Systems Analysis and Design Functional / Structural Modeling

GRAND VALLEY STATE UNIVERSITY
www.gvsu.edu

CIS641

Erik Fredericks // frederer@gvsu.edu

Adapted from materials provided by Gregory Schymik and the textbook (Systems Analysis and Design 5th/6th Ed.)

Outline!

CRC cards
UML diagrams
Class diagrams
Object diagrams

Relationships between models

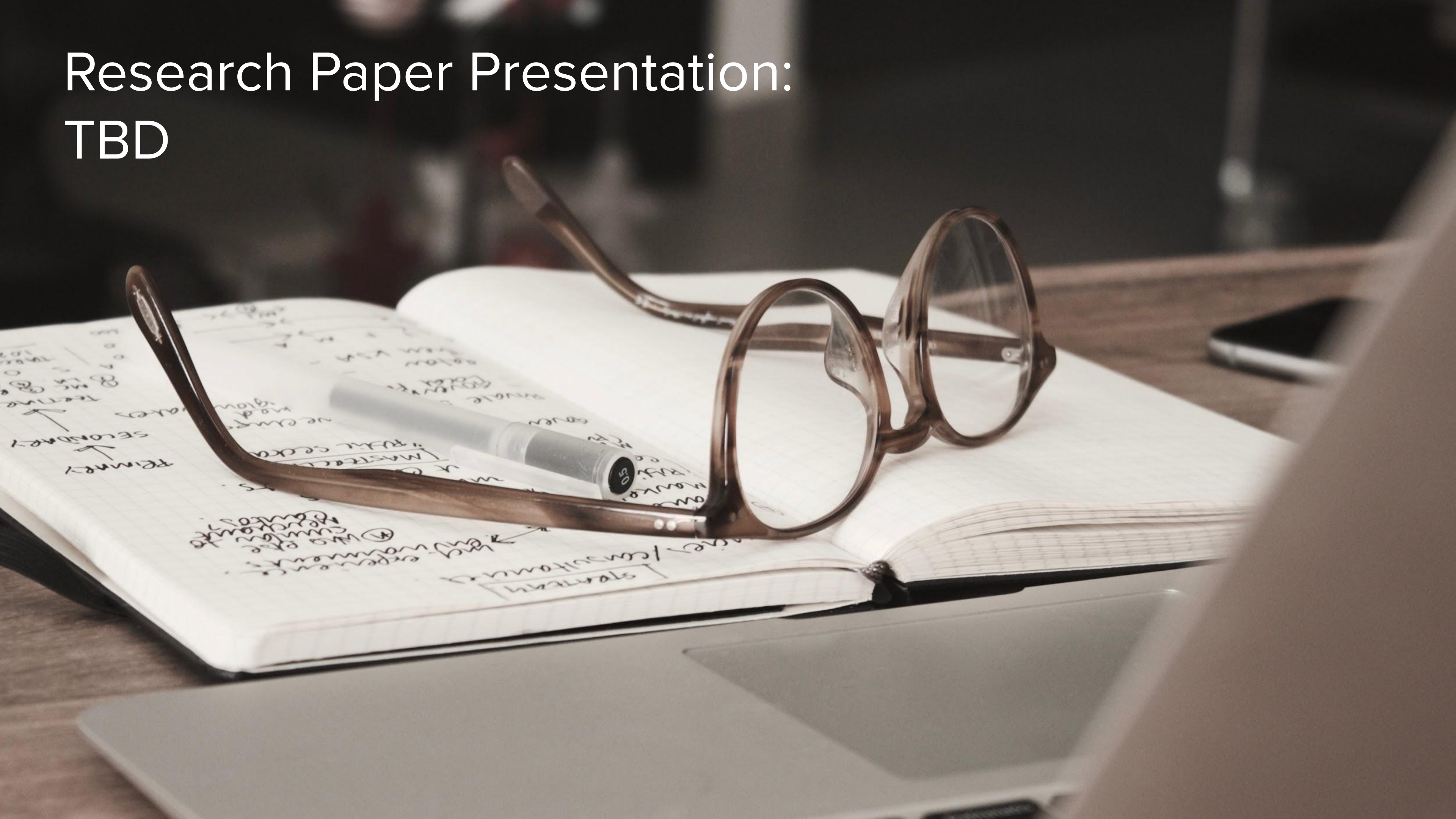
Resources ➔ Chapter 4!

- Jurgen Borstler's paper: OO Analysis and Design Through Scenario Role-Play https://www8.cs.umu.se/kurser/TDBA63/CRC_UMINF04.04.pdf

Resources on the UML

- http://www.uml.org/index.htm
- Find your own! There may be better resources on this for you out there

Research Paper Presentation: TBD



Requirements vs. User Stories

Requirement:

- Something the system must do to meet a business need/objective or contractual obligation

User Story (Agile):

- (Brief) statement of intent describing what the system must do for a user

Structural modeling

Or, let's take our requirements and turn them into *functional* models

Remember: models are **logical**

- Independent of implementation → hand-drawn or made on computer

Develop **use cases** from requirements

- Use case → business system interacting with environment
- Diagrams/descriptions of how users perform discrete activities

Develop **activity diagrams** from use cases

- Model business processes
- Depict **flow of data** between activities

Both of these models...

