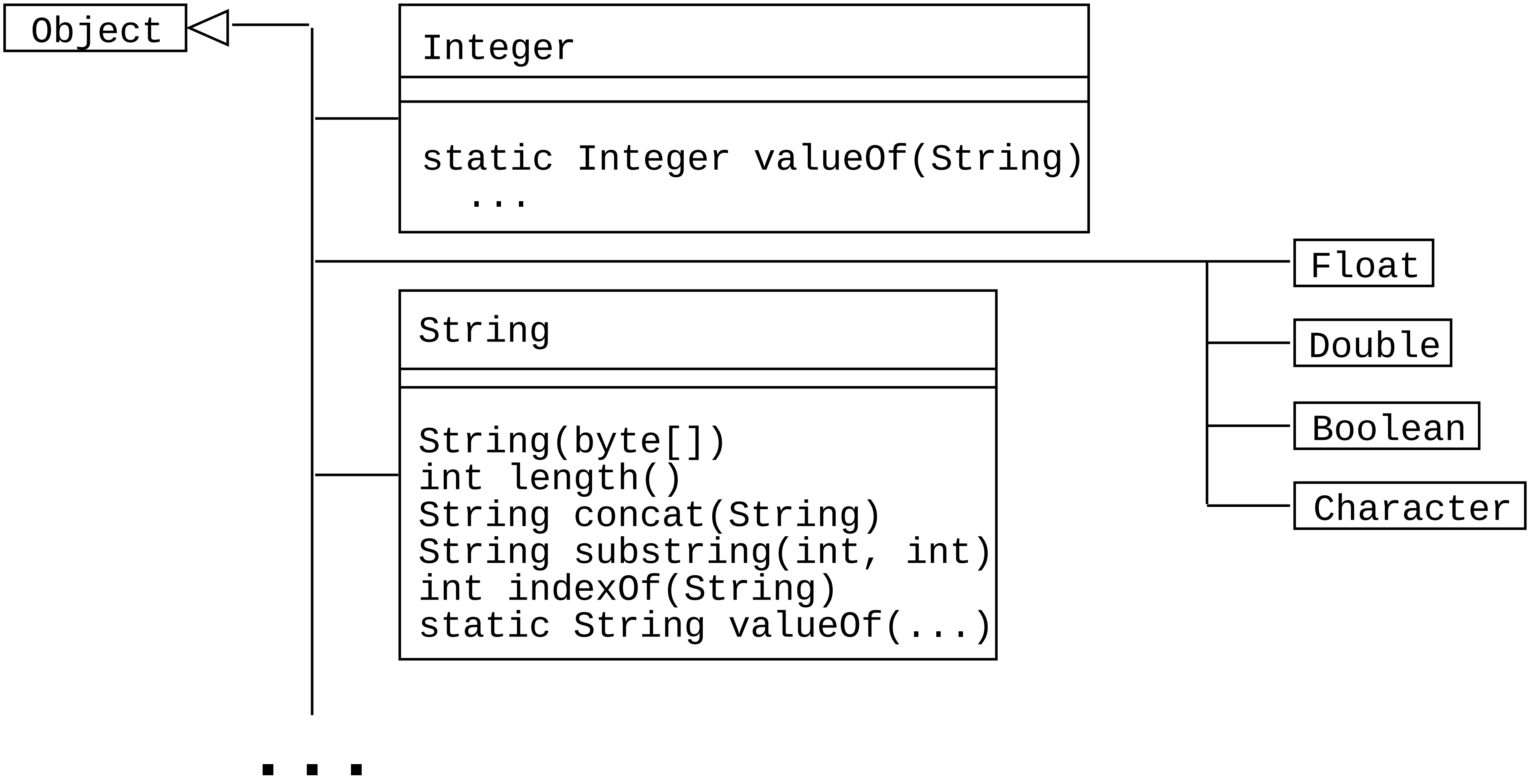
A. Key packages:

1. *java.lang*
2. *java.util*
3. *java.io*

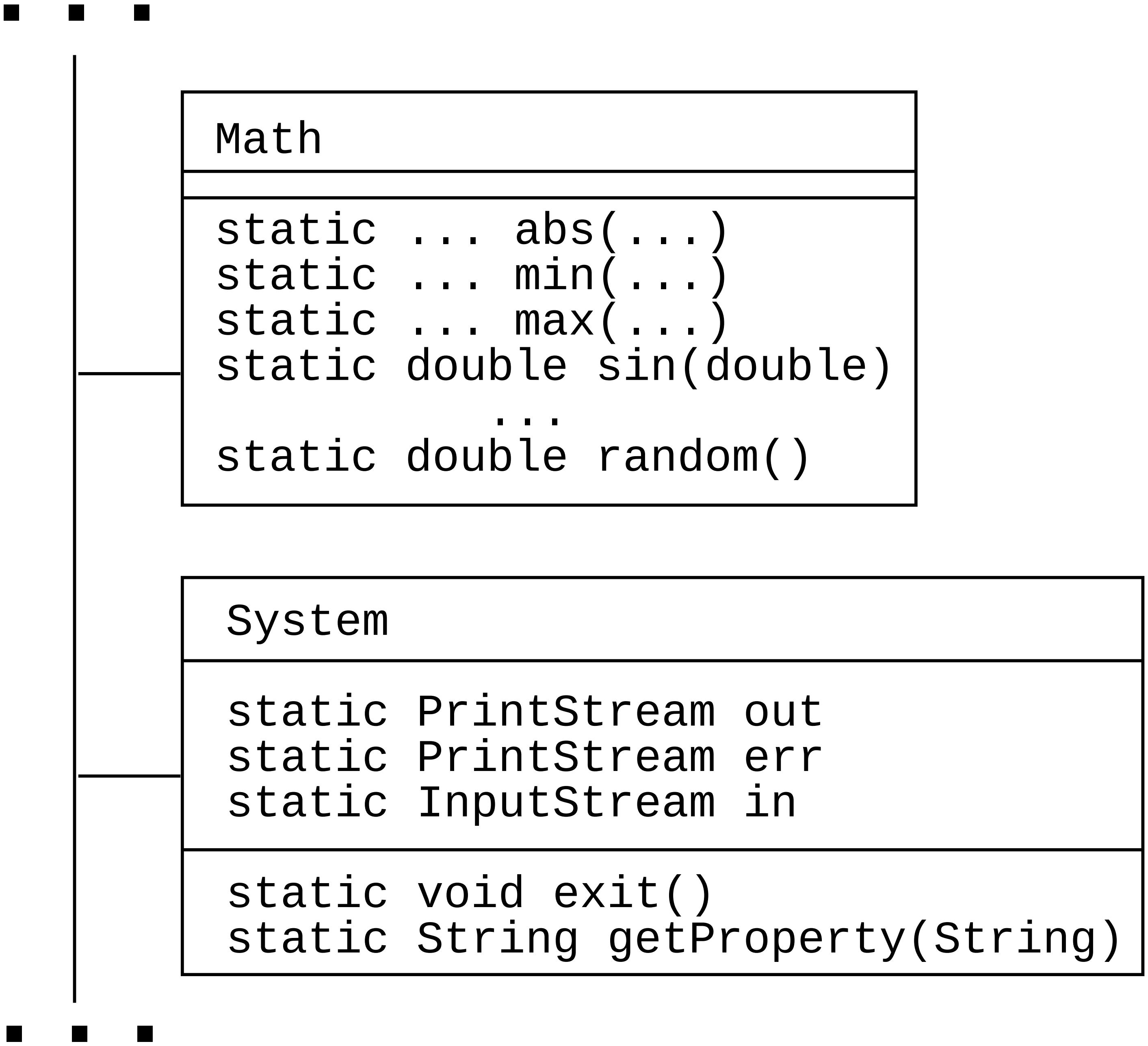
B. Central to work in 307.

