0 that we're going to study in this course consists of an integrated set of diagrams of various agreed shapes and forms designed to help system and software developers to define design visualize and document software system artifacts and in some cases some business models uml is a set of best engineering practices that has proven to be effective in modeling large and complex systems and is a very important part of object-oriented software development uml mainly uses graphical notation to express the software project's design uml helps teams to communicate explore potential designs and validate software architectural designs the purpose of uml is to provide a standardized annotation that can be used by all object-oriented methods as well as selecting and integrating best elements of the precursor notations uml has been designed for a wide range of applications hence it provides constructs for a wide variety of systems and activities for example distributed systems analysis system design and deployment uml diagrams can be divided into two main types structure diagrams and behavior diagrams there are seven types of structure diagrams composite structure diagram deployment diagram package diagram profile diagram class diagram object diagram and component diagram behavior diagrams include activity diagram use case diagram state diagram sequence diagram communication diagram interaction overview diagram and timing diagram now a few words about each one of them class diagram is this central modeling technique that is used in almost all object oriented methods this diagram describes the types of objects in the system and different kinds of static relationships that exist between them the three most important types of relationships in class diagrams but there are actually more of them are association which represents relationships between instances of types for example a person works for a company or a company has multiple offices inheritance which corresponds directly to inheritance in object oriented design and aggregation which is a form of object composition in object oriented design component diagrams illustrate how components are put together to form larger components or software systems and illustrates the architecture of software components and dependencies between them these software components can include runtime components executable components and source code components deployment diagram helps to model the physical aspect of an object-oriented software system this is a block diagram that shows the system architecture as deployment or distribution of software artifacts artifacts represent specific elements in the physical world that are the result of the development process the diagram simulates the runtime configuration in a static view and visualize the distribution of artifacts in the application in most cases this involves simulating hardware configurations along with the software components that host them object diagram is an instance of the class diagram it shows a detailed snapshot of the system state at a particular point in time the difference is that the class diagram is an abstract model of classes and their relationships however the object diagram represents an instance at a specific moment which is concrete in nature the use of object diagrams is rather limited namely to show examples of data structures package diagram is a structural diagram that shows packages and dependencies between them it allows us to display different views of the system for example it is easy to simulate a layered application composite structure diagram is similar to the class diagram and it is a kind of a component diagram used mainly in micro level system modeling but it depicts individual parts instead of whole classes this is a type of static structure diagram that shows the internal structure of a class and interactions that the structure makes possible this diagram can include internal parts ports through which the parts can communicate with each other or through which the class instances can communicate with various parts and with the outside world as well as connectors between parts and ports composite structure diagram is a set of interrelated elements that interact at runtime to achieve a specific goal each element has a dedicated role in this collaboration profile diagram allows us to create domain and platform specific stereotypes and define relationships between them we can create stereotypes by drawing shapes of stereotypes and linking them to composition or generalization through a resource oriented interface we can also define and visualize stereotype values use case diagram describes the functional requirements of a system in terms of use cases in essence it is a model of intended functionality of the system use cases and its environment actors use cases allow us to link what we need from the system with how the system meets those needs activity diagrams are graphical representations of workflows of

stepwise activities and actions with the support of choice iteration and concurrency they describe the control flow of the target system such as exploring complex business rules and operations as well as describing the use cases and business processes in uml activity diagrams are meant for modeling both computational and organizational processes state machine diagram is a type of diagram used in uml to describe system behavior which is based on david harrell's concept of state diagrams state machine diagrams depict permitted states and transitions as well as events that affect those transitions this diagram helps to visualize the entire life cycle of objects and thus helps to better understand state-based systems sequence diagram models the interaction of object based on a time sequence it shows how objects interact with each other in a particular use case similar to sequence diagram the communication diagram is also used to model the dynamic behavior of a use case when compared to sequence diagram communication diagram focuses more on showing object collaborations rather than the time sequence in fact communication diagrams and sequence diagrams are semantically equivalent and one can be generated from another interaction overview diagram focuses on the overview of the flow of control of the interactions this is a variant of activity diagram where the nodes depicted are the interactions or interaction occurrences interaction overview diagram describes interactions in which messages and lifelines are hidden we can link up the real diagrams and achieve a high degree of navigation between diagrams within an interaction of your diagram timing diagram shows the behavior of the object or objects at a given time period in fact this is a special form of a sequence diagram and the differences between them are that the axes are swapped so that the time increases from left to right and the lifelines are displayed in separate compartments arranged vertically so why are there so many diagrams the reason for this is that this can help to examine the system at different angles and there can be many participants involved in software development for example analysts architectures coders testers quality control customers technical authors these people are all interested in different aspects of the system and each requires a different level of details for example a coder would want to understand the design of the system to be able later to convert this design into some low-level code in contrast a technical writer is interested in the behavior of the system as a whole and would want to understand the product's features uml tries to be a language that all those participants can benefit from and possibly use at least one of its diagrams uml class diagram illustrates the structure of the system by describing classes their attributes methods and relationships between them just before we begin discussing the diagram let's remind ourselves what is a class in a nutshell class is a template for creating objects as well as representing objects initial state attributes and behavior methods each attribute has its own type each method has its own signature but in the class diagram only the class name is mandatory and it makes sense because even the best psychics will not be able to guess what this nameless square supposed to mean or what it generally refers to so let's take a look at the building blocks of a class diagram starting with the class itself the name of the class is written in the uppermost partition followed by the class attributes with their types written after the column and finally the last partition contains methods the type that the method can return is written after the column in the method signature you've probably noticed the minuses and pluses before the class attributes and methods these are the access modifiers or class visibility notation plus means public minus private hash protected and tilde means package local each parameter in a method can also be described in terms of its direction with the respect to the caller in out in out for example method 1 uses p1 as an input parameter and the value of p1 is somehow used by the method and the method does not change p1 method 2 accepts p2 as an input output parameter the p2 value is somehow used by the method and it accepts the output value of the method and the method itself can change p2 method 3 uses p3 as the output parameter in other words the parameter serves as a repository for the output value of the method we can use class diagrams at different stages of the software development lifecycle therefore we might want them to reflect a different levels of specification and thankfully there are three different perspectives or levels of specification that we can use a conceptual perspective is when class diagrams are interpreted as describing entities of the real world thus if we choose conceptual perspective we construct a diagram that represents the concepts in the domain these concepts relate to classes that