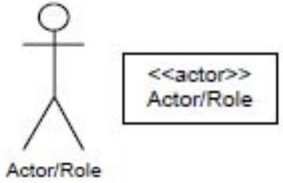
Discuss domain without implementation

- AKA problem domain models

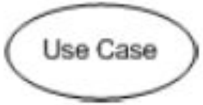
Physical models (later on) will tell us *how* implementation is supposed to be done

- For now, we care about *describing the problem*

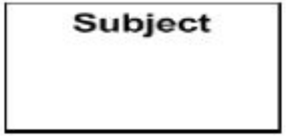
Use case diagrams



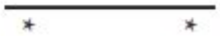
- Actors: user or system that interacts
    - Association: lines connecting actors and use cases
    - Inclusions, extensions, generalizations



- Major process in system (that gives a user benefit)

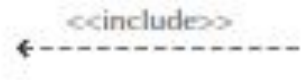


- Box describing scope of system

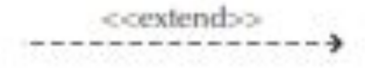


- Links an actor and a use case

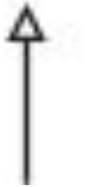
Use case diagrams



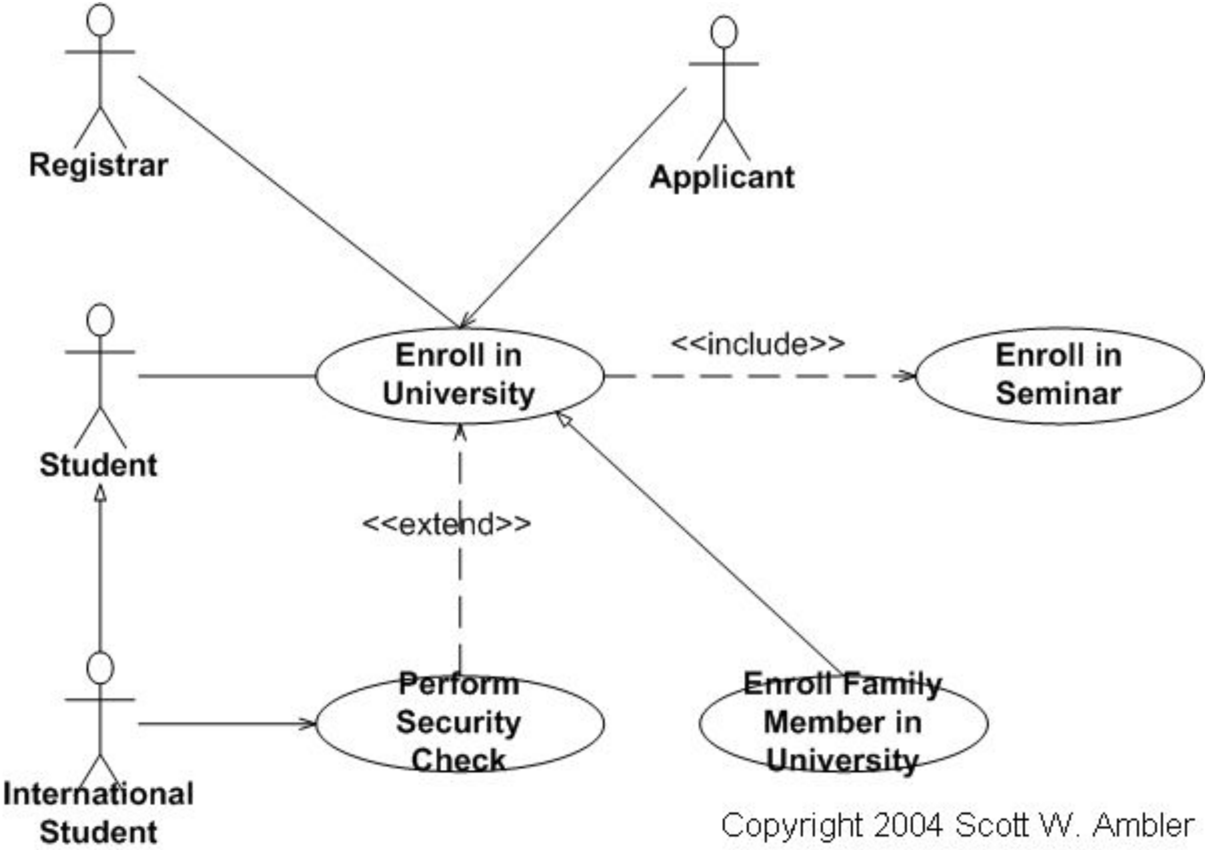
- Inclusion of one use case's functionality in another



- Extension of use case for optional behavior



- Specialized use case to be more generalized


Registrar
Applicant
Student
International Student
Enroll in University
<<include>>
Enroll in Seminar
<<extend>>
Perform Security Check
Enroll Family Member in University

How do we identify use cases?