C. Summarized in UML diagrams.

D. Package **java.lang**



java.lang, cont’d



. . .

```
Math

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

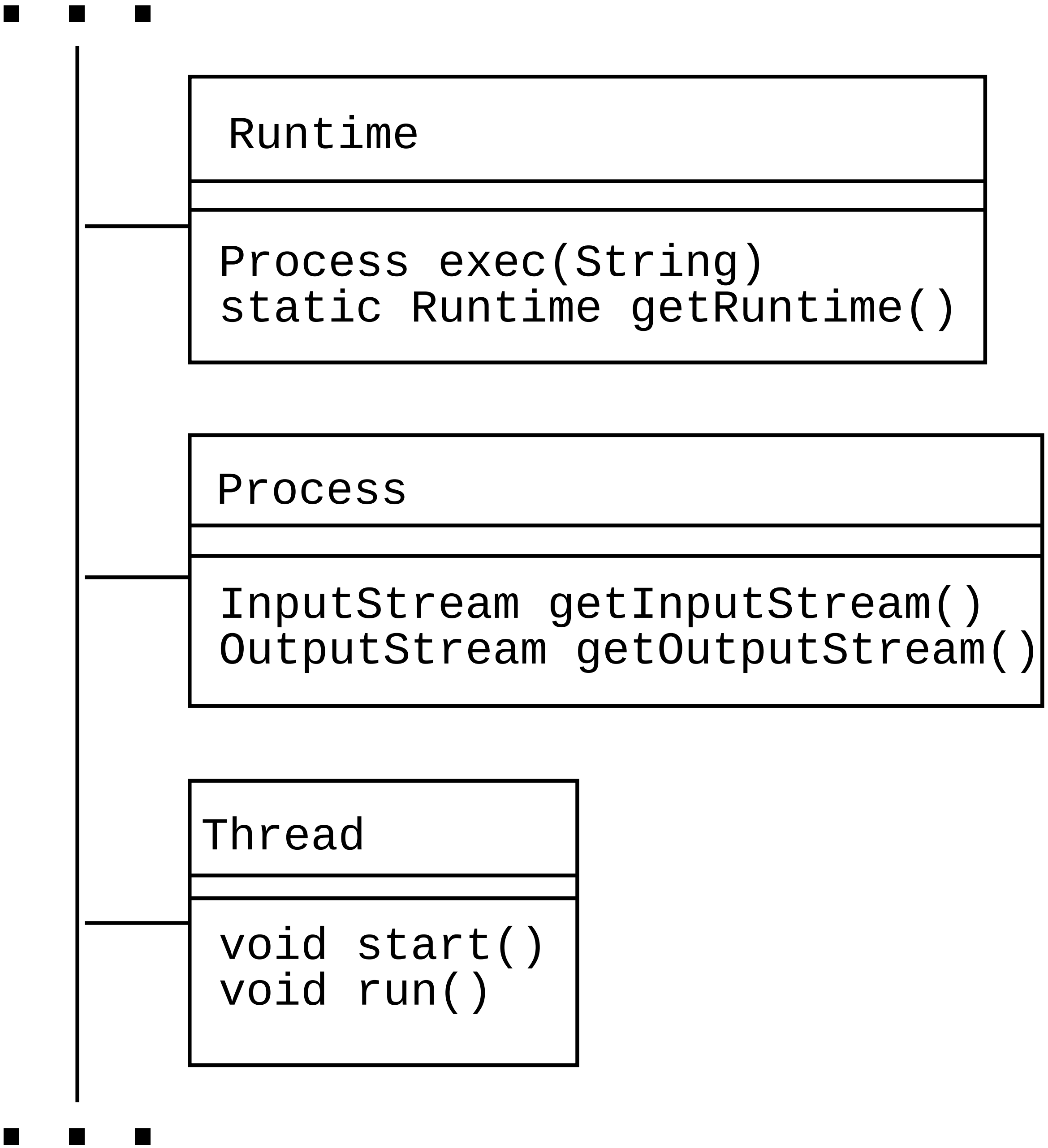
```
System

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

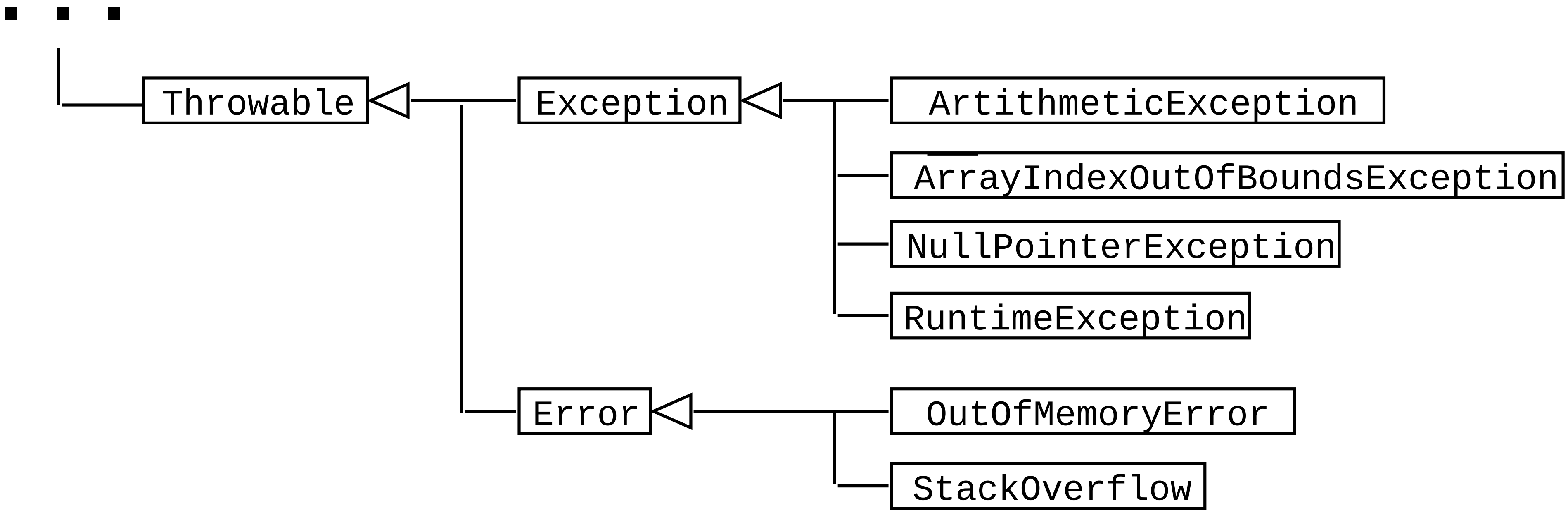
```
InputStream getInputStream()
OutputStream getOutputStream()
```

Thread

```
void start()
void run()
```

. . .

java.lang, cont’d



. . .

