Introduction to UML

What is UML

- UML stands for **Unified Modeling Language**
- It's a Language and not a Process:
  - It won't tell you how to think/model.
  - It will give you tools for expressing your thoughts.
- It is the work of "The 3 Amigos":
  - Grady Booch
  - Ivar Jacobson
  - James Rumbaugh
- UML is an emerging standard in software engineering and has been widely endorsed by the OMG (Object Management Group)

UML Diagrams

- UML tries to tackle both aspects of software systems:
  - Behavioral (***Dynamic***) aspect - how the different parts **interact** with each other.
  - Structural (***Static***) aspects - how the different parts **relate** to each other and with their environment.
- UML is a graphical language so it uses different diagrams to depict the above:
  - easy to understand.
  - fun to draw!
- Some UML diagrams can be forward engineered to actual code.
  - 0 > 1 rule - always prefer an automatic tool rather then doing it yourself - people are error-prone.
  - saves time.
- Some tools even let you reverse-engineer source code.

UML dynamic Diagrams

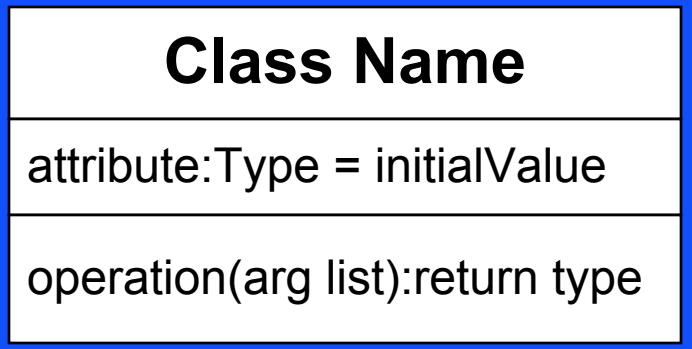
- Use case diagram
  - set of use cases and actors.
- Sequence diagram
  - emphasizes the time ordering of messages.
- Collaboration diagram
  - emphasizes the structural organization of objects that send and receive messages.
- State diagram
  - shows a state machine; especially important in modeling the behavior of an interface, class, collaboration.
- Activity diagram
  - shows the flow from activity to activity within the system.

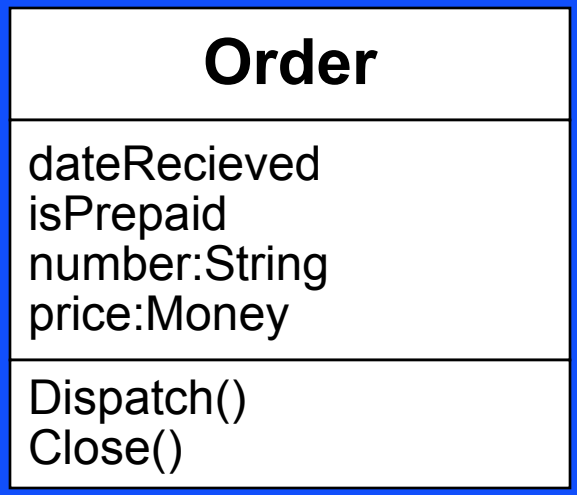
UML static diagrams

- Class diagram
  - set of classes, interfaces, collaborations and their. relationships
- Object diagram
  - illustrates data structures, the static snapshots of instances of the things found in class diagrams.
- Component diagram
  - illustrates static implementation view of the system; a component typically maps to one or more classes, interfaces or collaborations.
- Deployment diagram
  - illustrates static deployment view of a system; a node typically encloses one or more components.

Class diagram

- The best known feature of UML
  - You've probably seen them before...
- A Class consists of 3 parts
  - a name
  - it's attributes
  - it's operations
- You can assign access levels to every attribute/operation
  - e.g. public, protected, private.
  - the syntax defines additional notation for these constructs.
- Sometimes a more simplified version is used
  - when we're only interested in the class as en entity and not in it's attributes and operations.



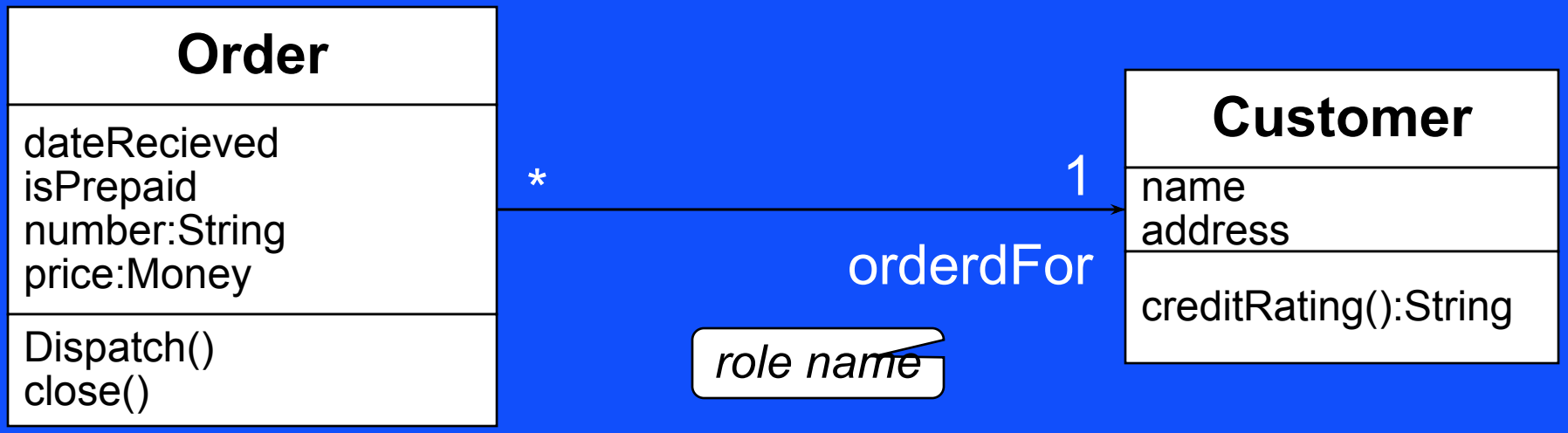


Relationships

- 4 kinds of relationships between classes:
  - Association (reference)
  - Aggregation
  - Composition
  - Generalization (inheritance)
- Relationships may be embellished by several constructs:
  - stereotypes
  - role names
  - multiplicity
  - constraints
  - navigability

