Chapter 15

Object-Oriented Testing

Issues in Testing Object-Oriented Software

- How is object-oriented (o-o) software different/special?
  - Need to clarify unit
  - Composition vs. decomposition
  - No reason for integration testing based on functional decomposition tree
  - Can still use the (better) idea of Call Graph based integration
- Properties of an o-o programming language
  - Inheritance
  - Encapsulation
  - Polymorphism
- Message communication
  - Message quiescence
  - Event quiescence

Levels of Object-Oriented Testing

Level	Item	Boundary
Unit	Method of an object? Class?	Program graph
Integration	MM-Path	Message quiescence
	Atomic System Function	Event quiescence
System	Thread	Source to sink ASF
	Thread Interaction	(none)

Notice the cascading levels of interaction:

- unit testing covers statement interaction,
- MM-Path testing covers method interaction,
- ASF testing covers MM-Path interaction,
- thread testing covers object interaction, and all of this culminates in
- thread interaction.

Re-usable Testing Techniques

Level	Item	Technique
Unit	Method of an object	Traditional specification and/or code based
	Class	StateChart-based
Integration	MM-Path	New definition?
	Atomic System Function	New definition?
System	Thread	New definition? (StateCharts)
	Thread Interaction	(as before)

Units in Object-Oriented Software

- Guidelines for units
  - A unit is the smallest software component that can be compiled and executed. (traditional view, usually in procedural code)
  - A unit is a software component that would never be assigned to more than one designer to develop. (project management view, appropriate for both procedural and o-o software)
  - Unit developer does detailed design, coding, and unit level testing
- Candidate o-o units
  - A single method
  - A class (generally accepted view)

Single Methods as Units

- Appropriate for large classes
- Consistent with the one designer/one unit practice
- Single method testing
  - reduces to unit testing for procedural units
  - o-o units are typically less complex (cyclomatic) than procedural units
- This choice mandates "intra-class" integration testing.

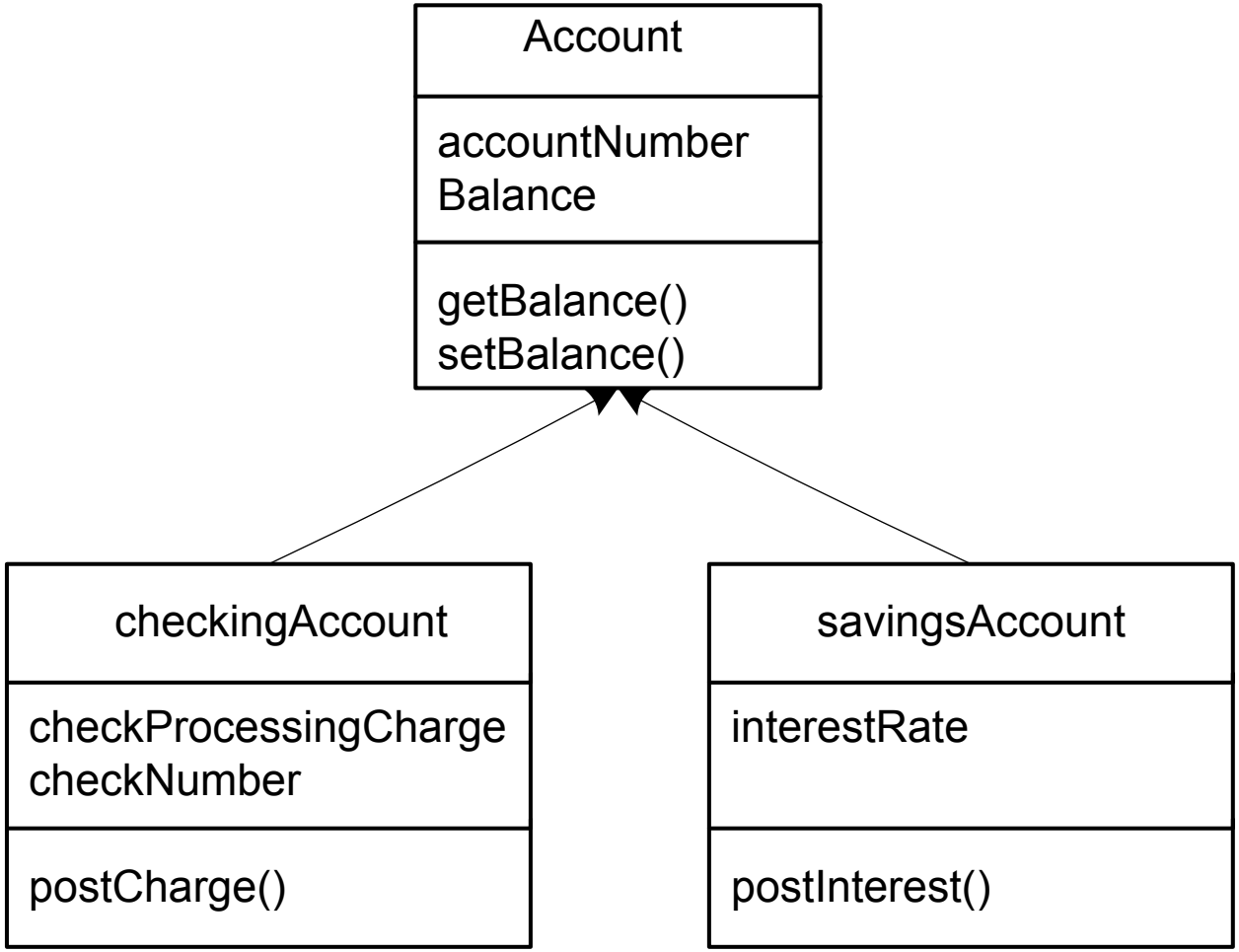
Classes as Units

- Generally accepted view
- In UML, the *de facto* behavioral model for a class is a StateChart.
- BUT, StateChart composition is possible only in extremely simple cases.
- Testing complexity shifts from unit level to integration level

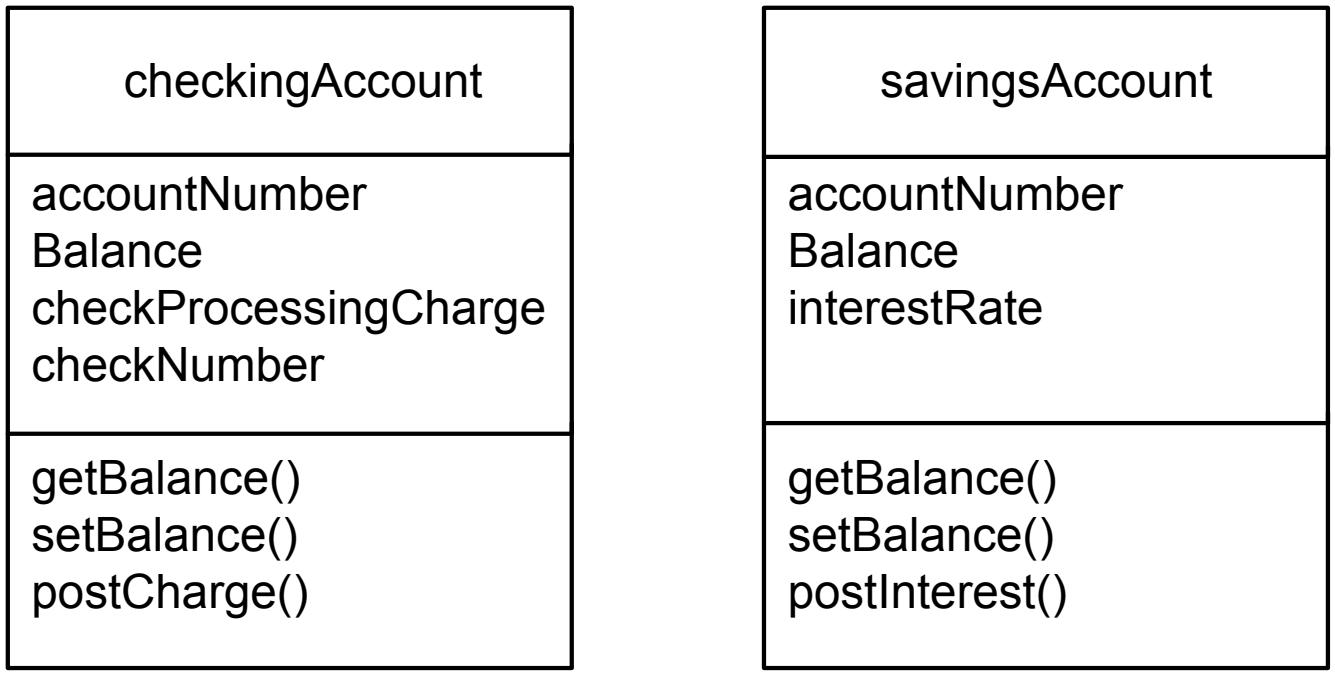
Composition and Encapsulation

- Composition
  - normal approach for o-o design
  - mandates strong unit level testing
  - traditional precepts of coupling and cohesion are good practice for o-o units
- Encapsulation
  - based on coupling and cohesion
  - shifts complexity and testing to integration level