Throwable

Exception

ArtithmeticException

ArrayIndexOutOfBoundsException

NullPointerException

RuntimeException

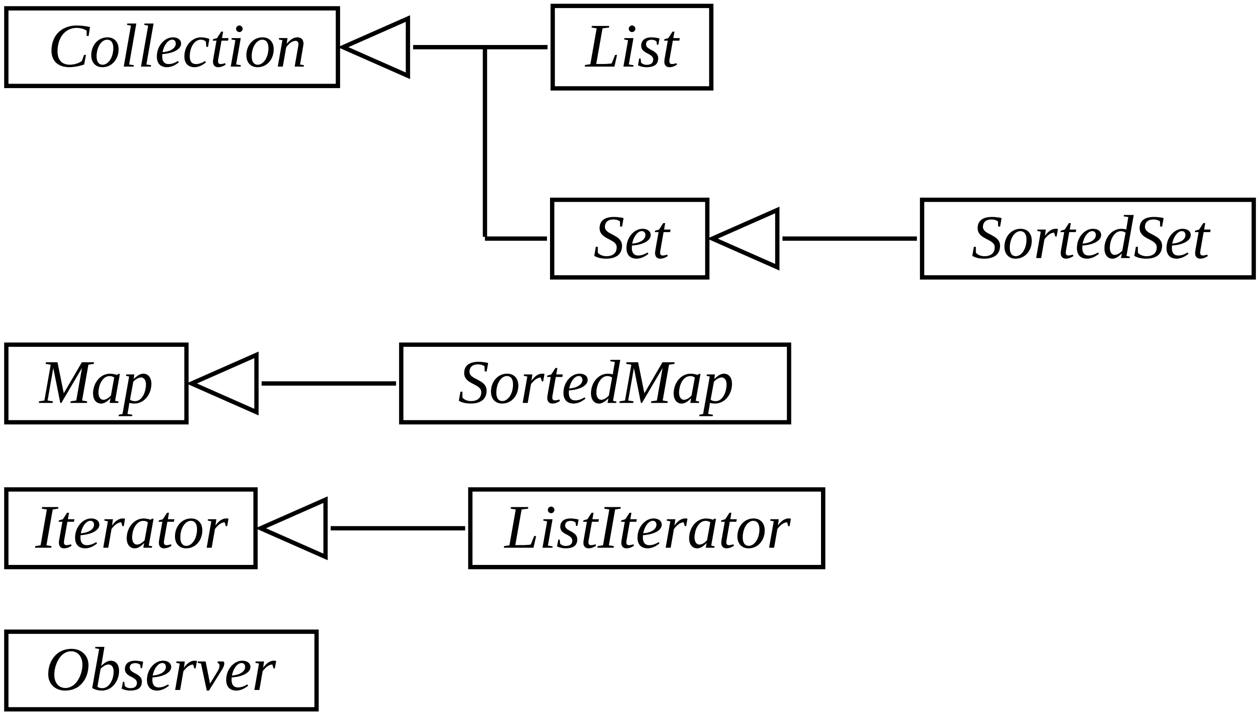
Error

OutOfMemoryError

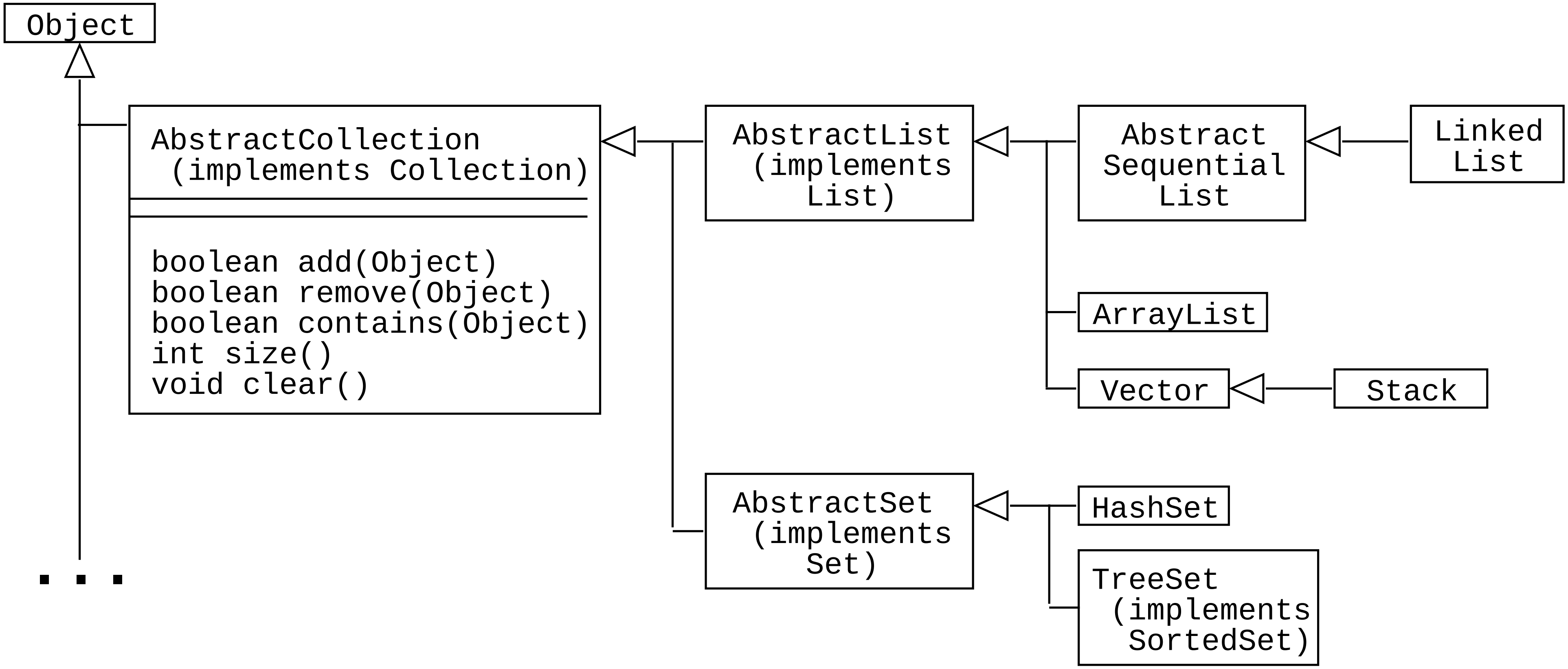
StackOverflow

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

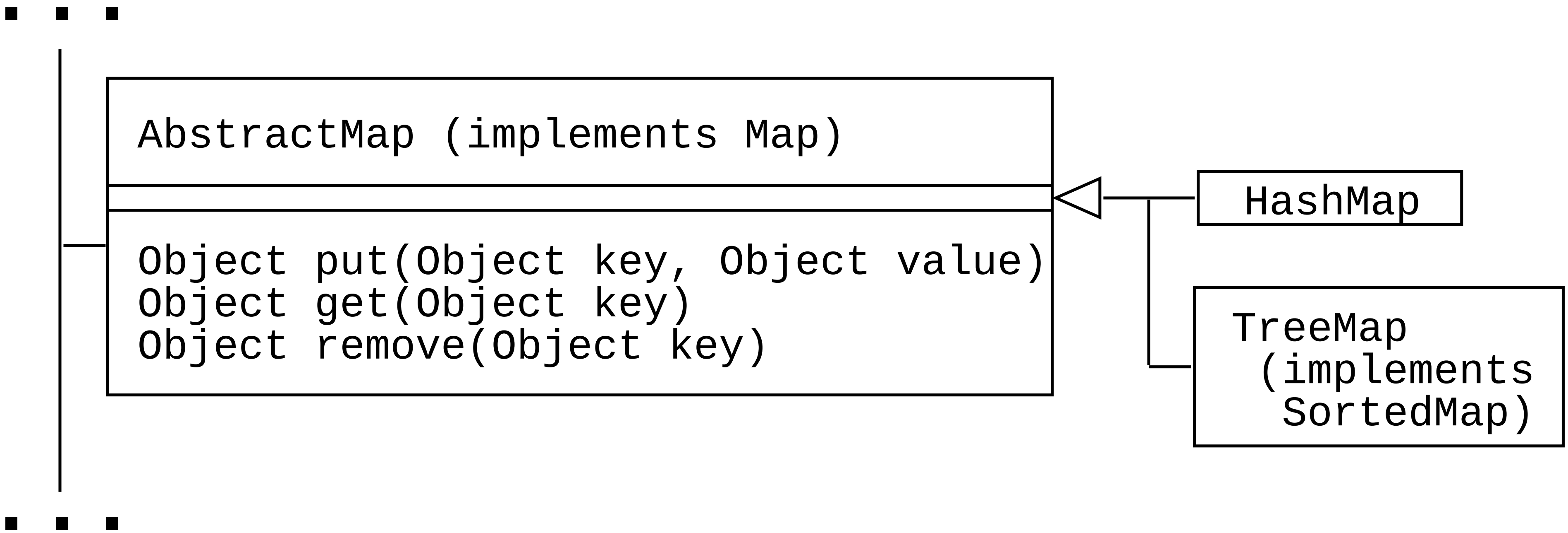
