## LESSON 6

## OBJECT ORIENTED DESIGN

In the *design* phase, we move from a requirements specification to a *design specification*. The design specification gives the system structure. The design tasks are to:

* Break the programming task into manageable parts.
* Define the relationships among the parts.
* Incorporate any required or appropriate pre-existing components.
* Keep the design independent of implementation language and hardware *details*. (We can, however, use knowledge of high-level architectures and language paradigms.)

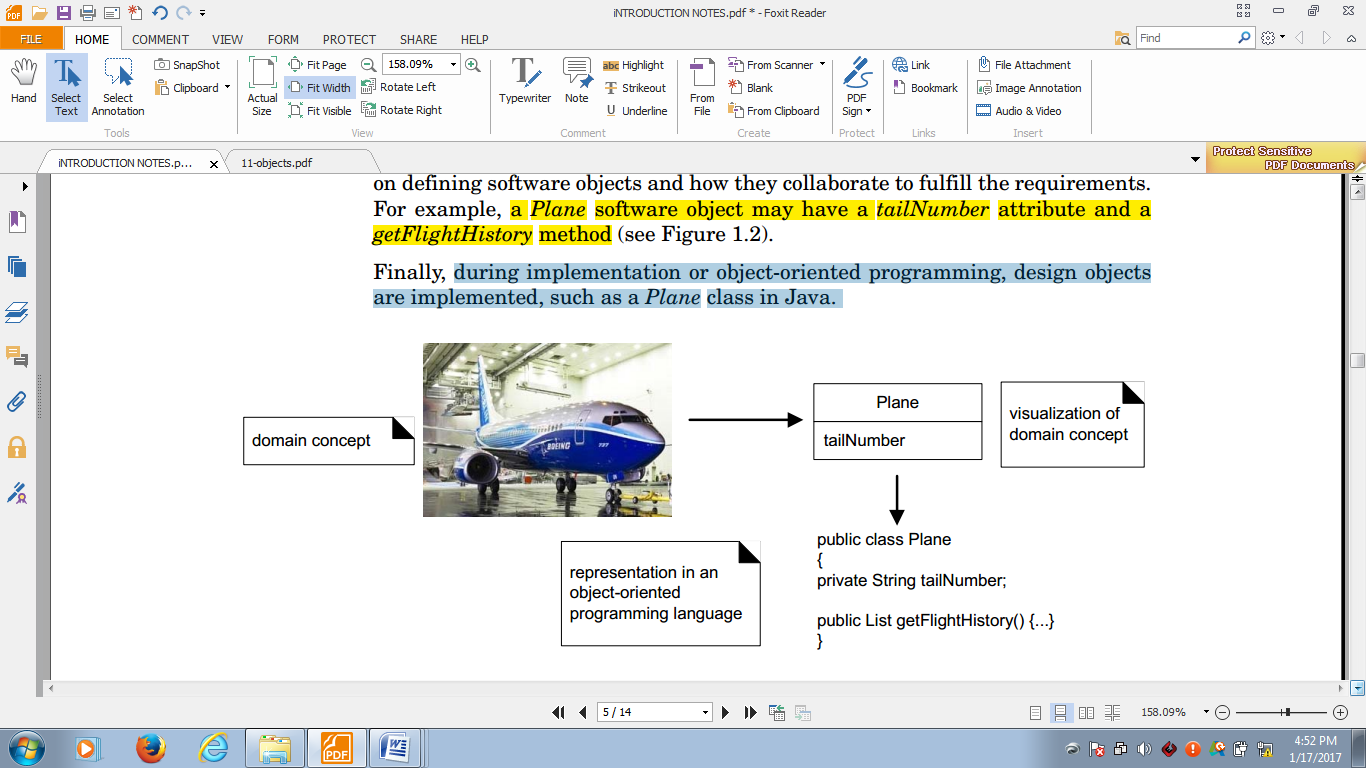
**Design** emphasizes a conceptual solution(in software and hardware) that fulfills the requirements, rather than its implementation. *Example*, a description of a database schema and software objects.

Design ideas often exclude low level or “obvious” details—obvious to the intended consumers.

We normally have two levels of designs:

1. During **object-oriented system analysis**, emphasis is on finding and describing the objects—or concepts—in the problem domain. e.g. Plane, Flight, and Pilot can be used to describe a flight information system.
2. During **object-oriented system design**, emphasis is on defining software objects and how they collaborate to fulfill the requirements e.g. a Plane software object may have a tailNumber attribute and a getFlightHistory method.

During **implementation or object-oriented programming**, design objects are implemented e.g: such as a Plane class in Java language



### Object relational modeling

A **model** is an abstraction of the underlying problem. It represents the **domain** - the actual world from which the problem comes.

Models consist of objects that interact by sending each other messages. Think of an object as "alive" agent. Objects have things they know (attributes) and things they can do (behaviors or operations). The values of an object's attributes determine its state.

A **model** is a semantically closed abstraction of a system which represents a complete and self-consistent simplification of reality, created in order to better understand the system.

**Why do we model?**

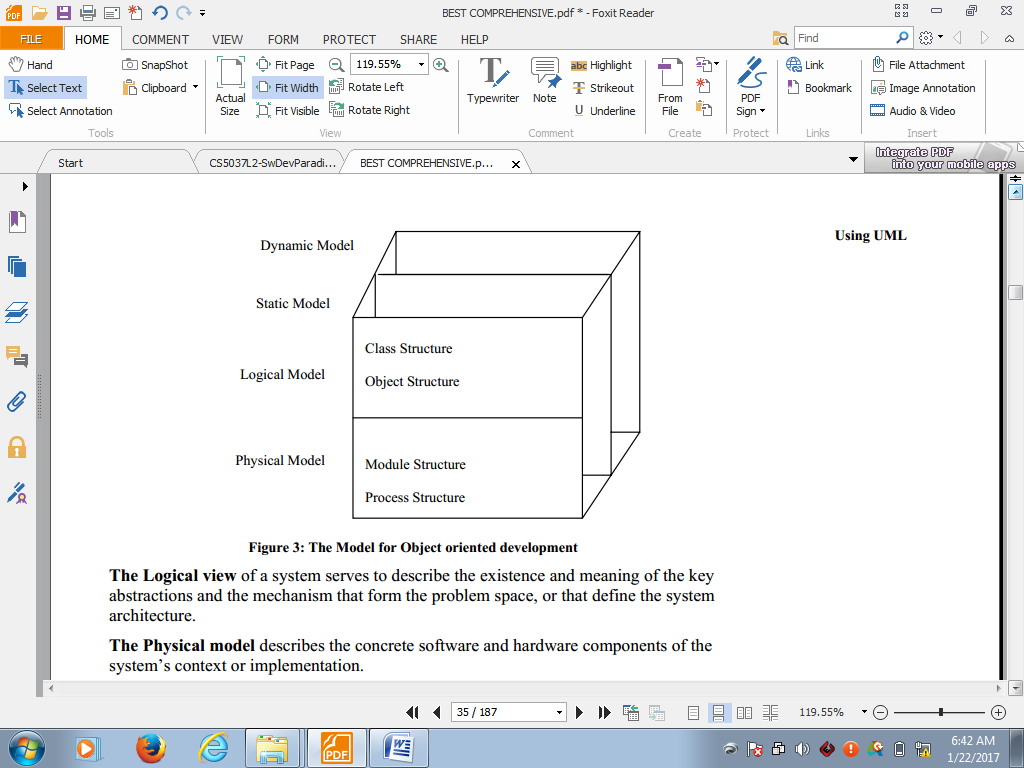
A model is a simplification of reality. We build models so that we can better understand the system we are developing .

**Benefits of Modeling**

1. Models help us to visualize a system as it is or as we want it to be.
2. Models permit us to specify the structure or behavior of a system.
3. Models give us a template that guides us in constructing a system.
4. Models document the decisions we have made in the process of analysis.
5. Provide structure for problem solving i.e. they are used as experiments to explore multiple solutions
6. They furnish abstractions to manage complexity
7. Reduce time-to-market for business problem solutions
8. Decrease development costs
9. Manage the risk of making mistakes

##### Types of models in Object Orientation

The models for object oriented development is as shown below:

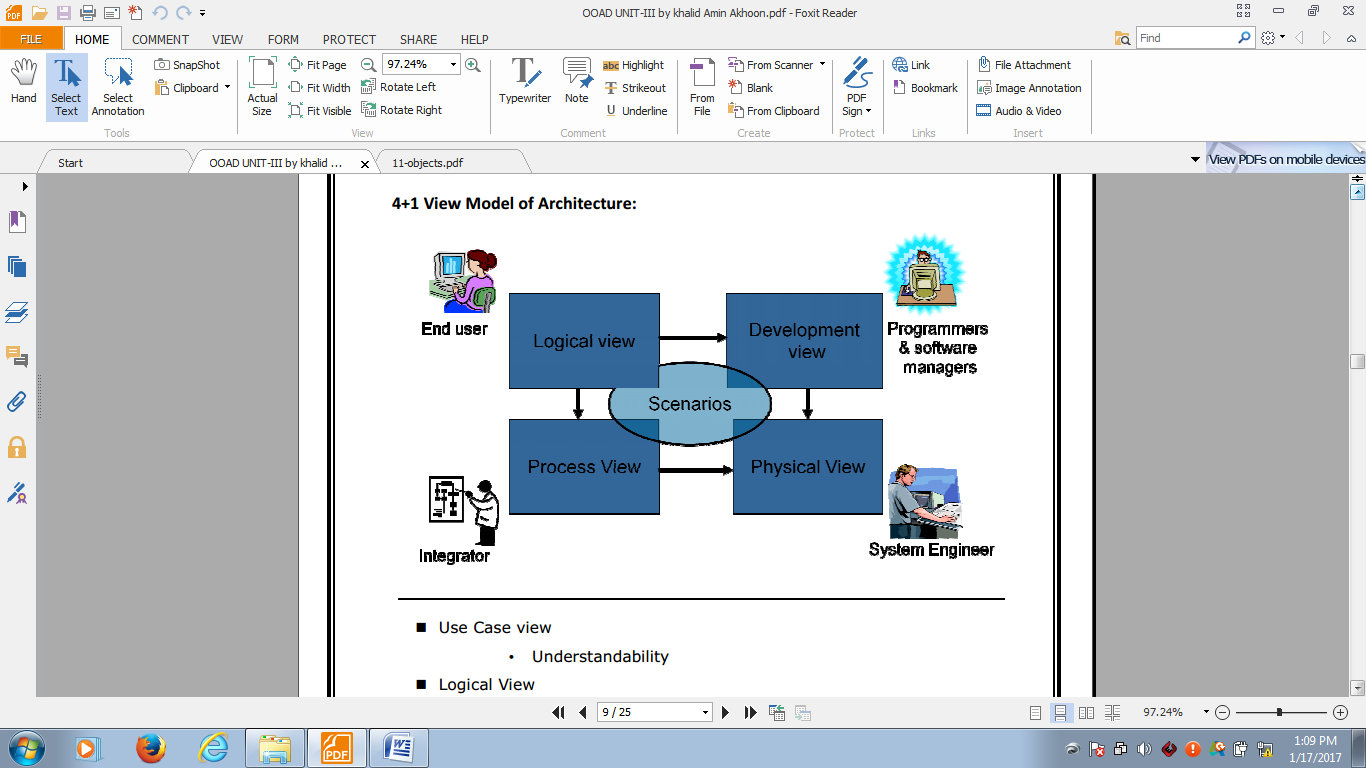


The **Logical model** of a system serves to describe the existence and meaning of the key abstractions and the mechanism that form the problem space, or that define the system architecture.

The **Physical model** describes the concrete software and hardware components of the system’s context or implementation.

##### View Models of System Architecture

A **view** is a projection into an organization and structure of a system’s model, focused on one aspect of that system.



1. Use Case view: Understandability
2. Logical View: Functionality
3. Process View: Interaction and performance i.e. Scalability and Throughput
4. Development/Implementation View: Software management
5. Deployment View i.e. System topology, Delivery and Installation.

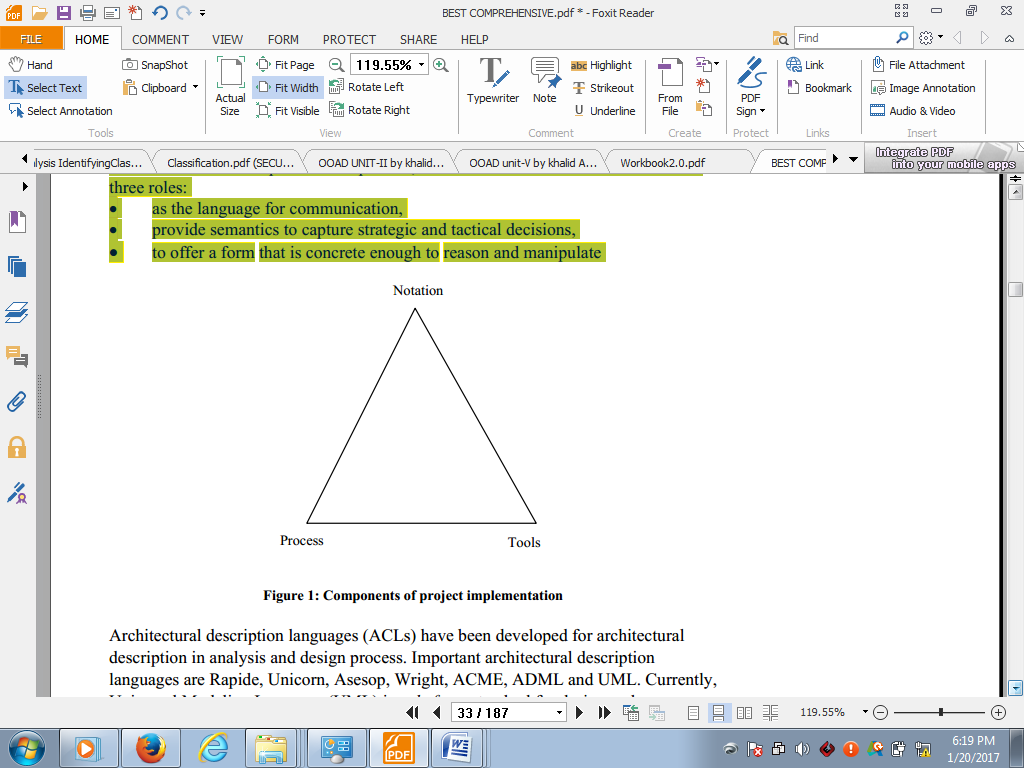
##### Overview of MVC architecture

make notes....