Implications of Inheritance



Implications of Inheritance—Flattened Classes

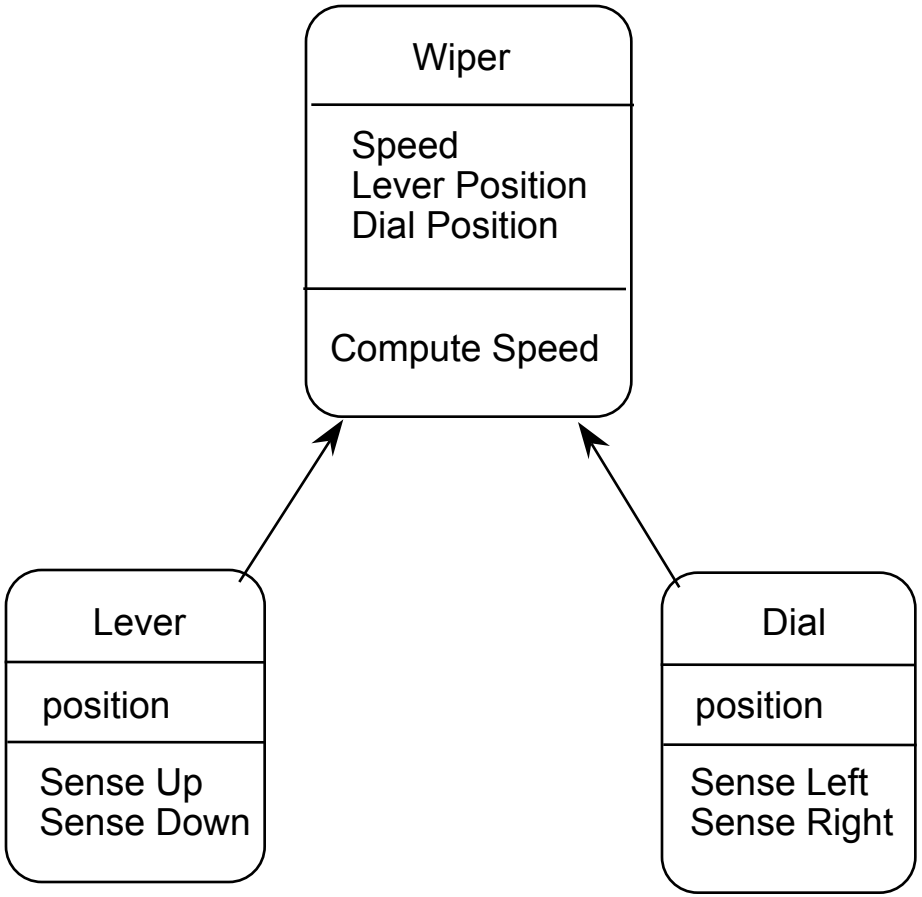


Example—O-O Windshield Wiper

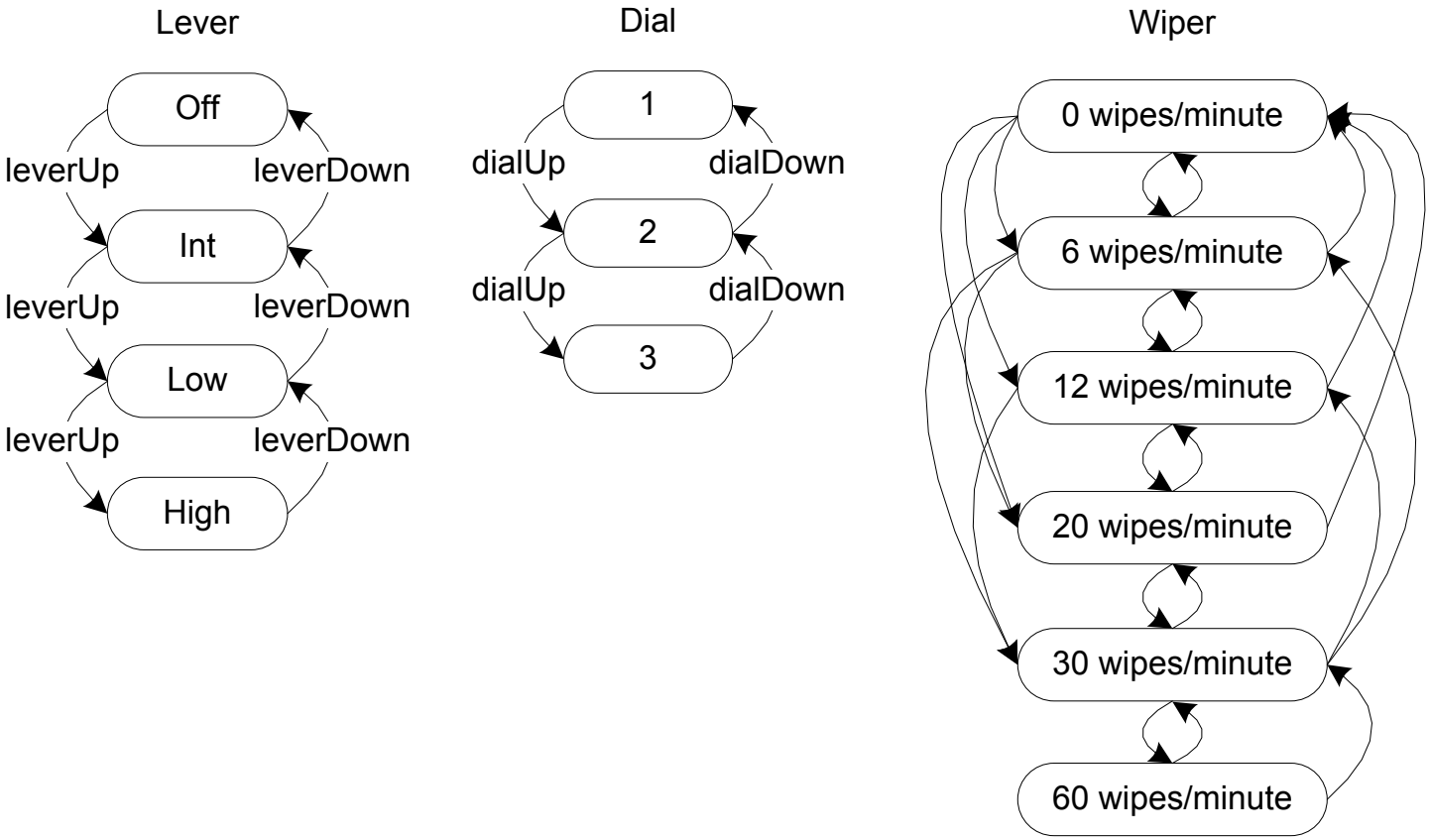
```
Class lever(leverPosition;
                private senseLeverUp(),
                private senseLeverDown() )
Class dial(dialPosition;
                private senseDialUp(),
                private senseDialDown() )
Class wiper(wiperSpeed;
                setWiperSpeed(newSpeed) )
```

Which unit controls the interaction between the Lever and the Dial?

