




















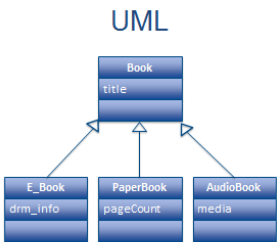
1-1	 <p>Object-Oriented Analysis and Design (Part 1)</p> 	<p>We will be covering Object-Oriented Analysis and Design for the next two modules. Object oriented analysis and design is currently the most accepted method for doing analysis and design of software. It supersedes older techniques such as structured analysis and design or data driven approaches.</p>
2	 <p>Overview</p> 	<p>In this module we will cover the basics of object-oriented analysis and design. The methods are based around the notion of simulating the problem in terms of objects.</p> <p>Objects correspond to the things in the problem.</p> <p>The behavior and information inherent in the problem are modeled by the objects and interactions between objects.</p> <p>The notation used to convey the structure and dynamics of the object-oriented solution is UML – the Unified Modeling Language.</p> <p>We will cover the basics of UML so you will be able to apply it to do your modeling.</p>
3	 <p>Objectives</p> <ul style="list-style-type: none"> <li>• Here is what you should be able to do upon completion of this module <ul style="list-style-type: none"> <li>○ Define basic object-oriented principles: object, message, class, inheritance, and polymorphism.</li> <li>○ Perform OO Analysis with CRC modeling</li> <li>○ Construct UML class diagrams</li> </ul> </li> </ul>	<p>Specifically, what you will be able to do once you have finished this first part of our coverage of object-oriented analysis and design is shown here.</p> <p>You will be able to define basic object-oriented principles such as object, message, class, inheritance, and polymorphism.</p> <p>You will be able to perform OO Analysis. We will encourage you to use a technique called CRC modeling. CRC stands for Class-Responsibility-Collaborator.</p> <p>You will be able to construct UML analysis class diagrams to model the static structure of the objects.</p> <p>In the second module, we will finish our treatment of object oriented analysis by covering dynamic analysis modeling. The UML model we will use is the activity diagram. We will then turn to object oriented design. The difference between object oriented analysis and design is a matter of focus. Analysis focuses on discovering and specifying</p>

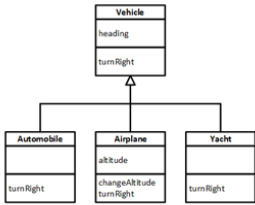
		the classes of objects inside the system. Design focuses on how these classes are implemented in the code.
4	 <h3>Outline</h3> <ul style="list-style-type: none"> <li>• Object-oriented principles (objects, messages, classes, inheritance, polymorphism)</li> <li>• Object-oriented Analysis <ul style="list-style-type: none"> <li>◦ Finding candidate classes</li> <li>◦ Performing CRC analysis</li> <li>◦ Constructing UML class diagram</li> </ul> </li> </ul>	<p>So we begin with a review of object-oriented principles. We need to make sure we are all in agreement about these concepts before we can go on to discuss object-oriented analysis and design.</p> <p>In this module, we will cover the first part of object-oriented analysis – finding candidate classes, modeling them using CRC analysis, and constructing the static model using UML class diagram notation.</p> <p>We will cover dynamic OO analysis modeling as well as object-oriented design in part 2 in the next module.</p>
5	 <h3>Next</h3> <ul style="list-style-type: none"> <li>• Object-oriented principles</li> </ul> 	Let's begin by reviewing basic object-oriented principles in the next video.



2-1	 <h2 style="text-align: center;">Object-Oriented Principles</h2> 	<p>In this video we will review the basic principles of object orientation. These principles are the basis for object oriented programming (in languages such as C++ or C# or Java, for example). They are also the basis for object-oriented analysis and design.</p> <p>It should be obvious that it makes sense to use object oriented analysis and design if you are going to be programming in an object-oriented programming language.</p>
2	 <h2 style="text-align: center;">Object-Oriented Principles</h2> <ul style="list-style-type: none"> <li>• Objects</li> <li>• Messages</li> <li>• Classes</li> <li>• Inheritance</li> <li>• Polymorphism</li> </ul>	<p>Here are the principles that we will be reviewing. We start with the basic concept of what an object is. We will then talk about how objects communicate by sending messages. One of the powerful aspects to object-oriented systems is that we can create classes of objects that abstract out common properties. We will discuss inheritance, or classes of classes. We will conclude with another powerful OO concept – polymorphism, which amplifies our use of abstraction.</p>
3	 <h2 style="text-align: center;">Objects</h2> <ul style="list-style-type: none"> <li>• An object is a thing</li> <li>• The thing can be something tangible or something conceptual</li> </ul> <div style="display: flex; justify-content: space-around;">   </div>	<p>We start with the concept of an object.</p> <p>An object is a thing. In other words it represents something that we care about... that we want to model. We want to create software to simulate these things that we care about.</p> <p>The thing that we are simulating can be something tangible, such as an airplane or pilot, or something intangible, such as an event like a takeoff or landing of an airplane.</p> <p>Everything we want to simulate has some data that needs to be remembered.</p>
4	 <h2 style="text-align: center;">In Analysis</h2> <ul style="list-style-type: none"> <li>• Something in the problem to be solved</li> </ul> 	<p>In analysis, these things are found in the problem space. We try to model the things in the problem to be solved.</p> <p>In this sense, analysis is a discovery process. The problem is naturally made up of objects. It is our job as analysts to look for them and model them.</p>

5	<p style="text-align: center;">In Code</p> <ul style="list-style-type: none"> <li>• a collection of data that describes something</li> </ul> 	<p>In design and code, objects are collections of data. Of course, the information encapsulated inside an object is used to simulate the object from the analysis.</p>
6	<p style="text-align: center;">Two Main Properties</p> <ul style="list-style-type: none"> <li>• Data</li> <li>• Behavior</li> </ul> <div style="display: flex; justify-content: space-around;">   </div>	<p>Recall from your programming, objects have two main properties. Data and behavior.</p> <p>Some synonyms for data are: state, attributes, fields, member data, and slots.</p> <p>Some synonyms for behavior are operations, methods, member functions, procedures, and services.</p>
7	<p style="text-align: center;">Objects Encapsulate Their Data</p> 	<p>The important thing to keep in mind is that the operations encapsulate the data. That is, the only way to access the information stored in an object from outside the object is to invoke its operations.</p>
8	<p style="text-align: center;">Messages</p> 	<p>The way to invoke an operation is to send a message to the object.</p> <p>A message is a request for service.</p> <p>An object will not perform any of its operations unless it receives a message.</p> <p>An object may perform the operation entirely by itself, using only its own data, or it may invoke the operations of other objects by sending messages to them in turn.</p> <p>A message has a name, parameters, and a result.</p> <p>The name, parameter types and result type are called the <i>signature</i> of the operation.</p>






		The set of operation signatures for an object is called its <i>protocol</i> .
9	<div> <div>Class</div>  </div>	<p>Classification is a form of abstraction. It means that we treat a group of objects in the same way. So we may have a bunch of books. Each book as an author and an isbn number. If we were modeling books in a library or a bookstore, we would want to be able to treat all books in the same way. We wouldn't want to have to model each book separately.</p> <p>Think of a class as a template for object instances.</p> <p>All of the objects described by a class behave in the same way, and encapsulate the same kind of data.</p> <p>There may be any number of object instances described by a class.</p> <p>One thing about a class in object-oriented systems is that a class has the ability to create or construct object instances.</p> <p>Some of the information stored in a class is descriptive information that is used when the class creates object instances.</p> <p>When you program in an object-oriented language, you create classes.</p> <p>The object instances don't get created until run-time.</p> <p>Our goal in object-oriented analysis and object-oriented design is to model classes, not object instances. That is, we are mainly interested in the <i>kinds</i> of objects we are dealing with.</p> <p>Now some of the weird stuff. A class is really a kind of object. It encapsulates data, as we just mentioned. It also has operations (such constructors). Something that has data and operations is called an object.</p> <p>Classes and object instances are stored differently in memory during the execution of</p>





