CSCC01 – Introduction to Software Engineering

Unified Modeling Language

Why Modelling?

- Communicate ideas effectively
- Understand a codebase
- Improve design
- Identify problems

What is UML?

- A modeling language, not a method
- Used to provide abstraction for object-oriented design
  - Forward design (before writing the code)
  - Backward design (after writing the code)

Types of UML diagrams

- Structural
  - Static view of the system
  - E.g. Class Diagram, Package Diagram
- Behavioral
  - Dynamic view of the system
  - E.g. Use Case Diagram, State Diagram

Fowler, M. (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language* , Addison-Wesley , Boston, MA .

Class Diagram

- A class diagram describes the classes in an object-oriented system and the various kinds of static relationships among them
- The following relationships can be represented in class diagrams:
  - Association (Aggregation, Composition, Dependency)
  - Generalization (Inheritance, Implementation)

Class Diagram - Classes

- Classes are represented as rectangles containing three compartments:
  - Class name
  - Attributes
  - Operations
- The visibility of attributes/operations can be specified using the symbols + - # ~ as follows:
  - public (+)
  - private (-)
  - protected (#)
  - default/package (~)

Class Diagram - Association

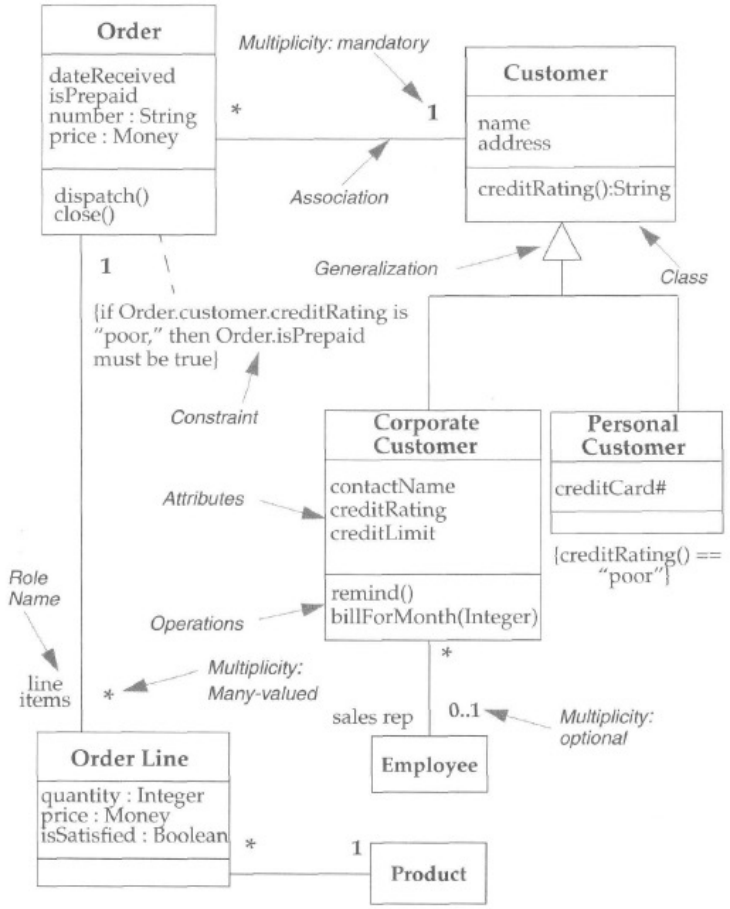
- In a class diagram, an association is represented as a link that might be augmented with information such as:
  - Name
  - Direction
  - Multiplicity
  - Role names
- Association types:
  - **Aggregation**: corresponds to an “is part of” relationship (represented by a white diamond)
  - **Composition**: a strong form of aggregation (represented by a black diamond)
  - **Dependency**: used to indicate that changing one element might lead to changing another one (represented by a dashed line/arrow)

Fowler, M. (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language* , Addison-Wesley , Boston, MA .

Class Diagram - Generalization

- A generalization is represented using an open arrow pointing towards the parent class/interface.
  - For interfaces, *<<interface>>* is written above the interface name and a dashed line is used for the arrow
- Abstract classes, interfaces, and abstract methods are written in italic font.

Class Diagram (Example)



Use Case

- A use case represents a functional requirement of a system (what the system should do) by modeling interactions between the users and the system
- It consists of a set of scenarios tied together by a common user goal
  - A scenario is a sequence of steps describing an interaction between a user and a system
  - Usually, a use case has a common all-goes-well case, and many alternatives that may include things going wrong and also alternative ways that things go well.
- It does not indicate how the behavior is implemented

Fowler, M. (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language* , Addison-Wesley , Boston, MA .

