# Object-Oriented Methodology

Dr. Sothy Vignarajah

Lecture II

# Software Development

- High quality software development is expensive. Change is from alphanumeric interfaces to event-driven graphical user interfaces (GUIs)
- Introduction of multi-layer client-server architecture, distributed databases, internet and so on.
- Software engineering techniques – modularity, modifiability, abstraction, reliability, information hiding, maintainabilty etc.

# Software Development

- History of software development is from bit patterns in machine languages, to mnemonics in assembly languages to macro instructions, to procedures to abstract data types to objects, frames, design patterns and business objects.

# Making up your own Methodology

- Typical Problems:
- Coding started too early
- Coding started too late
- Procedure ill planned and lacking method to proceed
- Immediate and final results not verified or validated.

# Making up your own Methodology

- Application architecture not planned clearly or its development not within control
- Development driven by an understanding of object-orientation that is too academic and not practical minded.
- Guidelines for analysis, design, and implementation are lacking
- Documentation of results and design decisions insufficient

# Object-Oriented Methods

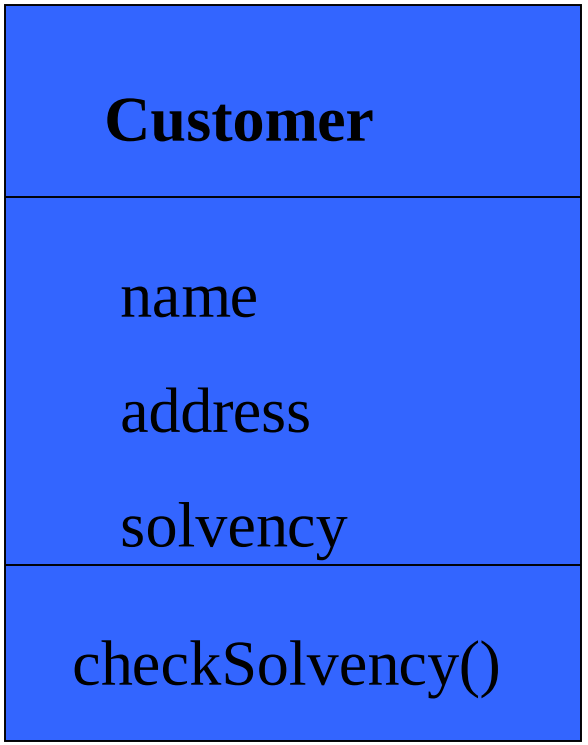
- Holistic thinking – individual facts and phenomena are seen as an integrated whole where links and interdependencies between the components form an essential part of the entirety
- Methodological Uniformity – results of an activity *i* in the object-oriented development process can be taken into activity *i + 1*

# Object-Oriented Methods

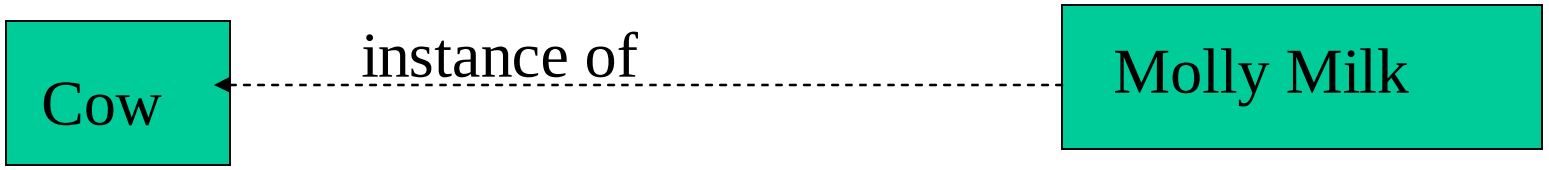
- Class concept is used to work with units of data and operations instead of a separation between the attributes and methods
- Evolutionary Development – A complex system is developed step by step