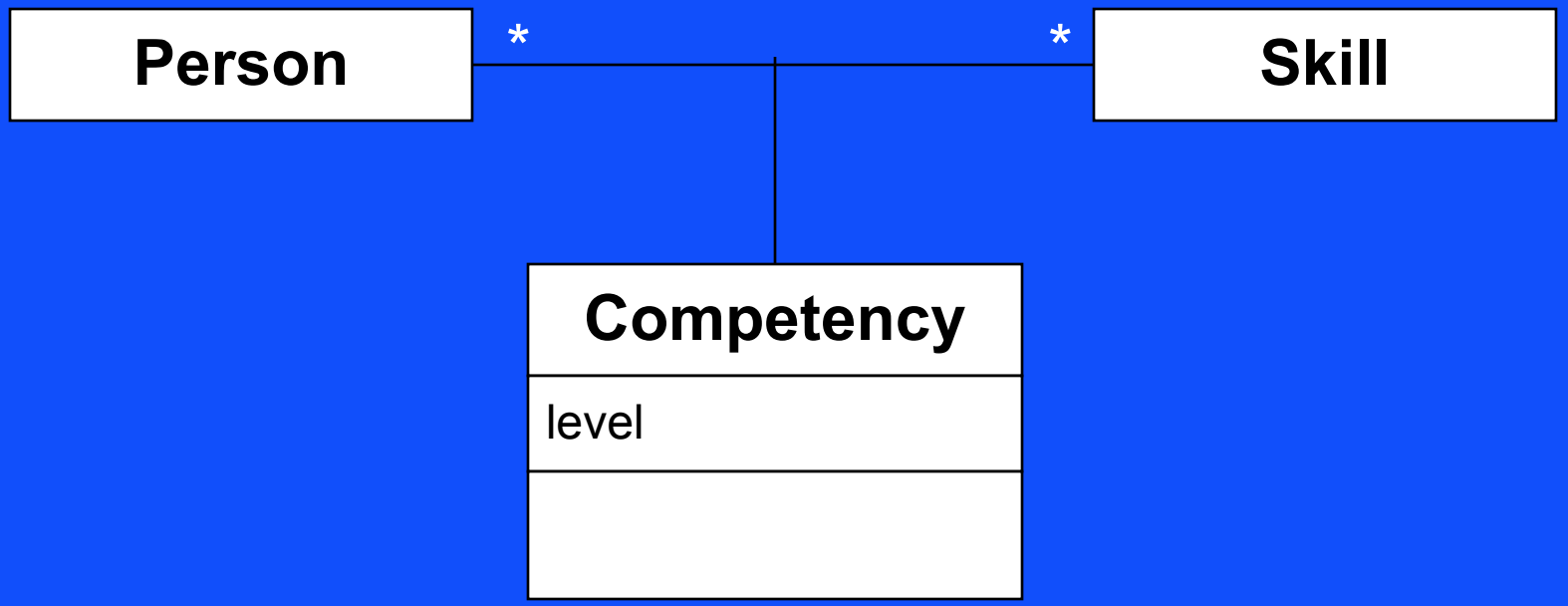
Association

- A relationship between instances of classes
  - usually implemented as a pointer or a reference.
- may include role assignments for each class
- may include constraints
- should include multiplicity
- should define navigability
- may be modeled as a class
  - special notation for this - next slide



Association class

- Captures the relationship between classes



Multiplicity

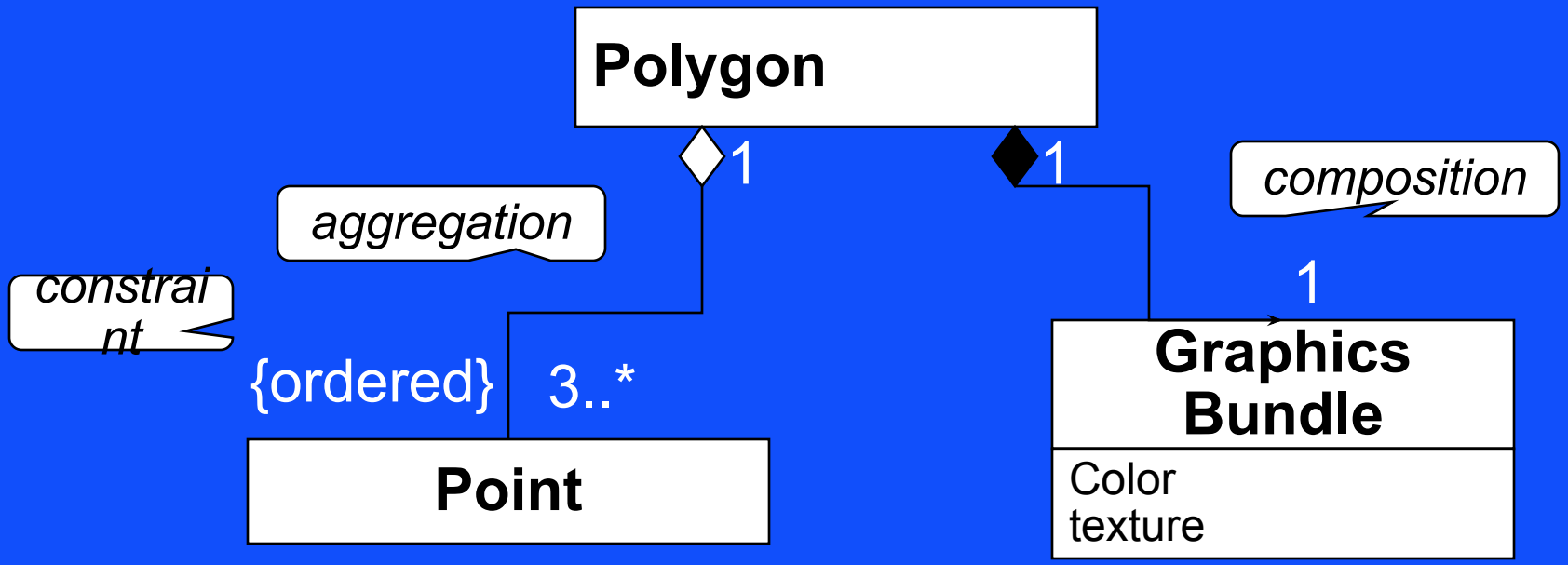
- How many instances (objects) of the associated class for a single instance of the associating class.
  - 1
  - 0..1
  - * (any number)
  - 1..* (at least one)
  - anything else: 2..4, 7, 6..* etc.
- In the following example
  - Every Order is associated to exactly one Customer.
  - A Customer may have any number of orders.

Association Vs Attribute

- Use attributes for small simple classes
  - strings
  - dates
  - money objects
  - primitives (int, real etc.)
  - other non-modeled entities.
- Use association for all the rest