Saturn Windshield Wiper Objects



Windshield Wiper Class Behavior



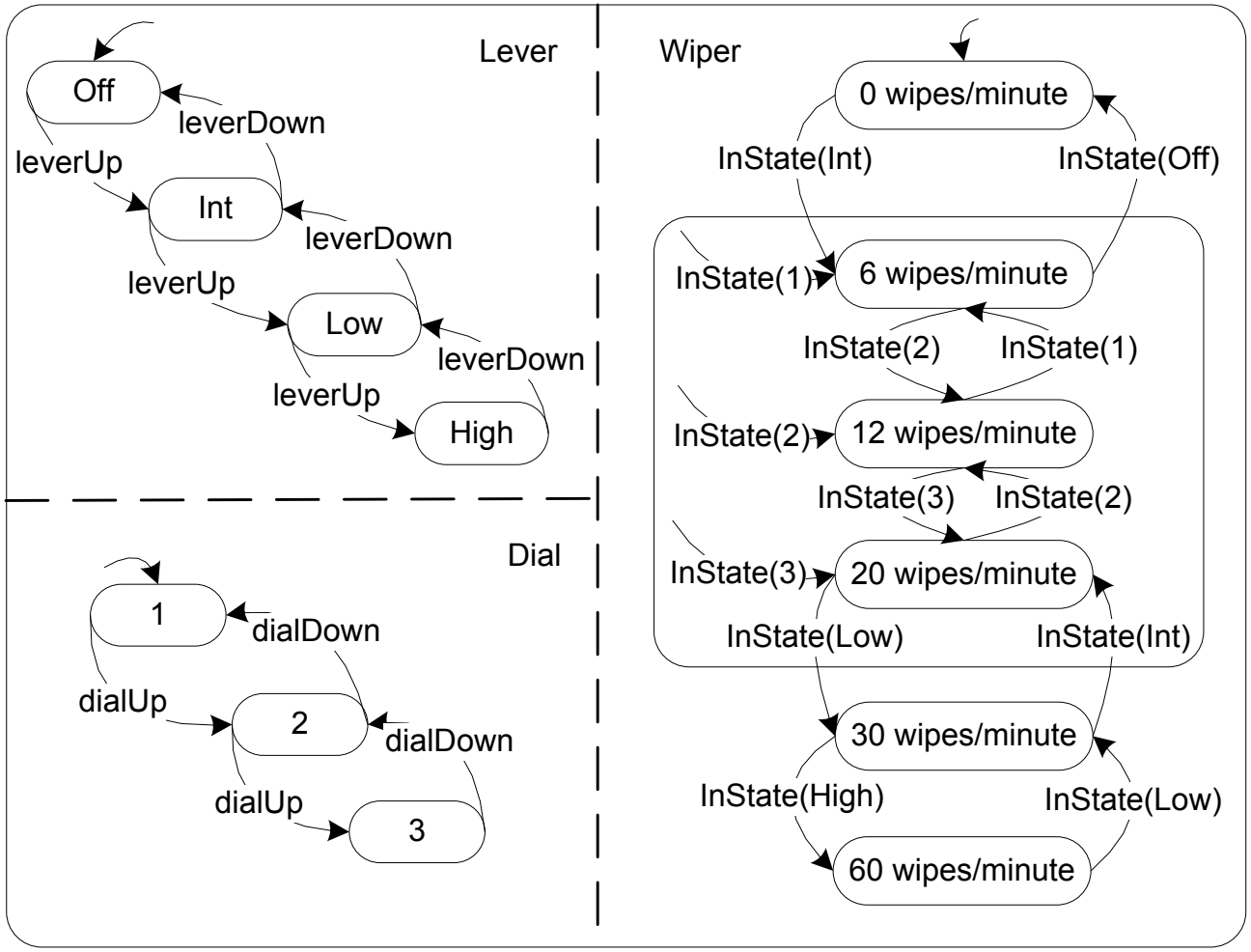
Unit Testing for Windshield Wiper Classes

- Can test only the Lever and Dial classes directly.
- Test cases to make sure event-oriented methods make correct changes to the position attributes.
- Need "mock objects" (o-o analog of drivers for procedural code) to provide inputs to test the wiper class.
- Note how the "line" between unit and integration testing is blurred!

Sample Unit Test Case for Lever

Test Case Name	Test Sense Up method
Test Case ID	Lever-1
Description	Verify that the LeverUp method correctly changes the value of the position attribute
Pre-Conditions	1. position = Off
Input(s)	Expected Output(s)
1. move lever to Int	2. position = Int
3. move lever to Low	4. position = Int
5. move lever to High	6. position = Int
Post condition(s)	1. position = High
Test Result?	Pass/Fail
Test operator	Paul Jorgensen
Test Date	Dec. 8, 2013

Windshield Wiper StateChart



StateChart-Based Testing for the Windshield Wiper

- Need an engine to execute the StateChart
  - execute "interesting" scenarios
  - save paths as skeletons of system test cases
- The InState events show how the Wiper class can be tested once the Lever and Dial classes have been tested.

Sample System Level Test Case

Test Case Name	Exercise all wiper speeds
Test Case ID	WindshieldWiper-1
Description	The windshield wiper is in the OFF position, and the Dial is at the 1 position; the user moves the lever to INT, and then moves the dial first to 2 and then to 3; the user then moves the lever to LOW.; the user moves the lever to INT, and then to OFF
Pre-Conditions	1. The Lever is in the OFF position
	2. The Dial is at the 1 position
	3. The wiper speed is 0
Input events	Output events
1. move lever to INT	2. speed is 6
3. move dial to 2	4. speed is 12
5. move dial to 3	6. speed is 20
7. move lever to LOW	8. speed is 30
9. move lever to HIGH	10. speed is 60
Post conditions	1. The Lever is in the HIGH position
	2. The Dial is at the 3 position
	3. The wiper speed is 60

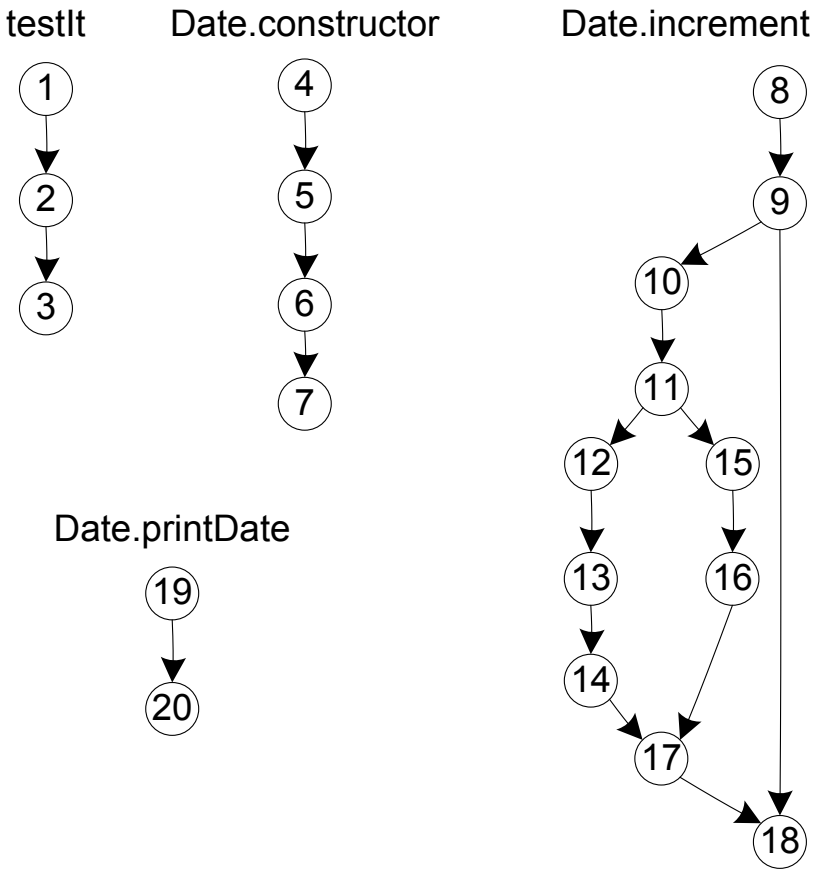
Integration Testing for O-O Software

- In general, o-o methods have low cyclomatic complexity, BUT
- The complexity shifts to the integration level.
- Possibilities
  - Call Graph based integration (very similar to procedural code strategies—pairwise or neighborhood)
  - MM-Paths for o-o code
- Not much help from UML
- UML collaboration diagram predisposes pairwise integration among classes