Use Case (Example)

Buy a Product

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

Alternative: *Authorization Failure*
At step 6, system fails to authorize credit purchase
Allow customer to re-enter credit card information and re-try

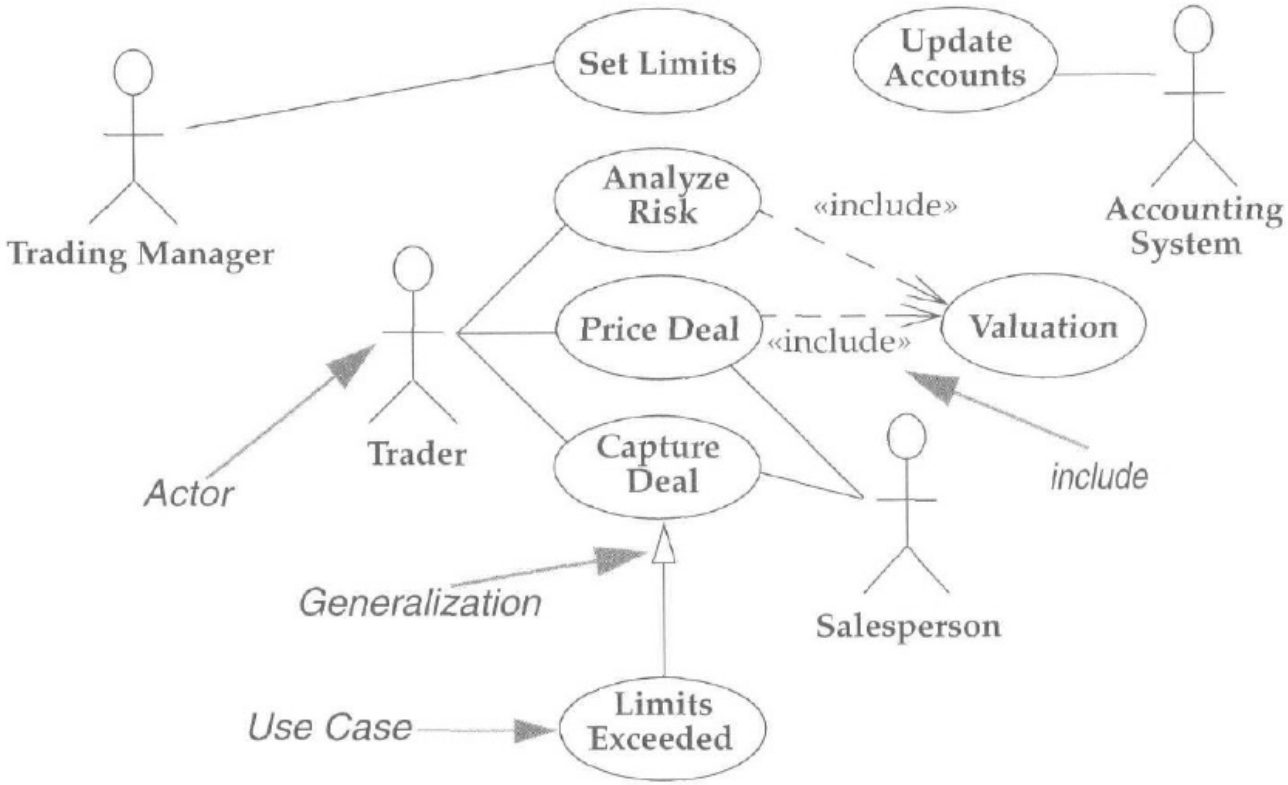
Alternative: *Regular Customer*
3a. System displays current shipping information, pricing information, and last four digits of credit card information
3b. Customer may accept or override these defaults
Return to primary scenario at step 6