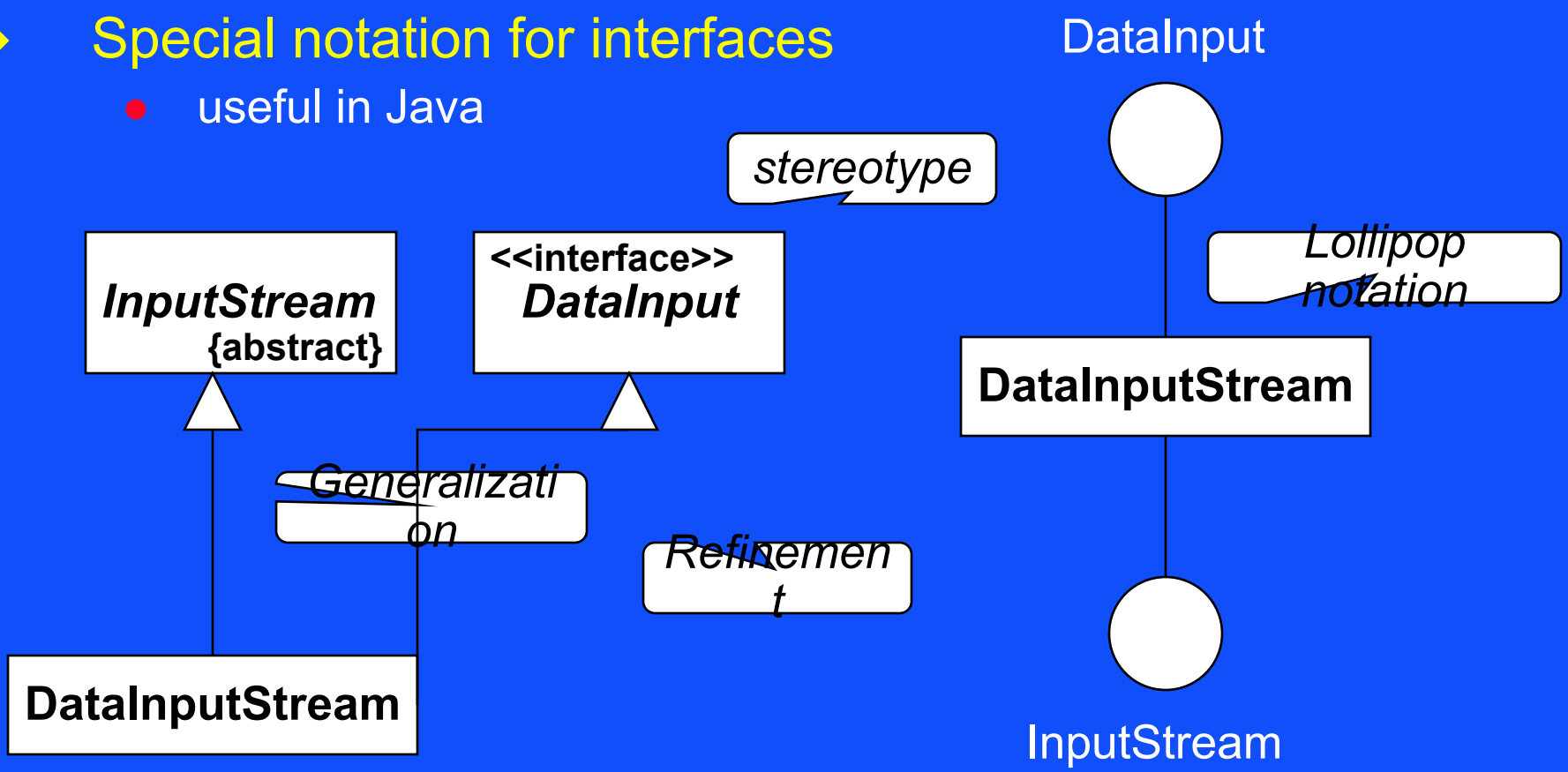
Aggregation & Composition

- Aggregation Vs. Association ?
  - Peter Coad gave the example of an organization and it's clerks.
  - A polygon and it's points.
- Composition is easier to explain
  - a part that is inseparable from the whole.
  - a polygon and it's graphics attributes.
  - usually expected to live and die with the whole.



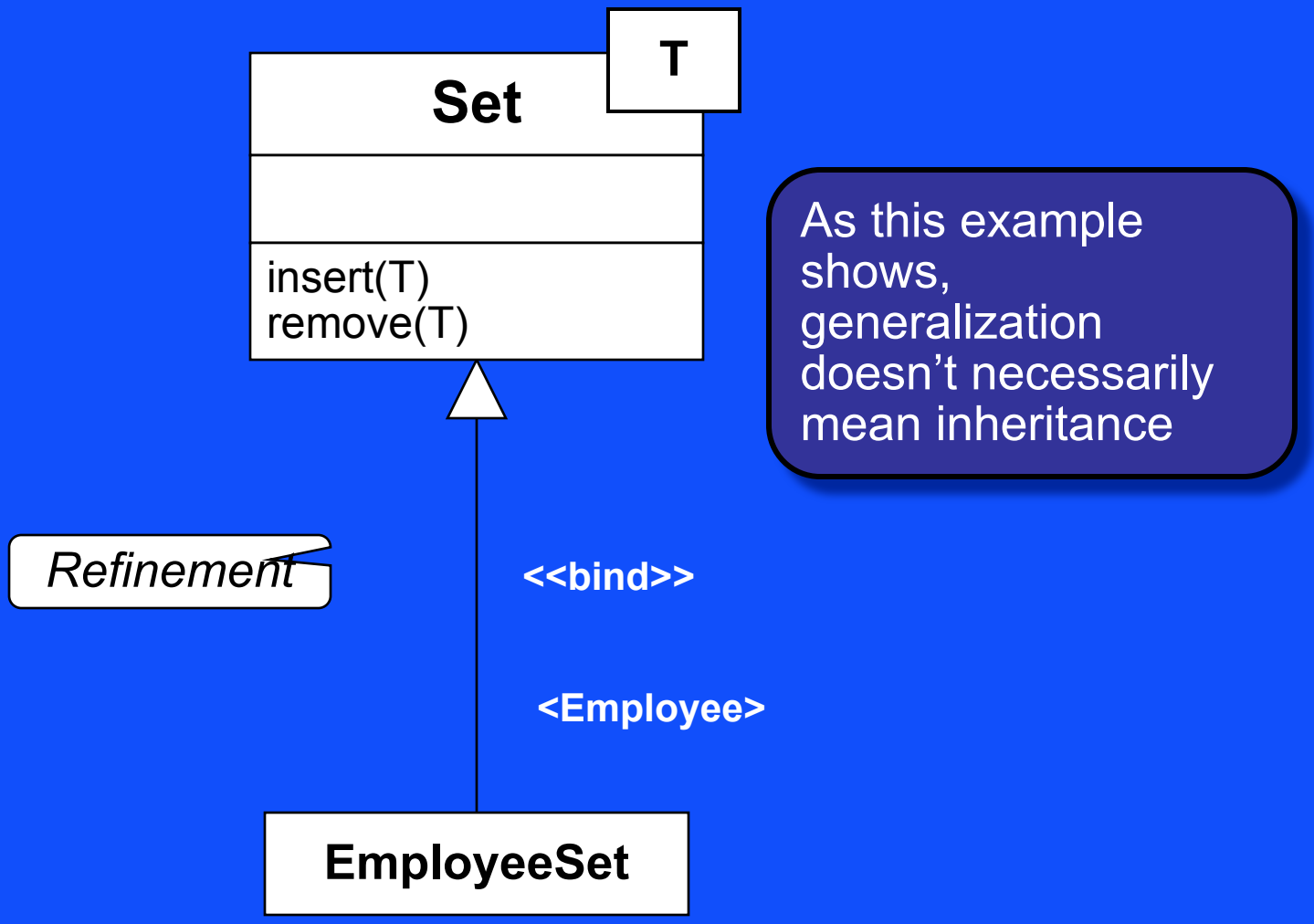
Generalization

- Inheritance
  - all base class attributes and operations are also part of the derived class.
  - e.g. Manager from Employee
- Special notation for interfaces
  - useful in Java