		<p>a program. Classes are loaded into static memory. That is, since there is only one copy of each class, there is no need for the operating system to dynamically manage the storage allocated to the classes.</p> <p>Object instances are created by sending constructor messages to the classes. Since object instances can be created and destroyed, they have to be stored in memory that is managed dynamically. There are various schemes for handling this dynamic memory allocation. We will not get into that here.</p>
10	<div>  <h3>Inheritance</h3>  </div>	<p>As in any classification scheme, there are often similarities among classes.</p> <p>These similarities may be represented in the form of generalization-specialization relationships among the classes.</p> <p>The generalization class is sometimes referred to as the <i>superclass</i> and the specialization as the <i>subclass</i>.</p> <p>We typically try to use the terminology generalization and specialization in analysis. Since superclass and subclass are programming language specific concepts, we normally will use these terms in design and programming.</p>
11	<div>  <h3>UML</h3>  </div>	<p>In the Unified Modeling Language, generalization specialization relationships are indicated by a line with a big fat arrow head pointing from the specialization class to the generalization class.</p> <p>Here we see part of a class diagram for a book store. We sell three types of books: e-books, regular paper books, and audio books. They are all books, however. One thing that all books have in common is that they have a title. Specializations have properties of their own. For example, only e-books have digital rights management information.</p> <p>The generalization class defines a protocol that all of the specialization classes inherit.</p>

		<p>The specialization classes may add new attributes or operations.</p> <p>The specialization classes may override operations by having operations with the same signatures as in the generalization but with different behavior implementations.</p> <p>Some of the operations in a generalization class may be <i>abstract</i>. Only the signature is specified, but no implementation. All of the specialization classes must override these abstract operations.</p>
12	<p style="text-align: center;">Polymorphism</p>  <pre> classDiagram     class Vehicle {         heading         turnRight     }     class Automobile {         turnRight     }     class Airplane {         altitude         changeAltitude         turnRight     }     class Yacht {         turnRight     }     Vehicle &lt; -- Automobile     Vehicle &lt; -- Airplane     Vehicle &lt; -- Yacht </pre>	<p>In an object oriented system, variables may contain references to object instances.</p> <p>The type of the variable is specified by a class.</p> <p>If the type of the variable is a generalization class, that variable may contain a reference to an instance of that class or any of its specializations, or, in fact, any descendent class in the inheritance hierarchy stemming from the generalization class.</p> <p>But here's the important point: The protocol for sending messages to the object instance that the variable is pointing to is given by the type of the variable, not by the class of the object instance it is pointing to.</p> <p>Here we see part of a class hierarchy for vehicles. If we have a variable of type vehicle, it should be able to contain a reference to an object instance of an automobile, or an airplane, or a yacht. We should be able to send the message turnRight to the object without knowing which kind of object we are dealing with, since they all have steering wheels. However, suppose the variable is pointing to an airplane. We can't send the message changeAltitude to that object because the type of the variable is Vehicle, and there is no changeAltitude operation defined for class Vehicle.</p>



13	<div data-bbox="673 100 714 126">  </div> <div data-bbox="479 147 544 178">Next</div> <div data-bbox="300 205 430 231">• OO Analysis</div> <div data-bbox="446 262 581 388">  </div>	<p>Well that about does it for our review of object oriented principles. We will be applying these concepts for the remainder of our treatment of object-oriented analysis and design.</p> <p>In the next video, we will begin our treatment of object oriented analysis. Specifically we will concern ourselves with looking at the problem and discovering candidate classes to be modeled.</p>
----	--	---


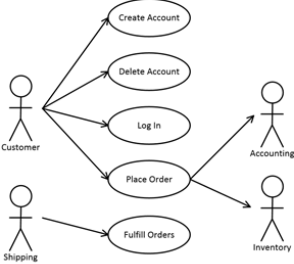


3-1	 	<p>In this video we will begin our treatment of object oriented analysis.</p> <p>The first step of OOA is to find some candidate classes.</p> <p>As we mentioned before, analysis is a discovery process. It is our job as analysts to discover what important classes of objects exist in our software system.</p>
2	 <ol style="list-style-type: none"> <li>1. Find classes of interest in problem domain</li> <li>2. Model static relationships</li> <li>3. Determine the responsibilities of the objects</li> <li>4. Model the dynamic behaviors</li> </ol>	<p>These candidate classes are the ones we will use to model the system during analysis.</p> <p>They are called candidate classes because they are our best guess at this point in the process as to the classes that we will need to eventually use to implement the system. It is early in the process, and so if we identify something as a candidate class, it may turn out to not be needed in the design. Alternatively, we may miss some class that will be needed down the line. We can always change our minds later. Things are pretty flexible at this point.</p> <p>We will create class diagrams that model the static relationships between the classes.</p> <p>We will then model the dynamic interactions between the classes using dynamic UML models such as activity diagrams.</p>
3	 <ul style="list-style-type: none"> <li>• Analysis objects hold information and operations pertinent to the problem domain</li> </ul> 	<p>So, how do we find the candidate classes? Where do we look for them?</p> <p>The information that we have about the system at this point is the problem statement and the set of requirements. Believe it or not, these documents are rich in information about the classes of objects that we will need to eventually implement in the design and code.</p> <p>Remember that objects encapsulate data and provide operations. At analysis time, then, let's look for what information and behaviors are mentioned in the problem statement and requirements.</p>

4	<p>Objects Have State</p> 	<p>Recall that objects are repositories for information. We say that objects have state. The state of an object is reflected in the information that is hidden inside the object. This information changes from time to time, thus resulting in state changes in the object.</p> <p>As you hopefully recall from our discussion of state machines, the state of an object has an effect on what happens when a message is sent to an object. The same message may result in different behaviors when the object is in different states.</p>
5	<p>Encapsulation</p> 	<p>We can model the objects as state machines without knowing how the data is implemented inside the object. That is, the object encapsulates, or hides, the data. We know it's there, but we don't need to know how it is stored. That is determined when we do design and coding.</p> <p>An object provides behaviors. We know what these behaviors are, and how they are specified, but we don't need to know how they are implemented. That's for design and code.</p>
6	<p>High Cohesion</p> 	<p>Another property we discussed earlier is that objects have high cohesion. An object is a collection of operations and data that are all related to a single purpose of the object. Adding an unrelated piece of data or functionality to a class specification reduces its cohesiveness. Likewise, removing essential data or operations also reduces its cohesion.</p>
7	<p>Loose Coupling</p> 	<p>Objects should be loosely coupled. There should be well defined dependencies between the objects. We would normally like to minimize the number of other objects any object is dependent on. This keeps the complexity of the dependency graph to a minimum.</p>


8	<div data-bbox="396 142 631 178" data-label="Section-Header"> <h3>Anthropomorphism</h3> </div> <div data-bbox="393 205 626 415" data-label="Image"> </div>	<p>Maybe you have noticed that we use terminology like objects “own” data and provide “services” to other objects. Objects communicate by sending messages.</p> <p>In other words, we use rather anthropomorphic terminology when talking about objects. Anthropomorphism means treating things like people. Anthro is Greek for person, morph means form. So anthropomorphism means giving human form to something. Usually the something is inanimate. We anthropomorphize lots of things. Like cars or boats or computers. We frequently talk about our software programs as though they were people. This seems quite natural. It is especially appropriate when talking about objects. Objects have state, just like people do. Objects request other objects to do things by sending them messages, just like people do.</p> <p>We will see in a bit how we can take advantage of this anthropomorphism when we model the classes using what is called CRC analysis.</p>
9	<div data-bbox="444 1098 583 1129" data-label="Section-Header"> <h3>Remember</h3> </div> <div data-bbox="298 1155 704 1182" data-label="List-Group"> <ul style="list-style-type: none"> <li>• We are modeling the problem in terms of</li> </ul> </div> <div data-bbox="467 1213 561 1245" data-label="Section-Header"> <h3>Objects</h3> </div> <div data-bbox="467 1266 578 1318" data-label="List-Group"> <ul style="list-style-type: none"> <li>• Not Data</li> <li>• Not Functions</li> </ul> </div>	<p>Above all, it is imperative that we remember that we are modeling the problem in terms of <b>objects</b>, not data, and not just functions.</p>
10	<div data-bbox="321 1465 708 1497" data-label="Section-Header"> <h3>Analysis is a Discovery Process</h3> </div> <div data-bbox="443 1528 583 1738" data-label="Image"> </div>	<p>Another thing to keep in mind is that analysis is a discovery process, not an invention process. Recall our discussion of the difference between discovery and invention. After we are done discovering and studying the classes, they should be understandable to all of the stakeholders. This means that there will not be any implementation classes in the analysis model. All of the classes should be reflected in the requirements or problem statement.</p>