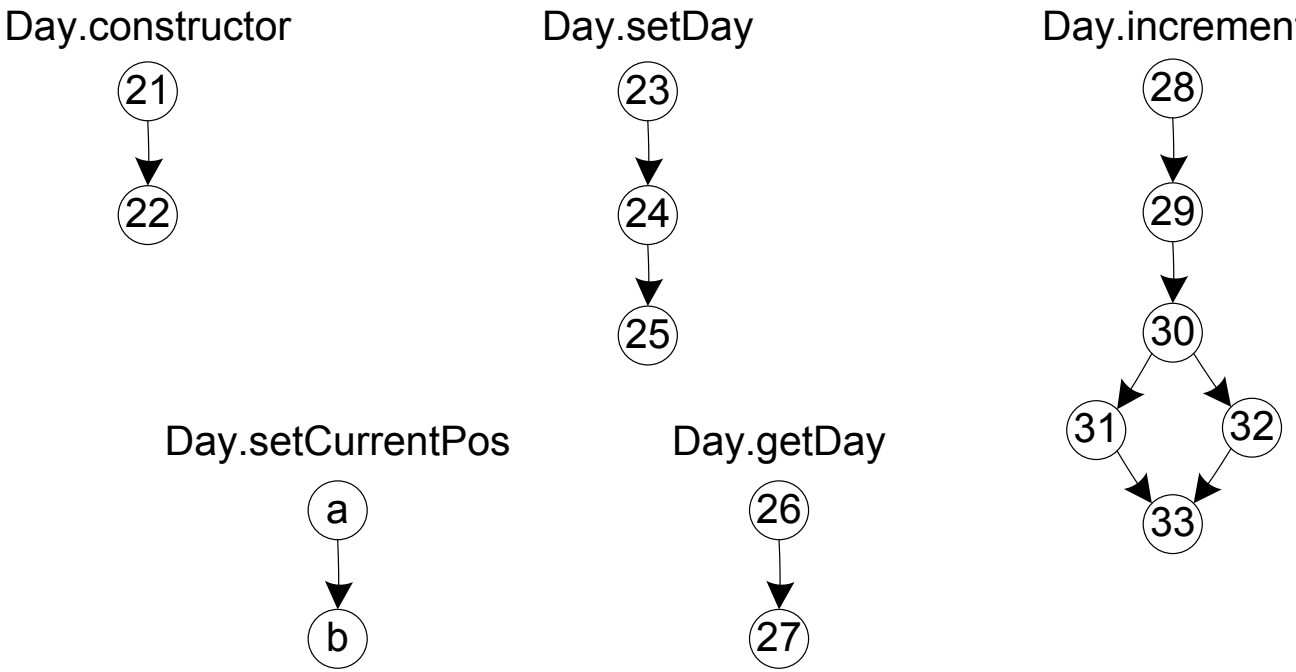
Testing Object-Oriented NextDate

- Example—our familiar NextDate
- Rewritten here in object-oriented style
- One Abstract Class—CalendarUnit
- Classes
  - testIt
  - Date
  - Day
  - Month
  - Year
- (See text for pseudo-code)
- Program graphs in next few slides