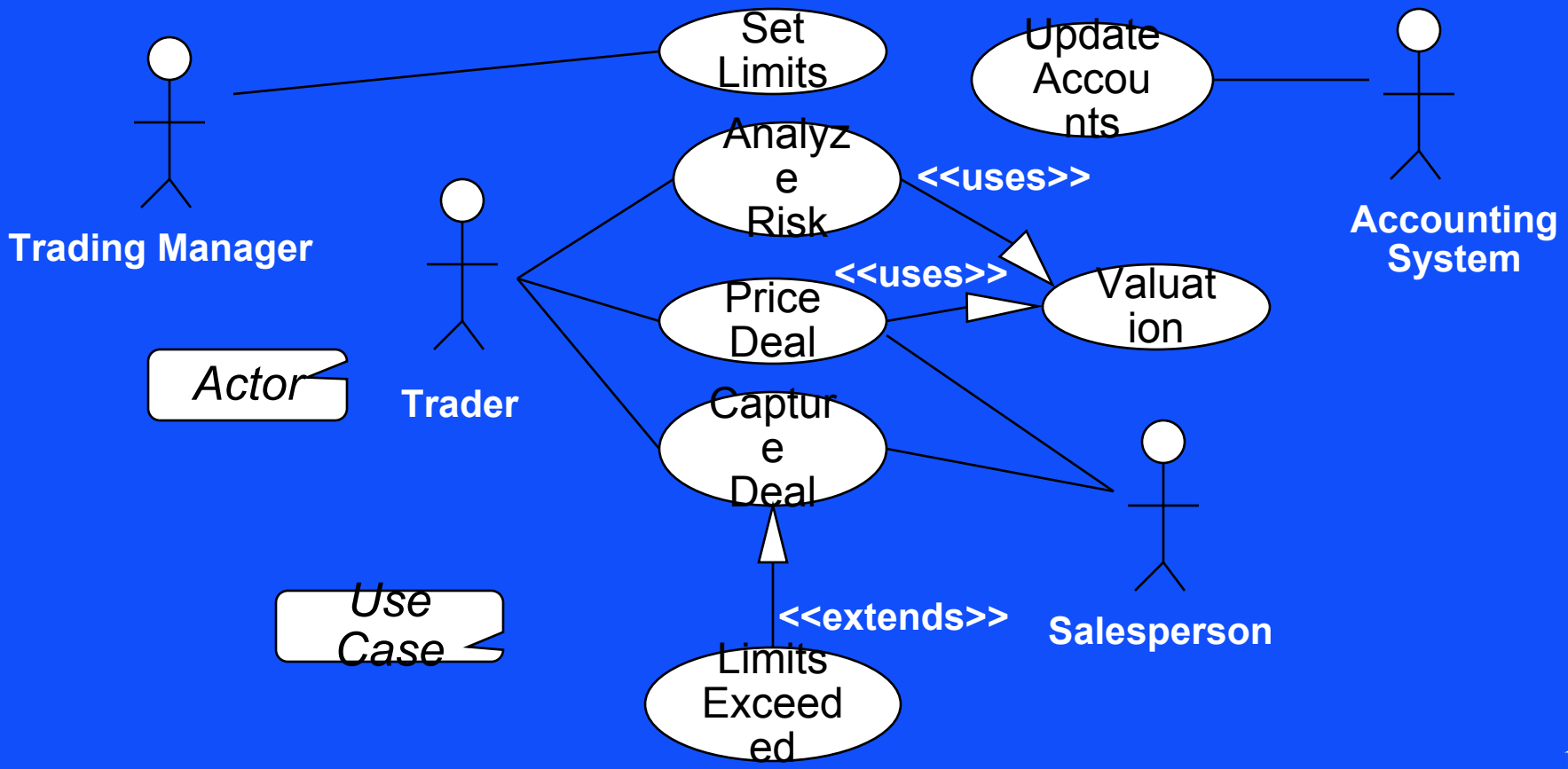
Parameterized Classes

- Also known as templates or generics



Use Case diagrams

- A.k.a scenarios or flows
- composed of Use Cases and Actors



Use Cases

- A typical interaction between a user and a computer system
  - captures some user-visible function.
  - achieves a discrete goal for the user.
- use cases may use other use cases
  - which in turn use other use cases…
  - the <<uses>> stereotype.
- use cases may extend other use cases
  - for example a scenario with an error extends a basic scenario.
  - The <<extends>> stereotype.
- use cases should be used like spices
  - rule of thumb - a typical complex system should include ~20 use cases (Coad & Yourdon - *"Object Oriented Analysis"*).

Actors

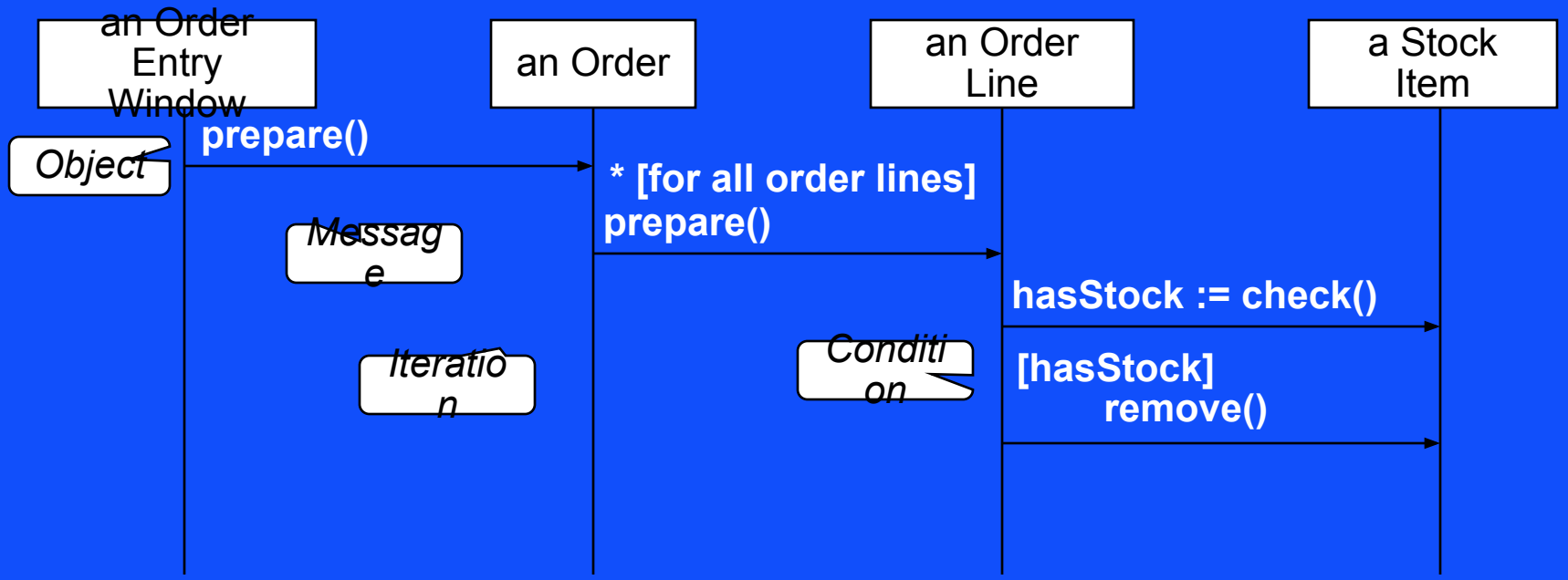
- A role that a user plays with respect to the system
  - an actual entity might be represented by more than one actor.
- Actors need not be human
  - a printer can be an actor who uses the use case “read file”.
  - The Accounting System in the previous example is probably a process and not an actual human-being.

