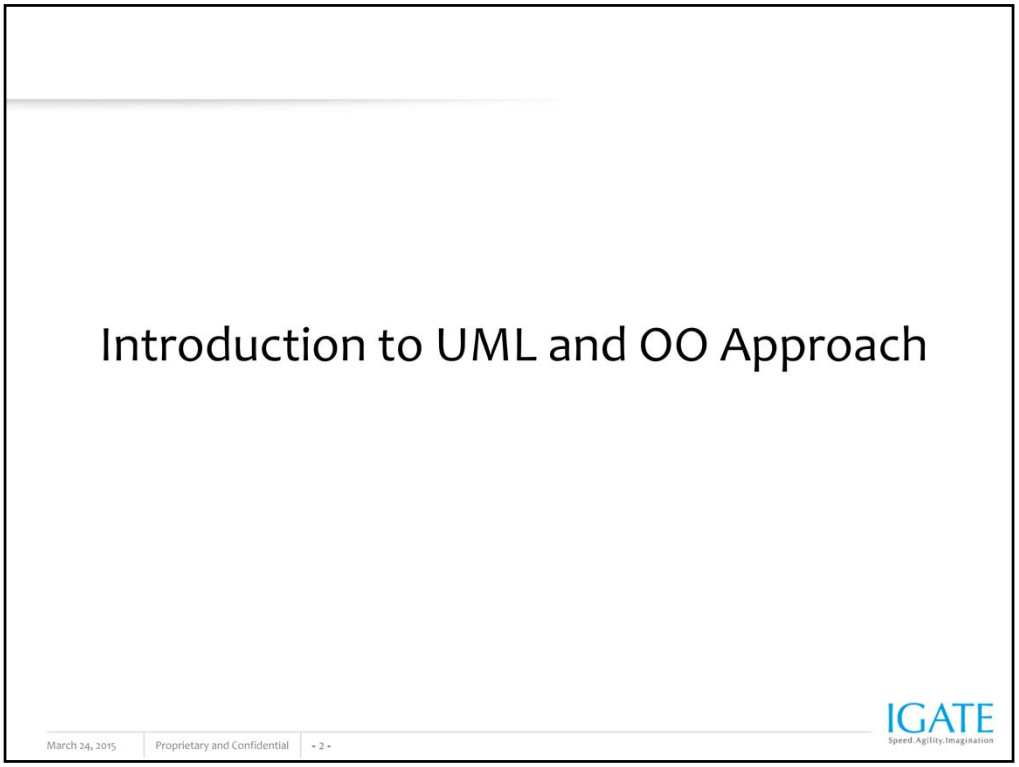
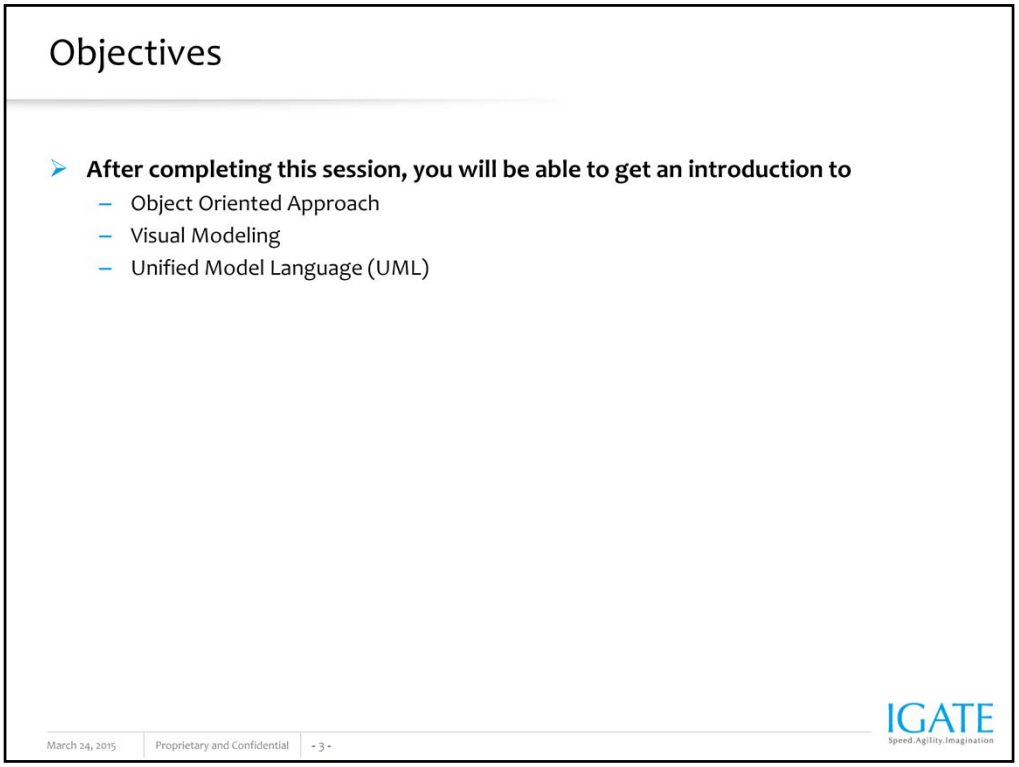
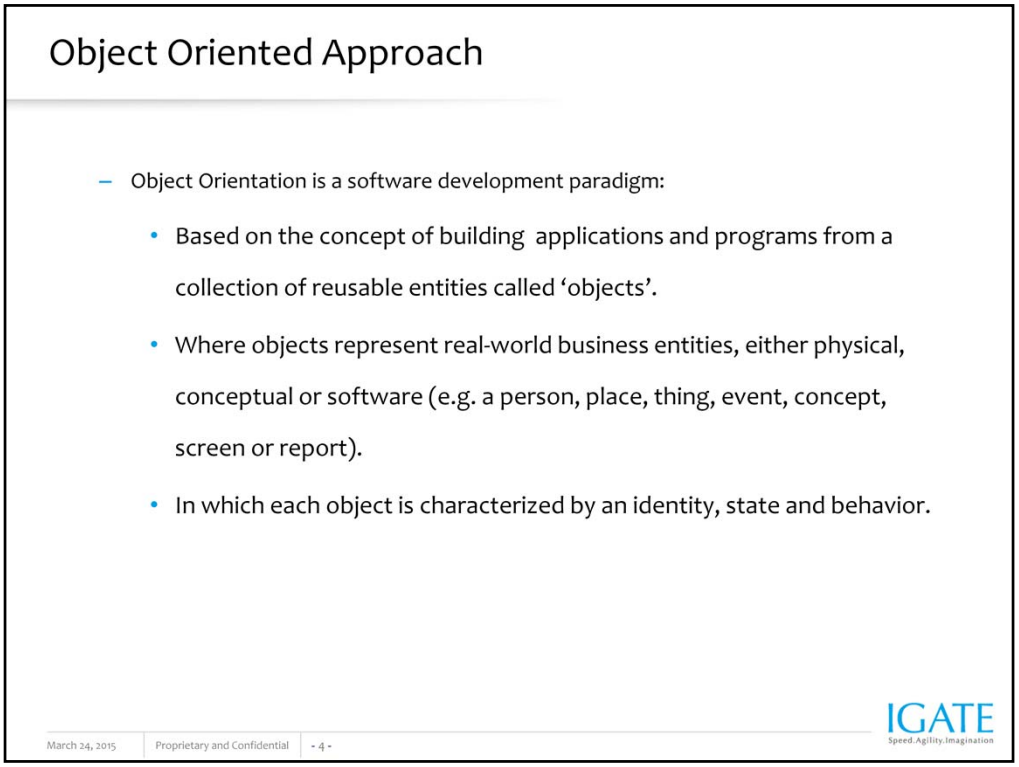

IGATE
Speed.Agility.Imagination


Introduction to UML and OO Approach
March 24, 2015
Proprietary and Confidential
- 2 -
IGATE
Speed.Agility.Imagination


