UML Primer

-Elango Sundaram

About UML

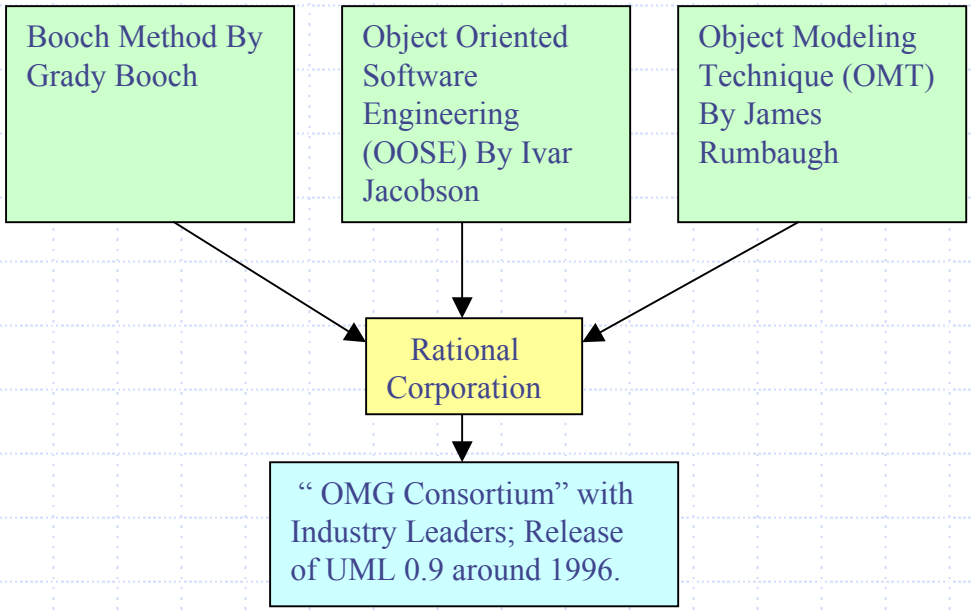
- UML Can be thought of as a blue print for Software
- Graphical notation for expressing underlying OOA&D ideas
- Can be used to design any type of application, hardware, operating system and network

- UML is becoming an industry standard

- Helps better communication between Client/ Vendor, Designer/Developer, Developer/Developer
  Can be used across domains

Brief History

- 1970's Structured Programs
- 1980's Abstract Data Types
- 1990's Object Oriented Programs



UML diagrams

- **Types of UML Diagrams**
  - **Structural Diagrams**
    - Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.
  - **Behavior Diagrams**
    - Use Case Diagram, Sequence Diagram, Activity Diagram, Collaboration Diagram, and State chart Diagram.
  - **Model Management Diagrams**
    - Packages, Subsystems, and Models.

Some Key terms

- **Data Type**
  - Domain of Allowed Values together with a set of allowed operations on it
  - Eg.. Integer (int)
- **Abstraction**
  - Representation of an Entity with only attributes/methods of Interest
    - Key concept in computer science and one of the oldest
    - FORTRAN I had built in Abstract Data Type (ADT) by way of floating points circa 1950s
    - Process Abstraction,Data Abstraction
    - A SQL procedure like calcValues() is an example of process abstraction
- **Encapsulation**
  - Grouping of subprograms and data they manipulate.
  - Eg. Classes in C++, Packages & Classes in Java

Some Key terms

- **Class**
  - Grouping of Data with associated Methods
  - A class is a description of a set of objects that share the same attributes, operations, methods,relationships, and semantics.
- **Object**
  - Instance of class
- **Inheritance**
    - Ability to extend exiting features of an entity and make some modifications
    - Such inherited entities become sub-classes
    - Calling a method is called a message
    - Entire collection of methods of an entity is referred as message interface or message protocol
    - Can be Single inheritance or multiple inheritance depending n the number of direct parents a class extends from)
    - Java Permits only Single Inheritance whereas C++ provides for multiple inheritance

Some Key terms

- Information Hiding
  - Preventing access to variable/methods to outsides
  - Example in Java private, protected..
- Overriding
  - A child deciding to change the way a parent functions; by changing the implementation
- Overloading
  - Method Overloading refers to the way a same method NAME can be made to function in different ways.
    - Example
      - add(int,int),
      - add(array of numbers)
- Interface
  - An interface is a named set of operations that characterize the behavior of an element.
- Method
  - A method is the implementation of an operation. It has the procedure/s that affect the results of an operation.