Program Graphs of Date Methods

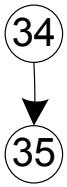


Program Graphs of Day Methods



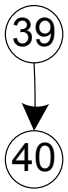
Program Graphs of Month Methods

Month.constructor



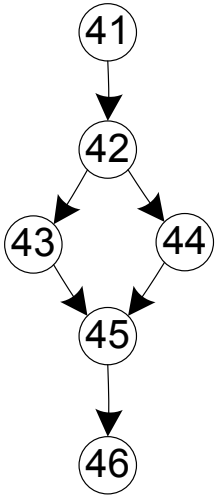
Month.setCurrentPos

Month.getMonth

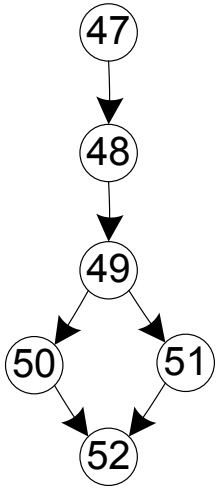


Month.setMonth

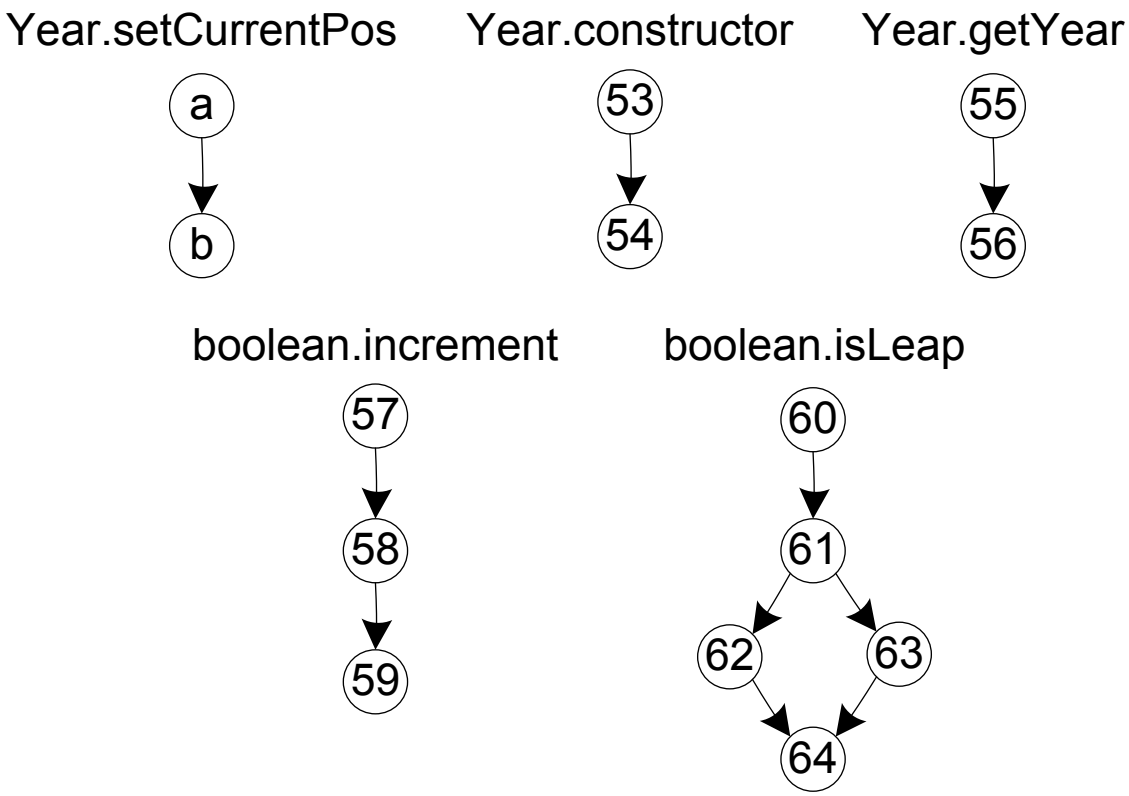
Month.getMonthsize



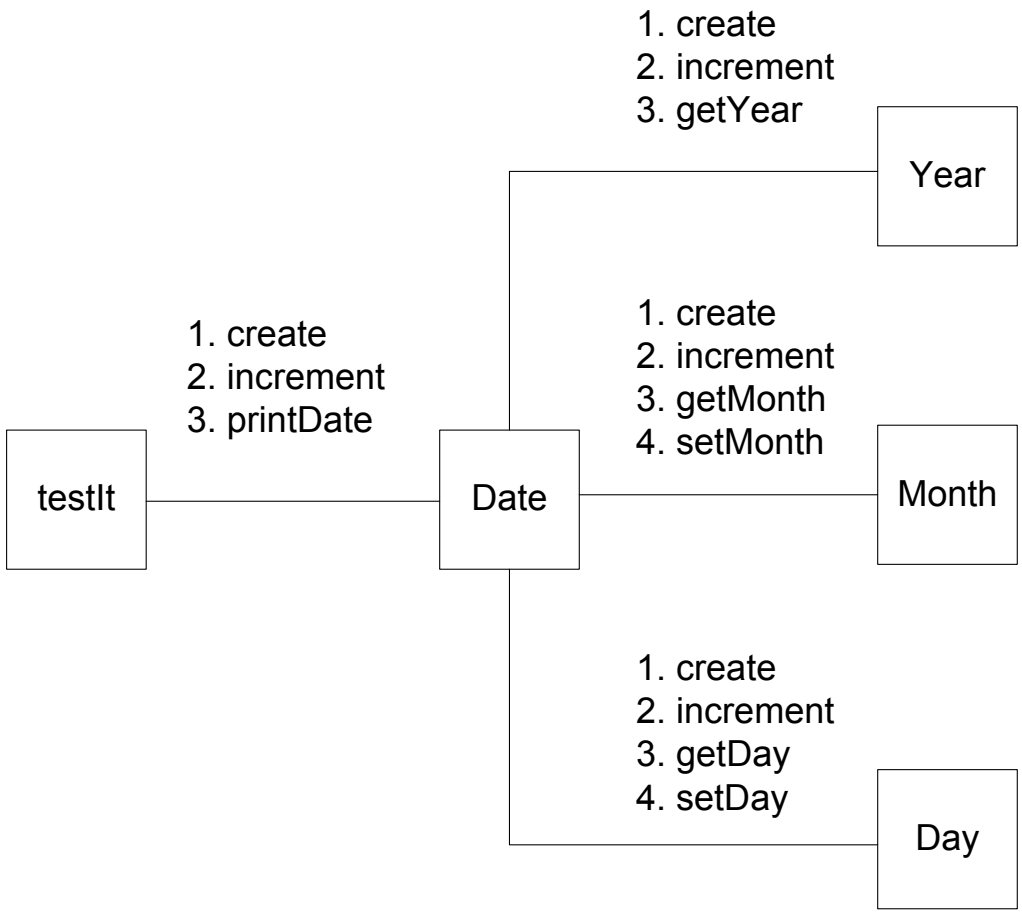
boolean.increment



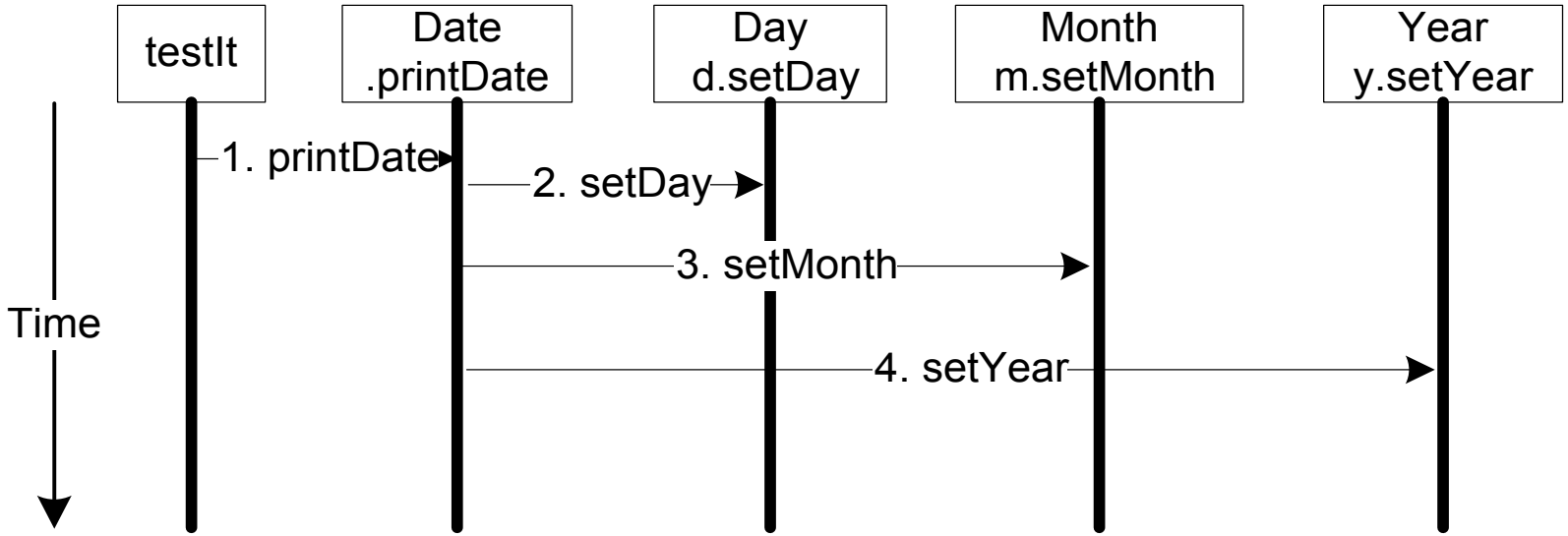
Program Graphs of Year Methods



ooNextDate Collaboration Diagram



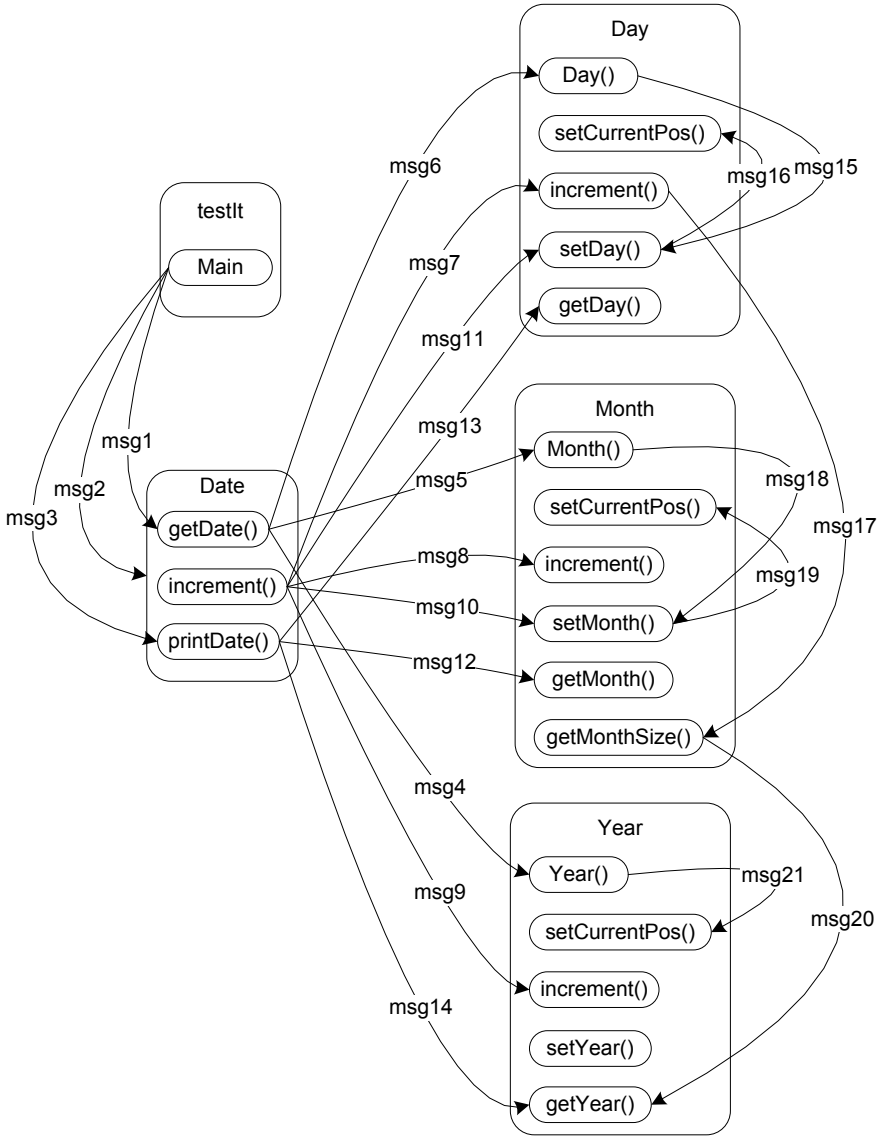
printDate Sequence Diagram



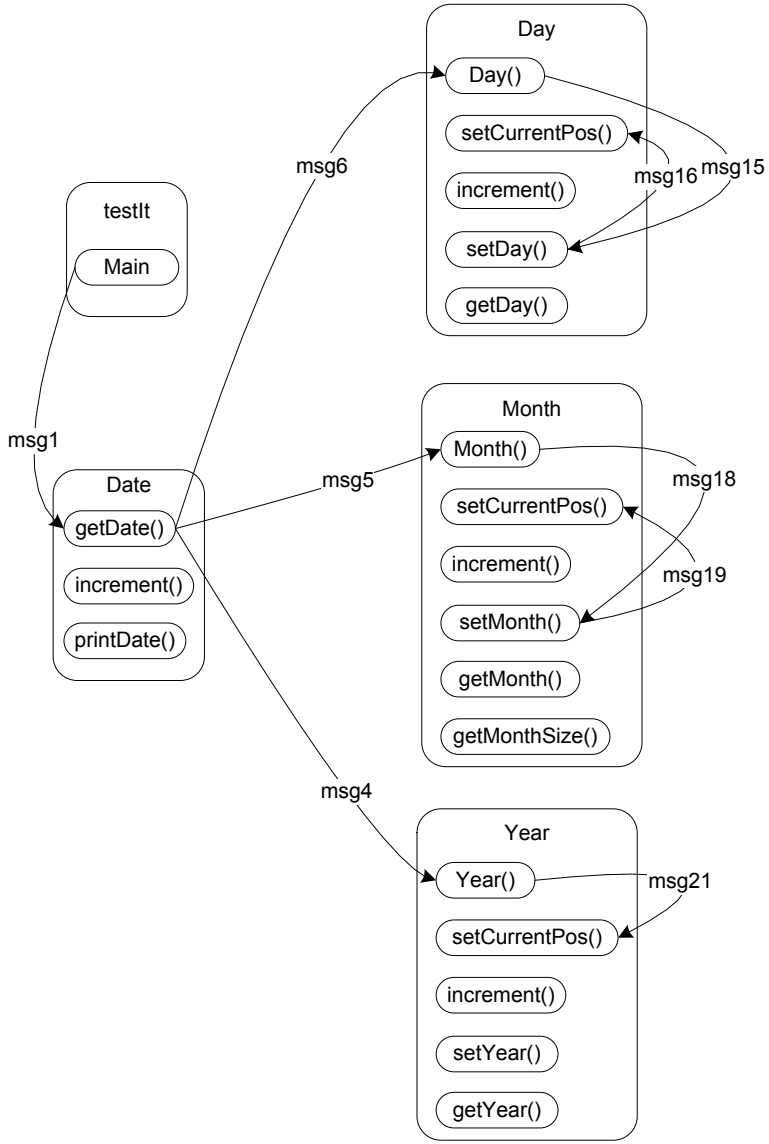
MM-Paths for O-O Software

- Definition: An object-oriented MM-Path is a sequence of method executions linked by messages.
- Call Graph slightly revised
  - nodes are methods
  - edges are messages
- Next three slides show sample Call Graphs for ooNextDate