Review requirements definition

Identify subject boundaries

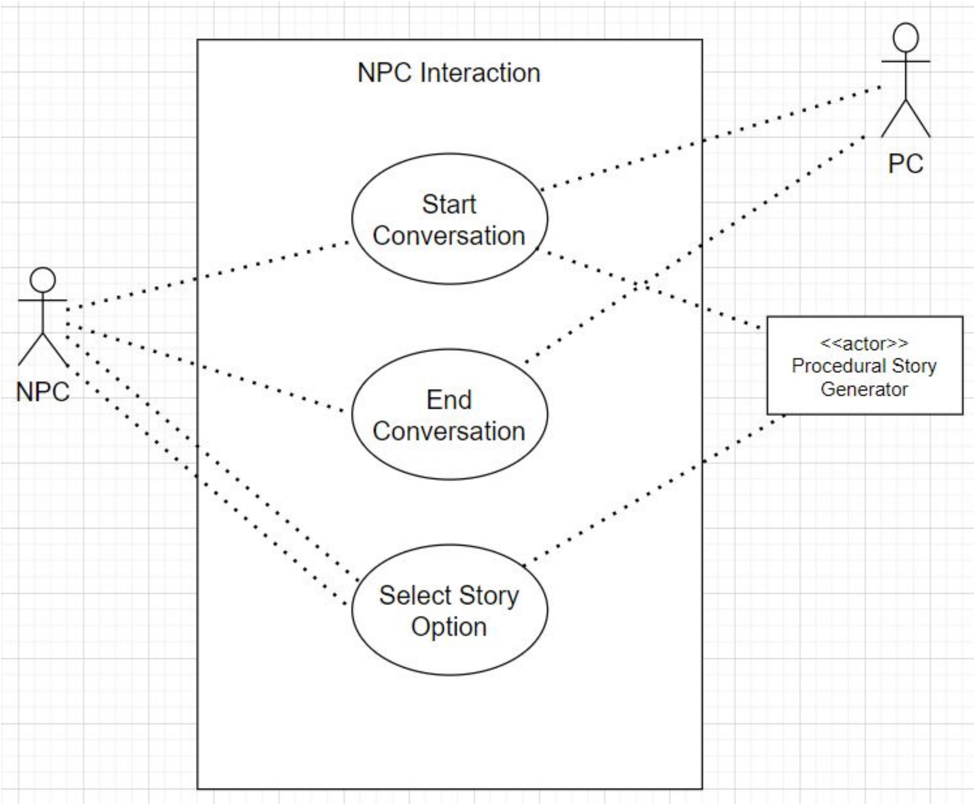
Identify primary actors and their goals

Identify business processes and major use cases

Carefully review current set of use cases

- Split/combine as necessary
- Identify additional (i.e., distinct) use cases

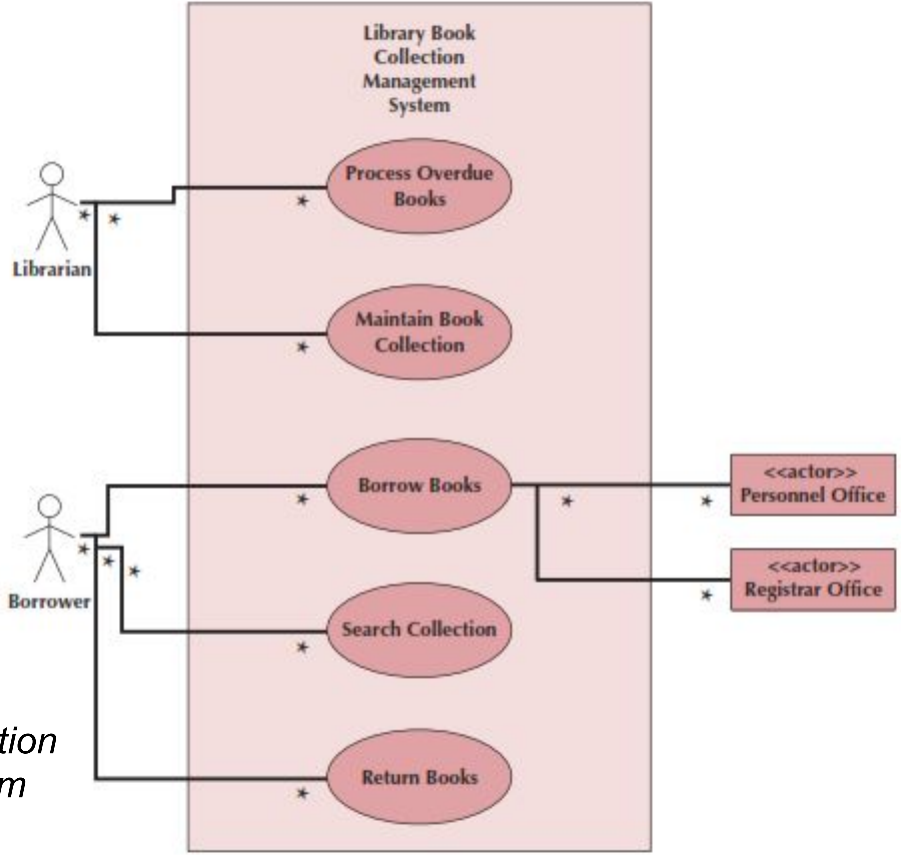
EXAMPLE


NPC Interaction
Start Conversation
End Conversation
Select Story Option
PC
NPC
<<actor>> Procedural Story Generator

Heuristic (adapt to your needs)

1) Place use cases
2) Place actors
3) Draw subject boundary
4) Add associations

(try to limit each diagram to 3-8 use cases)



Library Book Collection Management System Use Case Diagram

Team task:

Develop **2** use case diagrams for your term project

Requirements:

1) The diagrams should describe a **unique feature**
2) There should be at least **three** use cases per diagram (max eight)

~15-20 minutes

- Summon me if you have questions!

1) Who are the **actors**
2) What is the **boundary**
3) What are the **use cases**
4) Are there any **associations?**

But wait, there's more!

~~Now, I am going to shuffle a random person into each room~~

1) ~~Describe your use case diagrams to the newcomer~~
2) ~~Newcomer: give **constructive criticism**~~
3) ~~Together, come up with a **third** feature for that team!~~

~~Come up with **three** use case diagrams and turn them in Friday (10/01) by midnight~~

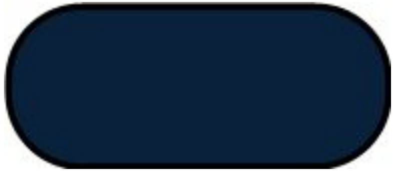
From use cases to activity (diagrams) (AD)

What is an activity again?

