Model Driven Solutions

Where Business Meets Technology

A division of Data Access Technologies, Inc.

Programming in UML: An Introduction to fUML and Alf

Tutorial for the OMG Executable UML Information Day
Presented by Ed Seidewitz

22 March 2011

Agenda

I. Introduction
II. Elements of Executable UML
III. Standard Model Library

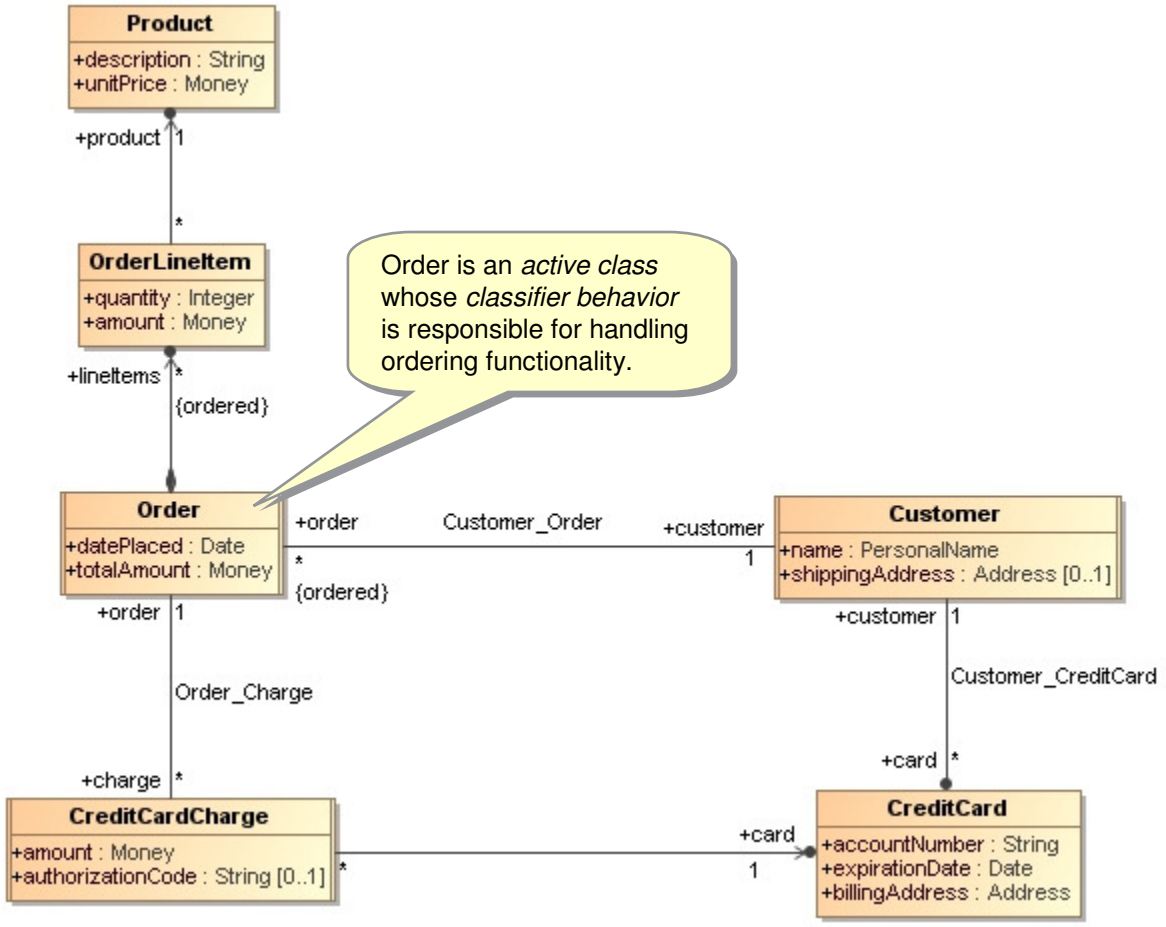
I. Introduction

A. A Motivating Example
B. Programming in UML
C. The Standards

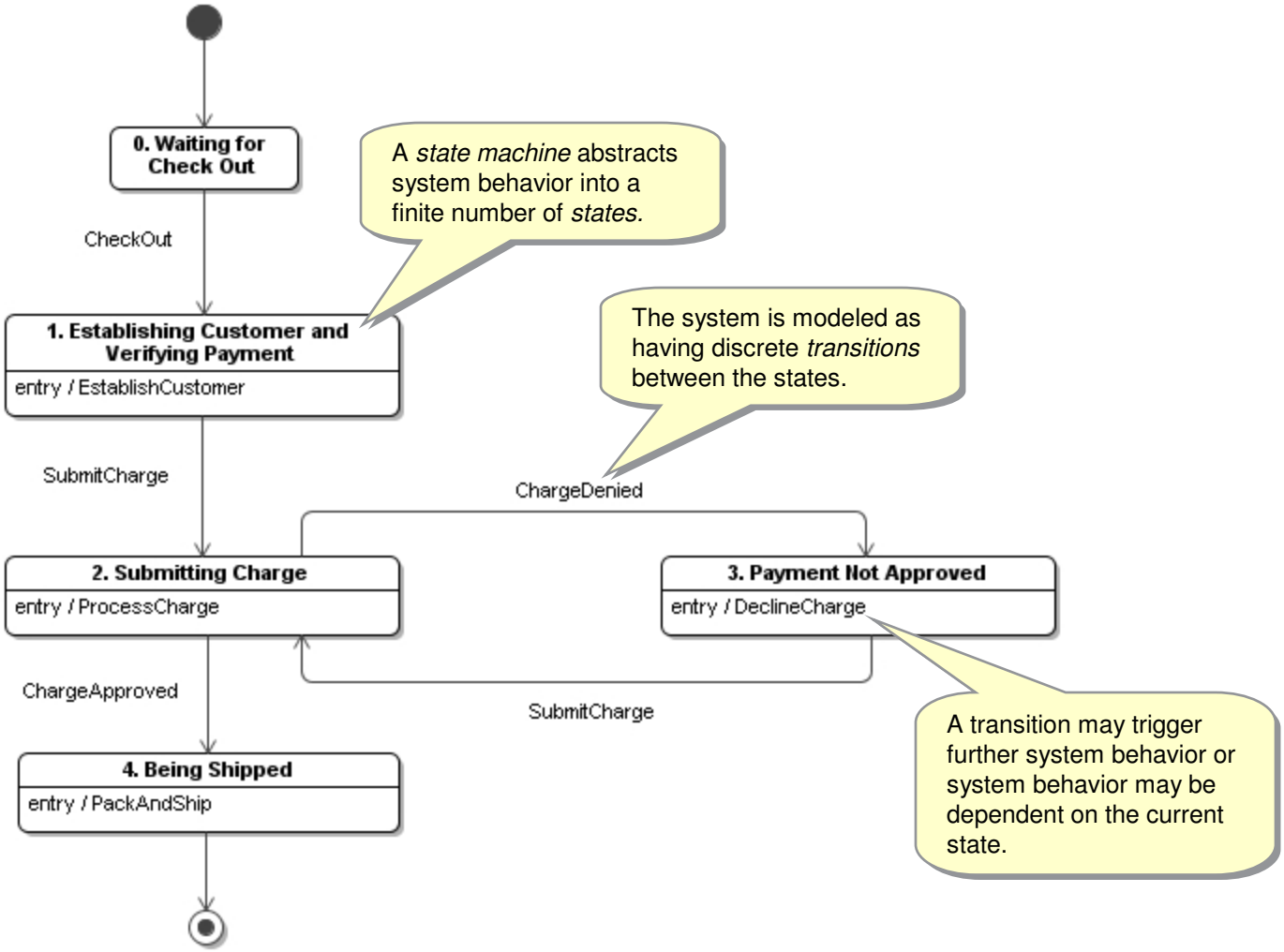
A. A Motivating Example

- E-Commerce Ordering System
  - Loosely based on the Online Bookstore Domain Case Study given in Appendix B of the book *Executable UML: A Foundation for Model Driven Architecture* by Stephen J. Mellor and Marc J. Balcer (Addison-Wesley, 2002)
- To be designed in UML
  - Class models for structure
  - State machine models for behavior
- To be implemented on the Web
  - Using Java/JEE
  - …or maybe C#/.Net
  - …or maybe LAMP
  - …or whatever…

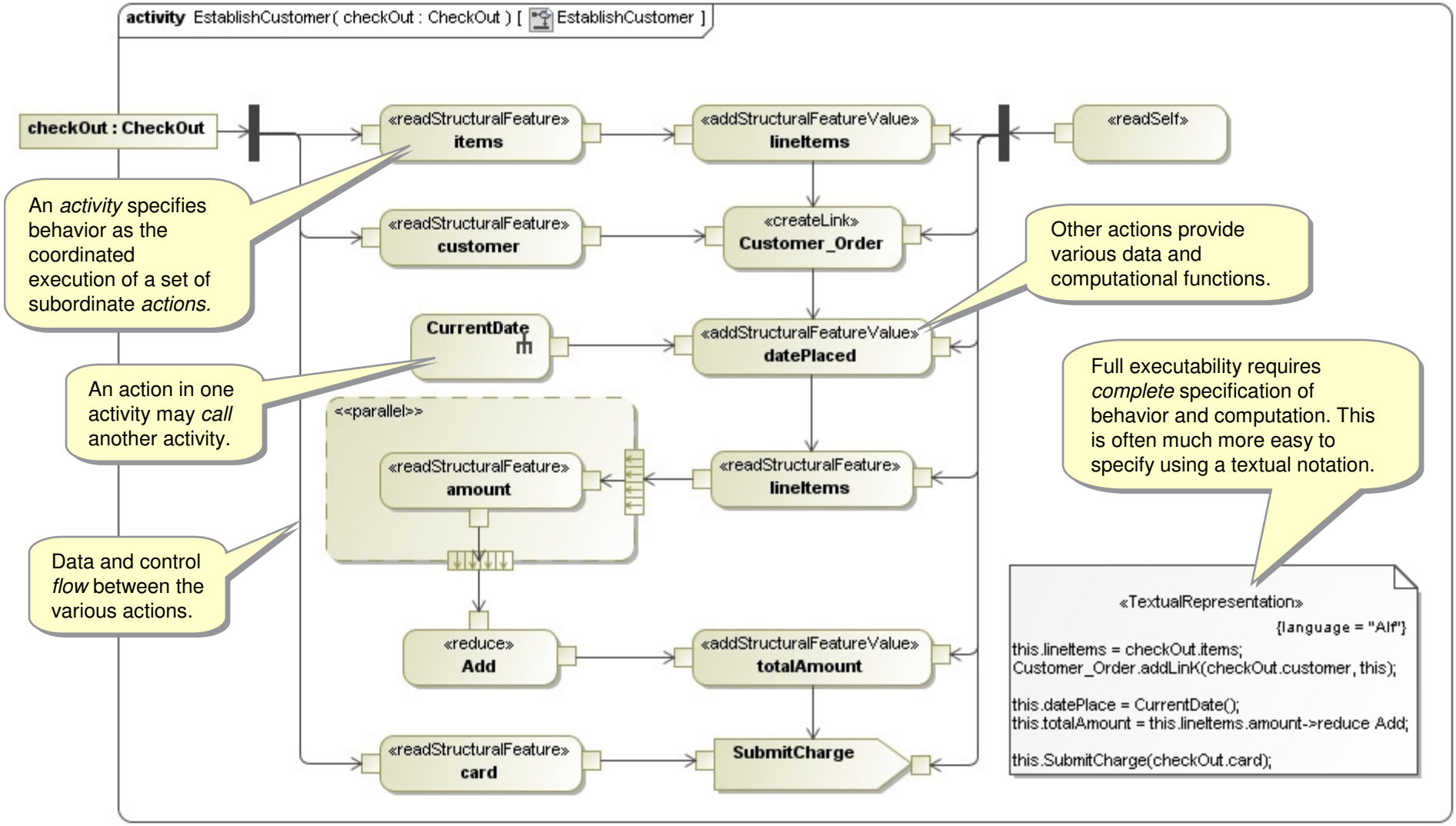
Ordering: Class Model



Order: Classifier Behavior



Order Behavior: EstablishCustomer Activity



The Question Is…

- If we are going to take the time to carefully design our system using UML, e.g.,
  - Structural models of classes and associations
  - Behavioral models using state machines or operations and messages
- Then why can't we use these directly to execute our system?

The answer is: We can!

- Just add detailed behavior…
  - …which is best done using a textual action language…
  - …which should be at the same semantic level as the rest of the model.

Executable UML: Perceived Issues

- Making models detailed enough for machine execution defeats the purpose of models for human communication.
- UML is not specified precisely enough to be executed (at least not in a standard way).
- Graphical modeling notations are not good for detailed programming.

Executable UML: Issue Resolutions

- Making models detailed enough for machine execution defeats the purpose of models for human communication.
  - **Executable models can still be more understandable than executable code.**
  - **Non-executable models are still useful, too.**
- UML is not specified precisely enough to be executed (at least not in a standard way).
  - **The Foundational UML (fUML) standard specifies precise semantics for an executable subset of UML.**
  - **fUML Version 1.0 formal specification now available.**
- Graphical modeling notations are not good for detailed programming.
  - **The Action Language for fUML (Alf) standard specifies a textual action language with fUML semantics.**
  - **Alf Version 1.0 finalization in progress.**

B. The Standards

- Unified Modeling Language (UML)
- Executable UML Foundation (fUML)
- UML Action Language (Alf)

Unified Modeling Language (UML)

The *Unified Modeling Language* (UML) is a graphical language for modeling the structure, behavior and interactions of software, hardware and business systems, standardized by the Object Management Group (OMG).

- UML Version 1.1 (first standard) – November 1997
- UML Version 1.5 (with action semantics) – March 2003
- UML Version 2.0 – August 2005
- UML Version 2.3 (current standard) – May 2010
- UML Version 2.4 – January 2011 (beta)
- UML Version 2.5 (spec simplification) – August 2011 (planned)

Executable UML Foundation (fUML)

Foundational UML (fUML) is an executable subset of standard UML that can be used to define, in an operational style, the structural and behavioral semantics of systems.