Interaction Diagrams

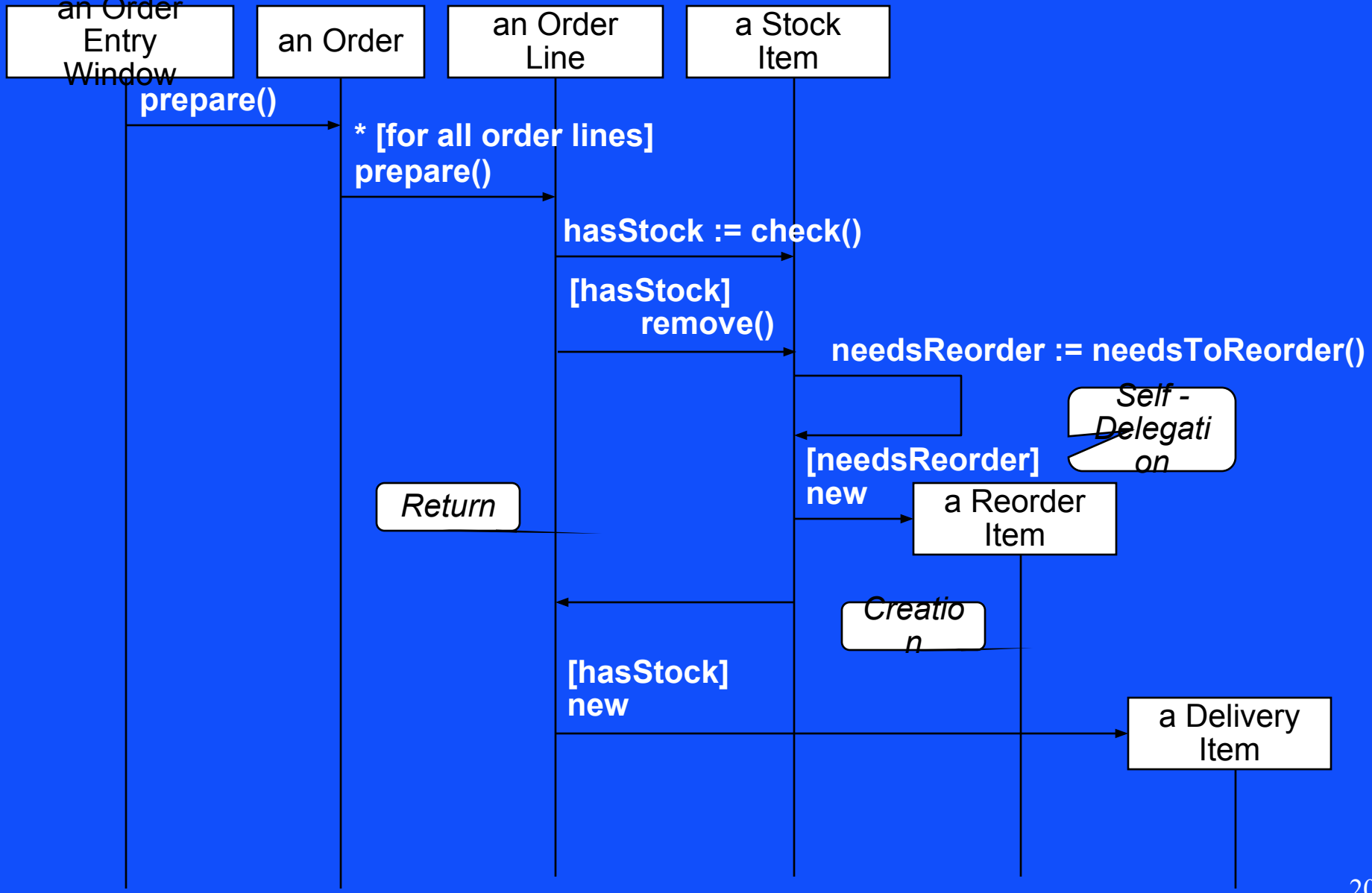
- Models that describe how groups of objects collaborate in some behavior
- typically captures the behavior of a single use case
- shows a number of example objects and the messages that are passed between these object within the use case
- 2 kinds of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams

Sequence Diagrams

- An object is shown as a box at the top of a dashed vertical line (the object's lifetime)
- messages
  - each message is an arrow between lifelines of 2 objects.
  - messages can show self-delegation.
  - message may be conditioned (sent only if condition is true).
  - iteration marker (e.g. sending a message to all elements in a collection).

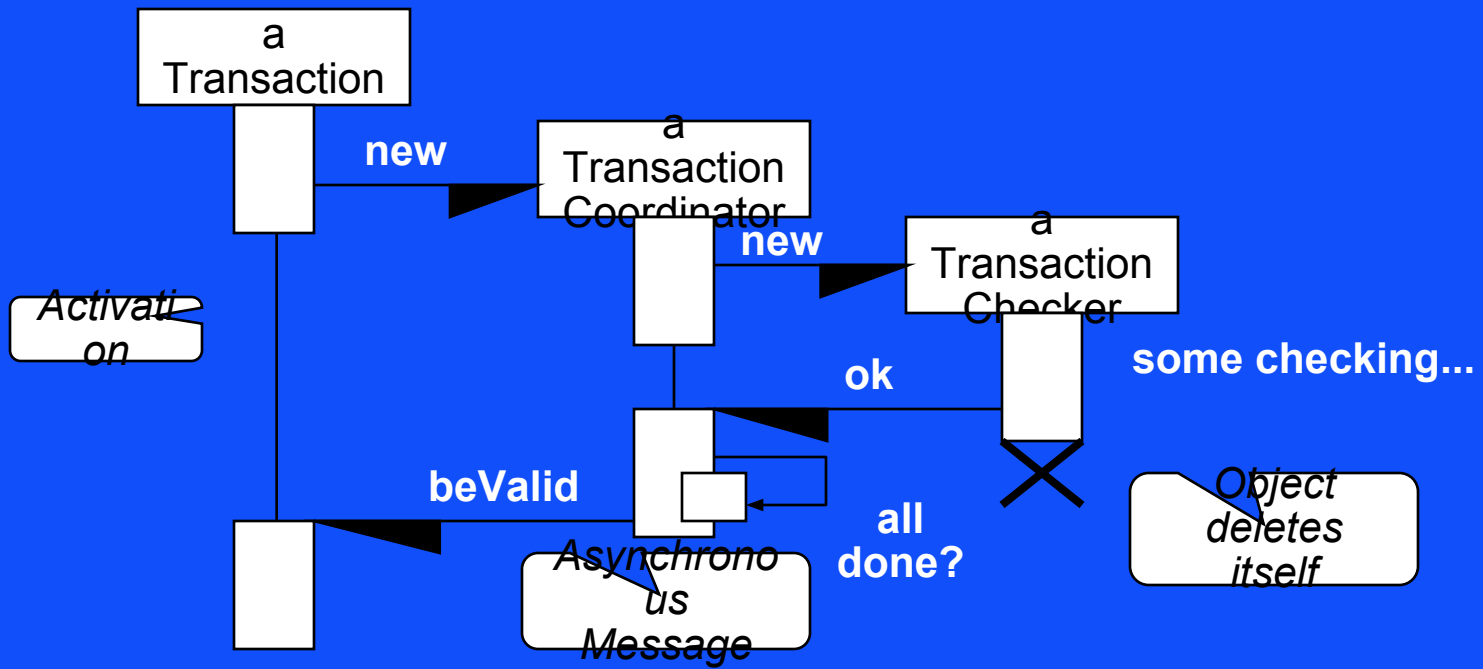


Sequence Diagrams (cont')



