

## **Modeling with UML**

- A language or notation intended for analyzing, describing and documenting all aspects of the object-oriented software system.
- UML uses graphical notations to express the design of software projects.
- UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

**(1) Use Case Diagram**

**(2) Class Diagram**

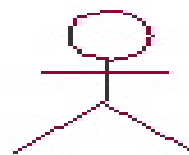
**(3) Interaction Diagram**

**(4) State Diagram**

## Use Case Diagrams

Used during the analysis phase of a project to identify and partition system functionality. They separate the system into *actors* and *use cases*.

- **Actors** represent roles that can be played by users of the system. Those users can be humans, other computers, or even other software systems. The only criterion is that they must be external to the part of the system being partitioned into use cases. They must access to that part of the system, and they must receive outputs from it.
- **Use cases** describe the behavior of the system. This behavior is described textually. It describes the inputs from actor and outputs to other actors, and the behaviors that convert the inputs to the outputs.
- The use case diagram shows which actors interact with each use case.



Actor



Use Case

## Use Case Diagrams (Example)

### • Accident Management System



### Accident Management System

- The fieldOfficer invokes the use case ReportEmergency to notify the Dispatcher of a new emergency.
- As a reponse, the dispatcher invokes the OpenIncident use case to create an incident report and initiate the incident handling.
- The dispatcher enters preliminary information from the fieldOfficer and orders additional unit with the AllocateResources.

## Template to describe a use case

- Name of the use case
- Participating actors (actors interacting with the use case)
- Flow of events: sequence of interactions of the use case.
- Entry conditions: conditions that need to be satisfied before the use case is initiated.
- Exit conditions: conditions that need to be satisfied after the use case is initiated.

### ReportEmergency

### Use case description

---

*Use case name*

ReportEmergency

---

*Participating actors*

FieldOfficer and Dispatcher

---

*Flow of Events*

1. The fieldOfficer activates the “report Emergency” function of her computer
2. The computer response a form
3. The fieldOfficer fill out the form by selecting the emergency level, type, location and submit the form.
4. Computer receives the form and notify the dispatcher.
5. The dispatcher acknowledges the report.
6. Computer displays the acknowledgment to the FieldOfficer.

*Entry condition*

The FieldOfficer is logged into her computer

*Exit Condition*

The FieldOfficer’s report has received an acknowledgment

## How to Draw Use Cases Diagrams

List a sequence of steps a user might take in order to complete an action.

### Place an order:

A user place an order with a sales company might follow these steps.

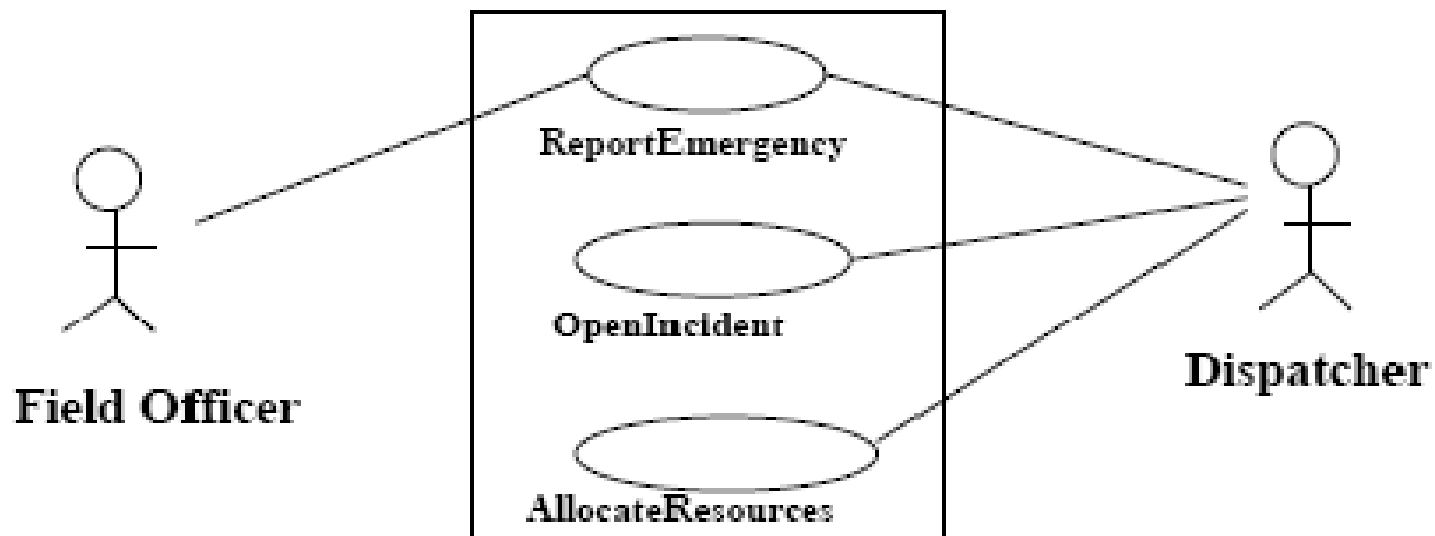
1. Browse catalog and select items.
2. Call sales representative.
3. Give shipping information.
4. Give payment information.
5. Get confirmation number from salesperson.



## Use case diagrams: types of relationships

- **Communication relationship:** Actors and use cases communicate when information is exchanged between them.

