Before You Start

Welcome to the Unified Modeling Language Tutorial in 7 days. The goal of this course is to give you the basic knowledge about UML diagrams. It consists of 7 days, 4 days have theory material, and 3 days are practical using of learned theory. The material is divided into parts fitting into one day.

Several lessons were introduced every day. After that a list of answered questions are given to you, and a list of questions and exercises to do by yourself, to strengthen your understanding and put your newfound knowledge in use.

Day 1 gives you an UML basics, introduces you with GRAPPLE and shortly explaines every UML diagram.

DAY 2, DAY 3 and DAY 4 presents ways of creating Class diagrams and Use Case diagrams, State diagrams, Sequence diagrams and Collaboration diagrams, Activity diagrams, Component diagrams and Deployment diagrams respectively. For all of the diagrams a simple example is provided, which will guide you trought the process of creating learned diagram.

DAY 5 and DAY 6 dives into practical use of the UML diagram during the process of modeling a system - the Digital Library. During these two days you'll meet with all the steps in process of modeling a business sytem. Parts of diagrams given in these lessons are not developed completely. This gives you a way of completing them and put you into a process of developing a model for a Digital Library.

In DAY 7 you'll browse the Internet to see what others made about UML teaching. You'll visit some sites that we recommends you, and you may search for new sites.

Learning the lessons, you'll meet with several symbols that have different meaning:

note ✎		This symbol appears in places where new therm or an UML definition is introduced. Text that follows is colored with aquamarine color.
💡		This icon highlights something which is very important and you must make attention to these paragraphs. Text that appear after this symbol is dark red and bold.
e		An example icon. After this one an example is given to you.
UML unified modeling language		After almost every lesson, this symbol stands for most important things that you have learned in that lesson.
⚠		When something may be very dificult or complex for solution, this icon appears. If you get the solution for the given problem be sure that you've a strong knowledge of that area.
answer		This is symbol for the answer. It appear in the Questions & Answers parts of the day.

Lets'get started with our work!

UML Tutorial CONTENTS

In this day, you'll learn about:

- What is Unified Modeling Language
- Modeling of Systems
- Guidelines for Rapid APPLication Engineering (GRAPPLE)
- Components of the UML

Introducing the UML (Unified Modeling Language)

As the world becomes more complex, the computer-based systems that inhabit the world also must increase in complexity. They often involve multiple pieces of hardware and software, networked across great distances, linked to databases that contain mountains of information. If you want to make systems that deal with this, how do you get your hands around the complexity?

The key is to organize the design process in a way that clients, analysts, programmers and other involved in system development can understand and agree on. The UML provides the organization.

Consider this: Would you tell a building contractor that you want a 4 bedroom, 3 bath home, about 2000 square feet - Start building it! We'll hammer out the details as we go along? We all know this is ludicrous. But sadly, this method of development is all too common in the software industry. Just as you would work with an architect to design a blueprint that would diagram exactly how the house is to be built, you will work with us on an UML diagram that will document exactly how your custom software system will be built.

The UML was released in 1997 as a method to diagram software design. It was designed by a consortium of the best minds in object oriented analysis and design. It is by far the most exciting thing to happen to the software industry in recent years. Every other engineering discipline has a standard method of documentation. Electronic engineers have schematic diagrams, architects and mechanical engineers have blueprints and mechanical diagrams. The software industry now has UML.

Before we move on next lesson consider some of the benefits of UML:

1. Your software system is professionally designed and documented before it is coded. You will know exactly what you are getting, in advance.
2. Since system design comes first, reusable code is easily spotted and coded with the highest efficiency. You will have lower development costs.
3. Logic 'holes' can be spotted in the design drawings. Your software will behave as you expect it to. There are fewer surprises.
4. The overall system design will dictate the way the software is developed. The right decisions are made before you are married to poorly written code. Again, your overall costs will be less.
5. UML lets us see the big picture. We can develop more memory and processor efficient systems.
6. When we come back to make modifications to your system, it is much easier to work on a system that has UML documentation. Much less 'relearning' takes place. Your system maintenance costs will be lower.
7. If you should find the need to work with another developer, the UML diagrams will allow them to get up to speed quickly in your custom system. Think of it as a schematic to a radio. How could a tech fix it without it?
8. If we need to communicate with outside contractors or even your own programmers, it is much more efficient.

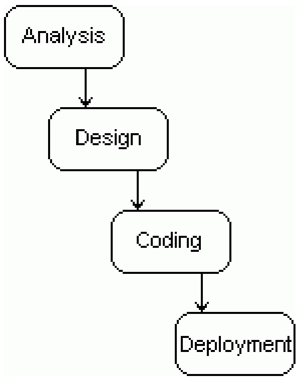
Using The Unified Modeling Language will result in lower overall costs, more reliable and efficient software, and a better relationship with all parties involved. Software documented with UML can be modified much more efficiently. Your software will have a future.

Modeling of systems, old way vs. new way

During this course we will deal with systems, or parts of some larger systems.

System is a combination of software and hardware that provides a solution for a bussines problem.

Process of developing that system involves a lot of people. First of all is the ***client***, the person who has the problem to be solved. An ***analyst*** documents the client's problem and relays it to ***developers***, programmers who build the software that solves the problem, test it and deploy it on computer hardware. This is necessary because systems today are so complex, knowledge has become so specialized that one person can't know all the facets of a bussines, understand the problem, design a solution, translate it into a program, deploy the program onto hardware, and make sure the hardware components all work together correctly.



The waterfall method for modeling of systems

- The old way of system modeling, known as the ***waterfall method***, specifies that analysis, design, coding and deployment follow one another. Only when one is complete can the next one begin. If an analyst hands off analysis to a designer, who hands off a design to a developer, chances are that the three team members will rarely work together and share important insights. Usually the adherents of the waterfall method give coding a big amount of project time, it takes a valuable time away from analysis and design.
- In the new way, contemporary software engineering stress continuing interplay among the stages of development. Analysts and designers, for example, go back and forth to envolve a solid foundation for the programmers. Programmers, in turn, interact with analysts and designers to share their insights, modify designs, and strenghten their code. The advantage is that as understanding grows, the team incorporates new ideas and builds a stronger system.

RAD (Rapid Application Development)

RAD consists of five segments:

Requirements gathering

This segment consists of a several actions. Before moving on to actions, it is important to know that if you don't understand what the client wants, you'll never build the right system.

Discover Bussines Processes: Here analysts gain an understanding of the client's bussines processes, by interviewing the client or a knowledgeable client-designated person and asks the interviewee to go through the relevant process-es step-by-step.
Product: Activity diagram(s).

Perform Domain Analysis: The analyst interviews the client with the gola of understanding the major entities in the client's domain. During the conversation between the client and the anlyst, another team member takes notes. The object modeler listens for nouns and starts by making each noun a class. Ultimatelly, some nouns will become attributes. He or she also listens for verbs, which will become operations of the classes.
Product: High-level class diagram and a set of meeting notes.

Identify Cooperating Systems: Early in the process the development team finds out exactly which systems the new system will depend on, and which systems will depend on it. A system engineer takes care of this action.
Product: Deployment diagram.

Discover System Requirements: In this action team goes through its first Joint Application Development (JAD) session. This session brings together decision makers from client's organization, potential users and the members of the development team. A facilitator moderates the session. The facilitator's job is to elicit from the decision-makers and the users what they want the system to do. At least two team members should be taking notes, and the object modeler should be refining the class diagram derived earlier.
Product: Package diagram.

Present Results to Client: When the team finishes all the Requirements actions, the project manager presents the results to the client.

Analysis

Now the team drills down into the results of the Requirements segment and increases its understanding of the problem. In fact, some of the actions begin during the Requirements segment, while the object modelere begins refining the class diagram.

Understand System Usage: In a JAD session with potential users, the development team works with the users to discover the actors who initiate each use case from the Requirements JAD session, and the actors who benefit from those use cases (Actor can be a system as well as a person).
Product: Use case diagram(s).

Flesh Out Use Cases: Lets continue to work with users. The objective is to analyze the sequence of steps in each use case.
Product: Text description of the steps in each use case diagram.

Refine the Class Diagrams: The object modeler, during the JAD sessions listening all the discussions, refines the class diagram. He or she should be filling in the names of the associations, abstract classes, multiplicities, generalizations and aggregations.
Product: Refined class diagram.

Analyze Changes of State in Objects: Also object modeler refines the model by showing changes of state wherever necessary.
Product: State diagram.

Define the Interactions Among Objects: At this moment development team has a set of use cases and refined class diagram, it's time to define how the objects interact. The object modeler develops a set of diagrams which includes state changes.
Product: Sequence and collaboration diagrams.

Analyze Integration with Cooperating Systems: The system engineer, proceeding in parallel with all the preceding steps, uncovers specific details of the integration with the cooperating systems. What type of communication is involved? What is the network architecture? If the system has to access databases, and if which are the types of databases.
Product: Detailed deployment diagram and if necessary data models.

Design

In this segment, the team works with the results of the Analysis segment to design the solution. Design and Analysys should go back and forth until the design is complete.

Develop and Refine Object Diagrams: Programmers take the class diagram and generate any necessary object diagrams by examing each operation and developing a corresponding activity diagram. This activity diagrams will serve as the basis for much of the coding in the Development segment.
Product: Activity diagrams.

Develop Component Diagrams: In this action programmers also play a major role. The task here is to visualize the components that will result from the next segment and show the dependencies among them.
Product: Component diagrams.

Plan for Deployment: After aiming the component diagrams, the system engineer begins planning for deployment and for integration with cooperating systems. Created diagram shows where the components will reside.
Product: Part of the deployment diagram developed earlier.

Design and Prototype User Interface: This involves another JAD session with the users, continuation of the prior JAD sessions. This is a typical indication of the interplay between Analysis and Design. A GUI analyst works with the users to develop paper prototypes of screen that correspond to groups of use cases.
Product: Screen shots of the screen prototypes.

Design Tests: Preferably, a developer or test specialist from outside the development team uses the uses case diagrams to develop test scripts for automated test tools.
Product: Test scripts.

Begin Documentation: Documentation specialists work with the designers to begin story-boarding the documentation and arriving at a high-level structure for each document.
Product: Document structure.

Development

With enough analysis and design, this segment should go quickly and smoothly. In this segment programmers take over.

Construct Code: With the class, object, activity and component diagrams in hand, programmers construct the code for the system.
Product: The code.

Test Code: This action feeds back into the preceding action and vice versa, until code passes all levels of testing, developed in previous segment (Design Tests).
Product: Test results.

Construct User Interfaces, Connect to Code and Test: Also and this action is a connection between Design and Development. Here a GUI specialist construct aproved interface prototypes and connects them to code. Also and testing the interface ensures that the interfaces work correctly.
Product: Functioning system, complete with user interfaces.

Complete Documentation: During the development segment, documentation experts work in parallel with programmers to complete of all documentation.
Product: System documentation.

Deployment

When development is complete, the system is deployed on the appropriate hardware and integrated with the cooperation systems.

Plan for Backup and Recovery: This action can start long before the development segment begins. The system engineer creates a plan for steps to follow in case the system crashes.
Product: The crash recovery plan.

Install the Finished System on Appropriate Hardware: This step performs the system engineer with any necessary help from the programmers.
Product: Fully deployed system.

Test the Installed System: After installing the software on appropriate computer(s), the development team tests the installed system. Does it perform as it's supposed to? Does the backup and recovery plan work? Results of this tests determine whether further refinement is necessary.
Product: Test results.

Celebrate: When all of the work is finished, the development team may go somewhere to celebrate for their success.

A skeleton of development process is GRAPPLE (Guidelines for Rapid APPLication Engineering), which consist of five segments: Requirements gathering, Analysis, Design, Development and Deployment. Each segment consists of a number of actions, and each action results in a work-product. UML diagrams are work products for many of the actions.

UML Components

The UML certain number of graphical elements combined into diagrams. Because it is a language, the UML has rules for combining these elements.

The purpose of the diagrams is to present multiple views of a system, and this set of multiple views is caled a **model**.

UML model describes what a system is supposed to do. It doesn't tell how to implement the system.

Let's take a brief look of all the UML diagrams. UML consists of nine basic diagrams, but bear in mind that hybrids of theese diagrams are possible.

Class Diagram
Things naturally fall into categories (computers, automobiles, trees...). We refer to these categories as classes.

A class is a category or group of things that have similar attributes and common behaviors.

Class diagram provide the representations used by the developers.
For detailed informations about Class diagrams Click Here!

Object Diagram

An object is an instance of a class - a specific thing that has specific values of the attributes and behavior.

Use Case Diagram
A use case is a description of a system's behavior from a user's standpoint.
For system developers, this is a valuable tool: it's a tried-and-true technique for gathering system requirements from a user's point of view. That's important if the goal is to build a system that real people can use. In graphical representations of use cases a symbol for the actor is used .

The actor is the entity that initiates the use case. It can be a person or another system.

- For detailed informations about Use Case diagrams Click Here!

State Diagram
At any given time, an object is in particular state. State diagrams represent these states, and their changes during time. Every state diagram starts with symbol that represents start state, and ends with symbol for the end state. For example every person can be a newborn, infant, child, adolescent, teenager or adult.
- For detailed informations about State diagrams Click Here!

Sequence Diagram
Class diagrams and object diagrams represent static information. In a functioning system, however, objects interact with one another, and these interactions occur over time. The UML sequence diagram shows the time-based dynamics of the interaction.
- For detailed informations about Sequence diagrams Click Here!

Activity Diagram
The activities that occur within a use case or within an object's behaviour typically occur in a sequence. This sequence is represented with activity diagrams.
- For detailed informations about Activity diagrams Click Here!

Collaboration Diagram
The elements of a system work together to accomplish the system's objectives, and a modeling language must have a way of representing this. The UML collaboration diagram is designed for this purpose.

- For detailed informations about Collaboration diagrams Click Here!

Componenet Diagram
Today in software engineering we have team-based development efforts, where everyone has to work on diferent component. That's important to have a component diagram in modeling process of the system.

- For detailed informations about Component diagrams Click Here!

Deployment Diagram
The UML deployment diagram shows the physical architecture of a computer-based system. It can depict the computers and devices, show their connections with one another, and show the software that sits on each machine.

- For detailed informations about Deployment diagrams Click Here!

System development is a human activity. Without an easy-to-understand notation system, the development process has great potential for error. Consisting of a set of diagrams, the UML provides a standard that enebles the system analyst to build a multifaceted blueprint that's comprehensible to clients, programmers, and everyone involved in the development process. It's necessary to have all these diagrams because each one speaks to a different stakeholder in the system.
The UML model tells what a system is supposed to do. It doesn't tell how.

Questions & Answers

Before reading an answer of every question, try answer it by yourself.

1	**When was UML released, and what is its purpose?** Click for the answer
2	**Give your opinion about the concept - system!** Click for the answer
3	**What is RAD and how much segments it has?** Click for the answer
4	**Count the UML diagrams, and name it!** Click for the answer
5	**What is a class, and what is an object?** Click for the answer

UML was released in 1997 as a method to diagram software design.

This closes the window

System is a combination of software and hardware that provides a solution for a bussines problem

This closes the window

RAD stands for Rapid Application Development, which consist of five segments: Requirements gathering, Analysis, Design, Development and Deployment.

This closes the window

There are nine UML diagrams: class, object, activity, use case, state, sequence, collaboration, component and deployment diagrams

This closes the window

A class is a category or group of things that have similar attributes and common behaviors. An object is an instance of a class - a specific thing that has specific values of the attributes and behavior.

This closes the window

Workshop

Answer these questions by yourself.

1	Why new ways of modeling systems are better than old ways?
2	Name the actions in Analysis segment of GRAPPLE, and give their work-products.
3	In which part(s) of RAD programmers take turn?
4	Why is it necessary to have a variety of diagrams in a model of a system?
5	Which diagrams provide a static view of a system?
6	Which diagrams provide a dynamic view (show change over time)?

UML Tutorial

CONTENTS

In this day, you'll learn about:

- **Class Diagrams**
- Visualising a class
- Associations between classes (aggregation, inheritance and generalization, interfaces and realizations)
- Visibility of attributes and operations
- **Use Case Diagrams**
- What is a use case model
- Relationships between use cases

Class Diagrams

During this several days you'll firm up your knowledge of object-orientation as you learn more about the UML.

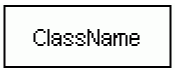
Next lessons will deal with class diagrams. How to extract classes, their attributes and methods from interviewing a client, and then bind the classes with relationships to form the class diagram. You'll learn more about:

The last part of this group of lessons consits of an example, which give you all the necessary steps for building a class diagram.

Visualising a Class

We mentioned in earlier lessons that things fall into categories - classes. The UML class diagram consists of several classes conected with relationships. But how UML represents a class?

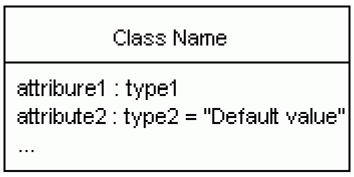
The rectangle is the icon for the class. The name of the class is, by convention, a word with an initial uppercase letter. It appears near the top of the rectangle. If your class name has more than one word name, then join the words together and capitalize the first letter of the every word.



Class name is written at the top of the rectangle

An **attribute** is a property of a class. It describes a range of values that the property may hold in objects of the class. A class may have zero or more attributes.

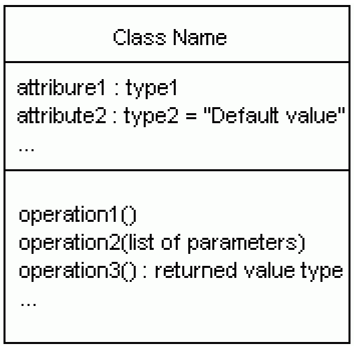
A one-word attribute name is written in lowercase letter. If the name consists of more than one word, the words are joined and each word other than the first word begins with an uppercase letter. The list of attribute names begins below a line separating them from the class name.



In the class icon, you can specify a type for each attribute's value (string, floating-point, number, integer, boolean or user defined types). Also you can indicate a default value for an attribute.

An **operation** is something that a class can do, or that you (or naother class) can do to a class.

Like an attribute name, an operation's name is written in lowercase letter. If the name consists of more than one word, the words are joined and each word except the first word begins with an uppercase letter. The list of operations begins below a line separating operations from the attributes.


Class Name
attribure1 : type1
attribute2 : type2 = "Default value"
...
operation1()
operation2(list of parameters)
operation3() : returned value type
...

Also operations may have some aditional information. In the parentheses that follow an operation name, you can show the parameter that the operation works on, along with the parameter's type. If the operation is function, then we must specify the type of the returned value.

Other additional features that can be attached to attributes of the class are constraints and notes.
Constraints are free form text enclosed in braces. The backeted text specifies one or more rules the class follows.
Notes usually are attached to attributes as well as operations, and they give aditional information to a class. A note can contain a graphic as well as text.

How can we derives classes from interviewing the clients?

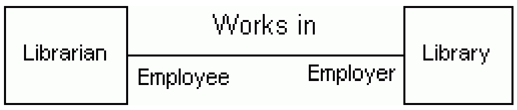
In conversations with clients, be alert to the nouns they use to describe the entities in their business. Those nouns will become the classes in your model. Be alert also to the verbs that you hear, because these will constitute the

operations in those classes. The attributes will emerge as nouns related to the class nouns.

Associations

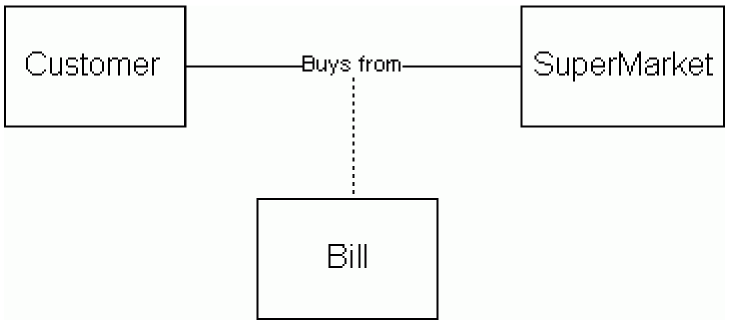
When classes are connected together conceptually, that connection is called an association.

You visualize the association as a line connecting the two classes, with the name of the association just above the line.



When one class associates with another, each one usually plays a role within that association. You can show those roles on the diagram by writing them near the line next to the class that plays the role.

Association may be more complex than just one class connected to another. Several classes can connect to one class.
Sometimes an association between two classes has to follow a rule. You indicate that rule by putting a constraint near the association line.



Just like a class, an association can have attributes and operations. In this case we have an association class.

You visualize association class the same way you show a regular class, and you use dotted line to connect it to the association line.

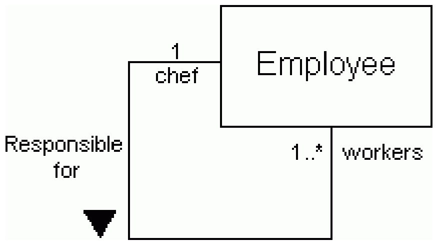
Multiplicity is a special type of association which shows the number of objects from one class that relate with a number of objects in an associated class.

One class can be relate to another in a

- one-to-one
- one-to-many
- one-to-one or more
- one-to-zero or one
- one-to-a bounded interval (one-to-two through twenty)
- one-to-exactly n
- one-to-a set of choices (one-to-five or eight)

The UML uses an asterisk (*) to represent ***more*** and to represent ***many***.

Sometimes, a class is in association with itself. This can happen when a class has objects that can play a variety of roles. These associations are called **reflexive associations**.



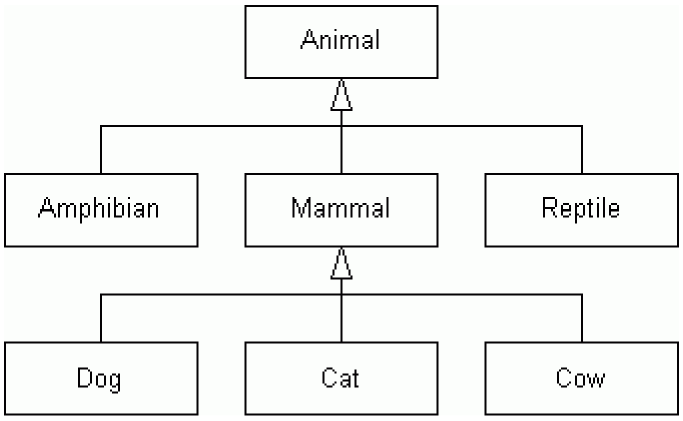
Reflexive association

Without relationships, a class model would be a list of rectangles that represent a vocabulary of system. Relationships show how the terms in the vocabulary connect with one another to provide a picture of the slice of the world you're modeling. The association is the fundamental conceptual connection between classes. Each class in an association plays a role, and multiplicity specifies how many objects in one class relate to one object in the associated class. Many types of associates are possible.

Inheritance & Generalization

If you know something about a category of things, you automatically know some things you can transfer to other categories.

If you know something is an animal, you take for granted that it eats, sleeps, has a way of being born, has a way of getting from one place to another... But imagine that mammals, amphibians and reptiles are all animals. Also cows, dogs, cats... are grouped in category mammals. Object-orientation refers to this as **inheritance**.



An inheritance hierarchy in the animal kingdom

One class (the child class or subclass) can inherit attributes and operations from another (the parent class or superclass). The parent class is more general then the child class.
In generalization, a child is substitutable for a parent. That is, anywhere the parent appears, the child may appear. The reverse isn't true, however.

In UML inheritance is represented with a line that connects the parent to a child class, and on the parent's side you put an open triangle.

If we look from the association's side, the inheritance stands for *is a kind of* association.

What should analysts do to discover inheritance?
The analyst has to realize that the attributes and operations of one class are general and apply to perhaps several other classes - which may add attributes and operations of their own. Another possibility is that the analyst notes that two or more classes have a number of common attributes and operations.

Classes that provide no objects are said to be abstract classes. You indicate an abstract class by writing its name in italics.

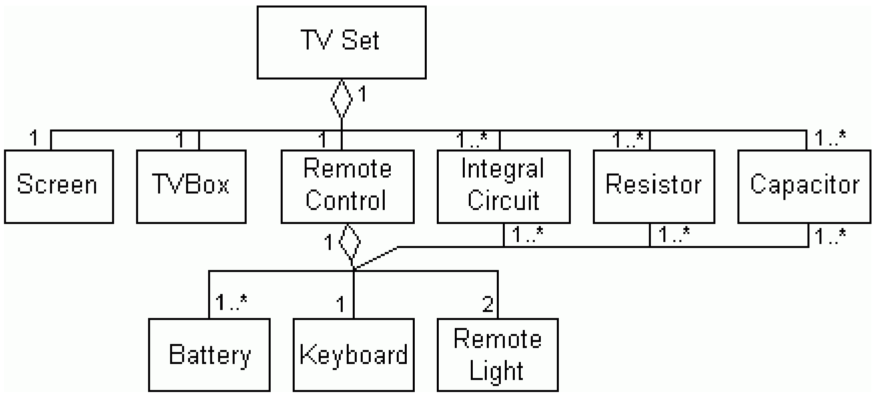
A class can inherit attributes and opeartions from another class. The inheriting class is the child of the parent class it inherits from. Abstract classes are intended only as bases for inheritance and provide no objects of their own.

Aggregations

Sometimes a class consists of a number of component classes. This is a special type of relationship called aggregation. The components and the class they constitute are in a part-whole association.

Aggregation is represented as a hierarchy with the "whole" class at the top, and the component below. A line joins a whole to a component with an open diamond on the line near the whole.

Let's take a look at the parts consisting a TV set. Every TV has a TV box, screen, speaker(s), resistors, capatitors, transistors, ICs... and possibly a remote control. Remote control can have these parts: resistors, capatitors, transistors, ICs, battery, keyboard and remote lights.

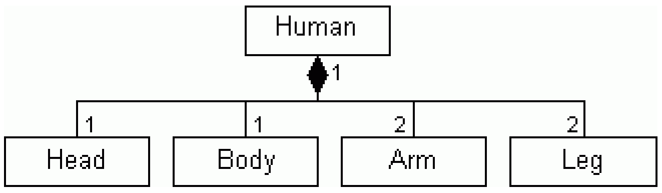


An aggregation association in the TV Set system

Sometimes the set of possible components in an aggregation falls into an OR relationship. To model this, you would use a constraint - the word OR within braces on a dotted line that connects the two part-whole lines.