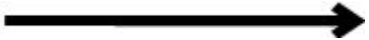
Activity diagram ➔ **sequences of activities/actions**

- Abstract and describe generalities
- Model behavior **independent** of objects
- Use for **any** type or process!

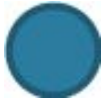
Activity diagram "parts"



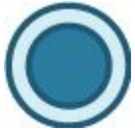
- Action/activity



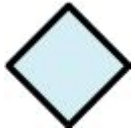
- Control flow



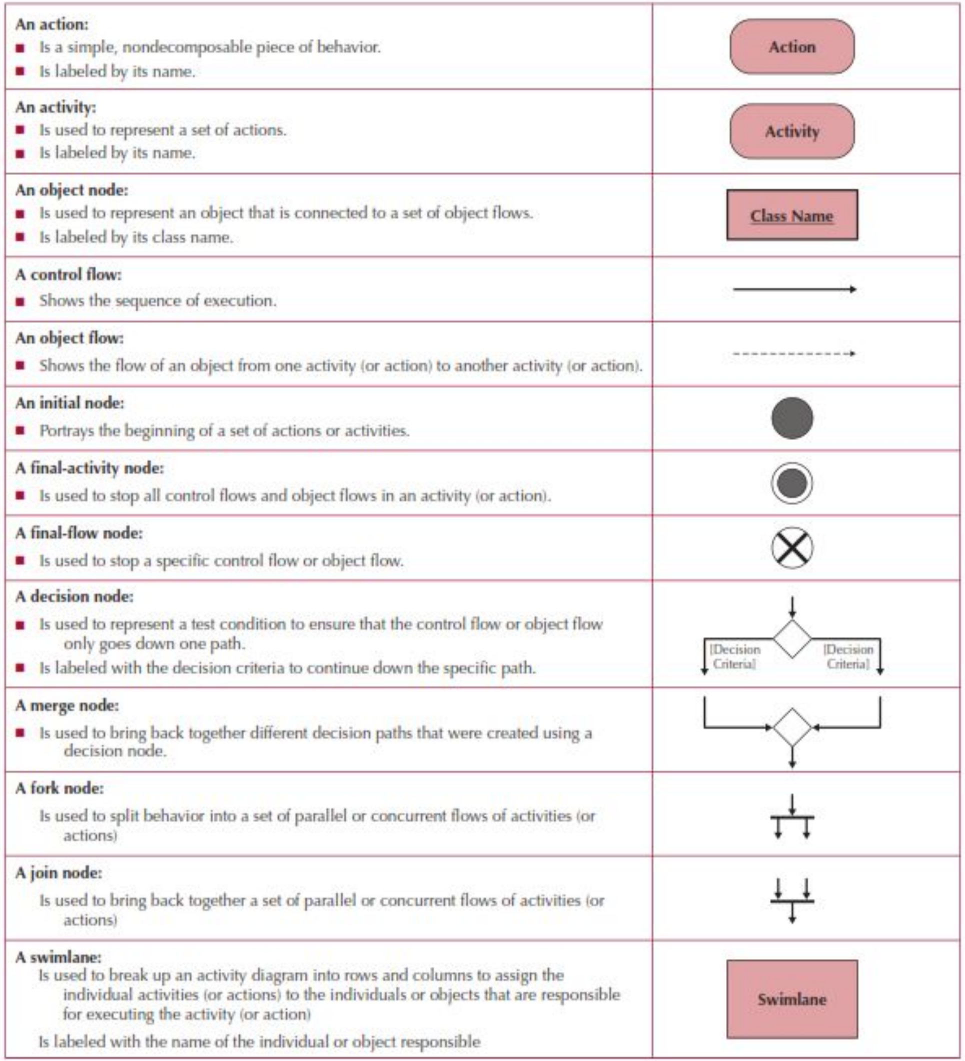
- Start (initial) node



- End (final) node

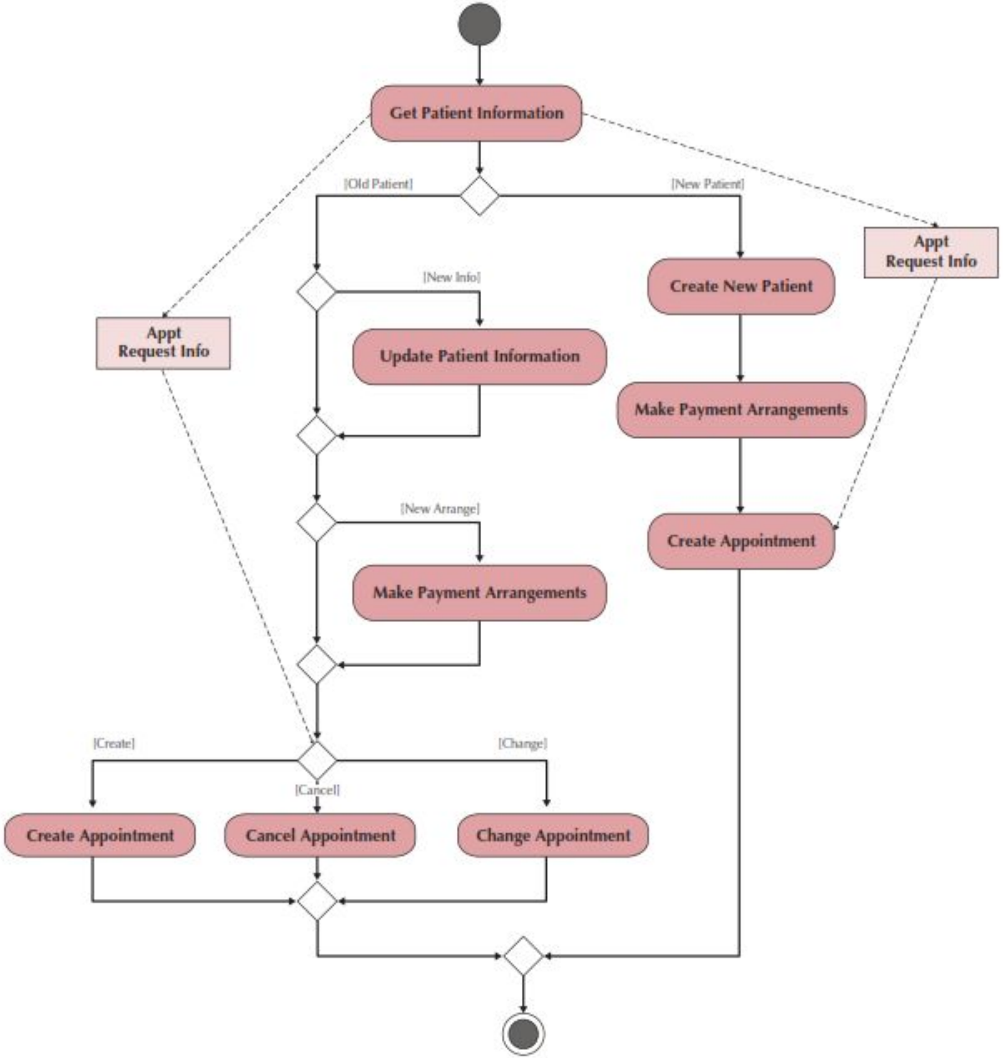


- Decision



An action: ■ Is a simple, nondecomposable piece of behavior. ■ Is labeled by its name.	Action
An activity: ■ Is used to represent a set of actions. ■ Is labeled by its name.	Activity
An object node: ■ Is used to represent an object that is connected to a set of object flows. ■ Is labeled by its class name.	<u>Class Name</u>
A control flow: ■ Shows the sequence of execution.	→
An object flow: ■ Shows the flow of an object from one activity (or action) to another activity (or action).	⇢
An initial node: ■ Portrays the beginning of a set of actions or activities.	●
A final-activity node: ■ Is used to stop all control flows and object flows in an activity (or action).	◉
A final-flow node: ■ Is used to stop a specific control flow or object flow.	⊗