### UML Language AND Meta-Models

#### Overview of UML

For successful project implementation the three essential components are: process, tools and notations.



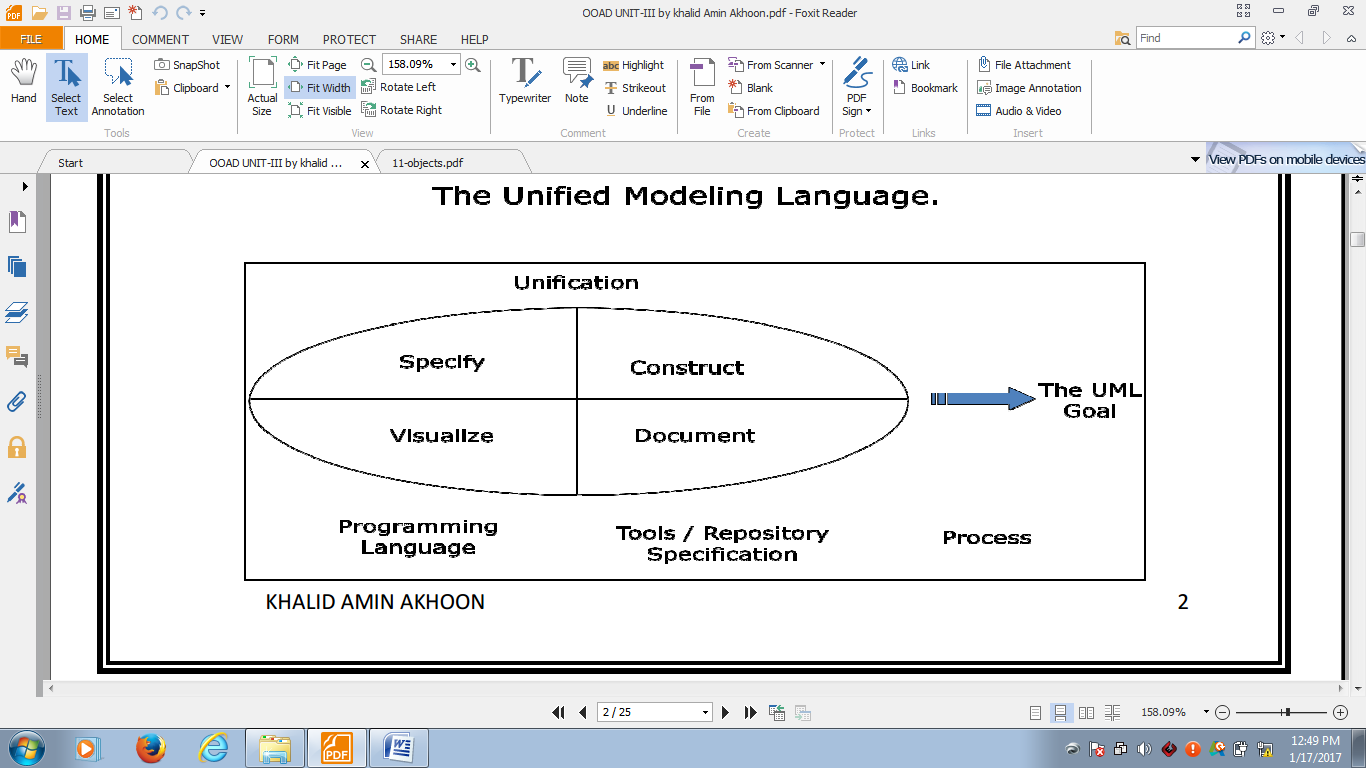
The notation serves three roles:

1. as the language for communication,
2. provide semantics to capture strategic and tactical decisions,
3. to offer a form that is concrete enough to reason and manipulate

#### Basic concept of unified modeling language(UML)

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system. UML is used to express the construct and the relationships of complex systems.

According to the Object Management Group( OMG) specification, "The Unified Modeling Language (UML) is a graphical language for: visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.



Example of artifacts : requirements , architecture , design , Source code , test cases

* **Visualizing**: A graphical notation articulates and unambiguously communicates the overall view of the system (problem domain).
* **Specifying:** UML provides the means to model precisely, unambiguously and completely.
* **Constructing**: Models built with UML have a “design” dimension to it; these are language independent and can be implemented in any programming language.
* **Documenting:** Every software project involves a lot of documentation - from the inception phase to the deliverables.

#### A brief History of UML

* Towards the mid-80s, OO approach got recognition.
* People were inclined to follow a functional phase with an object-oriented design phase.
* The navigation from functional analysis to implementation was difficult and often resulted in just the encapsulation of low level objects. The results were not satisfactory.
* People started using Object Oriented Methods to model the elements of the system. The Method gave the process as well as the representation rules for the modeling the analysis and design decisions.
* In the early 90s around fifty different Methods blossomed.
* The important among them were:

 Grady Booch method: based on categories and subsystems. Ref: *Object Oriented analysis and Design-Grady Booch*

 OMT: Object Modeling Technique by James Rumbaugh based on classes and associations. Ref: *Object Oriented Modeling and Design James Rumbaugh*

 Objectory or Object Oriented Software Engineering(OOSE)-by Ivar Jacobson Introduced the concept of use cases

Booch was strong in design, and Jacobson was strong in behavioral analysis.

* Informal unification of the above 3 methods started when the three started incorporating the views of each other in their methods. When the three came together at Rational Software Corp., they unified formally their methods to create UML.
* It was submitted to OMG in Jan 1997 and was formally adopted by OMG as a standard.(UML1.0-Jan97)

##### UML Features

The major features of UML are:

1. defined system structure for object modeling
2. support for different model organization
3. strong modeling for structure and behavior
4. clear representation of concurrent activities
5. support for object oriented patterns for design reuse.

##### Importance of UML

1. Blueprints are the standard graphical language that both architects and builders must learn as part of their trade.
2. When writing a software, the more complicated the underlying system, the more critical the communication among everyone involved in creating and deploying the software.
3. The UML gives everyone from business analyst to designer to programmer a common vocabulary to talk about software design.
4. The UML is applicable to object-oriented problem solving.

*Recall these :*

A **system** is a collection of sub-systems organized to accomplish a purpose and described by a set of models from different viewpoints. A **subsystem** is a grouping of elements of which some constitute a specification of the behavior offered by other contained elements.

A **diagram** is a graphical presentation of a set of elements/symbols.

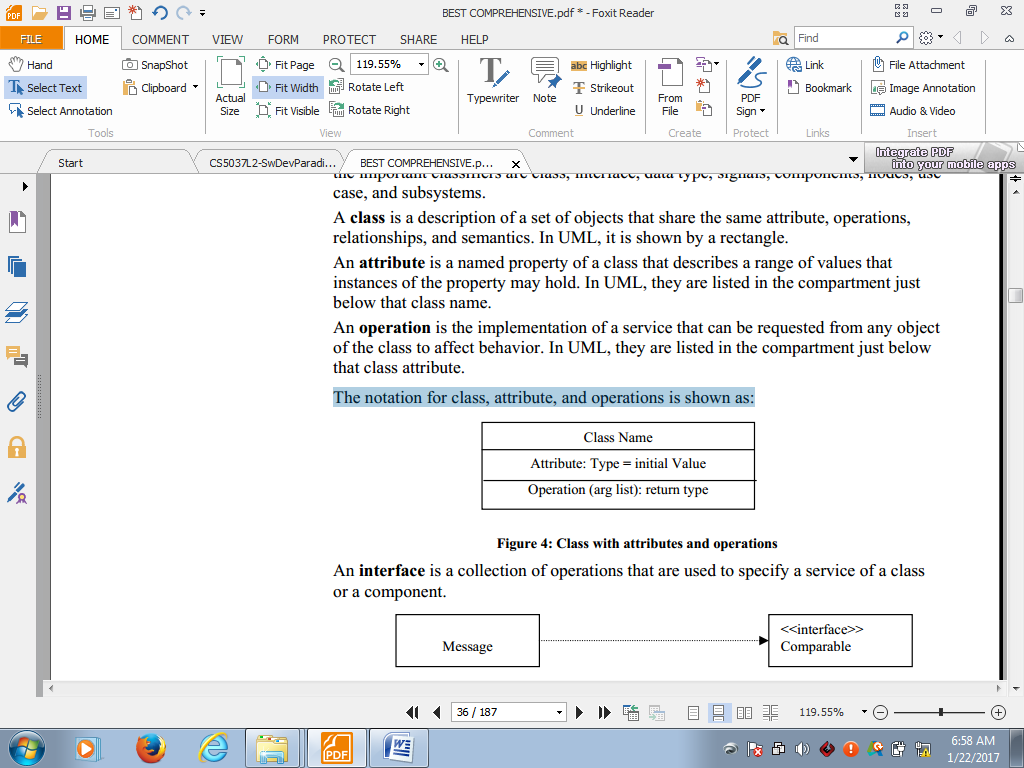
**Classes**  are the "blueprints" for objects. A class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity i.e. Objects are instances of Classes.

A **classifier** is a mechanism that describes structural and behavioral features. In UML the important classifiers are class, interface, data type, signals, components, nodes, use case, and subsystems.

A **class** is a description of a set of objects that share the same attribute, operations, relationships, and semantics.

An **attribute** is a named property of a class that describes a range of values that instances of the property(i.e.state) may hold.

An **operation** is the implementation of a service/function that can be requested from any object of the class to affect behavior. The notation for class, attribute, and operations is shown as:



### Components/Parts of a UML

"While modeling software is useful, it will not solve all design issues i.e. smart people and a good methodology are a must. But if used correctly, UML and the modeling process can help improve quality, completeness, scalability and reduce production time in many software projects.

The major building blocks of UML:

1. **Structural Notations**: These notations include static elements of a model. They are considered as nouns of the UML model which could be conceptual or physical. Their elements comprises class, interface, collaboration, use case, component, and node. It also includes actors, signals, utilities, processes, threads applications, documents, files, library, pages, etc.
2. **Behavioral Notations**: These notations include dynamic elements of a model, which comprises of interaction, and state machine. It also includes classes, collaborations, and objects.
3. **Grouping Notations:** These notations are the boxes into which a model can be decomposed. Their elements comprises of packages, frameworks, and subsystems.
4. **Annotational Notations:** These notations maybe applied to describe, illuminate, and remark about any element in the model. They are considered as explanations of the UML. Their elements comprised of notes which could be used for constraints, comments and requirements.