Objectives
After completing this session, you will be able to get an introduction to
Object Oriented Approach
Visual Modeling
Unified Model Language (UML)
March 24, 2015
Proprietary and Confidential
- 3 -
IGATE
Speed.Agility.Imagination

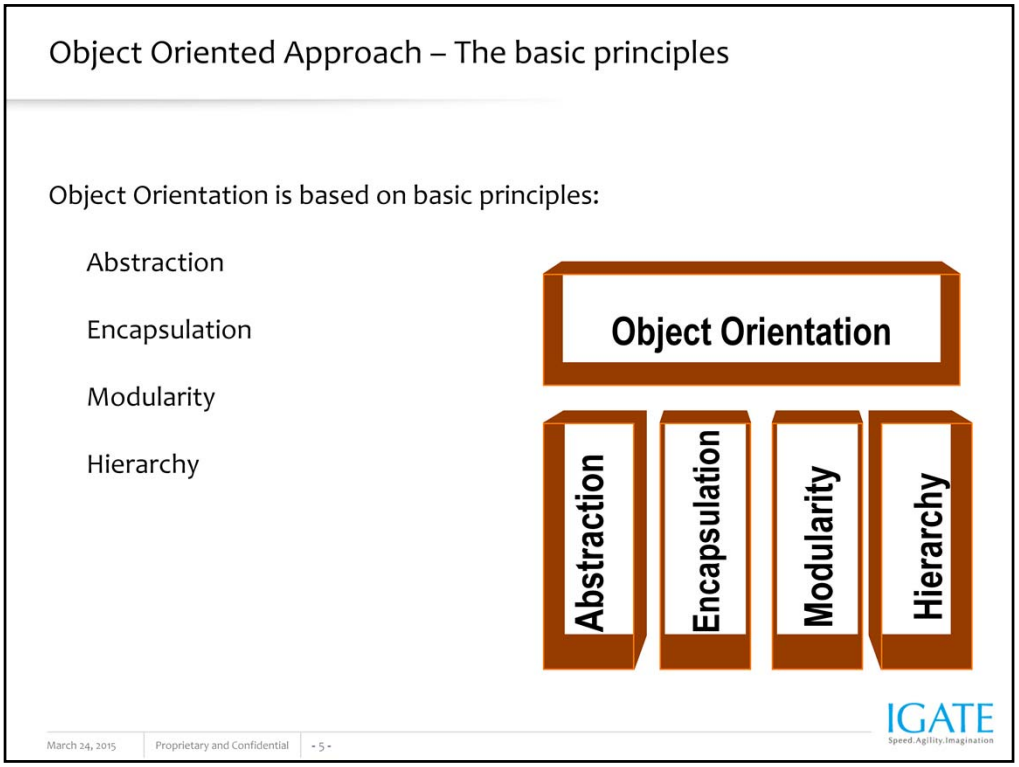


What is Object Oriented Technology?

• Object oriented technology is a well thought out methodology, developed for building many types of systems. It emerged as an alternative to the more traditional Procedural method.

• Procedural or functional method of software development, specified the steps(functions)that the computer needs to follow in order to achieve a particular goal. Where as, object-oriented technology aims on designing systems based on real-world, business entities (object).

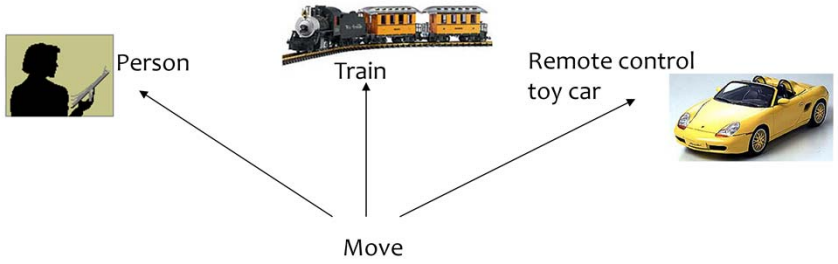
• An object-oriented model aims at reflecting the world we experience in reality. Thus, the objects themselves often correspond to phenomena in the real world that the system is to handle. For example, an object can be an invoice in a business system or an employee in a payroll system.



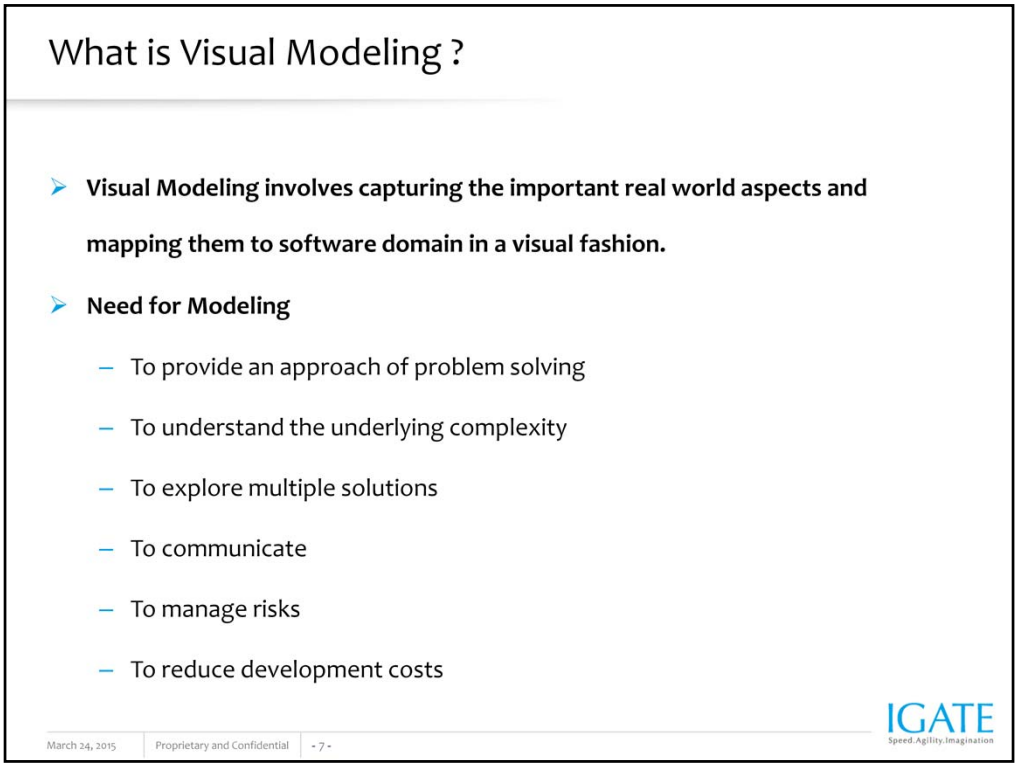
Abstraction is determining the essential qualities of an object. Abstraction helps provide a crisply defined conceptual boundary relative to user's perspective. Abstraction also means, that we look at the external behavior without bothering about internal details. Encapsulation literally means 'TO HIDE'. Encapsulation is the technique of separating the implementation details from the surrounding interface.
The concepts of abstraction and encapsulation are closely related - in fact they can be considered like two sides of a coin. Both need to go hand in hand. If we consider the boundary of a class interface, abstraction can be considered as the "User's" perspective, while encapsulation as the "Implementer's" perspective.
Modularity can be defined as the extent to which a system can be divided into internal components, called modules, which have high internal cohesion, low coupling and simple interfaces. A modular system is very easy to maintain. Modularity is obtained through decomposition - breaking up complex entities into manageable pieces.
Hierarchy is the systematic organization of objects in a specific sequence, in accordance to their complexity and responsibility. As we go up in the hierarchy, the abstraction increases.