A decision node: ■ Is used to represent a test condition to ensure that the control flow or object flow only goes down one path. ■ Is labeled with the decision criteria to continue down the specific path.	[Decision Criteria] [Decision Criteria]
A merge node: ■ Is used to bring back together different decision paths that were created using a decision node.	◇
A fork node: Is used to split behavior into a set of parallel or concurrent flows of activities (or actions)	
A join node: Is used to bring back together a set of parallel or concurrent flows of activities (or actions)	
A swimlane: Is used to break up an activity diagram into rows and columns to assign the individual activities (or actions) to the individuals or objects that are responsible for executing the activity (or action) Is labeled with the name of the individual or object responsible	Swimlane

Sample AD

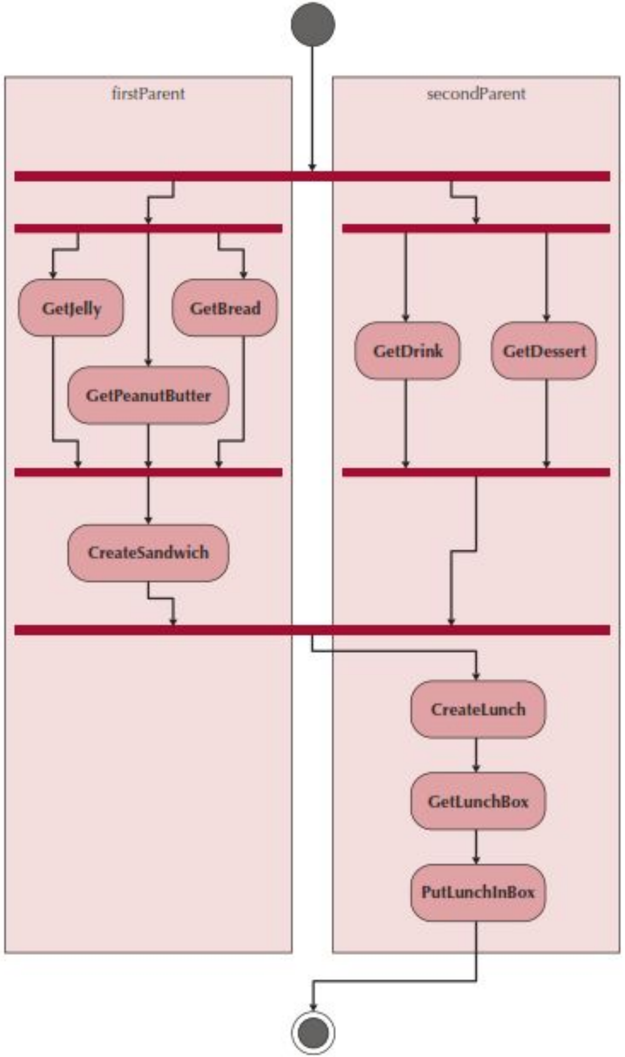

Get Patient Information
[Old Patient]
[New Patient]
Appt Request Info
[New Info]
Create New Patient
Appt Request Info
Update Patient Information
Make Payment Arrangements
[New Arrange]
Create Appointment
Make Payment Arrangements
[Create]
[Change]
[Cancel]
Create Appointment
Cancel Appointment
Change Appointment