- OMG RFP for the *Semantics of a Foundational Subset for Executable UML Models* – Issued April 2005
- fUML Version 1.0 Beta 3 (finalized) – February 2010
- fUML Version 1.0 (formal) – January 2011

Key Components

- **Foundational UML Subset (fUML)** – A *computationally complete* subset of the abstract syntax of UML (Version 2.3)
  - **Kernel** – Basic object-oriented capabilities
  - **Common Behavior** – General behavior and asynchronous communication
  - **Activities** – Activity modeling, including structured activities (but *not* including variables, exceptions, swimlanes, streaming or other "higher level" activity modeling)
- **Execution Model** – A model of the execution semantics of user models within the fUML subset
- **Foundational Model Library**
  - **Primitive Types** – Boolean, String, Integer, Unlimited Natural
  - **Primitive Behaviors** – Boolean, String and Arithmetic Functions
  - **Basic Input/Output** – Based on the concept of "Channels"

UML Action Language (Alf)

The *Action Language for Foundational UML* (Alf) is a textual surface representation for UML modeling elements with the primary purpose of acting as the surface notation for specifying executable (fUML) behaviors within an overall graphical UML model. (But which also provides an extended notation for structural modeling within the fUML subset.)

- OMG RFP for *Concrete Syntax for a UML Action Language* – Issued September 2008
- Alf Version 1.0 Beta 1 – October 2010

Key Components

- **Concrete Syntax** – A BNF specification of the legal textual syntax of the Alf language.
- **Abstract Syntax** – A MOF metamodel of the abstract syntax tree that is *synthesized* during parsing of an Alf text, with additional *derived* attributes and constraints that specify the static semantic analysis of that text.
- **Semantics** – The semantics of Alf are defined by *mapping* the Alf abstract syntax metamodel to the fUML abstract syntax metamodel.
- **Standard Model Library**
  - **From the fUML Foundational Model Library**
    - Primitive Types (plus Natural and Bit String)
    - Primitive Behaviors (plus Bit String Functions and Sequence Functions)
    - Basic Input/Output
  - **Collection Functions** – Similar to OCL collection operations for sequences
  - **Collection Classes** – Set, Ordered Set, Bag, List, Queue, Deque, Map

II. Elements of Executable UML

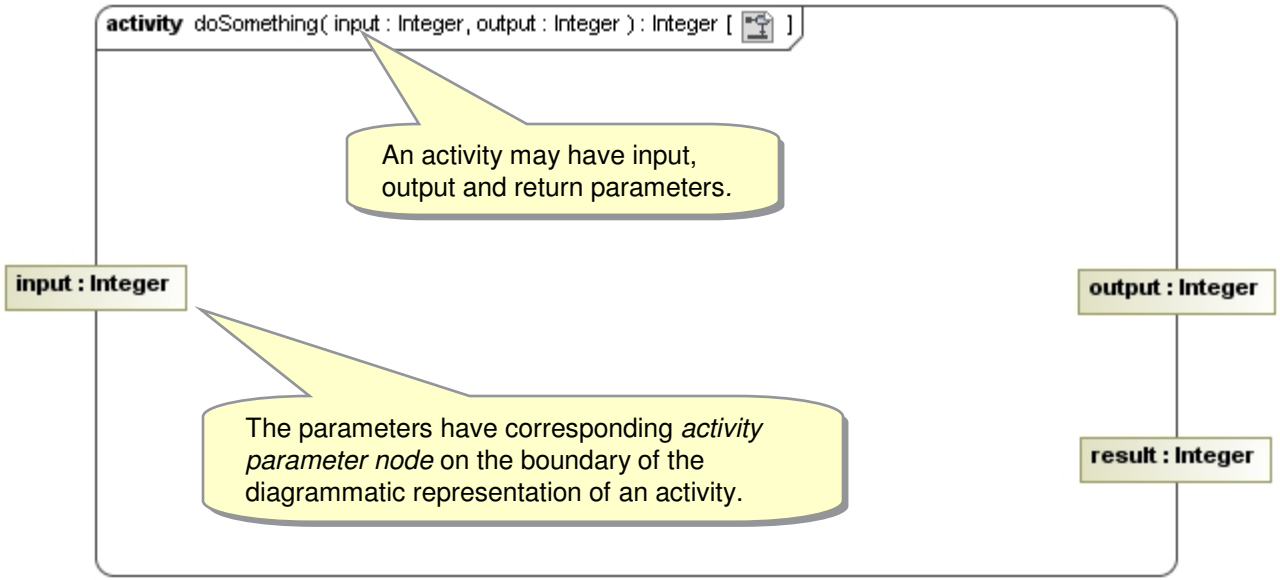
A. Activities
B. Actions
C. Structure
D. Asynchronous Communication

A. Activities

- Activities and Parameters
- Actions and Flows
- Textual Notation
- Tokens
- Offers
- Control Nodes
- Structured Nodes

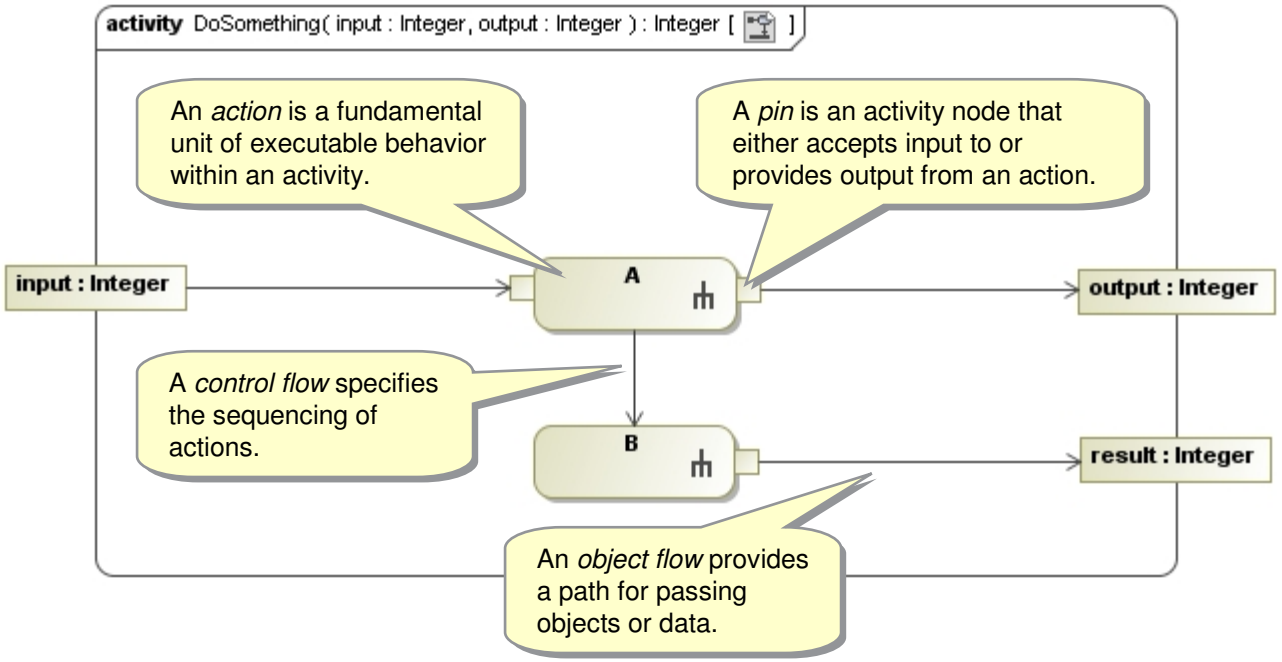
Activities and Parameters

An *activity* is a specification of behavior as the coordinated execution of subordinate *actions,* using a control and data flow model.

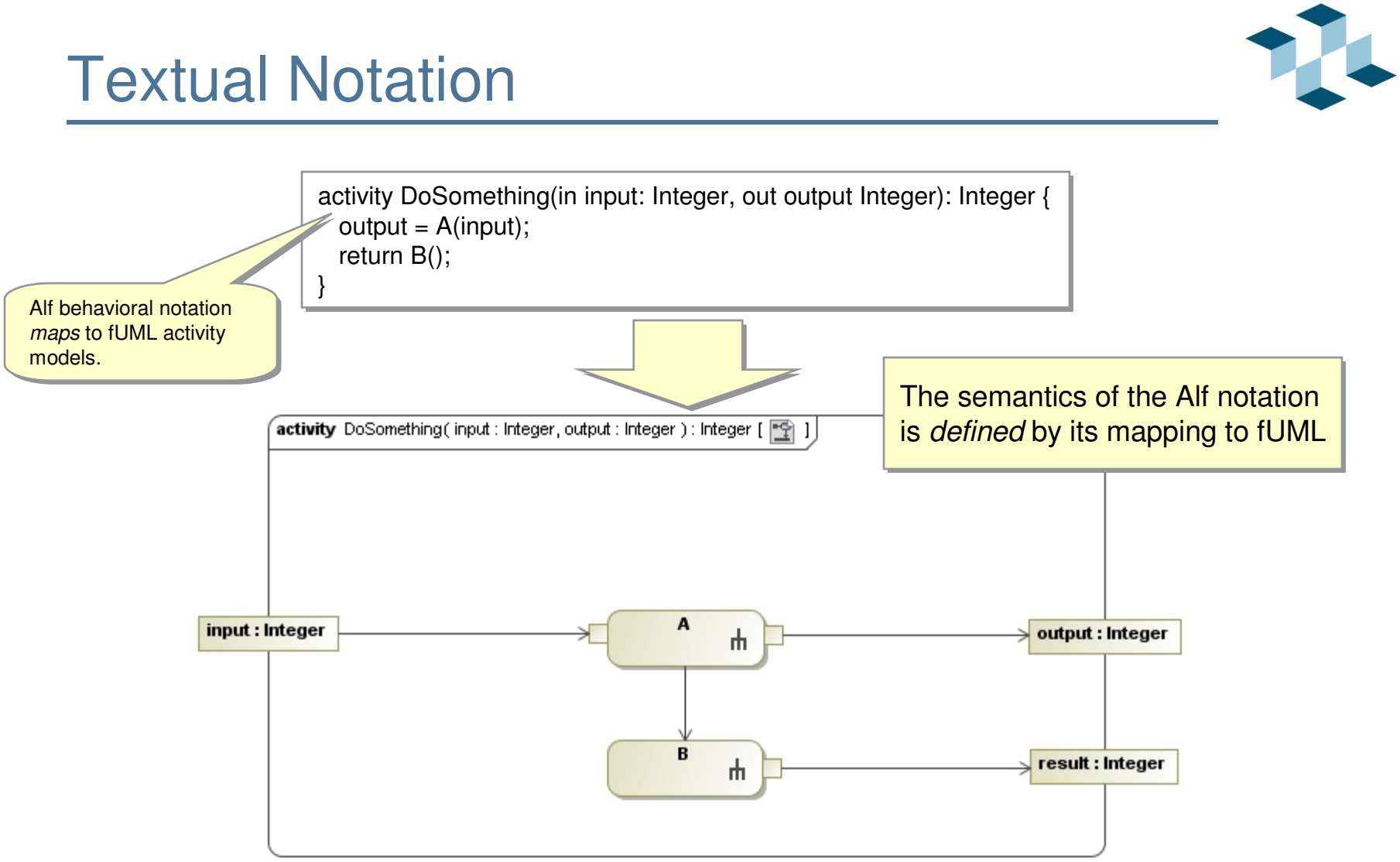


Actions and Flows

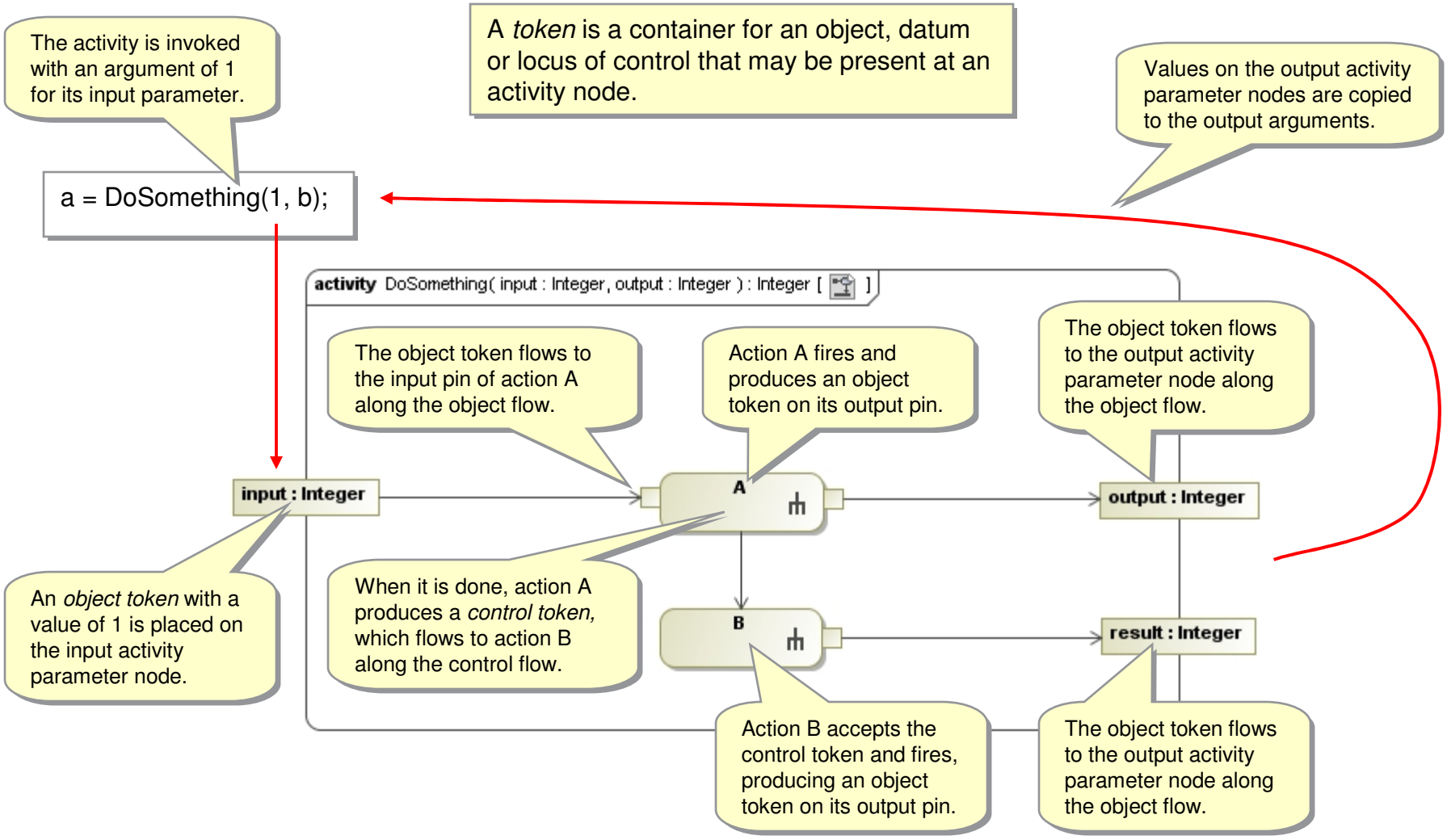
An *activity diagram* is a graph structure consisting of *activity nodes* connected by *activity edges.*