Object Oriented Approach - Polymorphism

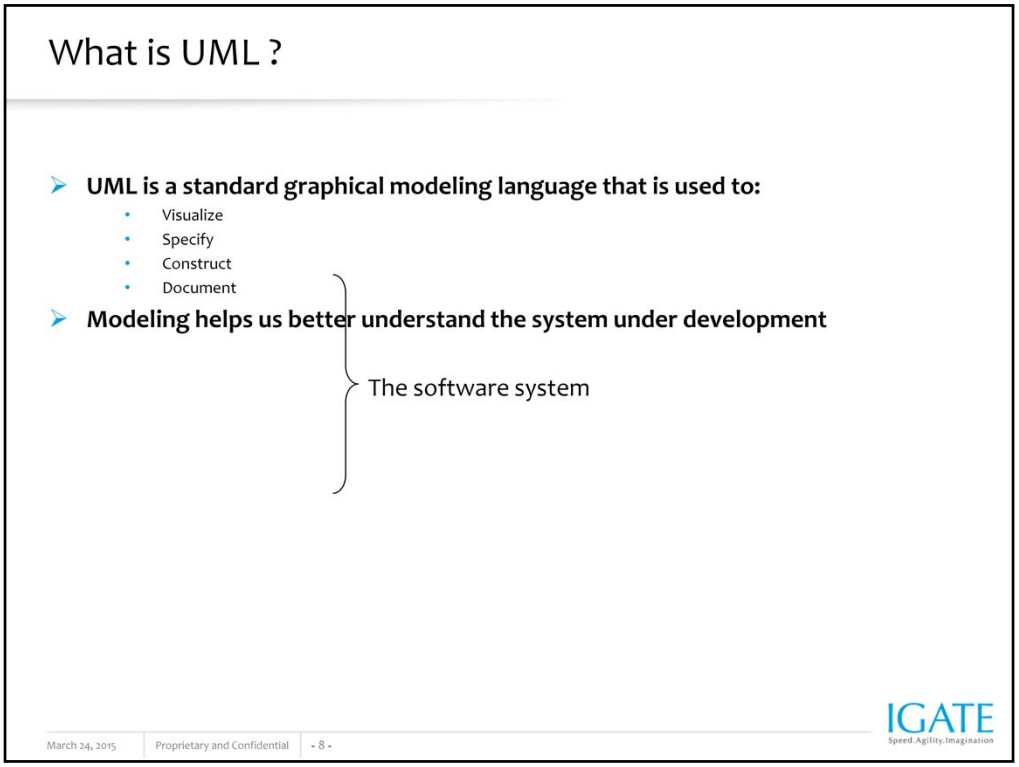
It is the ability to hide more than one implementation behind a single interface.
It is the ability of applying the same operation to different objects and have the resulting behavior vary depending on the class type.



The word Polymorphism is derived from the Greek word 'Polymorphous', which literally means 'having many forms'.
In object-orientation programming, there may be one or many implementations of a given interface. Polymorphisms allows the sending object to communicate with receiving objects, without having to understand what type of object it is, as long as the receiving objects support the messages.
Polymorphism allows different objects to respond to the same message in different ways!
Since we now have objects, which are self contained with their data and behavior, it becomes easier to address changes - thus reducing risks. The overall sizes of such system are usually more compact due to mechanisms like reuse which become available through objects.



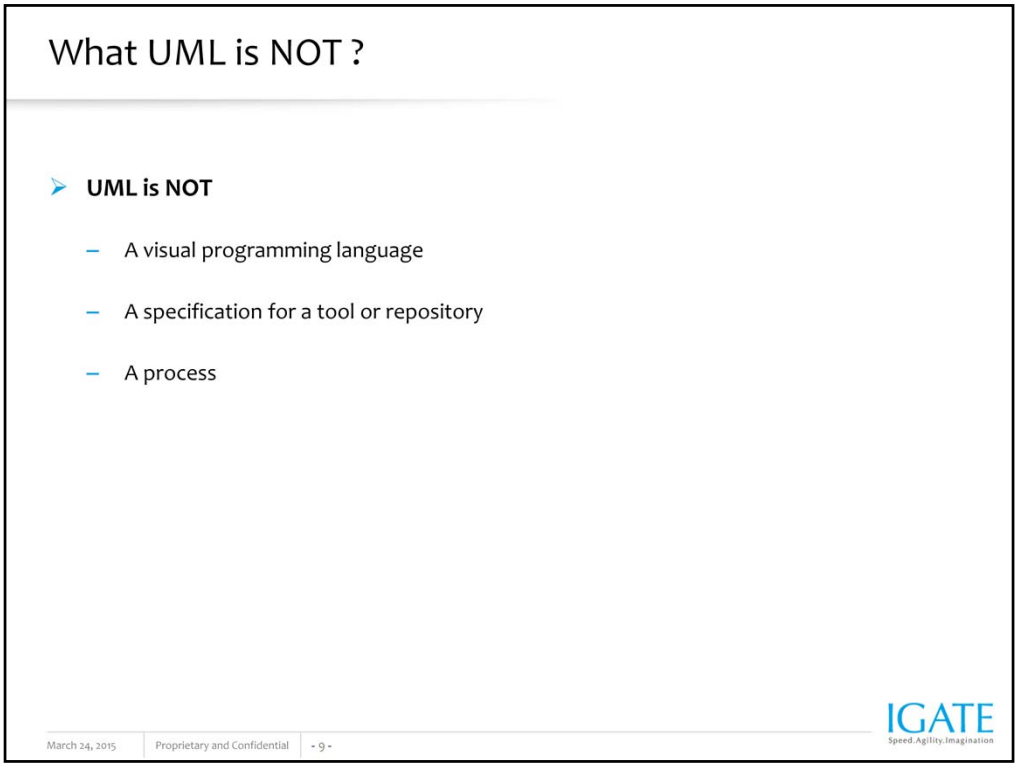
In the world today, we have business processes and software applications. As software professionals, our challenges lies in mapping the two.
Modeling involves capturing the important real world "things" and mapping these things to the computer system.
To do this, we need a method to show this mapping. This is modeling. And when modeling is done with symbols and notations, it becomes visual modeling.
Modeling creates a blueprint for the system we want to build, using a standard language that is understood by all. One such standard language is the UML.



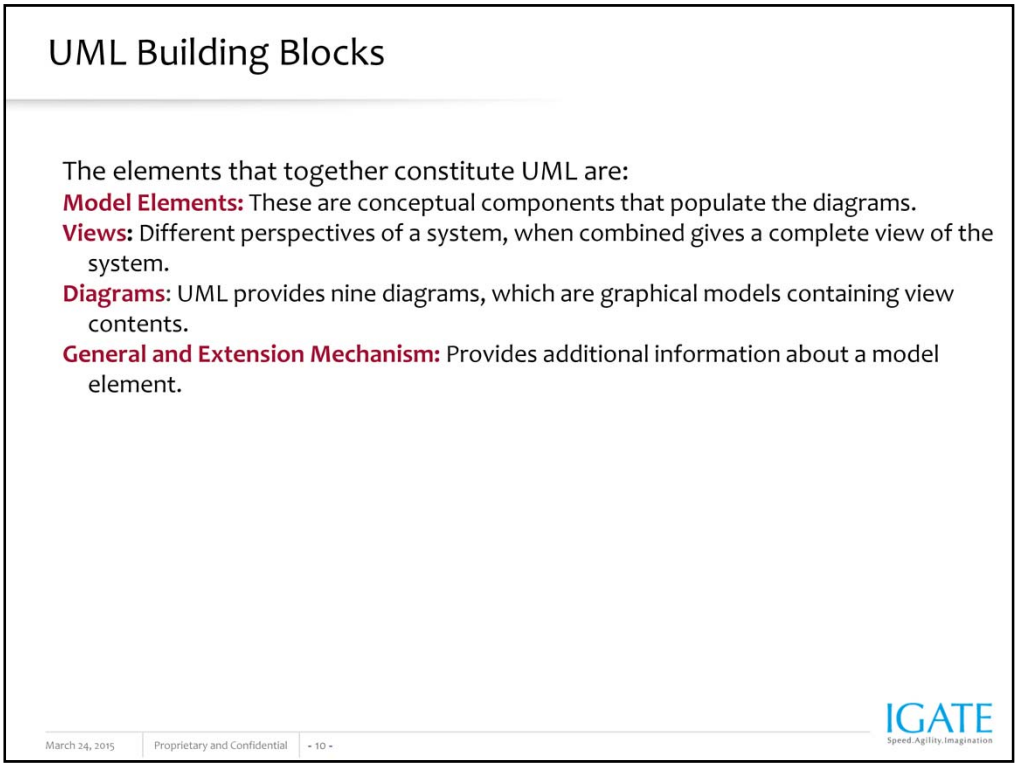
What is Unified Modeling Language?
UML is a modeling language, that is used to graphically depict the system. UML is used to do the following:
Visualising - Visual Model facilitates better communication and transcend what can otherwise be described textually.
Specifying - UML can help in specifying structure and behaviour of the system.
Constructing - Allows for construction of system from the various models.
Documenting - Models can help in documenting all decisions taken during the entire system development lifecycle.