A composite is a strong type of aggregation. Each component in a composite can belong to just one whole. The symbol for a composite is the same as the symbol for an aggregation except the diamond is filled.

If you examine the human's outside you'll find out that every person has: head, body, arms and legs. This is shown on this picture.



A composite association. In this association each component belongs to exactly one whole.

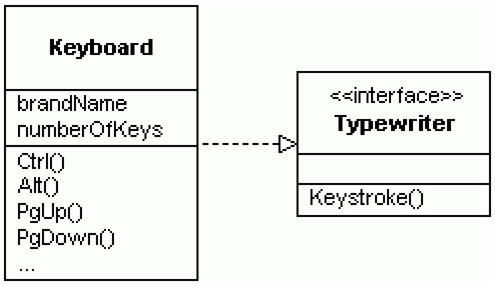
An aggregation specifies a part-whole association. A "whole" class is made up of component classes. A composite is a strong form of aggregation, and a component in a composite can be part of only one whole. Aggregations and composites are represented as lines joining the whole and the component with open and filled diamond, respectively, on the whole side.

Interfaces and Realizations

In previous lessons we learned how to refine classes from the interview with client, and relate them with different relationships between them. But it's possible that some of classes are not related to a particular parent, but their behaviors might include some of the same operations with the same signatures. You can code the operations for one of the classes and reuse them in the others.

An interface is a set of operations that specifies some aspect of a class behaviour, and it's a set of operations a class presents to other classes.
You model an interface the same way you model a class, with the rectangle icon, but interface has no attributes, only operations. Another way is with a small circle joined with line to a class.

The computer's keyboard is a reusable interface. Its keystroke operation has been reused from the typewriter. The placement of keys is the same as on a typewriter, but the main point is that the keystroke operation has been transfered from one system to another. Also on computer's keyboard you'll find a number of operations that you won't find on a typewriter (Ctrl, Alt, PageUp, PageDown...)



An interface is a collection of operations that a class carries out

To distinguish interfaces from classes, in stereotype construct we put *<<interface>>* or *I* at the begining of the name of any interface.

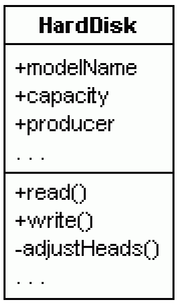
The relationship between a class and an interface is called realization. This relationship is modeled as a dashed line with a large open triangle adjoining and pointing to the interface.

Visibility

Visibility applies to attributes or operations, and specifies the extent to which other classes can use a given class's attributes or operations.

Three levels of visibility are possible (last symbols are used in UML classes to indicate different levels of visibility):

public level	usability extends to other classes	+
protected level	usability is open only to classes that inherit from original class	#
private level	only the original class can use the attribute or operation	-



Public and private operations in a HardDisk

Class Diagram - Example

Here is a brief descrition for a writing a text document:

Suppose that you're writing a document in some of famous text processing tools, like MicrosoftWord for example. You can start typeing a new document, or open an existing one. You type a text by using your keyboard.

Every document consists of several pages, and every page consists of header, document's body or/and footer. In header and footer you may add date, time, page number, file location e.t.c.

Document's body has sentences. Sentences are made up of words and punctual signs. Words consists of letters, numbers and/or special characters. Also in the text you may insert pictures and tables. Table consists of rows and columns. Every cell from table may hold up text or pictures.

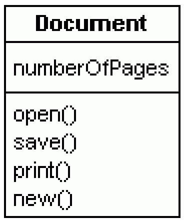
After finishing the document, user can choose to save or to print the document.

This is simplified explanation of creating a text document. If we extract the list of nouns form previous text, a following list can be achieved:

document, text processing tool, MicrosoftWord, text, keyboard, header, footer, document's body, date, time, page number, location of file, page, sentence, word, punctual sign, letter, number, special character, picture, table, row, column, cell, user

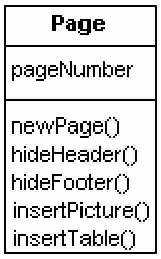
Nouns colored in red are candidate classes, and candidate attributes for our model.

Let's start with the document. As you can see this example deals with documents, thus a document will be the central class in our class diagram. Document has a several pages, therefore a ***numberOfPages*** will be one of the attributes for the **Document class**. For the operations we have: ***open()***, ***save()***, ***print()*** and ***new()***. Every document consists of pages. The Page will be also a candidate for the class.



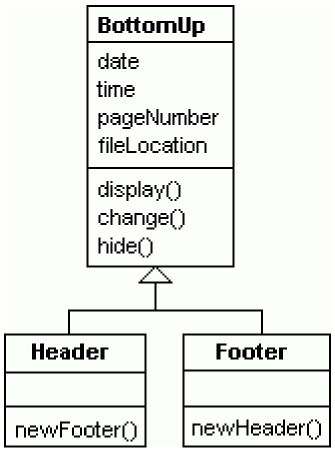
The Document class

The **Page class** will hold ***pageNumber*** as an attribute, and operations allowed here can be: ***newPage()***, ***hideHeader()*** and ***hideFooter()***. Operations for the header and footer tells us that the Header and The Footer can be also a classes.



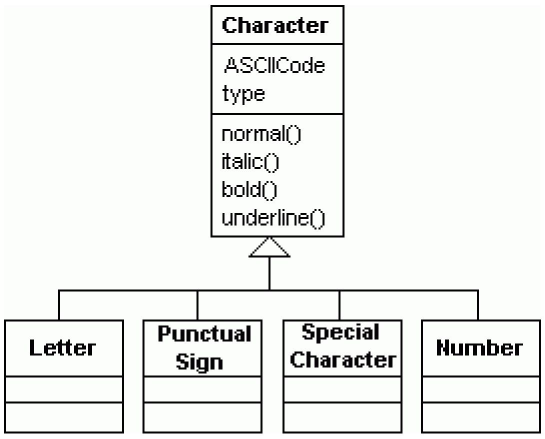
The Page class

The **Header class** and the **Footer class** has common attributes: ***date***, ***time***, ***pageNumber*** and ***fileLocation***. These attributes are optional for every header or footer and user may configure them. This will guide us that a common class can be introduced. This will be a good time to make an inheritance. Parent class will be **BottomUp** (this name is chosen because headers and footer appear in upper and bottom parts of every page) and will hold common attributes for header and footer, and these operations: ***display()***, ***hide()*** and ***change()***. Header and Footer classes (children of this class) will have only operations: ***newHeader()*** and ***newFooter()***.



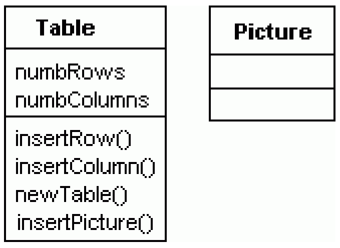
The BottomUp class and it's children

Before examining the document's body or text, lets take a look at the makeing of text actions. Document's text is made up of sentences. Sentences are made up of words and words are made up of characters. If words are array of characters and sentence is array of words, then a sentence is also an array of characters. Therefore a document's body can be an array of characters. For this purpose to make a document's text we'll use the Character class with its children. The **Character class** wil have ***ASCIIcode*** and ***type*** as attributes (type tells the type of the character - normal, italic, bold or underline), and ***normal()***, ***bold()***, ***italic()*** and ***underline()*** as operations. The Character class childrens will be: **Letter**, **PunctualSign**, **SpecialCharacter** and **Number**. Also in the document's body can be found tables and pictures. These are new classes in our class diagram.



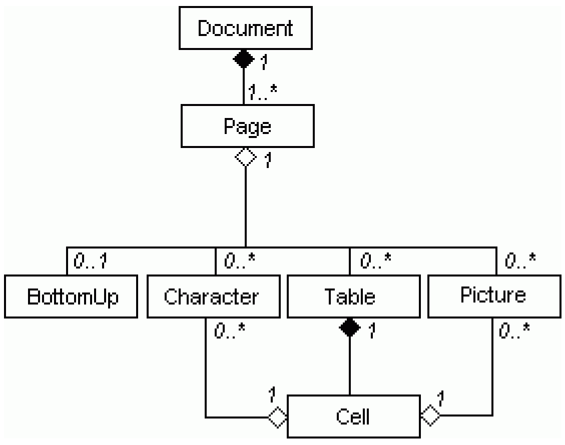
The character hierarchy

The **Table class** has ***numbRows*** and ***numbColumns*** as attributes and ***newRow()***, ***newColumn()*** and ***newTable()*** as operations. Every table consists of one or more cells. And in every cell, text or pictures can be placed.



The Table class and the Picture class

Haveing all this classes in front of us we can draw out the associations between them and gain the completed class diagram for this problem.



The class diagram for the makeing of text document

Given class diagram is not finished in detail. If you know more about typeing a document, then add information you know to the given diagram. This model can grow and grow, and makeing all the necessary steps of analysis and design you may reach the point, where a new text processing tool is modeled.

Use Case Diagrams

Class diagrams provide a static view of the classes in a system. Next diagrams provide a dynamic view and show how the system and its classes change over time.

The static view helps an analyst communicate with a client. The dynamic view, helps an analyst communicate with a team of developers, and helps the developers create programs.

Just as the class diagram is a great way to stimulate a client to talk about a system from his or her viewpoint, the use case is an excellent tool for stimulating potential users to talk about a system from their own viewpoints. It's not always easy for users to articulate how they intend to use a system. It's a fact of life that users often know more than they can articulate: The use case helps break the ice.

Interviews with users begin in the terminology of the domain, but should then shift into the terminology of the users. The initial results of the interviews should reveal actors and high-level use cases that describe functional requirement in general terms. this information provides the boundaries and scope of the system.

Later interviews with users delve into these requirement more closely, resulting in use case models that show the scenarios and sequences in detail. This might result in additional use cases that satisfy inclusion and extension relationships. In this phase it is important to your understanding of the domain, because if you don't understand it well you may create too much use cases and that could impede the analysis process.

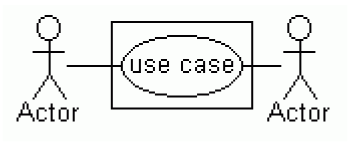
Use case are collection of scenarios about system use. Each scenario describes a sequence of events. Each sequence is initiated by a person, another system, a piece of hardware, or by the passage of time. Entities that initiate sequences are called ***actors***. The result of the sequence has to be something of use either to the actor who initiated it or to another actor.

It's possible to reuse use cases. One way - **inclusion**, is to use the steps from one use case as part of the sequence of steps in another use case. Another way - **extension**, is to create a new use case by adding steps to an existing use case.

Next lessons will guide you through representing a use case model and visualizing relationships among use cases. At the end an use case example will be given to you.

Introducing a Use Case Model

An actor initiates a use case, and an actor (possibly the initiator, but not necessarily) receives something of value from the use case.



Use Case model

An ellipse represents a use case, a stick figure represents an actor. The initiating actor is on the left of the use case, and the receiving actor is on the right. The actor's name appears just below the actor. The name of the use case appears either inside the ellipse or just below it. An association line connects an actor to the use case, and represents communication between the actor and the use case. The association line is solid, like the line that connects associated classes.

Each use case is a list of scenarios, and each scenario is a sequence of steps. Each scenario of each use case will also have its own page, listing in text from the:

- Actor who initiates the use case
- Preconditions for the use case
- Steps in the scenario
- Postconditions when the scenario is complete
- Actor who benefits from the use case

Use case diagrams add more power to the requirements gathering. They visualize use cases, they facilitate communication between analysts and users and between analysts and clients. In a use case diagram, symbol for the use case is an ellipse, actor has a stick figure as a symbol, and association line joins an actor to a use case. The use cases are usually inside a rectangle that represents the system boundary.

Relationships Among Use Cases

In lesson Use case diagrams, we mentioned that it is possible to reuse use case diagrams. One way was inclusion, another was extension. These are relationships among use cases, and there are two other kinds of relationships: generalization and grouping.

Inclusion
Inclusion enables you to reuse one use case's steps inside another use case.

To represent inclusion, you use the symbol you used for dependency between classes - dotted line connecting the classes with an arrowhead pointing to the depended-on class. Jus above the line, you add a stereotype-the word <<include>> enclosed in guillements.

Extension
Extension allows you to create a new use case by adding steps to an existing use case.

Also here a dotted arrowhead line is used to represent extension, along with a stereotype that shows <<extends>> in guillements. Within the base use case, the extension point appears below the name of the use case.

Generalization
Classes can inherit from one another and so can use cases. In use case inheritance, the child use case inherits behavior and meaning from the parent, and adds its own behavior. You can apply the child wherever you apply the parent.

Generalization is modeled the same way you model class generalization - with a solid line that has an open triangle pointing at the parent.

The generalization relationship can exist between actors as well as use cases.

Grouping
In some use case diagrams, you might have a multiple of use cases and you'll want to organize them. This could happen when a system consists of a number of subsystems.
The most straightforward way is to organize group related use cases into a ***package*** (a tabbed folder).

In next lesson Use Case Diagrams - Example we'll look closely to all of these relationships, to help you understand use cases as a part of use case diagrams.

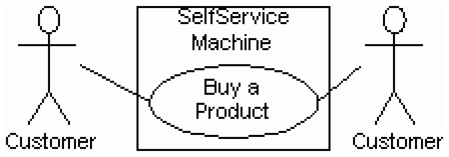
Use Case Diagram - Example

In this example we're going to model out use case diagrams for a self-service machine.

The main functions of self-service machine is to allow a customer to by a product(s) from the machine (candy, chocolate, juice...). Every user that you asked for a set of scenarios happening during usage of machine, can tell you that main use case can be labeled as "Buy a product". Let's examine every possible scenario in this use case. These scenarios would be revealed trough conversations with users.

The "Buy a product" use case

The actor in this use case is customer. This customer wants to buy some of the products offered by the self-service machine. First of all he/she inserts money into the machine, selects one or more products, and machine presents a selected product(s) to the customer. Use case diagram for this scenario can be represented as:



The ***Buy a Product*** *use case diagram*

But if we think a little, others scenarios immediatelly come to mind. It's possible that the self-service machine is out of one or more products, or the machine hasn't the exact amount of money to return to customer.

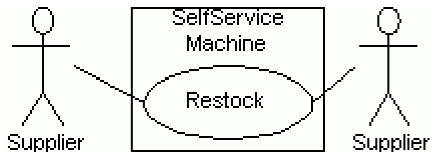
-Are we supposed to handle with these scenarios?

If we are, then let's return at the moment when the customer inserts money into machine and enters his or hers selection. After this imagine that machine is out of brand. In this case it's preferable to present a message to the customer that machine is out of brand and allow him or her to make another selection or return money back. If incorrect-ammount-of-money scenario has hapenned, then self-service machine is supposed to return original ammount of money to the customer.
The precondition here is hungry or thirsty customer, and postcondition is either product from the machine or the returned money.

This is use case scenario from one user's viewpoint (the customer). Buth there are other users too. A supplier has to restock the machine, and a collector has to collect the accumulated money from the machine. This tells us to create at least two more use cases: "Restock" and "Collect money". Let's look closely at both of them, by interviewing both suppliers and collectors.

The "Restock" use case

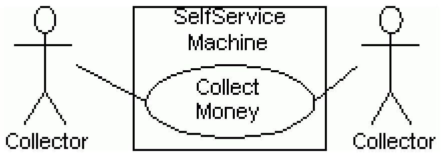
Actions that supplier must do every time interval (say, one or two weeks) are: Supplier unsecures the machine, opens the front of the machine, and fills each brand's compartment to capacity. (The supplier may fill each brand according to consuming of article). Then he/ she closes the front of the machine and secures it.
The precondition is the passage of the interval, and the postcondition is that the supplier has a new set of potential sales.
This use case diagram is presented on the following picture:



The ***Restock*** *use case diagram*

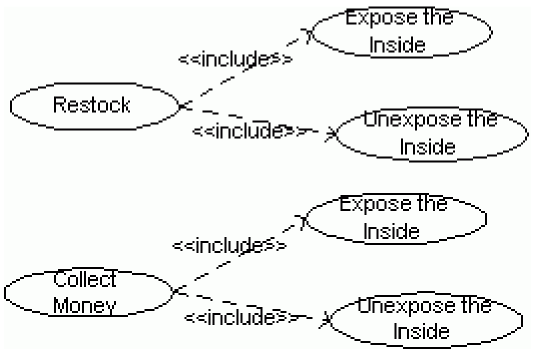
The "Collect Money" use case

Responsible for this use case is collector. Collector may be the same person as the supplier. The steps that collector must do are same as the steps of supplier, but the collector don't deal with products, he/ she deals with money. When the time interval has passed this person collects the necessary ammount of money from the machine.
The postcondition here is the money in hands of the collector.
Collect money use case diagram is following:



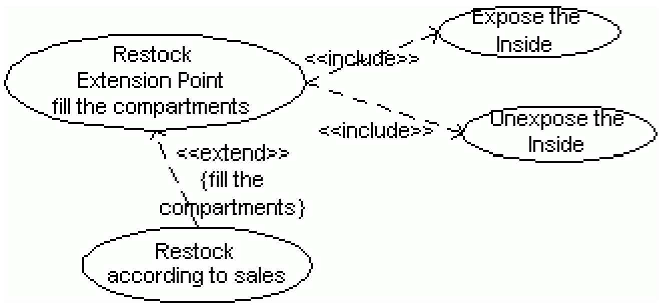
*The **Collect Money** use case diagram*

The steps of unsecuring the machine, front opening, closing and securing the machine are the same that supplier and collector must do. This is a good place to include a use case. Let's combine the "unsecure" and "pull open" steps into use case called "Expose the inside" and the "close machine" and "secure" with use case "Unexpose the inside".
By including a use case Restock and Collect use cases may look as this:



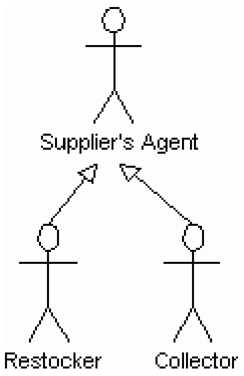
The inclusion process in Restock and Collect Money use cases

The Restock use case could be basis of another use case: "Restock according to sales". Here supplier may fill up brands with new products according of sale of that products. This is an extension of a use case.
After inclusion and extension the Restock use case can be:



The extension process in Restock use case

Also and generalization may take place to the actors supplier and collector. If both of them are the same person, let's say Supplier Agent, then the restocker and the supplier are both children of the Supplier Agent. This is showh on next picture:



A generalization between supplier and collector

We reached the end of our analysis for the self-service machine. Completed use case diagram (Click for the entirely use case diagram) shows the functional requirements of the system, and we are able to make next move - design and development.

Questions & Answers

If you learned all the lessons included in this day, now is the perfect time to test and strengthen your knowledge about class and use case diagrams as a part of the UML.

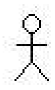
1	**How do you represent a class in the UML?** Click for the answer
2	**What is an abstract class?** Click for the answer
3	**What do you call the entity that initiates a use case?** Click for the answer
4	**What are the similarities between classes and use cases, and what are the differenties?** Click for the answer
5	**Name two advantages to visualizing a use case.** Click for the answer

Rectangle is a class symbol. You put class name at the top of it, list of attributes below, and on the bottom you put list of operations.

This closes the window

An abstract class serves as a basis for the inheritance but provides no objects.

This closes the window



The entity that initiates a use case is called an Actor.

This closes the window

Similiraties are: Both are structural elements. both can inherit.
Differencies: The class consists of attributes and operations.The use case consists of scenarios, and each scenario consists of a sequence of steps.The class diagram provides a static view of the parts of the system, the use case provides a dynamic view of behavior.The class shows the inside of the system. The use case shows how the system looks to an outsider.

This closes the window

With visualization, you can
(1) show use cases to users and get them to tell you additional information;
(2) combine them with other kinds of diagrams.

This closes the window

Workshop

These questions and exercises you must do by yourself.

1	What information can you show on a class icon, and why sometimes is necessary to add a constraint?
2	How do you discover inheritance?
3	What is a difference between an aggregation and a composite? Give an example for both of them.
4	Name the three levels of visibility and describe what each one means.
5	Is a use case the same as a scenario?
6	Sketch the diagram of a use case model for a VCR remote control. Be sure to include all the functions of the remote as use cases for your model. (Do not use controls for changing the chanells)
7 ⚠	Give yourself a possible interview with a client. Try to extract classes, attributes and operations from that conversation. After that try to relate them and make a class diagram.

UML Tutorial

In this day, you'll learn about:

- **State Diagrams**
- Transitions between states
- **Sequence Diagrams**
- Ways of creating sequences
- **Collaboration Diagrams**
- Writing collaboration diagrams

State Diagrams

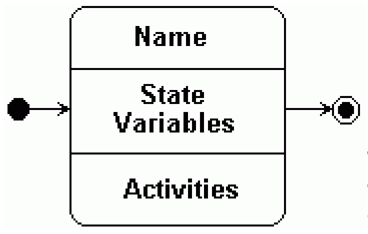
In next lessons you'll work with elements that you haven't worked before. That would be ***behavioral elements***, that show how parts of a UML model change over time. As the system interacts with users and possibly with other systems, the objects that make up the system go through necessary changes to acommodate the interactions. If you're going to model systems, you must have a mechanism to model change. That mechanism in UML is State diagrams.

- When you pull a switch, a light changes its state from off to on.
- When you make keystroke or mousemovement in screen saver mode on your computer, your computer leaves screen saver mode and returns to working - active mode.

The UML State diagram presents the states an object can be in along with the transitions between the states, and shows the starting point and endpoint of a sequence of state changes.

A state diagram shows the states of a single object.

States in the state diagram are represented with a rounded rectangle, and the symbol for the transition is a solid arrowhead line. A solid circle stands for starting point of a sequence of states, and bull's eye represents the endpoint.



The UML symbols in a state diagram. The top area stands for the state's name, middle area is for state variables, and the bottom is for activities. Transitions are represented with solid arrowhead lines, and circles are for starting and ending state.

Also you can add details to a state icon by dividing it to a three areas. The top area holds the name of the state (this name you must supply whatever the class is divided or not), the middle area hold state variables (timers, counters, dates...), and the bottom area hold activities (***entry***-what happenes when the system enters the state, ***exit***-what happenes when the system leaves the state, ***do***-what happenes when the system is in the state).

State Details and Transitions

You can indicate an event that causes a transition to occur (***a trigger event***) and the computation (***the action***) that executes and makes the state change happen.

To add events and actions you write them near the transition line, using a slash to separate a triggering event from an action.

Sometimes an event causes a transition without an associated action, and sometimes a transition occurs because a state completes an activity (rather than because of an event). This type of transition is caled ***triggerless transition***.
Also possible is a ***guard condition***. When it's met, the transition takes place. (For example screen saving mode is activated when time interval of n idle minutes has passed).

Some of the states may be more complex that is represented with a rounded rectangle. It is possible to have states inside one state. These states are called ***substates*** (sequential and concurrent).

Sequential substates come in sequences of states which occur one after another. Concurrent substates must consist of two or more sequential substates which occur at the same time. This is represented with dotted line between the concurent states.

If you have any problems understanding these concepts please refer to the State Diagram - Example lesson.

The UML supplies a symbol that shows that a composite state remembers its active substate when the object transitions out of the composite state. The symbol is Ⓗ connected by a solid line to the remembered substate, with an arrowhead that points to the substate.

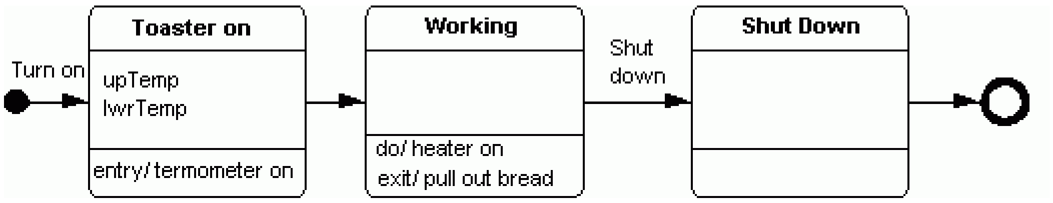
A message that triggers a transition in the receiving object's state diagram is called a ***signal***. In the object-oriented world, sending a signal is the same as creating a signal object and transmitting it to the receiving object. The signal object has properties that are represented as attributes. Because a signal is an object, it's possible to create inheritance hierarchies of signals.

State Diagram - Example

Suppose you're designing a toaster. You would build a plenty of UML diagrams, but here only state diagrams will be of our interest.

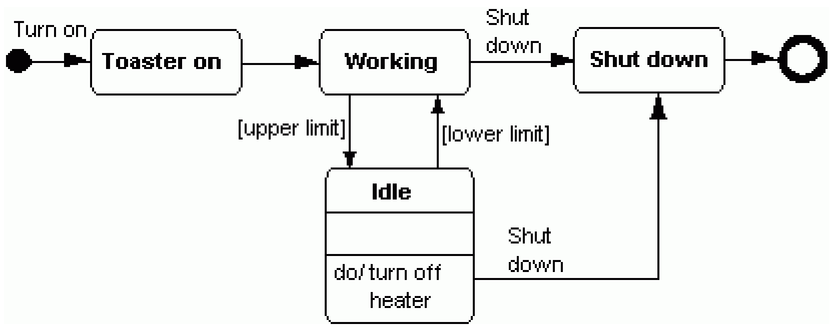
- What are the steps of making a toast?

First of all we must turn on the toaster, put in the bread and wait for several minutes to bake it. The initial state diagram is shown below:



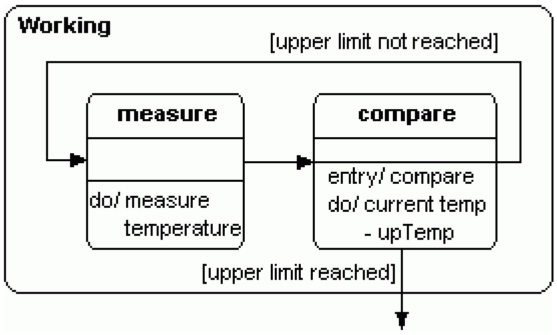
The initial state diagram for making a toast

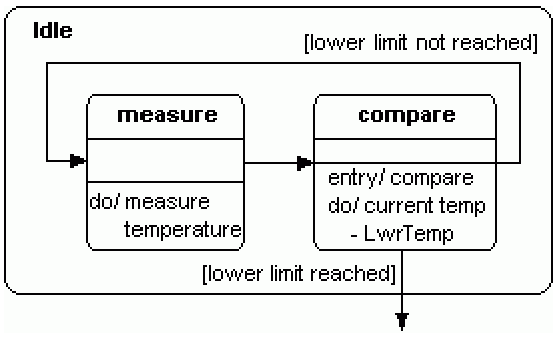
But this is not the final state diagram. To prevent burning out the bread, heater of the toaster must produce heat in temperature interval (upper and lower temperature limits). For this purpose thermometer measures the temperature of heater, and when the upper limit of temperature is reached then heater must go into idle state. This state resists until heater's temperature decreases to lower limit, and then working state is again aimed. With this new state, extended state diagram will be:



The extended state diagram for making a toast

Transition between working and idle state is not presented in details. To do this substates must be added.