AbstractSet
(implements
Set)

HashSet

TreeSet
(implements
SortedSet)

. . .

java.util, cont’d



. . .

AbstractMap (implements Map)

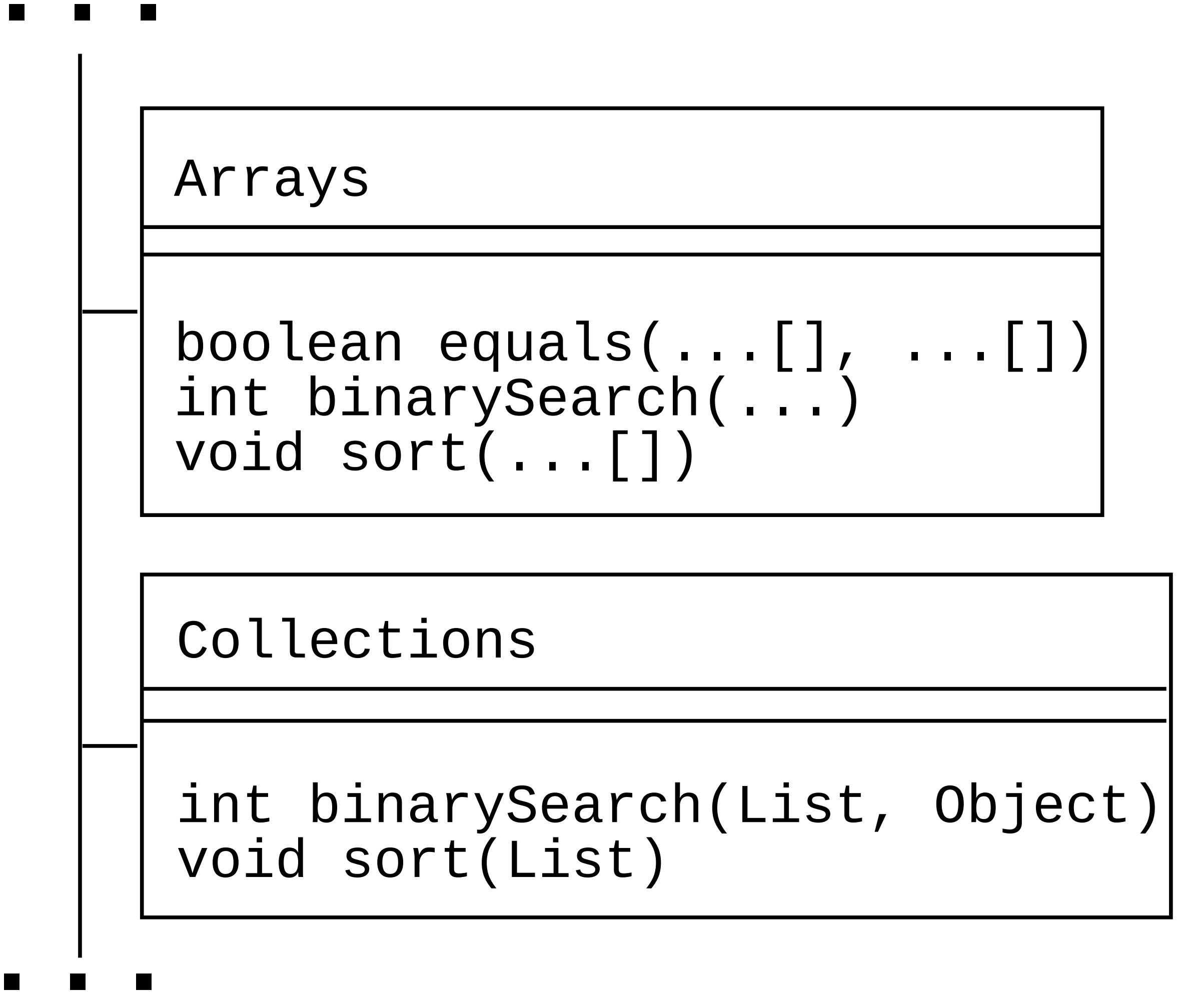
Object put(Object key, Object value)
Object get(Object key)
Object remove(Object key)

HashMap

TreeMap
(implements
SortedMap)

. . .

java.util, cont’d



...