# Permeable Model

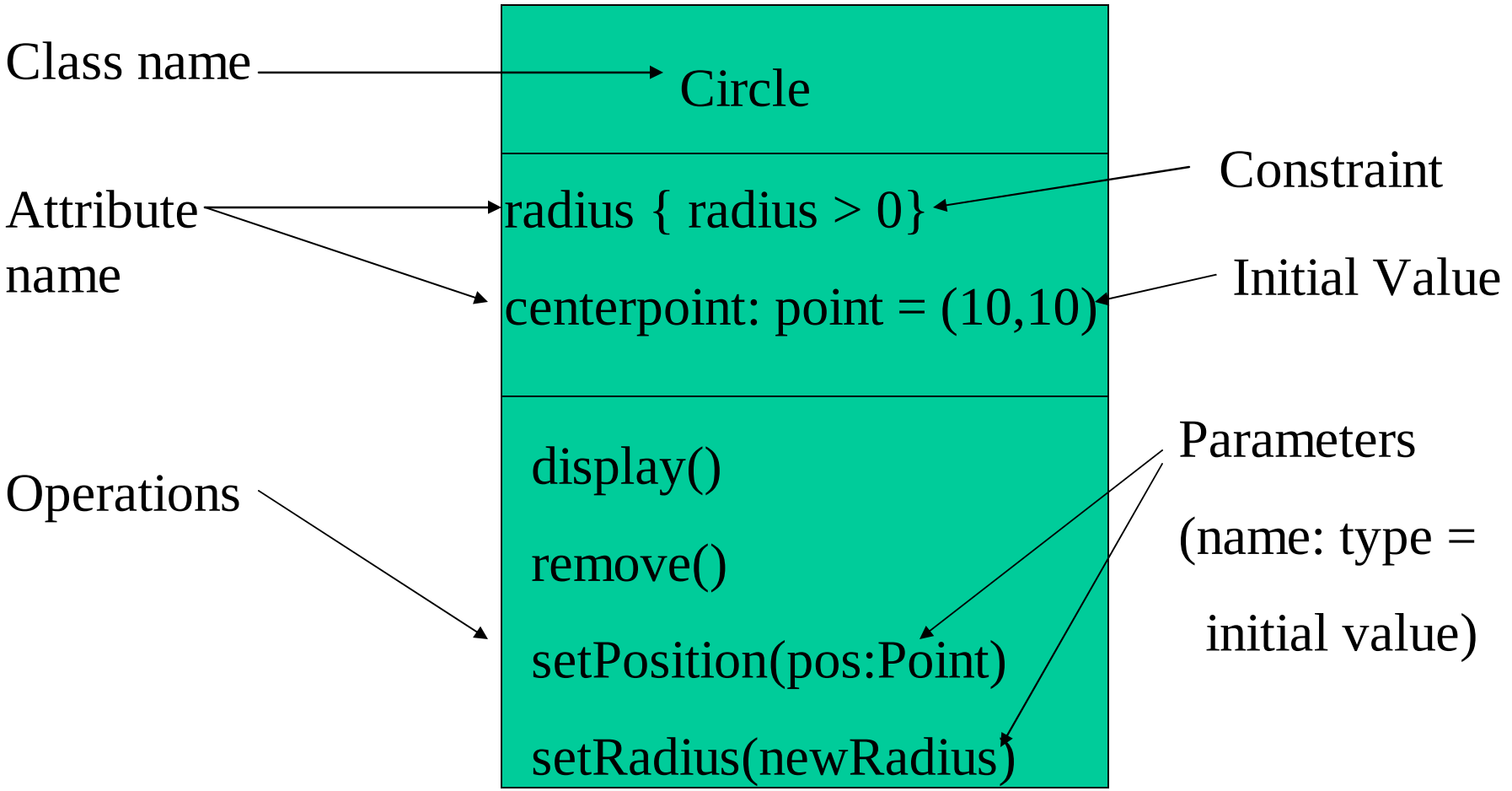
- Coded in Java
- Class Customer
- {
- String name;
- Address address;
- Solvency solvency;
- Public int checkSolvency()
- {
- }
- }