Use Case Diagram

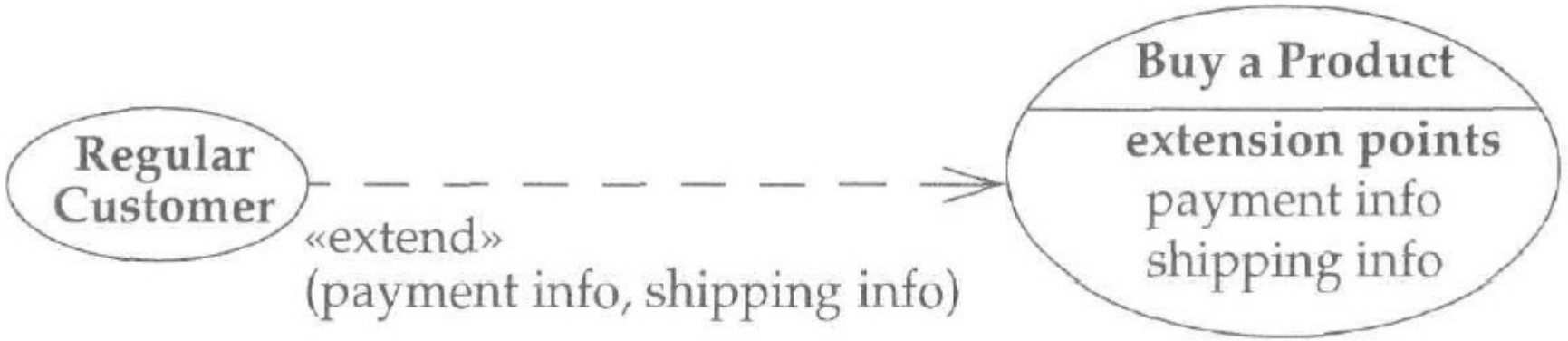
- A UML diagram that mainly consists of actors and use cases
  - Actors do not have to be human
  - A single actor may perform several use cases and a use case may have several actors performing it.
- It also models relationships between use cases:
  - **Include**: Used when some behaviour is similar across more than one use case.
  - **Generalization**: Used when a use case is similar to another one but does a bit more.
  - **Extend**: Similar to generalization but with more rules to it. The extending use case may add behaviour only at specific "extension points" declared in the base use case.

Fowler, M. (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language* , Addison-Wesley , Boston, MA .

Use Case Diagram (Example)



Use Case Diagram (Example)



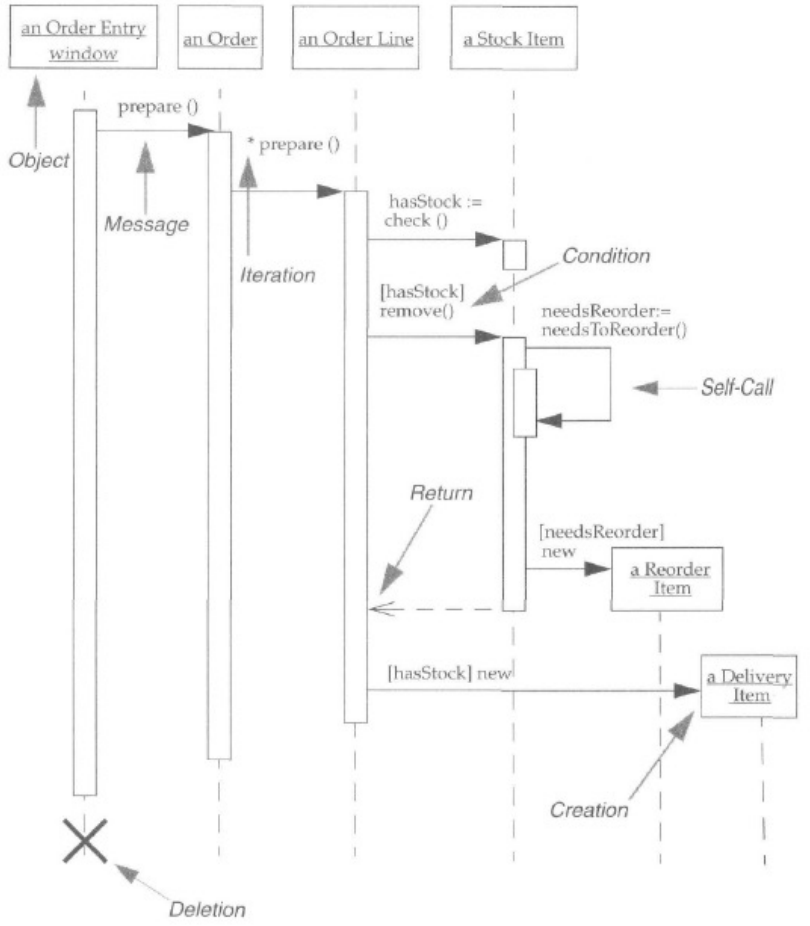
Interaction Diagram

- Typically, an interaction diagram captures the behaviour of a single use case. The diagram shows a number of example objects and the messages that are passed between these objects within the use case.
- Two types of interaction diagrams
  - Sequence diagram
  - Collaboration diagram

Sequence Diagram

- Objects are shown as boxes at the top of dashed vertical lines (called lifelines)
  - An activation box can be used to show when an object is active
- Each message is represented by an arrow between the lifelines of two objects. The order in which these messages occur is shown top to bottom on the page. Control information can be added to messages:
  - **Condition**: indicates that the message is sent only if the condition is true
  - **Iteration marker**: shows that a message is sent many times to multiple receiver objects (e.g. iterating over a list)

Sequence Diagram (Example)



Fowler, M. (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language* , Addison-Wesley , Boston, MA .