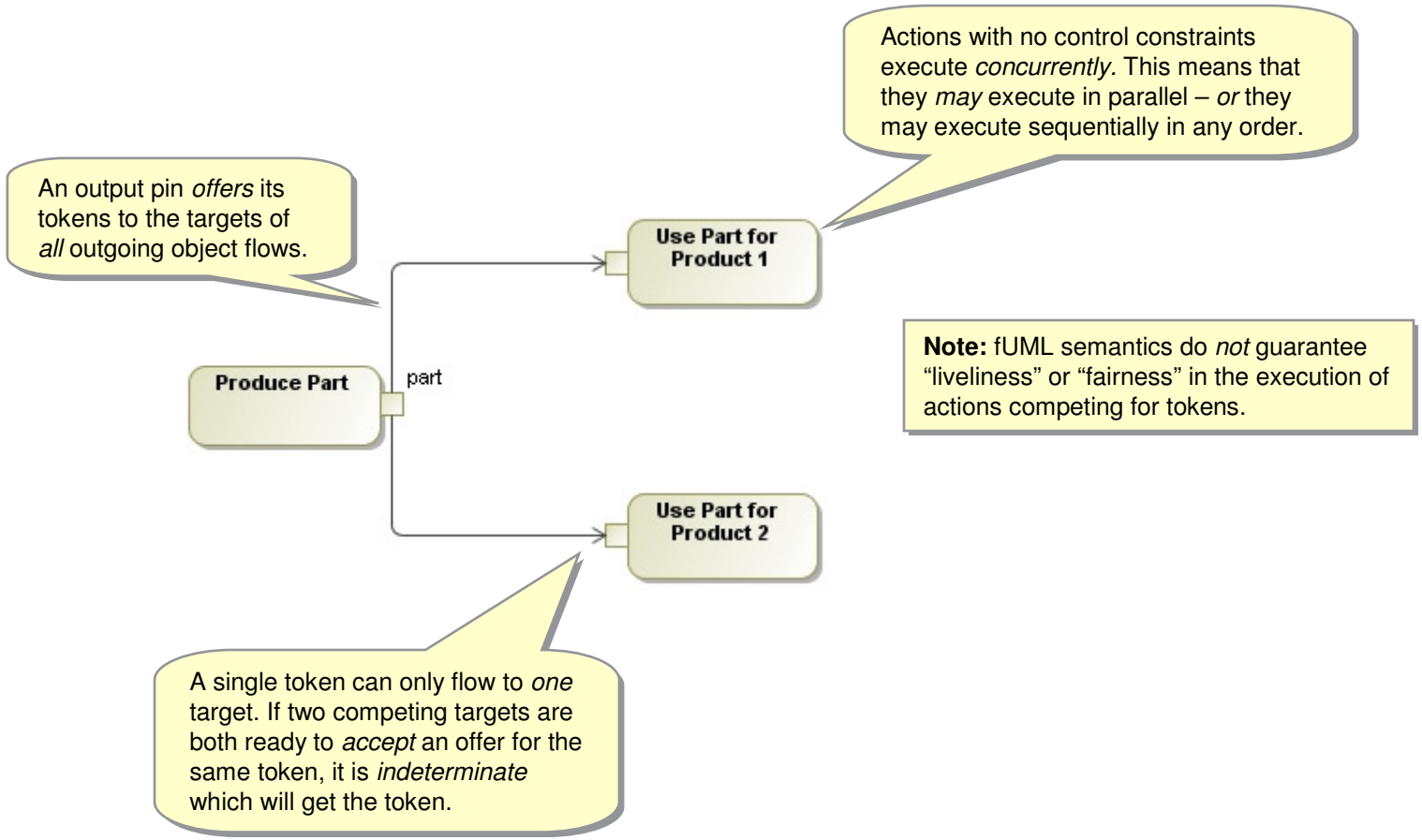
Textual Notation



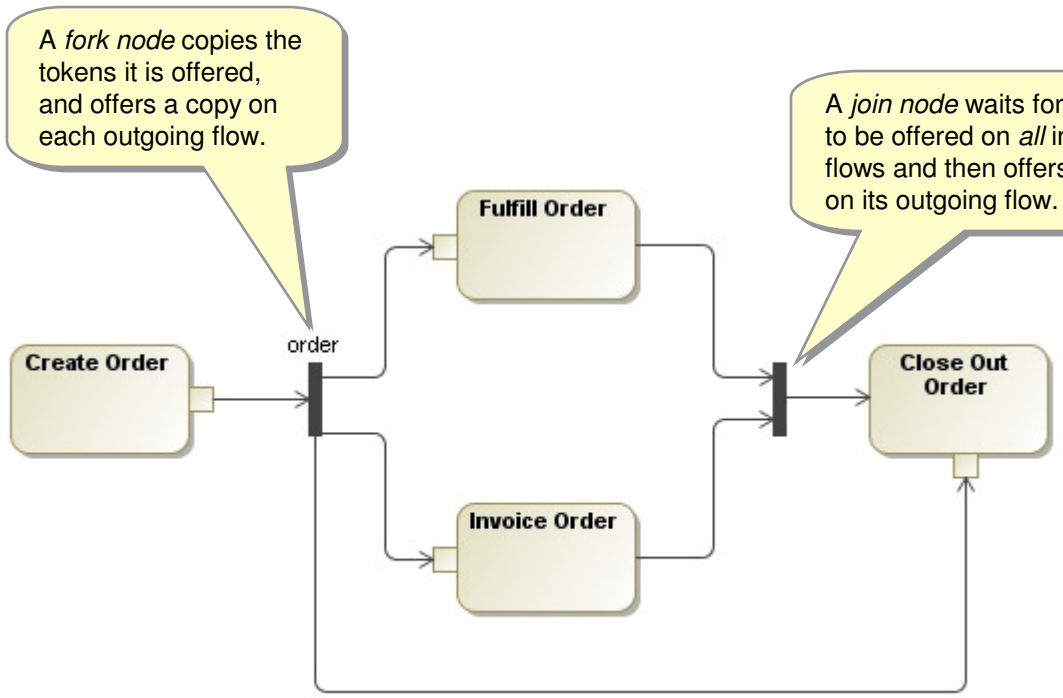
Tokens



Offers



Fork and Join Nodes

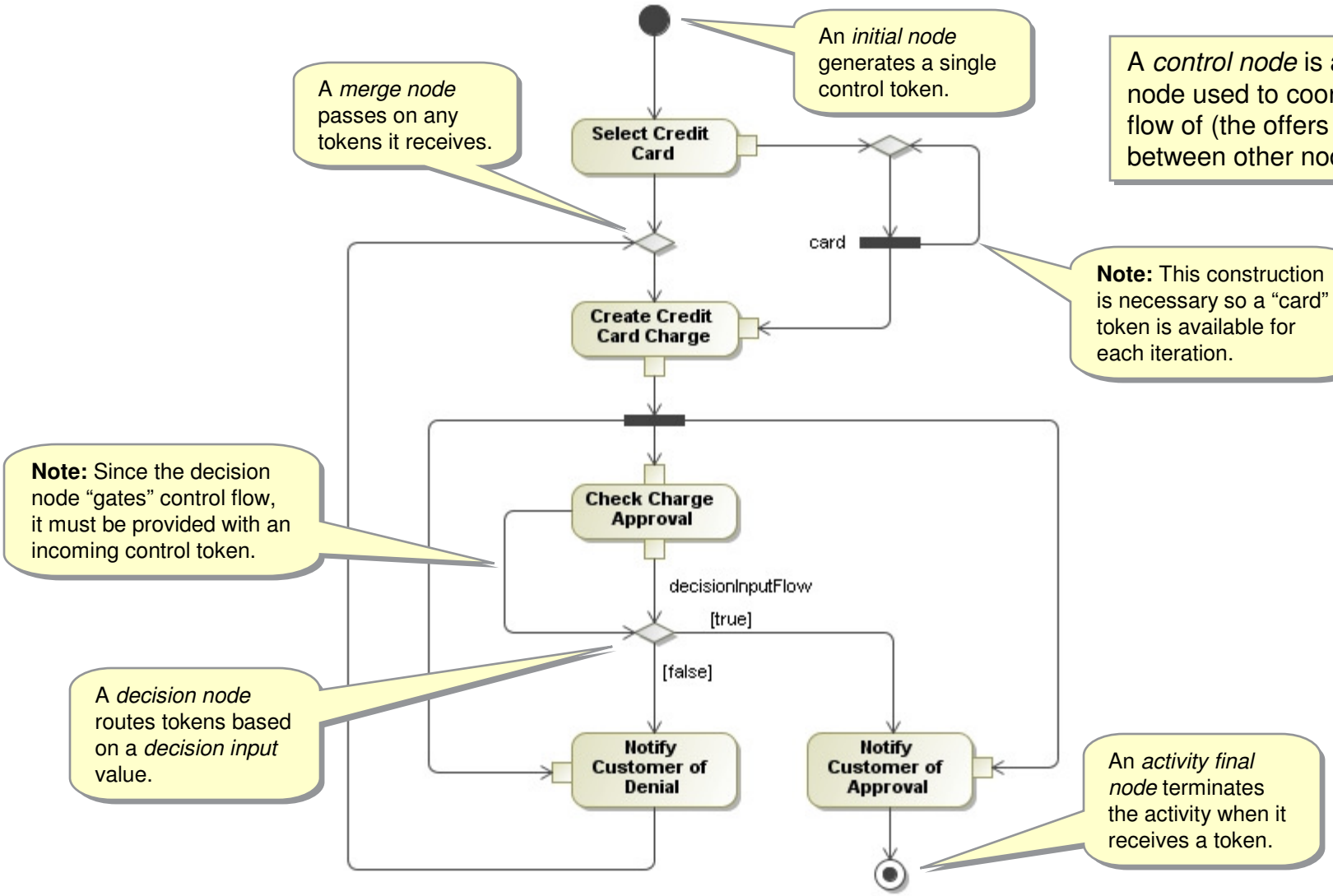


In the Alf textual notation, forks and joins are implicit in the *parallel block* notation.

```
order = 'Create Order'();
//@parallel
{
  'Fulfill Order'(order);
  'Invoice Order'(order);
}
'Close Out Order'(order);
```

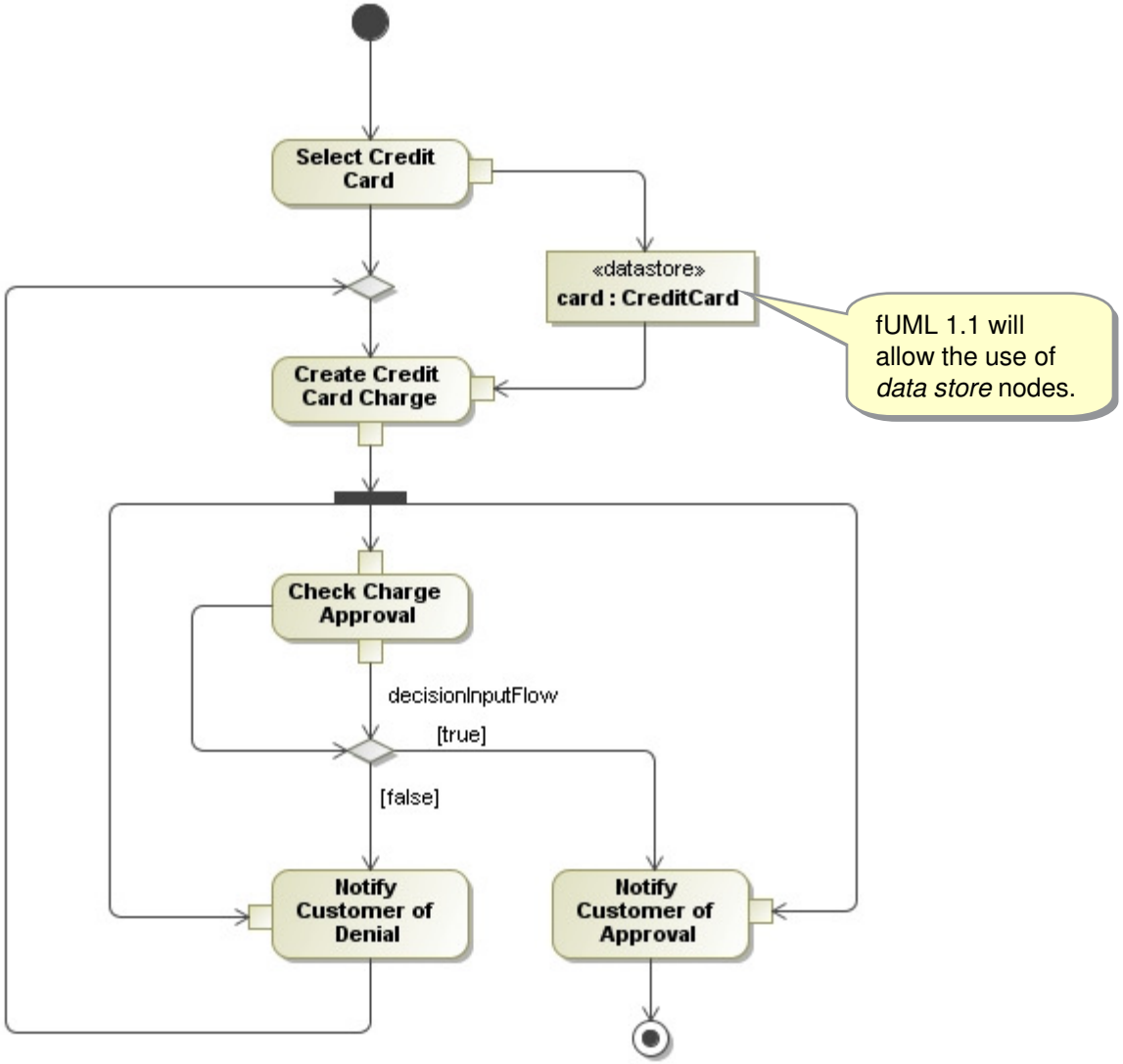
Note: Alf does not actually provide any notation for competition for tokens on output pins. A fork node is always inserted for multiple flows out of any output pin.