11	<div data-bbox="272 100 764 121" data-label="Page-Header"></div> <div data-bbox="479 142 548 180" data-label="Section-Header"><h3>Next</h3></div> <div data-bbox="293 201 704 233" data-label="List-Group"><ul style="list-style-type: none"><li>• An example of finding candidate classes</li></ul></div> <div data-bbox="464 281 561 375" data-label="Image">A cartoon illustration of a person with blonde hair, wearing a purple long-sleeved shirt, sitting at a desk and looking at a laptop. The person is holding a pen in their right hand.</div>	<p>In the next video we will introduce a running example that we will use for the rest of our treatment of analysis and design. We will see the discovery of candidate classes.</p>
----	--	---

4-1	<p style="text-align: center;">Candidate Class Example</p> <p style="text-align: center;">Internet Storefront Application</p> 	<p>In this video we will be introducing a running example that we will be using for our discussion of object oriented analysis and design in the rest of this module and into the next.</p>
2	<p style="text-align: center;">Internet Storefront</p> 	<p>Almost everyone is familiar with on-line storefront systems. We log in, browse product listings, select products and add them to a virtual shopping cart, and then check out by providing payment information and shipping information.</p> <p>You can see here a simple use case model of such a system.</p>
3	<p style="text-align: center;">Use Case Specification</p> <p><b>Use Case Name:</b> Place Order  <b>Use Case Goal:</b> Customer creates an order, adds items to it, and checks out  <b>Actors:</b> Customer (p), Inventory, Accounting  <b>Preconditions:</b> Customer is logged on  <b>Postconditions:</b> Order is created, inventory and accounting are adjusted  <b>Trigger:</b> Customer selects "create order"</p>	<p>Let's focus on the use case for a customer to place an order.</p> <p>Since there is a use case for logging in, we will assume as a precondition here that the customer is logged in.</p> <p>The use case starts when the customer selects "create order."</p>
4	<p><b>Main success scenario</b></p> <ul style="list-style-type: none"> <li>• Customer selects "create order"</li> <li>• System creates an empty shopping cart for the customer</li> <li>• Customer browses the catalog for product to add to the shopping cart</li> <li>• Customer selects "add to cart"</li> <li>• Customer selects quantity</li> <li>• System checks inventory to make sure there is sufficient stock and adds the item(s) to the shopping cart</li> <li>• Customer checks out</li> <li>• System updates inventory and accounting appropriately</li> </ul> <p><b>Alternate scenarios</b></p> <ul style="list-style-type: none"> <li>• Customer may delete items from a cart at any time before checkout.</li> </ul>	<p>Here is the main success scenario.</p> <p>Each step that starts with the subject "System" is what we call a "system operation." There are three kinds of things that can happen in a use case scenario: the actor sends a message to the system, the system sends a message to an actor, or the system does something internally. At the use case level of abstraction, this internal behavior is encapsulated. In analysis, it is our job to discover what is inside the system to make this work. Of course, what is inside the system is a collection of objects.</p>


5	<div><div>Candidate Classes</div><div><ul style="list-style-type: none"><li>• Customer</li><li>• Shopping Cart</li><li>• Catalog</li><li>• Product</li><li>• Order Item</li></ul></div></div>	<p>So what objects need to be inside the system in order to make this use case work? Sometimes to help us get started, we can use the noun-verb analysis technique. In more complicated cases, this technique may not work quite as smoothly as it does in this little example.</p> <p>The classes that sort of “pop out” at us are these.</p> <p>We will need a customer object to hold information about a customer (such as address or credit card information) and to provide the behaviors that a customer does (such as placing an order).</p> <p>Of course, there will need to be a shopping cart object to hold information about the order that the customer is placing.</p> <p>Since the customer is browsing a catalog to find items to place in the shopping cart, we will need a catalog object.</p> <p>The things that are selected from the catalog are referred to as “products” in the use case, so we will go with that term.</p> <p>The things that are added to the shopping cart are called “items” in the use case. Let’s call these things “order items” just to avoid using such a generic term as simply “item.”</p>				
6	<div><div>Next</div><div><ul style="list-style-type: none"><li>• Modeling a system operation</li></ul></div><div><table><tr><th colspan="2">Name</th></tr><tr><td>Responsibilities</td><td>Collaborators</td></tr></table></div></div>	Name		Responsibilities	Collaborators	<p>The next task is to study the candidate classes to identify their properties.</p> <p>Of course, the properties we are talking about are the information and behavior of the objects.</p> <p>There is a nice technique for doing this investigation. It is called CRC analysis. That’s the subject of the next video.</p>
Name						
Responsibilities	Collaborators					


<p>5-1</p>	<div data-bbox="422 199 600 235" data-label="Section-Header"> <h3>CRC Analysis</h3> </div> <div data-bbox="409 252 617 392" data-label="Image"> </div>	<p>In this video we will see how to use CRC analysis to investigate the properties of the candidate classes inside the system. CRC stands for Class-Responsibility-Collaborator.</p> <p>This technique takes advantage of the aspect of objects that they can be anthropomorphized. This means that objects act like little people. They know things and they can do things.</p> <p>In CRC analysis, the team members play the roles of objects in acting out scenarios.</p> <p>Of course, when we say scenario, we mean system operation. From the use case specifications, the system operations are elemental functions performed by the system. They cannot be further decomposed.</p> <p>However, now that we have some idea of what objects are inside the system, and we have documented the system operations by listing the classes of objects involved in performing the system operation, we can role play the objects to figure out how they collaborate to carry out the system operation.</p> <p>This is a good group dynamic technique because it allows the group to work together to solve the problem. Psychologists have found that groups almost always outperform individuals on creative tasks like this.</p> <p>Of course, it is also a synchronous activity, requiring all team members to be working together at the same time. It wouldn't work so well in an asynchronous environment.</p>
<p>2</p>	<div data-bbox="440 1533 579 1568" data-label="Section-Header"> <h3>CRC Card</h3> </div> <div data-bbox="412 1604 612 1759" data-label="Image"> </div>	<p>The CRC approach was invented by Kent Beck and Ward Cunningham, and published in a paper contributed to a conference on object oriented systems in 1989. You may remember that Ward Cunningham is the inventor of the wiki collaboration tool. Kent Beck is the inventor of Extreme Programming, and is responsible for coining the term, Agile software development.</p> <p>They developed the technique while teaching a class in the object oriented programming</p>