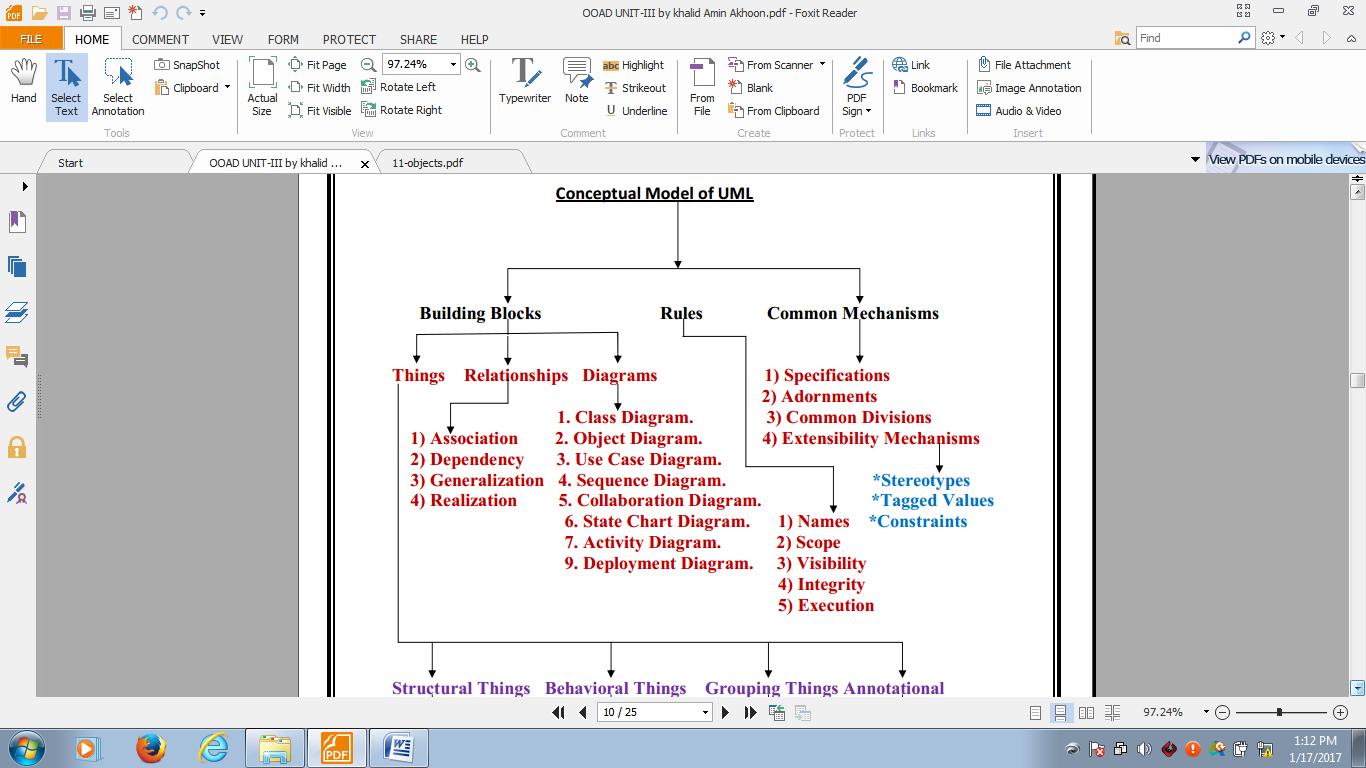
## LESSON 7

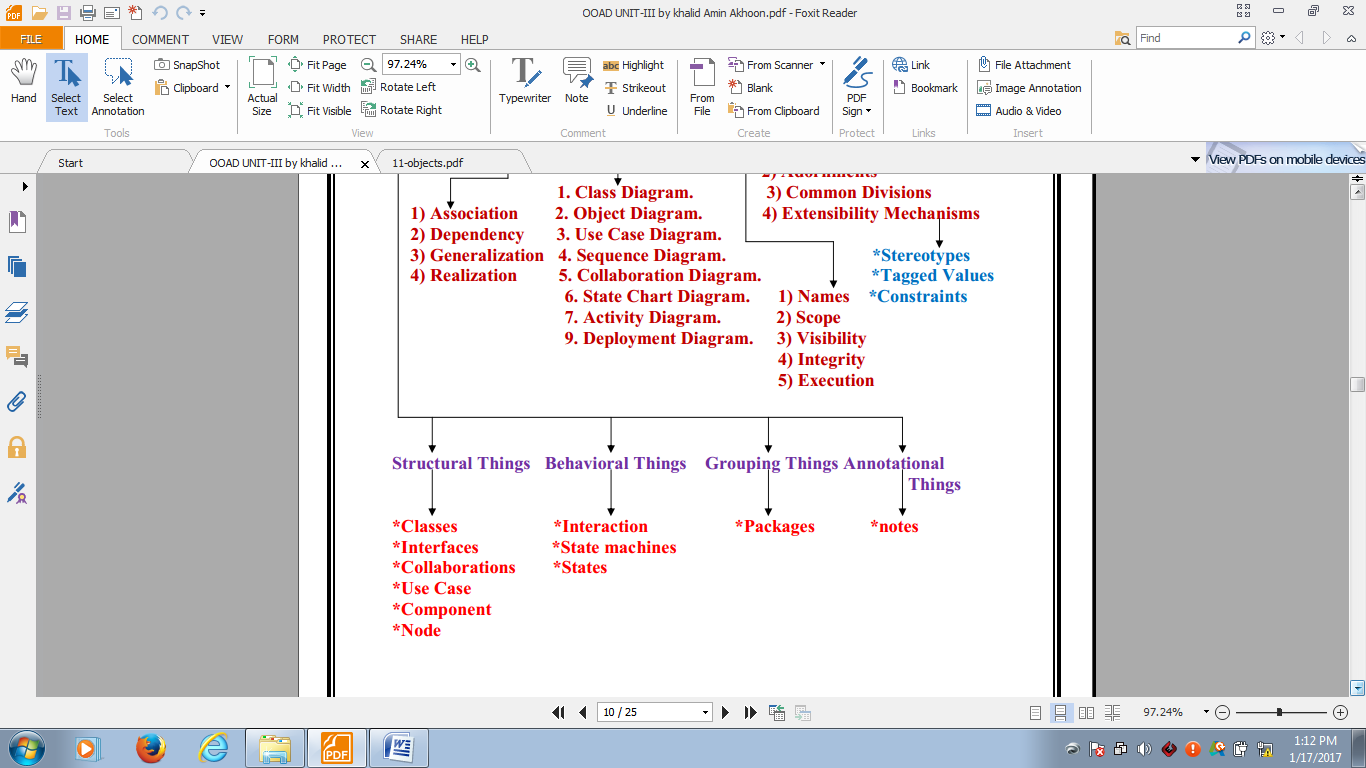
## UML MODELING

### Conceptual Meta-Model of UML

A **conceptual model** can be defined as a model which is made of concepts and their relationships.

It is the first step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other. As UML describes the real time systems, it is very important to make a conceptual model and then proceed gradually.





Conceptual model of UML can be mastered by learning the following three major elements:

* UML building blocks
* Rules to connect the building blocks
* Common mechanisms of UML

The building blocks of UML can be defined as: Things, Relationships and Diagrams

#### 1. Things

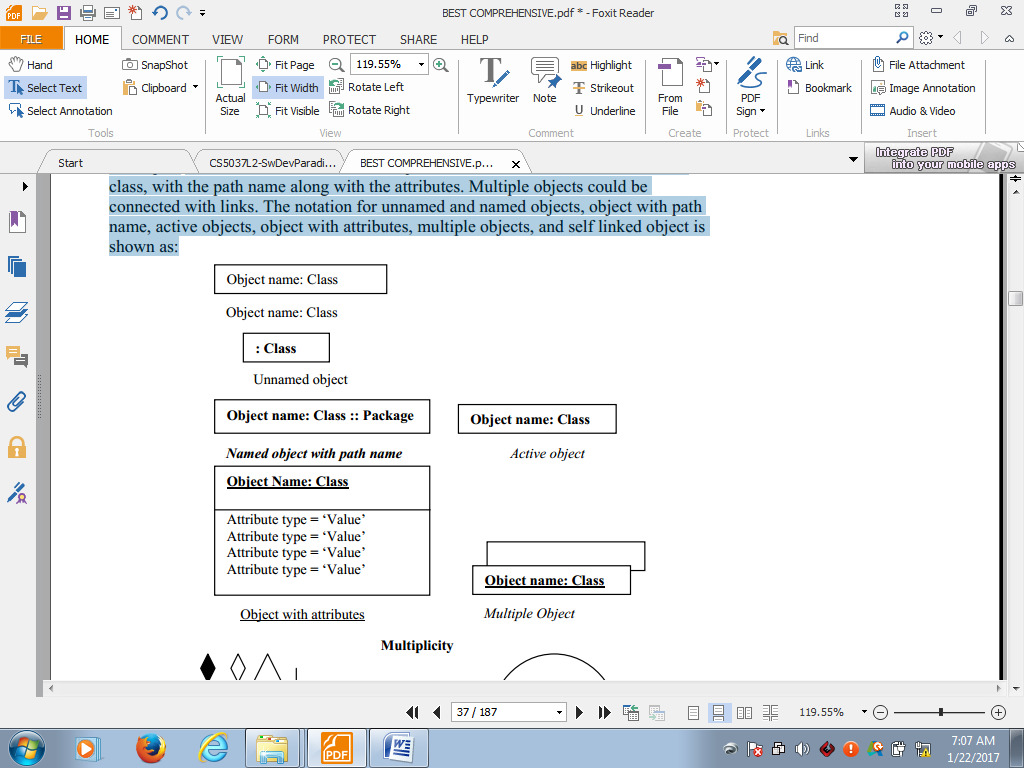
Things are the most important building blocks of UML. Things can be:

1. Structural
2. Behavioral
3. Grouping
4. Annotational

##### a. Structural things:

The Structural things define the static part of the model. They represent physical and conceptual elements. The following are the brief descriptions of the structural things:

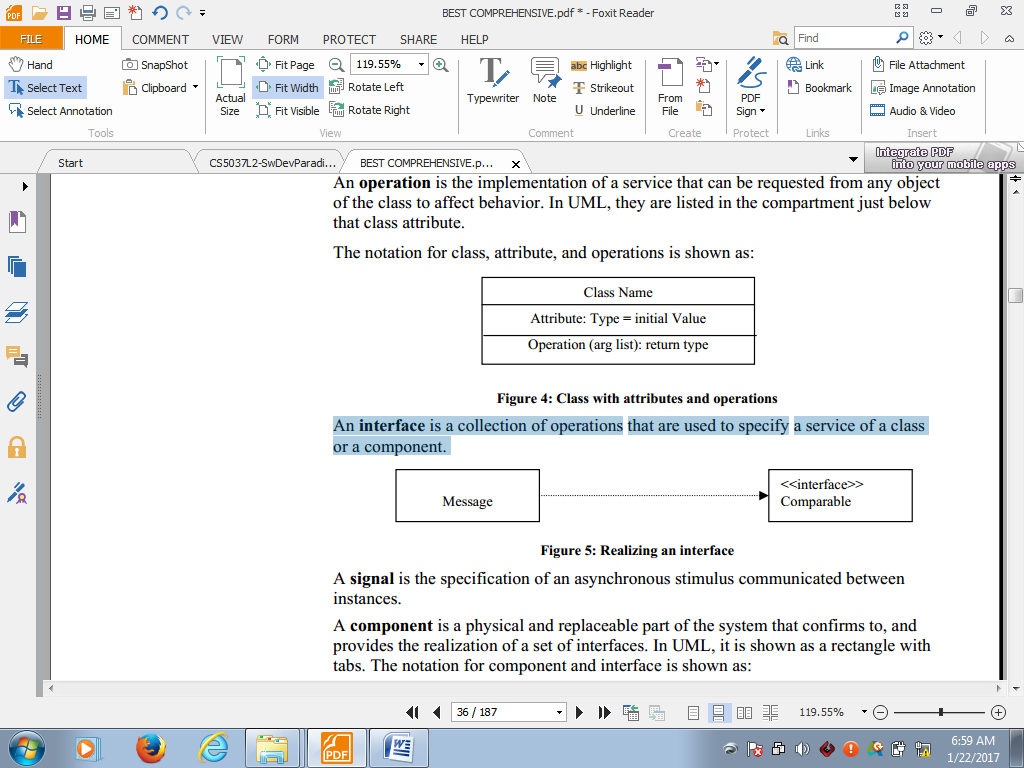
* An **object** is an instance of a class, that could be shown as the instance of a class, with the path name along with the attributes.



* **Class**: Class represents set of objects having similar responsibilities.

|  |
| --- |
| Class\_Name |
| Attributes |
| Operations |

* **Interface**: Interface defines a set of operations which specify the responsibility of a class. An **interface** is a collection of operations that are used to specify a service of a class or a component.



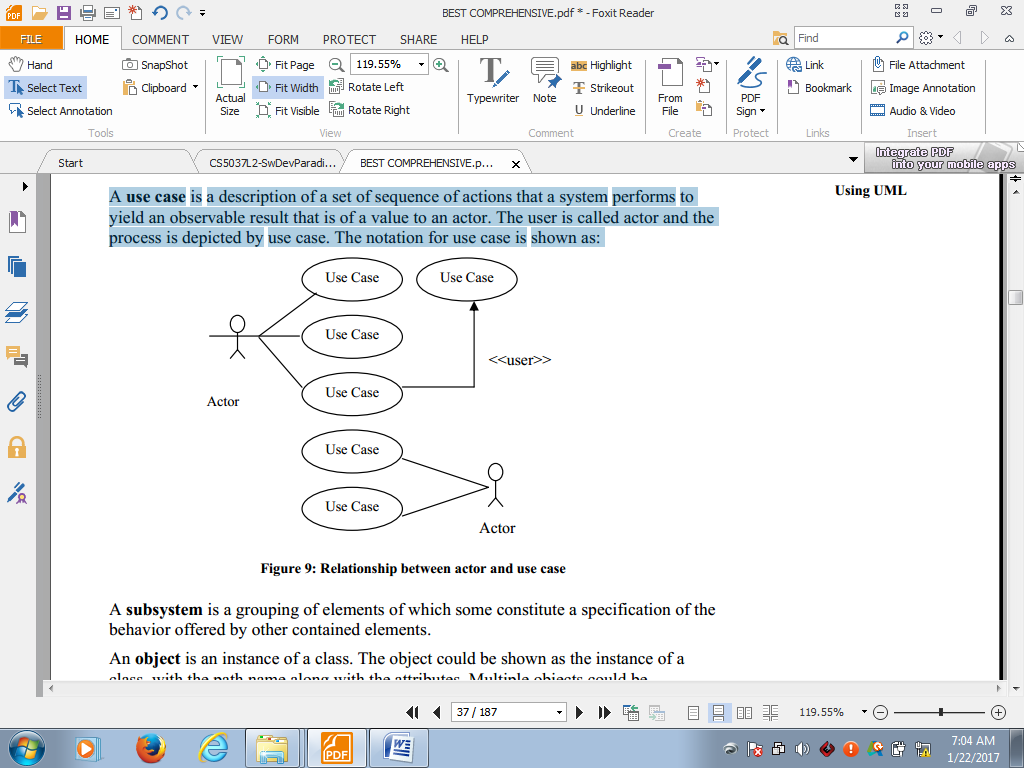
|  |
| --- |
| Interface |
|  |

A **signal** is the specification of an asynchronous stimulus communicated between instances.