Swim lanes

Assign responsibility to objects/individuals that perform the activity

Separation of roles

Vertical or horizontal, depending on what works best for you



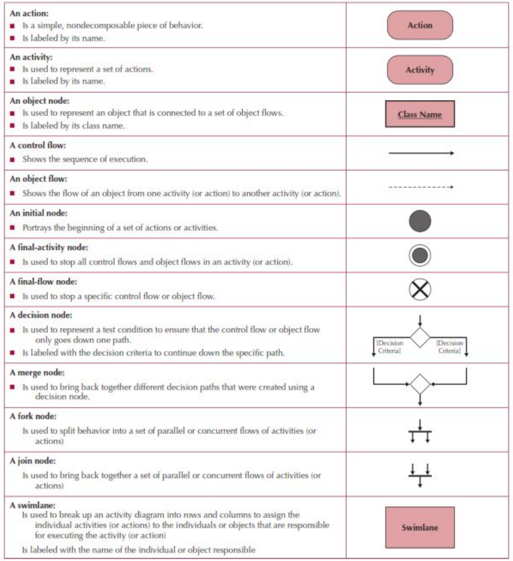
AD guidelines

1. Scope the modeled activity
2. Identify activities and connect with flows
3. Identify (any) decisions that are needed
4. Identify potential parallelisms
5. Draw the diagram!

Build an activity diagram for your term project

(This will be due in a few days!!!)

1) What is the activity and what do you need to model it?



An action: ■ Is a simple, nondecomposable piece of behavior. ■ Is labeled by its name.	Action
An activity: ■ Is used to represent a set of actions. ■ Is labeled by its name.	Activity
An object node: ■ Is used to represent an object that is connected to a set of object flows. ■ Is labeled by its class name.	Class Name
A control flow: ■ Shows the sequence of execution.	⟶
An object flow: ■ Shows the flow of an object from one activity (or action) to another activity (or action).	- - - - - - →
An initial node: ■ Portrays the beginning of a set of actions or activities.	●
A final-activity node: ■ Is used to stop all control flows and object flows in an activity (or action).	◉
A final-flow node: ■ Is used to stop a specific control flow or object flow.	⊗
A decision node: ■ Is used to represent a test condition to ensure that the control flow or object flow only goes down one path. ■ Is labeled with the decision criteria to continue down the specific path.	[Decision Criteria] ◇ [Decision Criteria]
A merge node: ■ Is used to bring back together different decision paths that were created using a decision node.	◇
A fork node: Is used to split behavior into a set of parallel or concurrent flows of activities (or actions)	
A join node: Is used to bring back together a set of parallel or concurrent flows of activities (or actions)	
A swimlane: Is used to break up an activity diagram into rows and columns to assign the individual activities (or actions) to the individuals or objects that are responsible for executing the activity (or action) Is labeled with the name of the individual or object responsible	Swimlane

Come up with **two activities** and turn them in Friday (TBD) by midnight (with your use cases)

Back to use cases

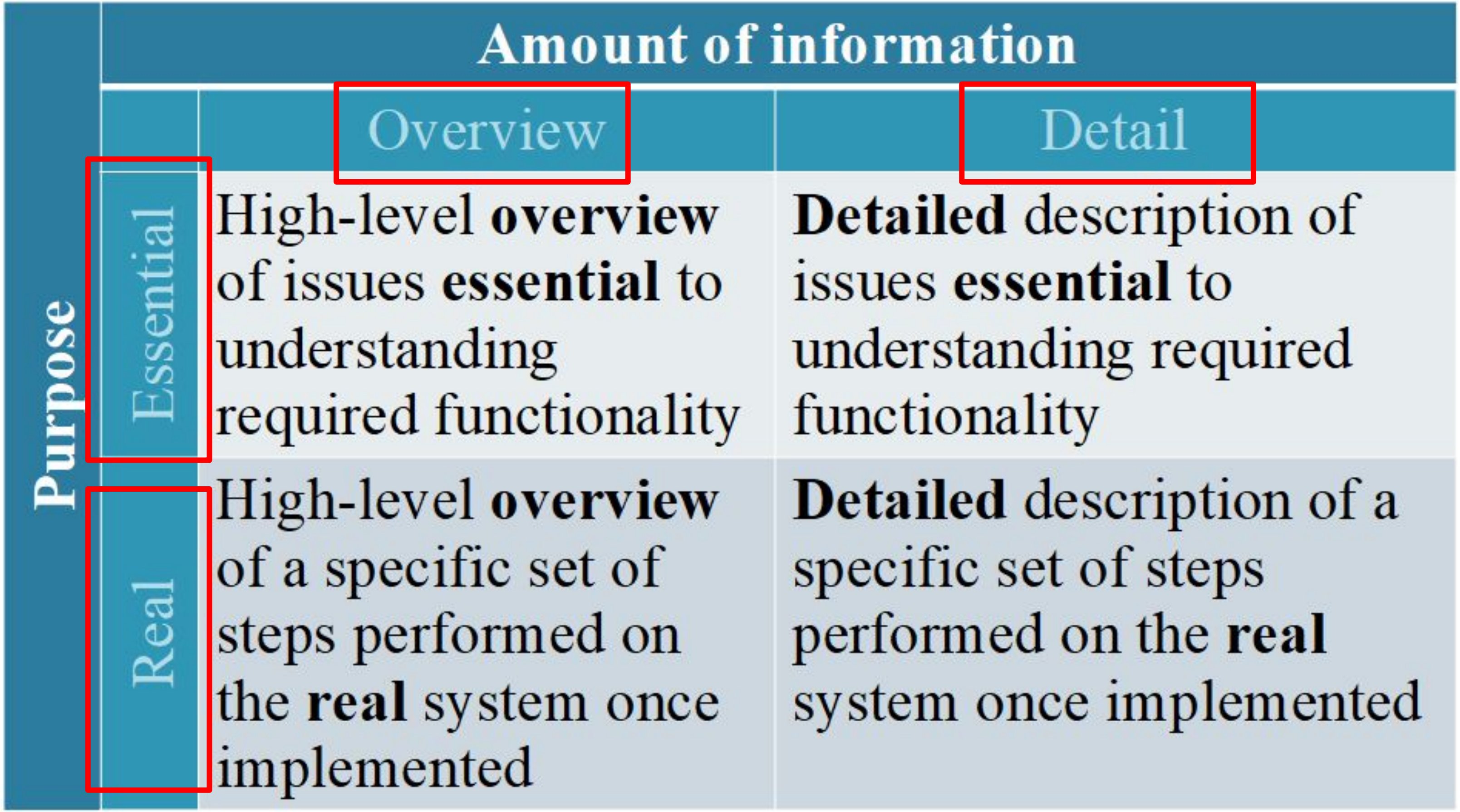
Important -- this is the primary aspect of all UML activities!