Prior to UML, there were many methods with similar modeling languages having minor differences in overall expressive power - but there was no single "leading" modeling language. Lack of disagreement on a general purpose modeling language discouraged new users from adopting OO approach. In the period around mid 90s, there were efforts made in the direction of unifying the prominent methods available at the time - after a couple of drafts, UML was adopted by OMG in 1997. Jim Rumbaugh, Ivar Jacobson and Grady Booch, joined hands to develop an open, unifying standard for modeling systems.

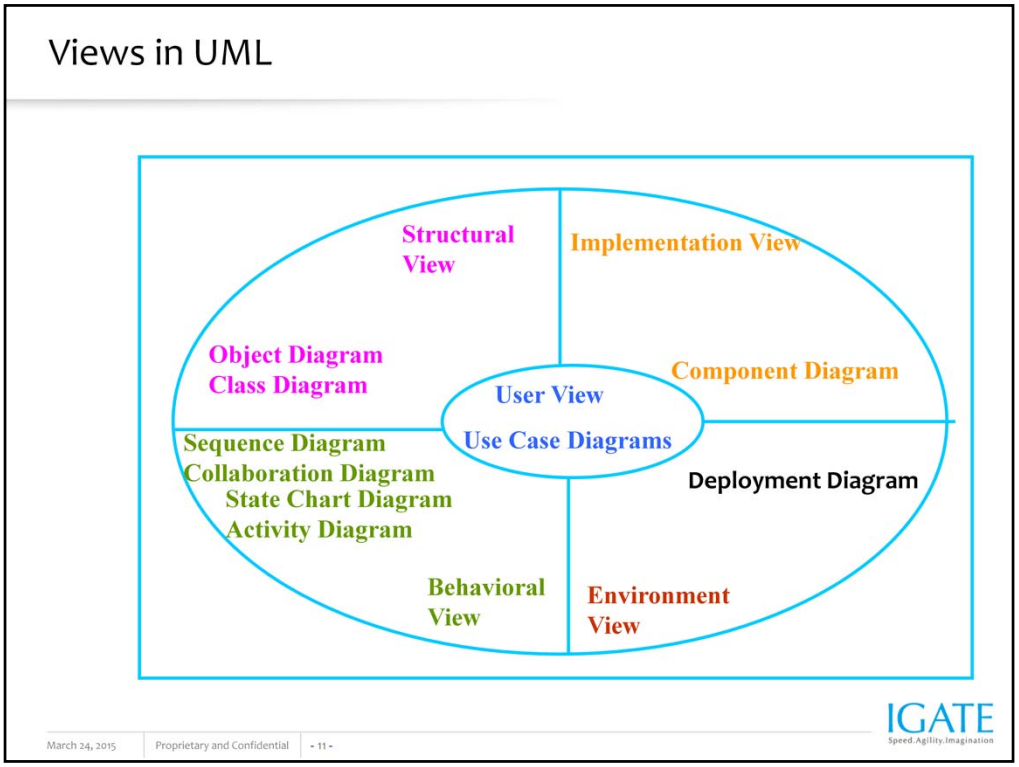
Why do we need to model? A model provides the blueprints of a system. It can encompass detailed plans, as well as more general plans of the system under construction. A good model includes those elements that are not relevant to the given level of abstraction. Every system can be described from different aspects using different models, and each model is therefore an abstraction of the system. We create models to better understand the system.



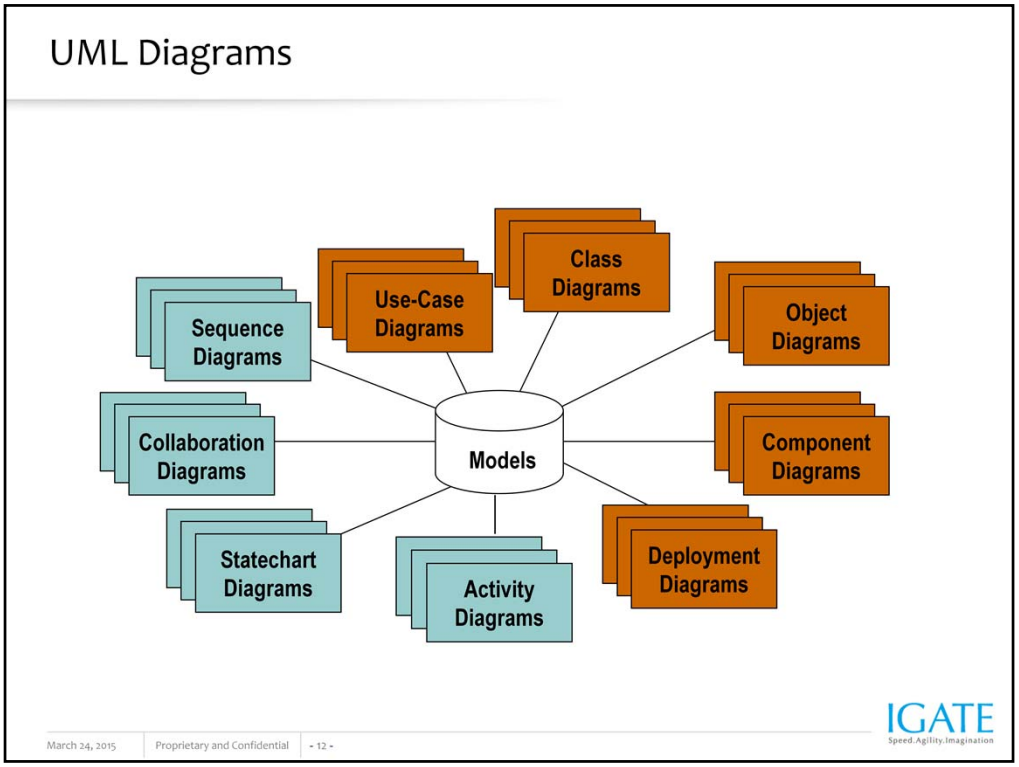
UML is not meant to be a visual programming language, rather it is a language meant for modeling - one can convey a concept or a specification but not a solution (which a program does).
It comprises of model elements - each with its own associated notation and semantics.
It is not meant to specify a tool or repository in terms of interfaces, storage or run time behavior.
UML is not a Process, but enables processes. UML does not require a process. However, it enables and promotes object oriented and component based processes.