**Collaboration:** Collaboration defines interaction between elements. A **collaboration** is a society of classes, interfaces and other elements that work together to provide some cooperative behavior that is bigger than the sum of its parts. i.e.



**Use case**: Use case represents a set of actions performed by a system for a specific goal. A **use case** is a description of a set of sequence of actions that a system performs to yield an observable result that is of a value to an actor. The user is called actor and the process is depicted by use case.i.e.



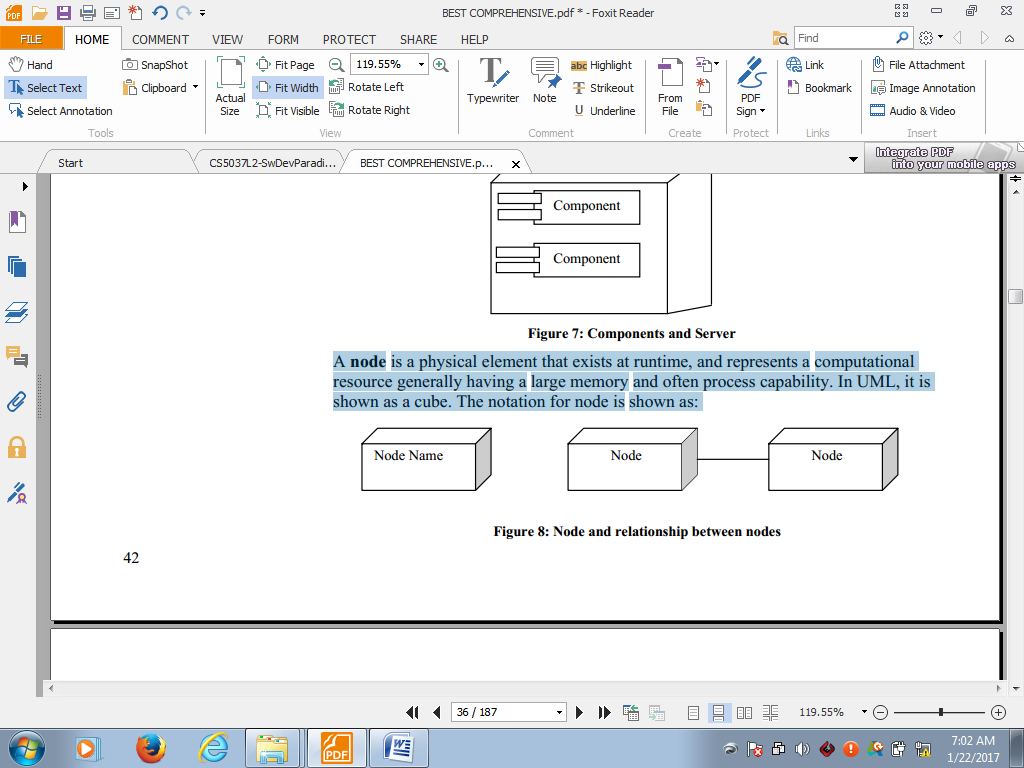
*Relationship between actor and use case*

**Component**: Component describes physical part of a system. A **component** is a physical and replaceable part of the system that confirms to, and provides the realization of a set of interfaces i.e.



**Node**: A **node** is a physical element that exists at runtime, and represents a computational resource generally having a large memory and often process capability.

Node



*Node and relationship between nodes*

##### b. Behavioral Things:

A behavioral thing consists of the dynamic parts of UML models. The following are the behavioral things:

* **Interaction**: Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

Message

Also called a **transition**, it is a relationship between two states indicating that an object in the first state will perform certain action and enter the second state when a specific event occurs and specific conditions are satisfied.

* **State machine:** State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change.

state

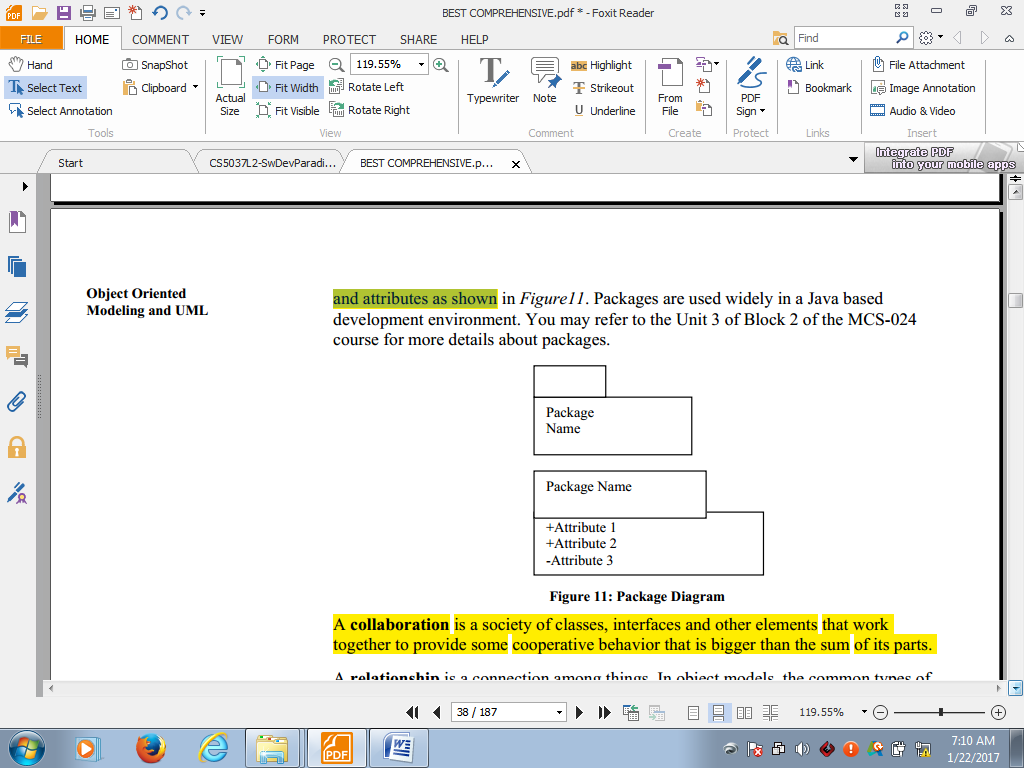
A state encompasses all the properties of the object along with the values of each of these properties.

An instance is a concrete manifestation of an abstraction to which a set of operations can be applied and which has a state that stores the effect of the operation.

##### c. Grouping Things:

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available:

**Package**: A package is a general purpose mechanism for organizing elements into groups. Package is the only one grouping thing available for gathering structural and behavioral things. It can also contain other packages. i.e.



##### d. Annotational Things:

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.

Note is the only one Annotational thing available:

* **Note**: A note is used to render comments, constraints etc of an UML element.

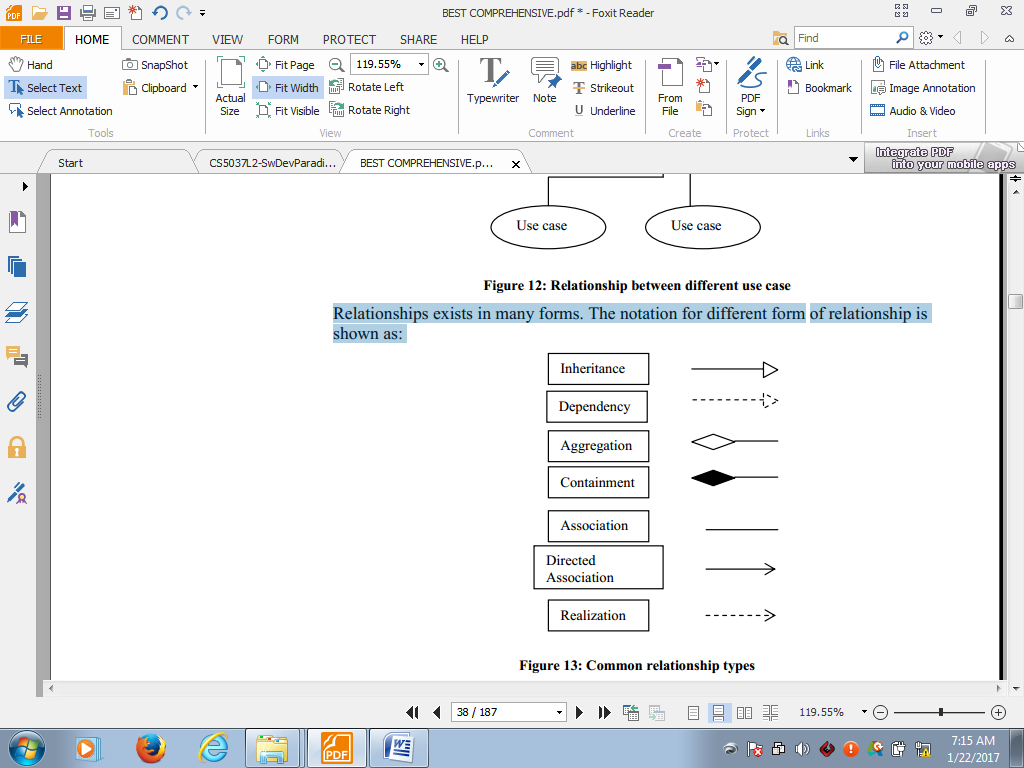
note

#### 2 Relationships

Relationship is another most important building block of UML. It shows how elements are associated with each other and this association describes the functionality of an application.

A **relationship** is a connection among things. Multiple objects could be connected with links to show relationship(s).

There are several kinds of relationships available in object models: inheritance, dependency, aggregation, containment, association, realization, and generalization. Relationships exists in many forms, the notation for different forms of relationship is shown as:

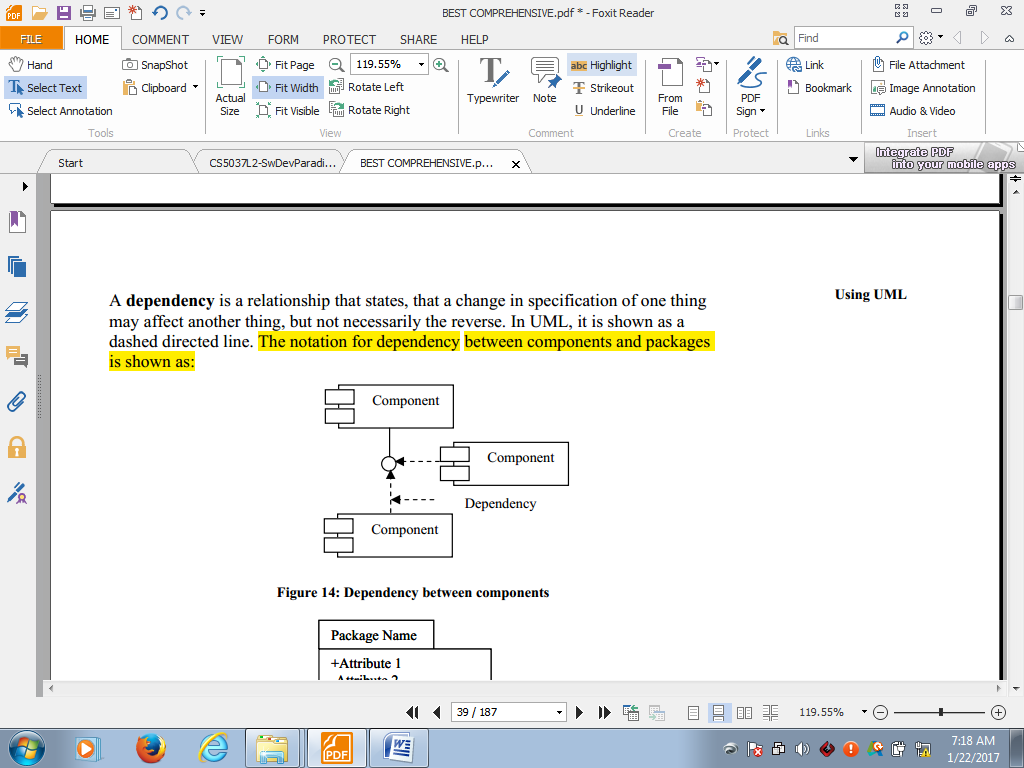
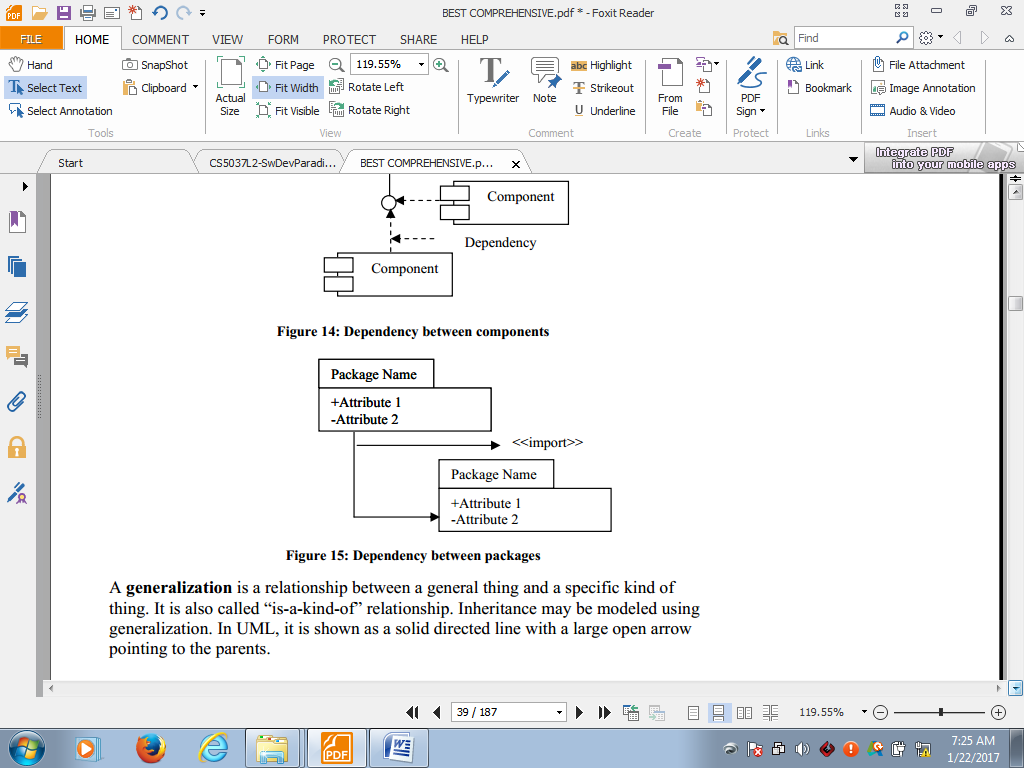
****

*common types of relationships*

**Dependency:** Dependency is a relationship between two things in which change in one element also affects the other one.