implement them the conceptual perspective is considered language independent a specification perspective is when diagrams are interpreted as describing abstractions of software or components with specification and interfaces but without any reference to a specific implementation thus if you look at the specification perspective you can generally see some software interfaces but not necessarily their implementation and finally the implementation perspective is when diagrams are interpreted as a description of software implementations in a particular technology or programming language thus if you choose to use this perspective you can choose to depict the actual software implementation now let's have a look at the relationships between classes i will describe the six main types of notation that are most common these are association inheritance implementation dependency aggregation and composition similar to relationships connecting objects associations connect classes in order for there to be a connection between classes there must be an association between them if we assume that we have two classes that interact with each other a continuous connecting line should be drawn between them indicating the association in the diagram often we can also see a verb written above the line that conveys its meaning in addition we can also specify the multiplicity that is the number of objects that can take part in the relationship multiplicity is specified as a comma separated list of intervals where each interval is represented as a minimum maximum for example one student can learn from many tutors and a tutor can teach many students inheritance sometimes it is also called generalization as the name implies this is a schematic representation of the relationship between the parent class and its descendants the hollow arrow is always directed towards the parent class a classic example of inheritance depicts a square class a rectangle class and a circle class as descendants of their common parent class we can connect each class with their parent class separately or combine those lines and draw one common arrow line that connects all common descendants with their parent if the sentence inherits from an abstract class then the name of such parent class is written in italics realization usually this refers to the relationships between an interface and objects that implement this interface for example the owner interface has methods for buying and selling private property and the relationships between the owner interface and the person and corporation classes that implement this interface are depicted by a dashed line with an arrowhead pointing at the interface dependency when an object one class uses an object of another class in its method and this object is not stored in any field then this kind of relationship is modeled as a dependency dependency is essentially a special case of the association of two classes in this case changes in one class will inexorably entail changes in the other for example this person class has a has read method with a book as an input parameter which returns true if for example the person has read the book dependency is drawn with a dashed line with arrowhead pointing towards the class on which the method of another class could depend aggregation a special type of relationship between classes when one class is a composite part of another for example a programmer's workstation could consist of a chair a desk a computer and a fan but when we delete the workstation class we still have all these classes but on their own they no longer form an aggregate class workstation aggregation is shown as a continuous align with an unfilled diamond connected to the class which represents the aggregate composition is a type of aggregation but this time classes that form the aggregator class are destroyed when the aggregator class is destroyed for example our body is made up of organs but they can't function by themselves composition is depicted similarly to aggregation but this time the diamond is solid and as a conclusion to this topic i just want to remind you that uml is not so much about a pretty picture uml is actually a powerful tool in the programmers toolkit if you can understand and utilize it properly say get in the habit of drawing your class diagram before you sit down and actually code your next e-commerce gaming financial and whatnot project time invested in drawing assets synth and clear uml diagram will eventually pay off and can save your project from design flows and other errors you might encounter without a proper plan in front of you uml component diagrams are used for modeling physical aspects of object oriented systems that are used for visualizing defining and documenting component-based systems as well as construction executable systems using forward and reverse engineering component diagrams are essentially class diagrams that focus on system components that are often used to model the static

implementation view of a system component diagram breaks down the system under development into various high levels of functionality each component is responsible for one clear goal within the entire system and interacts with other essential elements only on a need to know basis so how does it work data for example account and inspection id flows into the component through the port on the right and gets converted to a format that internal components can utilize interfaces on the right are known as required interfaces they represent the services the component needs to fulfill its responsibilities data then passes through various connections and other components before it is output through the ports on the left those interfaces on the left are also known as provided interfaces and other components of the system can interact with them in order to take advantage of the output results of the component it is important to note that the internal components are surrounded by a large box which can represent the system as a whole in that case there will be no component symbol in the upper right corner or a subsystem or component of the overall system in this case the box itself is a component a component is a modular part of a system that encapsulates its content in uml 2.