Control Nodes

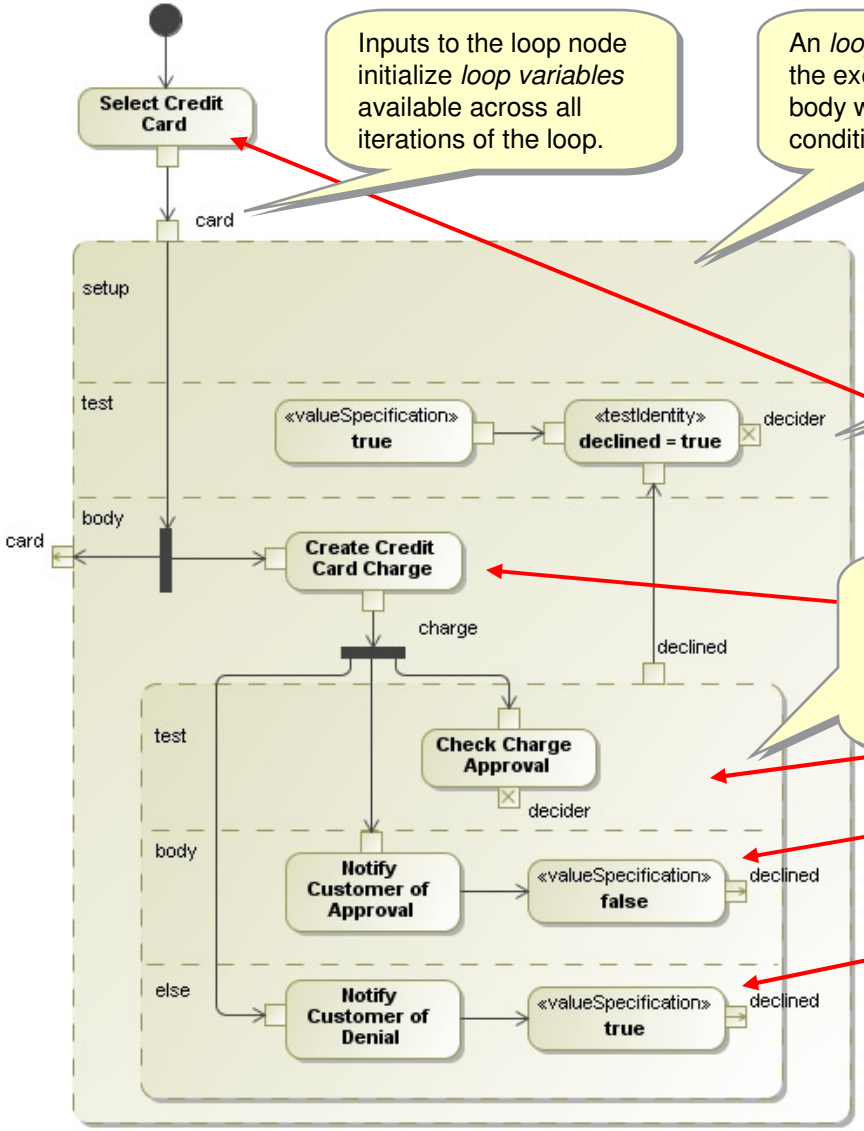


A *control node* is an activity node used to coordinate the flow of (the offers for) tokens between other nodes.

Data Stores



Structured Nodes



Inputs to the loop node initialize *loop variables* available across all iterations of the loop.

An *loop node* iterates the execution of its body while a condition is true.

By default, the condition is tested *after* execution of the body.

A *structured node* is an activity node used to group subordinate nodes into a control structure.

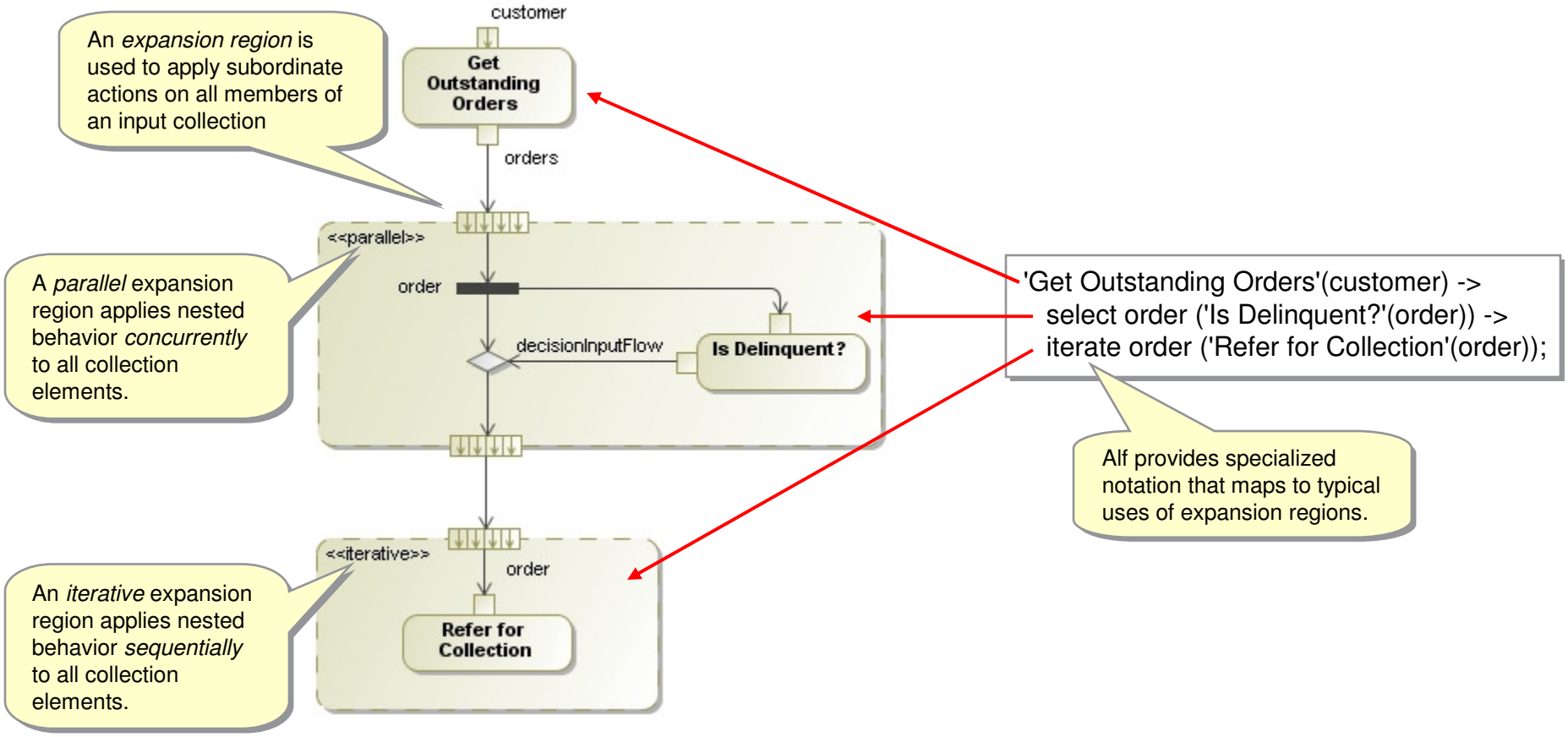
Note: There is no normative UML graphical notation for loop or conditional nodes.

A *conditional node* executes one *clause* or another based on the result of a test (or tests).

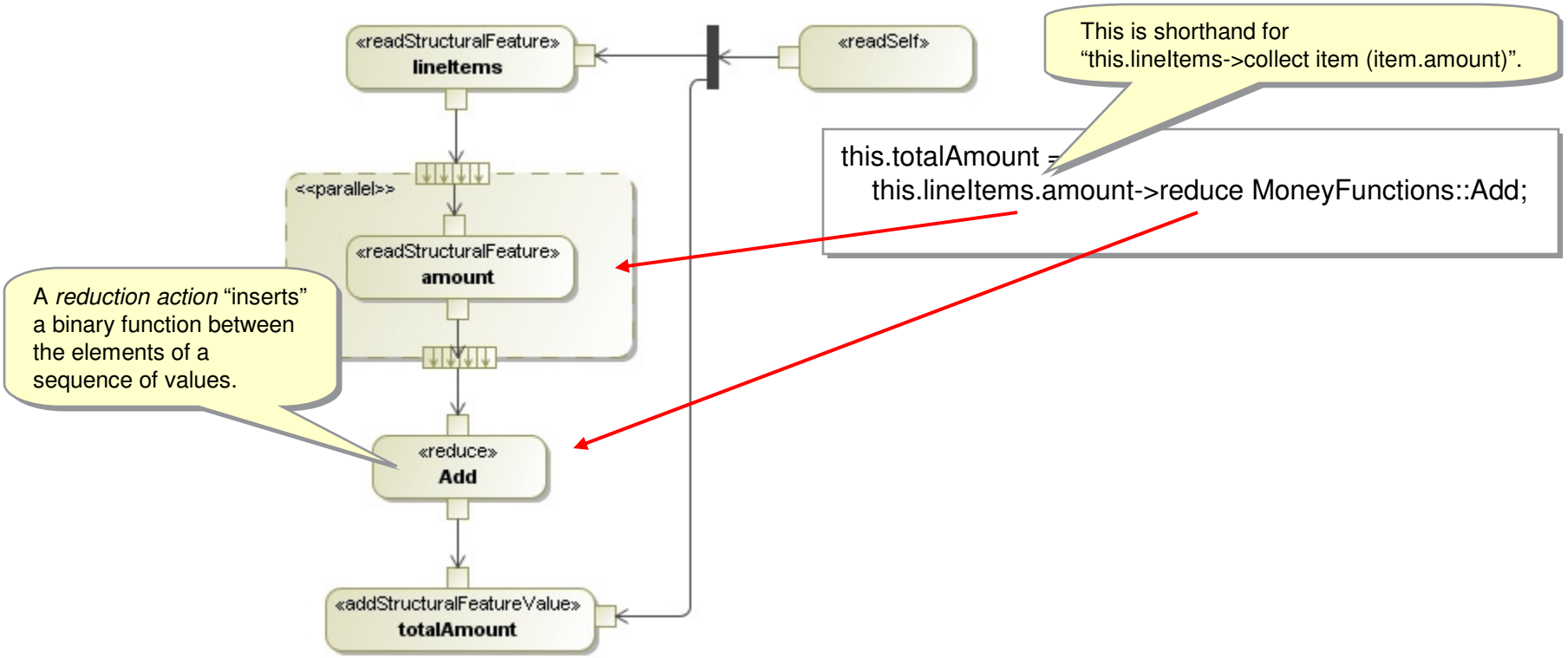
```
card = 'Select Credit Card'();

do {
  charge = 'Create Credit Card Charge'(card);

  if ('Check Charge Approval'(charge)) {
     'Notify Customer of Approval'(charge);
     declined = false; break;
  } else {
     'Notify Customer of Denial'(charge);
     declined = true;
  }
            true
} while (declined);
```

Alf also provides a traditional *break* statement.

Expansion Regions



Reduction



B. Actions

- Invocation Actions
- Object Actions
- Structural Feature Actions
- Link Actions

NOTE: Some of these actions will be discussed in more detail later.

Invocation Actions

- Call Behavior
  - Calling an activity — PlaceOrder(customer, product)
  - Calling a primitive behavior — Max(throttle, limit) count + quantity
- Call Operation — order.addProduct(product, quantity)
- Send Signal — vehicle.EngageBrake(pressure)
- Accept Event — accept (signal: EngageBrake)

Object Actions

- Value Specification — 1 true "Hello"
- Create Object — new Order()
- Destroy Object — order.destroy()
- Test Identity — order == myOrder name != customerName
- Read Self — this
- Read Extent — Order.allInstances()
- Read Is Classified Object — vehicle instanceof Car car hastype Hatchback
- Reclassify Object — reclassify order from PendingOrder to ClosedOrder

Structural Feature Actions

- Read Structural Feature
  - order.customer
- Add Structural Feature Value
  - order.datePlaced = today
  - order.lineItems->add(item)
  - order.lineItems->addAt(1,item)
- Remove Structural Feature Value
  - order.lineItems->remove(item)
  - order.lineItems->removeAt(1)
- Clear Structural Feature Value
  - order.card = null

Link Actions

- Read Link — Owns.person(house=>thisHouse) thisHouse.person
- Create Link — Owns.addLink(person=>jack, house=>newHouse)
- Destroy Link — Owns.removeLink(person=>jack, house=>oldHouse)
- Clear Association — Owns.clearAssoc(jack)

Computation

- Indexing (from 1 , *not* 0)
- Increment/Decrement
- Arithmetic/Logic
- Comparison
- Conditional
- Isolation

order.lineItems[i] order.lineItems->at(i)

index++ ++count

total + value address & mask

total > threshold index <= count

count != 0 && total/count > limit

count < min || count > max

$this.sensor.getReading().value

C. Structure

- Structural and Behavioral Models
- Classes
- Associations

Structural and Behavioral Models

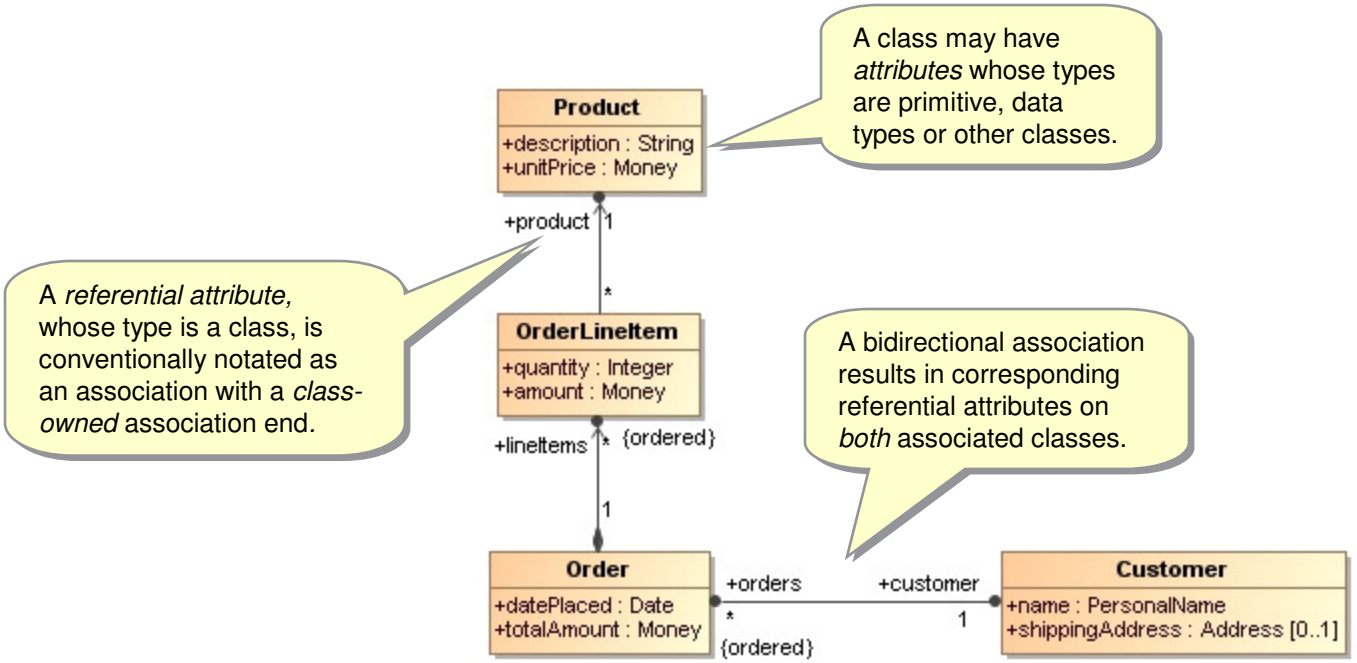
- A *structural model* (e.g., a class model) specifies the relevant (types of) *instances* in a domain that may exist at any one point in time.
  - *Structural semantics* define how a structural model constrains allowable instances.
- A *behavioral model* (e.g., an activity model) specifies behavior *over time*
  - *Behavioral semantics* define how a behavioral model changes the state of instances over time.