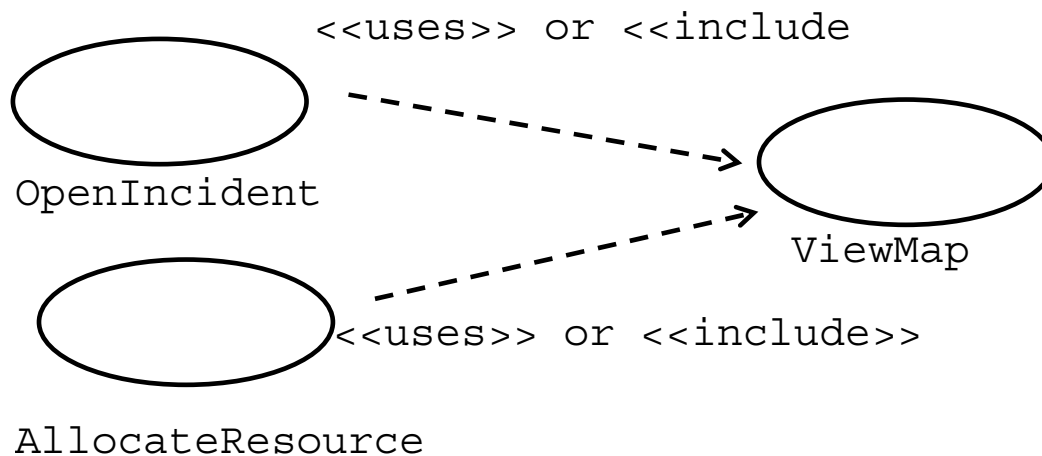
Depicted by solid line between actor and the use case.



## Uses relationship:

When describing a complex system, its use case model can become quite complex and can contain redundancy.

**-- Reduce the complexity of the model by identifying commonalities in different use cases.**



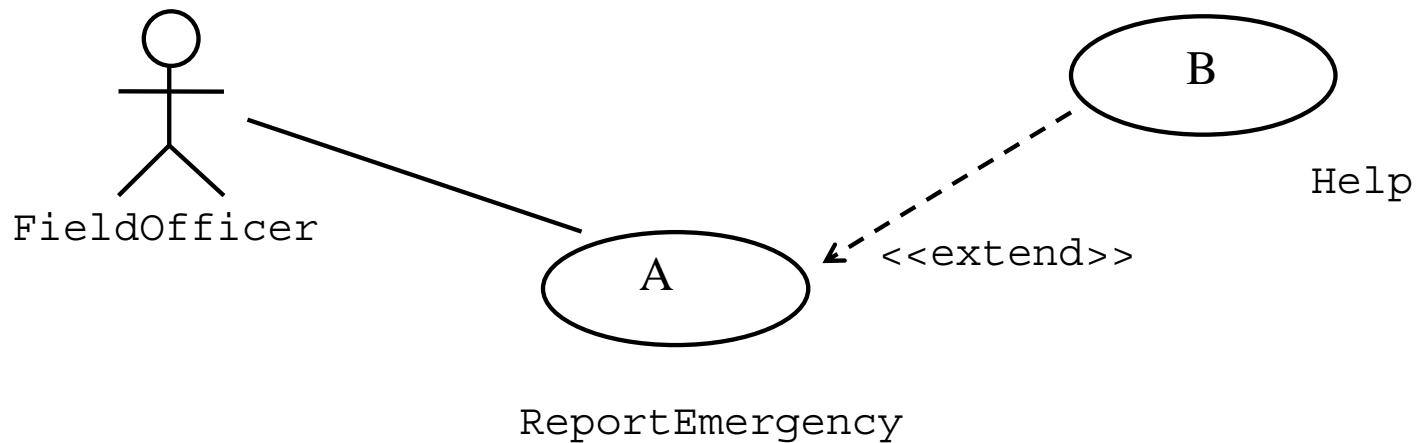
## Extend Relationship:

An alternate means for reducing complexity in the use case model.

– An extend association from Use Case B to Use Case A indicates that **B is an extension of A**.

- Example:

–The use case “ReportEmergency” is completed by itself, but can be extended by the use case “Help” for a specific scenario in which the user requires help





## When to Use: Use Cases Diagrams

- Use cases are powerful tools for analysts to use when partitioning the functionality of a system.
- Help analysts to structure use cases such that their textual descriptions contain a minimum of redundant information.
- Not design tools. They do not specify the structure of the eventual software, nor do they imply the existence of any classes or objects. They are purely functional descriptions written in a formalism that is completely separate from software design.

## Class Diagrams

- Used to refine the use case diagram and define a detailed design of the system.
- Classifies the actors defined in the use case diagram into a set of interrelated classes.

The relationship or association between the classes can be

- Associations
- Dependency
- Aggregation
- Inheritance

## Association, Aggregation, Composition

**Association:** all object have their own lifecycle and there is no owner.

Example: Teacher and Student.

Multiple students can associate with single teacher and single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

**Aggregation:** a specialize form of Association where all object have their own lifecycle but there is ownership.

Example: Department and teacher.

A single teacher cannot belongs to multiple departments, but if we delete the department teacher object will not destroy. We can think about “has-a” relationship.