Dependency

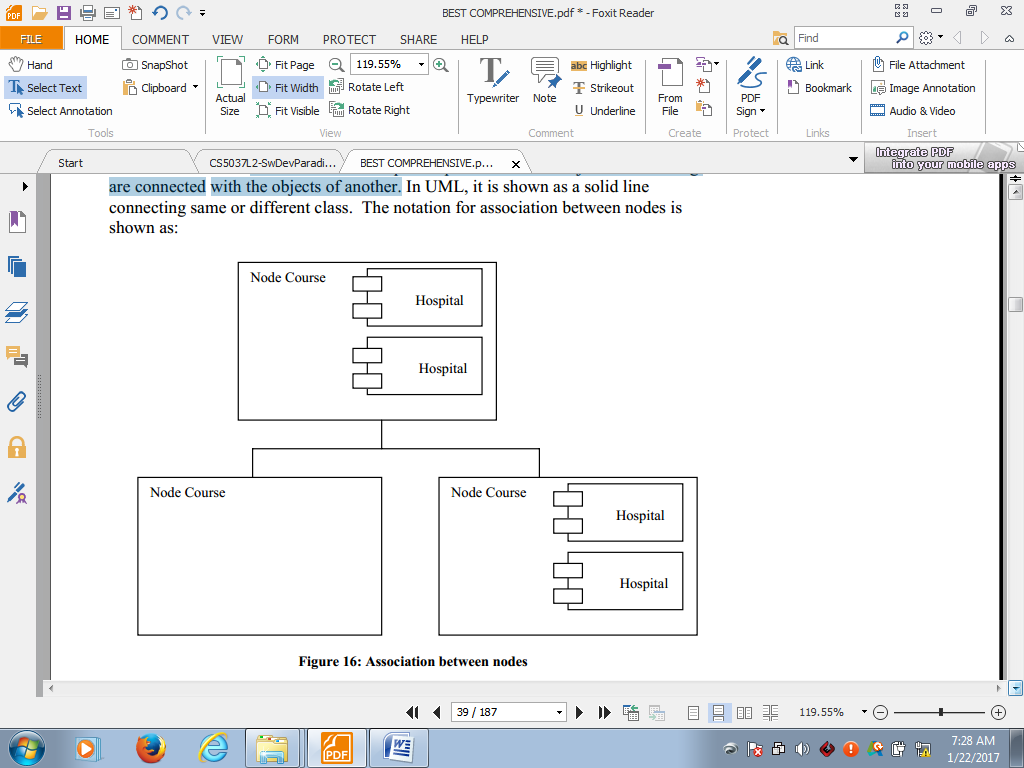
The notation for dependency between components and packages is shown as:

** **

**Association**: Association is basically a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship.

Association

It is a structural relationship that specifies that the objects of one thing are connected with the objects of another. An example of association among nodes is shown below:



The four enhancements that apply to association are name, role, multiplicity, and aggregation. Each class participating in an association has a specific role which is specified at the rear end of the association.

Multiplicity specifies how many objects maybe connected across an instance of an association which is written as a range of values (like 1..\*). The notation for roles and multiplicity between classes is shown as:

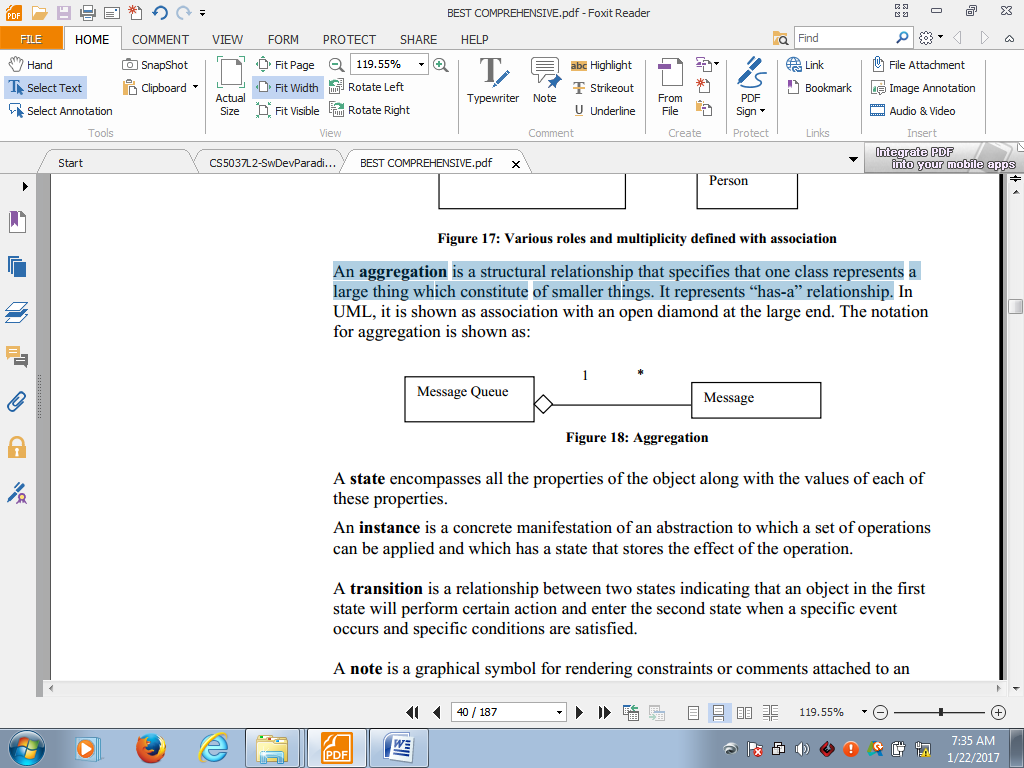


*various roles and multiplicity in association*

**Generalization/Inheritance:** Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes inheritance relationship in the world of objects.

Generalization

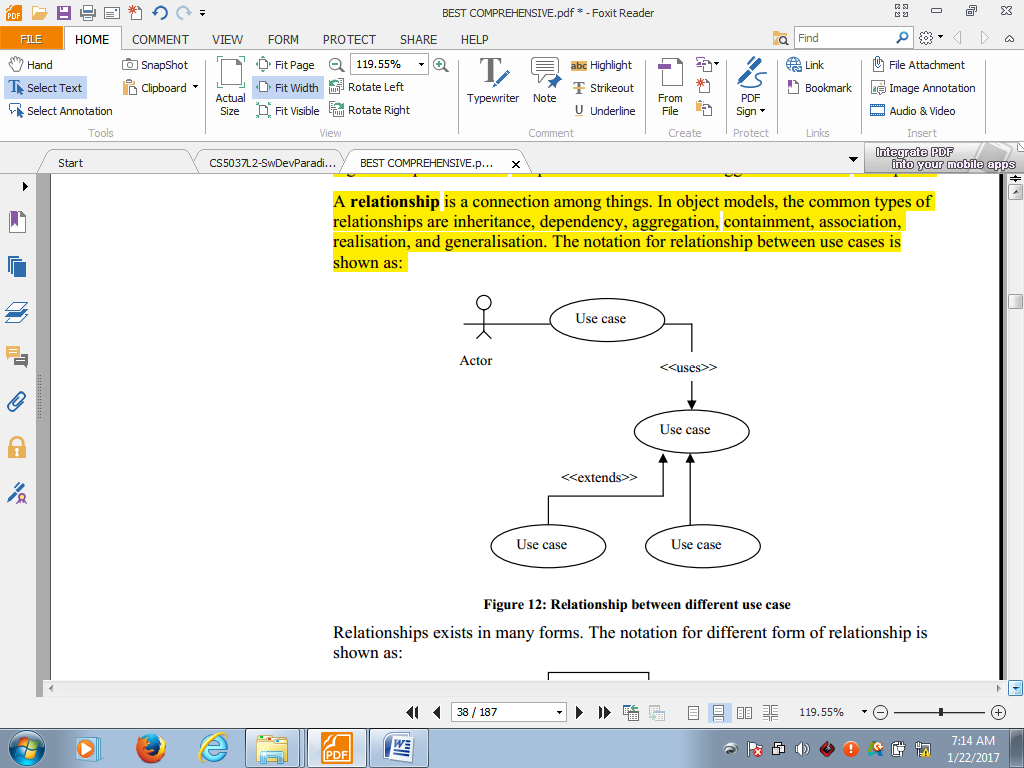
It is also called “is-a-kind-of” relationship i.e.An aggregation is a structural relationship that specifies that one class represents a large thing which constitute of smaller things.

****

**Realization**: Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility which is not implemented and the other one implements them. This relationship exists in case of interfaces.

Realization

The notation for relationship between use cases is shown as:



**Rules of the UML**

Like any language, the UML has a number of rules that specify what a well-formed model should look like.

A well-formed model is one that is semantically self-consistent and in harmony with all its related models.

The UML has syntactic and semantic rules for:

1. **Names**: What you can call things, relationships, and diagrams
2. **Scope**: The context that gives specific meaning to a name
3. **Visibility**: How those names can be seen and used by others
4. **Integrity**: How things properly and consistently relate to one another
5. **Execution:** What it means to run or simulate a dynamic model

# LESSON 8

## ......UML MODELING

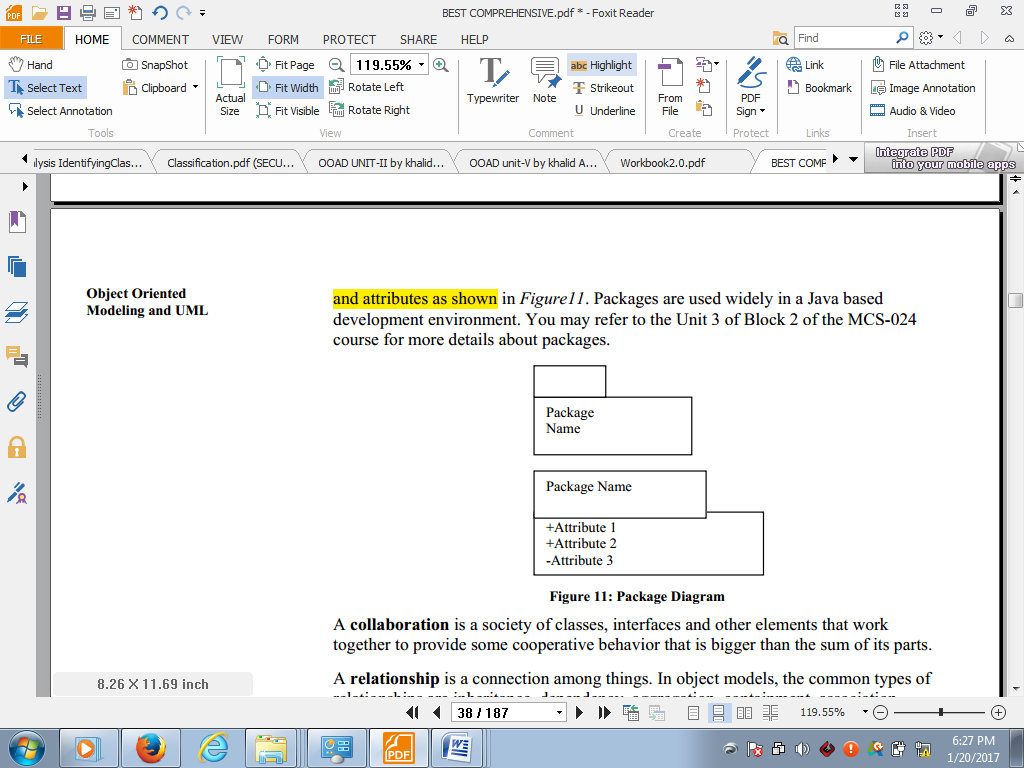
#### 3. UML Diagrams

UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system. UML includes the following nine diagrams :

**Recall:-Context Diagram**: shows the entire system in context with its environment, in a process model. It shows the overall business process as just one process (i.e., the system itself) and shows the data flows to and from external entities e.g.



1. **Package Diagram:** A package is a general purpose mechanism for organizing elements into groups. It can also contain other packages. The notation for the package shown below contains name and attributes.