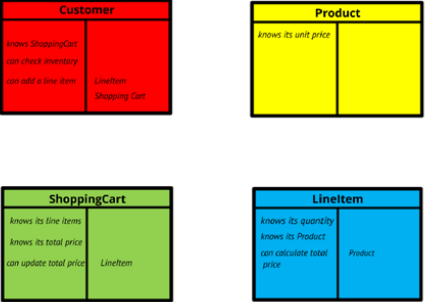
		<p>language, Smalltalk. The class was having a hard time understanding the concept of objects, since they had never programmed in an object oriented language before. Their main experience lay in functional programming languages such as Fortran or C.</p> <p>The story goes that the class was being taught off-site in a hotel. They were discussing the case study that the class was to be working on in the lounge after class one day, and began sketching out the classes for the solution on cocktail napkins. They came up with the idea that the teams could role play the objects and capture the properties of the objects on the napkins. Of course, this ability for the objects to be role played is because of the anthropomorphic nature of objects. When they got to class the next day, they decided to use index cards rather than napkins because they were easier to write on.</p> <p>Each card was labelled with the name of the class at the top. The bulk of the card was divided into two main areas, one for responsibilities and the other for collaborators.</p>
3	<div data-bbox="272 1167 760 1589"> <div data-bbox="272 1167 760 1188" style="background-color: #0070C0; height: 10px; width: 100%;"></div> <div data-bbox="418 1209 613 1241" style="text-align: center;">Responsibilities</div> <div data-bbox="297 1262 386 1287">An object</div> <div data-bbox="321 1318 438 1339"> <ul style="list-style-type: none"> <li>○ Knows things</li> </ul> </div> <div data-bbox="321 1451 438 1472"> <ul style="list-style-type: none"> <li>○ Can do things</li> </ul> </div> <div data-bbox="509 1251 678 1507">  </div> </div>	<p>An object has two kinds of responsibilities:</p> <ul style="list-style-type: none"> <li>- It can remember things (it has attributes).</li> <li>- It can do things (it has operations).</li> </ul> <p>There are naming conventions for these responsibilities. The first word of the responsibility is either “Can” or “Knows”. We use “Can” if it is the ability to do something, and “Knows” if it is the responsibility to know something.</p>





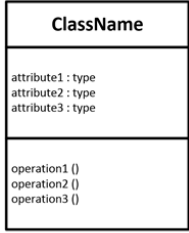

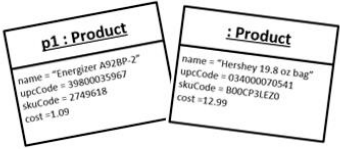
4	<div data-bbox="435 149 591 178" data-label="Section-Header"> <h3>Collaborator</h3> </div> <div data-bbox="298 207 664 256" data-label="List-Group"> <ul style="list-style-type: none"> <li>• Another object that is needed to help perform a responsibility</li> </ul> </div> <div data-bbox="443 275 581 375" data-label="Image"> </div>	<p>We also know that objects sometimes can't perform the entire behavior of being able to do something all on its own. It must request the services of other objects by sending them messages. They called this dependency a collaboration.</p> <p>For any responsibility to be able to do something that required the assistance of other objects, these other objects would be listed on the collaborator side of the CRC card.</p>
5	<div data-bbox="454 588 568 617" data-label="Section-Header"> <h3>Example</h3> </div> <div data-bbox="420 657 600 777" data-label="Diagram"> </div>	<p>Here's what a CRC card might look like while role-playing a Product object in studying a storefront application</p> <p>Notice that the collaborator, Tax, is listed opposite the responsibility, "Can compute tax." This is an object that is needed to help out with this behavior. We need this because it would not be cohesive to include tax information inside a Product object.</p> <p>If we hadn't discovered the need for the Tax class during our initial discovery process, we can easily add it to the list at this point.</p>
6	<div data-bbox="425 1100 596 1129" data-label="Section-Header"> <h3>CRC Process</h3> </div> <div data-bbox="295 1155 708 1354" data-label="List-Group"> <ul style="list-style-type: none"> <li>• Pick a system operation to model</li> <li>• Choose the object that will receive the trigger message from the actor</li> <li>• Repeat the following as long as there are responsibilities that need to be detailed <ul style="list-style-type: none"> <li>◦ Choose a collaborating object</li> <li>◦ Negotiate the contract between the client object and the server object</li> </ul> </li> </ul> </div>	<p>So here's the CRC process.</p> <p>We start by picking a system operation to study. We have taken our best guess as to the objects that are involved in the system operation, either because we think they hold information needed for the system operation, or they can perform part of the function called for in the system operation.</p> <p>A CRC card is created for each object involved in the scenario. Each team member takes ownership of one or more of the cards. It is up to that team member to play the role of that object when the scenario is being acted out.</p> <p>We start with the triggering event. This is typically a message that is sent to the system from the primary actor.</p> <p>One of the objects within the system will have to accept the responsibility to carry out the system operation. In some cases, if we're</p>

		<p>not sure which object should have this responsibility; we may make up a separate “controller” object just for this purpose. We write the name of this responsibility on the left side of the CRC card of this object.</p> <p>Now we repeat the following until the system operation is complete: If an object has a “can” responsibility listed, and that object can’t completely perform the operation, it’s usually because it involves information managed by some other object or objects.</p> <p>The team will determine which other objects are involved. The object doing the collaborating is called the client, and the objects providing the needed responsibilities are called servers.</p> <p>The team member playing the role of the client will write the names of the server objects on the collaborator side of the client CRC card. Usually, we keep the collaborator names even with the responsibility name.</p> <p>For each collaborator, the team member playing the role of the client will negotiate with the team member playing the role of the server object to make up the name of the responsibility provided by the server object.</p> <p>The team member playing the role of the server object will write the name of this responsibility on the left side of the CRC card.</p> <p>We keep doing this until all collaborations have been negotiated.</p>
7	<div> <div>Hints</div>  </div>	<p>Don’t get bogged down in discussions of how the objects communicate. We won’t worry about this until we get to Object Oriented Design.</p> <p>Don’t attempt to be dogmatic about the CRC cards. Take your best guesses about responsibilities and collaborations, but don’t agonize over these decisions. They are just a first cut. You’ll get a chance to refine your decisions later when you do static and dynamic OOA modeling. For now, we’re just</p>

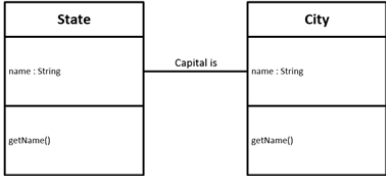
		<p>trying to get a general sense of the properties of the objects.</p> <p>Even though the first C in the name of the approach stands for Class, we have found it makes more sense to think of the cards as object instances of the classes. In some scenarios, you might find that you will need two or more instances of a particular class. Just use multiple cards. They can all be role played by the same team member or by different people. Your choice.</p> <p>The process works better if the entire team is in the same room, sitting around a table.</p> <p>A good medium to use is those 4x6 index cards. They give you enough room to write a lot. Of course, if you run out of room, you can certainly use multiple cards.</p> <p>This process works less well if the team members are distributed, and the cards are shared electronically. It doesn't work at all if the team members are participating asynchronously.</p> <p>You don't have to do this for all of the system operations, but choose a representative sample that will allow you to study all of the candidate classes of objects.</p>
8	<div> <div>Next</div> <div> <ul style="list-style-type: none"> <li>• An example of the CRC process</li> </ul> </div> <div>  </div> </div>	<p>In the next video we will continue our Internet storefront example. We will see how CRC analysis might work.</p>

6-1	 <p>CRC Example</p>	In this video we will continue our Internet storefront example. We will see how CRC analysis might work.
2	<p>Add Item to Cart</p> <p>After finding the product, the customer selects “add to cart.” The customer also indicates the quantity. The system checks inventory to make sure there is sufficient quantity in stock. The item is added to the cart, and the total price of the order is updated.</p>	We have several system operations to pick from. Let us choose “Add item to Cart.” The trigger would be selecting “add to cart” after finding the product in the catalog. The inputs to this system operation would be the product and the quantity. The system would check inventory, and if there are enough in stock, the item with its quantity would be added to the shopping cart. The total price would be updated.
3	 <p>The diagram shows four objects in a 2x2 grid:</p> <ul style="list-style-type: none"> <li><b>Customer (Red):</b> <ul style="list-style-type: none"> <li>knows ShoppingCart</li> <li>can check inventory</li> <li>can add a line item</li> <li>LineItem</li> <li>ShoppingCart</li> </ul> </li> <li><b>Product (Yellow):</b> <ul style="list-style-type: none"> <li>knows its unit price</li> <li></li> <li></li> </ul> </li> <li><b>ShoppingCart (Green):</b> <ul style="list-style-type: none"> <li>knows its line items</li> <li>knows its total price</li> <li>can update total price</li> <li>LineItem</li> </ul> </li> <li><b>LineItem (Blue):</b> <ul style="list-style-type: none"> <li>knows its quantity</li> <li>knows its Product</li> <li>can calculate total price</li> <li>Product</li> </ul> </li> </ul>	<p>The objects we think will be needed for this system operation are Customer, Product, LineItem, and ShoppingCart.</p> <ol style="list-style-type: none"> <li>1. Since the customer has already created an empty shopping cart, we can say that one of the responsibilities of Customer is to know its shopping cart.</li> <li>2. Before the product can be added to the cart, inventory must be checked. We let the Customer object take that responsibility. It doesn't need to collaborate with any other object to do this.</li> <li>3. The Customer then has to be able to add a line item to that shopping cart.</li> <li>4. It will need to collaborate with LineItem in order to set the quantity and product.</li> <li>5. LineItem then knows its quantity</li> <li>6. And its product.</li> <li>7. Customer can now collaborate with Shopping cart to add the line item.</li> </ol>