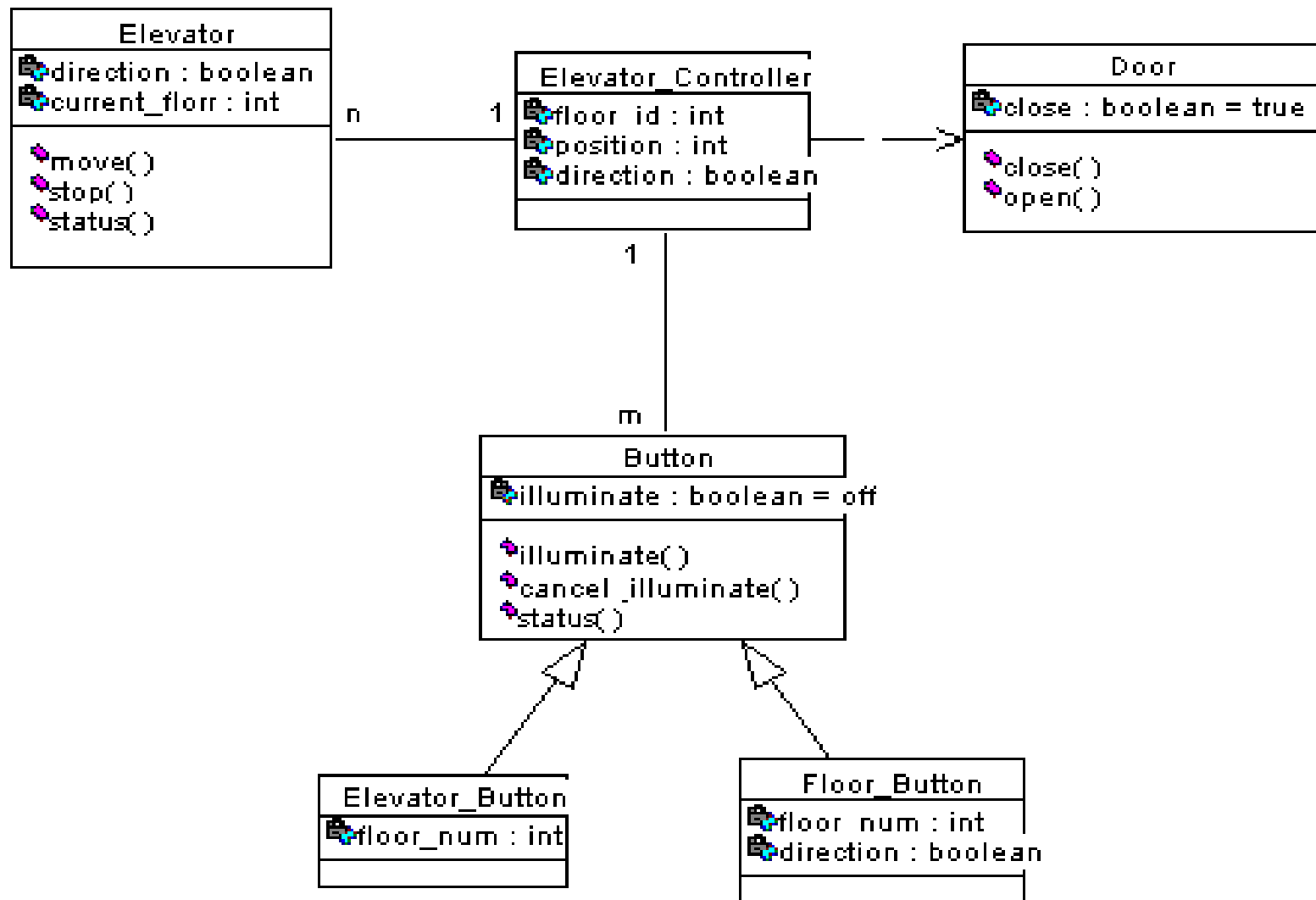
**Composition :** a specialize form of Aggregation. It is a strong type of Aggregation. Child object dose not have their lifecycle and if parent object deletes all child object will also be deleted.

Example: House and rooms.

## Associations: Relationships between instances of classes

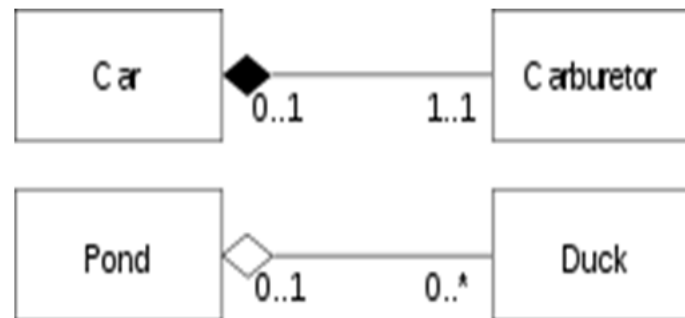
- The UML representation of an association is a line with an optional arrowhead indicating the *role* of the object(s) in the relationship, and an optional notation at each end indicating the *multiplicity* of instances of that entity.

<b>0..1</b>	No instances, or one instance
<b>1</b>	Exactly one instance
<b>0..* or *</b>	Zero or more instances
<b>1..*</b>	One or more instances



**Aggregation:** depicted as an unfilled diamond and a solid line.

**Composition:** depicted as a filled diamond and a solid line. It always implies a multiplicity of 1 or 0..1, as no more than one object at a time can have lifetime responsibility for another object.



## Interaction Diagrams

Model the behavior of use cases by describing the way groups of objects interact to complete the task.

- **sequence**
- **collaboration**

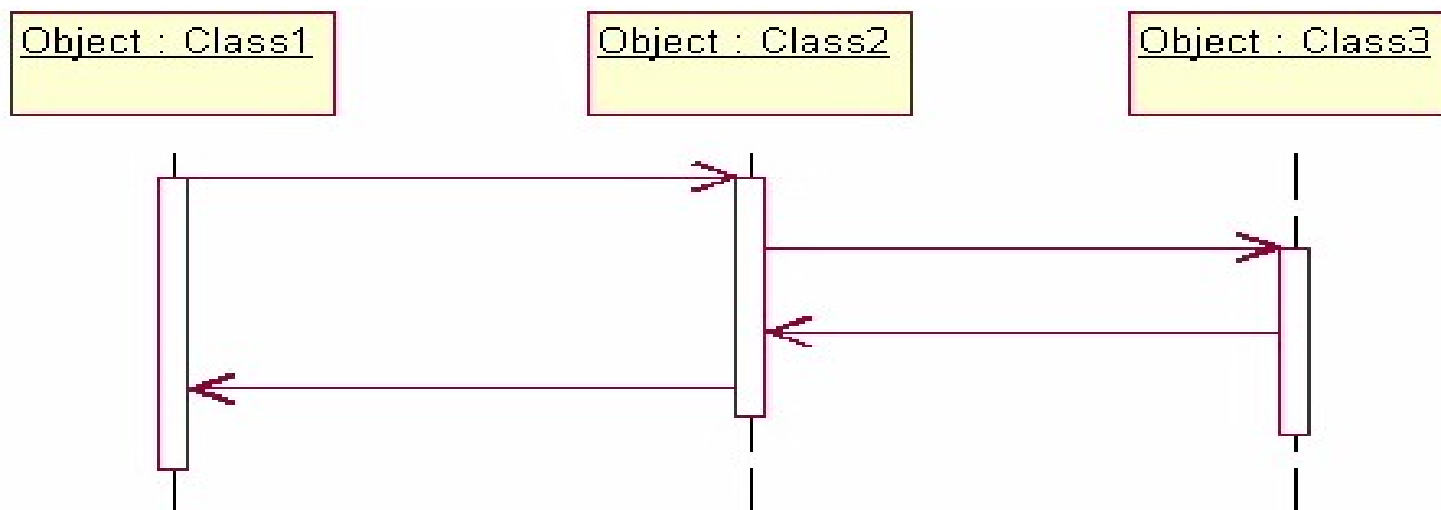
A sequence diagram represents the interaction between different objects in the system. Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass.