Arrays

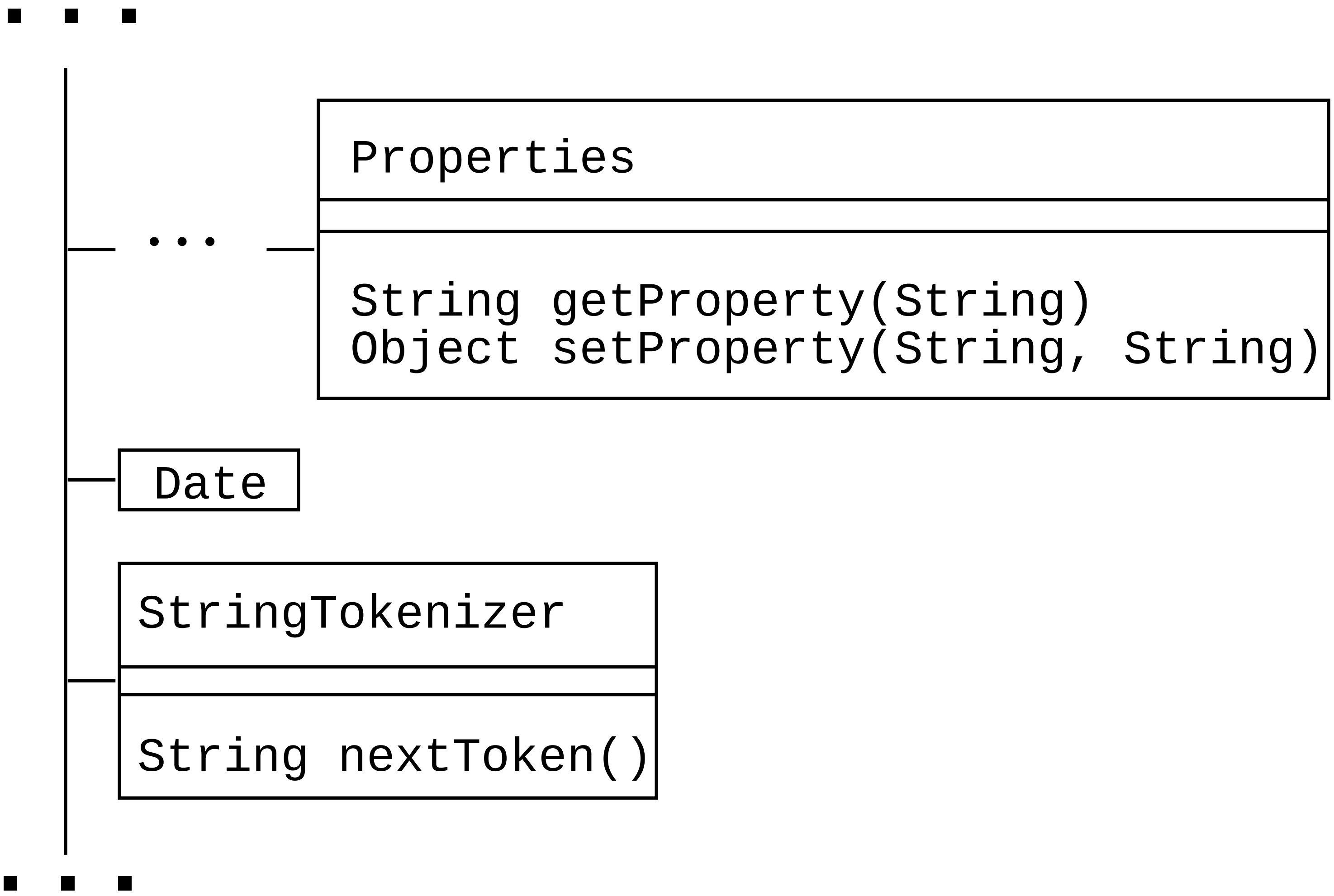
```
boolean equals(...[], ...[])
int binarySearch(...)
void sort(...[])
```

Collections

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

...

java.util, cont’d



. . .

. . .

Properties

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

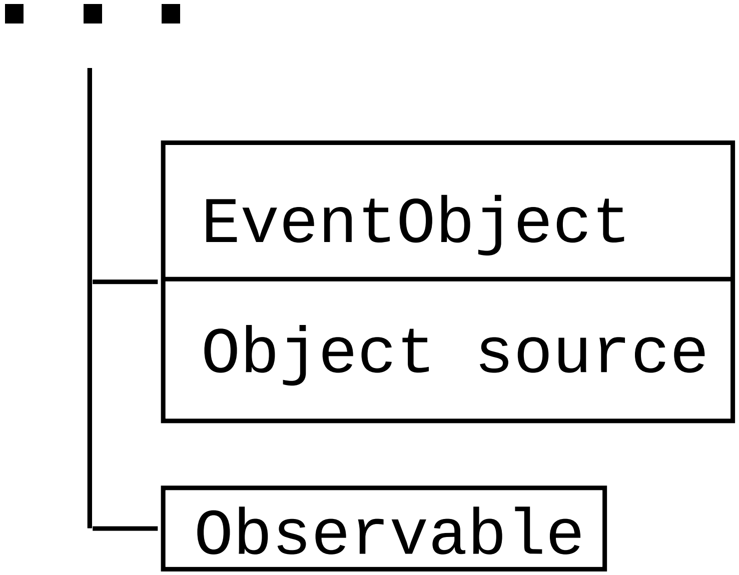
Date

StringTokenizer

String nextToken()

. . .