		<p>8. The shopping cart has the responsibility to know its lineItems.</p> <p>9. The system operation specifies that the total price be updated. We will let the shopping cart object keep track of its total price.</p> <p>10. It should also be able to update the total price.</p> <p>11. For this, it needs to collaborate with the lineItem.</p> <p>12. LineItem will have the responsibility to compute its total price by multiplying its quantity by its unit price.</p> <p>13. It will have to collaborate with the Product in order to get the unit price.</p> <p>14. The product then must have the responsibility to know its unit price. That is the right place to keep it because it is a property of the product, not each item.</p>
4		<p>So we now have discovered several properties of some of the classes within our system.</p> <p>In the next video we will see how we might represent the properties of our classes in UML, the Unified Modeling Language.</p>

7-1	 <p>Classes and Object Instances</p> <h1>UML</h1>	<p>In this video we will discuss how classes and object instances are documented in the Unified Modeling Language, UML.</p>
2	<p>Class</p>  <pre> classDiagram     class "ClassName" {         attribute1 : type         attribute2 : type         attribute3 : type         operation1 ()         operation2 ()         operation3 ()     } </pre>	<p>UML uses a three part box to document a class. The top part contains the name of the class (centered, in bold font). The middle part contains a list of the attributes (normal font, left justified). The bottom part contains a list of the operations (normal font, left justified).</p>
3	<p>Example</p>  <pre> classDiagram     class "Product" {         name : String         upcCode : UPC         skuCode : SKU         cost : Currency         computeTax()     } </pre>	<p>The information from the CRC cards may be used to fill in some of the attributes and operations.</p> <p>If the responsibility of an object is to know something, that usually maps to an attribute. But be careful. If the thing the object knows is another object, then this would be modeled by something called a link. We'll cover links in just a second.</p> <p>If the responsibility of an object is to be able to do something, that usually maps to an operation.</p>
4	<p>Object Instances</p>  <pre> classDiagram     class "p1 : Product" {         name = "Energiier A02BP-2"         upcCode = 39800035967         skuCode = 2749618         cost = 1.09     }     class ": Product" {         name = "Hershey 19.8 oz bag"         upcCode = 034000070541         skuCode = B00CP3LEZ0         cost = 12.99     } </pre>	<p>We can also model object instances in UML. An instance is a two part box.</p> <p>The bottom part consists of a list of its attribute names and their values, separated by equals signs.</p> <p>The syntax of what goes in the top box looks similar to a variable declaration in UML: a variable name, followed by a colon, followed by the type of the variable. (By the way this syntax is borrowed from the languages Ada and Pascal, in case you were wondering.) This line is written in underlined font.</p> <p>The name that appears to the left of the colon</p>

		<p>represents the reference or pointer to the object instance. In object oriented design, we will assign this reference value to variables, and pass it as an argument to operations.</p> <p>In analysis, we generally don't worry about the references. In fact, we typically don't even show anything to the left of the colon. That is, we elide the object reference name, and just show the colon and the name of the class of the object instance in the top box. This is called an anonymous object instance.</p>
5	<p><b>Object Instance Diagram</b></p> <pre> graph LR     S1[":State Name = 'MD'"] --- "Capital is" C1[":City Name = 'Annapolis'"]     S2[":State Name = 'NY'"] --- "Capital is" C2[":City Name = 'Albany'"] </pre>	<p>This is an object instance diagram. For one class diagram, there may be many object instance diagrams.</p> <p>When two object instances are physically connected, we say that they are linked. We discovered some of the links when we did CRC modeling. When we said that one object “knew” about another object, this is a link. In an object instance diagram, we represent a link as a line drawn between the two object instances. A link is a semantic relationship between two object instances. It represents information that is jointly held by the two objects. Here we see two object instances. One is an instance of the class State. The other is an instance of the class City.</p> <p>The link is shown as a line connecting the two object instances. Notice that the link is annotated with the semantic information shared by the two objects. We see attribute values represented in the object instances. The “Capital is” information is not an attribute of just the state or just the city. It is shared by the two objects. That’s why it’s represented by the link connecting the two objects.</p>

6	<div data-bbox="420 144 605 178" data-label="Section-Header"> <h3>Class Diagram</h3> </div>  <pre> classDiagram     class State {         name : String         getName()     }     class City {         name : String         getName()     }     State --&gt; City : Capital is   </pre>	<p>When links may exist between object instances, we say that there is an association between their corresponding classes. In the class diagram, we represent associations by lines drawn between the classes. The line is labeled with a name indicating the semantic information contained in the links. The diagram you see here shows two classes and an association.</p> <p>Note that no values for the attributes are shown, because these are classes, not object instances.</p>
7	<div data-bbox="383 623 644 655" data-label="Section-Header"> <h3>In the Analysis Model</h3> </div> <ul data-bbox="298 680 670 793" style="list-style-type: none"> <li>• No link directionality is assumed</li> <li>• Link is not stored in either object</li> <li>• Links are not necessarily conduits for messages</li> </ul>	<p>There are a few things to remember about links in the analysis model. First, links are not directional. We might like to represent them as arrows, but, as we will see when we get to design, an arrow indicates that a decision has been made about where the link information will be stored. That's a design decision, so we don't use arrows in our analysis model.</p> <p>Now you might notice that the name of the association implies a direction. The association only reads correctly in one direction. Maryland's capital is Annapolis. It doesn't make sense to say the Annapolis's capital is Maryland. That doesn't mean that the link is directional, though. The link information is just as much a part of the City object as it is the State object. So what do we do about the fact that the association name only makes sense when you read it one direction? The convention that is usually adopted is that the association name should read properly left to right or top down. Usually you will draw your class diagrams so the associations are either horizontal or vertical, so this rule is easy to follow.</p> <p>The last point to remember is that links or associations represent information. We should treat them as structural relations. Specifically, try not to think of an association as a conduit for sending messages between objects. Any object should be able to send a message to any other object it needs to, regardless of whether there is a link connecting the objects. All you need in order</p>



		<p>to send a message is a reference to the object you want to send the message to. This reference can be passed around as a piece of data and doesn't necessarily need to be stored as an attribute anywhere. This holds true for both analysis and design.</p>
8	<p><b>Association Names</b></p> <pre> classDiagram     class Senator {         name: String         party: String         perform()     }     class State {         name: String         perform()     }     class City {         name: String         perform()     }     Senator --&gt; State : represents     State --&gt; City : Capital is </pre>	<p>When you need to pick a name for an association, the name of the information being stored will give you a clue. Use a verb or verb phrase. The information contained in the link can be determined by reading the class names with the association name between them. We saw this in the previous example. The class names were State and City, and the association name was “capital is.” This reads sort of like a sentence: State capital is City. Another example might be the “represented by” association connecting the State and Senator classes.</p>
9	<p><b>Next</b></p> <ul style="list-style-type: none"> <li>• Adornments <ul style="list-style-type: none"> <li>○ Multiplicity</li> <li>○ Roles</li> <li>○ Comments</li> </ul> </li> </ul>	<p>In the next video we will discuss some additional UML notation for documenting associations.</p>