Classes

A *class* is a *classifier* of *objects* that persist in the *extent* of the class, with an identity that is independent of the value of their *attributes* at any one time.

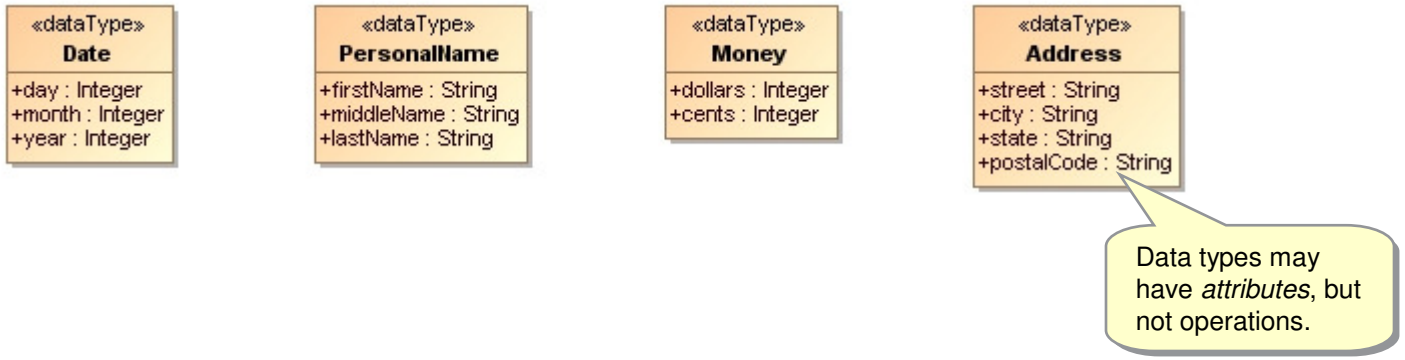
- Attributes
- Data Types
- Primitive Types
- Operations and Methods
- Structural Semantics
- Behavioral Semantics

Classes and Attributes



Data Types

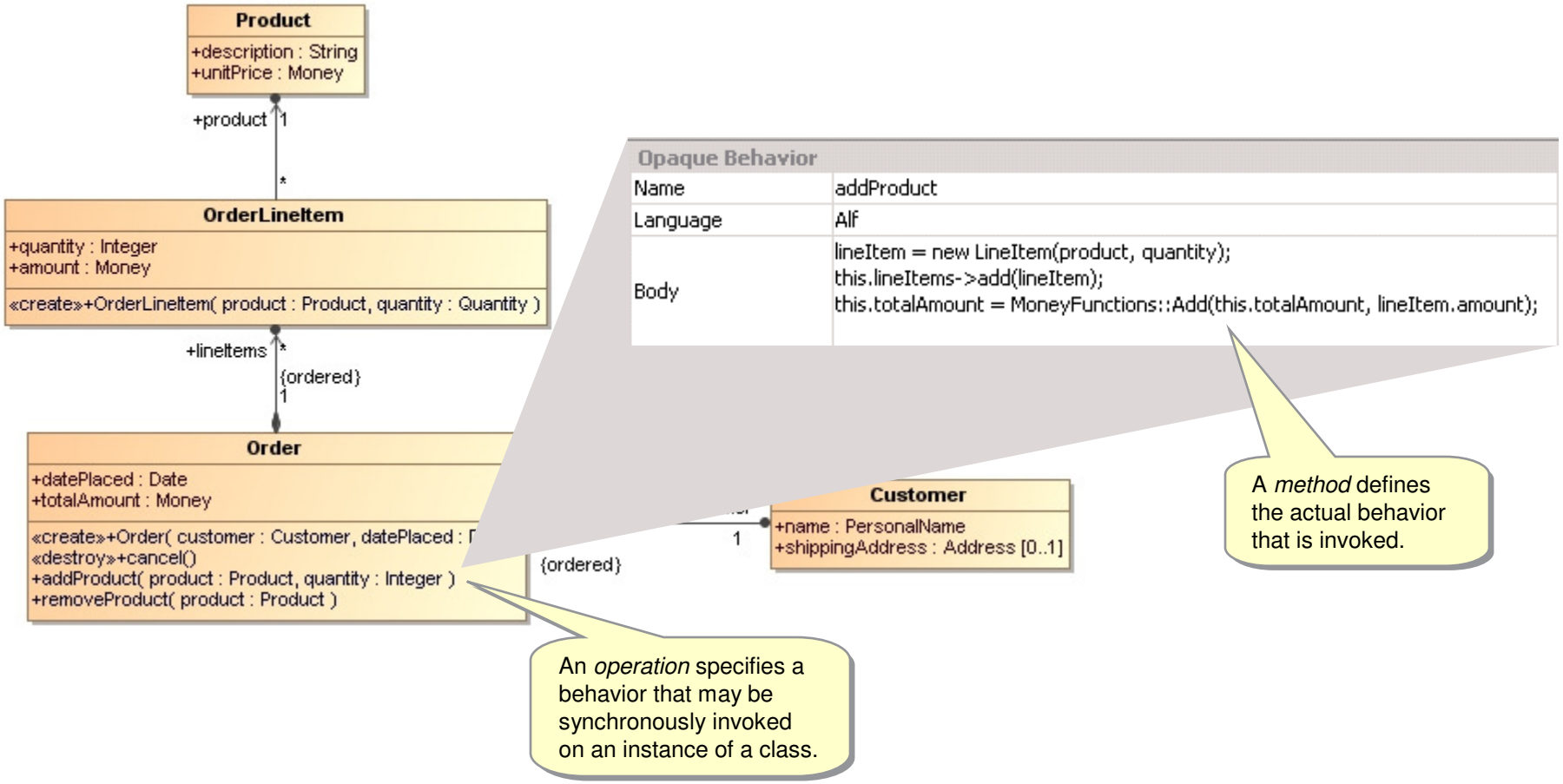
A *data type* is a *classifier* of transient *data values* whose identity is based on the values of their attributes.



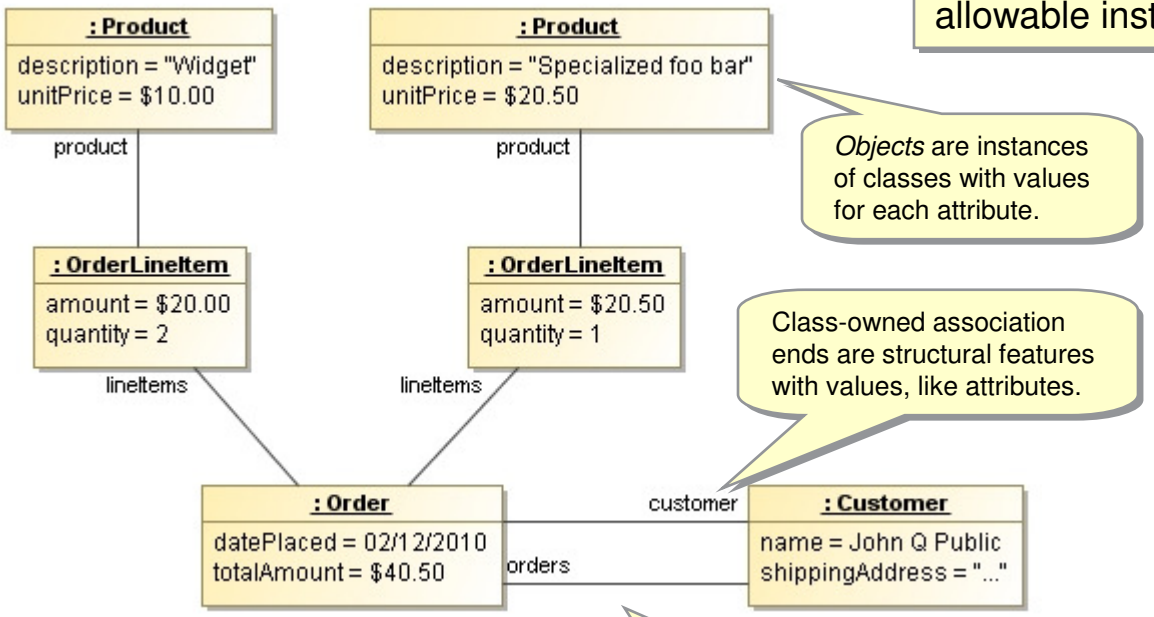
Primitive Types

- From UML 2.3 Auxiliary Constructs
  - Boolean
  - Integer
  - UnlimitedNatural
  - String
- fUML 1.1 will:
  - Use the new UML 2.4 PrimitiveTypes package
  - Include support for the new Real type

Classes: Operations and Methods



Classes: Structural Semantics



Structural semantics specify how a structural model constrains allowable instances.

Objects are instances of classes with values for each attribute.

Class-owned association ends are structural features with values, like attributes.

fUML does not actually give semantics to an association with class-owned ends, only to the ends as structural features.

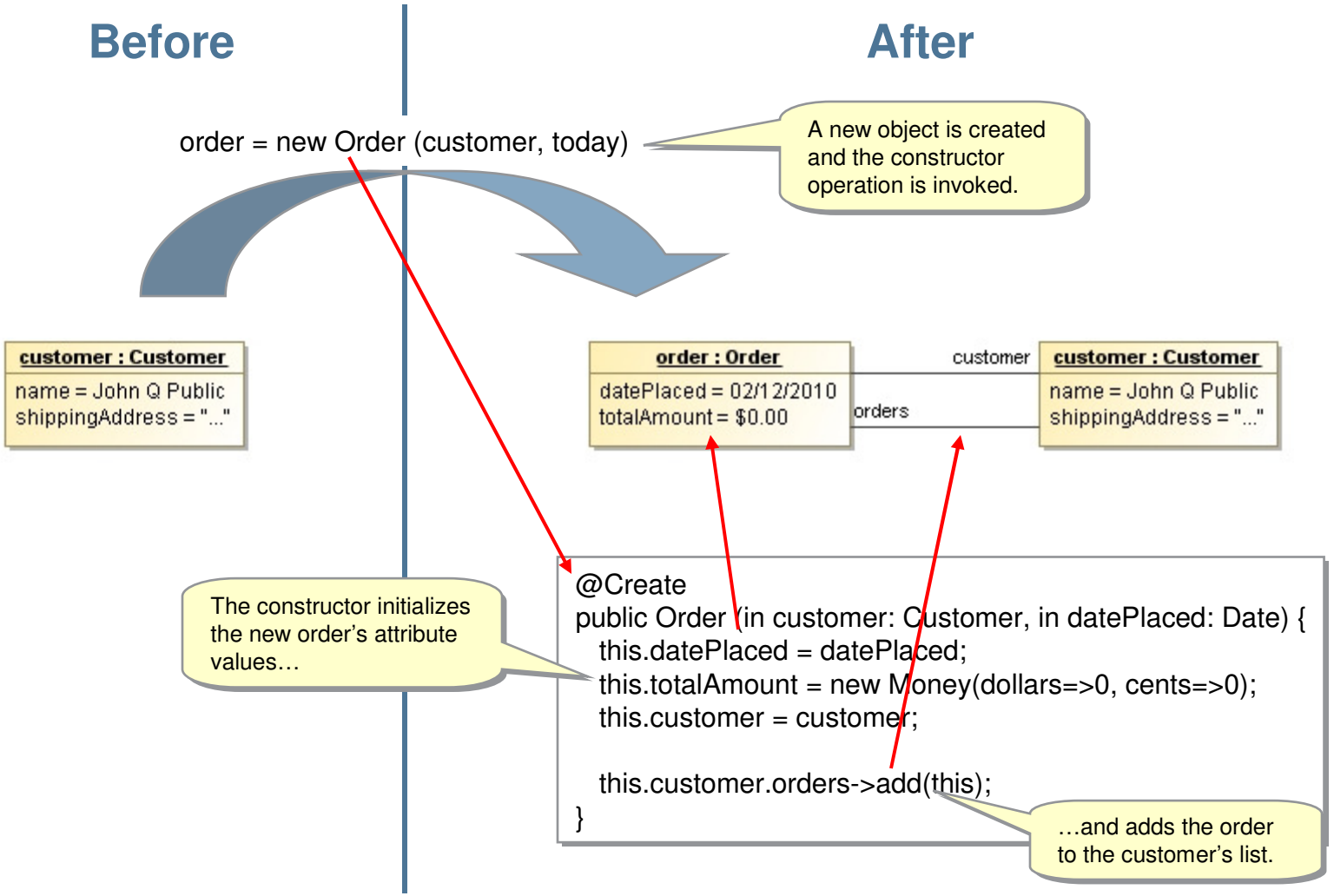
Note: fUML does provide "reified" semantics for associations that own their own ends, as will be discussed later.