## What Is a Message?

Software objects interact and communicate with each other by sending *messages* to each other.

## Sequence diagrams:

The diagrams are read left to right and descending.

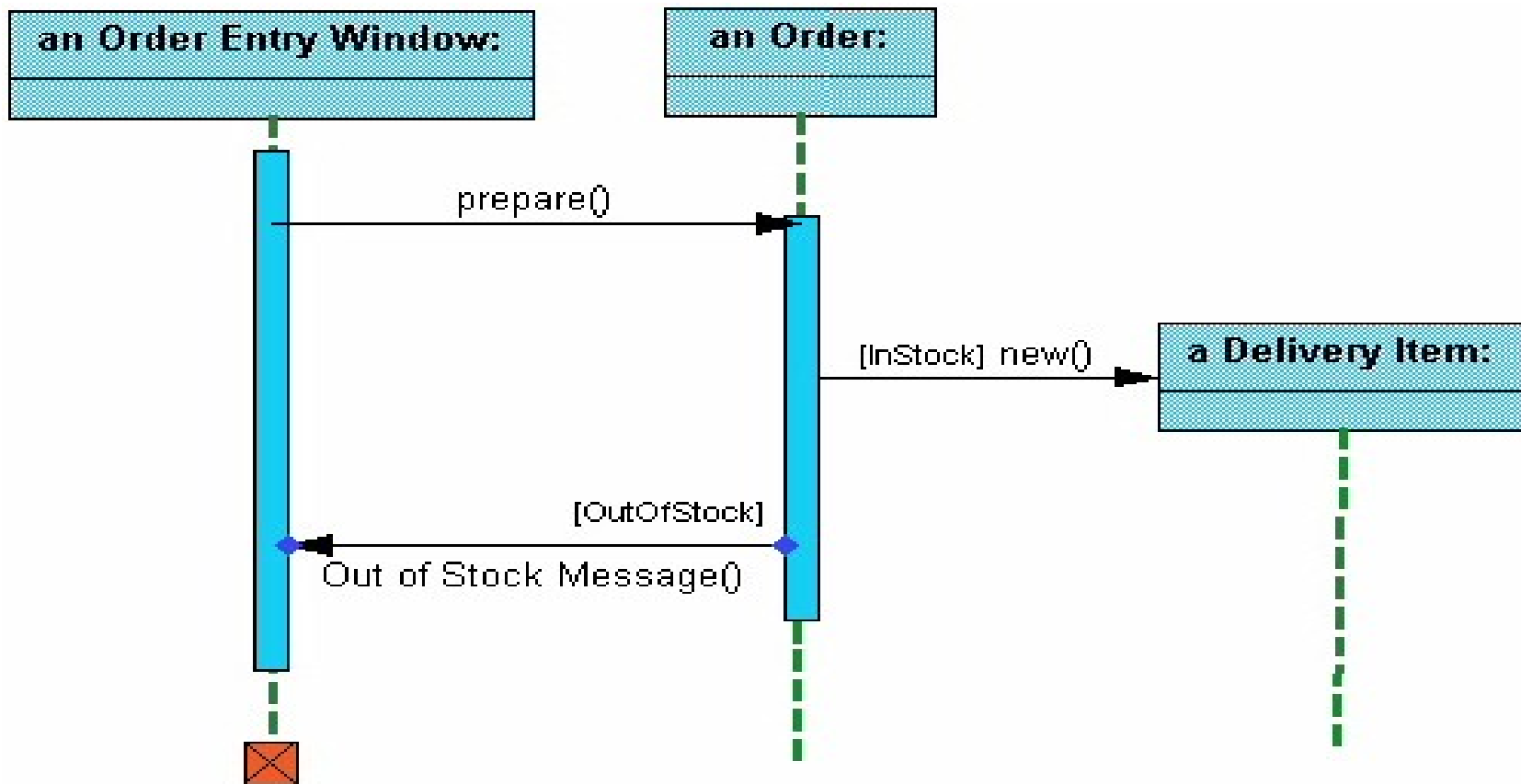


The naming system has one of the following three formats:

- objectName : ClassName (full description)
- objectName: (class not specified)
- : ClassName (object not specified)

## A sequence diagram for placing an order.

If the item is in stock, create a new Delivery Item object. If the item is [OutOfStock], sends a message back to the Order Entry Window object stating that the object is out of stock.