8-1	<div data-bbox="365 199 662 231" data-label="Section-Header"> <h3>Association Adornments</h3> </div> <div data-bbox="349 262 467 325" data-label="List-Group"> <ul style="list-style-type: none"> <li>• Multiplicity</li> <li>• Roles</li> <li>• Comments</li> </ul> </div>	<p>In this video, we will discuss some additional UML notation for describing properties of associations. We will see how to express multiplicity, roles and constraints.</p>
2	<div data-bbox="410 514 617 546" data-label="Section-Header"> <h3>Multiplicity: Fixed</h3> </div> <div data-bbox="332 583 695 762" data-label="Diagram"> <pre> classDiagram     class Senator {         name : String         party : String         castVote()     }     class US_Senate {         session : int         holdVote()     }     Senator "100" -- "1" US_Senate : member of   </pre> <p>The diagram shows a class <b>Senator</b> with attributes <code>name : String</code> and <code>party : String</code>, and a method <code>castVote()</code>. It is associated with a class <b>US Senate</b> which has an attribute <code>session : int</code> and a method <code>holdVote()</code>. The association is labeled "member of". The multiplicity at the <b>Senator</b> end is 100, and at the <b>US Senate</b> end is 1.</p> </div>	<p>The number of object instances that may be linked is indicated by the multiplicity notation near the ends of the association line. The symbol indicates the number of instances of the class that may be linked to an instance of the class at the other end of the association.</p> <p>If a fixed number of instances of a class may be linked to an instance of the class on the other end of the association, put that number near the end of the line. In this diagram, each senator is a member of one US Senate. The Senate has 100 members.</p>
3	<div data-bbox="402 993 625 1024" data-label="Section-Header"> <h3>Multiplicity: Range</h3> </div> <div data-bbox="321 1060 711 1249" data-label="Diagram"> <pre> classDiagram     class Company {         name : String         stockSymbol : String         hire()         terminate()     }     class Person {         name : String         ssn : SocSecNo         work()         drinkCoffee()     }     Company "0..1" -- "*" Person : employs   </pre> <p>The diagram shows a class <b>Company</b> with attributes <code>name : String</code> and <code>stockSymbol : String</code>, and methods <code>hire()</code> and <code>terminate()</code>. It is associated with a class <b>Person</b> which has attributes <code>name : String</code> and <code>ssn : SocSecNo</code>, and methods <code>work()</code> and <code>drinkCoffee()</code>. The association is labeled "employs". The multiplicity at the <b>Company</b> end is 0..1, and at the <b>Person</b> end is *.</p> </div>	<p>If the number is arbitrary, put an asterisk at the end of the line. If the number is a range of values, use "subrange" (double dot) notation. The subrange 0..1 is special, and it indicates that the link is optional.</p>
4	<div data-bbox="479 1360 548 1392" data-label="Section-Header"> <h3>Roles</h3> </div> <div data-bbox="313 1423 719 1602" data-label="Diagram"> <pre> classDiagram     class Company {         name : String         stockSymbol : String         hire()         terminate()     }     class Person {         name : String         ssn : SocSecNo         work()         drinkCoffee()     }     Company -- Person : employs     note over Company, Person "roles: employer, employee"     note over Company "multiplicity: 0..1"     note over Person "multiplicity: *"   </pre> <p>The diagram shows the same classes as in the previous diagram. The association is labeled "employs". The role "employer" is placed near the <b>Company</b> end, and the role "employee" is placed near the <b>Person</b> end. The multiplicity at the <b>Company</b> end is 0..1, and at the <b>Person</b> end is *.</p> </div>	<p>For clarification, we may indicate the role that an object plays in an association relationship. The role is annotated on the association line at the end near the object playing the role.</p> <p>So now we have both multiplicity and role names vying for space towards the end of the association line. Some tools will help with the placement of these artifacts, while others are a bit more free form. The UML rules simply say that these annotations go near the end of the association, but are not specific about exact placement.</p>

5	<div data-bbox="264 98 763 126" data-label="Image"></div> <h3 data-bbox="415 149 613 176">UML Comments</h3> <ul data-bbox="298 205 659 344" style="list-style-type: none"> <li>• Written inside curly braces {like this}</li> <li>• Or inside dog-eared boxes</li> <li>• Constraints restrict data <ul style="list-style-type: none"> <li>◦ Attributes</li> <li>◦ Links</li> </ul> </li> </ul> <div data-bbox="673 260 735 296" data-label="Image"></div>	<p>In UML, you can add a comment to any modeling element. If we put the comment inside curly braces, it is referred to as a property of the modeling element. The property is placed near the modeling element to which it applies.</p> <p>We can also put comments inside dog-eared boxes, and draw a dashed line to the modeling element to which the comment refers.</p> <p>One particularly useful type of comment is a constraint. Just as their name implies, constraints constrain or restrict information.</p>
6	<div data-bbox="264 646 763 674" data-label="Image"></div> <h3 data-bbox="383 697 646 724">Constraint Semantics</h3> <ul data-bbox="298 753 708 808" style="list-style-type: none"> <li>• Must be true when object is instantiated</li> <li>• Operations must preserve the constraints</li> </ul>	<p>Remember that information is represented in a Class diagram in two forms: Attribute values, and links. So a constraint can be a property that restricts the value of an attribute or an association.</p> <p>It is worth noting that multiplicities are a form of constraint. They have the effect of constraining the number of links that can be created, particularly multiplicities that have fixed lower or upper bounds.</p> <p>Constraints are usually expressed in mathematical or logic notation, but natural language may be used if the math is too hard to understand. There is a formal notation that is part of the UML called OCL: Object Constraint Language. The advantage of OCL is that it can be processed by a CASE tool.</p> <p>The semantics of constraints involve two rules:</p> <p>First: Whenever an object instance is constructed, the constraints must be true.</p> <p>Second: Any operation on an object or link that changes its value must preserve the constraint.</p>

7


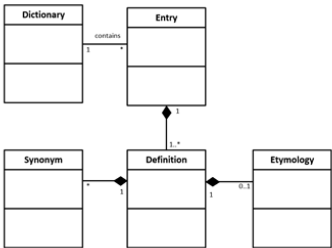


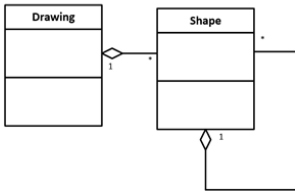

Next

- Composition

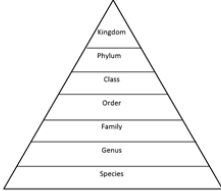
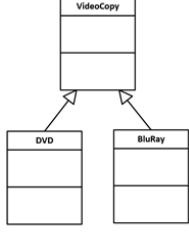


In the next video we will talk about a special kind of association, the composition relationship.

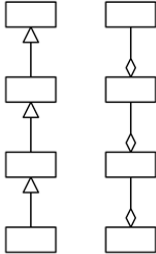
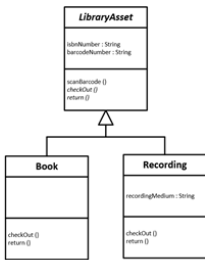
9-1	 <p>Composition</p>	<p>In this video, we will discuss a special kind of association, called a composition. A composition is a part-whole relationship.</p>
2	<p>Composition: Part-Whole</p>  <pre> classDiagram     class Dictionary     class Entry     class Definition     class Synonym     class Etymology      Dictionary "1" -- "*" Entry : contains     Entry "1" -- "1..*" Definition     Definition "1" -- "*" Synonym     Definition "1" -- "0..1" Etymology   </pre>	<p>One class, the part, represents pieces that are owned by the other class, the whole, or composite.</p> <p>The UML notation for a composition relationship is a line with a solid diamond on the end toward the composite class. Although it is an association, an association name is not necessary. Neither are role names. The meaning of the association is “composed of” and is indicated by the diamond.</p> <p>In the diagram here, a dictionary entry consists of some number of definitions. Each definition also includes a collection of synonyms, and an optional etymology. The synonyms and etymology are parts of the definition.</p> <p>The composite is an abstraction that stands in for all its parts. The dictionary entry IS all of its definitions. A definition IS its synonyms and etymology.</p> <p>The composite object encapsulates the part objects. Client objects deal with the composite object rather than the parts individually. This means that the composite will have to have operations that may be invoked by client objects that, in turn, invoke operations on the encapsulated part objects, since those part objects are not really visible outside the composition.</p> <p>One way to tell if a composition is appropriate is the cascading delete rule: If the composite goes away, so do its parts.</p> <p>This means that the lifetime of the parts is</p>