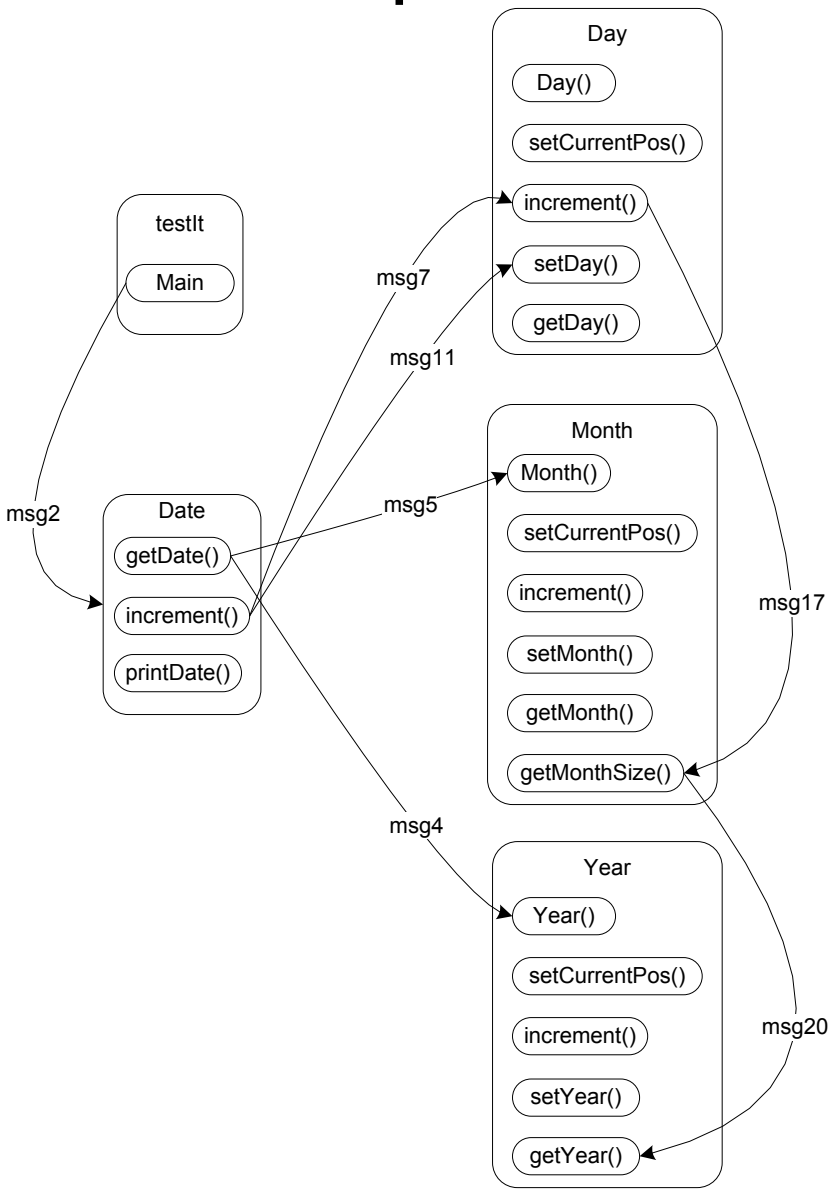
All MM-Paths as a Call Graph



MM-Path for January 3, 2013



MM-Path for April 30, 2013



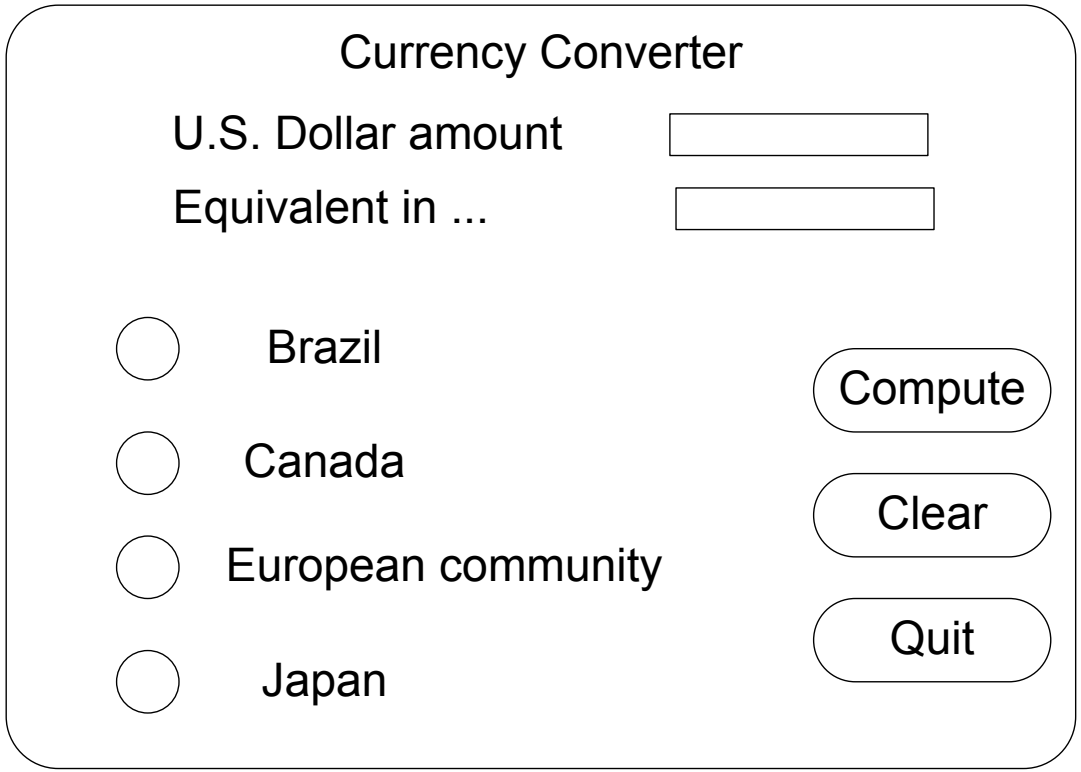
Integration Test Coverage Metrics

- Coverage metrics with respect to Call Graph
- Given a set of test cases that
  - covers each method (node coverage)
  - covers each message (edge coverage)
  - covers each path