*Substates in **Working** and **Idle** states*

Substates in working and idle states are very similar. both had measure and compare states, but differentiates in process of temperature comparison. Working state must compare current temperature with upper temperature limit (if it is reached, working state goes into idle state), and idle state compares current temperature with lower temperature limit (idle state is replaced with working state when temperature falls under lower limit).

It's necessary to have state diagrams because they help analysts, designers and developers understand the behavior of the objects in a system. Developers, in particular, have to know how objects are supposed to behave because they have to implement these behaviors in software. It's not enough to implement an object: Developers have to make that object do something.

Sequence Diagrams

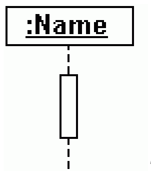
In previous lesson we learned state diagrams, which focus on the states of na object. But it's necessary to have communication between objects. The UML sequence diagram goes the next step and shows how objects communicate with one another over time. In this expanded field of view, you'll include an important dimension: ***time***. The key idea here is that interactions among objects take place in a specified sequence, and the sequence takes time to go from beginning to end.

The sequence diagram consists of ***objects*** represented in the usual way - as named rectangles (with the name underlined), ***messages*** represented as solid-line arroes, and ***time*** represented as a vertical progression.

Let's take a close look at this parts of sequence diagrams.

Objects

The objects are laid out near the top of the diagram from left to right. They're arranged in any order that simplifies the diagram. Extending downforward from each object is a dashed line called the object's ***lifeline***. Along the lifeline is narrow rectangle called an ***activation***. The activation represents an execution of an operation the object carries out. The length of the rectangle signifies the activation's duration. This is shown on next picture.



Representing an object in a sequence diagram

Messages

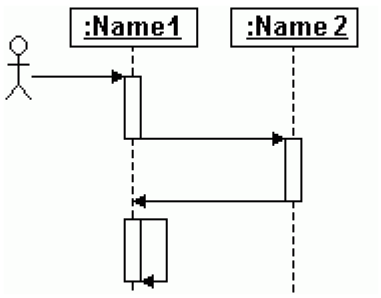
A message that goes from one object to another goes from one object's lifeline to the other object's lifeline. An object can send a message to itself-that is, from its lifeline back to its own lifeline. This is called ***recursion***. See recursion example.

Following table lists all of possible messages existing in the UML sequence diagrams, with their graphical representations:

simple	This is a transfer of control from one object to another.	→
synchronous	If an object sends a synchronous message, it waits for an answer to that message before it proceeds with its business.	➡
asynchronous	If an object sends an asynchronous message, it doesn't wait for an answer before it proceeds.	⇀

Time

The diagram represents time in the vertical direction. Time starts at the top and progresses toward the bottom. A message that's closer to the top occurs earlier in time than a message that's closer to the bottom.



The essential symbol set of the sequence diagram, with the symbols working together.

The actor-symbol initiates the sequence, but the stick figure isn't part of the sequence diagram symbol set.

In a sequence diagram, the objects are laid out from left to right across the top. Each object' lifeline is a dashed line extending downward from the object. A solid line with an arrowhead connects one lifeline to another, and represents a message from one object to another. Time starts at the top an proceeds downward. Although an actor typically initiates the sequence, the actor symbol isn't part of the sequence diagram's symbol set.

Ways of Creating Sequences

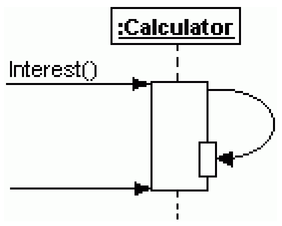
As use case diagram can show either an instance (one scenario) of a use case, or it can be generic and incorporate all of the use case's scenarios, and sequence diagrams can be ***instance*** or ***generic*** sequence diagrams. Generic sequence diagrams often provide opportunities to represent ***if*** statements and ***while*** loops. Enclose each condition for an "if" statement in square brackets. Do the same for the condition that satisfies a "while" loop, and prefix the left bracket with an asterisk. See Sequence Diagram - Example.

It's possible in sequence diagrams to create an object. When this occurs, you represent a created object in the usual way - as a named rectangle.

The difference is that you don't position it at the top of the sequnce diagram as you do with the other objects. Instead, you position it along the vertical dimension so that its location corresponds to the time when it's created. The message that creates the object is labeled ***Create()***.

Sometimes an object has an operation that invokes itself. This is called ***recursion***.

Suppose one of the objects in your system is a calculator, and suppose one of its operations computes interest. In order to compute compound interest for a timeframe that encompasses several compouding periods, the object's interest-computation operation has to invoke itself a number of times. Sequence diagram for this is following:



Representing recursion in a sequence diagram

Sequence Diagram - Example

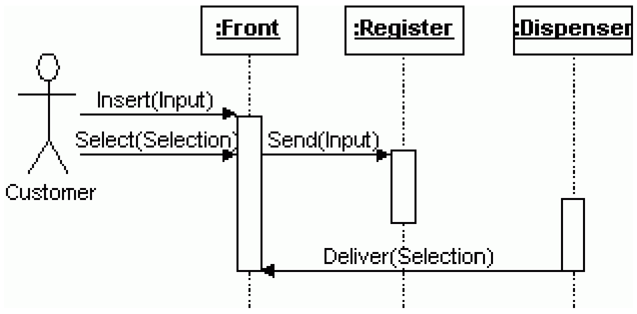
In this example, a sequence diagram for one of the privious examples - Self-service machine will be given. See Use Case Diagram - Example.

Let's assume that in the self-service machine, three objects do the work we're concerned with:

▸ **the front**	the interface the self-service machine presents to the customer
▸ **the money register**	part of the machine where moneys are collected
▸ **the dispenser**	which delivers the selected product to the customer

The *instance sequence diagram* may be sketched by using this sequences:

1 The customer inserts money in the money slot
2 The customer makes a selection
3 The money travels to the register
4 The register checks to see whether the selected product is in the dispenser
5 The register updates its cash reserve
6 The register has a dispenser deliver the product to the front of the machine

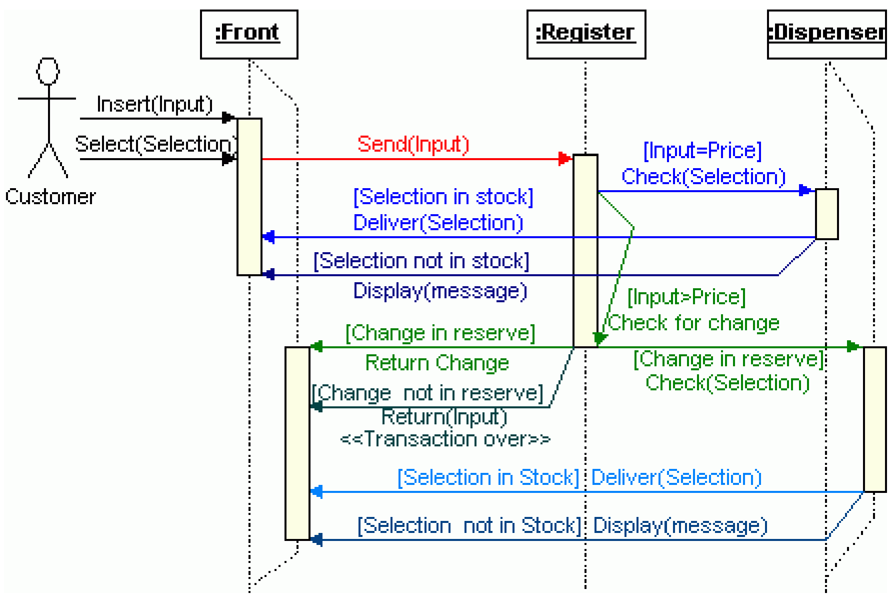


The "Buy a product" scenario. Because this is the best-case scenario, it's an instance sequence diagram

But, when use case diagrams were developed, two additional scenarios were introduced to represent out-of-product case and incorrect-amount-of-money case. If you consider all of a use case's scenarios when you draw a sequence diagram, you create a *generic sequence diagram*.

The out-of-product scenario will produce this sequences:

1 After selecting a sold-out brand, the "SOLD OUT" message flashes
2 Prompt for another selection must be displayed
3 The customer has the option of pushing a button that returns money
4 If the customer selects an in-stock brand, everything proceeds as in the best-case scenario if the input amount is correct. If not, the machine follows the incorrect-amount-of-money scenario
5 If the customer selects another sold-out brand, the process repeats until the customer selects an in-stock brand or pushes a button that returns his or her money

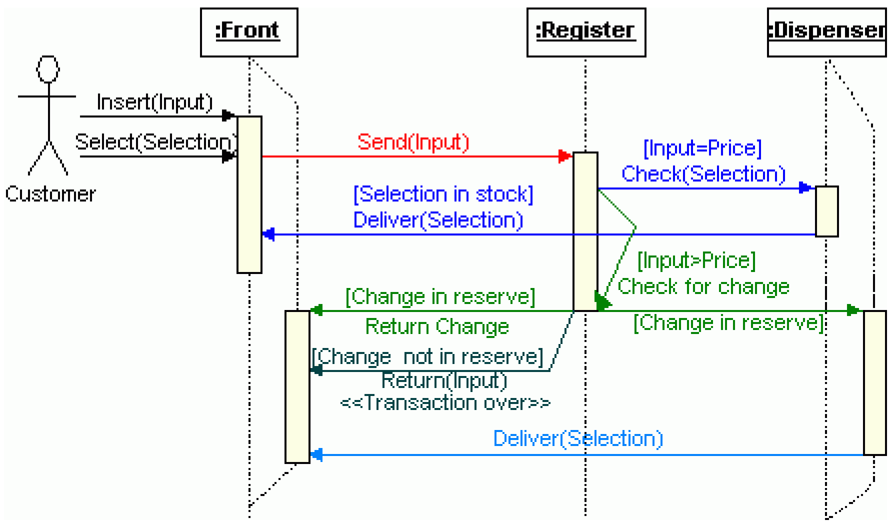


The "Out of product" scenario

Make attention on "fork" of control in the message, caused by if conditions. Because each path goes to the same object, the fork causes a "branch" of control in the receiving object's lifeline, separating the lifeline into separate paths. At some point in the sequence, the branches in the message merge, as do the forks in the lifeline.

For the incorrect-amount-of-money scenario, the list of sequences is given in the following table:

1	The register checks customer's input with the price of the product
2	If the amount is greater then the price, the difference is calculated, and register checks its cash reserve
3	If the difference is present in the cash reserve, the register returns the change to the customer and everything proceeds as before
4	Otherway, the register returns the input amount and displays a message that prompts the customer for the correct amount
5	If the amount is less than the price, the register does nothing and the machine waits for more money



The "Incorrect amount of money" scenario

Collaboration Diagrams

Collaboration diagrams like the sequence diagrams, shows how objects interact. It shows the objects along with the messages that travel from one to another. But, if sequence diagram does that, why UML need another diagram? Don't they do the same thing?

The sequence diagram and the collaboration diagram are similar. They're semantically equivalent, that is, the present the same information, and you can turn a sequence to a collaboration diagram and vice versa. The main distinction between them is that the sequence diagram is arranged according to time, the collaboration diagram according to space.

A collaboration diagram is an extension of object diagram. In addition to the associations among objects, collaboration diagram shows the messages the objects send each other.

Messages among objects are represented with arrows that points to receiving object, near the association line between two objects. A label near the arrow shows what the message is. The message typically tells the receiving object to execute one of its operations. A pair of parentheses ends the message. Inside the parentheses, you put the parameters the opearation works on.

Collaboration diagrams can be turned into sequence diagrams and vice versa. Thus, you have to be able to represent sequence information in a collaboration diagram. To do this you must know that:

Add a number to the label of a message, with the number corresponding to the message's order in the sequence. A colon separates the number from the message.

This is the basis of collaboration diagrams. In addition next two lessons will describe the process of writing collaboration diagrams.

Writing Collaboration Diagrams

You can show an **object's change of state** in a collaboration diagram, also you can show **conditions** the same way you represent them in sequence diagram.

In the object rectangle, indicate the state of the object. To the diagram, add another rectangle that stands for the object and indicate the changed state. Connect the two with the dashed line and label the line with a <<become>> stereotype.
You put the condition inside a pair of square brackets, and the conditionprecedes the message-label. The important thing is to coordinate the conditions with the numbering.

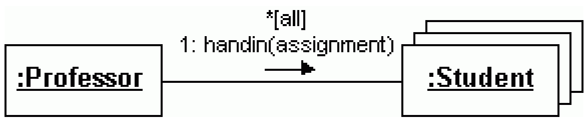
To get closer to this concepts please refer to the Collaboration Diagram - Example.

Also you can present **object creation**.

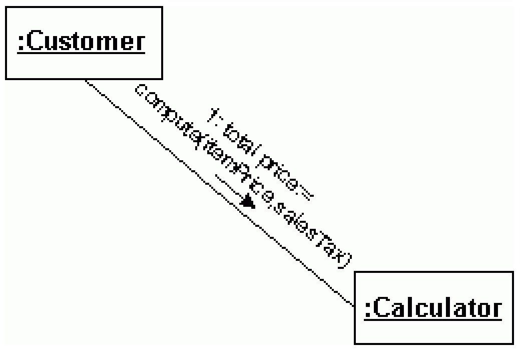
In object's creation process you add a <<create>> stereotype to the message that creates the object.

In the collaboration diagram, you can represent **multiple objects** and **returned results**.

The representation of multiple objects is a stack of rectangles extending "backward". You add a bracketed condition preceded by an asterisk to indicate that the message goes to all objects.
Returned results are written as expression that has a name of the returned value on the left, followed by ***:=***, followed by the name of the operation, and a list of possible attributes that operation can accept, to produce the result.The right side of the expression is called a ***message-signature***.



A proffesor asks a group of students to hand in an assignment.



A message can be a request for an object to perform a calculation and return a value.

In some interactions, a specific object controls the flow. This **active object** can send messages to passive objects and interact with other active objects. Also you might run into an object sending a message only after several other (possibly nonconsecutive) messages have been sent. That is, the object must **sinchronize** its message with a set of other messages.

The collaboration diagram represents an active object the same way as other, except that its border is thick and bold.
Sinchronizing messages, are precede with a list of the messages that have to be completed prior to that message take place. A comma separates one list-item from another, and the list ends with a slash.

Now when you have all this concepts, you're ready to go to the next lesson the Collaboration

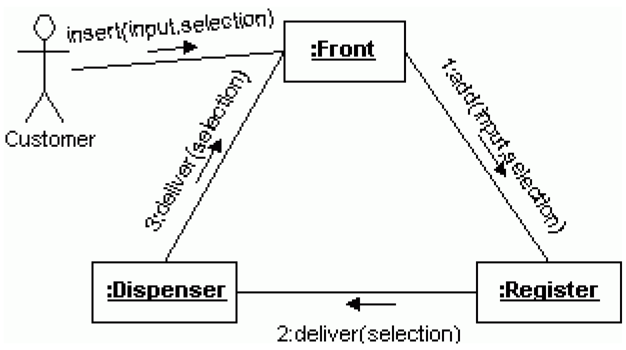
Diagram - Example.

Collaboration Diagram - Example

Self-service machine again will be target for this example. In previos example (See, Sequence Diagram - Example) the sequence diagram for this system was built. This is good time to try the ways of converting sequences into collaboration diagrams.

The best-case scenario consists of several steps:

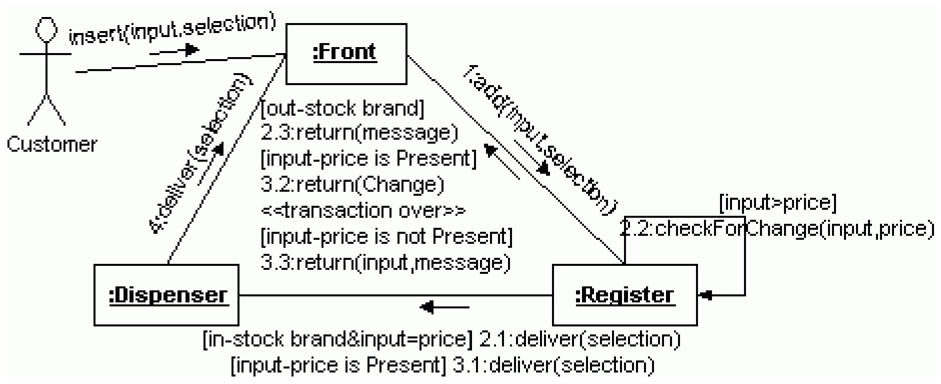
1. The customer inserts money in the machine and makes selection of one or more product(s) present in the machine.
2. When the register gets the money (in this case customer puts the correct amount of money, and there is allways a product in the selected brand), the selected product is delivered to the dispenser.
3. The dispenser delivers the product to the front of the machine, and the customer gets that product.



But when we examined this machine and its work in detail, new scenarios were introduced: incorect-amount-of-money and out-of-product scenario. While building the sequence diagrams, conditions were introduced to display the work of the machine. Here conditions will be used too.

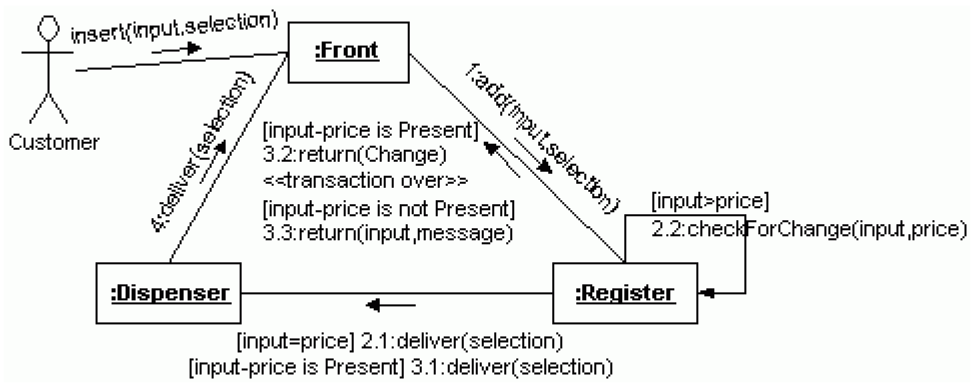
You put the condition inside a pair of square brackets, and the condition precedes the message-label. The important thing is to coordinate the conditions with the numbering. When you're modeling *if* statement there are two possible conditions. Messages for these conditions have a same number, and you add a decimal point and another number. This is called **nesting**.

In out-of-product scenario, the machine has to display an out of product message, and prompt the customer for another product or return the money back. If customer makes another selection this process must be repeated. Also a correct-amount-of-money scenario may be introduced. Collaboration diagram for this case is shown on the picture:



As you can see, from this picture two possible scenarios may happen. *Out-of-product* and *Product-in* are that two possibilities. If we assume that this happened after message *1:add(input,selection)*, the number for this messages will be *2*, and we add *1* for the Product-in and *2* for the out-of-product scenarios, after the decimal point.

What happens when the machine doesn't have the correct change? The machine has to display an out of change message, return the money, and prompt the customer for the correct change. In effect, the transaction is over. When the machine has the correct change, it returns the change to the customer and delivers selected product.
Two branches that are produced from nesting process are numbered with ***2.1*** and ***2.2***. And three posibilities in condition-message 3 will be:***3.1, 3.2*** and ***3.3***. Collaboration diagram is shown on the following picture:



Thats all for this example, and for this day. All you have to do now is answering the questions given in Questions & Answers and Workshop part from this day. Next day will introduce to you the last three UML diagrams.

Questions & Answers

Here is the list of answered questions to see what level of knowledge, you have get this day.

1	**In what important way does a state diagram differ from a class diagram, an object diagram or a use case diagram?** Click for the answer
2	**What is a trigegerless transition?** Click for the answer
3	**How do you represent the flow of control implied by a *while* statement in sequence diagrams?** Click for the answer
4	**How do you represent a message in a collaboration diagram?** Click for the answer
5	**How do you show changes of state in a collaboration diagram?** Click for the answer

A state diagram models the states of just a single object.
A class diagram, object diagram, or a use case diagram models a system, or at least part of a system.

This closes the window

A state diagram models the states of just a single object.
A class diagram, object diagram, or a use case diagram models a system, or at least part of a system.

This closes the window

A triggerless transition is a transition that occurs because of activities within a state, rather than in response to an event.

This closes the window

To represent the flow of control implied by a ***while*** statement,enclose the condition in square brackets, and precede the left bracket with an asterisk.

This closes the window

To represent message in collaboration diagrams arrow near the association line is putted
The arrow points to the receiving object.

This closes the window

Inside an object's rectangle, indicate its state. Add another rectangle for that object and show the changed state. Connect the two with a dashed line and label the line with a <> stereotype.

This closes the window

Workshop

In this day you have learned about state, sequence and collaboration diagrams. To confirm your knowledge about these diagrams try answering these questions and draw some diagrams given below.

1	Must every state diagram have a final state (represented by the bull's eye)?
2	What is the difference between sequential substates and concurrent substates?
3	Define synchronous message and asynchronous message.
4	In a generic sequence diagram, how do you represent the flow of control implied by an ***if*** statement?
5	How do you show sequential information in a collaboration diagram?
6	Is it necessary to include both a collaboration and a sequence diagrams in most UML models that are built?
7 ⚠	Move your mouse. Take a look at your monitor, and you'll see how mouse pointer moves from one position to another. Try modeling (state, sequence and collaboration diagram) this process. What computer parts will be involved in this scenario?

UML Tutorial

CONTENTS

In this day, you'll learn about:

- **Activity Diagrams**
- How to create an activity diagram
- **Component Diagrams**
- Grouping components of the system and creating component diagram
- **Deployment Diagrams**
- Physical organization of the system

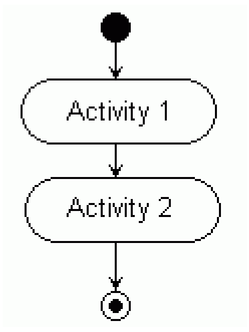
Activity Diagrams

If you've ever taken an introductory course in programming, you've probably encountered the ***flowchart***. The flowchart shows a sequence of steps, processes, decision points and branches. Novice programmers are encouraged to use flowcharts to conceptualize problems and derive solutions.

The UML activity diagram is much like the flowcharts of old. It shows steps (called, appropriately enough, ***activities***) as well as decision points and branches. It's useful for showing what happens in a business process or an operation. You'll wind it an integral part of system analysis.

First and foremost, an activity diagram is designed to be a simplified look at what happens during an operation or a process. It's an extension of the state diagram. The state diagram shows the states of an object and represents activities as arrows connecting the states. The activity diagram highlight the activities.

Each activity is represented by a rounded rectangle - narrower and more oval-shaped than the state icon. The processing within an activity goes to completion and then an automatic transmission to the next activity occurs. An arrow represents the transition from one activity to the next. Also an activity diagram has a strating point represented by filled-in circle, and endpoint represented by a bull's eye.



Transition from one activity to another in the Activity Diagram

In next lesson will be explained ways of creating an activity diagram, rules between activities, and at the end an example for this diagram will be given to you.

The activity diagram is an extension of the state diagram. State diagrams highlight states and represent activities as arrows between states. Activity diagrams put the spotlight on the activities. Each activity is represented as a rounded rectangle, more oval in appearance than the state icon. The activity diagram uses the same symbols as the state diagram for the startpoint and the endpoint.

Building an Activity Diagram

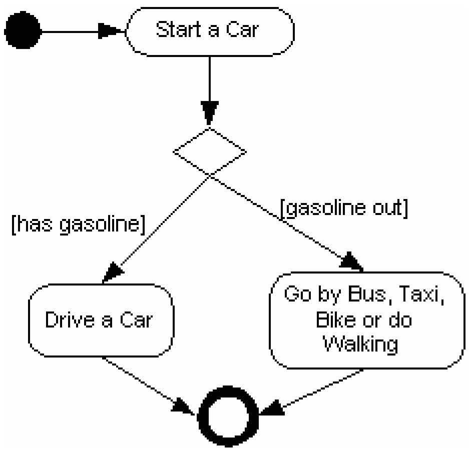
In this lesson will be explained ways of representing decisions, concurrent paths, signals and swimlanes in the activity diagrams.

Decisions

Decision point can be represented in two ways, it's all to you to make decision what way to use in your activity diagrams.

One way is to show the possible paths coming directly out of an activity. the other is to have the activity transition to a small diamond and have the possible paths flow out of the diamond. Either way, you indicate the condition with a bracketed condition statement near the appropriate path.

Imagine that you have to go to the work. You get your car, put the key in the ignition, and there are two possible situations: your car will start or it will not start it's engine. These possible cases will produce two other activities: drive a car, or go by a bus, taxi, bike or go walking. This scenario is shown on this picture (make attention of two ways showing a decision):



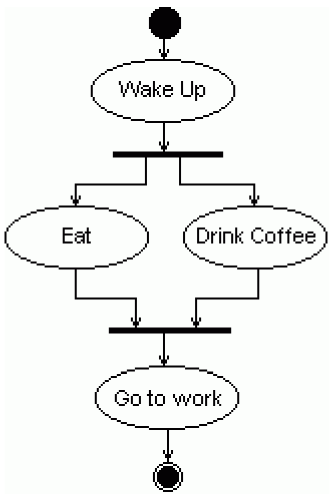
Activity diagram showing two ways of decision

Concurrent paths

When you modeling activities, very frequently you'll have a occasion to separate a transition into two separate paths that run at the same time (concurrently), and the come together.

Split is represented by a solid bold line perpendicular to the transition and show the paths coming out of the line. To represent the merge, show the paths pointing at another solid bold line.