Some Key terms

- Polymorphism
  Characteristics by which the class/Object exhibits dynamic behavior

Eg.

```
public class Child extends Parent{
     public void invoke(){
       System.out.println("CHILD");
     }
}
public class Parent {
     public void invoke(){
       System.out.println("PARENT");
      }
}
///
Parent p = (Parent) new Child();
p.invoke();  .. This will invoke the  child..
///
```

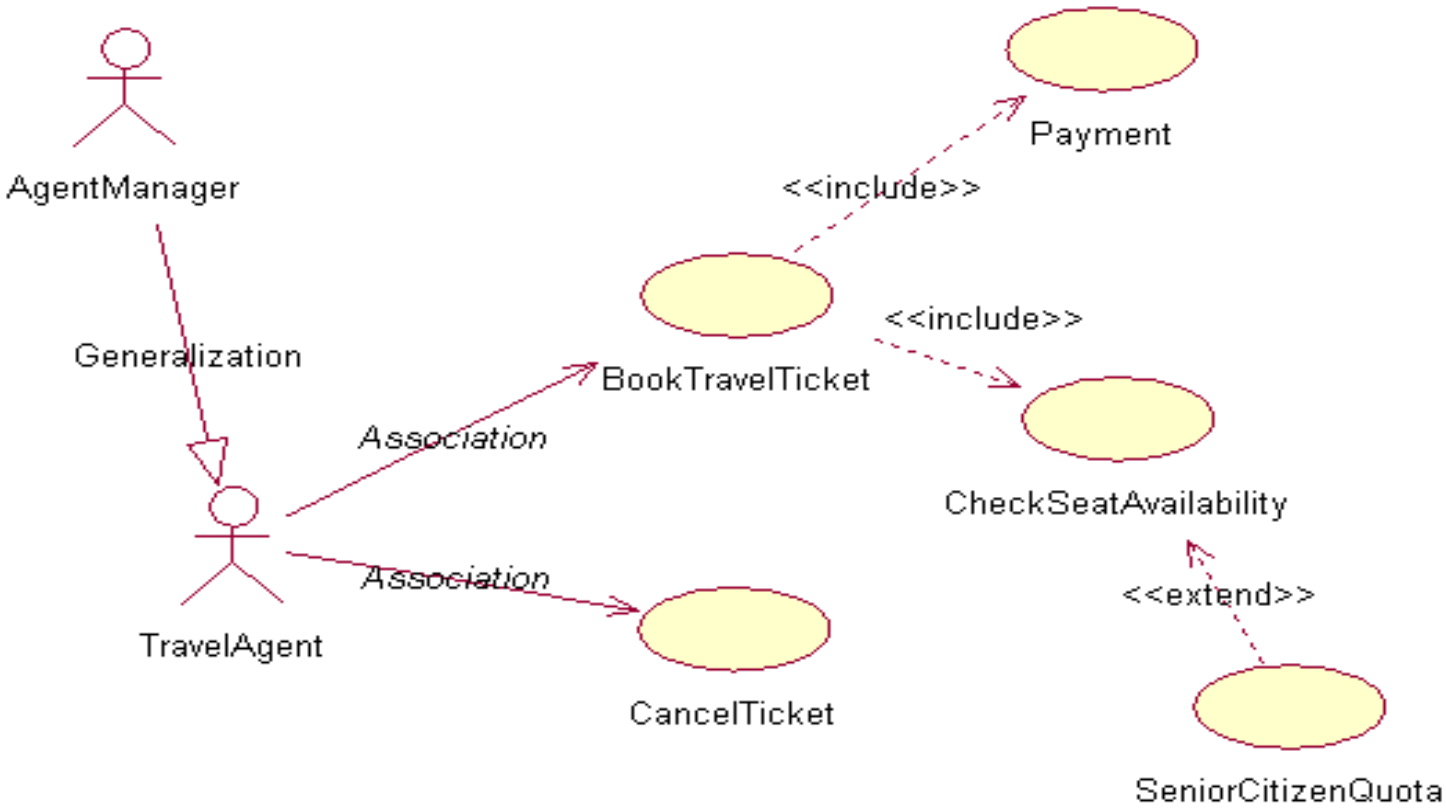
Use Case Model

- A use case represents a coherent unit of functionality provided by a system
- A use case diagram is a graph of actors, a set of use cases and the relationships between these elements.
- A use case is shown as an ellipse containing the name of the use case.
- An extension point is a reference to one location within a use case at which action sequences from other use cases may be inserted.
- The use cases may be enclosed by a rectangle that represents the boundary of the containing system.

Use Case Contd.

- Relationships among use cases are as follows:
  - Association – The participation of an actor in a use case, i.e. instances of the actor and instances of the use case communicate with each other.
  - Include – An include relationship from use case A to use case B indicates that an instance of the use case A will also contain the behavior as specified by B.
  - Extend –An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented (subject to specific conditions specified in the extension) by the behavior specified by A.
  - Generalization – A generalization from use case A to use case B indicates that A is a specialization of B.

Use Case Contd.



Use Case Contd.

Captures business scenarios of the application

Use Case : Name
Actors
Pre Conditions
Descriptions
Post Conditions
Special requirements

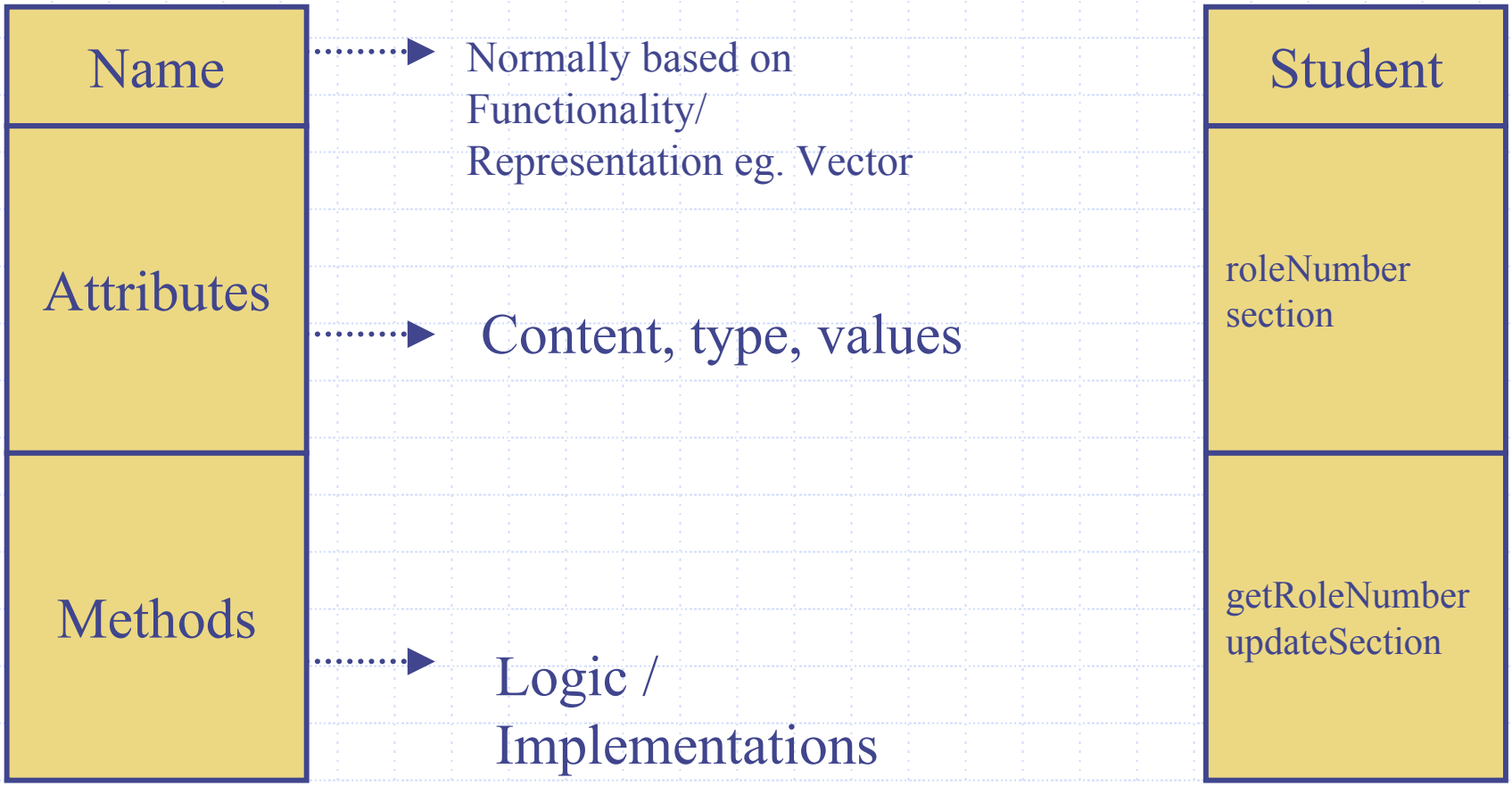
Use Case : BookTravelTicket
Actors: TravelAgent, AgentManager
Pre Conditions: authentication
Descriptions •Check Availability of Ticket •Payment
Post : Display Tickets Booked
Special requirements:

Class Diagrams

Classes Diagrams show

- Classes
- Interrelationships
  - Inheritance
  - Generalization
  - Associations

A Class

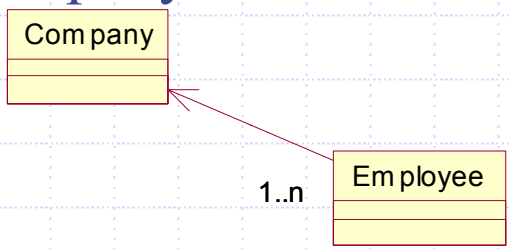

Name
Attributes
Methods
Normally based on Functionality/ Representation eg. Vector
Content, type, values
Logic / Implementations
Student
roleNumber
section
getRoleNumber
updateSection

Class Diagram Contd.

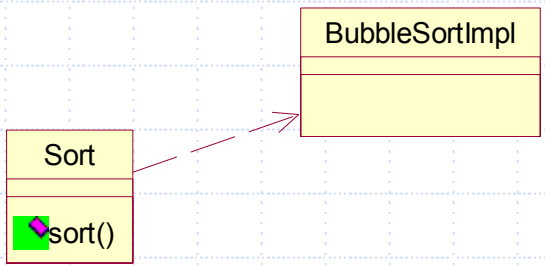
- **Association**
  - An association declares a connection (link) between instances of the associated classifiers (e.g.,classes).
- **Multiplicity**
  - The multiplicity property of an association end specifies how many instances of the classifier at a given end (the one bearing the multiplicity value) may be associated with a single instance of the classifier at the other end. A multiplicity is a range of nonnegative integers.
    - 0..1
    - 1
    - 0..*
    - *
    - 1..*
    - 1..6
    - 1..3,7..10,15,19..*

Class Diagram Contd.

- **Association & Multiplicity**
  - The Company has 1 to many Employees.



- Dependency
  - A dependency indicates a semantic relationship between two model elements. In the following example, the **sort()** method depends on the BubbleSortImpl to do the sorting.



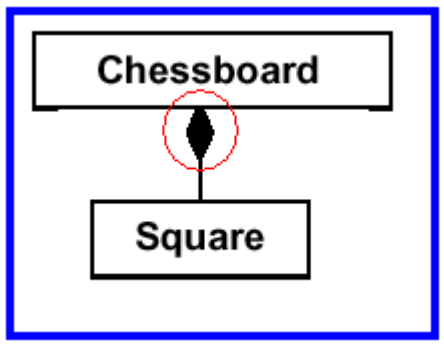
Class Diagram Contd.