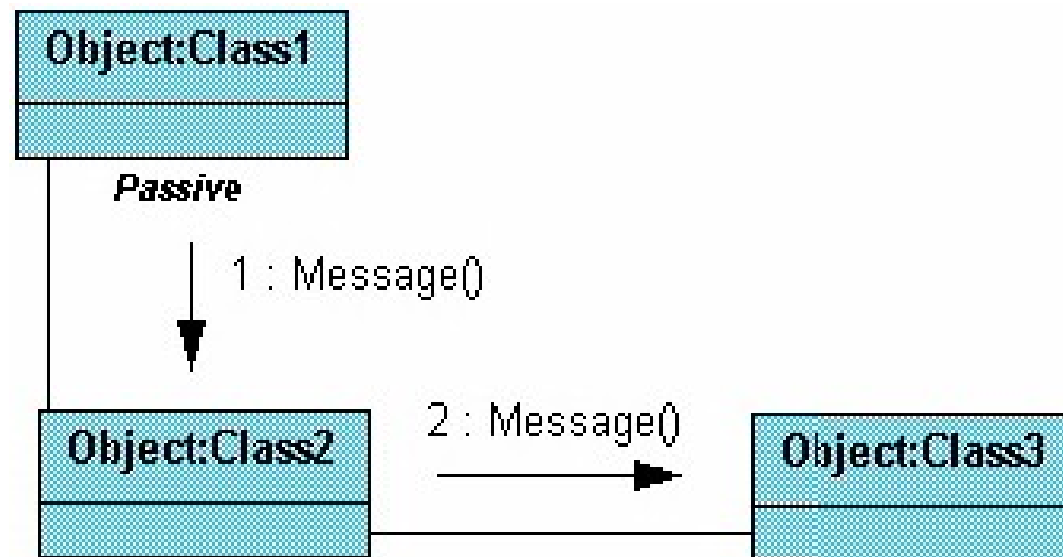


## Collaboration diagrams:

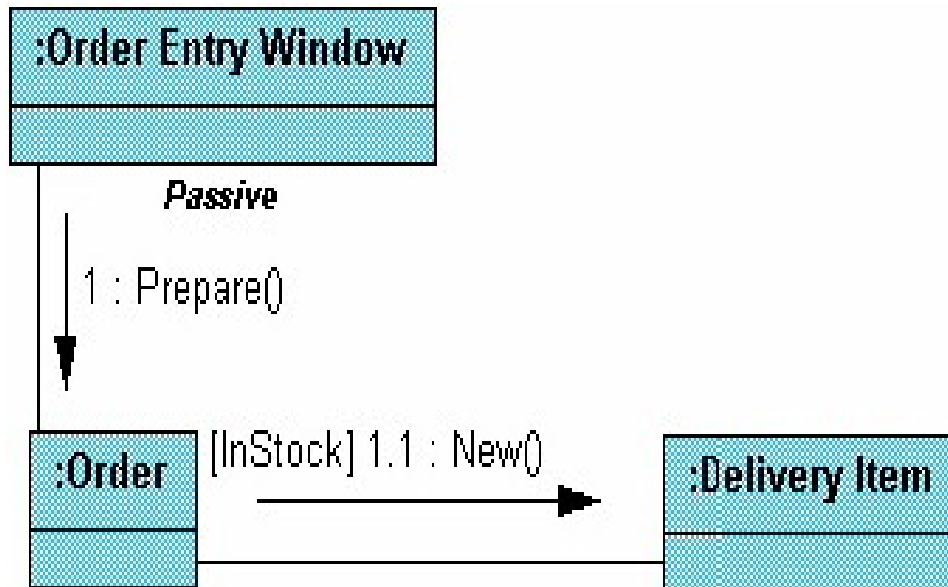
- Show the relationship between objects and the order of messages passed between them.
- Helps to identify all the possible interactions that each object has with other objects.

The objects are listed as icons and arrows indicate the messages being passed between them.

The numbers next to the messages are called sequence numbers.



## Collaboration diagram for the placing an order use case



### When to Use: Interaction Diagrams

When you want to model the behavior of several objects in a use case.

They demonstrate how the objects collaborate for the behavior.

## State Diagrams

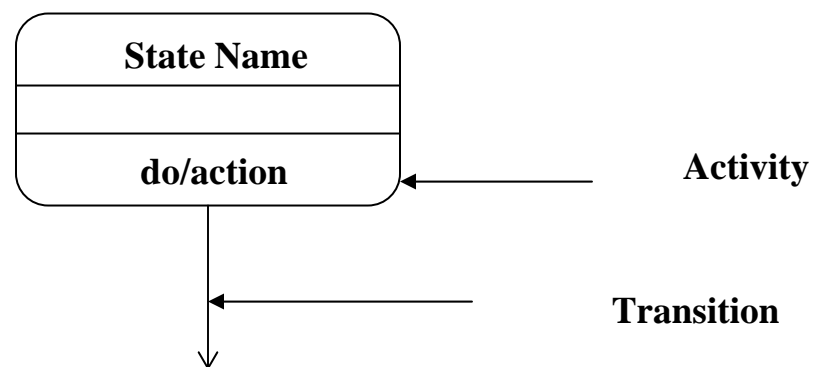
Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

- Describe all of the possible states of an object as events occur.
- Captures the transition of the object's state from an initial state to a final state in response to events affecting the system.

### How to Draw: State Diagrams

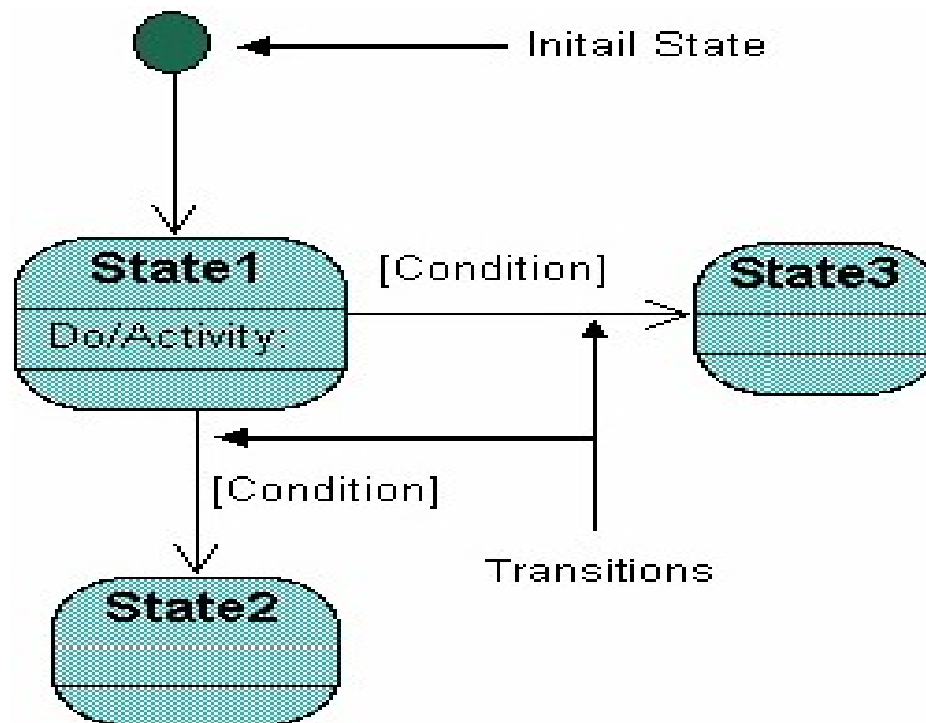
Rounded boxes representing the state of the object and arrows indicating the transition to the next state.

The activity section of the state symbol depicts what activities the object will be doing while it is in that state.