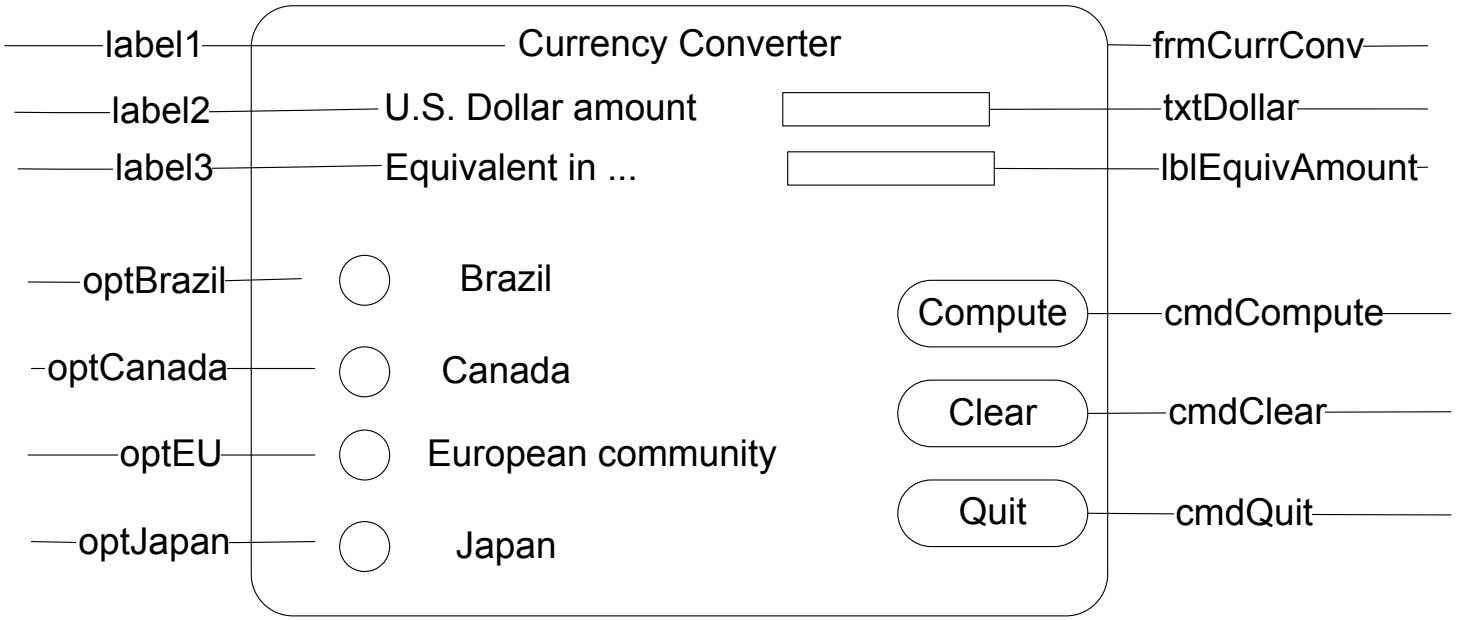
System Testing for O-O Software

- Nearly equivalent to system testing for procedural software.
- Some possibilities for sources of system test cases...
  - Use cases
  - Model-Based system testing

Second Example (event-driven system testing)



Second Example (continued)



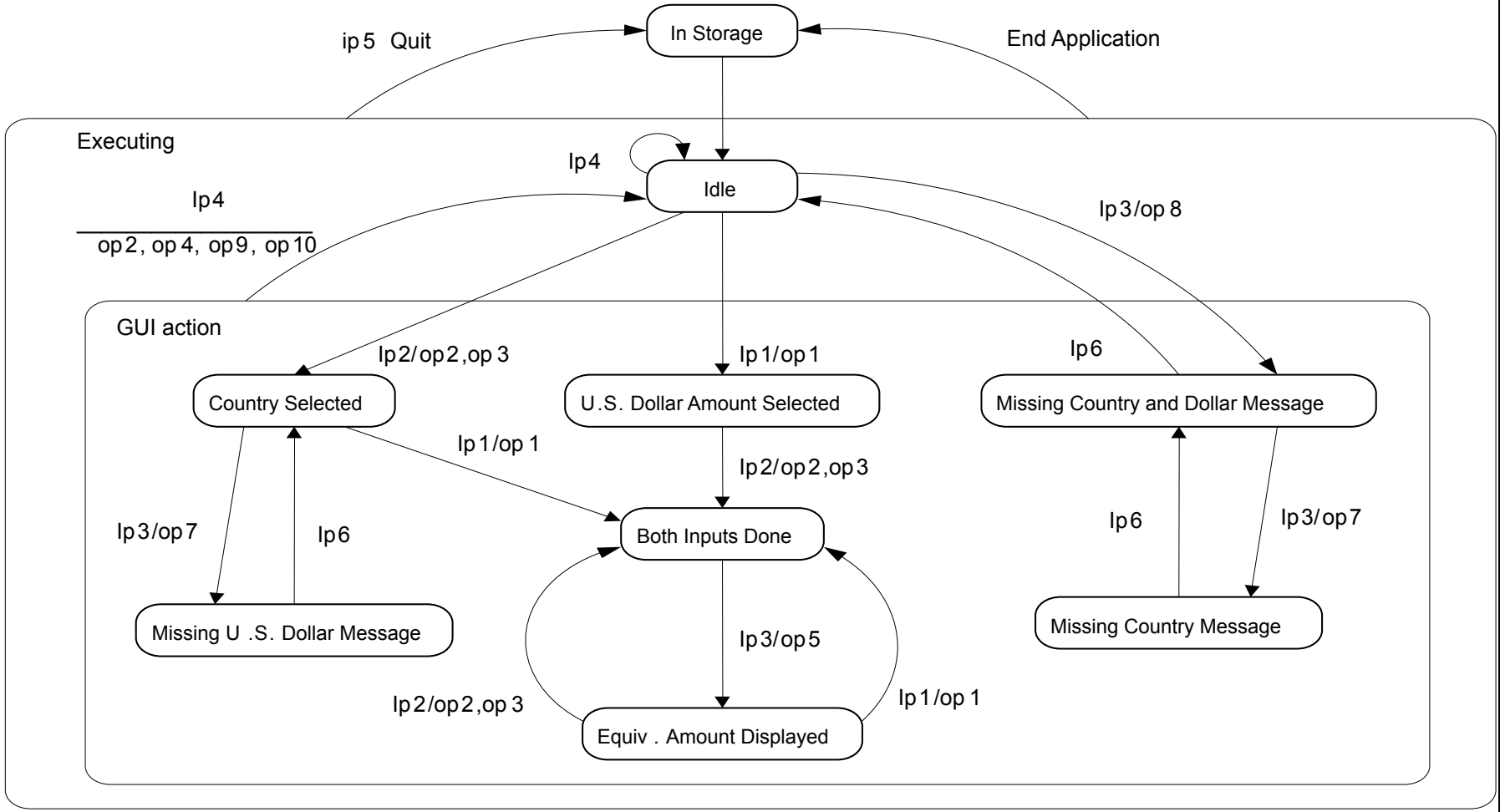
Input Events for the Currency Converter

	Input Events		Input Events
ip1	Enter US Dollar amount	ip2.4	Click on Japan
ip2	Click on a country button	ip3	Click on Compute button
ip2.1	Click on Brazil	ip4	Click on Clear button
ip2.2	Click on Canada	ip5	Click on Quit button
ip2.3	Click on European Community	ip6	Click on OK in error message

Output Events for the Currency Converter

	Output Events		Output Events
op1	Display US Dollar Amount	op4	Reset selected country
op2	Display currency name	op4.1	Reset Brazil
op2.1	Display Brazilian Reals	op4.2	Reset Canada
op2.2	Display Canadian Dollars	op4.3	Reset European Community
op2.3	Display European Community Euros	op4.4	Reset Japan
op2.4	Display Japanese Yen	op5	Display foreign currency value
op2.5	Display ellipsis	op6	Error Msg: Must select a country
op3	Indicate selected country	op7	Error Msg: Must enter US Dollar amount
op3.1	Indicate Brazil	op8	Error Msg: Must select a country and enter US Dollar amount
op3.2	Indicate Canada	op9	Reset US Dollar Amount
op3.3	Indicate European Community	op10	Reset equivalent currency amount
op3.4	Indicate Japan		

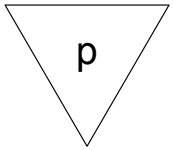
StateChart for the Currency Converter



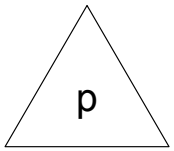
Sample System Level Test Case

System test Case -3	Normal usage (dollar amount entered first)
Test performed by	Paul Jorgensen
Pre-Conditions	txtDollar has focus
Input events	Output events
1. Enter 10 on the keyboard	2. Observe 10 appears in txtDollar
3. Click on optEU button	4. Observe “euros” appears in label3
5.Click cmdCompute button	6. Observe “7.60” appears in lblEquivAmount
Post-Conditions	cmdClear has focus
Test result	Pass (on first attempt)
Date run	May 27, 2013

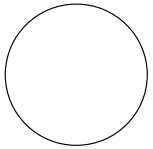
Constructs for Event- and Message-Driven Petri Nets



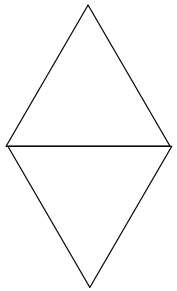
Port Input Event



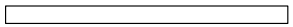
Port Output Event



Date (place)



Message send/return

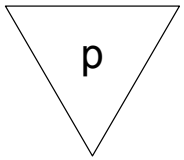


Method execution path

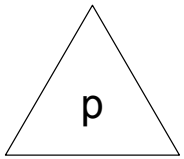
Framework for Object-Oriented Dataflow Testing

- Need a construct to express the role of messages
- An extension of Event-Driven Petri Nets
- Event/Message–Driven Petri Nets