java.util, cont'd

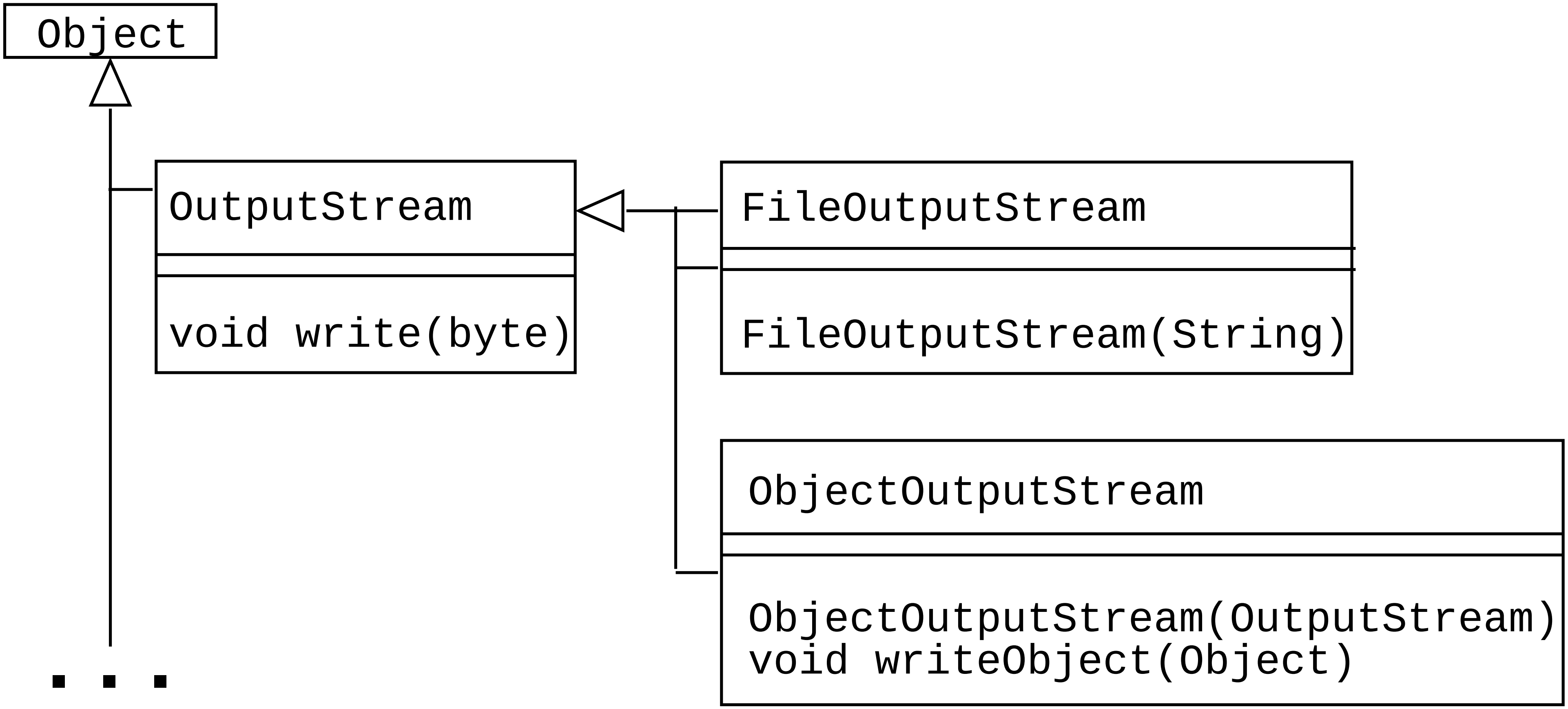


F. Package `java.io`

Interfaces: *Serializable*

• • •

java.io, cont’d



Object

OutputStream

void write(byte)

FileOutputStream

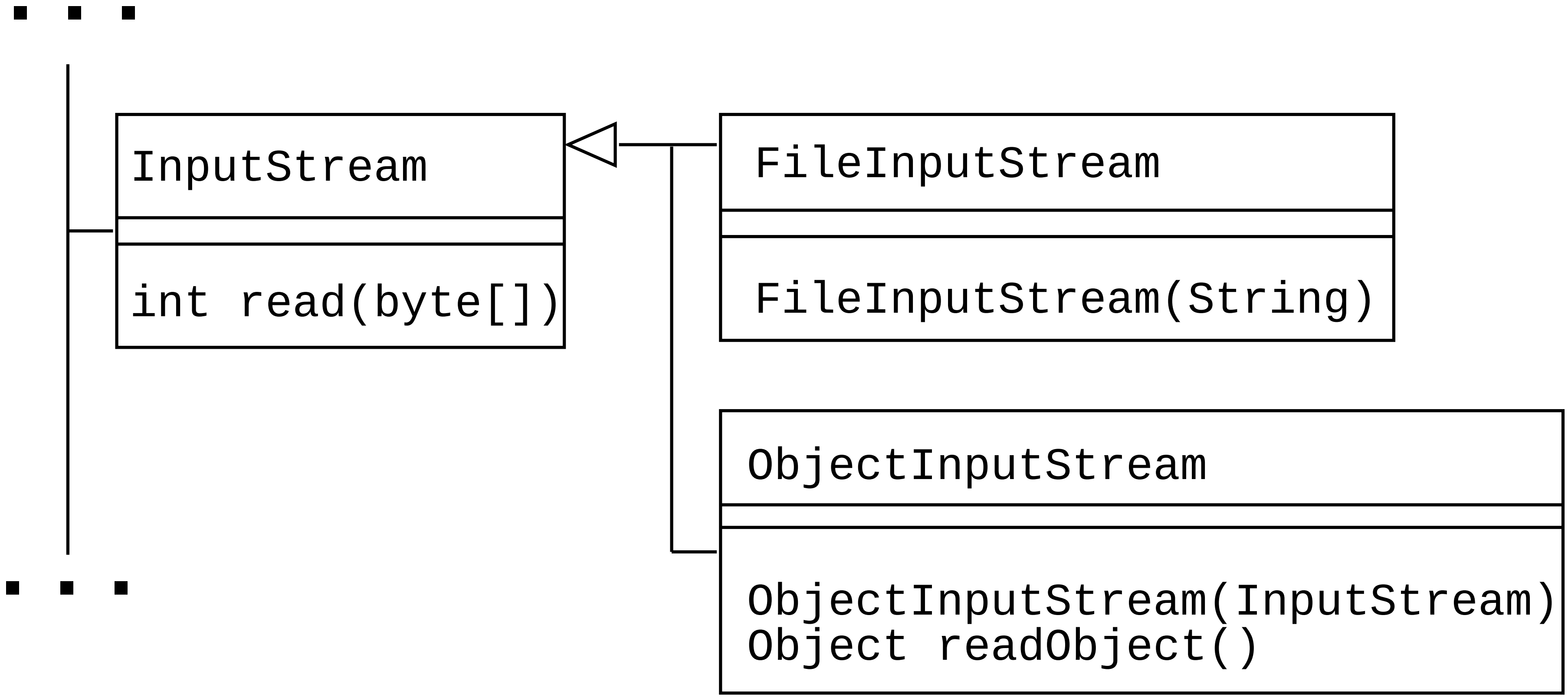
FileOutputStream(String)

ObjectOutputStream

ObjectOutputStream(OutputStream)
void writeObject(Object)

. . .

java.io, cont’d



. . .