Activity diagram representing a transition split into two paths that

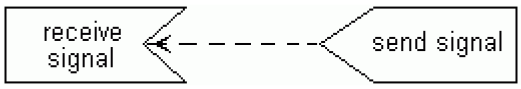


run concurrently and then come together

Signals

During a sequence of activities, it's possible to send a signal. When received, the signal causes an activity to take a place.

The symbol for sending a symbol is a convex pentagon, and the symbol for receiving a signal is a concave polygon.



Sending and receiving a signal

Swimlanes

The activity diagram adds the dimension of vizualizing roles. To do that, you separate the diagram into parallel segments called ***swimlanes***. Each swimlane shows the name of a role at the top, and represents the activities of each role. Transitions can take place from one swimlane to another.

It's possible to combine the activity diagram with the symbols from other diagrams and thus produce a ***hybrid diagram***.

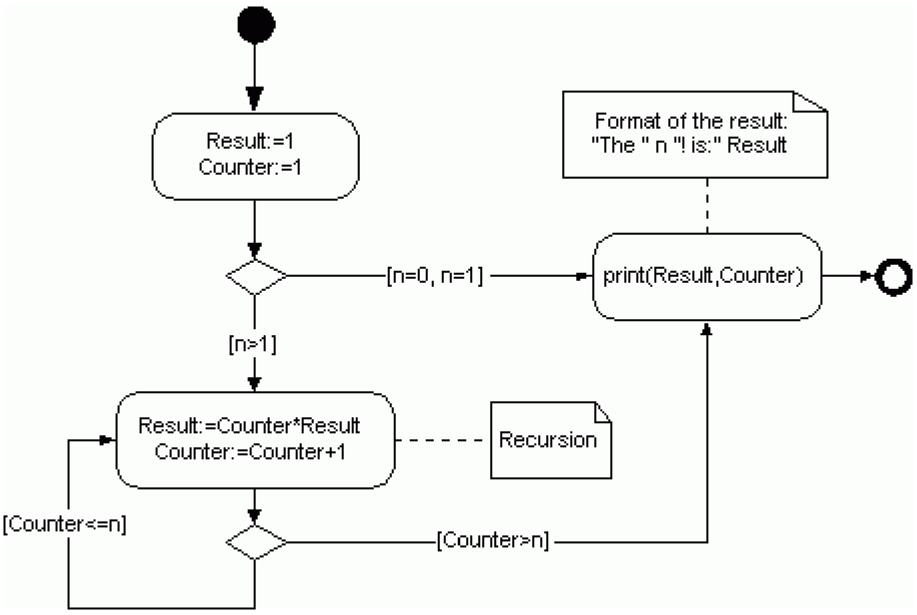
Following lesson will give you a written example for an activity diagram.

Activity Diagram - Example

This example will deal with mathematics. Sometimes (when calculating combinations) you'll need to calculate a factoriel of n - ***n!***.
The formula for this is *n!=n*(n-1)*(n-2)*...*2*1.*

Let's dive into the problem. From definition of factoriels we have 0!=1 and 1!=1. For the rest of the numbers we must use the given formula. In computer programming this problem is written with recursion. Here also when building an activity diagram recursion will be introduced. You might call the operation *computeFact(n)*. You'll need a counter to keep track of whether or not the operation has reached the *n*th factoriel, a variable that keep a track of your computations, and two more to store the values of 0! and 1!.

The complete activity diagram is shown on next picture:



If you're dealing with computer programming you will conclude that this activity diagram is very similar with a procedure (function) that computes factoriels.

More About Interfaces and Components

In previous lessons, you learned about diagrams that deal with conceptual entities. In next lessons, you're going to learn about a UML diagram that represent a real-world entity: ***software component***.

But what is a software component?
A software component is a physical part of a system. It resides in a computer, not in the mind of an analyst.

What's the relationship between a component and a class?
Think of a component as the software implementation of a class. The class represents an abstraction of a set of attributes and operations. And one component can be implementation of more than one class.

You model components and their relationships so that:

1. Clients can see the structure in the finished system
2. Developers have a structure to work toward
3. Technical writers who have to provide documentation and help files can understand what they're writing about
4. You're ready for reuse

When you deal with components, you have to deal with their interfaces. Interface is the object's "face" to the outside world, so that other objects can ask the object to execute its operations, which are hiden with encapsulation.

An interface is a set of operations that specifies something about a class's behavior. It is a set of operations the a class represent to other classes. For you as a modeler, this means that the way you represent an interface for a class is the same as the way you represent an interface for a component. As is the case with a class and its interface, the relation between a component and its interface is called realization. See, Interfaces & Realizations.

You can replace one component with another if the new component conforms to the same interfaces as the old one. You can reuse a component in another system if the new system can access the reused component trough that component's interfaces.

As you progress in your modeling career, you'll deal with three kinds of components:

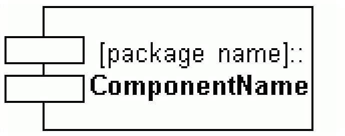
1. ***Deployment components***, wich form the basis of executable systems (DLL's, executables, ActiveX controls, JavaBeans)
2. ***Work product components***, from which deployment components are created (data files and source code files)
3. ***Execution components***, created as result of a running system

**Instead of representing a conceptual entity such as a class or a state, a component diagram represents a real-world item - a software component. Software components reside in computers, not in the minds of analysts.
A component is accessible trough its interface. The relation between a component and its interface is called realization. When one component access the services of another, it uses an import interface. the component that realizes the interface with those services provides an export interface.**

Component Diagrams

A component diagram contains components, interfaces and relationships. Also other types of symbols that you've already learned can also appear in a component diagram.

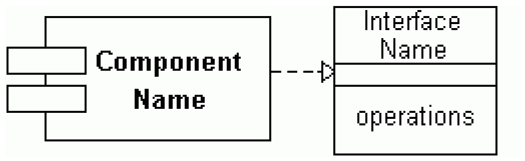
The component diagram's main icon is a rectangle that has two rectangles overlaid on its left side. You put the name of the component inside the icon. The name is string. If the component is a member of a package, you can prefix the component's name with the name of the package.



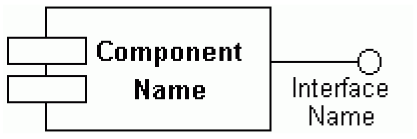
The component icon

A component and the interfaces it realizes are representable in two ways.

One way is to show the interface as a rectangle that contains interface-related information. It's connected to the component by the dotted line and empty triangle that vizualize realization.
Other way to represent interface is as a small circle connected to the component by a solid line, which represents a realization relationship.



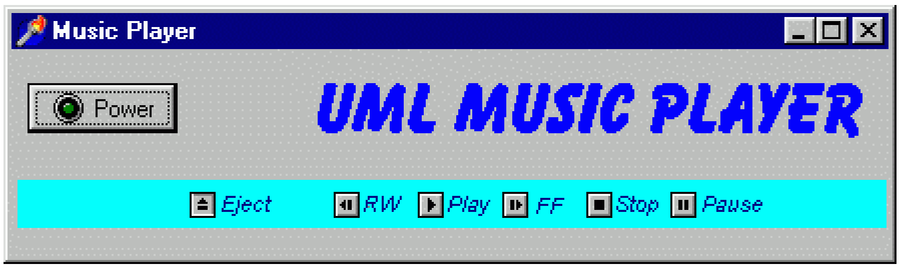
Interface can be represented as a rectangle containing information, connected to the component by a realization arrow



Interface is a small circle, connected to the component with a solid line, wich stands for realization

Component Diagram - Example

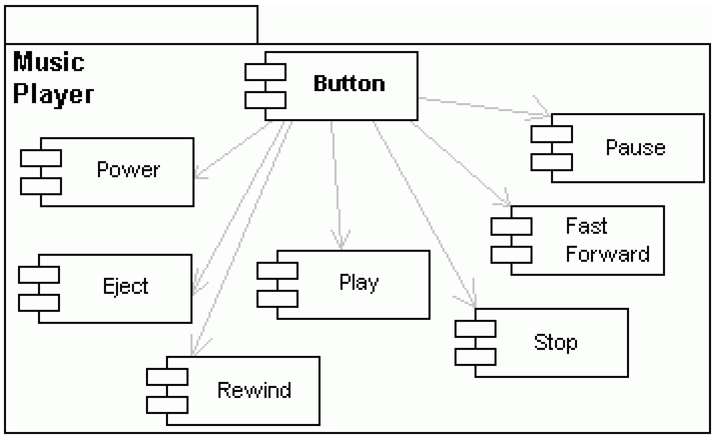
Suppose that we need to build up a software for playing a music from a CD-ROM Drive. A visual programming language might be used (VisualBasic or Delphi for example). If language supports multimedia controls, than we can use its components an reprogramm them if necessary, or we can programm new components. One possible graphical design for our player might be:



As you can see this UML Music Player needs these controls:

- play
- stop
- eject
- pause
- fast forward
- rewind
- power

These controls will be realized by ***buttons***, thus we'll have a button performing these controls. If we look at buttons as separete components, we can draw out a component UML diagram. This is shown on the following picture:



The component diagram for the MusicPlayer.

All the components shown on the previous diagram belongs to one global component - Button, but actions they perform are diferent. We must obtain these actions by programming them.

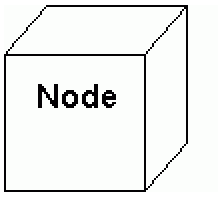
Deployment Diagrams

We reached the end of the UML diagrams. Now it's perfect time to look at the hardware. As you can see, we moved from items that live in analysis, to components that live in computers, to hardware that lives in the real world.

Hardware, of course, is a prime topic in a multicomponent system. A solid blueprint for hardware deployment is essential to system design. The UML provides you with symbols for creating a clear picture of how the final hardware setup should look.

The main hardware item is a ***node***, a generic name for any kind of computing resource. Two types of nodes are possible. A ***processor*** is a node that can execute a component, and a ***device*** that can't execute a component. A device typically interfaces in some way with the outside world.

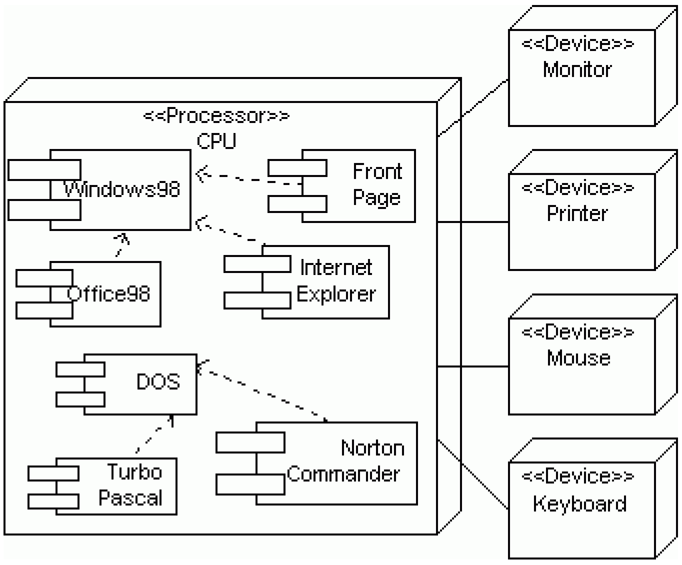
In the UML a cube represent a node. Node has its name, and you can use a stereotype to indicate the type of resource it is. If a node is a part of package, then and the name of package preceeds the name of the node. A line joining the two cubes represents a connection between two nodes. You can use a stereotype to provide information about the connection.



The cube is representation of a node in the UML

Also every node deployes some of software components in the system. To indicate deployed components, you show them in dependency relationships with a node.

Let's build a deployment diagram for a home computer system. Computer consists of these hardware elements: CPU, monitor, printer, mouse, keyboard. And installed software components: Windows98, Office98, InternetExplorer4, FrontPage, DOS, NortonCommander, TurboPascal.



The main node is CPU which consists of all the software components with their relationships. Other hardware elements are devices connected to the main processor unit.

You may extend this diagram by adding a modem and connection to the Internet. Try to redraw the diagram with this extension.

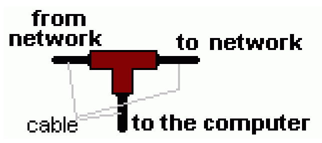
The UML deployment diagram provides a picture of how the physical system will look when it's all put together. A system consists of nodes (represented by

a cube) with line connecting them, called connections. There are two types of nodes: a processor (can execute a component) and a device (interface with the real world). Deployment diagrams are useful for modeling networks.

Deployment Diagram - Example

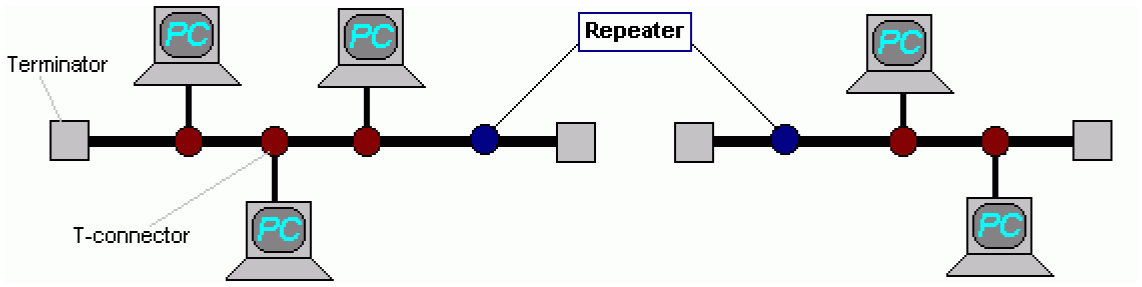
This example will describe you the thin Ethernet network. If you're familiar with computer networks this one will be easy to you, but if not then try to understand the following explanation:

The thin ethernet is a popular type of network. It is used in local places eg.rooms or buildings, where connecting cable can be placed. There are some mathematics calculations for the length of the cable of all the network, and length of cable from computer to computer. These calculation for the deployment diagram are not important.

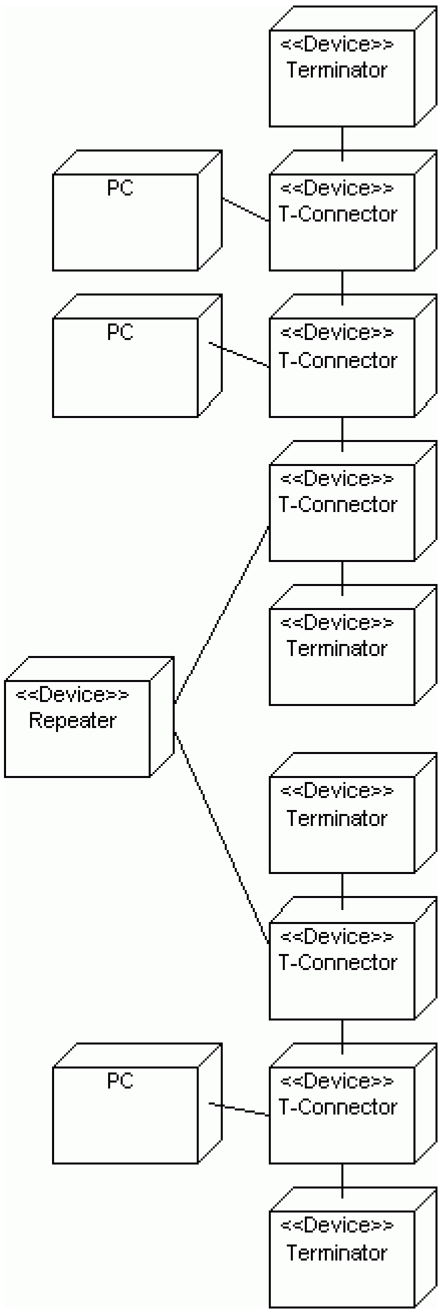


Computers connect to a network cable via connection devices called T-connectors, or sometimes vampire-taps are used. T-connector has three connecting points. Its form is similar with T-letter and thats why their name cames from. Network cable goes from one end-point to the other end-point of the connector, and the third end-point has a cable going to the computer.

Because length of the network is limited, ending points of the cable must have a devices called terminators, which gave infinite resistance to the end of cable. Also one network segment may join another via a repeater. The repeater is a device which amplifies the signal to reduce losing of signal. The thin Ethernet is represented on the following picture:



When we get familiar with the thin Ethernet network concept let's draw the deployment diagram for it. This diagram wil look like:



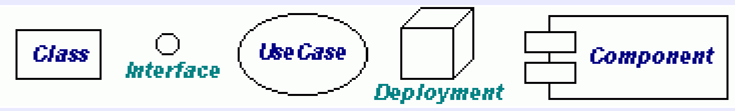
Deployment Diagram for Ethernet network

With this example all of the UML diagrams are shown to you. In this day, only that you have to do, are the exercises. But before make some exercises, A global picture with all of the UML symbols is given to you. See it.

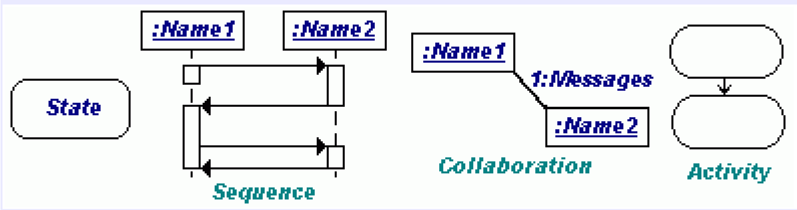
UML Diagram Set of Symbols

This is the complete set of symbols that UML introduces in its diagrams.

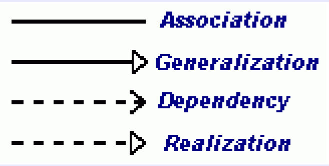
Structural Elements



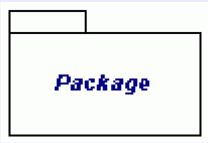
Behavioral Elements



Relationships



Grouping



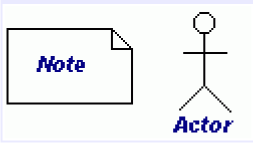
Extension

<<Stereotype>>

{ Constraint }

{ tagged value }

Annotation & Actors



Questions & Answers

In this day you've finished the set of UML diagrams. Test your knowledge about activity diagrams, component and deployment diagrams.

1	**What are the two ways of representing a decision point in activity diagrams?** Click for the answer
2	**How do you represent signal transmission and reception?** Click for the answer
3	**What are three types of components?** Click for the answer
4	**What do you call the relationship between a component and its interface?** Click for the answer
5	**What are the two kinds of nodes in deployment diagrams?** Click for the answer

One way is to show a diamond with branches coming out of it.
The other is to show branches coming directly out an activity.
Either way, put a bracketed condition on each branch.

This closes the window

Signal transmission is represented by a convex pentagon, and signal reception is represented by a concave pentagon.

This closes the window

The three types of components are:
(1) **Deployment components**,
(2) **Work product components**,
(3) **Extension components**.

This closes the window

The relationship between a component and its interface is called **realization**.

This closes the window

The two kinds of nodes in deployment diagram are **processors** (which can execute components, and **devices** (which connect to the outside world.

This closes the window

Workshop

Please answer following questions, and draw the necessary diagrams. This will strength your knowledge in activity diagrams, component diagrams and deployment diagrams.

1	With everything that a state diagram shows, do we really need activity diagrams?
2	What is a swimlane? Give an example.
3	What is an export interface? What is an import interface?
4	Name the ways when you can replace and when you can reuse a component. Why is important to have a reusable components?
5	How do you represent a node in the deployment diagram? What kind of information can appear on a node?
6 ⚠	Draw activity diagram for connecting up a computer to the Internet. After that draw a deployment diagram for a home computer system connected with the Internet.

UML Tutorial CONTENTS

In this day, you'll learn about:

- **Modeling of Digital Library**
- Discovering Business Processes
- Domain Analysis
- JAD Session

The Digital Library - Example

We finished UML diagram's theory. Now is a perfect time to dive into practice. This and next day are concerned with Digital Libraries. If you're not introduced with them, don't worry. Things will be much clearer after these two days.

Back in time we remember the clasical library structure in which there is a librarian with a great deal of paperwork, the user must be physically there to get the book and know which book is of his/hers interest. If he/she doesn't know the desired book, than is in the hands of librarian to recommend some book for him/her.

The new wave in this direction, based on the software development and Internet is headed into new library entity- the ***Digital Library***. This kind of library is world-wide accessed, with a quick review possibilities, digital books and a lot of multimedia files. They can recommend you a book or a journal, give a statistical aspect of the recent books and a lot of features that are making the life easier – all of that with the comfort of lying in the sofa with remote keyboard in your hands. Remote reservation is even making possible of getting a hard copy of a book or journal using a efficient postal system connected with the library.

Day 5 and Day 6 will guide you trough the process of analyzing and implementing a digital library. You will find out all the steps introduced in Rapid Application Development process, and their work-products UML diagrams. First two lessons, during interviews, will describe you the work of one digital library in detail. Let's go to work now.

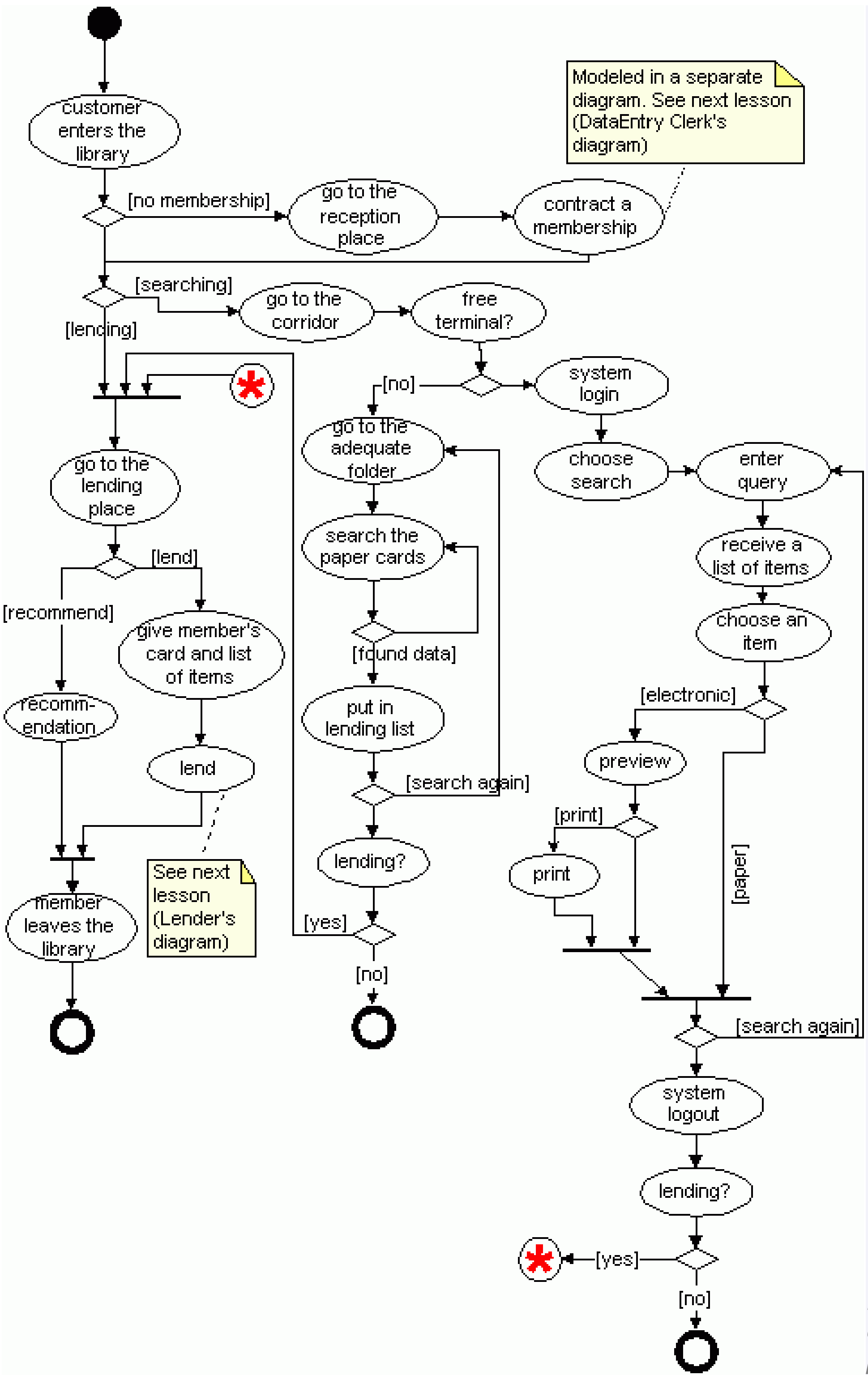
Discovering Business Processes *1*

In this lesson you'll find how to make an interview with a client. This will be a dialog between an analyst and a client. The final product of this conversation will be an activity diagram.

In the following text ***A*** will stand for Analyst and ***C*** for the client.