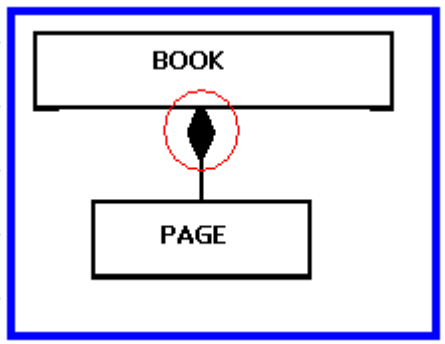
- **Generalization**
  - A generalization is a relationship between a more *general* element and a more *specific* element.

  - **Aggregation**
    - This defines Whole and part relationship between two classes.

Class Diagram Contd.

- **Composite aggregation**
  - Composite aggregation is a strong form of aggregation. This means that the composite object is responsible for the creation and destruction of the parts. If a composite object is destroyed, it must destroy all of its parts.

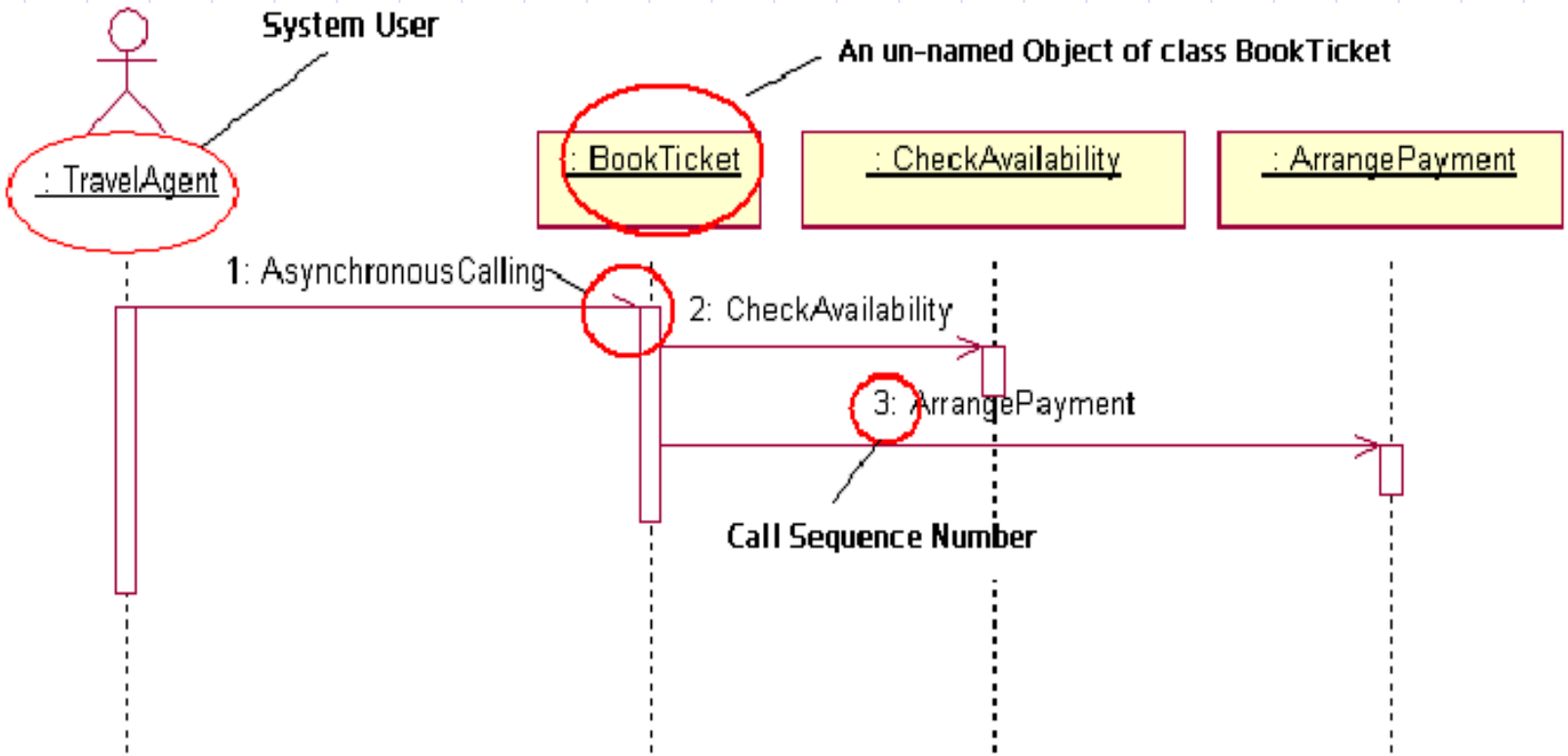




Sequence Diagrams

- A sequence diagram shows an interaction arranged in time sequence. It does not show the associations among the objects.