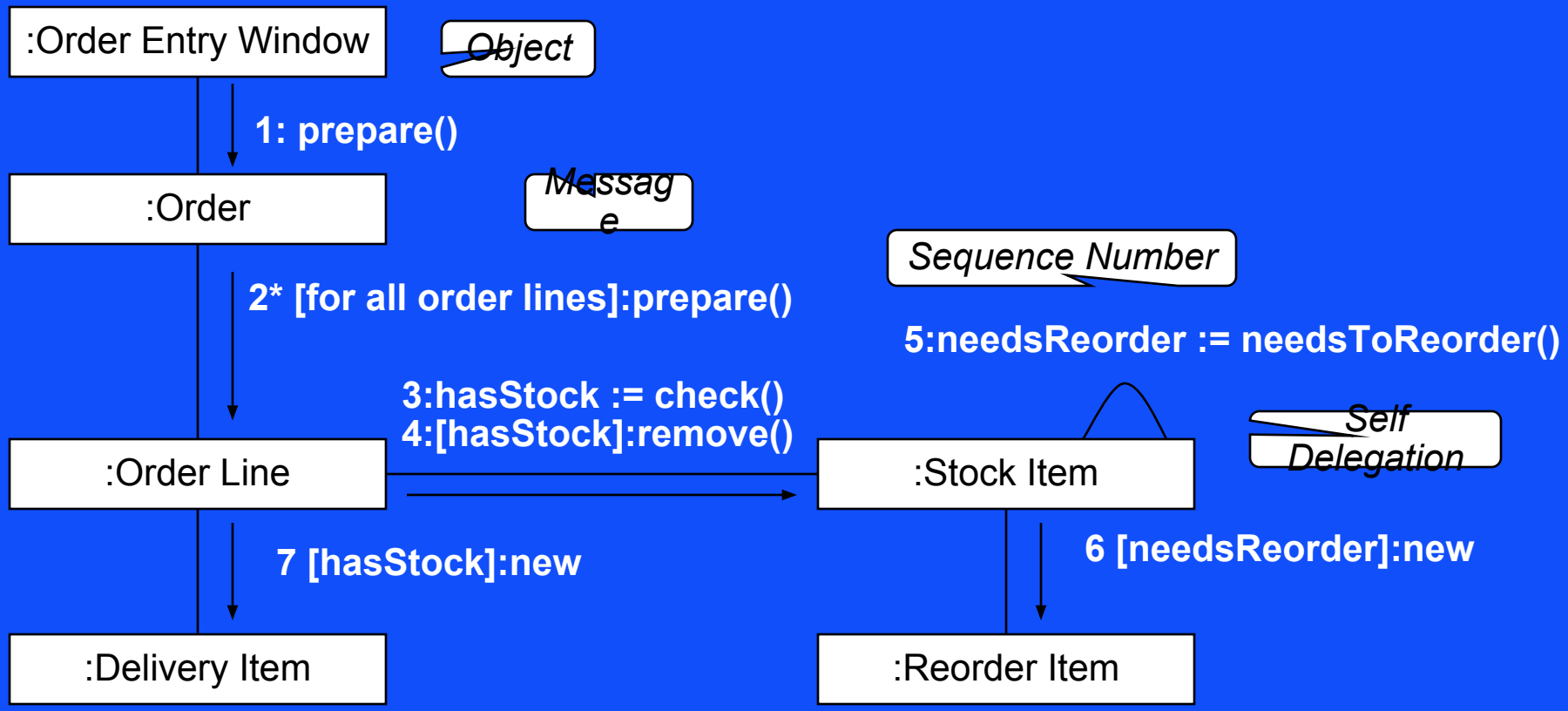
More on Sequence Diagrams

- Object deletion is marked with an X
- sometimes use activations a.k.a. lifebars instead of lifelines
- notation for concurrent sequence diagrams
  - the half arrow stands for an asynchronous message
  - rigorous use of activations



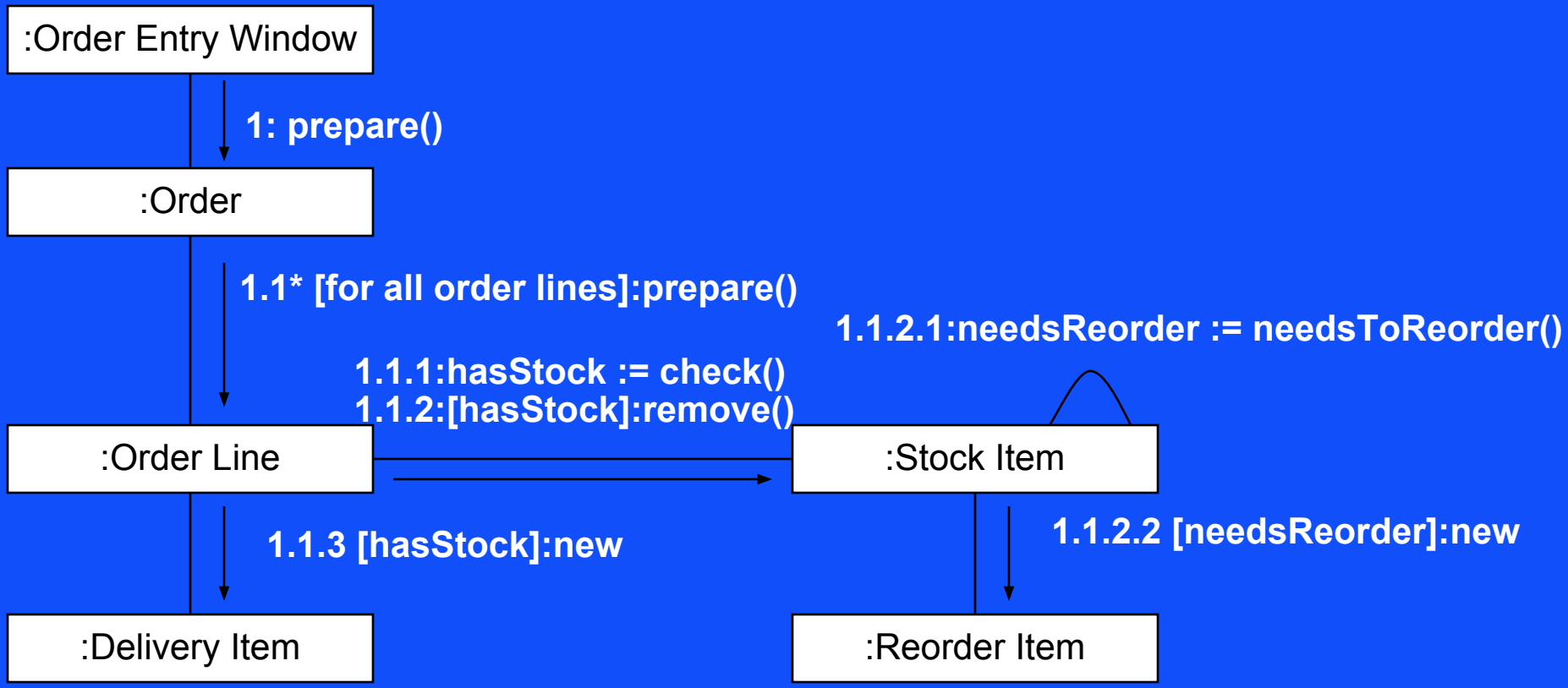
Collaboration Diagrams

- An object is shown as a box
- lines indicate relationships
- arrows show control flow
- numbering for actions