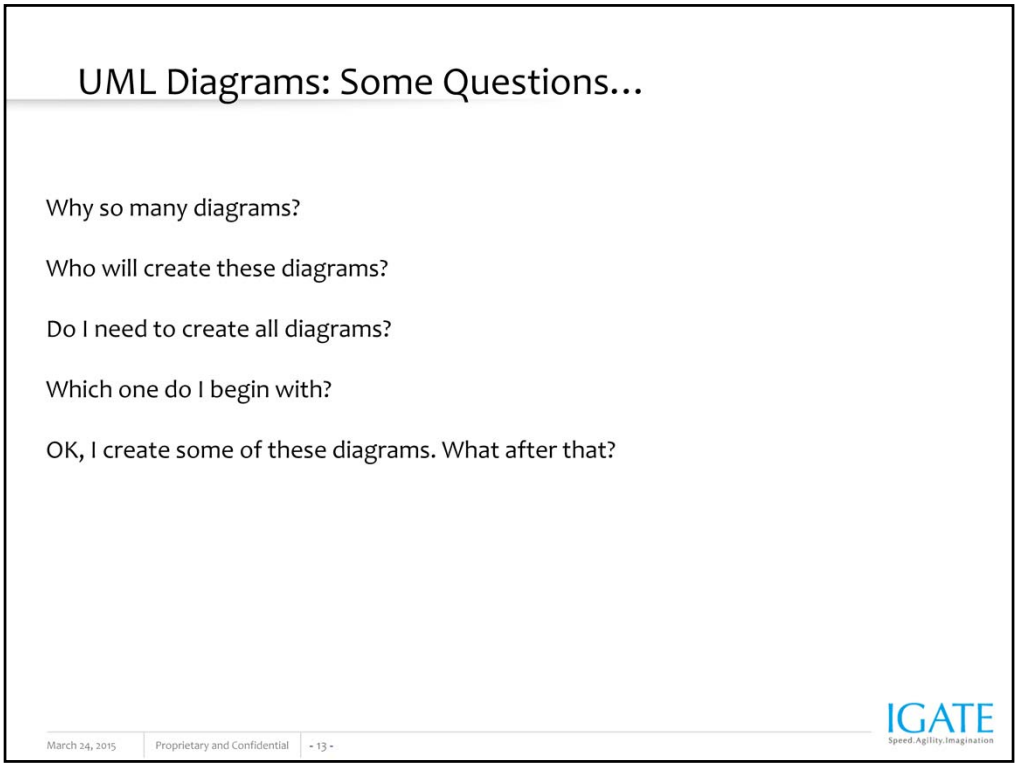
Model elements form the atomic level of the UML hierarchy. Each element has a predefined meaning and a graphical notation and semantics associated with it.
To understand the system complexity, different viewpoints are required - this can be obtained through different views and diagrams.
Often, one has to decide which views/ diagrams are required for the system under consideration. When deciding on this, consider the reason for communication of models. Depending on what aspects of the system need to be emphasized on, the views/ diagrams can be chosen. (To that extent, each view/diagram is an independent entity in itself).
In addition, UML provides general and extension mechanisms, which can be used to provide additional information about model elements, or extend the available notation and semantics.



For the end user, the User View is useful to understand the functionality that will be provided by the system.
Analysts and Designers can get the view of the structural aspects of the system through the Structural view.
The Behavioral view can give important inputs in terms of Performance, scalability and through put - which can be made use of by the System integrator.
Programmers would find the implementation view helpful, while the Environment view can convey decisions relating to system topology, delivery mode, installation and communication.



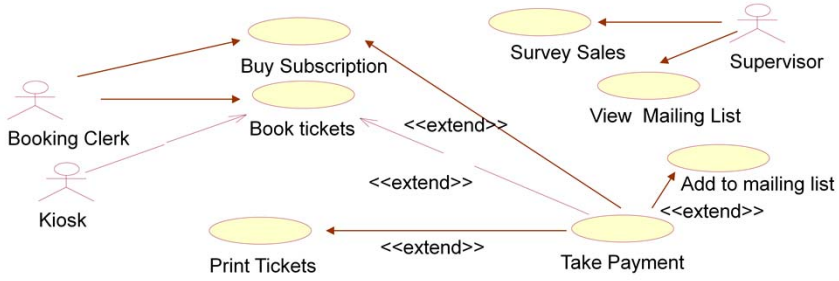
UML provides 9 different diagrams to capture different aspects of the system under development. These diagrams are briefly touched upon in the subsequent slides.



Understanding the answers to these questions is very important to get a proper perspective to UML.

UML Diagrams: Use Case Diagram

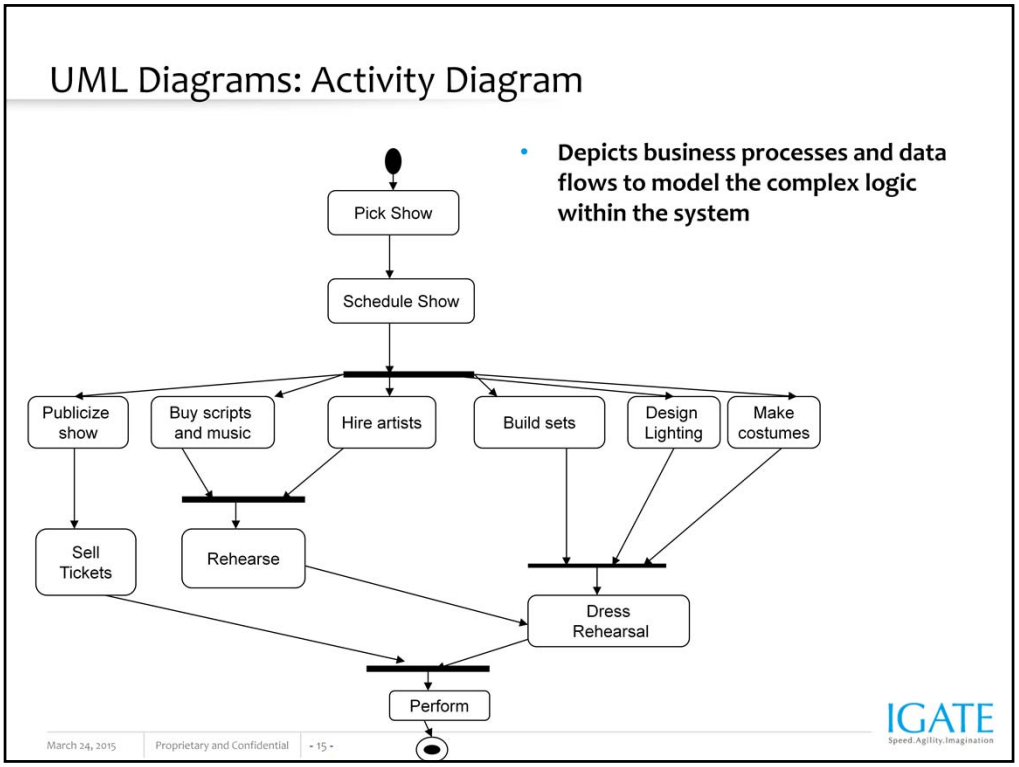
- **Used to capture the functional requirements of the system from a user's perspective**
- **Shows actors, use cases and their inter-relationships**
  - Actor: External to system, directly interacting with the system
  - Use Case: Functionality required from the system