A	Let's start with the work. Tell me what's happening when a customer enters in your library?
C	He or she goes to the corridor where data search can be done. After selection of which books or journals to lend, the customer goes to the place where he or she can take the sample of that book or journal.
A	You said data search. What kind of data?
C	There is main data record for every book, author or journal, sorted alphabetically. Every record consist of different data according to type of the product: (book or journal). These records are printed on paper cards, and they are putted on different folders. Access to these records can be done also by computer terminals which can be found in the corridor.
A	That means that your library is computerized. Why using a computers and a paper records?
C	As we live in hi-technological era, we must to follow time steps. Computers were introduced few years ago, but paper records are here since the library was built. Also some impatient customers, can use them if all of the computer terminals are occupied.
A	Are the same data put in the paper records and in the computer records?
C	Yes. All of old data from paper records is converted to computer records, and new data that we write in paper cards also is writen in electronic format.
A	Where is your database located on?
C	Our database is located on a computer server. This server is a heart of our local area network - all of the terminals located in our library.
A	Let's backtrack to the data. You said that same data is stored in the electronic format and in the paper format. Next in our conversation we can refer to these with same name, let's say index.
C	I agree.
A	When we talk about index, what kind of information is necessary to perform a search?
C	Data for the books and for the journals is very similar. Both have ISBN-the unique number for every book/journal, title, publishing year, publishing house, but books have author and genre, and journals have volume and area. For the data search are usefull ISBNs, titles, authors, genres, volumes and areas. Retreived results specifies the type of the books or journals.
A	Just a second. What are the kinds of books and journals?
C	Except paper books and journals, electronic books and journals are supperted in our library.
A	Electronic books and journals. How do you manage these types of items (books / journals)?
C	The server we had is used for storage of these types of items. We produced some of the electronic items, but also in our database we have Internet addresses from electronic books or journals from other libraries we used to cooperate.
A	You have a lot of ammount of data. Is everyone concerned with data input and data maintenance?
C	We have employees to do this.
A	You said employees. I think that they are not parts from this conversation for now. We will talk about them later. The customers are not finished yet. Before going to the process of lending let's assume what we have at this point of interview, things come clearer to me now. Every customer can perform a data search for the books or journals. Electronic formats of books retrived from the search, can be accessed imidiately. Paper formats can be lended if there is a free sample. Now you may go to the process of lending.
C	At the begining, I've stress out that user who is in the library after data search can perform a lending. Lending can be done if books or journals are in paper format, and it's done in the lending area in the library. Searched books or journals can be given to the user if free samples

	of them are in the library.
A	Let's look at the both cases - present and not present sample of the book/journal. What is the flow of events?
C	When sample is not present, the user can make a reservation for it. After that library must inform the user for a free sample, when one is set free. User, than should go the the library and make a lending for that sample. I must say that free sample is kept for the user few weeks after informing it.
A	What if there is a free sample?
C	In that case emloyee who is concerned with lending (let's say - lender), must check the users data to see if user is a library member or not. If the user is a library member, he or she takes the ISBN of lended books or journals and personal data for the user, and writes them in the lending record. Also a date is writen in this record, which is stored in the lending database. Every user has a limit of books/journals to lend, and also there is a limit for time keeping of books/journals.
A	Do you allow employees to lend books or journals?
C	Yes we do. Every employee in our library has the same record as members. That we have lending records for employees too.
A	You said that lending records are stored in a lending database. Is this a work of "lender" or someone else do this?
C	Our local network doesn't consists only from searh terminals. A "lender" also has a terminal from which he or she can check for user data and free samples. Also a quick reports can be done to see which users have books to return.
A	Wait a minute. Our conversation gone in other direction. We started to talk about employee's duties, we didn't finished with the customers. Let's see to made a lending a free sample of the book or journal can be present in the library, and a lending record is produced after it.
C	That's it. I'll stress again that only paper formats can be lended, because they have a samples. Electonic formats of books and journal doesn't have a samples. In other words they have infinite number of samples, or they are present in the library everytime.
A	Is it possible to have no present samples of books or journals from searched user's list? Can users ask a lender to recommend them a book or journal?
C	Certainly lender can take a brief look at the user's history of lended books and journals, and if he or she had read a books or journals from the user's interest areas, then he or she can recommend something to the user. User can accept that or not.
A	This will be all for now. I think that user's diagram is done, and we can take a look at it:



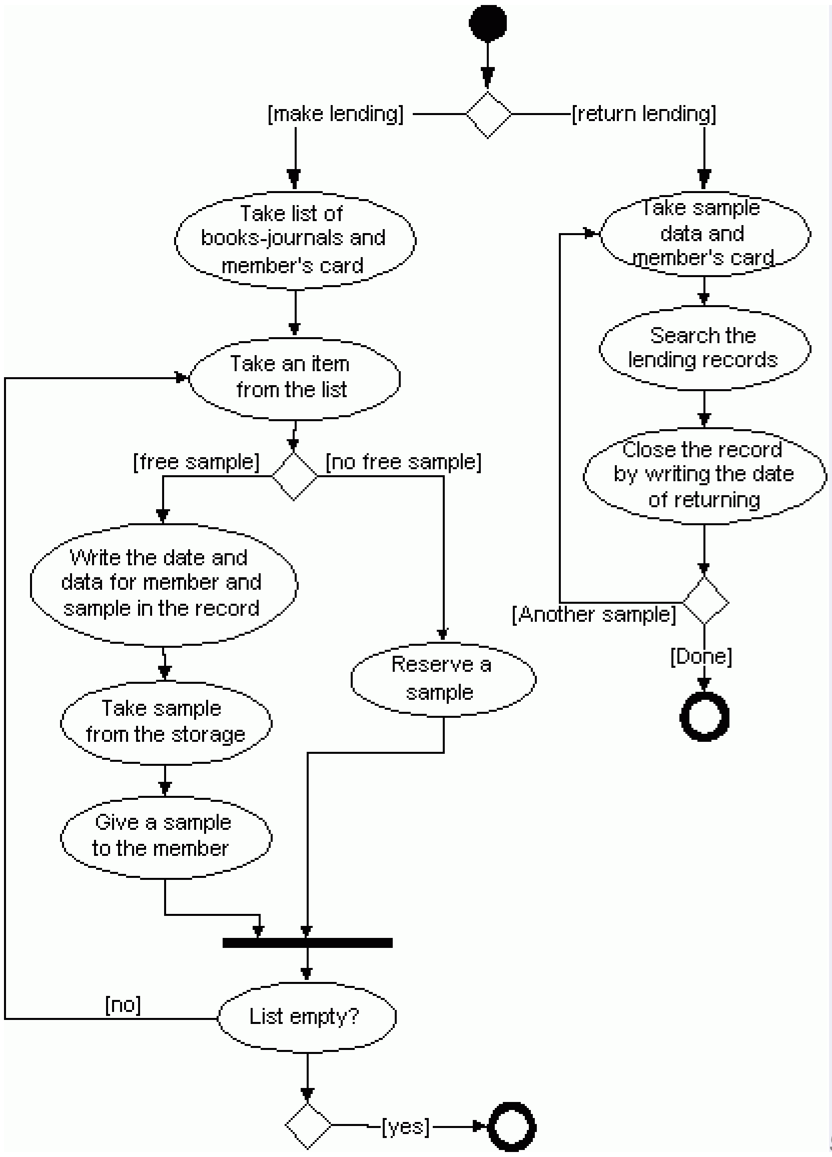
Member Activity diagram corresponding to the previous interview

Discovering Business Processes *2*

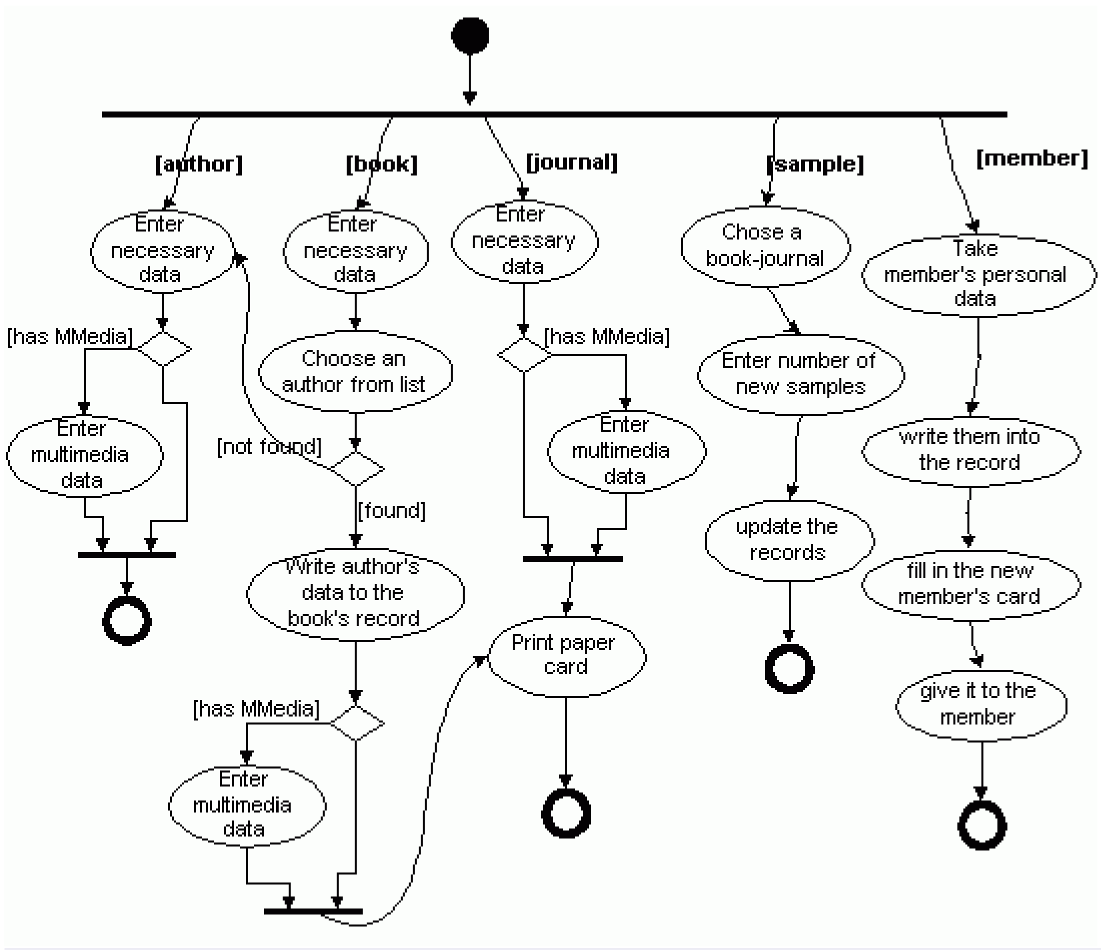
This lesson is extension of previous one. Now we'll give you the interview with the client - explanation of work of the employees in the library. Let's see at all given details and analyze the produces activity diagram.

A	In previous interview we talked about customers, and all the activities related to them, like lending, reservation, recommendation, and search. In lending and recommendation process interaction with the employees was introduced. Let's turn to the employees now, and talk about employees which interact with clients, books and journals.
C	Allright. The lenders were introduced in previous interview, and we see that their work is to take care of lending and returning of books and journals. They post reports to the members who have books to return, and members who have reserved books and their samples are present in the library.
A	Is it all for the lenders?
C	Yes.
A	When we talk about a customers, we mentioned that for huge amount of data, some of the employees are employed. Can you tell me more about them?
C	After buying a computers, a part of our old employees, involved in process of writing paper cards, were trained to work with computers. These DataEntry clerks now operate with some of terminals for data entering and maintenace. Also for maintenance of our databases DatabaseSpecialists were employed.
A	If I waht to become a member to your library. What I'm supposed to do?
C	First of all every new member must go to the reception place to conclude his or her membership. Employee that enters new memberships, opens a new record with a name and last name of the member, address and place of living and Personal Registration Number added to it. Also telephone e-mail and date of birth are stored in this record. After registration, a member's card is given to the member, and a paper record is printet for the library's documentation.
A	How do you make data input of books and journals?
C	Books and journals have a plenty of attributes in common. That are ISBN, title, year of publishing and publishing house. Also every item has a number of samples. Sometimes a multimedia data are supported with the item. If that is a propaganda material in paper form we convert it into electric form and put it into the database.
A	Do you support multimedia data only for electronic items, or paper items has multimedia too?
C	The form of the book/journal is not important. Both for paper and electronic formats we support multimedia data. Popular books in the world, can be more popular if a multimedia is supported for them. Every user is curious to meet with these types of information.
A	If I perform a data search, can I browse the multimedia data supported for the item? And what kind of multimedia is supported?
C	It's true. During computer's searching in returned results if an item has a multimedia data, user can browse that data. Types of multimedia which are supported are Text, Audio, Video and Picture.
A	How do you enters a data for the book's author?
C	I'm sorry. I forgot to tell you that authors are entered separately. For the authors name and last name, year and place of birth are inputed. During the process of book's input if the book's author is in the entered list of authors, then he or she is chosen from that list. Otherway we used an author creation process. Also for books their genre is inputed.
A	What about a journals?
C	For the journals volume and area are entered.
A	You said that a parallel evidence is made (paper and electronic). How do you represent multimedia data in your paper cards?
C	Multimedia data is entered only in electronic format. I said that paper cards are printed from electonic records, but only necessary information for cards is printed.
A	We examined a work of lenders, employees who enters a data for authors, books, journals and

	members. But I don't see here a sample input. In lending process you said that free samples can be lended. How do you perform sample input?
C	I notioned when we talk about book entering, that process of book's input also consists of inputing a number of new samples. But it is possible new samples to arrive in library, and a record for the book to exist in database. In that case only sample data is entered. This data consists of sample's signature, which only concerns the lenders (information about stored place of sample in the storage room).
A	Do you think that we have finished the employee's analysis?
C	Sure. We reached the end. I hope that all my spelling out all the steps in library work, makes clearer view to you about libraries like ours.


[make lending]
[return lending]
Take list of books-journals and member's card
Take an item from the list
[free sample]
[no free sample]
Write the date and data for member and sample in the record
Take sample from the storage
Give a sample to the member
Reserve a sample
[no]
List empty?
[yes]
Take sample data and member's card
Search the lending records
Close the record by writing the date of returning
[Another sample]
[Done]

The Lender's Activity diagram gained from previous interview



The Data Entry Clerk's Activity diagram extracted from the interview

Domain Analysis

Before haveing additional interviews, the development team first must work within the context of the business-process interview. This is the job of an ***object modeler***. The objective is to produce initial class diagram.

The object modeler looks for nouns, verbs and verb phrases. Some of the nouns will become classes in the model, some will become attributes. The verb and verb phrases can become operations or the labels of associations.

Let's put ourselves in the role of the modeler and start developing the class diagram. From the client's interview we can filtered out these nouns:

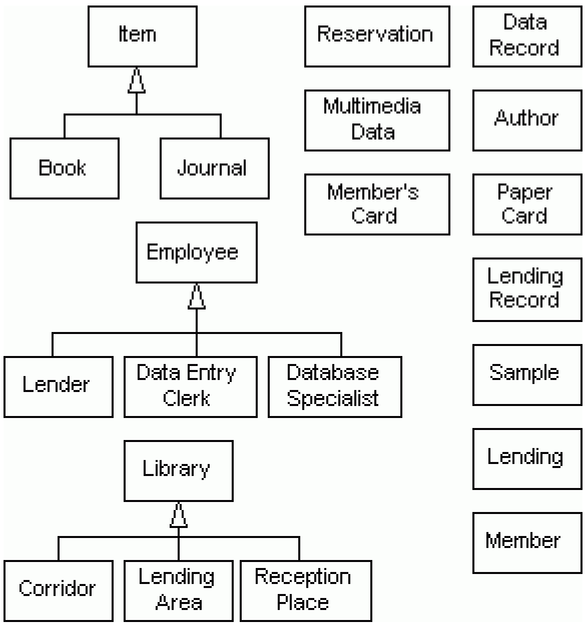
corridor, data record, storage folder, local network, computer server, computer terminal, paper book, paper journal, electronic book, electronic journal, item, date, limit of time, limit of copies, book, journal, sample, author, paper card, library, database, ISBN, title, publishing year, genre, publishing house, volume, area, reservation, employee, lending, lending area, lender, user, member, lending record, report, number of samples, DataEntry clerk, DatabaseSpecialist, reception place, name, lastname, address, place of living, PersonalRegistrationNumber, telephone, e-mail, date of birth, member's card, library's documentation, multimedia data, information, text, audio, video, picture, signature

Move your mouse over the word list and you'll see what nouns from the list can be eliminated as an attributes, synonyms with other nouns, and nouns that represent classes out of the domain's scope.

And these verbs:

go, search, lend, take, sort, print, put, do, has, have, convert, write, manage, access, read, preview, take a sample, reserve, give, inform, make, set free, check the data, recommend, post, buy, operate, maintain, conclude, browse, input

Now we'll try to form some meaningful groups of nouns.
One group consists of ***people***: member, author, employee, lender, DatabaseSpecialist, DataEntry clerk. This group could stand some subdivision because everyone except the member and author are employees.
Another group consists of ***library's items*** like books, journals, samples, reservation, lending.
Third group consists of ***areas within the library***: corridor, lending area, reception place.
In forth group may be these nouns: member's card, data record, paper card, multimedia data, lending record.



Abstract classes partition the class diagram into meaningful groups

This is the initial class diagram, next we'll discover associations between classes and produce more descriptive class diagram tha this one.

Forming Associations

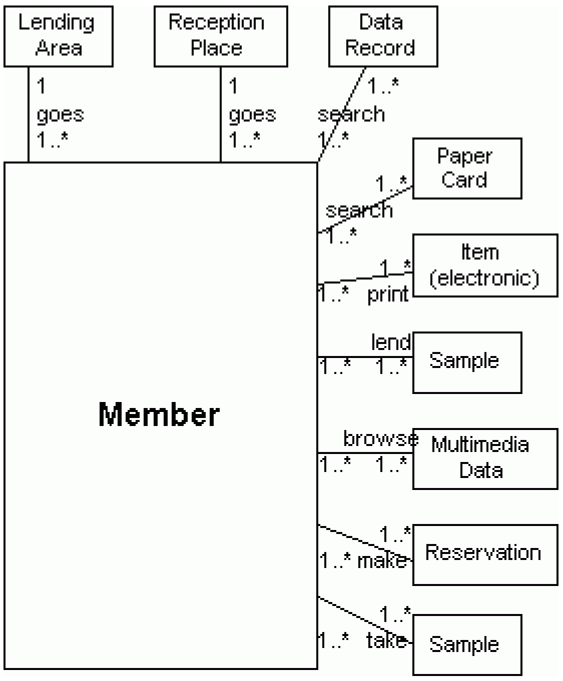
Now we'll create and label associations between some of the classes. The verbs and verb phrases can help us with the labeling, but we won't limit ourselves to the ones from the interview.

Labels that are somewhat more descriptive might suggest themselves.

Let's start with the member first. Let's label the associations by generating phrases that characterize the associations. Here are some phrases that immidiately come to mind:

- The member searches the paper cards or data records.
- The member goes to the lending place.
- The member lends a sample.
- The member makes a reservation.
- The members takes a sample.
- The member browse the multimedia data.
- The member goes to the reception area.
- The member prints the electronic item.

When we labeled out the associations, we can put multiplicities into associations lines. After this customer's class will look like this:

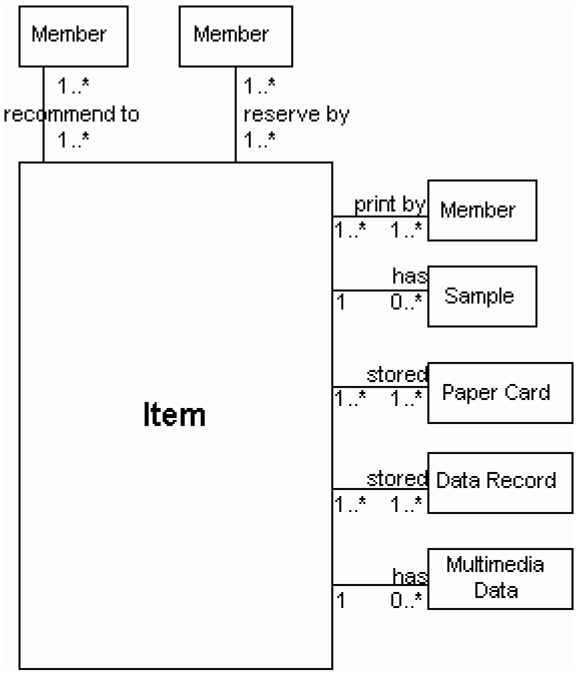


The member class with included multiplicities in the associations

The associations for the item's (here item is standing for both book and journal) class might be the phrases from the following list:

- The item can be recommended to the member.
- The item is reserved by a member.
- The item (electronic) is printet by a member.
- The items are sorted alphabeticaly in paper cards and data records.
- The item has sample(s).
- The item has multimedia data.

After filling out associations names and multiplicities, the item's class will be:

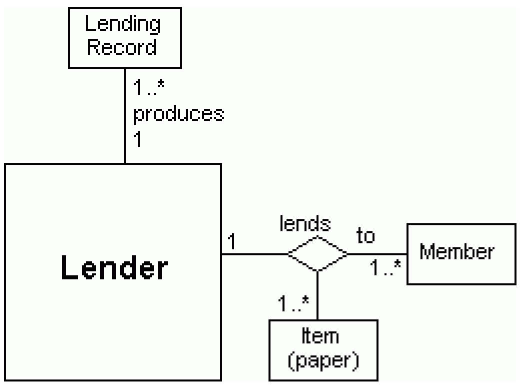


Associations with the items (books and journals)

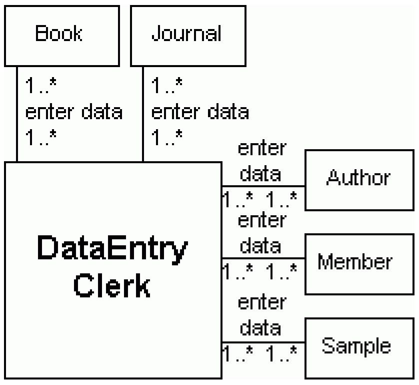
Interesting, also is the process of forming associations and multiplicities for the emplyee's class. Bear in mind that employees are also a members of the library, thus all the member's class associations can be putted into the emplyee's class, but some of the employees perform some actions that members are not allowed:

- The lender produces a lending record.
- The lender lends an item to the member.
- The DataEntry clerk enters data for the books.
- The DataEntry clerk enters data for the journals.
- The DataEntry clerk enters data for the authors.
- The DataEntry clerk enters data for the members.
- The DataEntry clerk enters data for the samples.

After filling out associations names and multiplicities, the lender's and DataEntry clerk's classes will be:



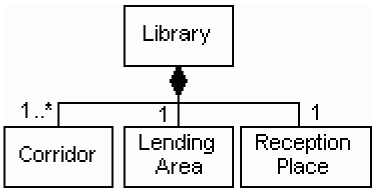
Associations with the lender



Associations with the DataEntry clerk

Try drawing out the associations with the Author's class and Sample's class. This will be also an exercise for you given in the workshop for this day.

We've been forming and naming abstract classes and associations, and another organizational dimension awaits. The next step will be finding out classes that are components of other classes. A library for instance consists of a set of parts, but for as interesting will be: corridor, lending area and reception area. Next picture shows this compozition.



Compozition in the Digital Library domain

If you have some probles understanding processes of forming associations, multiplicities and compozitions, please see these lessons learned in DAY 2: Associations and Aggregations.

Filling Out the Classes

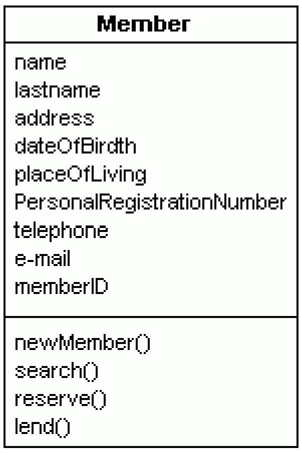
We can begin the refinement now by adding some attributes and operations. Our most important classes appear to be Member, Author, Employee, Book, Journal, and Sample.

The Member

In nouns list some of the nouns are appropriate to be used as member's attributes. And from vers list we can chose some of the possible operations for the member. These two lists are given in the following table:

List of attributes	List of operations
name	*newMember()*
lastname	
address	*search()*
date of birdth	
place of living	*lend()*
PersonalRegistrationNumber	
telephone	*reserve()*
e-mail	
memberID	

The member class now will look as:

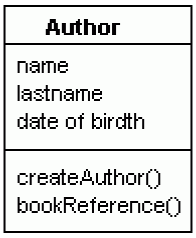


The Member class

The Author

Lists of attributes and operations for the author will be:

List of attributes	List of operations
name	*createAuthor()*
lastname	*bookReference()*
date of birdth	



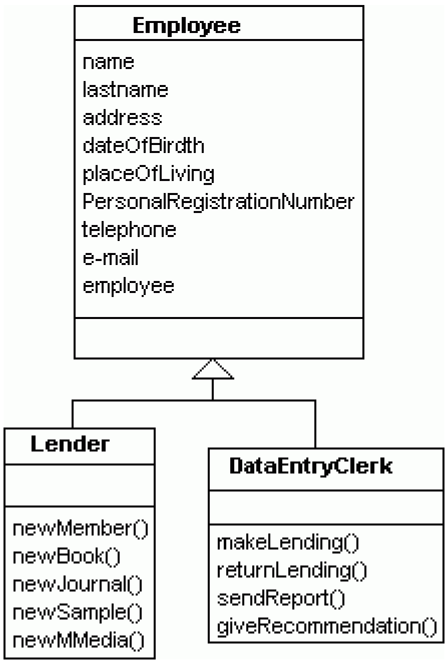
The Author class

The Employee

Lender and the DataEntry clerk are childern of the abstract class Employee. First we'll assign attributes to the Employee class and the children classes will inherits them. Also we must bear in mind that every employee has a member record. This will guide us that in the Employee class list of member's attributes may appear, plus an attribute to differs employees from members. Also DataEntry clerk class will have new attribute which determines the type of data entry performing that clerk. List of attributes and operations (for Lender and DataEntry clerk) are given in fillowing table:

List of attributes for the Employee class	List of operations for the Lender class	List of operations for the DataEntryClerk class
name		*newMember()*
lastname	*makeLending()*	
address		*newBook()*
date of birdth		
place of living	*returnLending()*	
PersonalRegistrationNumber		*newJournal()*
telephone		*newSample()*
e-mail		
memberID	*sendReport()*	
employee		*newMultimedia()*

The structure of the Employee class with its subclasses is given on the following picture:



The Employee class and its children

Try finding out an attributes and operations for the Book class. This class will be very simmilar with next - the Journal class.

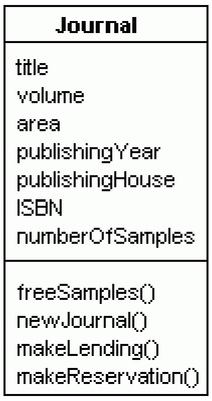
The Journal

List of attributes and operations for the Journal class are following:

List of attributes	List of operations
title	*freeSamples()*
volume	
area	*newJournal()*
publishing year	
publishing house	*makeLending()*
ISBN	

status	*makeReservation()*
numberOfSamples	

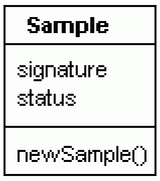
The "status" attribute indicates the type of the journal: electronic or paper. The Journal class will be:



The Journal class

The Sample

The lending process may produce lended book or journal only if a free sample of that book or journal is present in the library. For that purpose a Sample class must be present. The sample class will have only two attributes (in this moment) - *signature* and *status* (lended or free), and one operation *newSample()*. The Sample class is present on next picture:



The Sample class

When you're putting together interview results, business processes, and domain analyses, keep a model dictionary. This is a glosary of all the terminology in the model. It will help you maintain consistency and avoid ambiguity.
Also it's not a good idea to have every detail of your class model in one hude diagram. You'll need a master diagram that shows all the connections, associations and generalizations, but it's the best to elide attributes and operations from this picture. You can turn the spotlight on selected classes by putting them in separate diagrams.