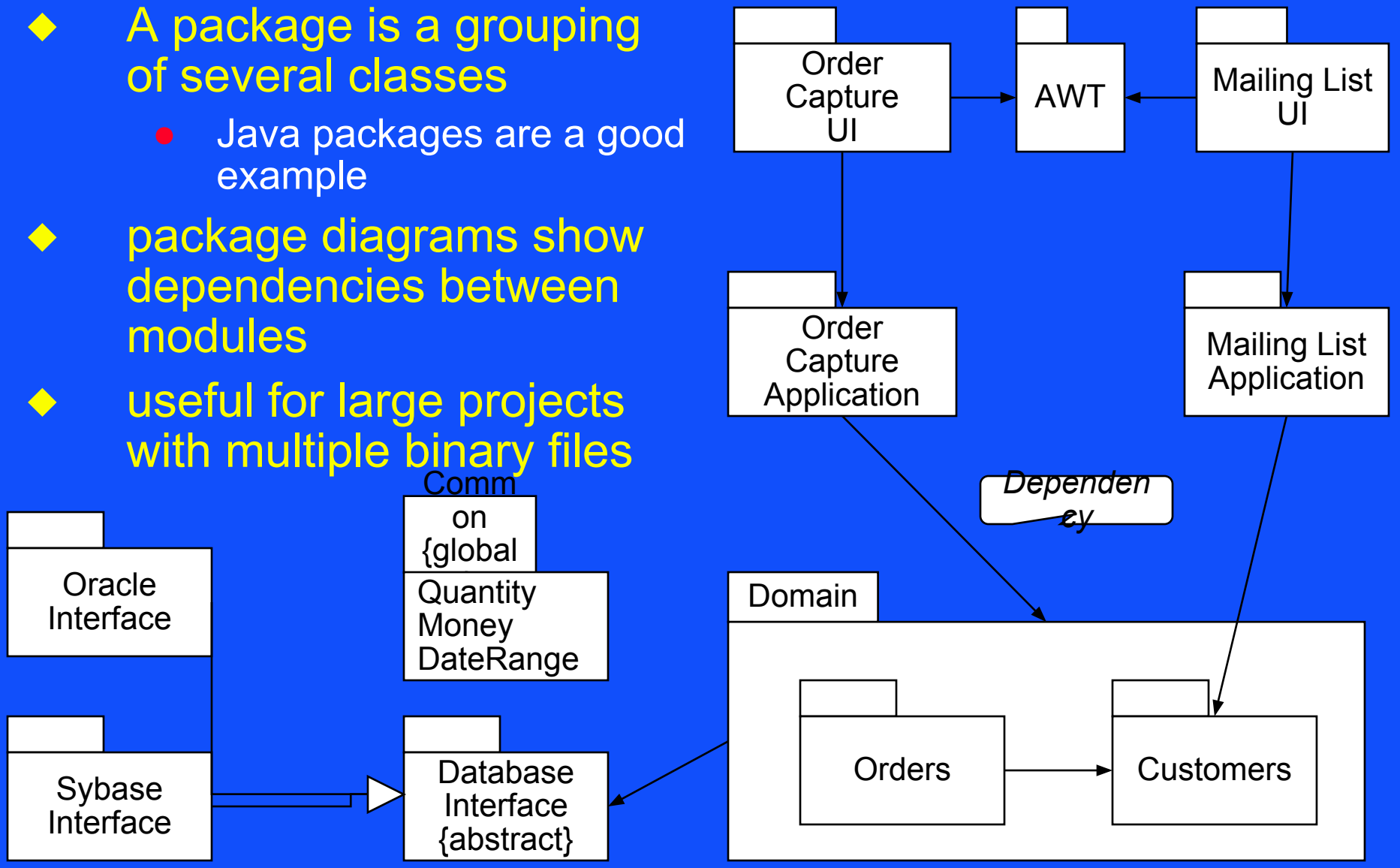
Collaboration Diagrams (cont')

- Same diagram with decimal numbering



Package Diagrams

- A package is a grouping of several classes
  - Java packages are a good example
- package diagrams show dependencies between modules
- useful for large projects with multiple binary files



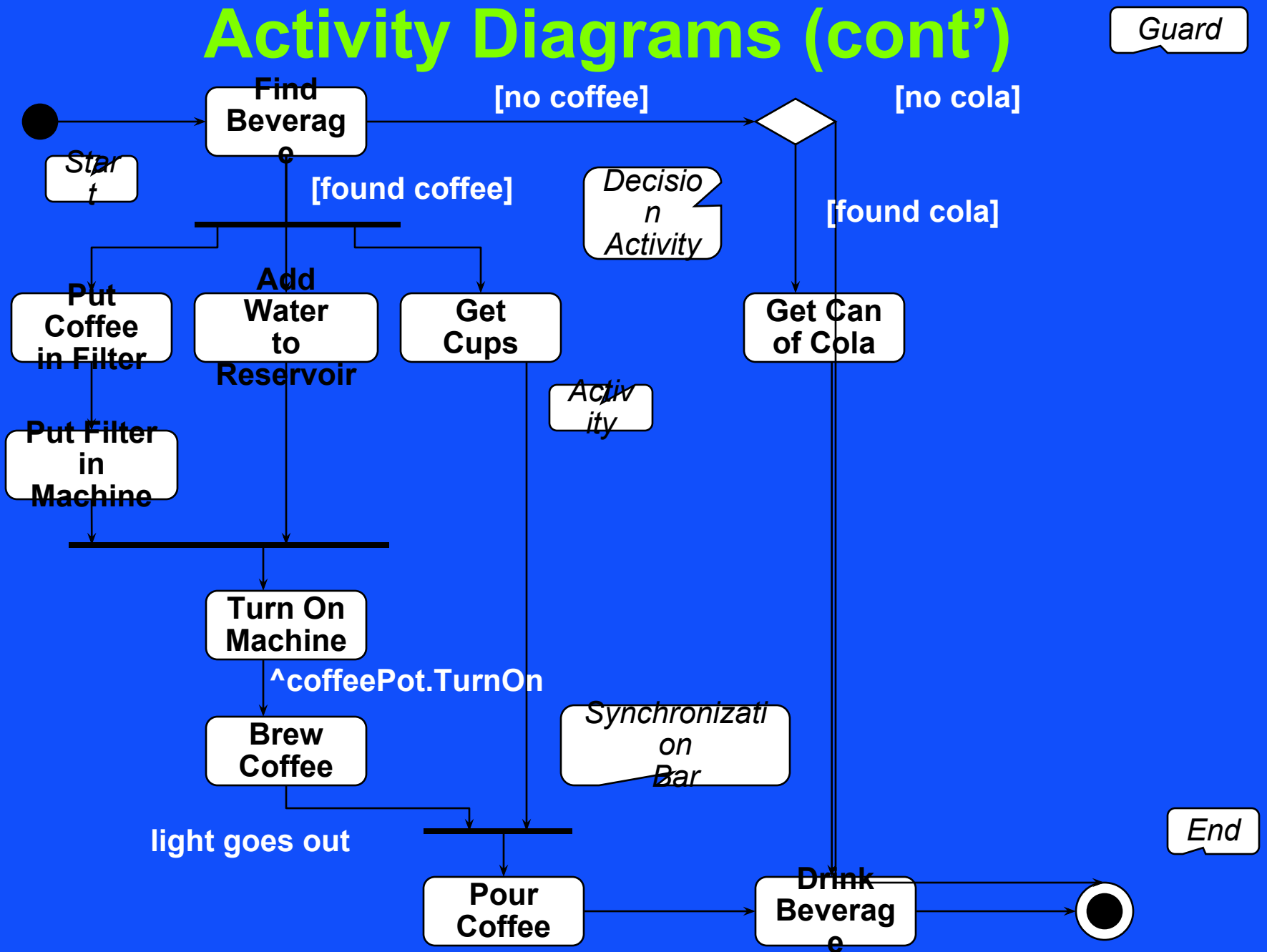
State Diagrams

- Based on the work of David Harel
- See cs236368 - Formal Specification for Complex Systems