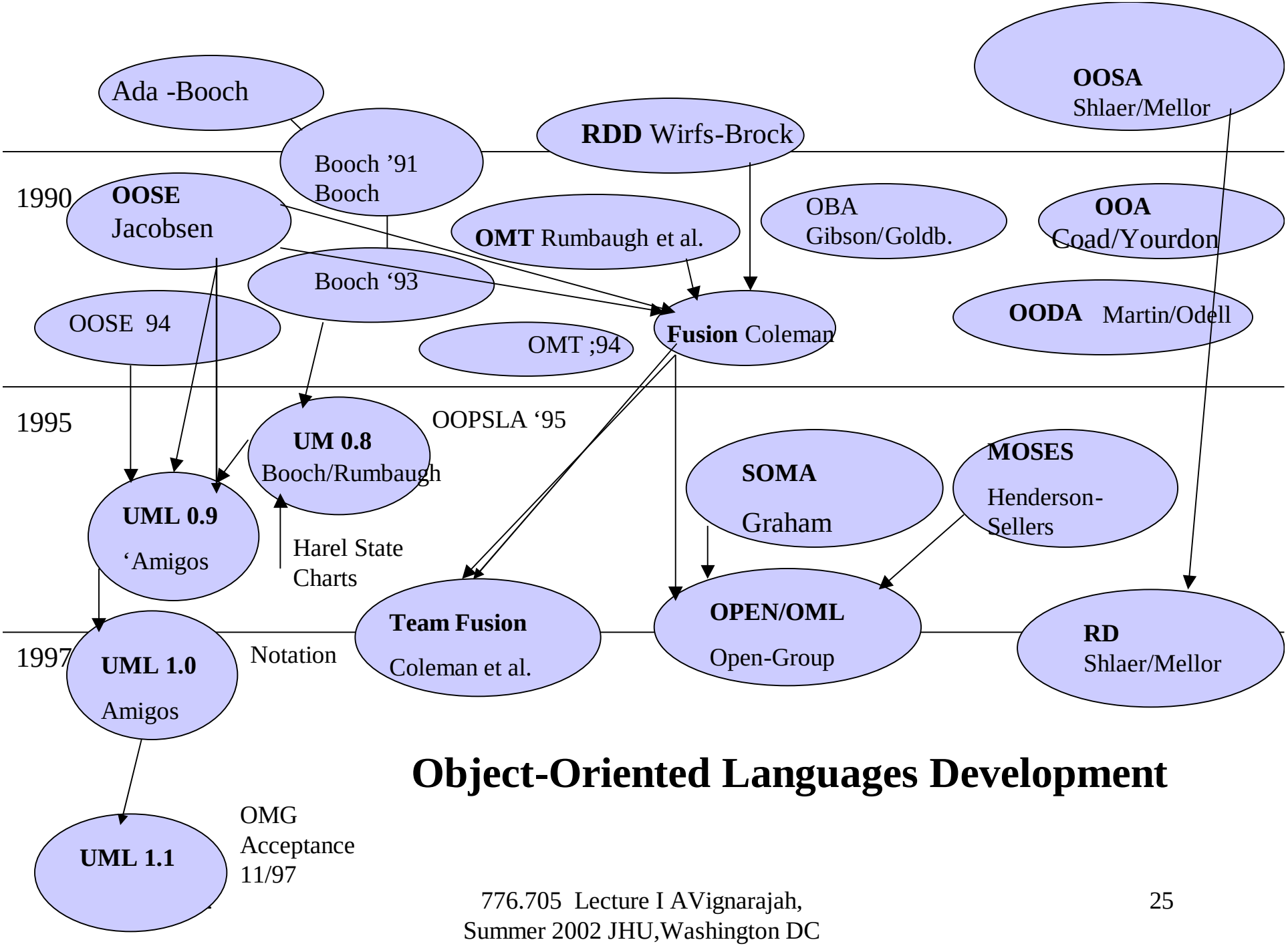


# Class - Object

# Class






# Object-Oriented Languages Development

# History of Object Orientation

- Object-orientation and object-orientation languages is almost 30 years old. Books on object-oriented analysis design started to appear in the early 1990s.As shown in figure:
- The authors included Grady Booch, Coad and Yourdon, Rumbaugh, Wirfs-Brock and Johnson, Shlaer and Mellor, Martin and Odell, Henderson-Sellers, and Firesmith.