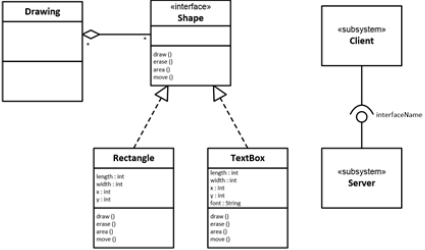
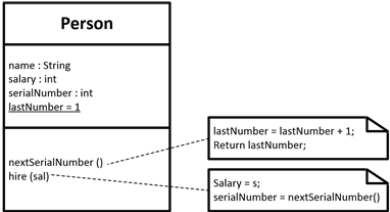
		<p>controlled by the composite. The parts are created by the composite, and the parts are destroyed by the composite.</p> <p>The composite has exclusive use of the parts. No object other than the composite may send messages to any of the part objects.</p>
3	<p style="text-align: center;">Aggregation</p> 	<p>If not all of these constraints apply to your situation, but you still want to express a part-whole relationship, you may use what is called an Aggregation relationship in UML. This relationship is indicated by an open diamond on the end toward the aggregator class. An aggregation is simply a collection, but the parts are not necessarily encapsulated within the aggregator. An object may be part of multiple different aggregations, something that is not allowed in composition. In this example we see what is called a recursive aggregation. An object instance of Shape can be made up of other object instances of Shape. We wouldn't want to make this a composition relationship because we would want to be able to send messages to any of the shapes in the hierarchy without having to send the message to the top level object.</p>
4	<p style="text-align: center;">Properties</p> <ul style="list-style-type: none"> <li>• Transitive</li> <li>• Anti-symmetric</li> <li>• Synergism</li> </ul>	<p>Here are some properties shared by both compositions and aggregations.</p> <ul style="list-style-type: none"> <li>- Transitive – If a handle is part of a door, and a door is part of a car, then a handle is part of a car.</li> <li>- Anti-symmetric – If a door is part of a car, then a car cannot be part of a door.</li> <li>- The aggregate or composite may have some attributes or operations of its own, in addition to those in the part objects. A composite or aggregate is greater than just the sum of its parts.</li> </ul>
5	<p style="text-align: center;">Next</p> <ul style="list-style-type: none"> <li>• Generalization-Specialization</li> </ul> 	<p>In the next video, we will discuss the generalization/specialization relationship that might exist between classes.</p>


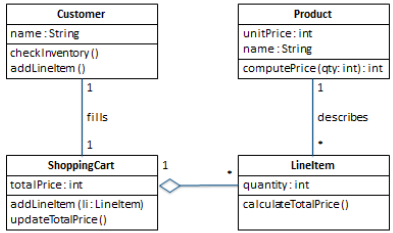


10-1	<p style="text-align: center;">Generalization-Specialization</p> 	<p>In this video, we will discuss how to model generalization/specialization relationships that may exist between classes.</p>
2	<p style="text-align: center;">Generalization</p> 	<p>Even though we draw associations, compositions and aggregations in the class diagram, they actually represent links that exist between object instances.</p> <p>Generalization/specialization relationships exist between classes. Even so, we draw them on the same class diagram.</p> <p>A generalization/specialization relationship will never have a realization in an object instance diagram.</p> <p>We are familiar with generalization/specialization relationships. In code this is implemented by inheritance.</p>
3	<p style="text-align: center;">Rules</p> <ul style="list-style-type: none"> <li>• Specializations <b>inherit</b> all properties <ul style="list-style-type: none"> <li>○ Attributes</li> <li>○ Operations</li> <li>○ Associations</li> <li>○ Constraints</li> </ul> </li> <li>• Specialization may <b>add</b> new properties</li> <li>• Specialization may <b>override</b> operations</li> </ul>	<p>Some rules apply to generalization/specialization relationships. Specializations may add new</p> <ul style="list-style-type: none"> <li>- Attributes</li> <li>- Operations</li> <li>- Links</li> <li>- Constraints (the constraints are ANDed)</li> </ul> <p>Specializations may only override</p> <ul style="list-style-type: none"> <li>- Operations</li> </ul> <p>Specializations “inherit” all properties from their generalizations:</p> <ul style="list-style-type: none"> <li>- Attributes</li> <li>- Operations</li> <li>- Links</li> <li>- Constraints</li> </ul>

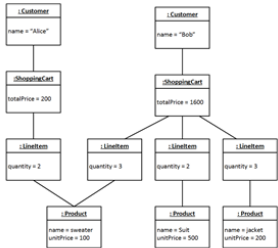



<p>4</p>	<p>Generalization vs. Aggregation</p> 	<p>It's interesting to note that there are similarities between generalization and aggregation relationships.</p> <p>They are both hierarchical relationships.</p> <p>In generalization:</p> <ul style="list-style-type: none"> <li>- You accumulate properties as you go down the hierarchy.</li> <li>- Messages sent to an object instance may be executed by one of its super-classes.</li> </ul> <p>In aggregations and compositions:</p> <ul style="list-style-type: none"> <li>- You accumulate properties as you go up the hierarchy.</li> <li>- Messages sent to an object may be delegated to one of its parts</li> </ul> <p>In fact, it is sometimes useful to simulate inheritance through aggregation. So, instead of inheriting attributes or operations from a superclass, you delegate these properties to another class. We take advantage of this in many of the object-oriented design patterns, such as the Gang of Four patterns.</p> <p>We will discuss object oriented design patterns in a later module in this course.</p>
<p>5</p>	<p>Abstract Class</p> 	<p>Sometimes it is useful to be able to describe a class that is just a placeholder in the inheritance hierarchy. An abstract class cannot have any direct object instances of its own.</p> <p>An abstract class may have abstract operations as well as concrete operations. An abstract operation in an abstract class is simply a specification of an operation. It is up to subclasses to implement these operations.</p> <p>There is no such thing as abstract attributes.</p> <p>An abstract operation is indicated with italics font.</p>

6	<p style="text-align: center;"><b>Interface</b></p> 	<p>In UML a class that has all abstract operations and no attributes is a special kind of classifier. It is called an Interface. In the class diagram, it looks like a class. To indicate that it is an interface, we use the stereotype «interface» as shown here. All operations are abstract (in italics font), and there are no attributes.</p> <p>An interface specifies an abstract protocol to be “realized” by either a class or a subsystem.</p> <p>If the interface is realized by a class, we use the triangle with a dashed line between the interface and the class.</p> <p>If the interface is realized by a package, the relationship is shown using the ball and socket notation.</p>
7	<p style="text-align: center;"><b>Classifier Scope</b></p> 	<p>Attributes that are stored in the class rather than object instances are referred to as classifier scope variables.</p> <p>Classifier scope is indicated with underline font.</p> <p>This is the same as static attributes in Java, C++ and C#.</p> <p>Operations that are executed by the class rather than an object instance are called classifier scope operations. They are the same thing as static operations in object oriented programming languages.</p> <p>There don't need to be any object instances in order to invoke a classifier scoped operation.</p>
8	<p style="text-align: center;"><b>Next</b></p> <ul style="list-style-type: none"> <li>• Static modeling example</li> </ul>	<p>We will conclude this module with an example of converting the information from our CRC cards into a UML class diagram.</p>