InputStream

int read(byte[])

FileInputStream

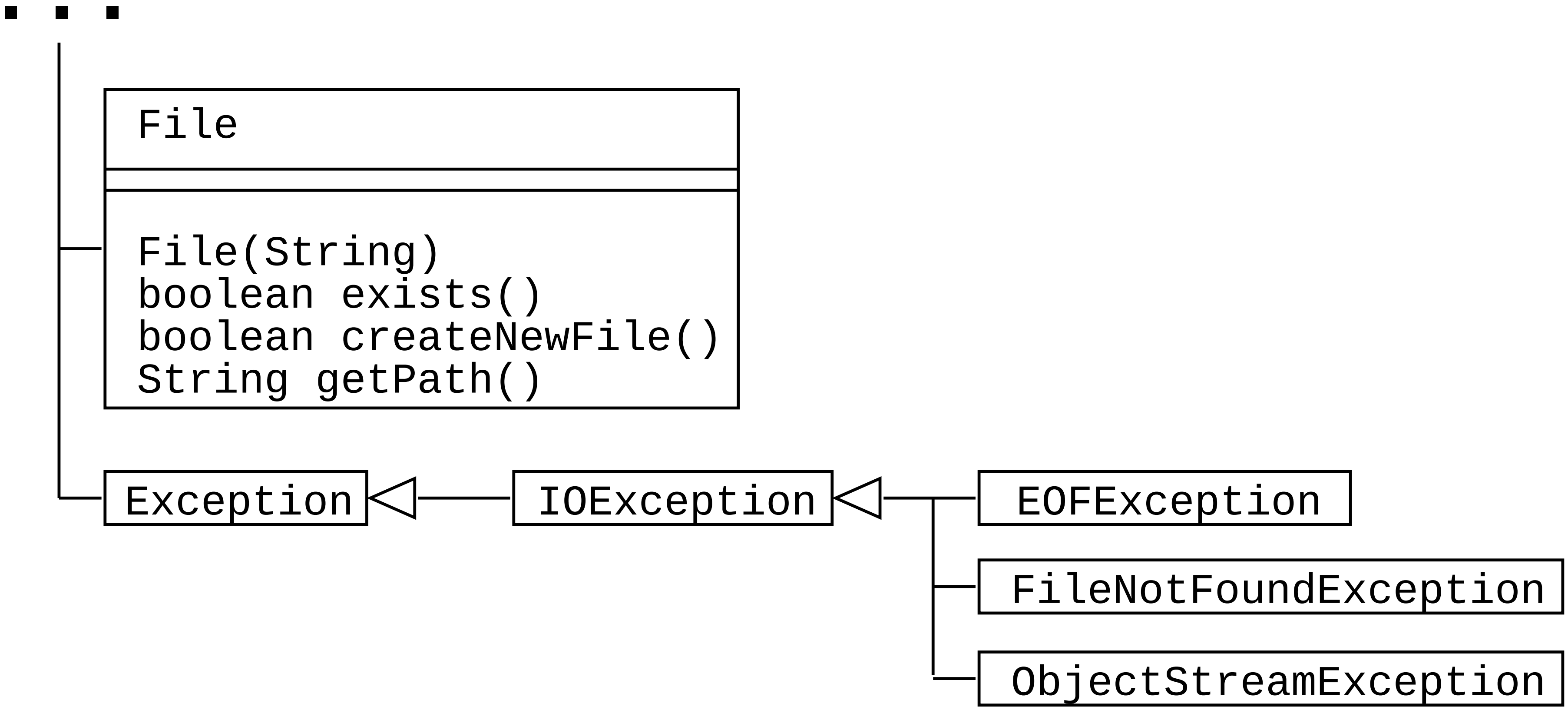
FileInputStream(String)

ObjectInputStream

ObjectInputStream(InputStream)
Object readObject()

. . .

java.io, cont’d



II. Review of "canned" model data.

II. Review of "canned" model data.

A. For initial testing of model/view design.

II. Review of "canned" model data.

A. For initial testing of model/view design.

1. For Milestone 6, it’s entirely "canned".

II. Review of "canned" model data.

A. For initial testing of model/view design.

1. For Milestone 6, it’s entirely "canned".

2. Get concrete examples from requirements.

II. Review of "canned" model data.

A. For initial testing of model/view design.

1. For Milestone 6, it’s entirely "canned".

2. Get concrete examples from requirements.

3. For Milestone 8 we’ll get data from UI and do computation in model.

Canned model data, cont’d

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

Canned model data, cont’d

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

1. E.g., an ***operational*** iterator method.

Canned model data, cont’d

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

1. E.g., an ***operational*** iterator method.

2. Or ***operational*** list-producing method.

Canned model data, cont’d

C. Examples we’ve seen:

Canned model data, cont’d

C. Examples we’ve seen:

1. Milestone 6 `MonthlyAgenda` iterators delivered the same data every time.

Canned model data, cont’d

C. Examples we’ve seen:

1. Milestone 6 `MonthlyAgenda` iterators delivered the same data every time.

2. Milestone 6 `Lists` methods called `generateSampleList()` method that delivered the same list every time

Canned model data, cont'd

C. Examples we've seen:

1. Milestone 6 `MonthlyAgenda` iterators delivered the same data every time.

2. Milestone 6 `Lists` methods called `generateSampleList()` method that delivered the same list every time

3. For Milesteone 8, all model methods generate data ***using a real calendar***.

III. View data collection and validation.

III. View data collection and validation.

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

III. 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:

View data collection, cont’d

C. Once raw data are collected they are:

1. Converted by Model, from their raw form.

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.

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.

IV. Exception handling in data validation

IV. Exception handling in data validation

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

IV. 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"*.

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.

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.

Exception handling in data validation, cont’d

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

D. We’ll now discuss this.

V. Quick review of exception handling.

V. Quick review of exception handling.

A. Normally, method returns to caller.

V. 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.

Review of exception handling, cont’d

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

2. Return to nearest method that does catch.

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.

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.

Review of exception handling, cont’d

C. Different languages provide different styles.

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

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.