# History of Object Orientation

- Others to make a contribution were Goldberg and Rubin, and Jacobson.
- In the early 1990s, works of Grady Booch and James Rumbaugh became very popular. Works of Grady Booch was more commercial and technical while those of James Rumbaugh was more structure oriented.

# History of Object Orientation

- In 1995, Grady Booch and James Rumbaugh began to combine their methods to form a common notation called Unified Method (UM). Ivar Jacobson joined in integrating his so-called *use cases*. They called themselves '*Amigos*'.It had a large market share and in 1997 UML 1.1 was submitted to the OMG for standardization.

# Use Case Diagram

- Use case diagrams show actors(involved *party*, *event*, *external system*, *dialog*, *boundary*, *control*, *entity*) use cases, and their relationships.
- Use cases describe the basic processes in the application area.
- It describes the relationship between a set of use cases and the actors involved in these use cases. They represent both context and structure for the description of how a business event is handled.

# Use Case Diagram

- The business *event* for example could be the written damage report of a home-insured person.
- The business *process* for example could be the damage claim home insurance, which describes the entire process of handling such an event.

# Use Case Diagram

- The business process (Damage claim home insurance) also includes activities that are not directly supported by the software or the application to be developed, like visit to the premises by a loss adjuster.
- Use cases usually describe only those activities that are described by the software under development, and their contact points with the environment of this software.

# Use Case Diagram

- All of the use cases form the model which describes the requirements to be met by the external behavior of the entire system.
- Use cases *do not* represent a design approach, and they do not describe the internal behavior of the system.
- Use cases are a tool for *requirement* determination.

# Use Cases

- Use cases are not process diagrams, data flow charts, nor functional models.
- Use cases support communication with future users, the customer, the technical department, and so on.
- Use cases describe external system behavior, that is what the system is supposed to do.