Developing the Vision

Now it's time for the team to work on the technical backbone for the Digital Library. They've got business processes and class diagrams. Can they begin with the coding now, or not? They're not even close to writing a program, first, they have to develop a vision of the system.

The team will use its business process knowledge and newly acquired domain knowledge to see where infusion of technology enhances the library work. The players in this conversation are an analyst, a modeler, a librarian, a lender, a DataEntry clerk, a member and a system engineer. A facilitator runs the meeting. The facilitator distributes copies of the business process diagrams developed in previous lessons.
See Discovering business processes 1, and Discovering business processes 2.

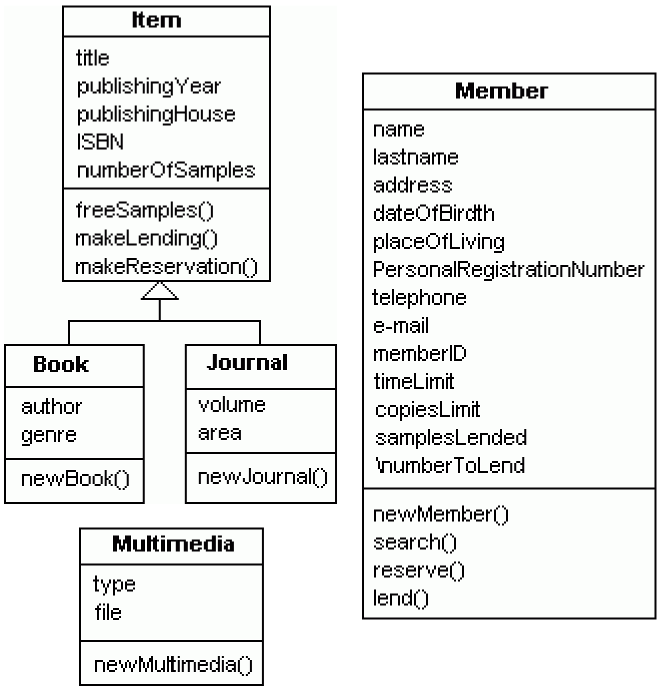
This conversation is about information movement. Some of the processes in the library business depends on the movement of the information. If we can speed that movement along - something technology is really good at - we'll meet our goal. From this conversation was concluded that information movement takes place when:

- The member lends a book's sample
- The member lends a journal's sample
- The lender recommends a book to the member
- The lender recommends a journal to the member
- The member contracts the membership with the DataEntry clerk

Next will be given a part from this conversation. Here modeler has some interesting ideas about adding new attributes in the class diagram. Let's see what is happening:

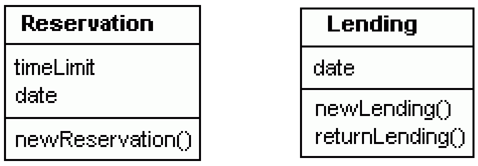
Analyst	Let's go to the process of lending. It was mentioned that members has time limit and copies limit for a lending. How the lender knows how many books or journals to give to the member?
Lender	First of all I take a look at member's lending record, and count how many items he or she has lended. Also I check the lending date to see if there is a books or journals with overloaded time limit.
Modeler	I've been working on my class diagrams while I've been listening to all of you. And I have some questions to ask. In the Member class, we may add new attributes like timeLimit and copiesLimit which are common for every member. Also an attribute that holds the number of lended samples - samplesLended may be introduced, and a derived attribute - numberToLend, which is the difference between copiesLimit and samplesLended.
Lender	That's a nice idea. Then I'd know the number of allowed samples to lend to the member.
Analyst	What about timeLimit?
Modeler	We have an attribute that holds the lending time. If we add timeLimit to it, a result is aimed. Then if the result is greater than current date, we can inform the member that he or she has a time pass over for a lended book or journal.
Analyst	What about multimedia data? It was said that for some of the books, journals or authors multimedia data is supported.
Modeler	It's a perfect time to introduce a new class in our class diagram - Multimedia. For that purpose, Multimedia class will have two attributes *type* (text, audio, video or picture) and *file* (physical location of the multimedia file) and one operation - *newMultimedia()*.
Facilitator	Analyst are you satisfied?
Analyst	I agree. Now things comes clearer to me. Every part of the puzzle is putted on its place.
Modeler	Here's another possibility. Look carefuly at the Book and the Journal classes. Both are haveing common attributes like ISBN, title, publishing year, publishing house, number of samples. They are differing in author and genre data for the book, and volume and area data for the journal. This is perfect situation for a new inheritance. Common attributes can be putted in new parent class, let's say **Item**. Also in this new class some

	of the operations can take place like: freeSamples(), makeLending() and makeReservation(). With this we can save memory space in our database.
Facilitator	I'm glad that we're making some optimization. Before going to the next steps, san I suggest you to look at new classes?
	(All agree)



Changes in the Member class. New Multimedia class and hierarchy between books and journals, the parent class is the new one - Item class

Analyst	During reservation process or lending process, current date must be somewhere stored. In our class diagram I don't see place in the classes where this data is stored. Please tell me where do you put information about date of lending and date of reservation?
Modeler	According to many-to-many association between the Sample class and the Member class, we are supposed to add a new class for that association holding the intersection data for that association. Right here we can add two new classes, the Reservation and the Lending. Here date of reservation and date of lending can take place. Also in the Reservation class an attribute for timeLimit of the reservation can be added. This is necessary for calculating the time period of holding a reservation to the member. These new classes are given in the following pictures



New classes added in our class diagram are the Reservation class and the Lending class

Analyst	Allright. For this moment I think that we've resolved all the problems about classes. Now, can we move on to some ideas about what the system should specifically do?
Facilitator	Sure. Ideas everyone?
Member	I think that this sytem will be more productive if we, the members, can access to all the data in the library from home. Sometimes, when I go to the library with tought to lend some book that I realy need, I hear the Lender's words: "Sory, but we're out of free samples for that book. Can I recommend you something else?". In that moment I'm saying into me: "My chef will kill me. I spent so much time and I didn't finished my work. I hope that there will be a little bit crowd in traffic."
Analyst	Yes. Our system has to somehow keep the members from walking to the library so much. Obviously they have to go to the library when they made a reservation from their home and a free sample is holded for them. Also a lending process can be done if a member is present in the library.
System	I think we're onto something. We can allow members to access to the library's LAN

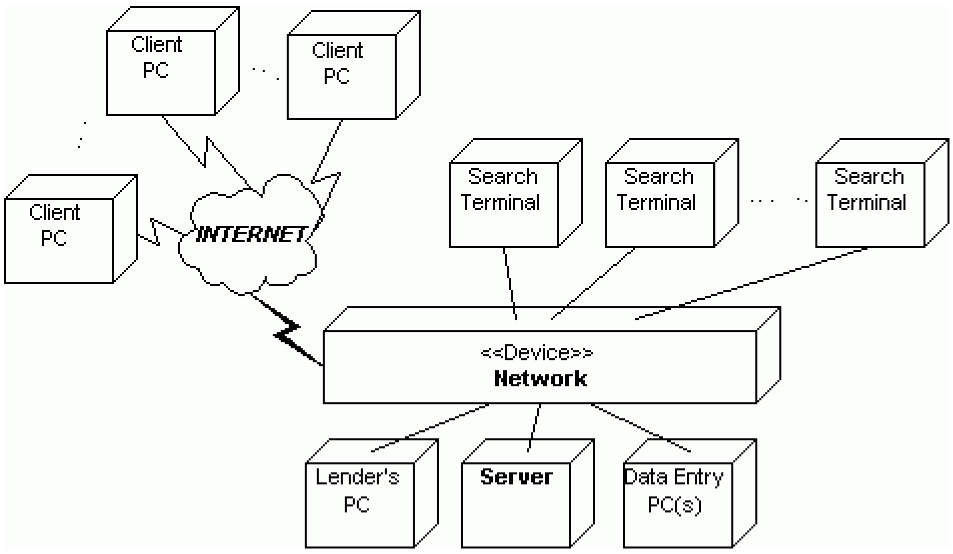
Engineer	directly from their homes or work offices, using the global network - Internet. Then the information would move around very quickly. Also I heed to tell you that our system will cooperate with library's LAN, but new software will be implemented everywhere.
Analyst	The system you're talking about would resolve a number of issues. Like data search from home and makeing a reservation for a book or journal from your home or office. I'll have a feeling that I'm in the library and using their local area network and I don't care for a free terminal.
Member	That's beautifull. When I have a lot of work to do, I can turn on my computer, make a connection with the library and search the data to find out a free sample of books or journals I need to lend. But from library's network I can't make a reservation for a book of journal. How can I make a reservation from my home or office?
System Engineer	We are going to implement a Web interface that allows you to access data stored in the library's database, and allows you to make a reservation from your home or office by clicking a button that sends a message from your computer with your personal data and book's data that you want to reserve. This message will open a new reservation record in our database holding the data until you get your reservation, or time limit is oversteped.
Member	But how can my computer know my personal data?
System Engineer	First of all, Web interface can be used only be library's members. In our class diagram we have memberID - number printed on your member's card with all the neccesary personal data. When you search the data you make a selection of books/journals to reserve. This list is also sent when you click the button. But if you aren't a member in the library, first that you must do is to make a library's membership.
Librarian	Is it possible some of our employees to access the system if they aren't present in the library too?
System Engineer	Of course. Data entry clerks can do their job even if they are not present in their work offices. They can access the database data from every place in the world that has a computer and Internet connection.
Librarian	How are you going to make this happen?
System Engineer	Let's don't worry about that right now. Very soon you'll find out how this will be realized.
Facilitator	So we're all set, then? Our system will incorporate the library's local area network and Internet. Now is time to give a name for our system.
System Engineer	How about Digital Library.
Facilitator	Can we all agree with Digital Library?
	(All agree)
Facilitator	Okay. I think our work here is done.

JAD Session

Now that the team has a vision for the system, can the programmers program and the system engineers engineer? Absolutely not. The team must center the Digital Library system around the user's needs, not around nifty technology. Altough they have a few insights from the team meeting, they haven't exposed the Digital Library concept to a group of employees and end users - the members, to get feedback and ideas from the user's point of view. The next GRAPPLE action does just that. In a **Joint Application Development session**, the team will gather and document system requirements.

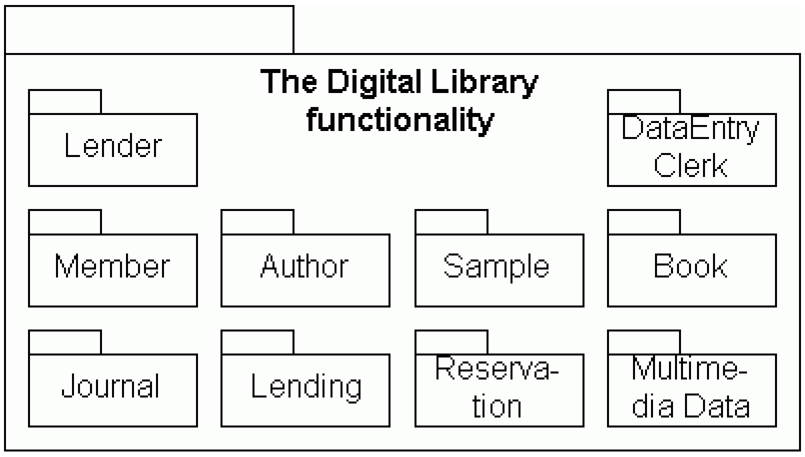
The JAD session takes place in conference room. Led by a facilitator, it's called a ***joint*** because it includes members of the development team along with potencial system users and domain experts. The development team members are analyst, a modeler, two programmers and a system engineer. The potential users are two members, a lender, two data entry clerks and a librarian. This meeting will produce a package diagram that shows the major functionality of the system.

This session starts with a facilitator's speech about the Digital Library concept. The structure of a Digital Library system is represented on this picture, which is given to all the participants in the meeting.



The physical structure and connections between components in the Digital Library

The group starts their conversation by figuring out what the major pieces of functionality should the system had. It was decided to use following functionality of the system:



The packages of functionality for the Digital Library

Next will be given a part of the meeting conversations between some of the participants:

Facilitator	Now that we have the major pieces, does anyone have a preference as to where to start?
Member1	How about the Member part?
Facilitator	Sounds good. Alright, what kinds of functionality would you waht to see in this package? Remember, group, just because we're doing a piece that happens to not coincide with your particular job, you can still participate. everyoune's insights are welcome.
Member1	I like to perform a search from my home and selected books and journals reserve until I go to the library and lend them.
Facilitator	Okay. What else?
Member2	I want to print electronic books and journals directly from my office.
Analyst	By the way I don't see any information about electronic items in our class diagram. How can we manage these items?
Modeler	You're right. In the Item class we must add new attributes for an electronic books/journals. A ***status*** will be one. With this one we can check the type of the item (electronic or paper). Also we must add an attribute that holds the location of the electronic item (on the local storage device or on Internet). This attribute will be named as ***link***.
Facilitator	Has anyone had another kind of functionality to add?
Member1	I want to take a list of recommended books and journals onto my computer.
Facilitator	OK. You'll notice that I'm writing these in labeled ellipses. We refer to these as *use cases*. We'll be asking some of you to come back and help us analyze those use cases, but that's another meeting.

When JAD session has finished, participants had a set of requirements that appear as use cases arranged in the packages. The list of uses cases for every package is given to you.

The member package use cases:

- enter data for searching
- transmit the data to the library's database
- retreive a list of items
- make a reservation for some of them
- transmit the reservation to the library's database
- receive notification from library's database
- take a list of recommended books
- take a list of recommended journals

The book package use cases:

- select an author
- receive an author data
- enter data for a new book
- edit data for an existing book
- enter multimedia data
- receive notification from multimedia adding
- check for free samples
- receive number of free samples
- check for reservations
- receive number of reservations
- find out the oldest reservation
- make a reservation
- receive notification for reservation
- make a lending
- transmit lending data
- receive notification for lending

The sample package use cases:

- add new sample
- set it free
- edit data for an existing sample
- receive a request for free samples
- return number of free samples

The reservation package use cases:

- receive a request for reservation
- perform a reservation
- transmit a reservation result to the member
- delete a reservation

The lending package use cases:

- receive a request for new lending
- perform a lending
- transmit a lending result to the lender and book
- delete lending
- receive oldest reservation
- receive a request for returning a lending
- return a lending
- transmit a result to the lender
- check for reservations
- transmit information for reservations

The author package use cases:

▸ enter new author	▸ receive request for author's data
▸ edit old author's data	▸ transmit data to the book

The lender package use cases:

▸ enter new lending	▸ perform a lending return
▸ receive notification from lending	▸ receive a result from returning a lending

For exercise try adding the use cases in DataEntry clerk package, Multimedia package, and to the Journal package.

***Click here** to see how will look our functionality package diagram filled with use cases!*

In a JAD session, the development team meets with potential users and domain experts to gather the requirements for the system. The result is a package diagram in wich each package represent a major piece of functionality. Use cases inside a package elaborate on the functionality.

Questions & Answers

First day of practical using the UML diagrams has finished. Before going to the next day, test your knowledge about requirements gathering process.

1	**In these two days we're going to model out a Digital Library. Can you give a definition for a Digital Library, and why is it so important nowadays?** Click for the answer
2	**How do we make use of nouns in the interview with an expert?** Click for the answer
3	**How do we use the verbs and verb phrases?** Click for the answer
4	**How are we representing system requirements?** Click for the answer
5	**Does class modeling stop after the domain analysis?** Click for the answer

The Digital Library is a library which is not located on a building, but is located on computer servers.
Book's content is in electronic format, containing a lot of multimedia data (ordinary books don't have this kind of data)Access to the books is done by Internet.
Digital Libraries take a great progress today, because a lot of paper materials are converted into electronis format.

This closes the window

Nouns became candidates for class names and class attributes.

This closes the window

Verbs and verbs phrases become candidates for operations and for names of associations.

This closes the window

We're using the UML package diagram along with use cases to represent system requirements.

This closes the window

No it doesn't. Class modeling continues to evolve after the domain analysis.

This closes the window

Workshop

Answer the following questions, and draw out the necessary diagrams.

1	Which UML diagram is appropriate for modeling a business process?
2	How do you know which classes to eliminate from the candidate class list?
3	Can some of the JAD session participants be the same people who participated in the earlier team meeting?
4	Try drawing out the associations with the Author's class and Sample's class. See Forming Associations.
5	In Filling Out the Classes lesson, we fill in the class icons with attributes and operations. This was done for every class except the Book class. Try finding out the attributes and operations for this class and draw the complete class.
6	Your turn was in JAD session to find out use cases for the Journal, Multimedia and DataEntry clerk packages. If you didn't finished your work then, it's time to finish it now.

UML Tutorial CONTENTS

In this day, you'll learn about:

- **Analysis of the Digital Library**
- The use cases analysis
- Interactions in the Digital Library
- Integration with other systems
- **Design of the Digital Library**

Analysis of the system

We've finished the requirements gathering process for the Digital Library model. During this day an analysis of the Digital Library model will be done.

First we'll meet with use case analysis, after what a use case diagram for every package will be gained. Next we're going to discover and analyze an icteractions among objects in the Digital Library system, and we'll produce collaboration diagrams. And at the end of the day we'll examine the interactions between our Digital Library with existing systems, like Library's local network, and we'll produce detailed deployment diagram, and a few user interfaces will be introduced.

Let's start with the work, and learn the material given in this, the last day, of UML course.

Developing the Use Cases

The use cases from the package diagram in JAD Session lesson from DAY 5, give a good picture of what the system will have to do. The team will have to flesh out each one. They've moved gradually from understanding the domain to understanding the system. The use cases have provided the bridge.

At no point in the JAD session did the development team discuss how the system would accomplish all the activities specified in the panolopy of use cases. The idea was just enumerate all the possible use cases. As the use cases become fleshed out in this hour, notice how the components of the Digital Library system start to materialize. At this point in the development effort, the system begins to take center stage.

To analyze the use cases, we have to run another JAD session. The discussion in this JAD session is intended to derive analysis for each use case.

The use case JAD session is usually the most difficult one, as it calls for the participants - potential users of the finished system - to become analysts. In their own niche, each one is a domain expert, and you have to tap into their expertise. Typically, they're not used to either verbalizing or analyzing what they know. It's also probably true that they haven't been part of a system design effort before, and they may be uncomfortable with trying to specify what a system should do to help them do their work.

The system-related steps in the scenario are extremely important. They'll show how the system is supposed to work. When the JAD session participants tell us these steps, they're telling us, in efect, what the system will ultimately look like. After this JAD session, we should have a good idea about the components of the system.

In Introducing a Use Case Model lesson in DAY 2, we see that each use case is a sequence of scenarios, and each scenario is a sequence of steps. For each scenario in each use case, we'll want to show:

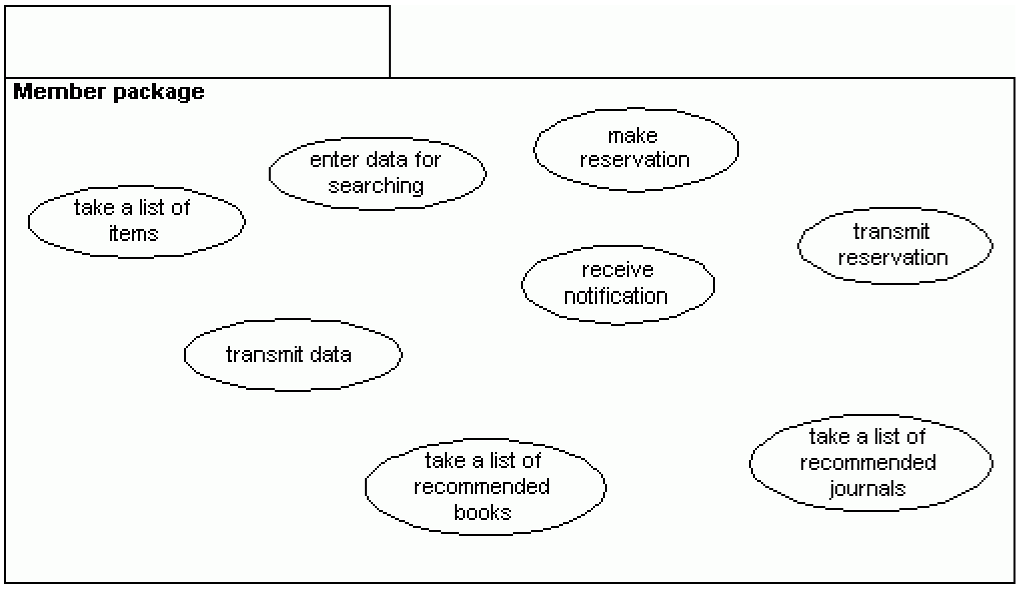
- A brief description of the scenario
- Assumptions for the scenario
- The actor who initiates the use case
- Preconditions for the use case
- System-related steps in the scenario
- Postconditions when the scenario is complete
- The actor who benefits from the use case

Use cases that we're going to examine are numerous. That's why when we're performing an use case analysis number of them will be examined in detail, and the rest of them will be given to you as an exercise. Now let's dive into process of analysis the use case.

The Use Case Analysis 1

In this lesson we'll meet with the process of analysis of the use cases. We're going to examine a member's package of use cases and a book's package of use cases. The rest of them are given to you, to try yourself in the use case analysis.

The Member package



The Member's package of use cases developed in earlier stages of the system analysis

- Enter data for searching

The good-one sentence description for this use case may be: ***The member enters a data for serching into search form.***

Here initialing actor will be Member, which is also a benefiting actor.

The assumptions are that a member wants to find out books or journals. Another assumption is that the Digital Library's interface has a form dedicated for search data entry.

The precondition is that a member has a necessary data to make a search. The postcondition is that a member enters necessary data into Digital Library.

The steps in the use case are:

- From the Library's LAN or from home/office, the member activates the user interface for search entry.
- The search form appears on the screen.
- The member enters necessary data patterns into search form.

- Transmit data for searching

The description for this use case may be: ***Take a data pattern entered in search form, and send it to the Digital Library's database search engine.***

The assumptions are that we have communication via the Internet or library's LAN, and we have a search form in the member's interface.

note We must make repetitions of some of the assumptions, because each use case will eventually appear on a separate page in the design document, which will serve as a reference about the system.

The precondition is entered data pattern in search form, the postcondition is that the data pattern has arrived in the Digital Library database search engine.

The steps in the use case are:

- Click on the SUBMIT button on the search form.
- Activate the transmiting mechanizam for data in network(s).

- The data pattern arrives in Digital Library database search engine.
- On the member's screen appears a message that data has been sent, and to wait a moment during receiveing the data.

Because this use case is related to the previous one, we must make a change to the Member package. It has to show <<include>> dependency between this one and "Enter data for searching" use case.

- Take a list of searched items

This one use case is the last part for completion the searching process. Description here is: ***Receive a list of book or journals, after entering a data in search form and send it to the Digital Library database.***

The assumptions are that a member uses a search form, and a communication via network(s) is good.

The precondition is clicked SUBMIT button on search form, and the postcondition is printed list of items on the member's screen. The actor here again is Member.

The steps of this use case are:
- Wait until search is performed (in paralell a Digital Library search engine performs a search).
- Digital Library database transmits a list of items to the member's interface via the network(s).
- The member receives a list of items satisfying entered search-data pattern.

Inclusion also appears in this use case. <<include>> joins this use case with the previous one.

- Make reservation

Description for this use case will be: ***The member wants to make a reservation for the item gained from searching process or recommendation process.***

As you can see this use case depends on two different use cases - search and recommendation. Thus reservation process may take place only when one of these two processes take turn. Thats because an inclusion is presented between these three use cases.

The assumptions are a list of items, and a button for reservation.

The precondition is clicked RESERVE button, and a postcondition is transmited reservation record to the Digital Library.

The steps of this use case are:
- The member receives a list of items (from searching or from recommendation)
- The member clicks the RESERVE button for the items that he or she wants to reserve
- The member fills in the information in the form that appears for the reservation

- Transmit reservation

This use case wouldn't be explained in detail, because it is almost the same with the "transmit the data for searching" use case. Only we'll say that this use case is related with "make reservation" and "receive notification" use cases. The connection is <<include>>.

- Receive notification

Description is: ***After pressing the SUBMIT button in the reservation form, a notification from the Digital Library must arrive that the reservation has been accepted, and infrom the member with the status of reservation.***

Here status of reservation is for haveing or not a free sample for reserved item. If a free sample is present how many weeks it'll be holded for the member.

The assumptions are good communication in the network(s) and clicked SUBMIT button.

The precondition is clicked SUBMIT button, and a postcondition is received status of reservation

The steps of this use case are:
- Wait until notification is received
- Transmit the necessary information via the network to the member's interface
- Members receives the status of reservation

- Take a list of recommended items

Description is: ***The member activates the recommendation part of the system, and gets a list of recommended items***

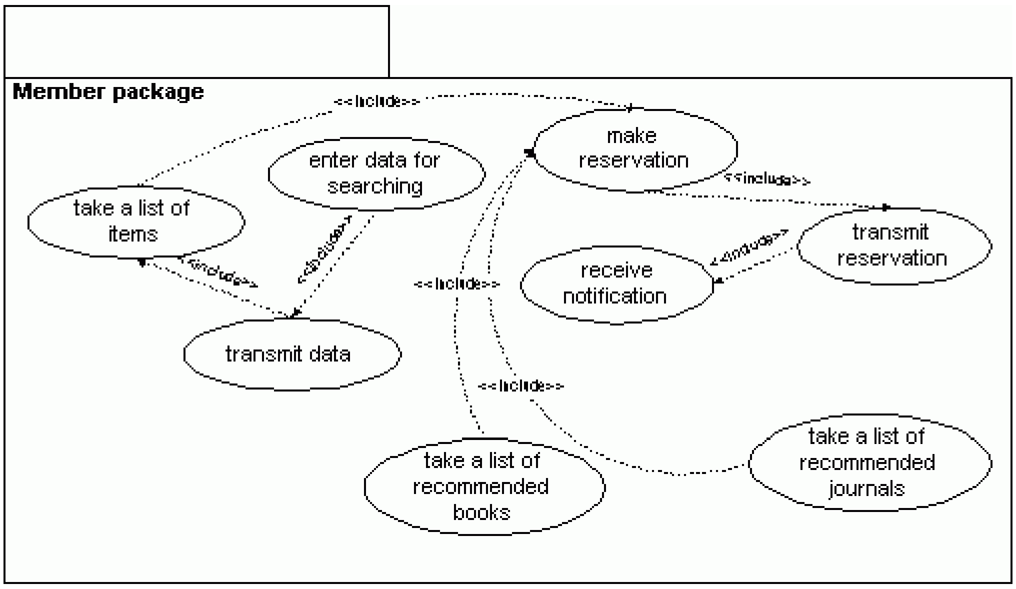
The assumptions are clicking the RECOMMEND button, and a working recommendation algorithm.