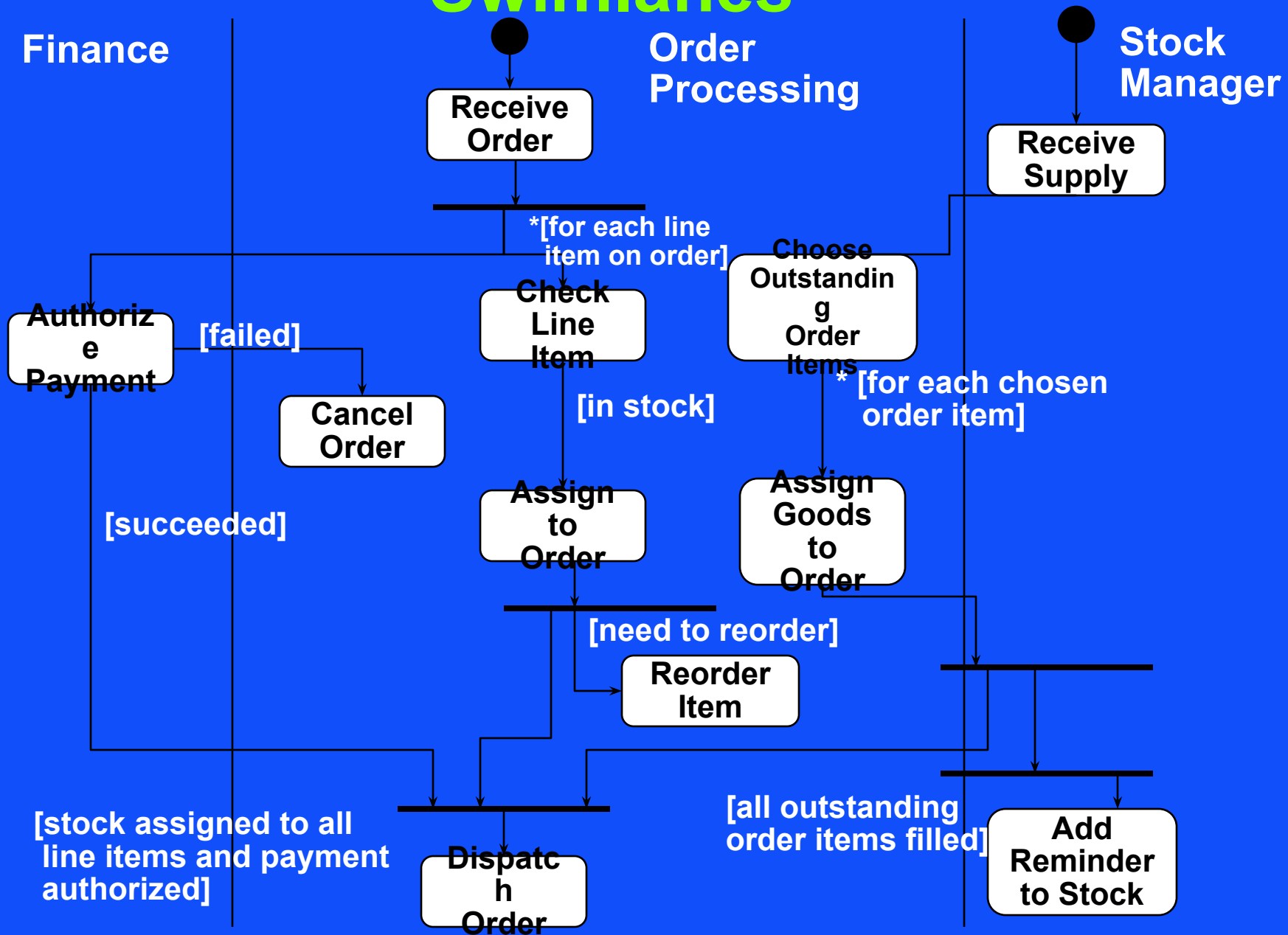
Activity Diagrams

- As Fowler describes: “…one of most unexpected parts of UML”
- Sort of flow-charts but much more powerful.
- Comprised of:
  - activities
  - synchronization bars
- the coffee-brake example (next slide)

Activity Diagrams (cont')



Swimlanes

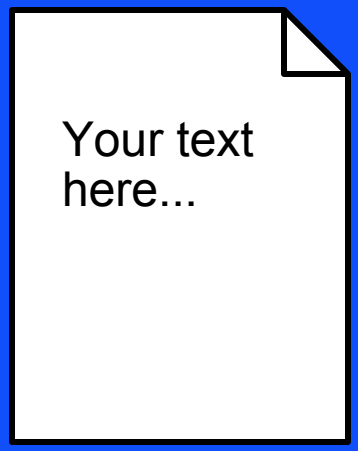

Finance
Order Processing
Stock Manager
Receive Order
Receive Supply
*[for each line item on order]
Authorize Payment
[failed]
Cancel Order
Check Line Item
[in stock]
Choose Outstanding Order Items
* [for each chosen order item]
[succeeded]
Assign to Order
Assign Goods to Order
[need to reorder]
Reorder Item
[stock assigned to all line items and payment authorized]
Dispatch Order
[all outstanding order items filled]
Add Reminder to Stock

Component & Deployment Diagrams

- Show the physical breakdown of a system.
- Consult documentation for more details.

Notes

- “If you don’t have a notation for it - use a note!”
- Can be attached to practically any UML construct
- Can be just “hanging in the air”
- Use them for providing additional explanations and key points to remember



Tools of the Trade

- Rational Rose
  - The same company that employs the 3 amigos.
  - Demo version (only C++ code generation and 30 classes Max.) available - free.
  - Full version includes also Java code generation.
  - `http://www.rational.com/products/rose/index.jtmpl`
- I-Logix Rhapsody
  - Includes state diagrams
  - emphasis on real-time applications
  - demo CD available on request
  - `http://www.ilogix.com`

Recommended Reading

- ***UML Distilled*** - Martin Fowler
  - a good quick & dirty reference for UML.
  - was used as a source for these slides.
- ***The unified modeling language user guide*** - G. Booch, J. Rumbaugh, I. Jacobson
  - long…
- ***The unified modeling language reference manual*** - J. Rumbaugh, I. Jacobson, G. Booch,
  - even longer…