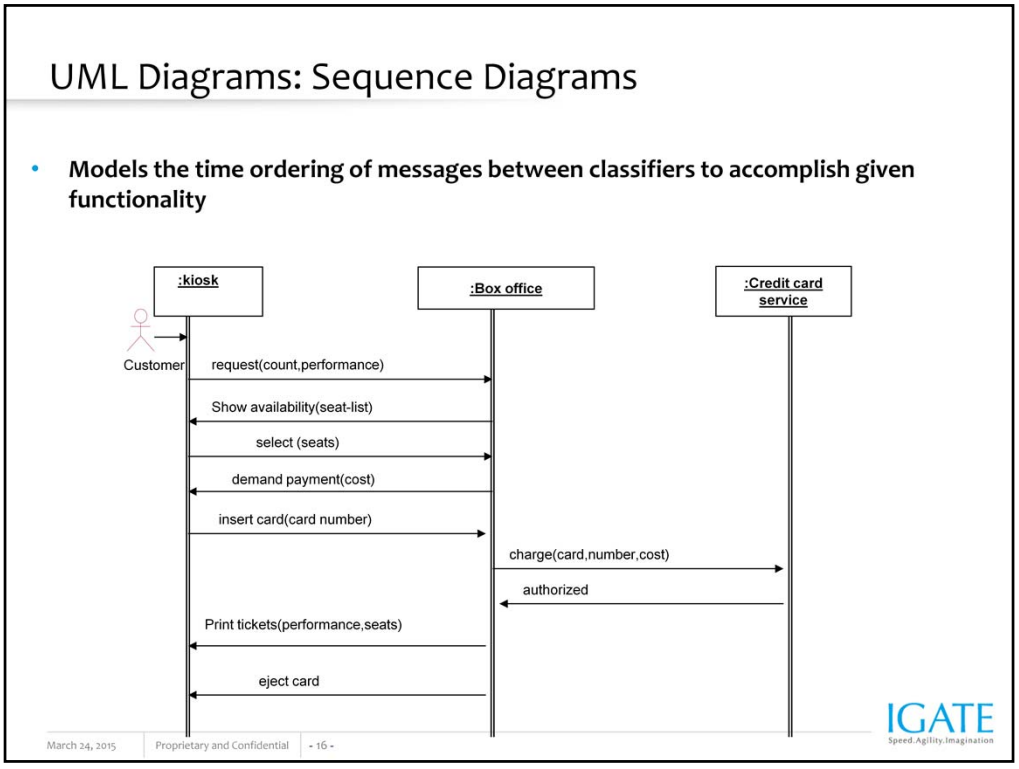
IGATE Speed.Agility.Imagination

• A use-case diagram is created to visualize the interaction of your system with the outside world. The use case diagrams can contain actors, use cases and relationships between actors and use cases.

• An actor is a user of the system. It could be a person, system or device who/which is outside the boundary of the system and uses the system.
• Use cases are services or functions provided by the system to its users.
• Use Case Diagrams are drawn to understand the functional requirements of the system.



- Provides a way to model the workflow of a business process.
- Similar to flowchart.
- Used for modeling the sequence of activities in a process.

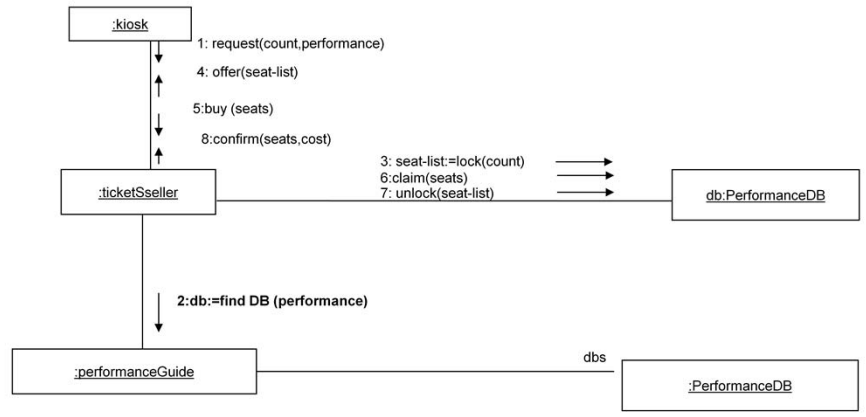


• Sequence diagram shows object interaction in a time-based sequence.

• It is associated with use cases. It shows you step by step what must happen to accomplish a piece of functionality in the use case. A sequence diagram is also known as interaction diagram.

UML Diagrams: Collaboration Diagrams

- **Focuses on the structural organisation of objects that send and receive messages. Known as communication diagrams in UML 2.0.**

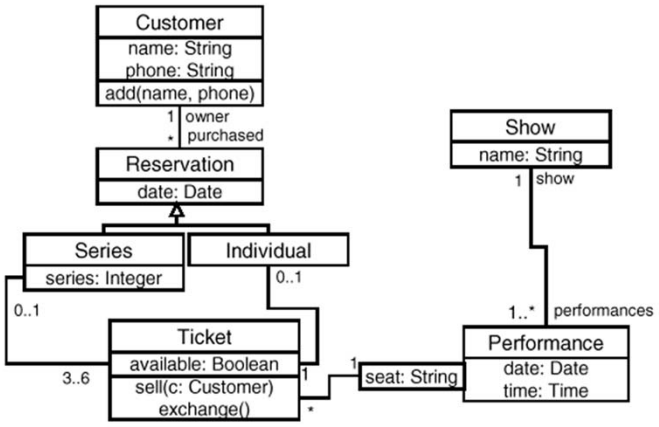


IGATE Speed.Agility.Imagination

• A collaboration diagram is a different graphical view of a use case flow of event. This type of diagram shows the objects and their links to one another.
• A collaboration diagram is also known as an interaction diagram. Semantically it is similar to a sequence diagram. It provides a view of interactions or structural relationships that occur between objects and object like entities.

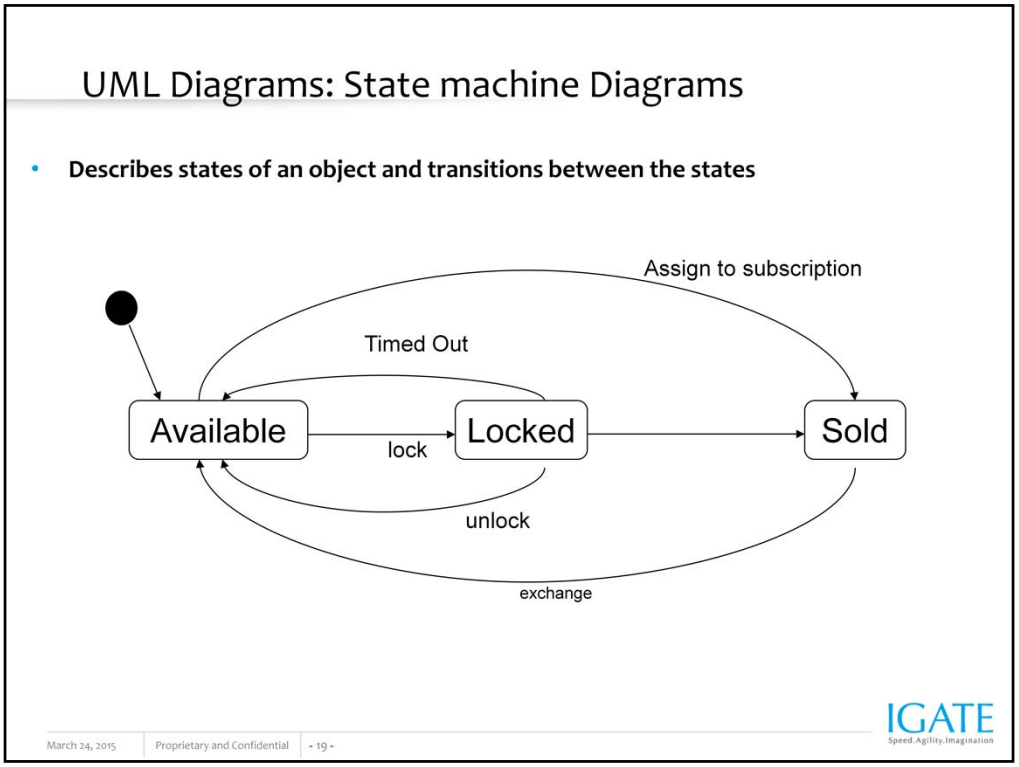
UML Diagrams: Class Diagrams

- **Models a collection of static model elements such as classes, their contents and relationships**