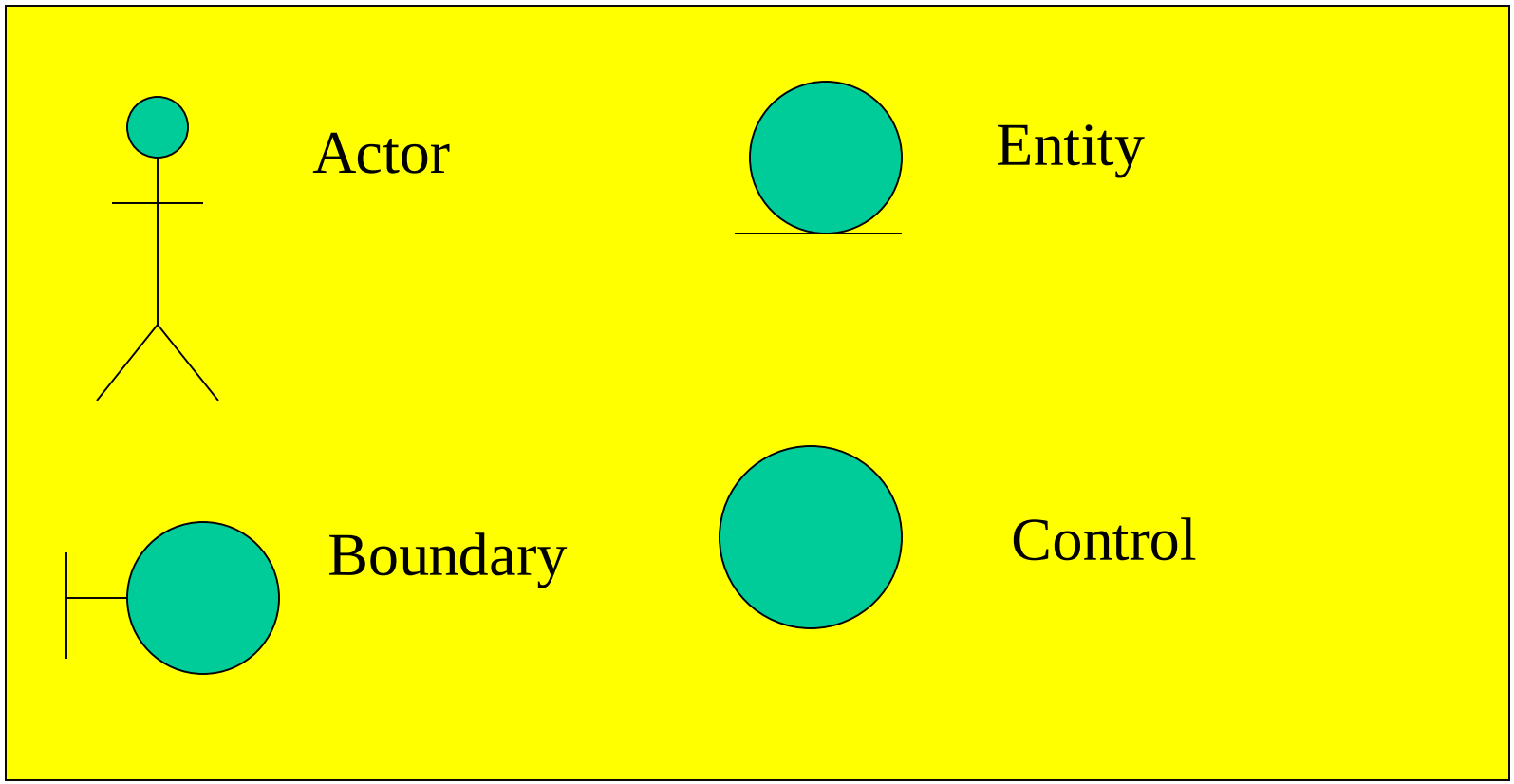
# Use Cases

- Use Cases do not describe how it comes into being, ie. which system design and which implementation contribute to this external system behavior are questions which the use cases do not provide an answer for.

# Use Case Diagram

- A use case diagram includes a set of use cases represented by individual ellipses, and a set of actors and events involved. The use cases are joined by straight lines with the classes involved. A frame around the use cases symbolizes the system boundaries.