- Defines actions performed by users
- Describes basic system functions

1) Use cases are the **building blocks** of the whole system

2) Each use case describes **one function**
   a) And one function only!

Types of use cases



		Amount of information	
		Overview	Detail
Purpose	Essential	High-level **overview** of issues **essential** to understanding required functionality	**Detailed** description of issues **essential** to understanding required functionality
	Real	High-level **overview** of a specific set of steps performed on the **real** system once implemented	**Detailed** description of a specific set of steps performed on the **real** system once implemented

Use case description

Overview:

- Name, ID Number, Type, Primary Actor, Brief Description, Importance Level, Stakeholder(s), Trigger(s)

Relationships:

- Association: Communication between the use case and the actors
- Extend: Extends the functionality of a use case
- Include: Includes another use case
- Generalization: Allows use cases to support inheritance

Flow of events

- Normal flow: the usual set of activities
- Sub-flows: decomposed normal flows to simplify the use-case
- Alternate or exceptional flows: those not considered the norm

Optional characteristics (complexity, time, etc.)

Use case description guidelines

1. Write in the form of subject-verb-direct object
2. Make sure it is clear who the initiator of the step is
3. Write from independent observer's perspective
4. Write at about the same level of abstraction
5. Ensure the use case has a sensible set of steps
6. Apply the KISS principle liberally.
7. Write repeating instructions after the set of steps to be repeated

Creating a use case description

1. Pick a high priority use-case and create an overview:
   - List the primary actor
   - Determine its type (overview or detail; essential or real)
   - List all stakeholders and their interests
   - Determine the level of importance of the use case
   - Briefly describe the use case
   - List what triggers the use case
   - List its relationship to other use cases

2. Fill in the steps of the normal flow of events required to complete the use-case

Creating a use case description

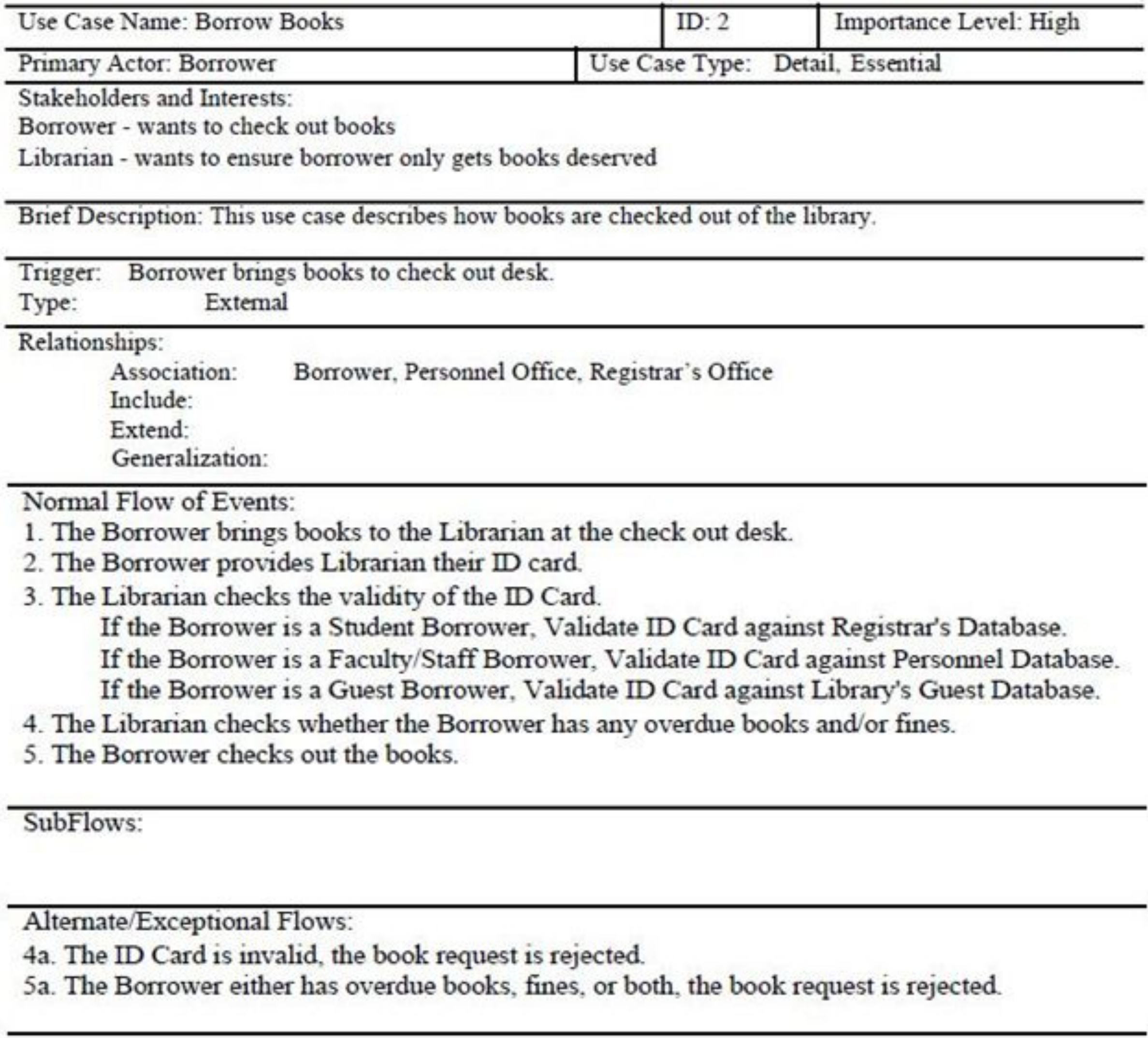
1. Ensure that the steps listed are not too complicated or long and are consistent in size with other steps