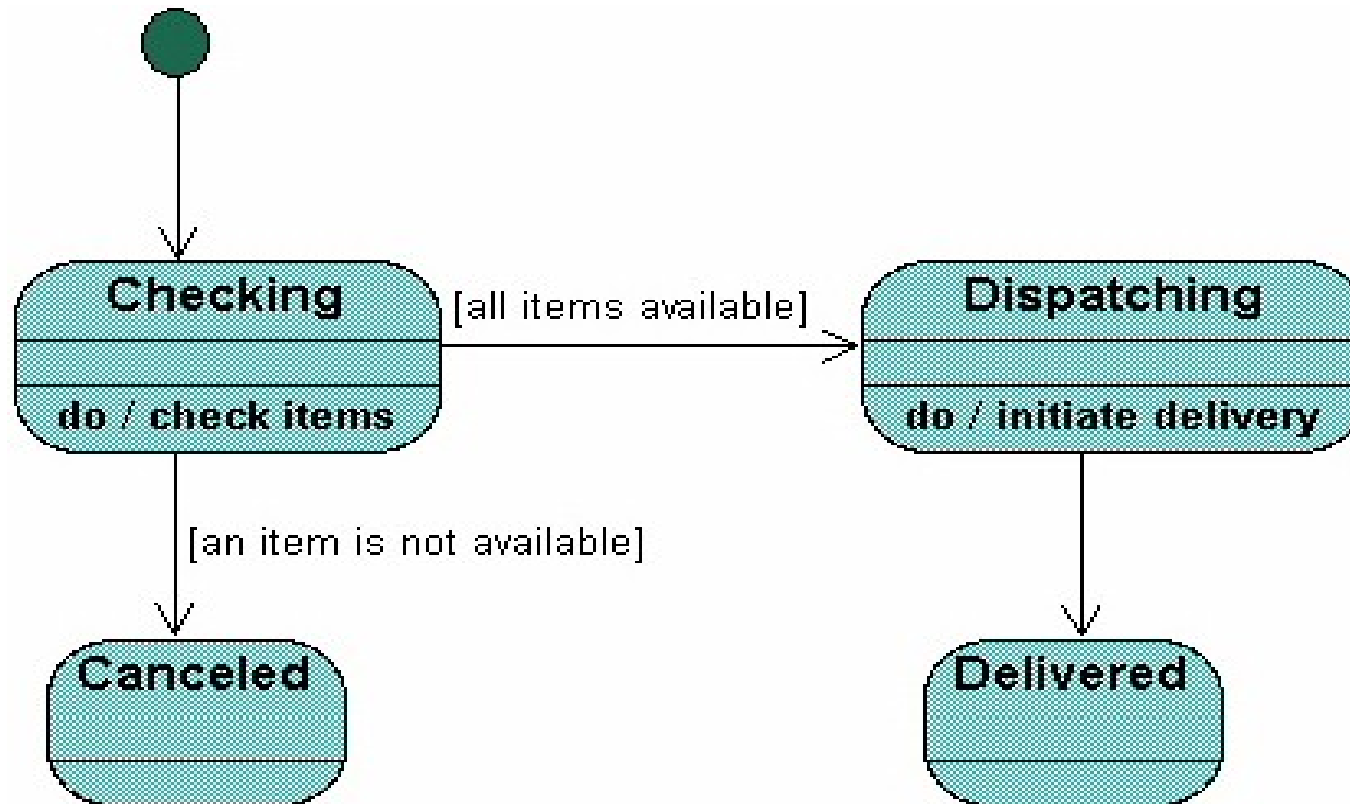
All state diagrams begin with an initial state of the object.

After the initial state the object begins changing states. Conditions based on the activities can determine what the next state the object transitions to.



### A State Diagram for an Order object.

When the object enters the Checking state it performs the activity "check items." After the activity is completed the object transitions to the next state based on the conditions [all items available] or [an item is not available].