11-1	<p style="text-align: center;">Static Modeling Example</p> <p style="text-align: center;">Internet Storefront</p>	<p>Let's revisit our Internet Storefront application. The last thing we did was CRC analysis. We now want to create a static model using UML Class Diagram notation</p>
2	<p style="text-align: center;">CRC Cards</p> 	<p>Here are the CRC cards that we developed during our role playing exercise. As you recall, we identified four classes. Customer, Product, Lineltem and ShoppingCart. We can use the information that we gained during our role playing to fill in the UML class diagram.</p>
3	<p style="text-align: center;">Class Diagram</p> 	<p>Here are the four classes drawn as three-part boxes in UML. I've already put in the class names.</p> <ol style="list-style-type: none"> <li>1. The first thing that we did in our CRC activity was to note that the Customer object already had created an empty ShoppingCart object. We said that the Customer “knew” its ShoppingCart. Now usually, when we say that an object “knows” something, this becomes an attribute. However, since ShoppingCart is another class, this will become an association between Customer and ShoppingCart.</li> <li>2. Let's give it a name. Let's say a customer <i>fills</i> a shopping cart.</li> <li>3. How many shopping carts can be filled by each Customer? Just one.</li> <li>4. How many customers can fill one instance of Shopping cart? Exactly one.</li> <li>5. The first thing that we decided that had to be done is to check inventory. We decided to let the Customer object do this, so we'll add an operation checkInventory to the Customer class.</li> <li>6. Then the actor tells the system to add an order item, specifying the Product and the quantity. We include an operation addLineltem in the Customer class to accept this responsibility.</li> <li>7. The Customer object will then need to</li> </ol>

		<p>create an instance of the <code>LineItem</code> class, providing the quantity and product to the constructor. We will make quantity an attribute of a <code>LineItem</code> object.</p> <ol style="list-style-type: none"> <li>8. But since <code>Product</code> is another class, we will implement “Line Item Knows its Product” as an association between <code>Line Item</code> and <code>Product</code>.</li> <li>9. Let’s name this association “describes.” A product describes a line item.</li> <li>10. How many <code>Product</code> objects will describe any <code>LineItem</code>? Exactly one.</li> <li>11. How many <code>LineItem</code> objects can be described by the same <code>Product</code> object? Any number.</li> <li>12. Once the <code>LineItem</code> object is created, we need to add it to the <code>ShoppingCart</code>. This is an operation, <code>addLineItem</code>.</li> <li>13. We said that <code>ShoppingCart</code> “knows” its line items. This will have to be an association. If we think about this, it appears that a shopping cart is a container for multiple line items. This is best modeled by an aggregation. It wouldn’t be a composition (with the solid diamond) because we don’t want to completely encapsulate the line items within the shopping cart, but we do want to indicate that the shopping cart is a collection of line items.</li> <li>14. The multiplicity on the aggregate end of the association is usually 1, but not necessarily. In this case a line item can only be in one shopping cart.</li> <li>15. How many line items can we put in the shopping cart? Any number.</li> <li>16. Next we’ll deal with the total price of the order. Let’s make <code>totalPrice</code> an attribute of the <code>ShoppingCart</code>.</li> <li>17. After the line item has been added to the shopping cart, we will need to update the total price. We will include an operation, <code>updateTotalPrice</code>, to do this.</li> <li>18. <code>updateTotalPrice</code> will have to ask the <code>LineItem</code> to help with this, so we include an operation, <code>calculateTotalPrice</code>, in the line item class.</li> <li>19. The total price is based on the quantity and the unit price. The line item object knows its quantity, but, as we discussed during our CRC analysis, it’s better to put the unit price in the <code>Product</code> object, since it is a property of a <code>Product</code> rather than each line item.</li> </ol>
--	--	--

		<p>Now there are two ways to do the computation. The <code>LineItem</code> could call an operation <code>getUnitPrice</code> in the <code>Product</code> and then multiply it by the quantity. The main problem with this is that it breaks the encapsulation of the unit price within the <code>Product</code> class. A better approach would be to put an operation, <code>computePrice</code>, in the <code>Product</code> class. We would pass in the quantity. The <code>computePrice</code> operation would multiply the quantity by the unit price and return the result. This encapsulates the unit price. There is an added benefit of this approach. Say the unit price was on a sliding scale, where there is a discount based on quantity. The algorithm for applying this discount could also be encapsulated inside the <code>Product</code> class.</p> <p>20. So we will make <code>unitPrice</code> an attribute of <code>Product</code>.</p> <p>21. There are two more attributes that I'd like to add. These didn't come out in the CRC modeling, but I'm going to add them to facilitate talking about object instances. Let's put a <code>name</code> attribute in the <code>Customer</code> class.</p> <p>22. And let's put a <code>name</code> in the <code>Product</code> class.</p> <p>And there's our class diagram.</p>
4	<p style="text-align: center;">Instance Diagram</p>  <pre> classDiagram     class Customer {         name     }     class ShoppingCart {         totalPrice     }     class LineItem {         quantity     }     class Product {         name         unitPrice     }     Customer "1" -- "1" ShoppingCart     ShoppingCart "1" -- "1" LineItem     LineItem "1" -- "1" Product   </pre> <p>The diagram illustrates the state of objects at a specific point in time. It shows two <code>Customer</code> objects: <code>Alice</code> and <code>Bob</code>. <code>Alice</code> has a <code>ShoppingCart</code> object with a <code>totalPrice</code> of 200. This cart contains one <code>LineItem</code> object with a <code>quantity</code> of 2, which points to a <code>Product</code> object (sweater) with a <code>unitPrice</code> of 100. <code>Bob</code> has a <code>ShoppingCart</code> object with a <code>totalPrice</code> of 1600. His cart contains three <code>LineItem</code> objects: one with a <code>quantity</code> of 3 pointing to a <code>Product</code> (sweater, <code>unitPrice</code> 100), one with a <code>quantity</code> of 2 pointing to a <code>Product</code> (suit, <code>unitPrice</code> 500), and one with a <code>quantity</code> of 3 pointing to a <code>Product</code> (jacket, <code>unitPrice</code> 200).</p>	<p>Here is an example of an instance diagram that would conform to the rules we established in the class diagram. As we said earlier, there can be any number of instance diagrams for one class diagram. The class diagram sets the rules for things like multiplicity, so the instance diagram must conform to those rules.</p> <p>In this case, we see two customers. Alice only has a shopping cart with one line item in it. Bob has a shopping cart with three items. The first item happens to be for the same product as in Alice's order. Five sweaters are being ordered in all: two for Alice and three for Bob. In addition to the sweaters, Bob is ordering two suits and five jackets.</p>

5	<div data-bbox="264 98 764 1100"> <div data-bbox="264 98 764 136" style="background-color: #0070C0; color: white; text-align: center; padding: 5px;">  </div> <div data-bbox="428 142 602 178" style="text-align: center;"> <h2 style="color: #0070C0;">End of Module</h2> </div> <div data-bbox="298 205 730 357"> <ul style="list-style-type: none"> <li>• You should now be able to do the following: <ul style="list-style-type: none"> <li>○ Define basic object-oriented principles: object, message, class, inheritance, and polymorphism.</li> <li>○ Perform OO Analysis with CRC modeling</li> <li>○ Construct UML class diagrams</li> </ul> </li> </ul> </div> </div> <div data-bbox="784 98 1446 1100"> <p>Well, we've finally come to the end of this module. What you should be able to do now is repeated here.</p> <p>You should be able to define basic object-oriented principles such as object, message, class, inheritance and polymorphism.</p> <p>You should be able to do CRC analysis modeling.</p> <p>You should be able to construct a UML class diagram with the information from the CRC cards produced in CRC analysis.</p> <p>In part 2 of our treatment of object oriented analysis and design, we get into dynamic analysis modeling. We will be using UML activity diagrams for this. Then we will change our focus to Object Oriented Design. We will be producing abstractions of the code, using both static models and dynamic models.</p> <p>If you haven't already started it, you may contribute to the discussion forum for this module.</p> <p>Once you are ready, you can take the quiz.</p> <p>And you can now proceed with the team project by doing CRC analysis and creating a UML class diagram.</p> </div>
---	--