Classes: Behavioral Semantics

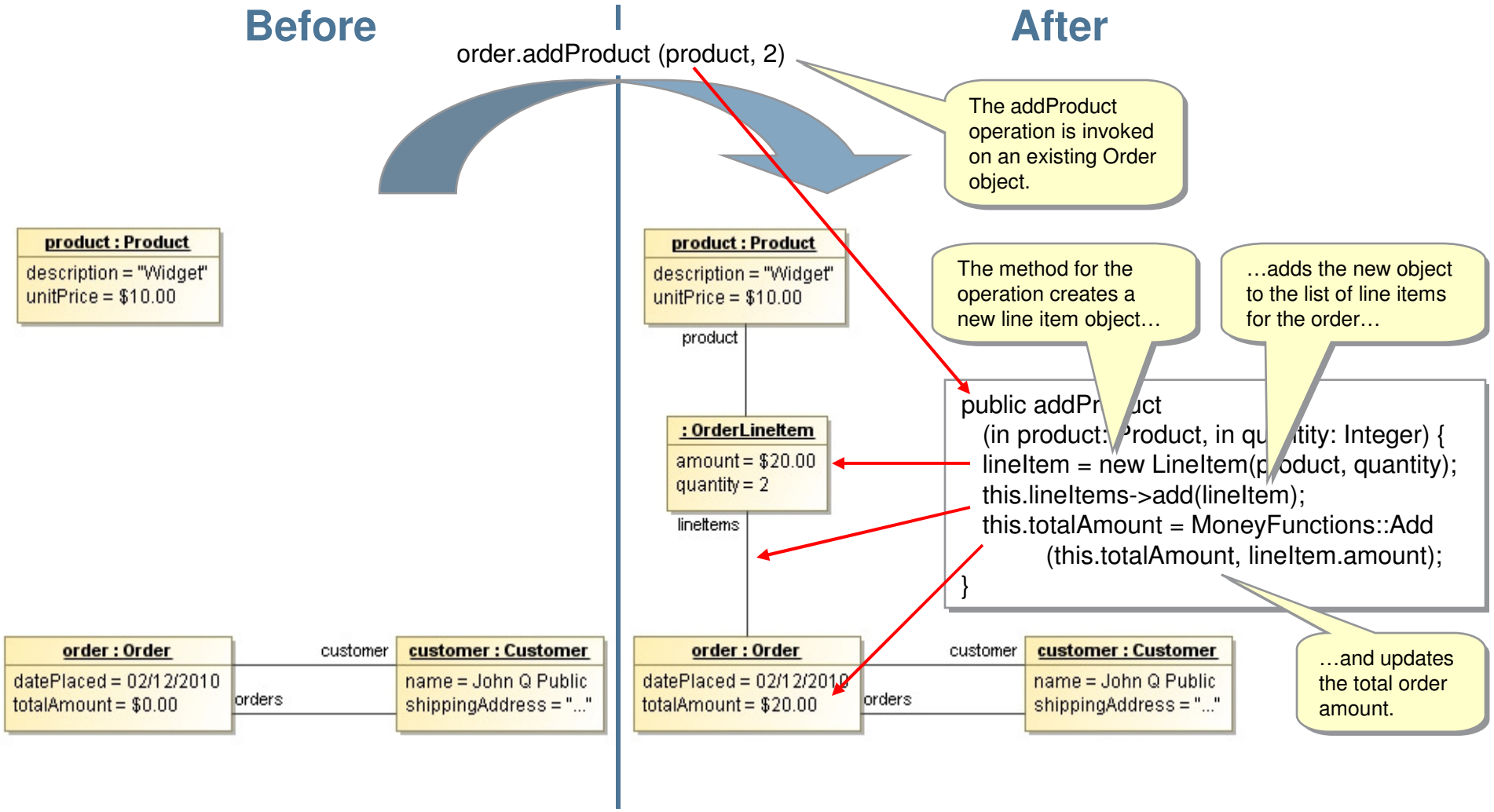
Behavioral semantics specify how a behavioral model changes the state of instances over time.

- Creating an Order
- Adding a Line Item
- Canceling an Order

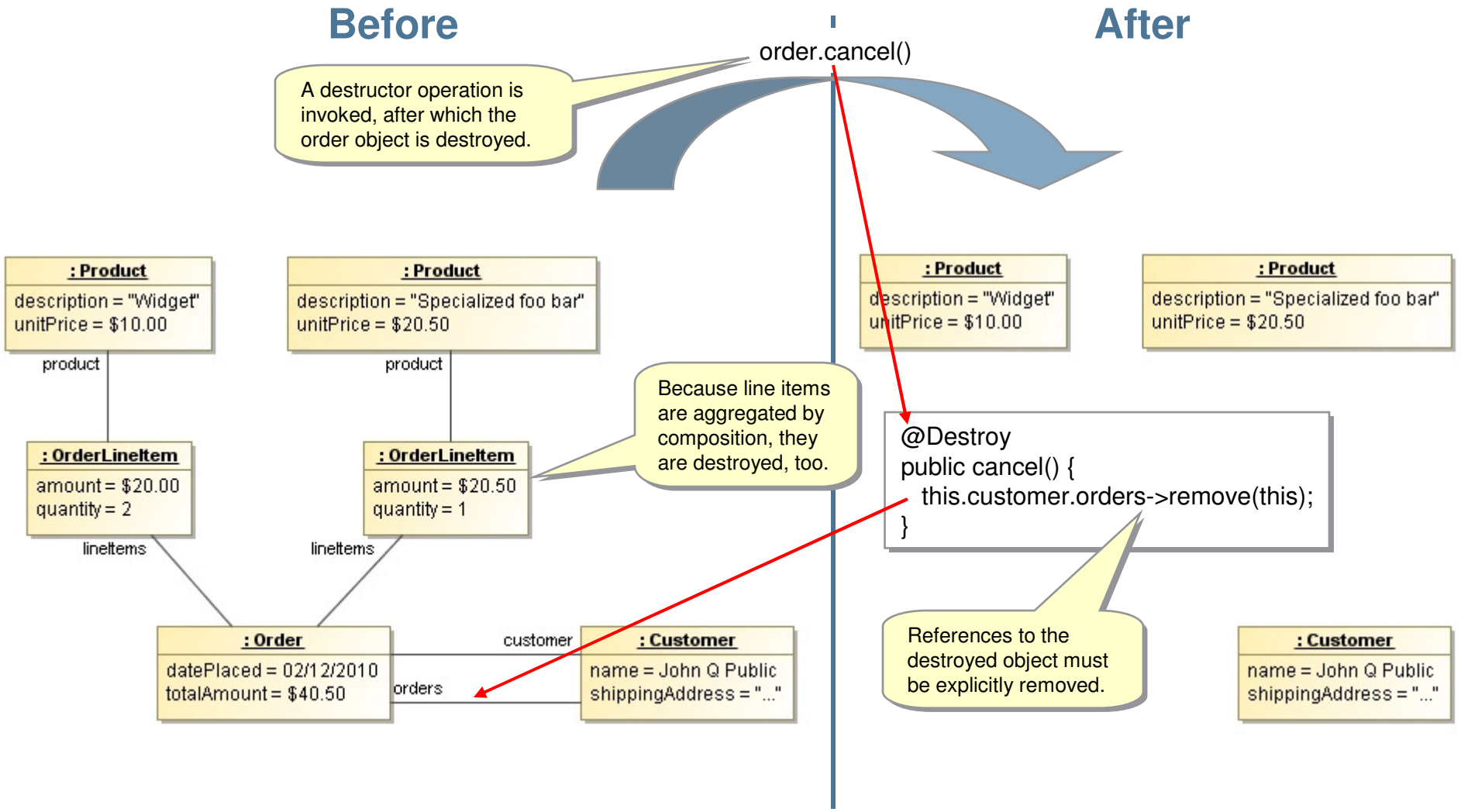
Creating an Order



Adding a Line Item



Canceling an Order

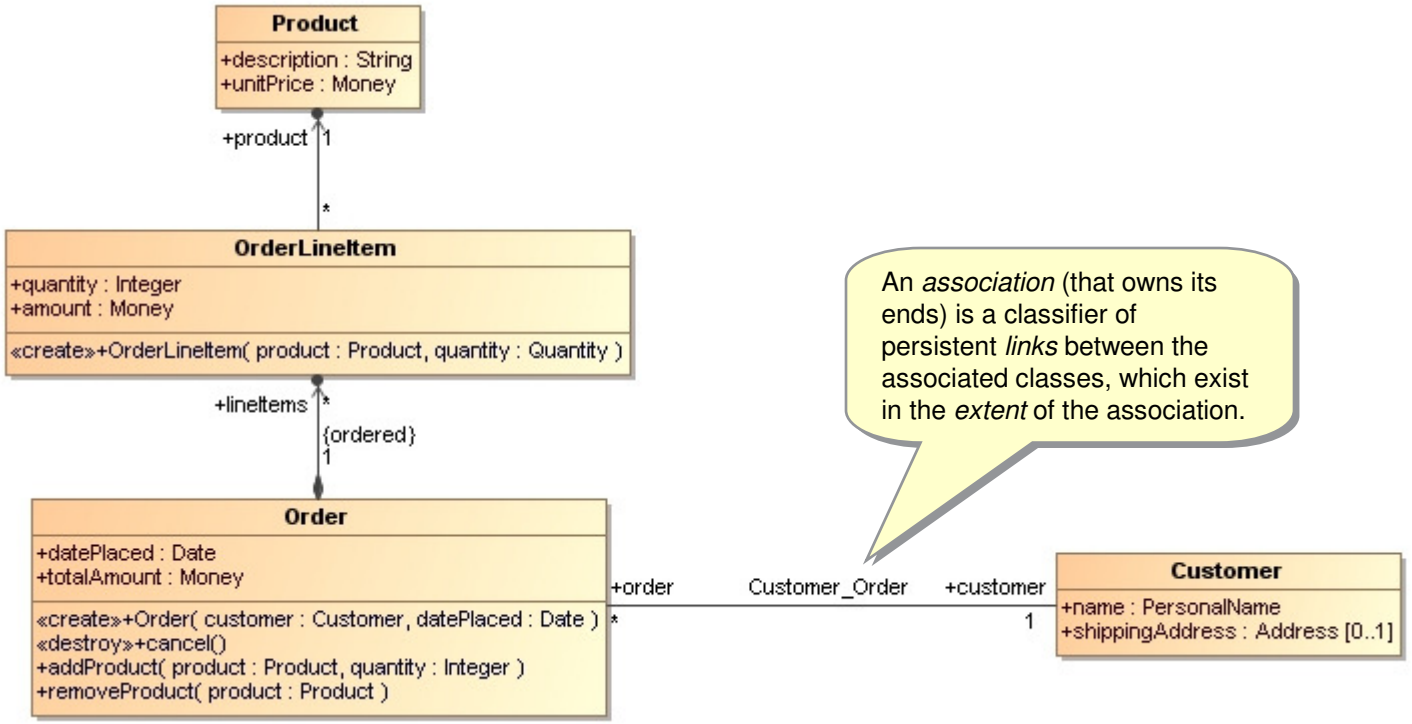


Associations

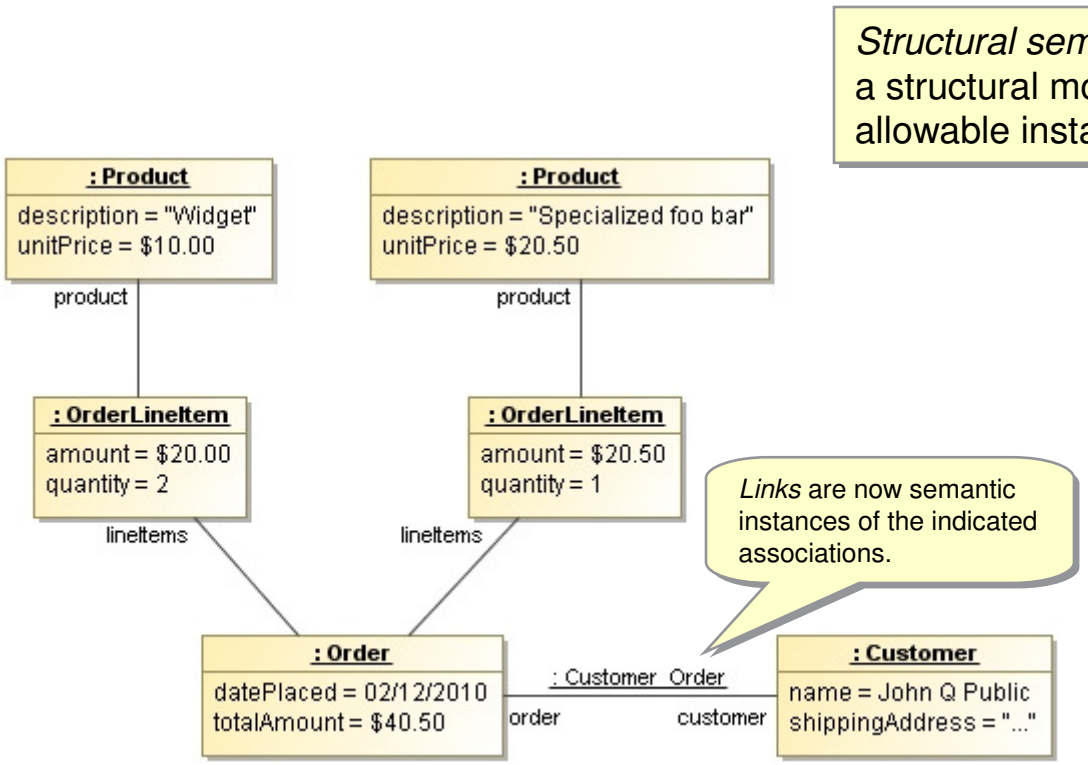
An *association* is a *classifier* whose instances are *links* that relate other instances.