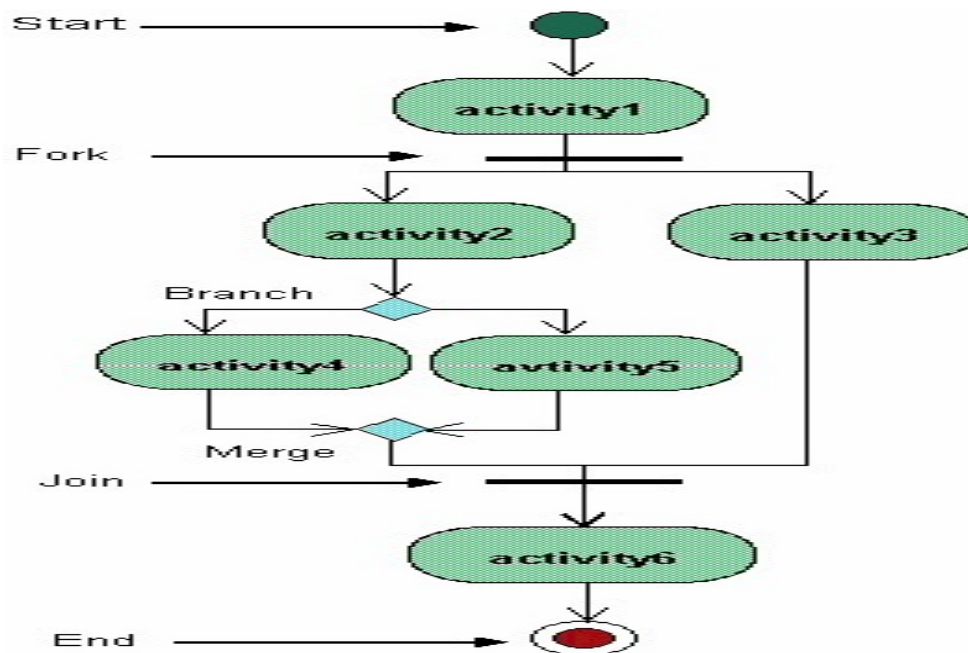


## Activity Diagram

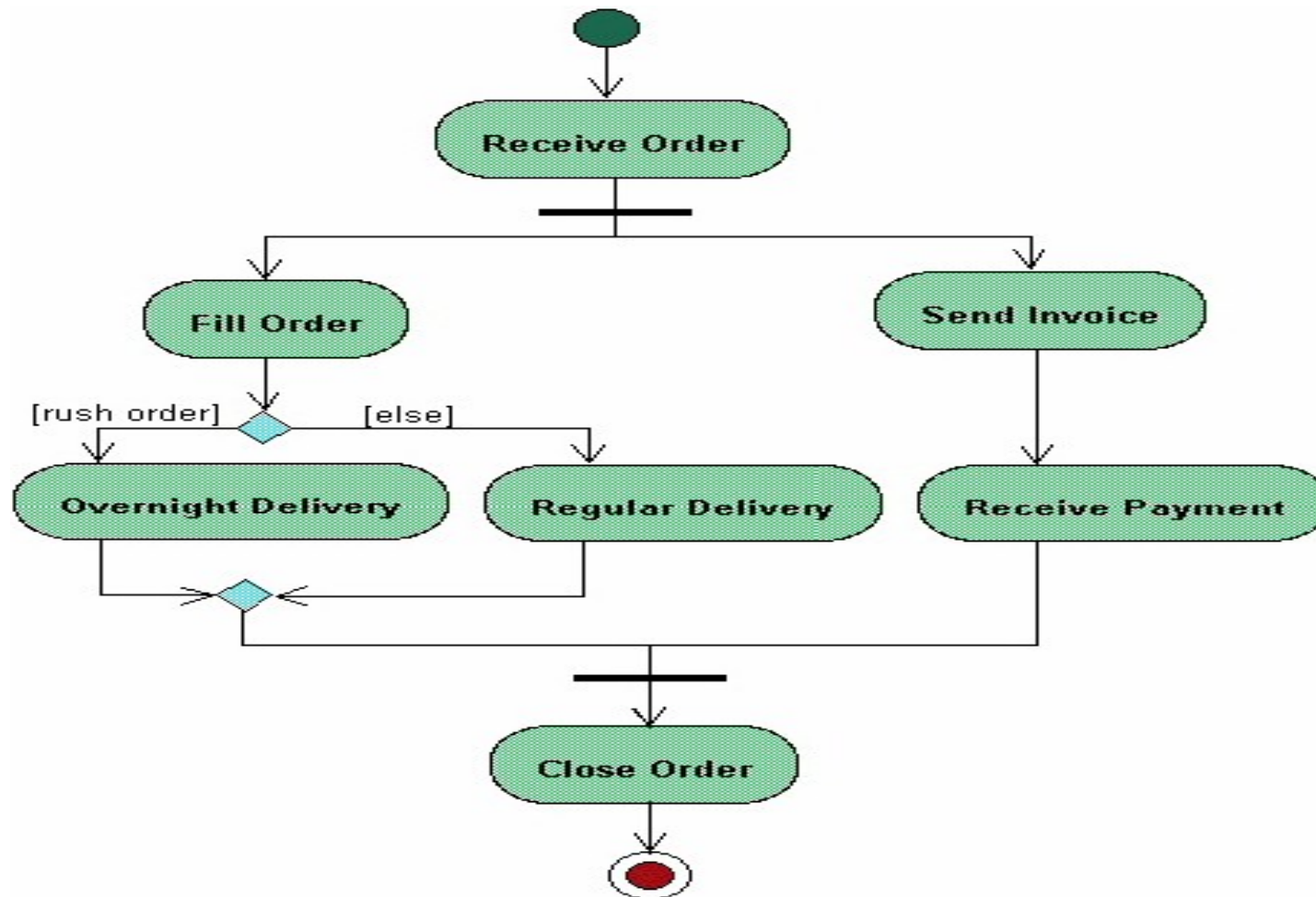
Describe the workflow behavior of a system. Describe the state of activities by showing the sequence of activities performed.

### How to Draw: Activity Diagrams

- Have branches and forks to describe conditions and parallel activities.
- A fork is used when activities are occurring at the same time.
- The branch describes what activities based on a set of conditions.