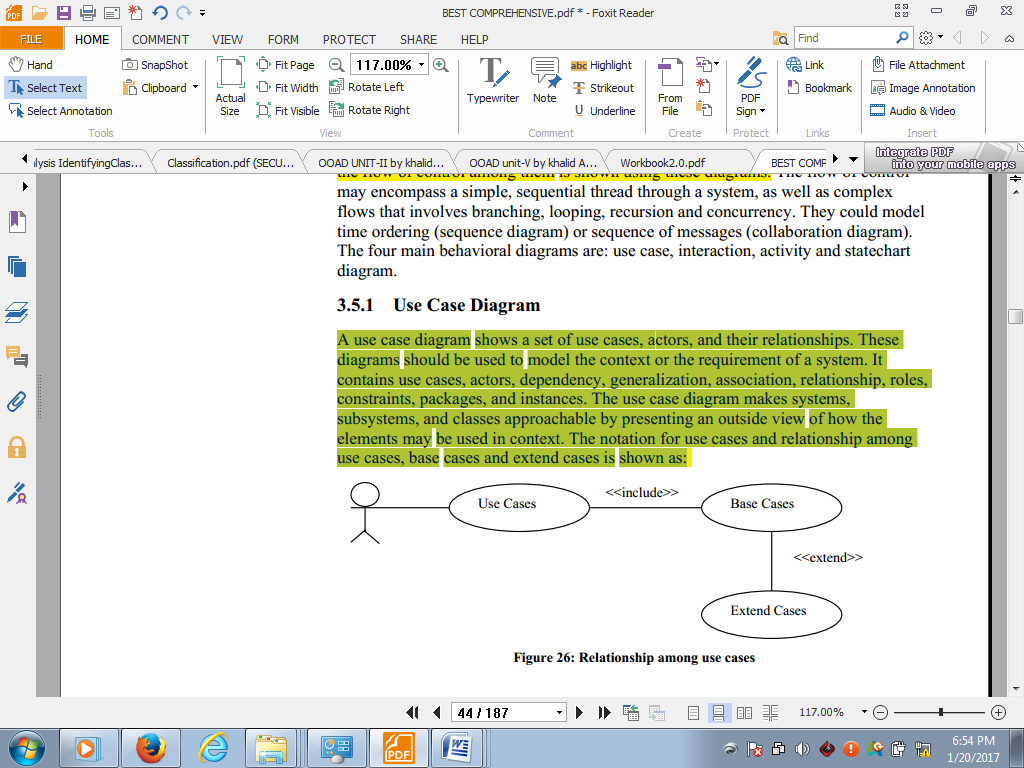


*fig: package diagram*

Behavioral diagrams are used to visualize, specify, construct, and document the dynamic aspects of a system. The interaction between objects indicating the flow of control among them is shown using these diagrams:

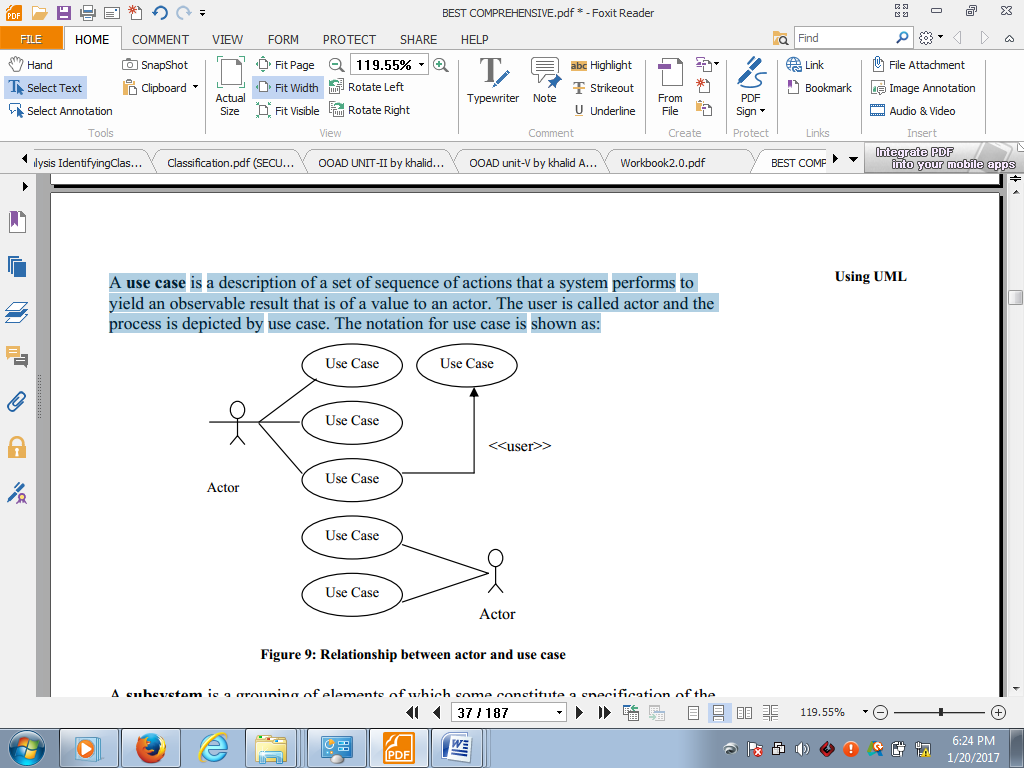
1. **Use case diagram:** A use case diagram shows a set of use cases, actors, and their relationships.

These diagrams are used to model the context or the requirement of a system. The notation for use cases and relationship among use cases, base cases and extended cases is shown as:



*Fig: relationship among usecases*

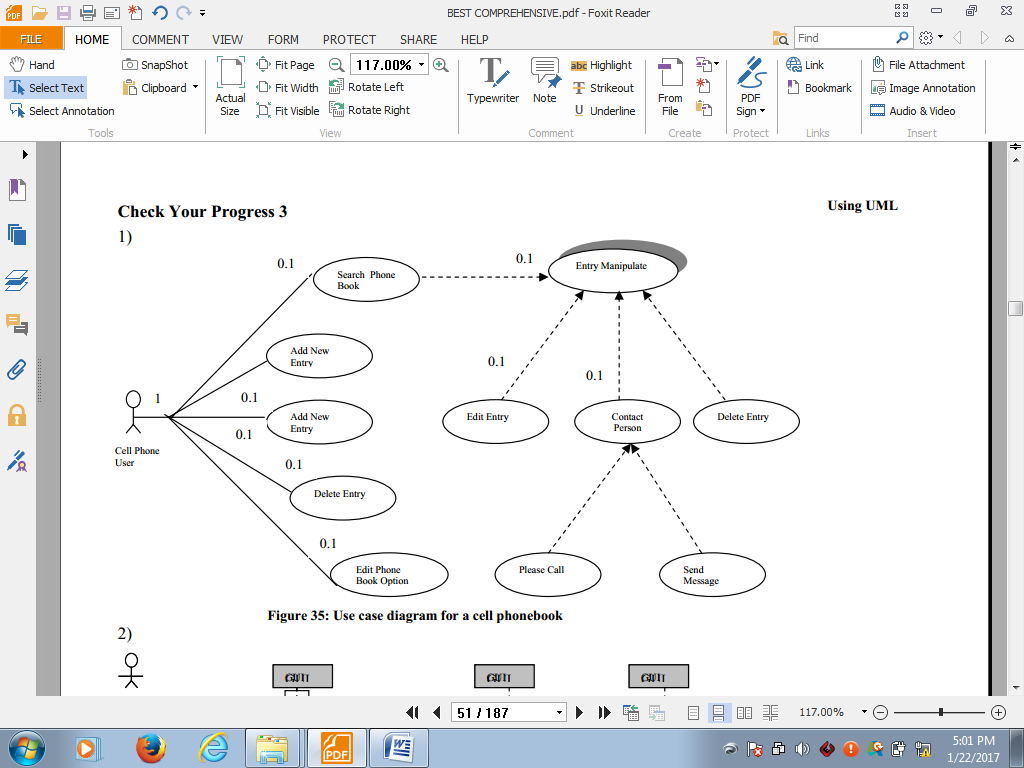
A *use case* is a description of a set of sequence of actions that a system performs to yield an observable result that is of a value to an actor. The user is called actor and the process is depicted by use case. The notation for use case is shown as:



*Fig: relationship between actor and use case*

The use case diagram makes systems, subsystems, and classes approachable by presenting an outside view of how the elements maybe used in context.

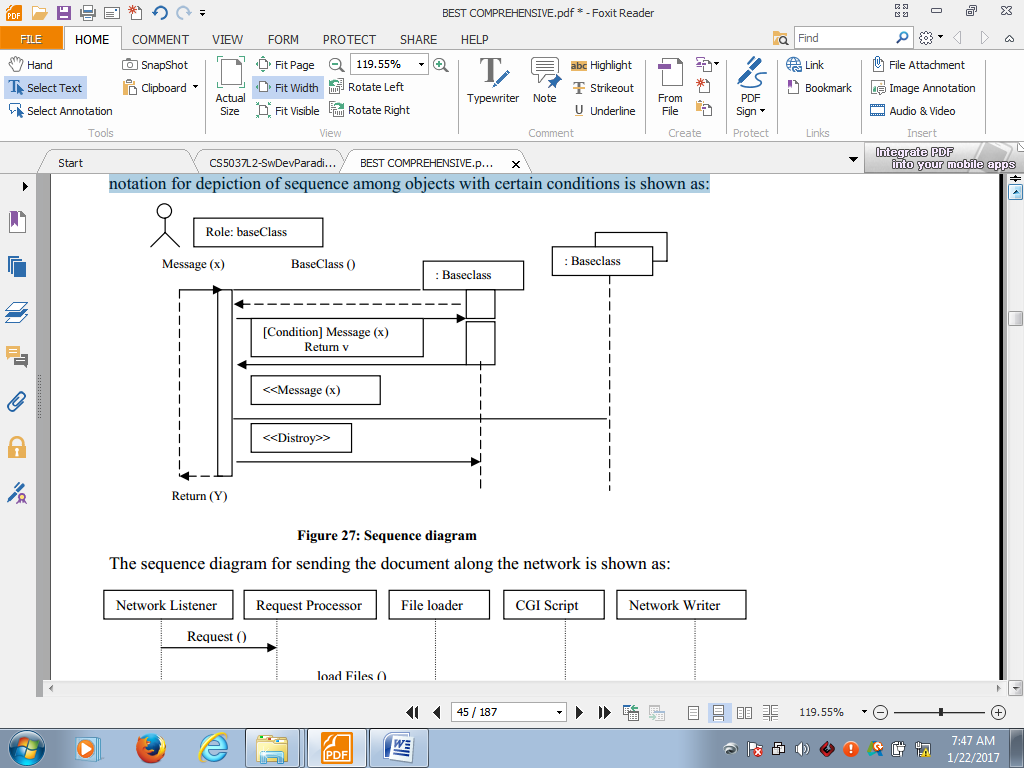
*E.g. sequence diagram for log on scenario*



*Fig: use case diagram for a cell phone book*

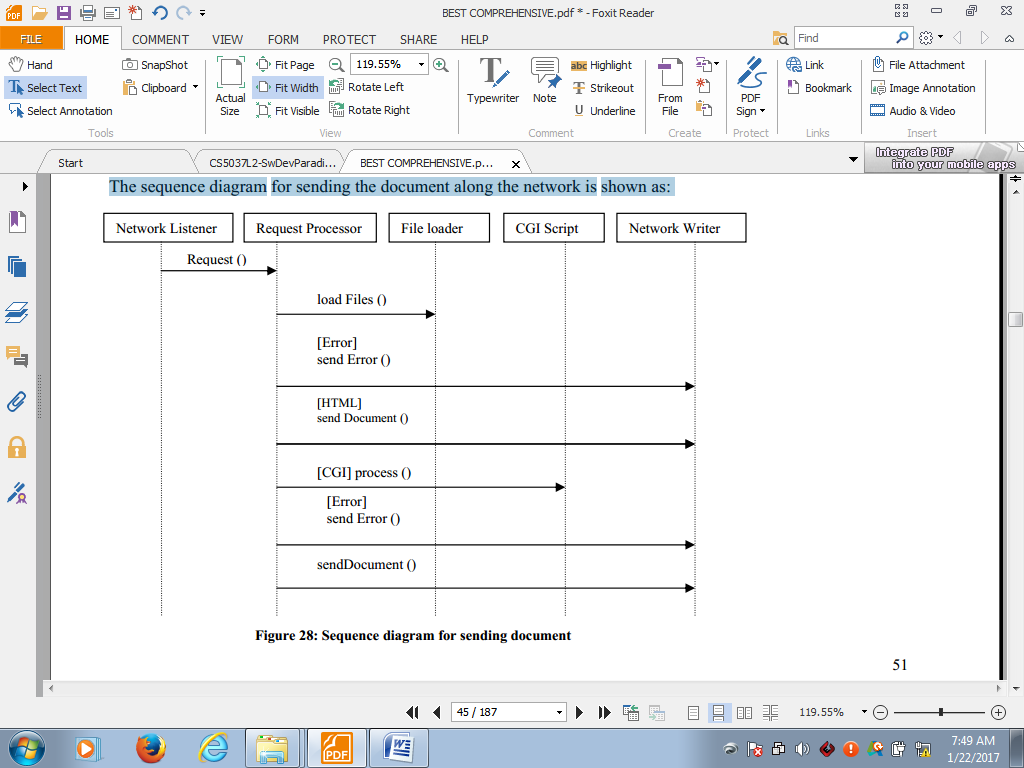
Interaction diagrams shows an interaction, consisting of a set of objects and their relationships, including the messages that maybe dispatched among them. These diagrams should be used to model the dynamic aspect of the system. It includes sequence diagrams and collaboration diagrams.