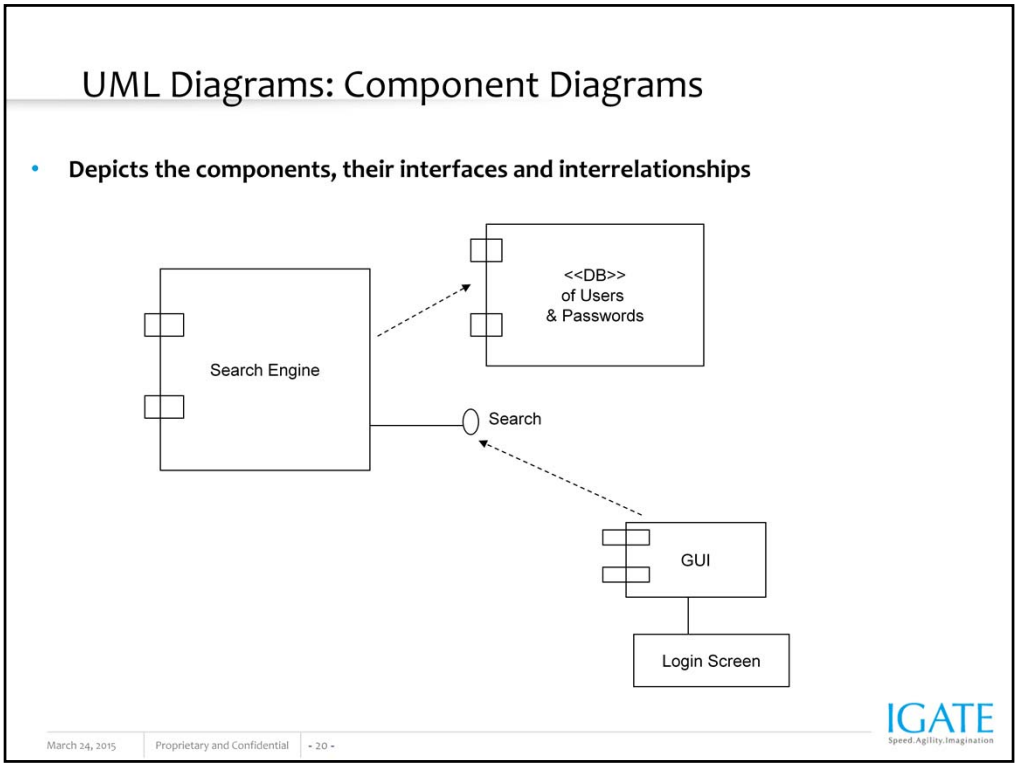


IGATE Speed.Agility.Imagination

• Class diagrams are created to show the structure of the system. Class diagrams contain classes, relationships, and multiplicity indicators.
• Objects in interaction diagrams (both sequence and collaboration diagrams) are grouped into classes, where a class is a collection of objects with common structure, common behavior, common relationships, and common semantics.
• Each class also has data associated with it. This is the structure of the class, represented by its set of attributes. The class behaviour is represented using operations.



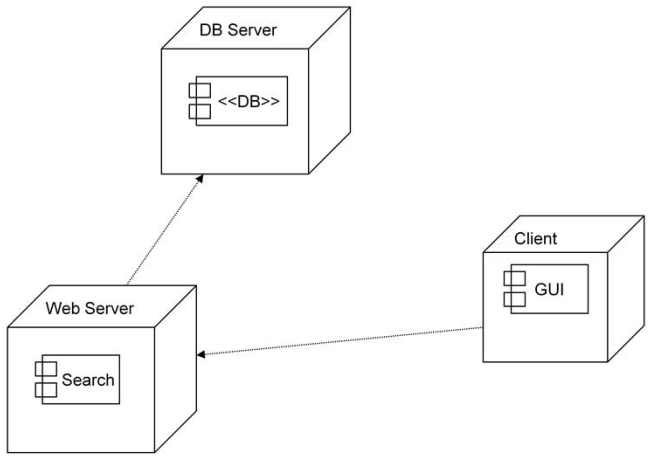
- Models the dynamic behavior of objects.
- The Statechart diagrams show the sequences of states that an object goes through, the events that cause a transition from one state to another, and the actions that result from a state change.
- Statechart diagrams are closely related to activity diagrams.
- Each state represents a named condition during the life of an object during which it satisfies some condition or waits for some event.
- A statechart diagram typically contains one start state (mandatory) and multiple end states (end states are optional).



- It provides a physical view of the current model.
- Illustrates the organization and dependencies among software components.
- Component diagrams contain: Components, Interfaces, Realization and Dependency relationships.

UML Diagrams: Deployment Diagrams

- **Shows the execution architecture of the application**

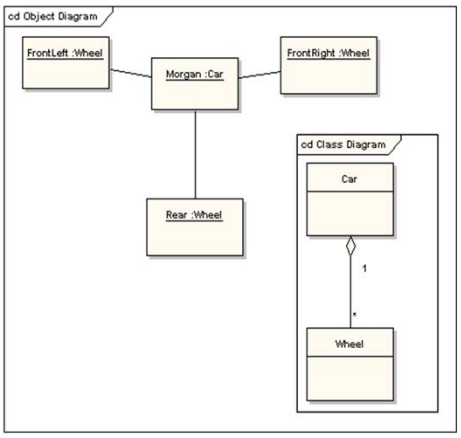


IGATE
Speed.Agility.Imagination

- Depicts the physical resources in a system including nodes, components, and connections.
- Provides a picture of the hardware requirements of system.

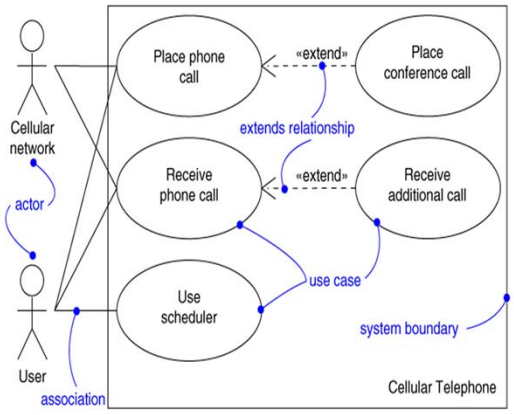
UML Diagrams: Object Diagrams

- **A snapshot diagram depicting objects and their relationships at a given point in time**



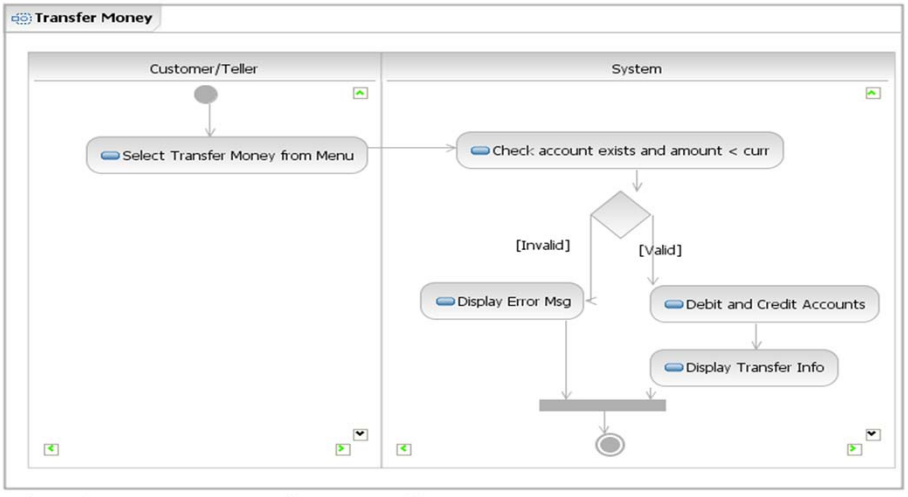
- The object diagrams describe an instantiation of class diagram at a particular instance of time. They may be used to explore different configurations of objects - which when combined can be generalized into relevant class diagrams.
- Depending on the multiplicity values associated with class relationships, there may be different number of objects of the classes that exist in the class diagram. “Links” relate the objects - they are instances of relationships of class diagrams.
- Since Object diagrams indicate objects of system at particular instance of time, they can be used to verify system performance in terms of memory utilization of objects & communication links between them. Thought can be given to minimize use of resources by techniques like pooling or queuing, batch processing etc.