## Activity diagram for processing an order



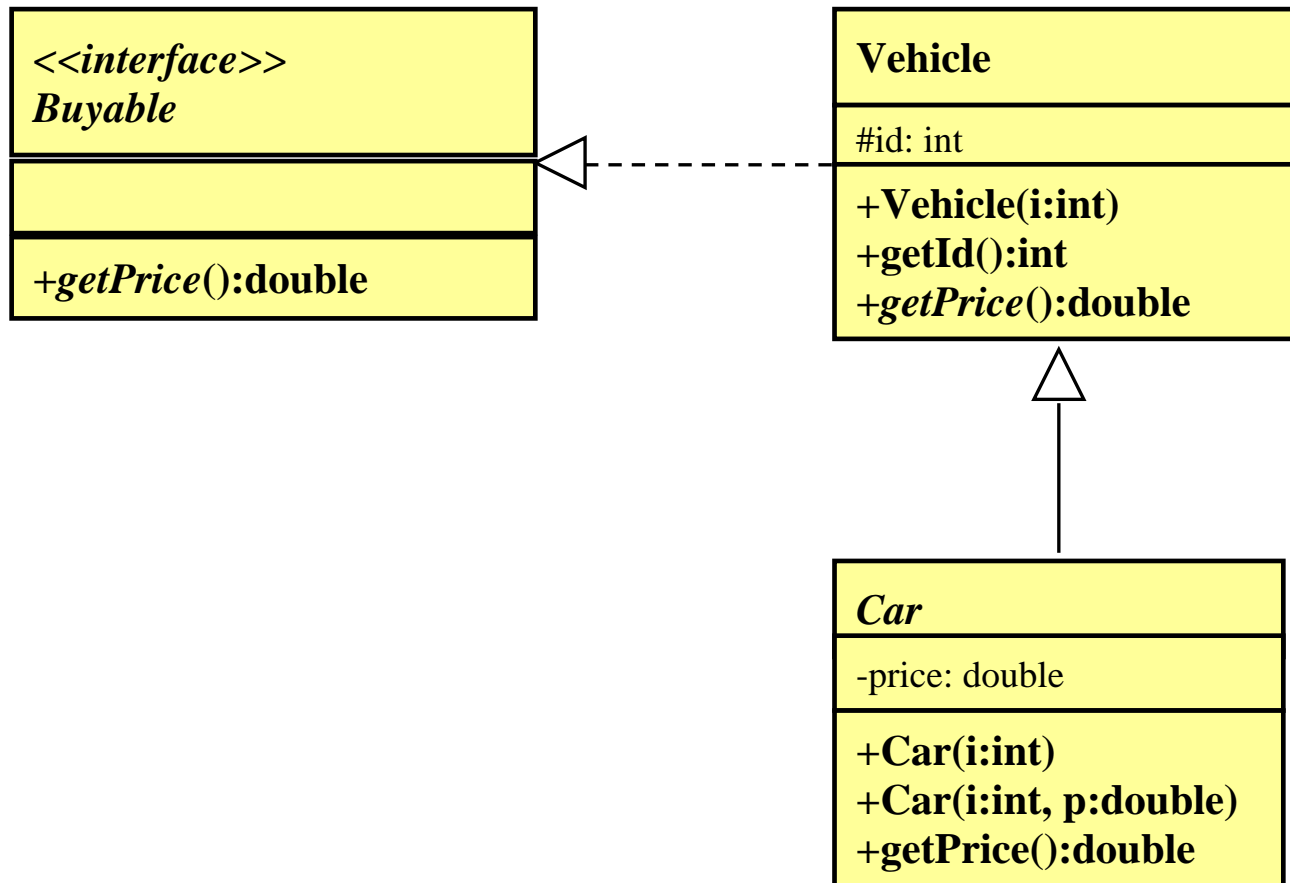
Draw the UML class diagram showing the following data types and the relationships between those types. In the diagram, include the variable and method declarations for each data type and also indicate their visibility (public, private or protected) in UML notation. Please note that the bodies of the methods have been replaced by ellipsis (...) to eliminate details that are not necessary in your diagram.

```
public interface Buyable{
    public double getPrice();
}
public abstract class Vehicle implements Buyable {
    protected int id;

    public Vehicle(int i) {...}
    public int getId() {...}
    public abstract double getPrice();
}

public class Car extends Vehicle {
    private double price;
    public Car(int i) {...}
    public Car(int i, double p) {...}
    public double getPrice() {...}
}
```

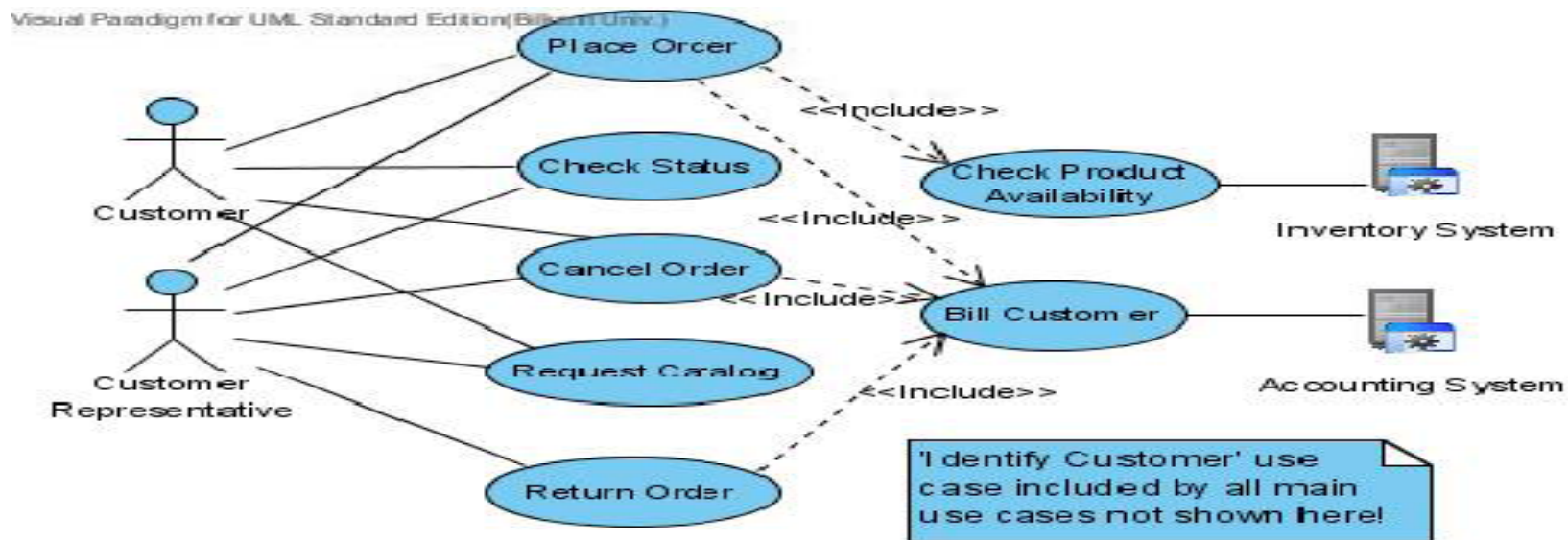




A mail-order company wants to automate its order processing. The order processing system should be accessible to customers via the Web. Customers can also call the company by phone and interact with the system via a customer representative.

The system allows customers to place orders, check the status of their orders, cancel an existing order and request a catalog. Customers may also return a product but this is only possible through the phone, not available on the Web. When placing an order, the customer can check the availability of the product from the inventory system. When the customer places the order, credit card number, billing address and a specification of the cost of the order are used on the invoice. When the customer cancel order or return order, he/she will be billed.

**Draw a UML use case diagram for the order processing system described above.**



Design a simulation of a basketball conference. Each conference has 10 teams. Each team has 12 players. Each player has a specific height, speed, and accuracy. Players know which team they belong to. Some players are scholarship players. Scholarship players need to record their current grade-point average. Players may be transferred between teams. Teams play basketball games against other teams in the conference. The result of each game is determined using a function based on the height, strength, speed, and accuracy of the players on each team.

**Name the classes and draw a UML class diagram showing classes (including variables, methods and relationships between classes with appropriate multiplicities.)**