- Sequence diagram has two dimensions:
- the vertical dimension represents time and
- The horizontal dimension represents different objects.
- Sequence diagram shows:
- Types of objects involved in a scenario
- Messages exchanged
- Return values

Sequence Diagrams Contd..



Collaboration Diagrams

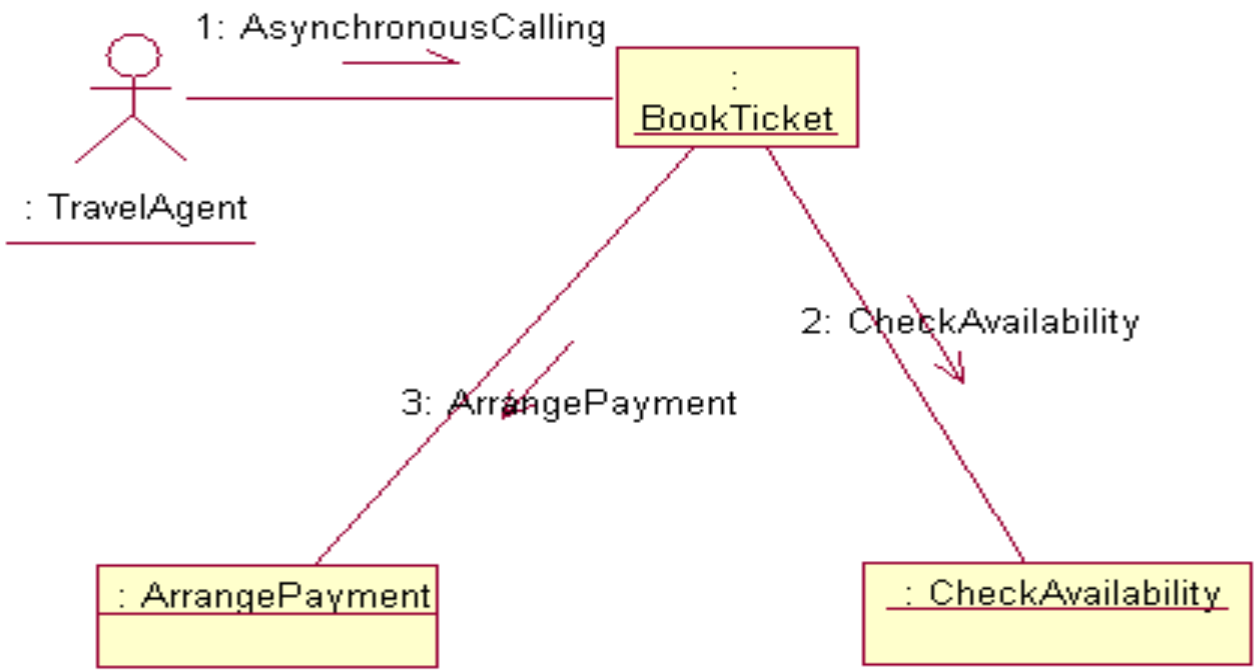
A collaboration diagram shows the relationships among the objects playing the different roles.

Collaboration diagram does not show time as a separate dimension,

The sequence of interactions and the concurrent threads must be determined using sequence numbers.

Collaboration and Sequence Diagram are called **Interaction Diagrams**.

Collaboration Diagrams



THANK YOU FOR YOU TIME.

Happy Coding and Diagramming.