The precondition is clicked RECOMMEND button, and a postcondition is received list of items.

The initiating and benefiting actor is Member.

The steps of this use case are:

- Activate the recommendation part from the user's interface.
- The Digital Library activates a recommendation algorithm which produces a list of items.
- Print this list of items to the member's screen.

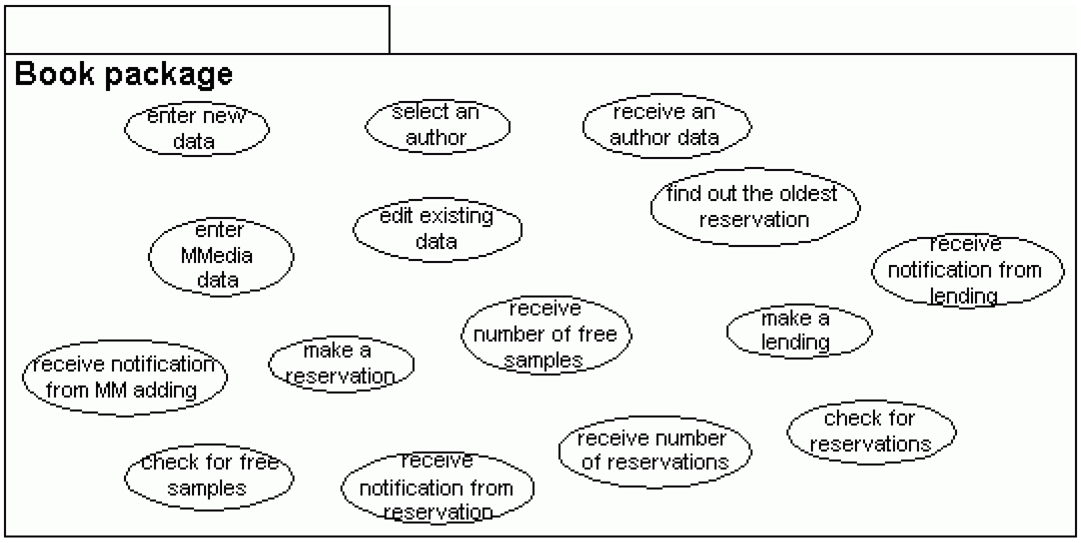


The updated use case diagrams for the Member package

The Use Case Analysis 2

Now let's take look at the Book package use cases, and analyze them separately.

The Book package


Book package
enter new data
select an author
receive an author data
find out the oldest reservation
enter MMedia data
edit existing data
receive notification from lending
receive number of free samples
make a lending
receive notification from MM adding
make a reservation
check for reservations
receive number of reservations
check for free samples
receive notification from reservation

The Book package of use cases developed in earlier stages of the system analysis

- Enter data for a new book

Description: ***Fill in the data in book's entry form.***

Initialing actor is DataEntry clerk.

Assumptions: Activated entry form in the DataEntry clerk's interface, and entered information.

Precondition is new book's data.
Postcondition is entered data into Digital Library database.

The steps are:
- From the library's LAN or from internet the DataEntry clerk activates an interface for a book's entry.
- The form appears on the screen.
- The clerk enters necessary information.

- Select an author

Description: ***Select a book's author from the author list.***

Initialing actor is DataEntry clerk.

Assumptions: There are author records in the database, and user makes a selection from that list.

Precondition is book's author data.
Postcondition is selected author's data.

The steps are:
- In data entry form for the book, a clerk selects one author from the list of authors.
- This information is send to the database.
- Searching is performed into the database for that information.

An inclusion may be introduced here. This use case includes steps from "edit data for an existing book" use case and from "enter data for a new book" use case.

- Retreive author's data

Description: ***Receive all the necessary information about the selected author in previous use case.***

Assumptions: User has made a selection from the author's list.

Precondition is selected author.
Postcondition is retreived author's information.

The steps are:

- After searching, database selects the record for that author and extracts necessary information about the author.
- Sends that information to the book's data entry form.
- Prints the information onto the user's screen.

This use case includes the "select an author" use case.

- Enter multimedia data

Description: ***If the book has a multimedia data, then it is entered into the book's record.***

Assumptions: Book has a multimedia data, and a multimedia file(s) is(are) saved somewhere in the server.

Precondition is book has a multimedia data and a MMEDIA button is clicked.
Postcondition is entered multimedia data.

The steps are:

- If book has a multimedia data, a MMEDIA button is clicked, and a entry form appears on the screen.
- User enters type of multimedia data, and a location on the server where that data is saved.
- SUBMIT button is clicked and multimedia record is transmited to the database.

An inclusion may be introduced here. This use case includes steps from "edit data for an existing book" use case and from "enter data for a new book" use case.

- Receive notification from multimedia adding

Description: ***After filling the multimedia entry form, database sends a message how succesfull was performed a process of adding new multimedia data.***

Assumptions: Filled multimedia data entry form, good communication via network, and existing multimedia file.

Precondition is transmited multimedia entry form to the database.
Postcondition is received status of this transaction.

The steps are:

- The multimedia data arrives into the database, and a process of creation of new record is started.
- This process may finish with a result, or may end with failure.
- The type of ending of the creation process is returned to the book's data entry form.

This use case is inclusion of previous one - "enter multimedia data" use case.

- Edit data for an existing book

Description: ***If a wrong information is entered for some book, it is possible to change that information.***

Assumptions: Existing book's record for changeing, and clicked EDIT button.

Precondition is false entered information into book's record.
Postcondition is changed book's record.

The steps are:

- User clicks the EDIT button and filled form with the old book's information appears on the screen.
- False information is selected and changed.
- If an author's information or multimedia information has to be changed, the same process like entering new data is repeated.

- Make a reservation

Description: ***Member wants to reserve a sample for the book, and after that a lending can be performed in the library.***

Assumptions: Book must be present in the retreived list of books during the search or recommendation process, books must have samples present in the library.

Precondition is Member wants to make a reservation for a book.
Postcondition is book's necessary data and member's necessary data are transfered to the Digital Library.

The steps are:

- Member clicks the RESERVE button and fills in the necessary information to perform a reservation.
- The Digital Library database receives request for reservation.
- The steps of checking the data with database records are made.
- Digital Library checks for free samples, and performs a new reservation.
- After that a wait period is introduced, while the notification from the reservation is achieved.

- Receive notification from reservation

Description: ***After clicking the RESERVE button, and filling the form, Digital Library receives a request for a reservation, and after performing a reservation process, a status message is received.***

Assumptions: RESERVE button, and sent request for reservation.

Precondition is clicked RESERVE button.
Postcondition is reservation status.

The steps are:

- The status of performing a new reservation is achieved.
- This status is sent trough the network to the user.
- Meggase about status of reservation is printed on the screen.

This use case includes "make a reservation" use case.

- Check for free samples

Description: ***Count how many samples from that book are available, and return that number to the reservation process.***

Assumptions: request for reservation, sample(s) for the book.

Precondition is new reservation.
Postcondition is number of free samples

The steps are:

- Receive a request for a number of free samples.
- Count the available samples (those one with status="available").
- Sent that number to the Digital Library.

This use case includes "make a reservation" use case.

- Return number of free samples

Description: ***Gives a number of free samples.***

Assumptions: samples of that book, request for number of free samples.

Precondition is request for number of samples.
Postcondition is transmiting the number of free samples

The steps are:

- Receive the transmited number of free samples.
- Check the result to determine the reservation status.
- Transmit the reservation status to the Digital Library.

This use case includes "check for free samples" use case.

- Make a lending

Description: ***Member wants to lend a book.***

Assumptions: user must be present in the library, book must be in paper form, a free sample for the book must be available.

Initiatin actor is Lender and benefiting actor is Member.

Precondition is request for lending.
Postcondition is transmited lending request to the Digital Library.

The steps are:

- Lender activates interface for makeing and returning a lending.
- Lender enters data for the sample and the member in the lending data entry form.
- After pressing the LEND button a new lending process is activated.

- Receive notification for lending

Description: ***After pressing the LEND button in the lending form, a new reservation record is writen to the database, and the result of this transaction is returned to the lender's interface.***

Assumptions: communication via network, and LEND button present in the interface.

Precondition is entered data in lending form.
Postcondition is status of performed transaction.

The steps are:

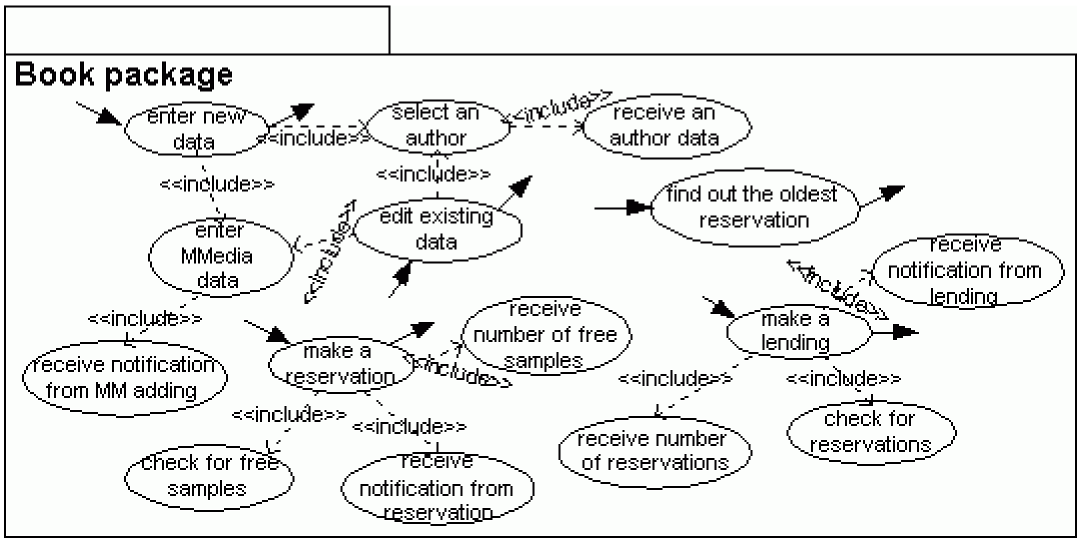
- Wait until lending process ends, and receive a lending result status.
- Transmit the status via network to the lender's screen.
- If there are free samples, member gets one.

This use case includes "make a lending" use case.

Another use cases related to the lending process are:

Receive number of reservations	This use case take place when new lending record is created. If a book has a pending reservations (free samples with status="reserved" are available), and no free samples with status="available" are present, then lending process must fail.
Check for reservation	If a member has a reservation for a book's sample, then if a holded sample is available, the member takes that sample, else a lending process will fail.
Find out the oldest reservation	This use case is used in the process of updating reservation records. When a member make a request for a reservation and there is no free sample, then when only one sample is set free, this use case take place and finds out the oldest reservation for that book, and sets the sample's status from "available" to "reserved". Sample can be set free when it is returned from a user who lends it.

Inclusions that appears here are between: "make a lending" and a "check for reservation" use cases; "make a lending" and a "receive number of reservations" use cases.



The updated use case diagrams for the Book package

Analyzing the use cases has helped conceptualize the working parts of the system. Although we know a lot about the use cases, we still have to model how those working parts will interact with one another and how (and when) they change state. Passing this information to the programmers will make their jobs a lot easier. They will have a vision of how to code classes and make them wotk together.
See Interactions in the System.

Refine the Class Diagram

Now is the time to complete our class diaram. We will analyze every class gained from previous analysis.

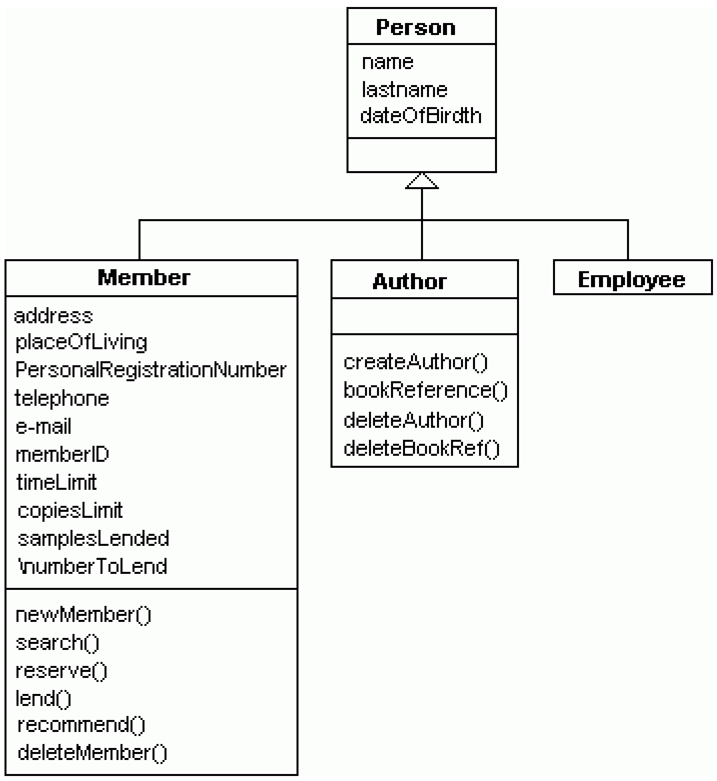
It is obvious to have a deleting process invert to the creation process. This is said because in almost every class we have a creatind operation (in Object Oriented programming known as ***constructor***), therefore an operation for deleting an object must be obtained in every class (in Object Oriented programming this is known as ***destructor***).

A new attributes must be added for a Member class. They are ***alias***, ***password*** and ***status***. Alias and password will serve as an attributes for logging onto the system. It is obvious that Internet interface will allow a lot of people to access the Digital Library, but only those who have contracted membership with the library, can browse the data, make reservations and printing the electronic items. The Status attribute is introduced to distinguish between ordinary member and library employee-member. We said that every employee also can be a member of a library.

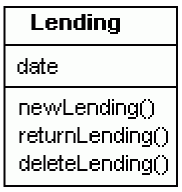
New operations for the Item class can be introduced according to the previous use case analysis. This operations are ***findOutTheOldestReservation()***, ***checkForReservation()*** and ***receiveNumberOfReservations()***, and they are described in previous lesson.

If we take a look at Member class attributes, Employee class attributes and an Author class atributes, we'll see that most of them are comon for these three classes. Therefore an inheritance may be introduced. We will create hew class ***Person*** wich will hold name, lastname and date of birdth attributes.

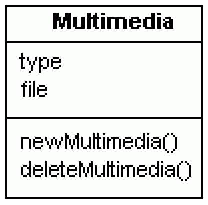
Description for classes is not longer necessary, because you are going to see all the classes with their last and final state. They are presented in the following list of classes.



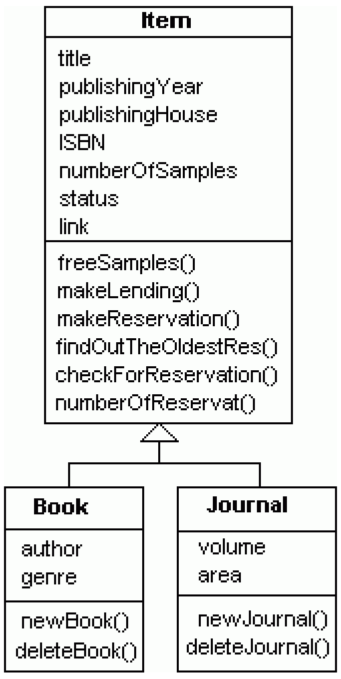
The parent class Person with it's children



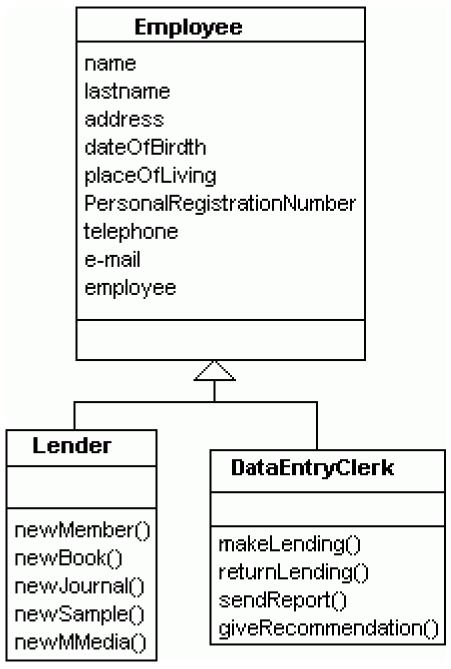
The Lending class



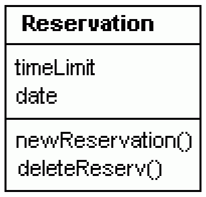
The Multimedia class



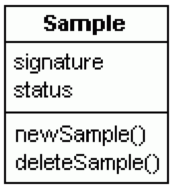
The Item class hierarchy



The Employee class and it's children



The Reservation class



The Sample class

***Click Here** to see associations between these classes!*

Interactions in the System 1

One way to find out the interactions in the system is to enumerate the system components suggested in each package of use cases. The task to perform is to show how the system components interact in order to complete each use case.

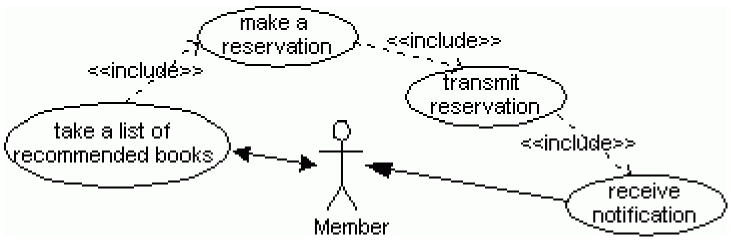
Altough we didn't explicitly analyze all the use cases in all the packages in the previous lessons, we can still extract the system components those use cases assume. Let's start to reanalyze the packages that were analyzed in previous lessons. That were the Member package and the Book package. We're going to find interactions between components of these packages, but only for reservation and lending process. Because this course is too short to comprise all the interactions between components, we'll analyze only these two interactions, and a rest of them you'll need to do like an exercise in the workshop part of the day.

The Member package

The member package use cases:

- enter data for searching
- transmit the data to the library's database
- retreive a list of items
- make a reservation for some of them
- transmit the reservation to the library's database
- receive notification from library's database
- take a list of recommended books
- take a list of recommended journals

Let's exsamine again the process of reservating the book. Suppose that you have to reserve a book from retreived list of recommended books. Now let's look at the steps necessary to perform these operations.



The part of use case diagram in the Member package

- Take a list of recommended books

The steps of this use case are:

1 Activate the recommendation part from the member's interface.
2 After clicking the RECOMMEND button on the interface, a signal is sent to the Digital Library, to activate the recommendation algorithm.
3 The Digital Library activates a ***recommendation algorithm**** which produces a list of recommended books.
4 The Digital Library sends this list of books to the member's interface.
5 Print this list of items to the member's screen.

* For the purpose of Digital Library a special algorithm for recommendation is developed. If you need some detailed information about this algorithm please refer to the webmaster of this course.

To activate the reservation process, a RESERVE button must be obtained beyond every item in the list. Pressing this button another use case sequence is activated. This sequence includes "make reservation" use case, "transmit reservation" use case, and "receive notification" use case. The steps for everyone use case are given in following tables:

- Make reservation

1 The member receives a list of items (from searching or from recommendation)
2 The member clicks the RESERVE button for the items that he or she wants to reserve
3 The form for acknowledging a reservation appears

4 The member fills in the information in the form

- Transmit reservation

1 Click on the SUBMIT button on the reservation form.
2 Digital Library transmits the reservation request to the Reservation database.
3 The request arrives in the Reservation database.
4 On the member's screen appears a message indicating "request sent", and a message " wait to receive acknowledge".

Before going to the "receive notification" use case we'll take a look at Reservation package use cases. A "new reservation" and "transmit notification" use cases will take part at this process of interactions.

- New reservation

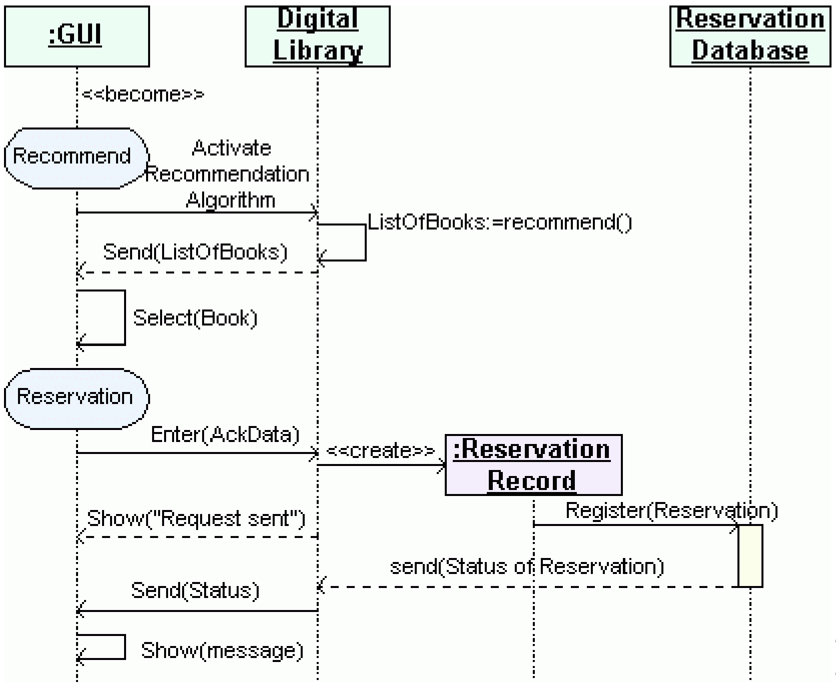
1 A request arrives in the Reservation database, and the process of creation new reservation is activated.
2 Result of this process is status of reservation, which is sent to the Digital Library.
3 Send the status of reservation to the Digital Library.

- Transmit notification

1 Digital Library sents the status of reservation to the member's interface.
2 Status arrives in the member's side.

- Receive notification

1 Member's interface receives arrived status.
2 Status is decoded.
3 A message is printed on the member's screen according to the decoded status of reservation.



The sequence diagram for reservation process of recommended book

As you can see some of use case steps are modified. These modifications are example of the way one phase of a project can influence another.
Note also the state changes on the first lifeline. It's intended to clarify how the user interface sets up to handle a particular kind of activity. We could include all the possible state changes as separate state diagrams, but this would be overkill. Putting them on the sequence diagrams appears to be more economical.

Interactions in the System 2

Suppoose that member wants to lend a book. First of all, member must be present in the library, the book for lending must be in paper format, and a free sample must be present in the library's storage. If all of these conditions are realized then a sample of book can be lended to the member. To avoid going to the library and search for free sample, the member can perform a reservation to the Digital Library system, and after receiving a notification that a free sample for the reservation is present in the library, can go to the library and make his or hers lending.

Let's look at use cases in the Book package necessary for this operation.

The Book package

The book package use cases:

- select an author
- receive an author data
- enter data for a new book
- edit data for an existing book
- enter multimedia data
- receive notification from multimedia adding
- check for free samples
- receive number of free samples
- check for reservations
- receive number of reservations
- find out the oldest reservation
- make a reservation
- receive notification for reservation
- make a lending
- transmit lending data
- receive notification for lending

For process of lending we'll need these use cases: "make a lending", "transmit lending data", "receive number of reservations", "check for reservation" and "receive notification from lending". Let's take a look at steps of every use case.

- Make a lending

1. Activate the make/return lending part from the lender's interface.
2. A lending form appears on the lender's screen.
3. The lender enters the necessary data.
4. Lending data are transmited to the Digital Library

- Transmit lending data

1. The lender clicks the SUBMIT button on the lending form, and lending data is sent to the Digital Library
2. The Digital Library receives this data, and a request for new lending is sent to the Lending database
3. The request is sent to the Lending database, and an acknowledge is waited to receive

When data is transmited to the Lending database, a mechanizam for adding a new record in the database is activated. This operation incorporates these use cases located in the Lending package: "receive request for new lending", "perform a lending" and "transmit lending result". When performing a lending, also results from the Book package use cases are obtained. This process needs results for pending reservation made by that member for that book, and this will be done only if a result from "receive number of reservations" use case is not equal to zero. Checking for pending reservations is necessary, because if a reservation is made for a book (a sample for that book is holded for the member), then when a lending is done, that reservation from Reservation database must be deleted.