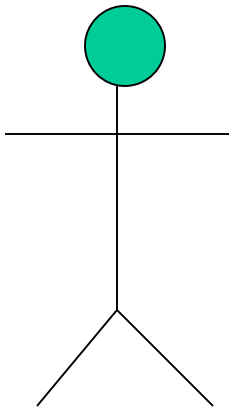
# Notation



# Notation

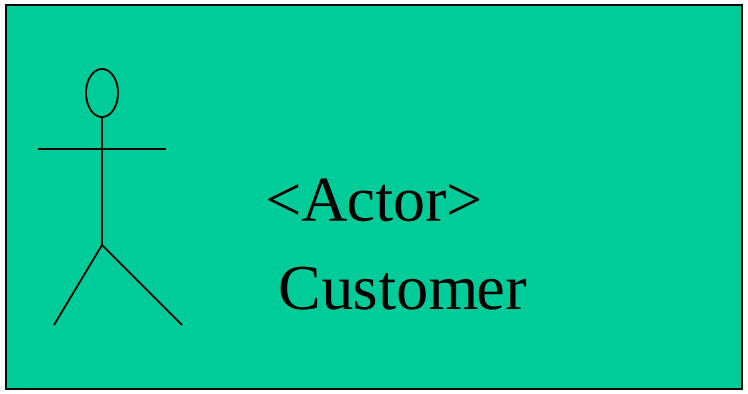
Customer

<Actor> Textual stereotyping

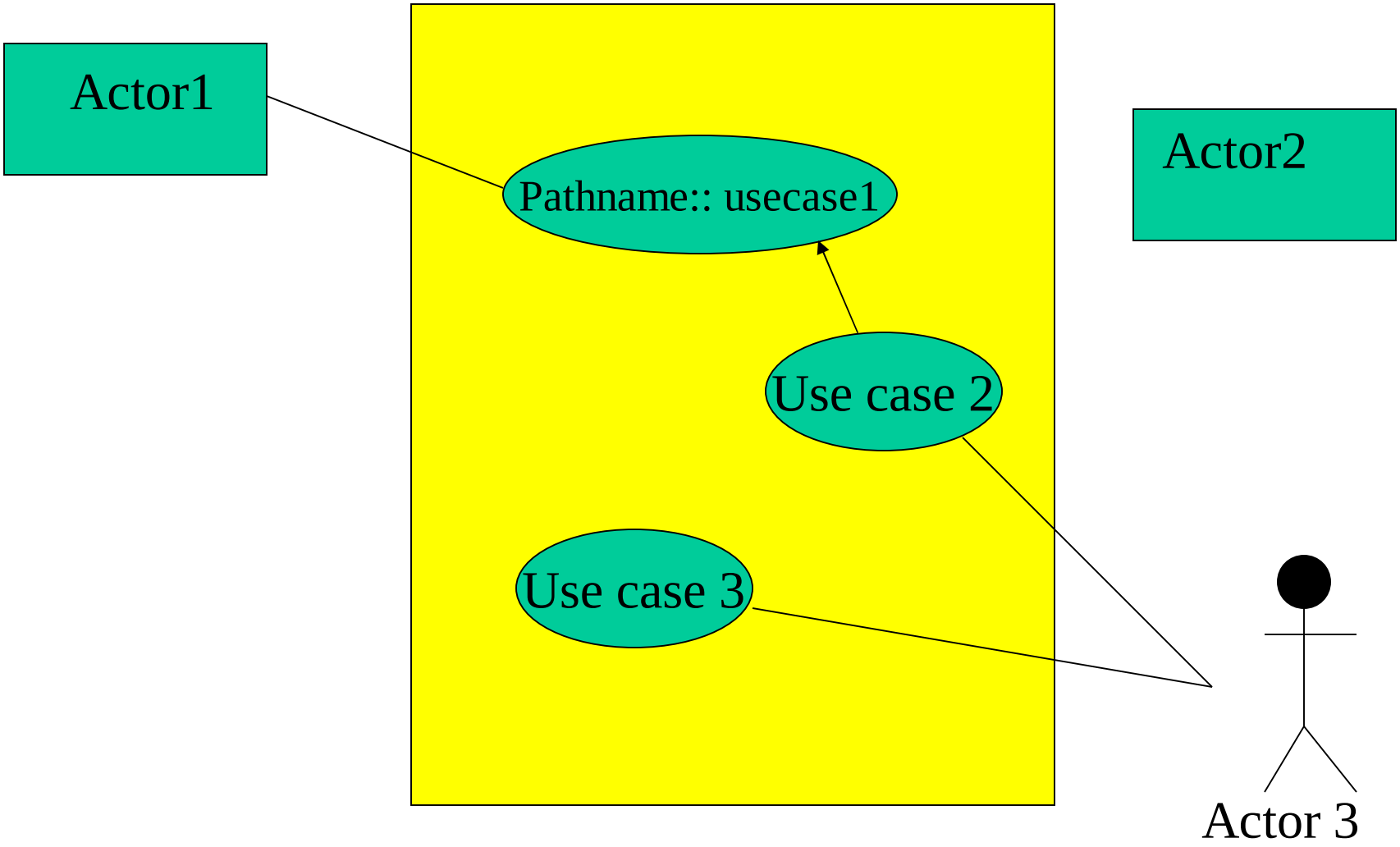


<Customer> Visual stereotyping

# Notation



# Use Case Diagram



# Notations

- *include* substitutes the ‘uses’ relationship of UML 1.1 and is used to denote that another use case occurs inside a use case. The construct is suitable to extract identical sections occurring in several use cases in order to prevent redundancy.

# Notations

- *extend* – used to show that in certain circumstances or at a specific point (the so called extension point) a use case is extended with another use case.
- Generalization (UML 1.3) permits sub use cases to inherit behavior and semantics from super use cases, in analogy with the generalization relationship between classes.

# Class diagrams

- It shows a set of static model elements, in particular classes, and their relations.It is a relationship between actors and use cases.
- It represents both context and structure for the description of how a business event is handled.