UML Diagrams: Exercises

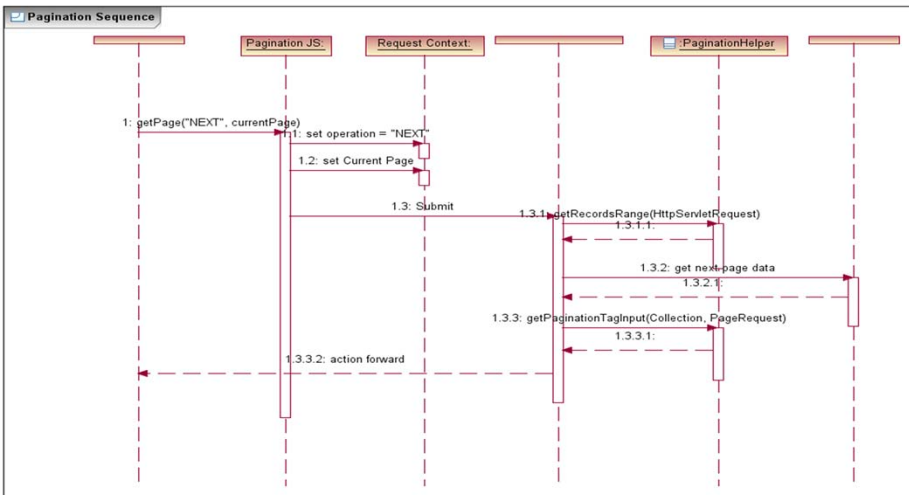


- **Explain your understanding of this diagram**

UML Diagrams: Exercises

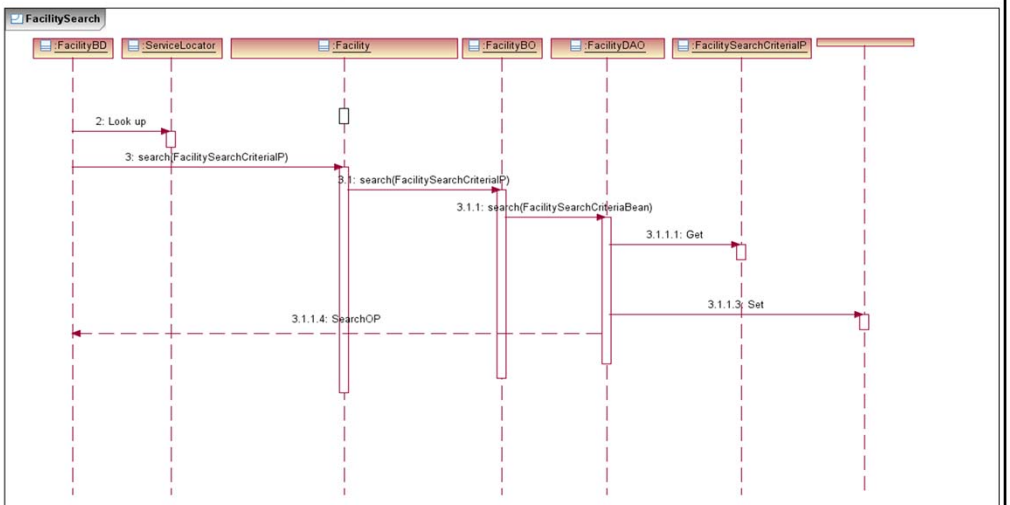


UML Diagrams: Exercises



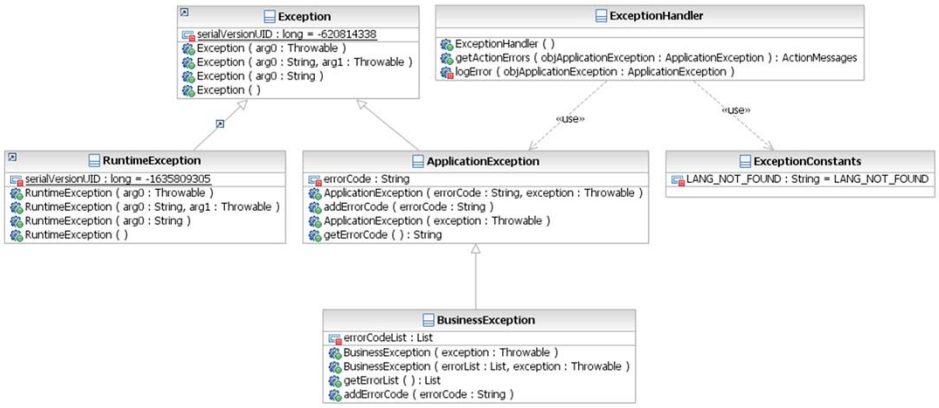
- **Explain your understanding of this diagram**

UML Diagrams: Exercises



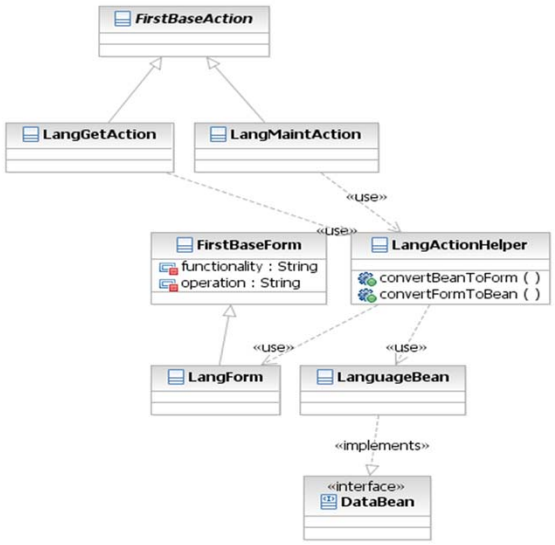
- **Explain your understanding of this diagram**

UML Diagrams: Exercises



- **Explain your understanding of this diagram**

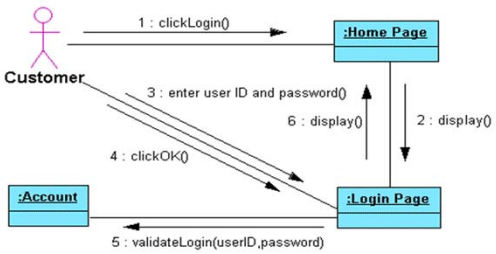
UML Diagrams: Exercises



- **Explain your understanding of this diagram**

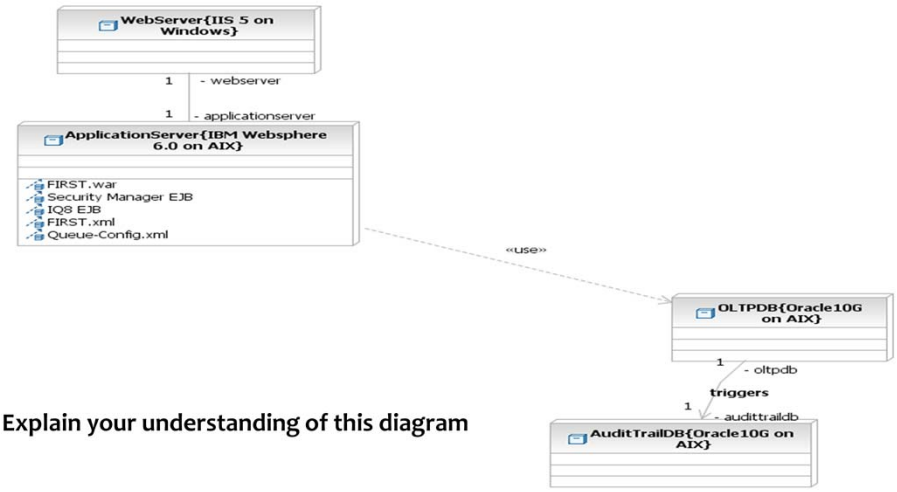
IGATE
Speed.Agility.Imagination

UML Diagrams: Exercises



- Explain your understanding of this diagram

UML Diagrams: Exercises



- **Explain your understanding of this diagram**

Summary

- **You now have an understanding of**
  - Object Oriented Approach
  - Visual Modeling
  - Unified Model Language (UML)