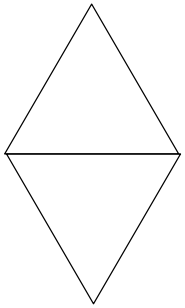
Elements of Event/Message–Driven Petri Nets



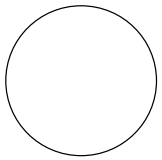
Port Input Event



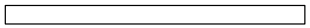
Port Output Event



Message send/return

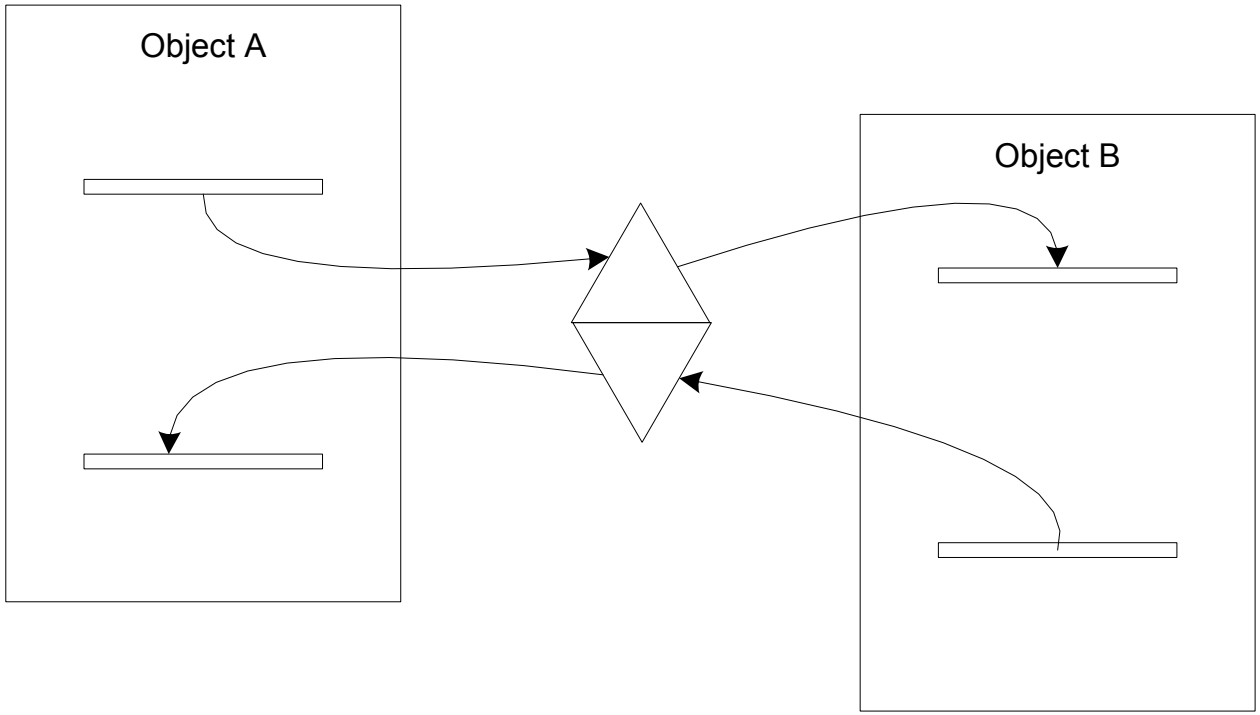


Date (place)

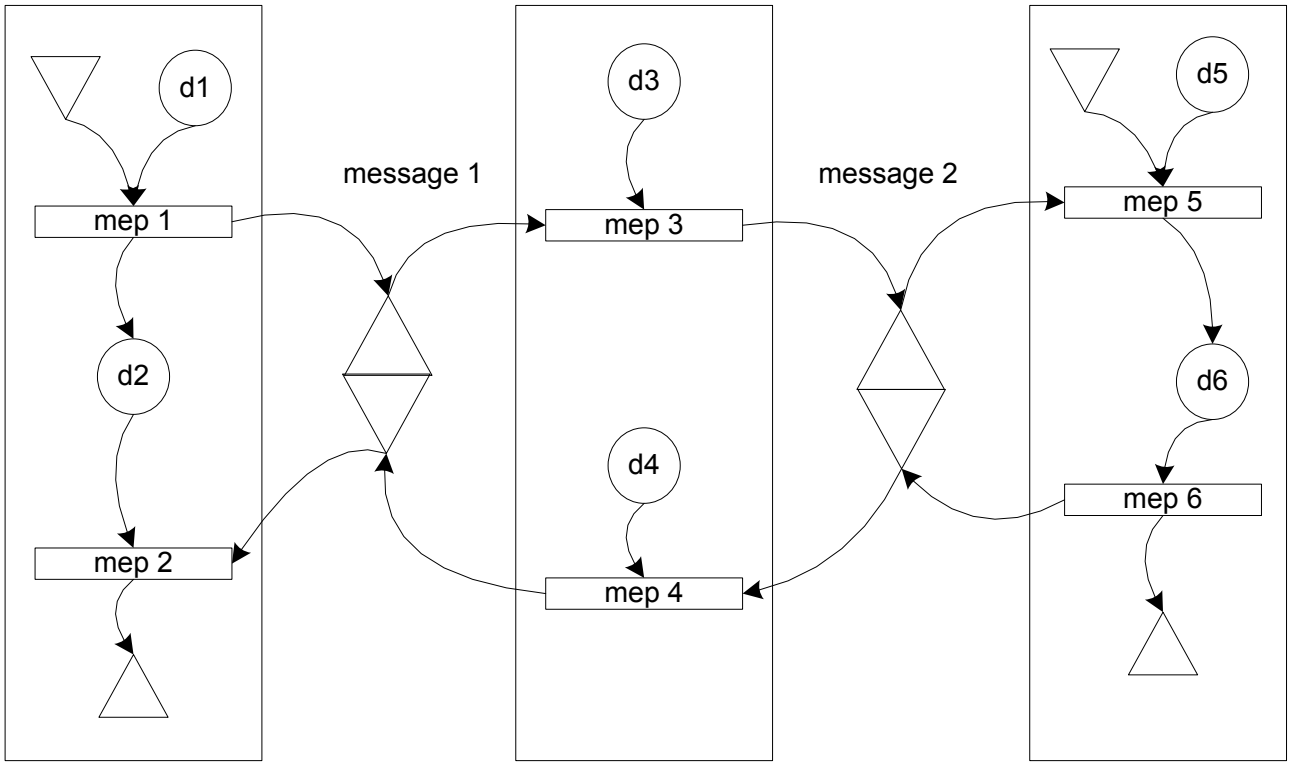


Method execution path

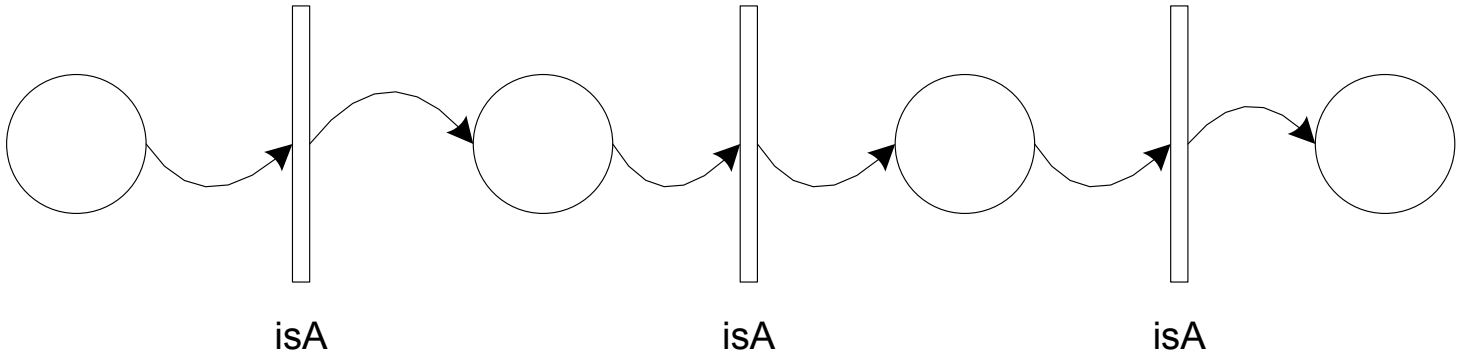
Message from object A to object B



Data Flow from Message Passing



Data Flow from Inheritance



Observations and Conclusions

- Classes/objects are more complicated than procedures
  - need to deal with inheritance, polymorphism
  - good encapsulation in o-o design helps
- Complexity shifts from methods to integration
- Dataflow testing seems appropriate for o-o integration testing.
- Not much help for Model-Based Testing at the integration level.