- Receive request for new lending

1. The Lending database receives a request for performing a process of creating a new record
2. This activates the process of creating and adding a new record to the database

- Perform a lending

1 Database has activated its mechanizams for creating a new record
2 The result of creation a new record is aimed
3 This result is sent to the Digital Library

- Transmit lending result

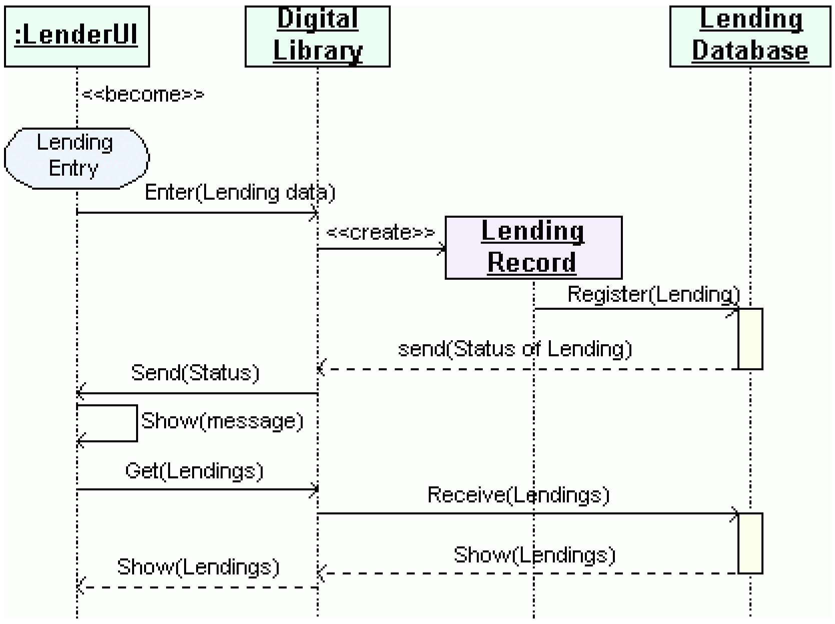
1 Result from lending is sent to the Digital Library
2 This result is transmited to the lender's interface

Now we can continue with use cases from the Book package:

- Receive notification from lending

1 Lender's interface has received result from lending
2 This result is decoded, and a message is printed on a screen
3 A request for current state in the member's lending record is sent to the Lending database
4 A list of records is retreived form the Lending database
5 This list is printed on the lender's screen

The sequence diagram for these use cases steps is shown on the next picture:



The sequence diagram for "Make a lending"

As you can see some of use cases in the given list were not used for this purpose. They are internaly used in the activation of object in the Lending database. The operation for creation a record incorporates these use cases, and after that returns a result, that is dotted line message going out from the activation in the LendingDB lifeline.

For an exercise try drawing out the sequence diagrams for "return a lending" use case from the Book package, and for "enter data for a new journal" use case from the Journal package.

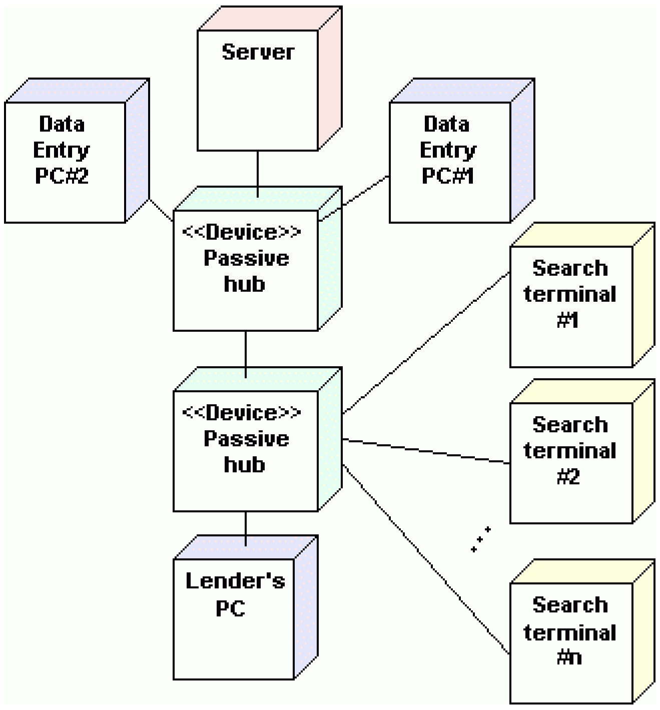
After you model interaction among components, the system is much closer to becoming reality. As you model the interactions, you may find that it's appropriate to modify the use cases at the base of these interactions. The results of this analysis should make it easy for programmers to code the system objects and how they communicate with one another.

Analyze Integration with Cooperate Systems

After the GRAPPLE analysis segment has produced the generel concept of the Digital Library system, a system engineer will strat thinking about how the physical structure should look. He or she will start considering alternative network topologies and how to implement them, and will start figuring out which software components belong on which nodes in the network.

The development team have made decision that a library local network will take part in the Digital Library system, and that an Internet interface will be implemented on the LAN's server.

First let's look at the physical architecture of the library's local area network.



The Deployment diagram for the library LAN

As you can see from this deployment diagram, library LAN consists from these hardware components:

Component name	***Amount***
Server	**1**
DataEntry PC	**2**
Lender's PC	**1**
Search terminal	**6**
Passive hub	**2**

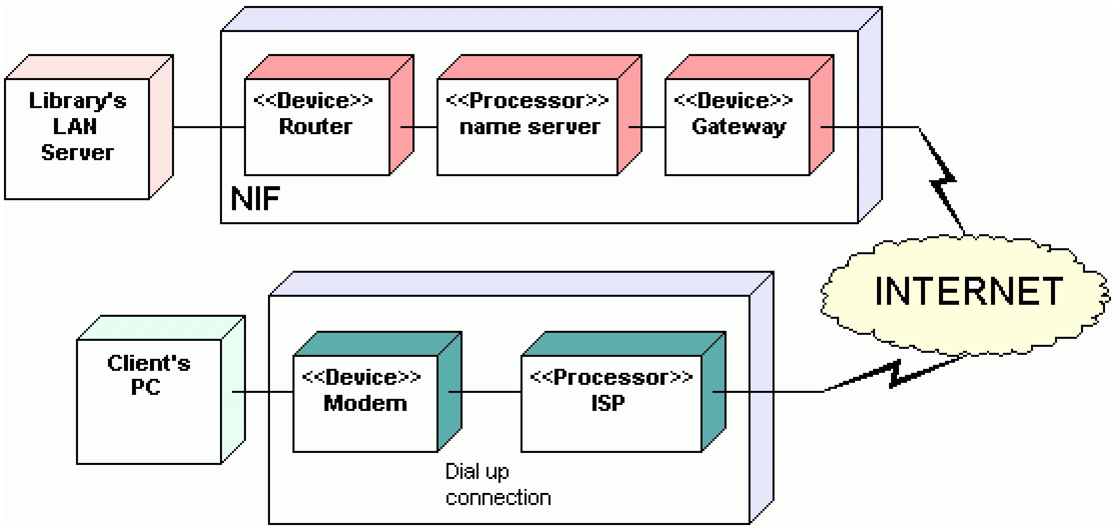
Server and DataEntry PCs are located in the same room, also one Passive hub is located in that room. A cable from this passive hub goes out from the room, to the corridor. There another passive hub is located. From this hub cables goes to the Search terminals located in the corridor, and to the Lender's PC located in the Lending area.

When we have a physical architecture in front of us, let's look at possibility od implementing an Internet interface on the Server. This interface will be used locally from the Search terminals, from Lender's PC and from DataEntry PCs and trough the Internet from member's homes or offices. Therefore a new hardware must be implemented in the library's LAN.

Library's LAN will have the same structure. New hardware will be added only to the Server's side, and that is a Network Interconnection Facility (NIF). The NIF consists of a ***name server*** (a database that validates connections), a ***router*** (a device for linking networks together), and a ***gateway*** (a device for translating information from one communication protocol to another). The gateway allows Internet members to access the Digital Library from home or from their office.

On the member's side only a computer with Internet connection, and Internet browser is necessary. Also a member must have an alias and password to make access to the Digital Library's Web page.

To ilustrate the deployment, the System Engineer delivers the deployment diagram shown on the following picture:



The Deployment diagram for the Digital Library system

Designing Look (GUI)

You've come trough a lot of use-case driven analysis. Now you're going to look some of aspects of system analysis. They are traceable to use cases, and both are extremly important to the final product. Graphical User Interfaces (GUIs) determine system usability. Let's dive into GUI desing process.

Some General Principles of GUI Design

User interface design, equals parts art and science, draws upon the vision of the graphic artist, the findings of the human factors researcher, and the intuitions of the potential user. Some general principles in GUI design are:

1.	Understand what the user has to do. User interface designers typically perform a ***task analysis*** to understand the nature of the user's work. Our use case analysis roughly corresponds to this.
2.	Make the user feel in control of the interaction. Always include the capability for the user to cancel an interaction after it's started.
3.	Give the user multiple ways to accomplish each interface-related action (like closing a window or a file) and forgive user errors gracefully.
4.	Because of cultural influences, our eyes are drawn to the upper-left corner of a screen. Put the highest priority information there.
5.	Take advantage of spatial relationships. Screen components that are related should appear near one another, perhaps with a box around them.
6.	Emphasize readability and understanding. Use active voice to communicate ideas and concepts.
7.	Limit the number of colors you use. Limit that number severely. Too many colors will distract the user from the task at hand.
8.	If you're thinking of using color to denote meaning, remember it's not always easy for a user to see an association between a color and a meaning.
9.	As is the case with the color, limit your use of fonts. Avoid italics and ornate fonts.
10.	Try to keep components (like the buttons and list boxes) the same size as much as possible. If you use different-size components, a multiplicity of colors, and a variety of fonts, you'll create a patchwork that GUI specialists call a ***clown-pants design***.
11.	Left-align components and data fields - line them up according to their left-side edges. This minimizes eye movements when the user has to scan the screen.
12.	When the user has to read and process information and then click a button, put the buttons in a column to the right of the information or in a row below and to the right of the information. This is consistent with the natural tendency to read left to right. If one of the buttons is a default button, highlight it and make it the first button in the set.

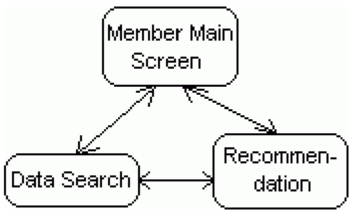
The GUI JAD Session

For this session, you recruit potential users of the system. For the Digital Library we'd recruit members, lenders and data entry clerks. The users' participation in the session is a two-part affair. In the first part, they derive the user interface screens. In the second, they approve prototypes generated by the development team.

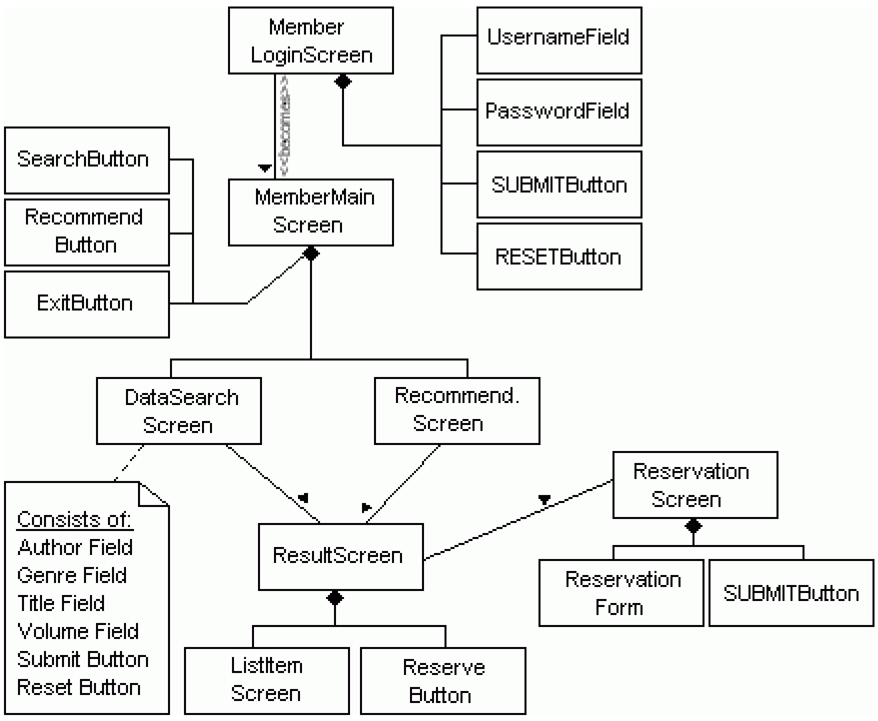
How the users derive the screens? The facilitator suggests a use case to strat from, and the users discuss ways to implement that use case via the system. When they're ready to start talking at the level of a specific screen, the users work with paper mockups. The facilitator provides a large sheet of easel paper in landscape view to represent the screen. Post-is note stickies represent the GUI components. The users' task is to work as a group to position the components appropriately.

When they reach agreement on which components should be on a screen and where those components should be located, development team members create prototype screens. As they work, they use appropriate GUI principles. Then, they present those screens on computers, and the users make any necessary modifications.

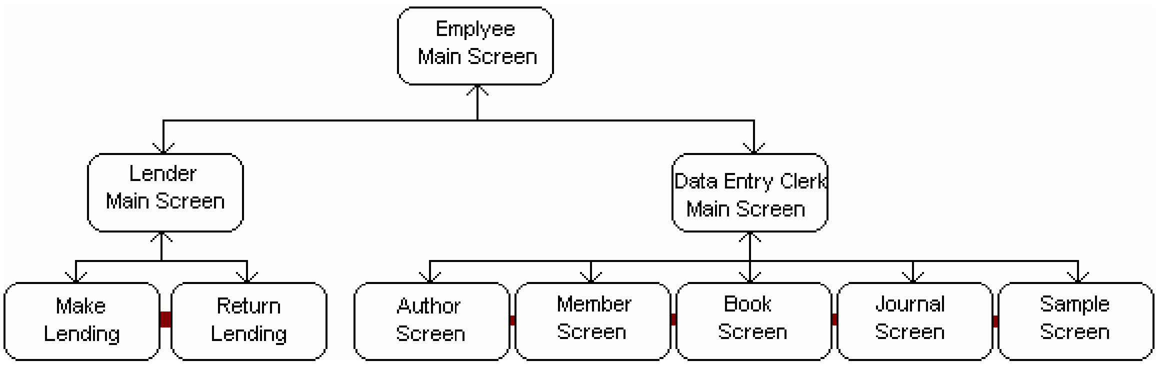
It's your turn to draw out the users' screens. Let your imagination and GUI principles guide you when constructing these screen shots. To make your work easier a set of UML diagrams for the member, lender and data entry clerk screens are given to you. Look at these diagrams and draw out necessary screen shots. DataEntry clerk's diagrams are too large and complex, and if you want you may draw out these screen shots or not.



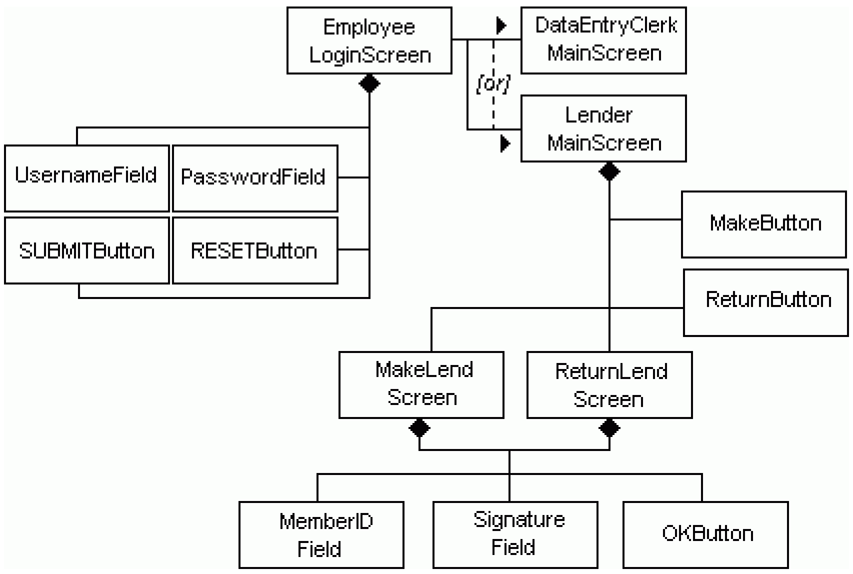
The UML state diagram for the Member interface



The UML composite diagram for the Member screen hierarchy



The UML state diagram for high-level screen flow in the Employee interface



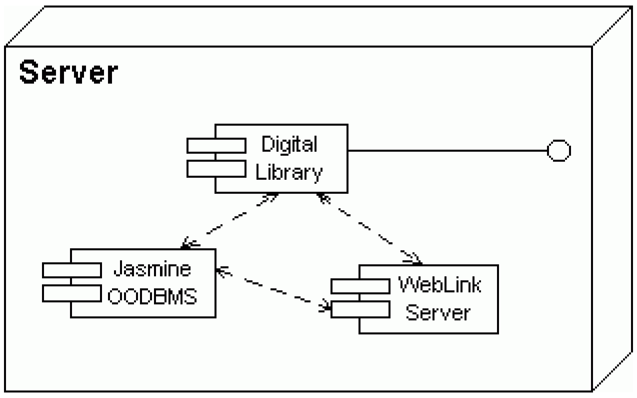
The UML composite diagram for the Employee-Lender screen hierarchy

Something About Implementation of Digital Library

When all appropriate analysis and design parts of GRAPPLE are complete, the team complies its result into a design document and hads off copies to the client and to the programmers. It then falls to the programmers to start turning the design into code.After writing the code it will be teste, rewriten according to the results of the tests, and retested - a process that will continue back and forth until the code passes all tests. The use case analysis forms the basis of the tests.

The one possible implemented Digital Library that you'll wisit after today's workshop, is coded in Jasmine OODBMS. Jasmine is used because it is Object Oriented language, has possibility of adding Multimedia data into your database, and allows you to create stand alone applications, or applications that have Web interfaces. Other agvantages of using Web interface are client software shouldn't be installed to the client's machine (client only should have an Internet browser) and Web software doesn't depend on the hardware platform that runs on.

The Server side in the deployment diagram developed for the purpose of Digital Library, now will look like:



The server node with part of its components

* The Jasmine OODBMS and WebLink are parts from the Jasmine package. With Jasmine OODBMS we created necessary classes, and with WebLink we implemented the web interface.

Document specialists begin creating documentation for the system, and they create training materials as well. A godd document creation effort should proceed like a good system development effort, with careful planing, analysis, and testing, and should begin early in the development process.

The main idea of modeling one system is to focus intense efforts on analysis and design so that implementation confronts as few challenges as possible and the result of the project is a system that fully meets the client's needs.

Questions & Answers

We reached the end of this course. Answer these questions, and compare your answer with the given possible answer.

1	**What we want to show for every scenario in the use case, to perform use case analysis?** Click for the answer
2	**What are the parts of a typical use case diagram?** Click for the answer
3	**Why would the original use cases fail to capture all the nuances in the first place?** Click for the answer
4	**In a sequence diagram, how do you show an "activation", and what does it represent?** Click for the answer
5	**What UML diagrams are useful for describing user interfaces?** Click for the answer

For each scenario in use case, we'll want to show:
- A brief description of the scenario
- Assumptions for the scenario
- The actor who initiates the use case
- Preconditions for the use case
- System-related steps in the scenario
- Postconditions when the scenario is complete
- The actor who benefits from the use case.

This closes the window

The parts of every use case diagram are:
- The initiating actor
- The use case and
- The benefiting actor.

This closes the window

Because they're results of JAD sessions with system users, not system developers. You'll notice all the additions and changes were system-related, not business-related. After you finish the sessions with the potential users and have a chance to analyze the use cases, it's not uncommon that modifications like these emerge.

This closes the window

An activation is represented as a small narrow rectangle on an object's lifeline. It represents the time period during which the object performs an action.

This closes the window

State diagrams and Composite diagrams (these are parts of Class diagrams) are appropriate when designing the possible user interfaces.

This closes the window

Workshop

This is your last workshop. Please answer these questions, and solve the problems. I don't think that you'll have any troubles.

1	Does it ussually happen to modify use case diagrams and class diagram?
2	Why collaboration diagrams were not used in the process of analysis the Digital Library?
3	Give three reasons for limiting the use of color in a GUI.
4	We did analysis for Book and Member package use cases. Try yourself in a role of analyst and make analysis for the Employee and Lending packages.
5	Try drawing out the sequence diagrams for "return a lending" use case from the Book package, and for "enter data for a new journal" use case from the Journal package.
6	This is not an exercise. You may visit one of possible Digital Library systems developed by using this model. Just follow this L I N K (This opens a new window). How do you like it?

Congratulations!

You have finished your UML learning material. It's time to relax yourself and browse the Internet to see what others made about UML learning.

We give you these addresses because learning some material is process with no end. For advanced users this is great advantage to strength their knowledge, and become an UML experts.

Let's start with our journey. To visit the site just click the picture or given Internet address.

UML™ Resources

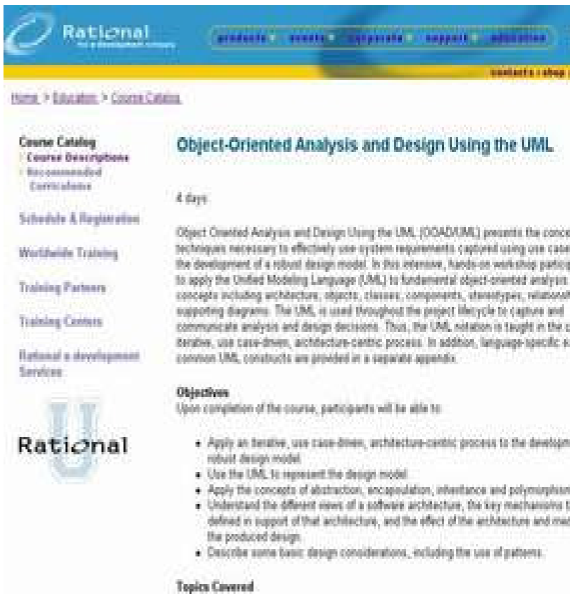
UML 1.3 Documentation

- OMG UML v. 1.3 specification is available in the following formats:
  - PostScript (10 MB)
  - Zipped PostScript (1.5 MB)
- Archived information referencing the UML™ v.1.1

UML for XML Schema Mapping Specification

- Word document (89 KB)
- PDF document (60 KB)
- Read the press release

Rational WEB site



Object-Oriented Analysis and Design Using the UML

4 days

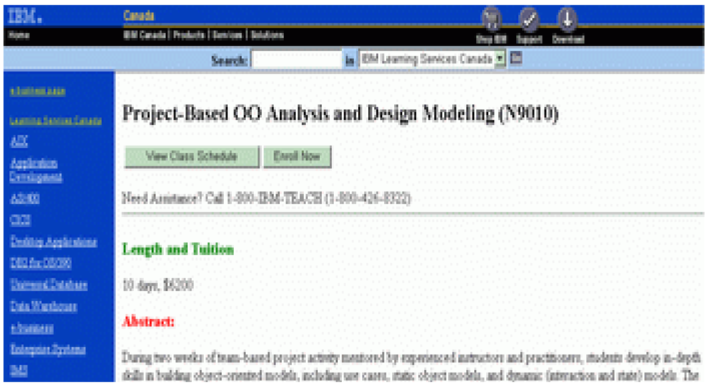
Object Oriented Analysis and Design Using the UML (OOAD/UML) presents the conce
techniques necessary to effectively use system requirements captured using use case
the development of a robust design model. In this intensive, hands-on workshop partic
to apply the Unified Modeling Language (UML) to fundamental object-oriented analysis
concepts including architecture, objects, classes, components, stereotypes, relations
supporting diagrams. The UML is used throughout the project lifecycle to capture and
communicate analysis and design decisions. Thus, the UML notation is taught in the c
iterative, use case-driven, architecture-centric process. In addition, language-specific e
common UML constructs are provided in a separate appendix.

Objectives
Upon completion of the course, participants will be able to:

- Apply an iterative, use case-driven, architecture-centric process to the developm
robust design model.
- Use the UML to represent the design model.
- Apply the concepts of abstraction, encapsulation, inheritance and polymorphism
- Understand the different views of a software architecture, the key mechanisms t
defined in support of that architecture, and the effect of the architecture and me
the produced design.
- Describe some basic design considerations, including the use of patterns.

Topics Covered

Rational WEB site



Project-Based OO Analysis and Design Modeling (N9010)

Need Assistance? Call 1-800-IBM-TEACH (1-800-426-8322)

Length and Tuition

10 days, $6200

Abstract:

During two weeks of team-based project activity mentored by experienced instructors and practitioners, students develop in-depth skills in building object-oriented models, including use cases, static object models, and dynamic (interaction and state) models. The

IBM Canada WEB site

Genesis Development Corporation WEB site

Object-Oriented Business Pr
Modeling

Audience: Analysts, designers, implementors, business represent
managers

Prerequisite: General knowledge of Object Technology

Purpose: This course covers business process modeling and speci
attendees to capture requirements through use case analysis, det
appropriate business processes and business objects and develop
between the objects to faithfully describe your current business
there, business process reengineering principles may be applied
refinement of the business processes utilized. Tools selection cri
corporate practices are also reviewed.

Duration: Three to Five Days (customized to your needs)

Copyright © 1999 Essential Strategies, Inc.

The Unified Modeling Language (UML)

The Unified Modeling Language (UML) is not billed as "data modeling" techniques, but as an "object modeling" technique. Instead of entities, it models "object classes". Close examination of its models, however, shows these to look suspiciously like entity/relationship models. Indeed, Ivar Jacobson even calls these classes ***entity objects***. [Jacobson 1992, p. 132]

Because of a confluence of ideas, techniques, personalities, and politics, UML promises to become a standard notation for representing the structure of data in the object-oriented community. It was developed when the "three amigos" of the object-oriented world, James Rumbaugh, Grady Booch, and Ivar Jacobson, among others, agreed to adopt as standard a variation on a notation originally developed by David Embley and his colleagues [*Embley, et al. 1992*]. The UML was published by the Object Management Group in 1997 [*OMG 1998*]. Messers. Rumbaugh, Jacobson, and Booch have written what are purported to be the authoritative texts on UML [*Rumbaugh, Jacobson, & Booch 1999*] [*Jacobson, Booch, & Rumbaugh 1999*] [*Booch, Rumbaugh, & Jacobson 1999*], although many books on the subject are available.

As a system of notation for representing the structure of data, the UML ***static diagram*** is functionally the exact equivalent of any other data modeling, entity/relationship modeling, or object modeling technique. Its ***classes*** of ***entity objects*** are really entities, and its ***associations*** are relationships. It has specialized symbols for some things that are already represented by the main symbols in other notations, and it lacks some symbols used in e/r diagrams. It does, however, have a more extensive ability to describe inter-relationship constraints.

Essential Strategies WEB site

Acknowledgement

Some paragraphs in this tutorial are extracted from the book:

"**Sams Teach Yourself UML in 24 Hours**" writen by Joseph Schmuller
Sams Publishing 1999

Other literature used to form this interactive tutorial is diploma work:

"**Digital Library - concept and implementation**" by Slobodan Kalajdziski
University "St. Cyril and Methodius" Skopje 2000

Some of examples are product of imagination of the author.

Special thanks to
prof.D-r.Danco Davcev and
ass.M-r.Vladimir Trajkovic from Faculty of Electrical Engineering - Skopje