VI. Design diagram notation

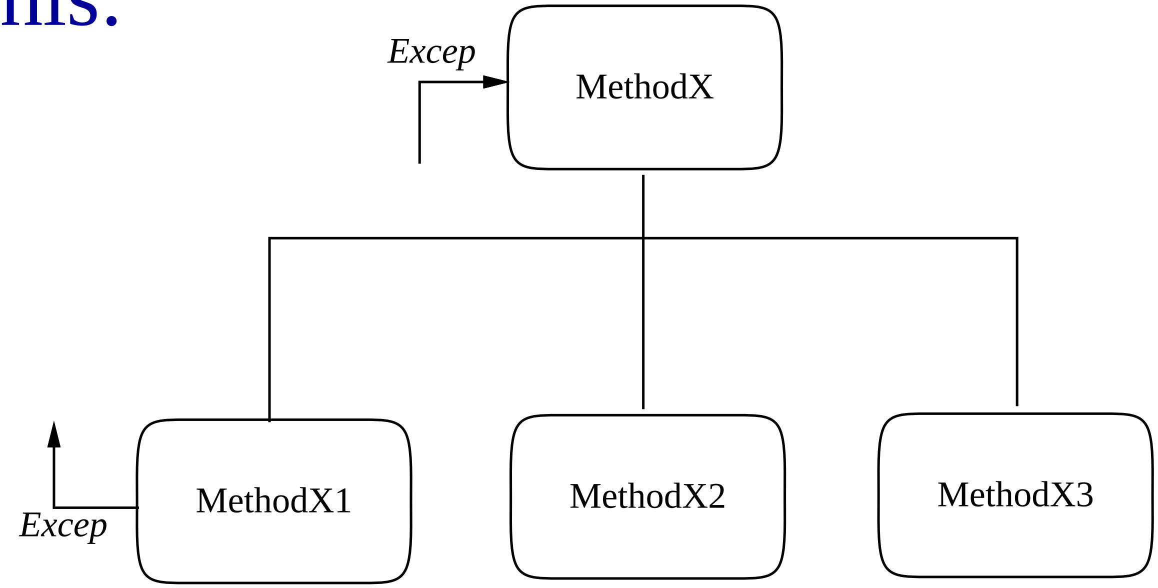
VI. Design diagram notation

A. Shown with labeled arrows.

VI. Design diagram notation

A. Shown with labeled arrows.

B. Like this:



Exception handling, cont’d

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

Exception handling, cont’d

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

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

Exception handling, cont’d

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

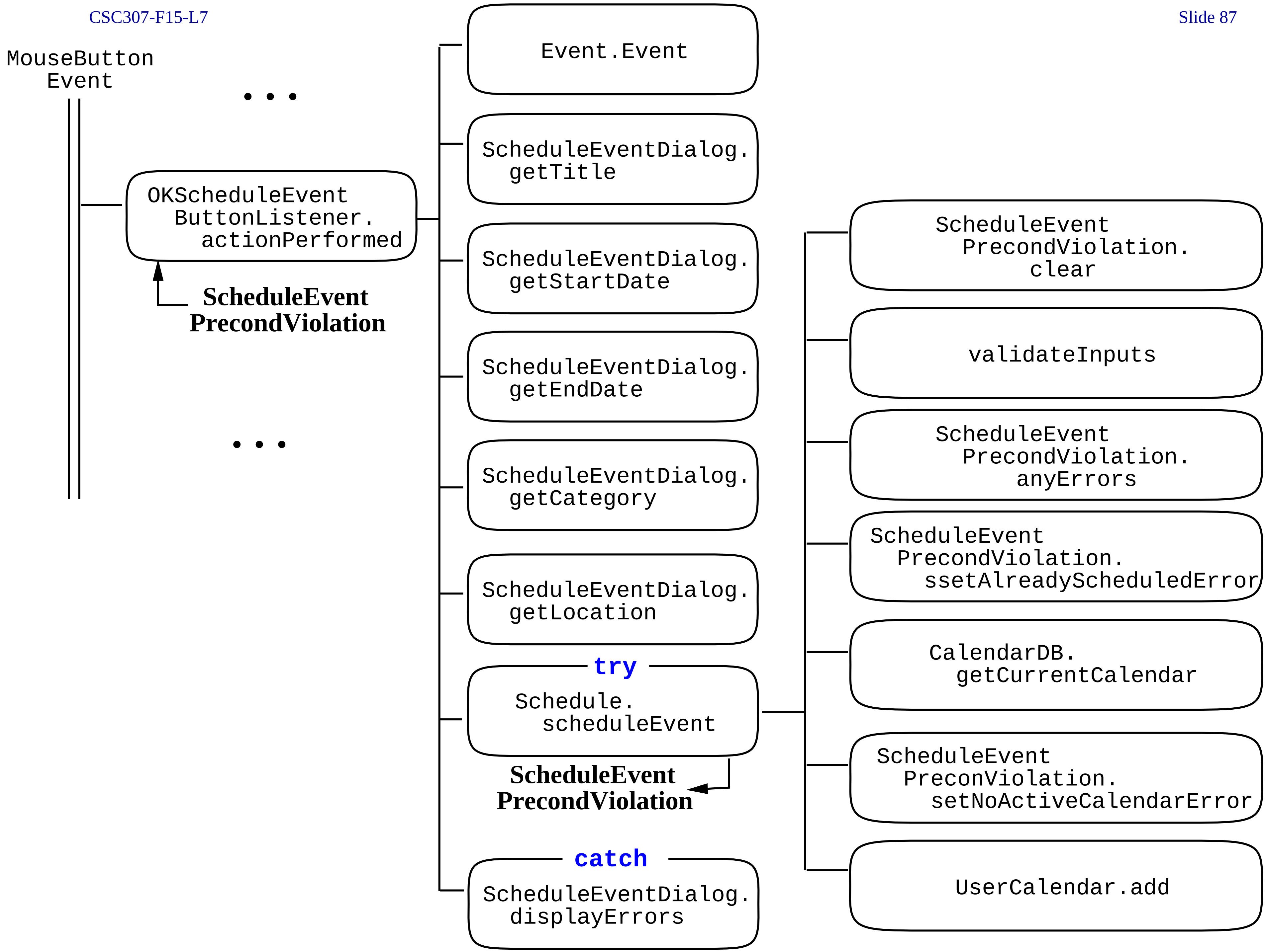
2. `X2` and `X3` return in normal way.

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

VII. Example Model-View Communication

VII. Example Model-View Communication

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.

Example, cont’d

B. Model throws to view.

C. Throw when input errors detected.

Example, cont’d

B. Model throws to view.

C. Throw when input errors detected.

D. See code for

```
OKScheduleEventButtonListener.
    actionPerformed()
```

VIII. A key design pattern -- Observer/Observable.

VIII. A key design pattern -- Observer/Observable.

A. Useful when multiple views change, based on changing model, e.g.,

VIII. A key design pattern -- Observer/Observable.

A. Useful when multiple views change, based on changing model, e.g.,

1. ***CalTool:*** daily, weekly, monthly views

VIII. A key design pattern -- Observer/Observable.

A. Useful when multiple views change, based on changing model, e.g.,

1. ***CalTool:*** daily, weekly, monthly views
2. ***Testtool:*** all UIs that display questions and tests in some form

B. Java's *Observer* interface.

```
interface Observer {

  public void update(
    Observable o,
    Object arg)

}
```

C. Java’s *Observable* class.

```
class Observable {
    void addObserver(Observer o)
    void setChanged()
    boolean hasChanged()
    void notifyObservers()
    void notifyObservers(Object arg)
}
```

D. Typical usage

```
class model extends Observable { ... }
class View implements Observer { ... }

class UserCalendar extends Model {

     . . .

    public void add(ScheduledItem item) {

        . . .

        items.add(item);
        setChanged();
    }
}
```

```
public class OKScheduleEVentButtonListener
        implements ActionListener {

    public void actionPerformed() {

        . . .

        userCalendar.add(...);
        userCalendar.notifyObservers();
    }
}
```

```
public class MonthlyAgenda extends View {
    public MonthlyAgenda(
        UserCalendar userCalendar) {

        . . .

        userCalendar.addObserver(this)
    }

    public void update(Observable o,
            Object arg) {

        /* Get items from model ... */

    }
}
```