- Classes and Associations
- Structural Semantics
- Behavioral Semantics

Classes and Associations



Associations: Structural Semantics

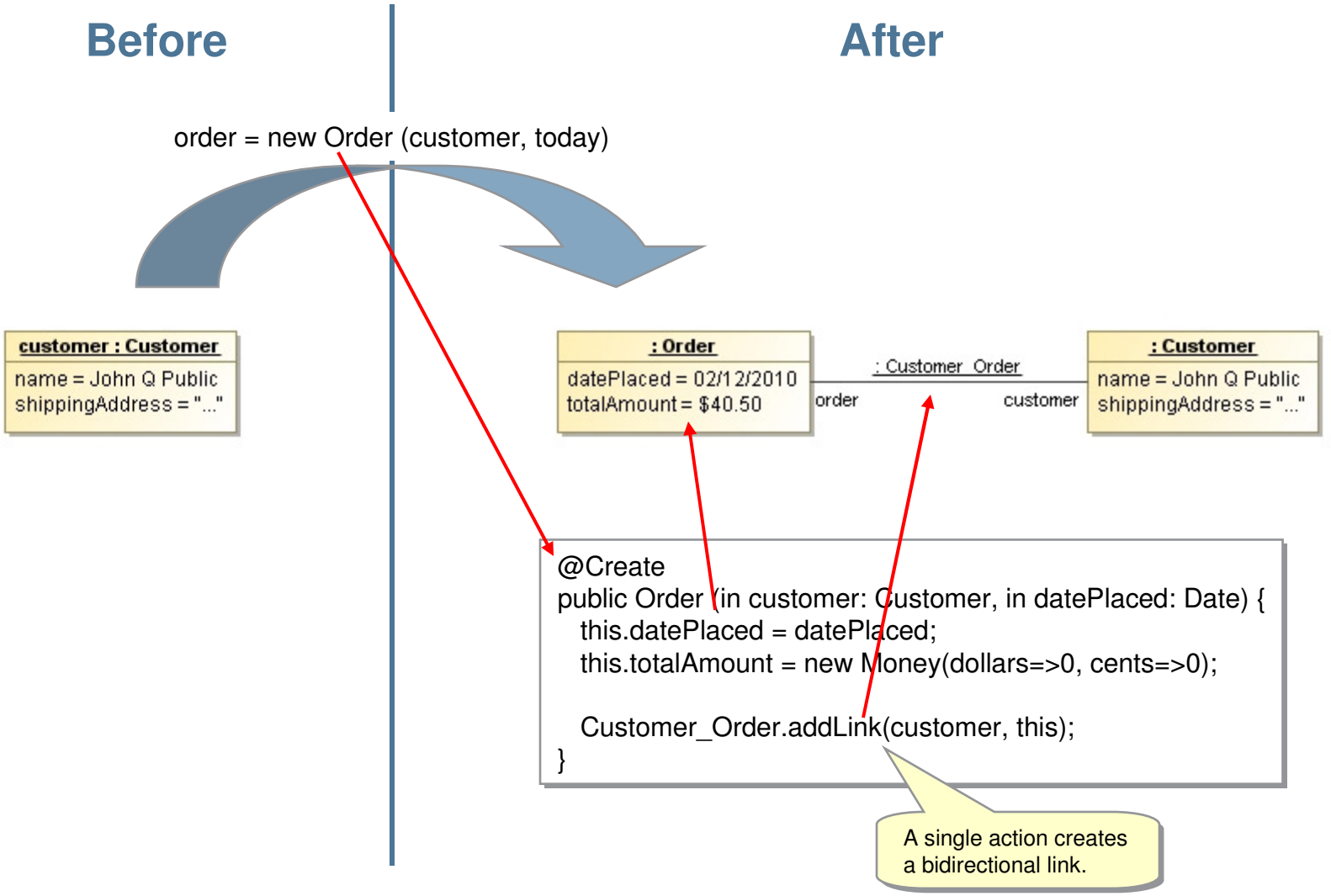


Associations: Behavioral Semantics

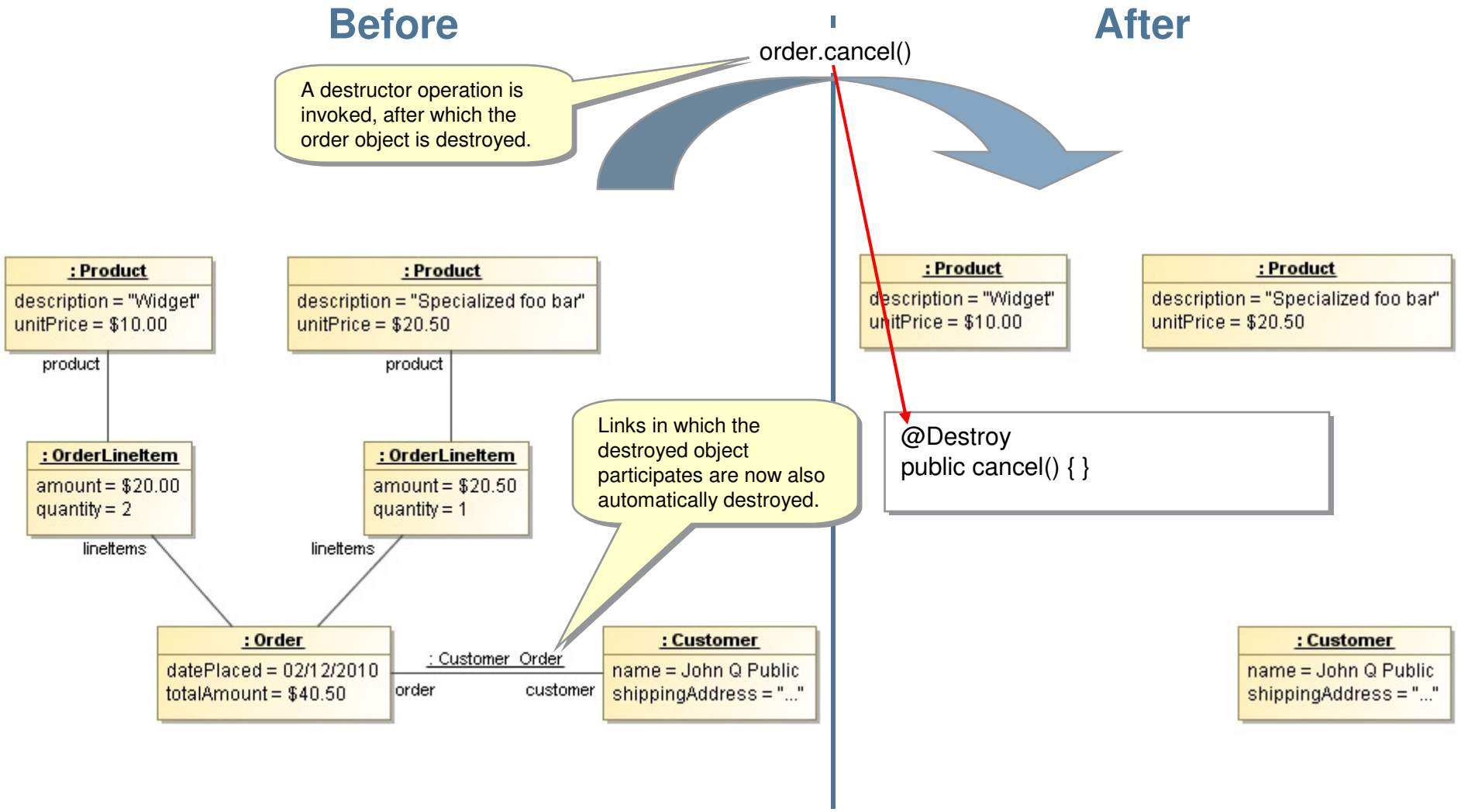
Behavioral semantics specify how a behavioral model changes the state of instances over time.

- Creating an Order (revised)
- Canceling an Order (revised)

Creating an Order (revised)



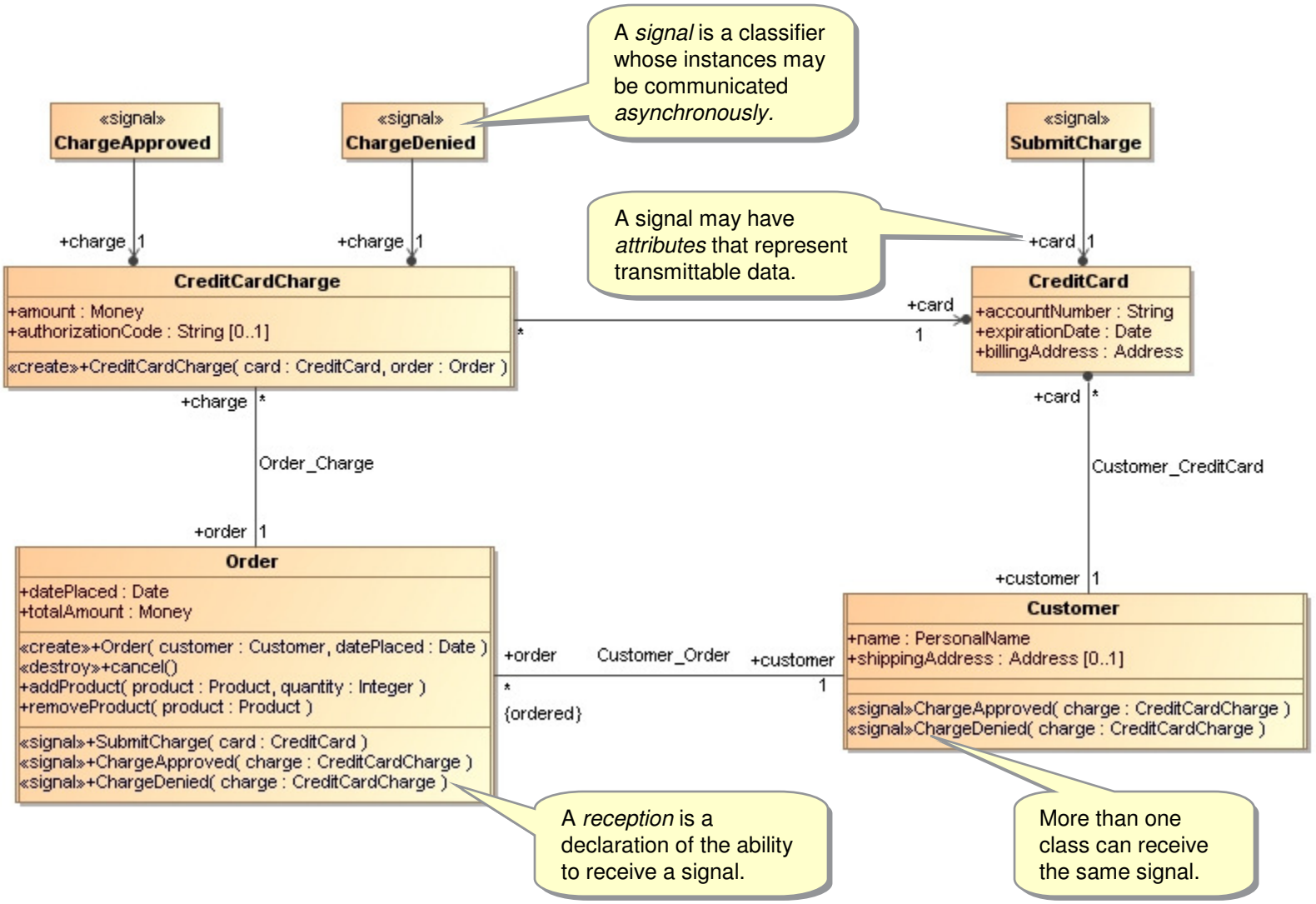
Canceling an Order (revised)



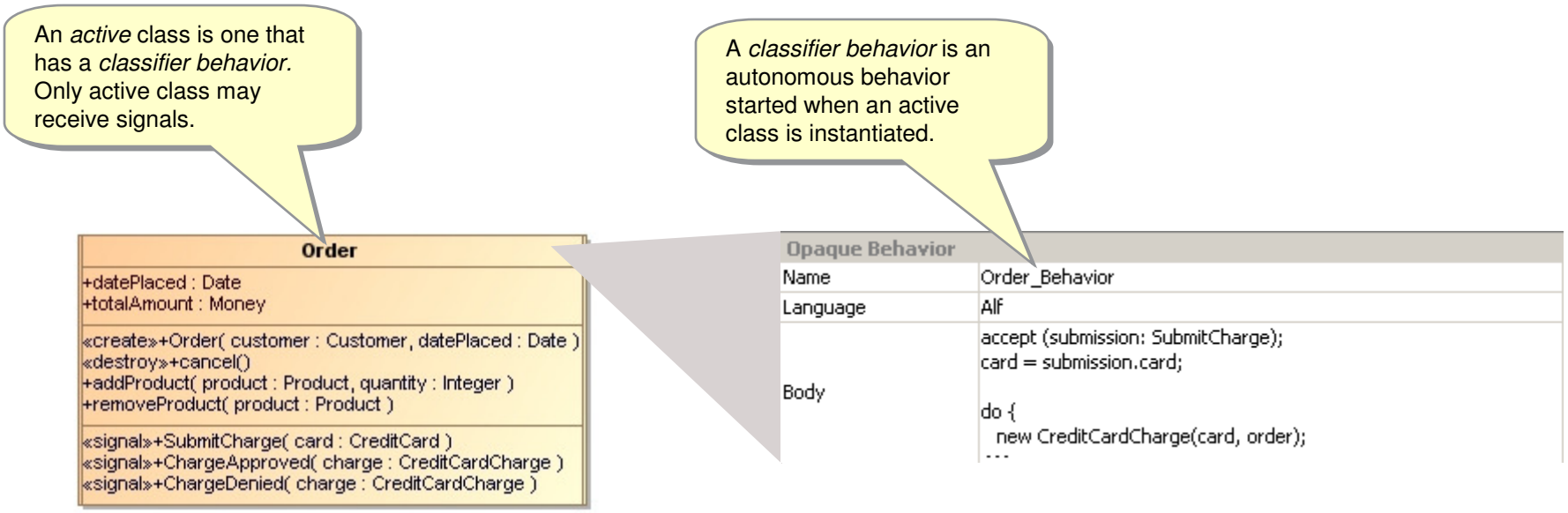
D. Asynchronous Communication

- Signals and Receptions
- Classifier Behaviors
- Asynchronous Behavior

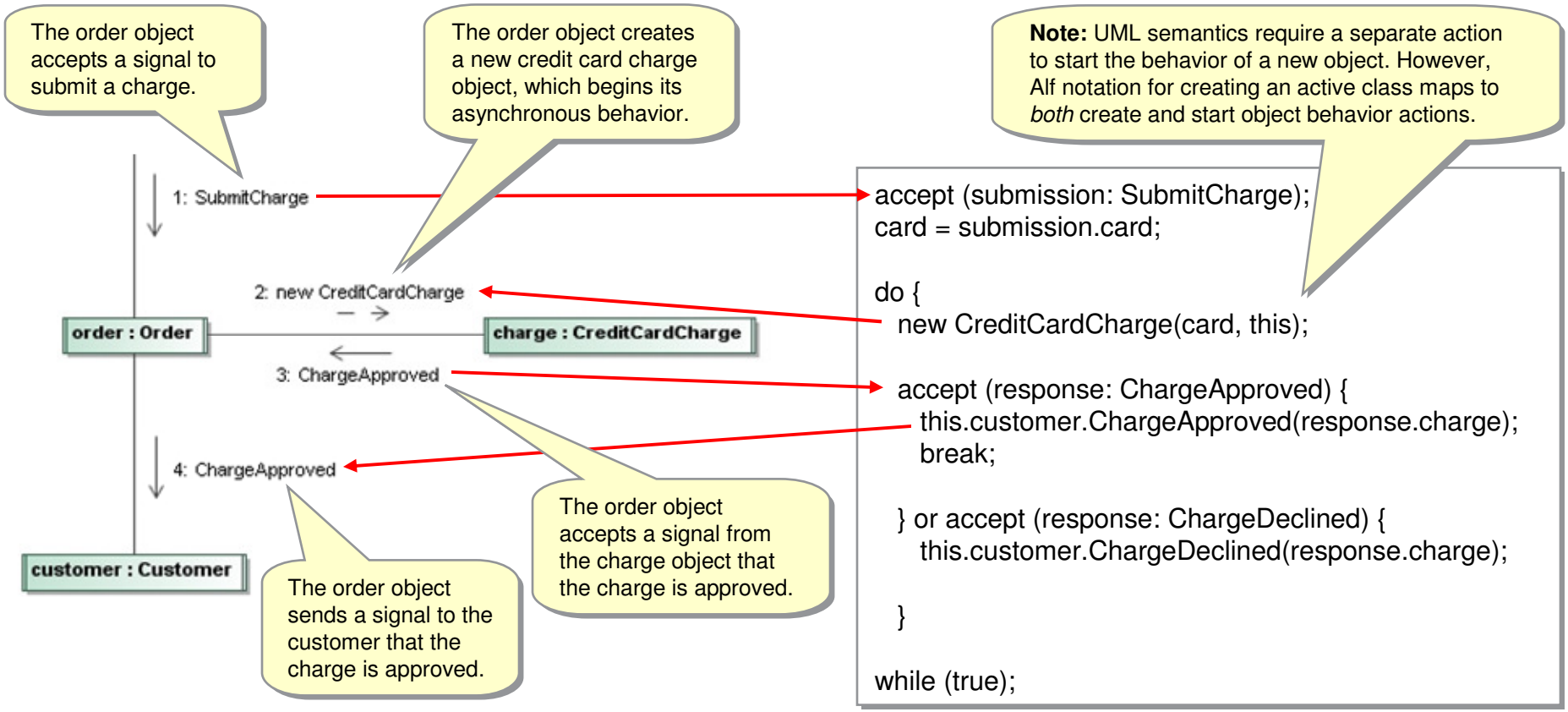
Signals and Receptions



Classifier Behaviors



Asynchronous Behavior



III. Standard Model Library

- Primitive Behaviors
- Collection Functions
- Collection Classes
- Basic Input/Output