1. **Sequence diagram:** A sequence diagram is an interaction diagram that emphasize the time ordering of messages. The notation for depiction of sequence among objects with certain conditions is shown as below:



In UML it is shown as a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis. It has a global life line and the focus of control. An object life line is the vertical dashed line that represents existence of an object over a period of time. The focus of control is tall and thin rectangle that shows the period during which an object is performing an action.

*Example:* The sequence diagram for sending the document along the network is shown as:



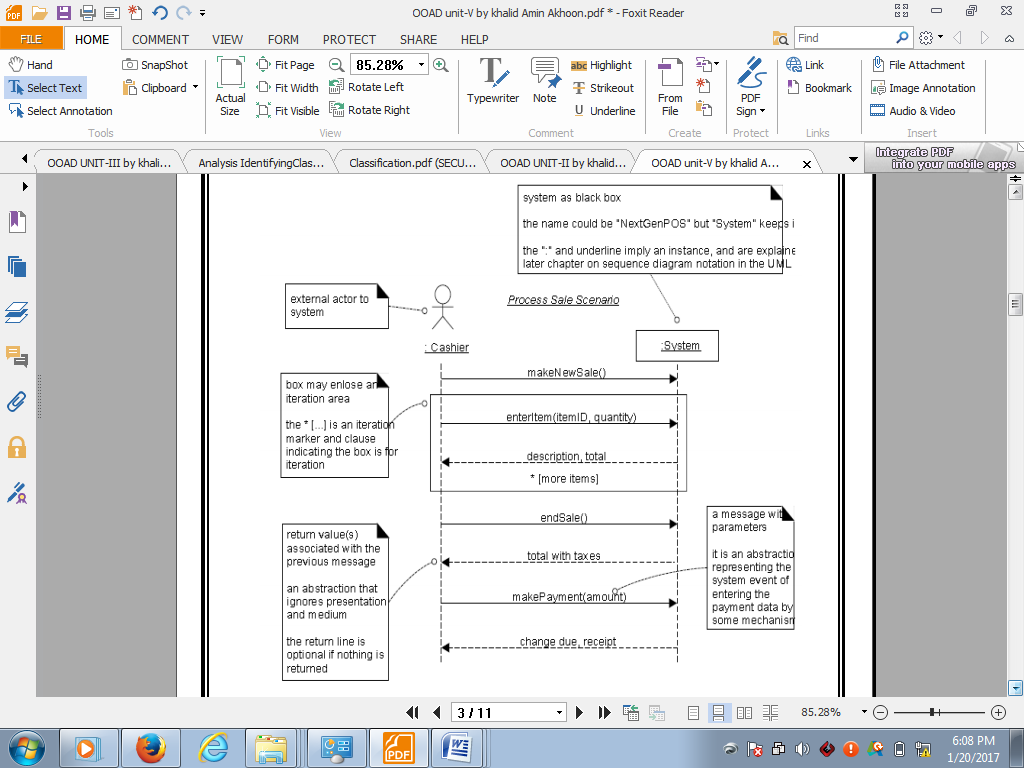
**System sequence diagrams**

Before proceeding to a logical design of how a software application will work, it is useful to investigate and define its behavior as a "black box." System behavior is a description of what a system does, without explaining how it does it. One part of that description is a system sequence diagram.

A system sequence diagram is a fast and easily created artifact that illustrates input and output events related to the systems under discussion. It illustrate events from external actors to a system. Other parts include the use cases, and system contracts.

A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and inter-system events.

*Example:* scenario of the Process Sale use case

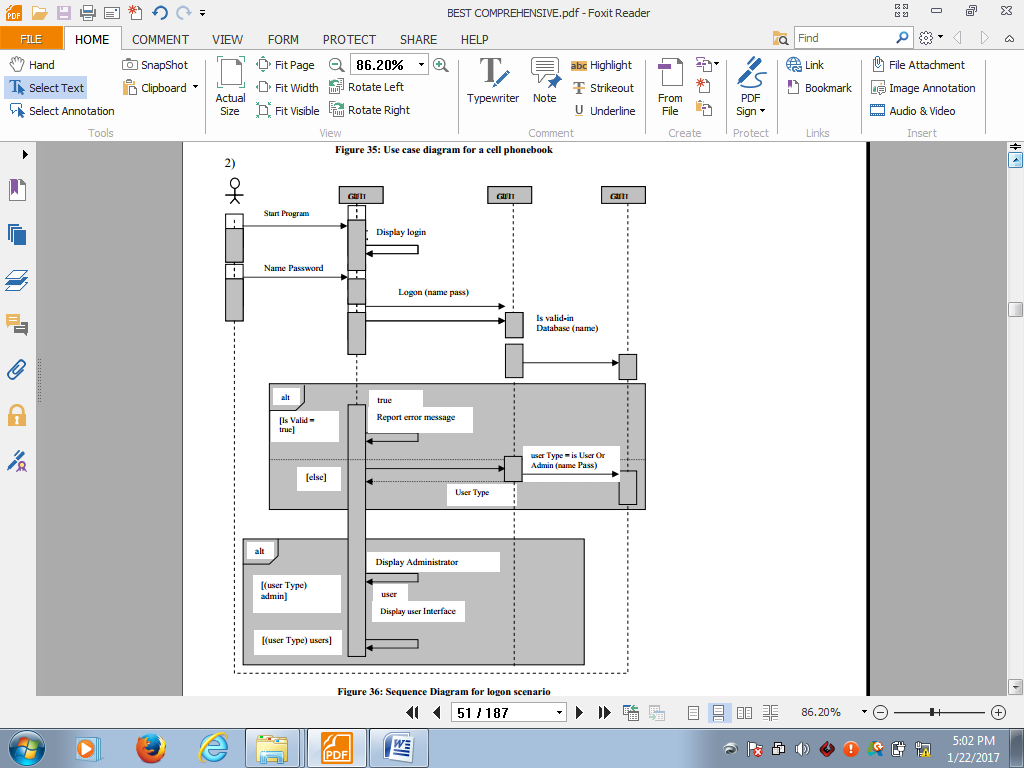


*Fig: SSD for process sale scenario*

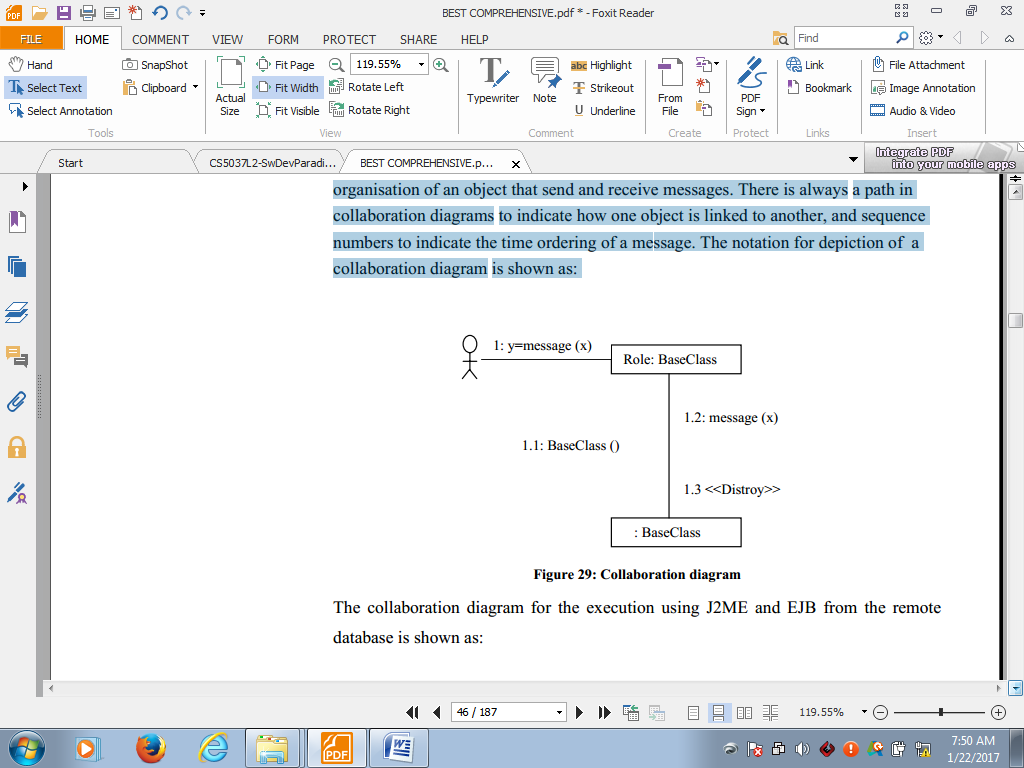
The above diagram indicates that the cashier generates makeNewSale, enteritem, endSale, and makePayment system events.

SSDs can also be used to illustrate collaborations between systems.

Other UMLs

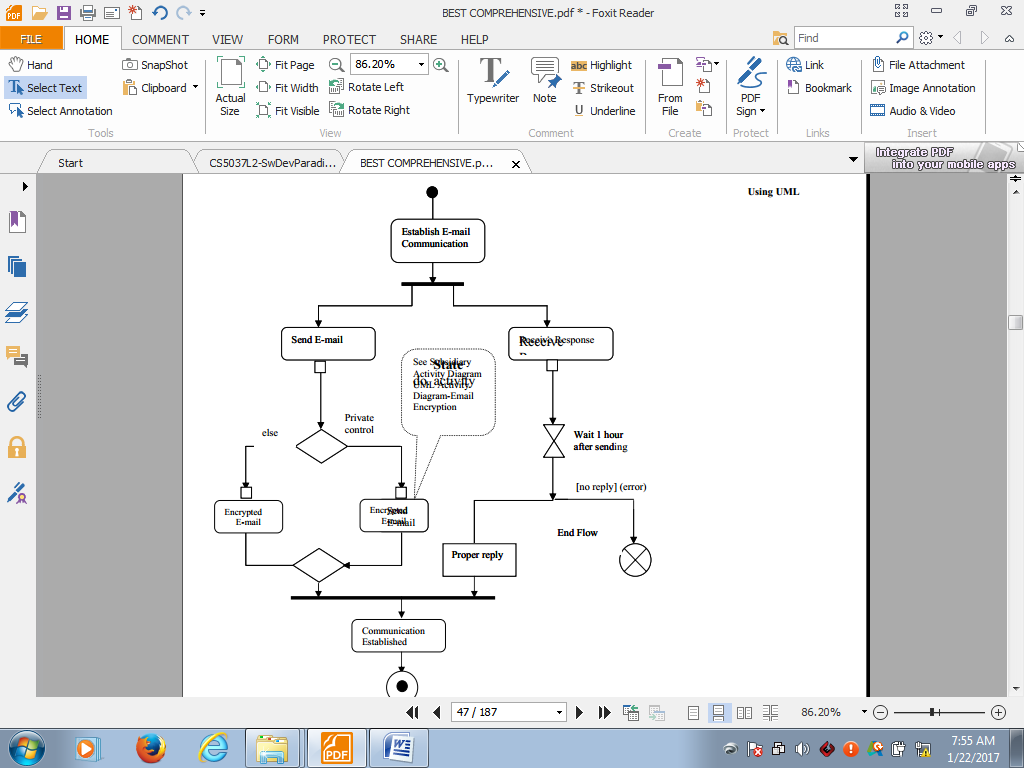


1. **Collaboration diagram:** A collaboration is a society of classes, interfaces and other elements that work together to provide some cooperative behavior that is bigger than the sum of its parts. Collaboration diagrams are interaction diagrams that emphasize the structural organization of an object that send and receive messages. There is always a path in collaboration diagrams to indicate how one object is linked to another, and sequence numbers to indicate the time ordering of a message. The notation for depiction of a collaboration diagram is shown as:



1. **Activity diagram:** Activity diagrams show the flow from one activity to another. An activity is an ongoing non atomic execution within a state machine. Activity ultimately results in some action, which is made up of executable atomic computations that result in a change in state of the system, or the return of a value. It contains activity states, action states, transition states, and objects.

*For example*: The activity diagram for encryption of a message sent through e-mail is shown as:

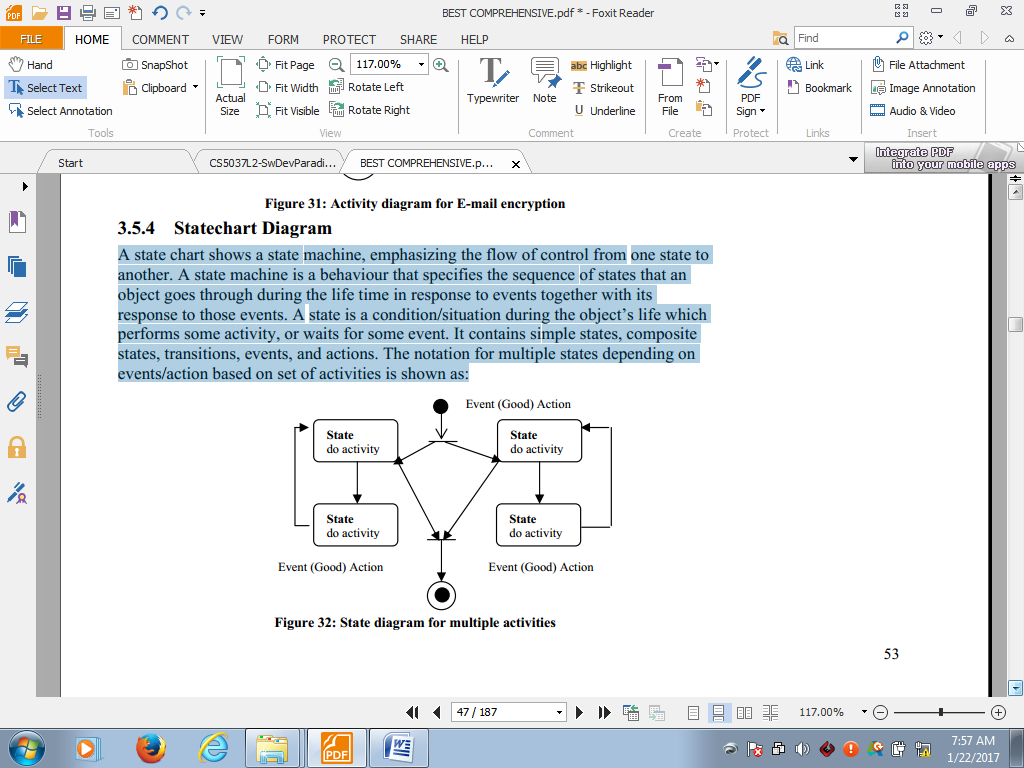


*Fig: Activity Diagram*

Communication diagrams:.......