Primitive Behaviors

- **Integer Functions** – Arithmetic, Comparison, Conversion
- **Unlimited Natural Functions** – Comparison, Conversion
- **Boolean Functions** – Logical Operations, Conversion
- **Bit String Functions** – Bit-wise operations, Conversion
- **String Functions** – Concatenation, Size, Substring

- *Coming in fUML 1.1:* Real Functions

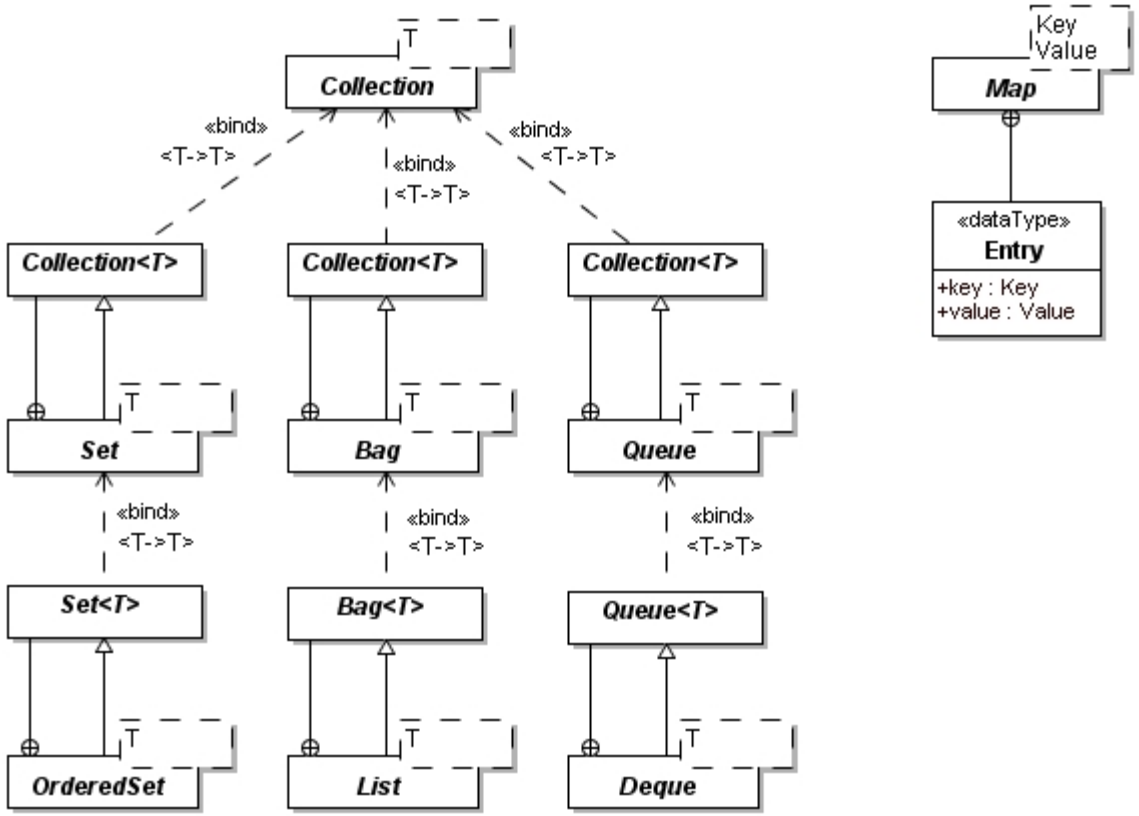
Collection Functions

Collection functions operate on sequences of values of any type.

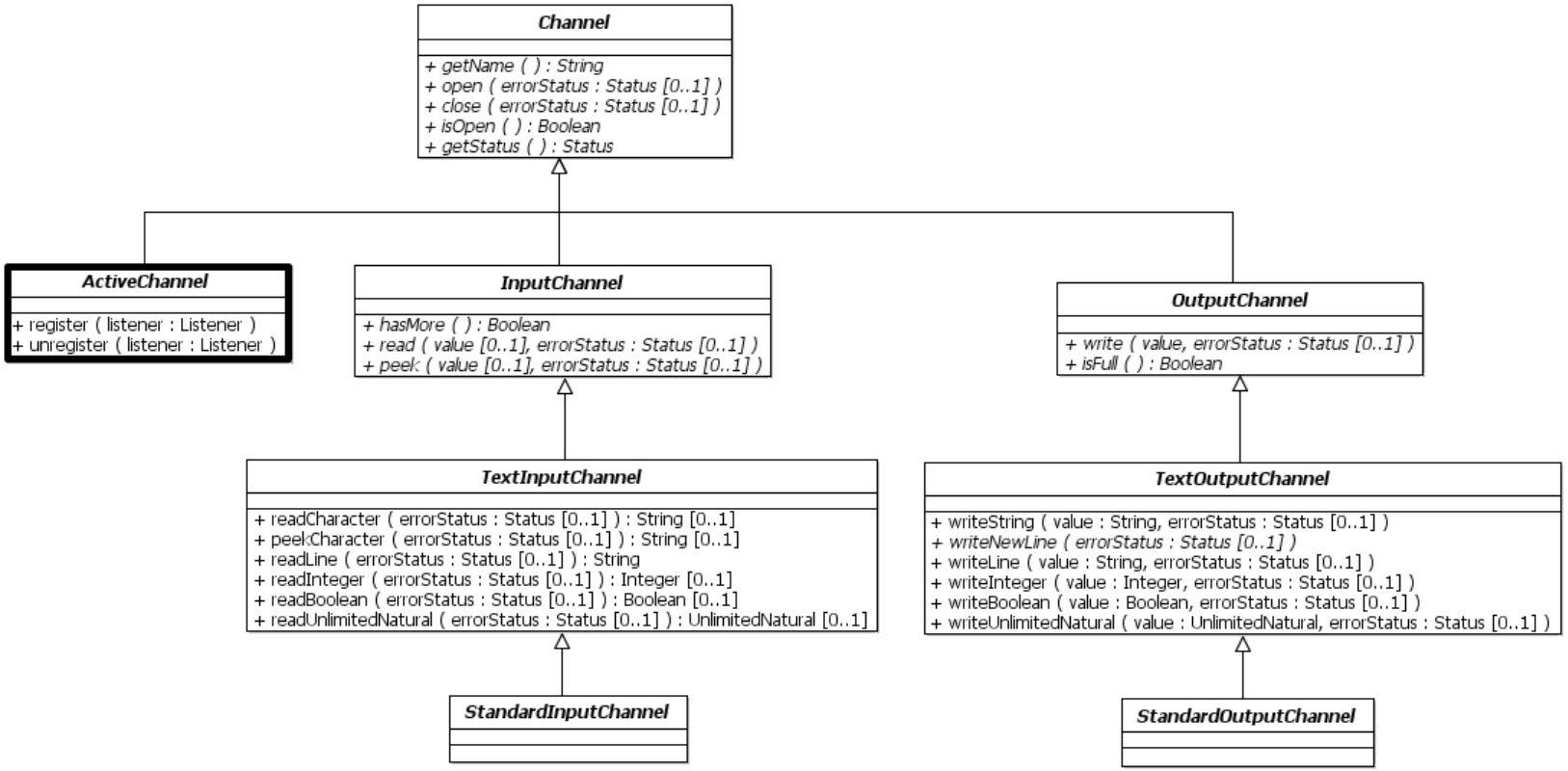
- Testing and accessing functions
  - Examples: seq->isEmpty(), seq1->equals(seq2), seq->at(n)
- Non-mutating funtions
  - Examples: seq2 = seq->including(x), seq3 = seq1->union(seq2)
- Mutating “in place” functions
  - Examples: seq->add(x), seq1->addAll(seq2), seq2->remove(x)

Collection Classes

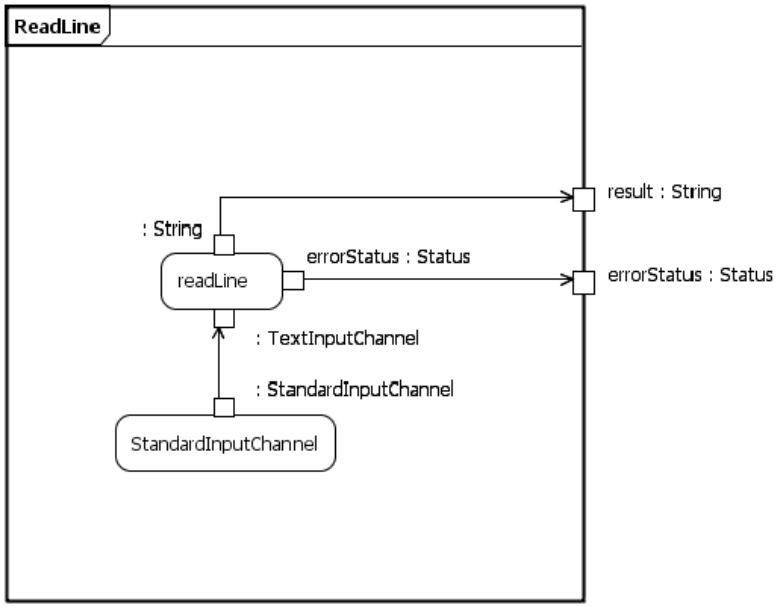
Collection classes define objects that represent collections of values of a given type.

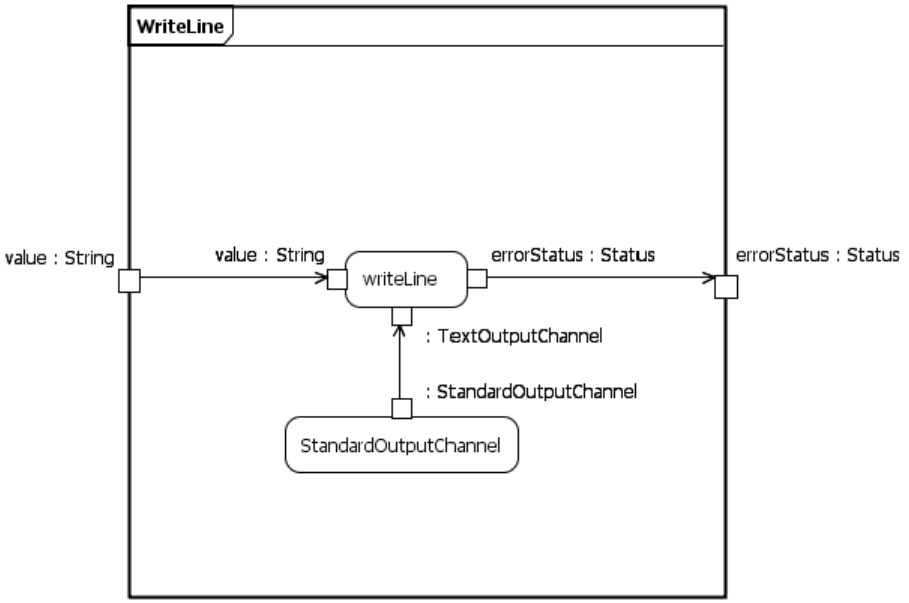


Basic Input Output: Channels



Basic Input Output: Reading and Writing Lines

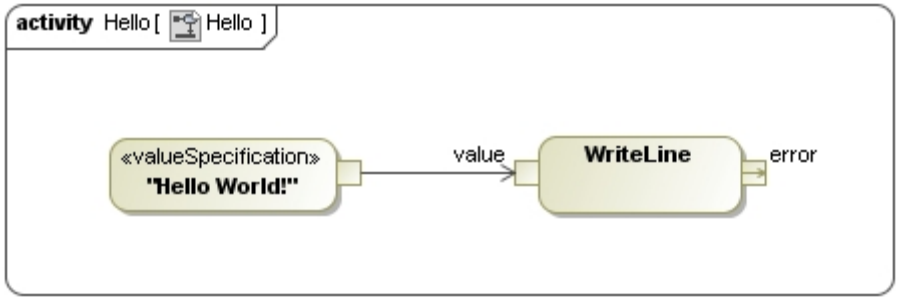





```
activity ReadLine
   (out errorStatus: Status[0..1]): String {

   return StandardIntputChannel.allInstances().readLine(status);

}
```

```
activity WriteLine
   (in value: String, out errorStatus: Status[0..1]) {

   StandardOutputChannel.allInstances().writeLine(result, status);

}
```

Hello World




```
activity Hello() {

  WriteLine("Hello World!");

}
```

References

- Foundational UML (fUML)
  - *Semantics of a Foundational Subset for Executable UML Models* standard, http://www.omg.org/spec/FUML/Current
  - fUML Open Source (Reference) Implementation Project, http://www.modeldriven.org/fuml
- Action Language for fUML (Alf)
  - *Action Language for Foundational UML (Alf)* standard, http://www.omg.org/spec/ALF/Current
  - Alf Open Source (Reference) Implementation Repository, http://lib.modeldriven.org/MDLibrary/trunk/Applications/Alf-Reference-Implementation
    - /dist – Full parser for the current version of the specification
    - Full implementation in progress
- Contact
  - Email: ed-s@modeldriven.com
  - Twitter: @seidewitz or http://twitter.com/seidewitz