1. **State Chart diagram:** A state chart shows a state machine, emphasizing the flow of control from one state to another. A *state machine* is a behavior that specifies the sequence of states that an object goes through during the life time in response to events together with its response to those events. A state is a condition/situation during the object’s life which performs some activity, or waits for some event. It contains simple states, composite states, transitions, events, and actions.

The notation for multiple states depending on events/action is based on set of activities is shown as:



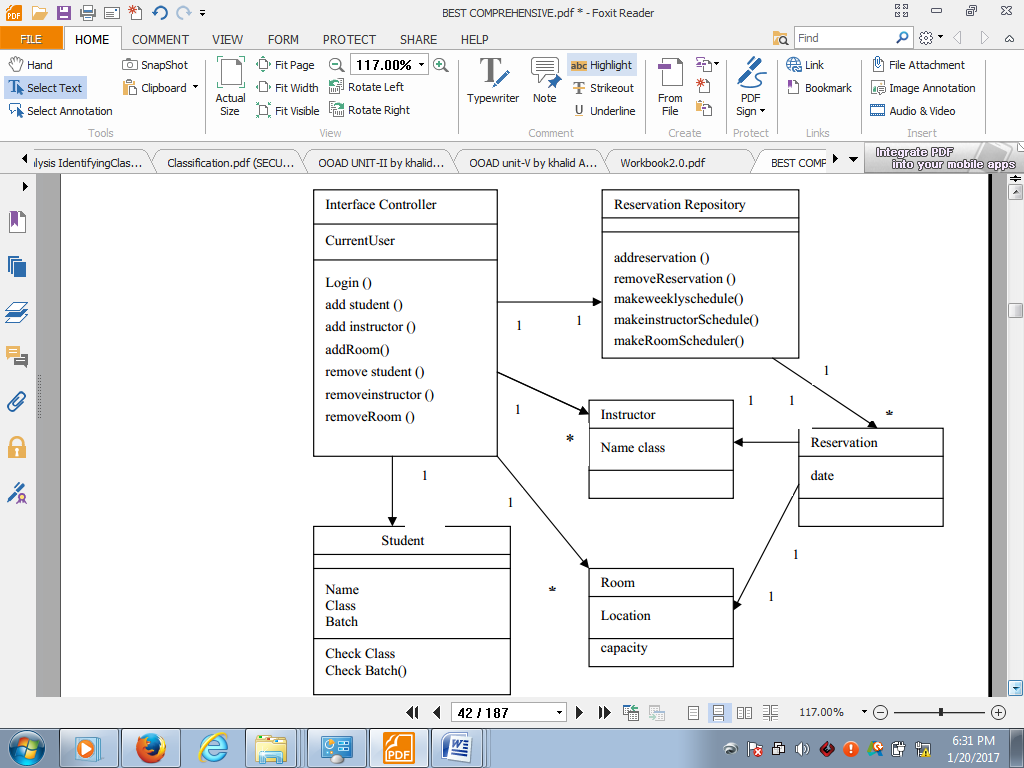
*Fig: a state diagram for multiple activities*

1. **Class diagram:** A class diagram is used to support functional requirement of system.

In a static design view, the class diagram is used to model the vocabulary of the system, simple collaboration, and logical schema. It contains sets of classes, interfaces, collaborations, dependency, generalization and association relationship.

*For example*, if in any college, there are limited classrooms that have to be allocated to different

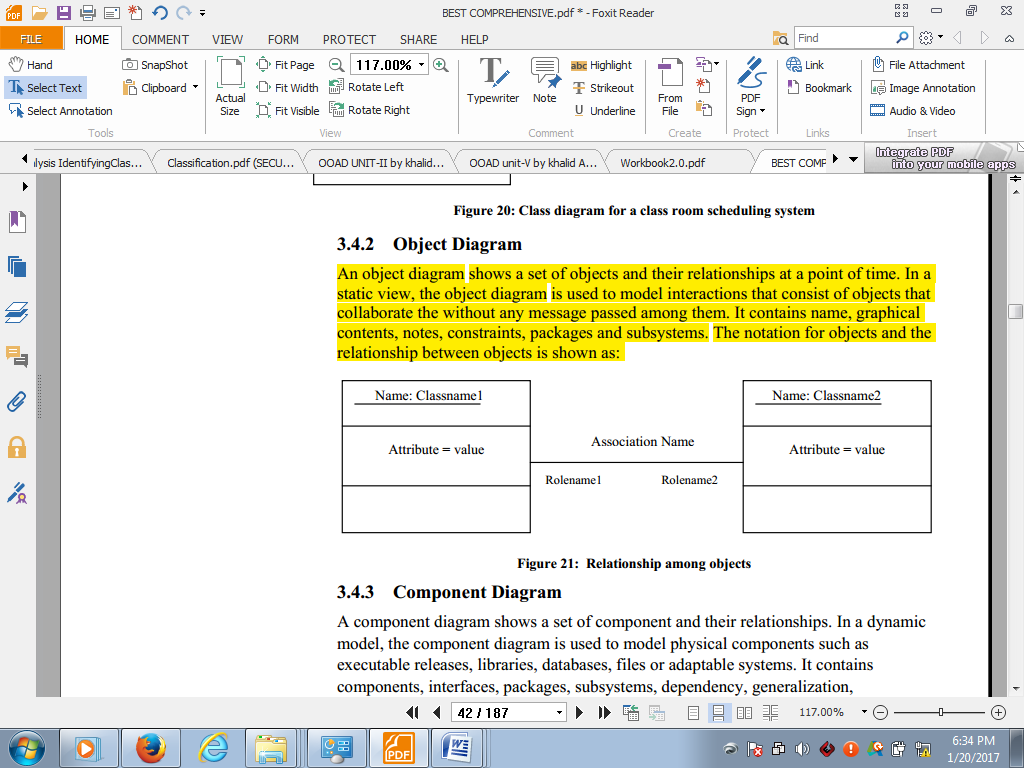
classes and instructors are fixed for all classes, then the class diagram for the allocation of classrooms and instructors is shown as:



*Fig: Class diagram*

Other diagrams which help in objects relational modeling includes:....

1. **Object diagram:** An object diagram shows a set of objects and their relationships at a point of time. In a static view, the object diagram is used to model interactions that consist of objects that collaborate without any message passed among them. It contains name, graphical contents, notes, constraints, packages and subsystems. The notation for objects and the relationship between objects is shown as:

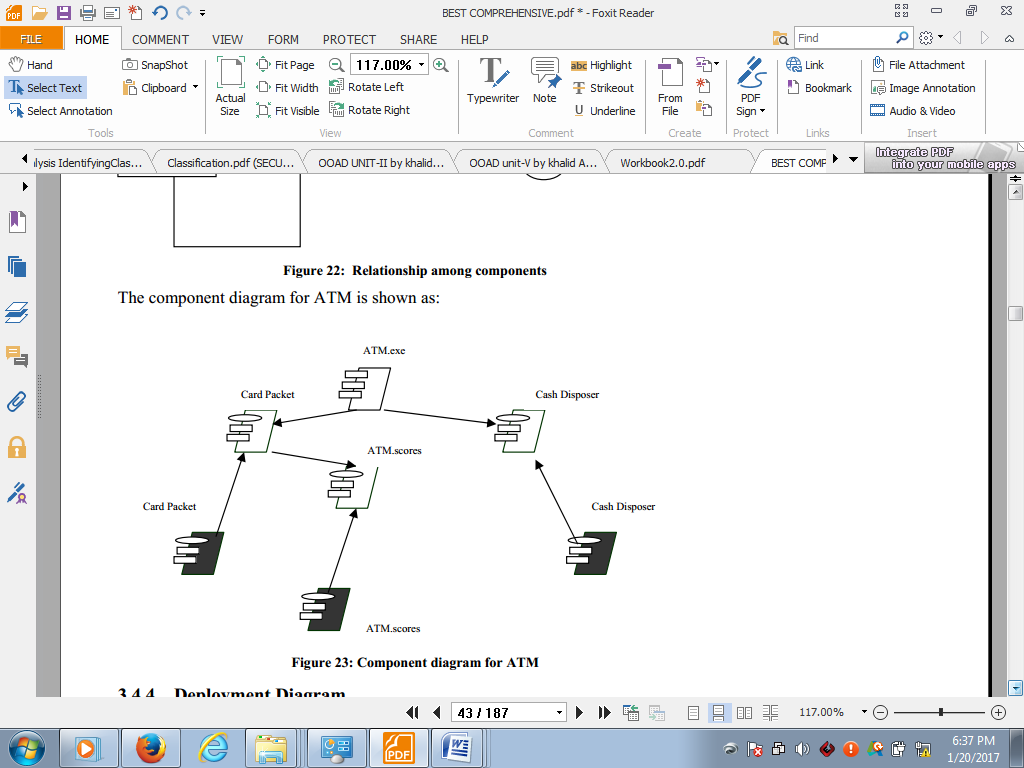


*An object diagram to show relationships among objects*

1. **Component Diagram***:* A component diagram shows a set of components and their relationships. In a dynamic model, the component diagram is used to model physical components such as executable releases, libraries, databases, files or adaptable systems. It contains components, interfaces, packages, subsystems, dependency, generalization, and association relationship. The notation for components and relationship between components is shown as:

|  |
| --- |
|  |
| *relationship among components* |

*Example: component diagram for an ATM system*

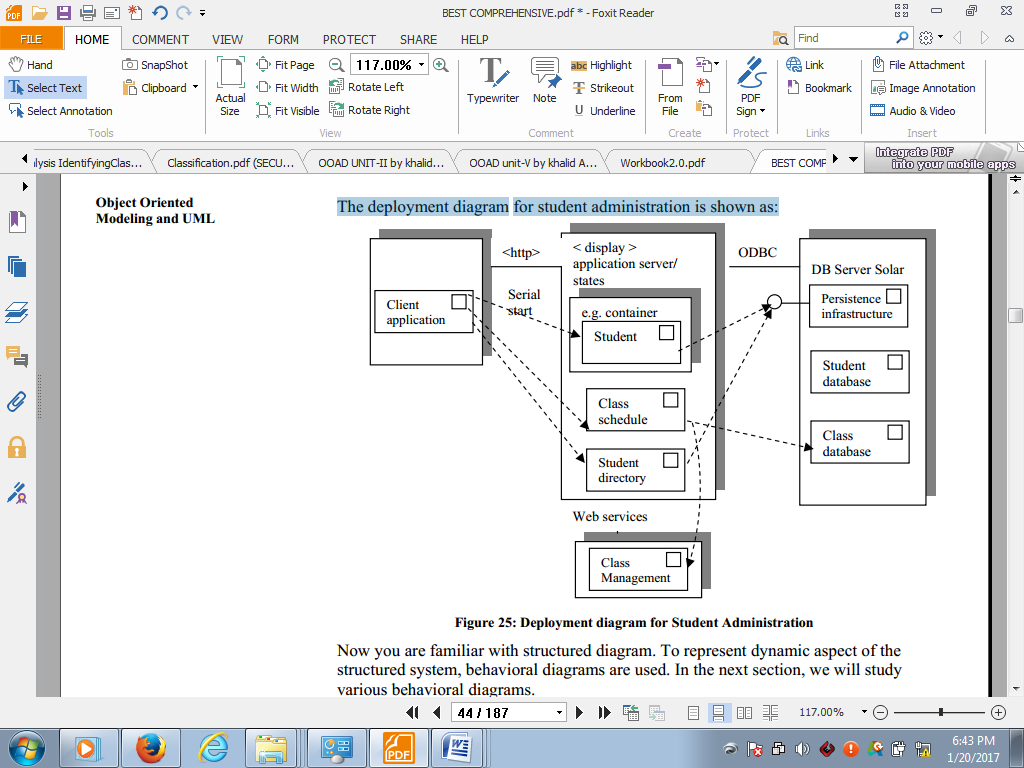


*Fig: Component diagram for an ATM system*

1. **Deployment diagram:** A deployment diagram shows all the nodes on the network, their interconnections, and processor execution. In a dynamic model, a deployment diagram is used to represent computational resources.

|  |  |
| --- | --- |
| The notation for nodes and relationship between processors and devices is shown as: |  |

*E.g.* The deployment diagram for student administration is shown as:

**

*Fig: Deployment Diagram*