2. Identify and write the alternate or exceptional flows

3. Carefully review the use-case description and confirm that it is correct

4. Iterate over the entire set of steps again

Sample use case description



Use Case Name: Borrow Books | ID: 2 | Importance Level: High

Primary Actor: Borrower | Use Case Type: Detail, Essential

Stakeholders and Interests:
Borrower - wants to check out books
Librarian - wants to ensure borrower only gets books deserved

Brief Description: This use case describes how books are checked out of the library.

Trigger: Borrower brings books to check out desk.
Type: External

Relationships:
Association: Borrower, Personnel Office, Registrar's Office
Include:
Extend:
Generalization:

Normal Flow of Events:

1. The Borrower brings books to the Librarian at the check out desk.
2. The Borrower provides Librarian their ID card.
3. The Librarian checks the validity of the ID Card.
   If the Borrower is a Student Borrower, Validate ID Card against Registrar's Database.
   If the Borrower is a Faculty/Staff Borrower, Validate ID Card against Personnel Database.
   If the Borrower is a Guest Borrower, Validate ID Card against Library's Guest Database.
4. The Librarian checks whether the Borrower has any overdue books and/or fines.
5. The Borrower checks out the books.

SubFlows:

Alternate/Exceptional Flows:
4a. The ID Card is invalid, the book request is rejected.
5a. The Borrower either has overdue books, fines, or both, the book request is rejected.

Verifying and validating a use case

Use-cases must be verified and validated before beginning structural and behavioral modeling

Utilize a walkthrough:

- Perform a review of the models and diagrams created so far

- Performed by individuals from the development team and the client (very interactive)
  - **Facilitator**: schedule and set up the meeting
  - **Presenter**: the one who is responsible for the specific representation being reviewed
  - **Recorder** (scribe) to take notes and especially to document errors

Rules for verification and validation

1. Ensure one recorded event in the flows of the use-case description for each action/activity on the activity diagram

2. All objects in an activity diagram must be mentioned in an event of the use-case description

3. The sequence of the use-case description should match the sequence in the activity diagram

4. One and only one description for each use-case

Rules pt. 2

5. All actors listed in a use-case description must be shown on the use-case diagram

6. Stakeholders listed in the use-case description may be shown on the use-case diagram (check local policy)

7. All relationships in the use-case description must be depicted on the use-case diagram

8. All diagram-specific rules must be enforced

On to Chapter 5!

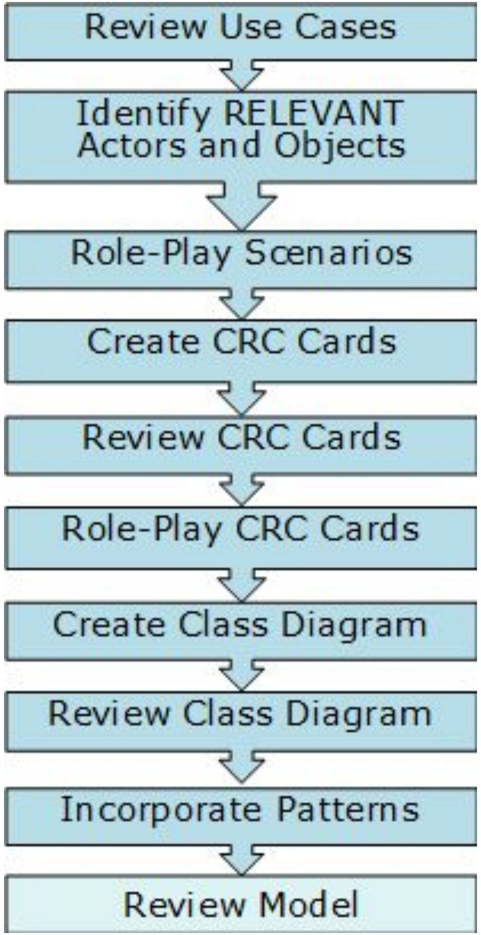
But first!

TBD

A process!

Very iterative!

- We'll see this as we go



Models

Functional models ➔ represent system **behavior**

Structural models (i.e., what we're doing now) ➔ represent **objects and their relationships**

- People
- Places
- Things
- etc.

Structural models

Drawn **iteratively**

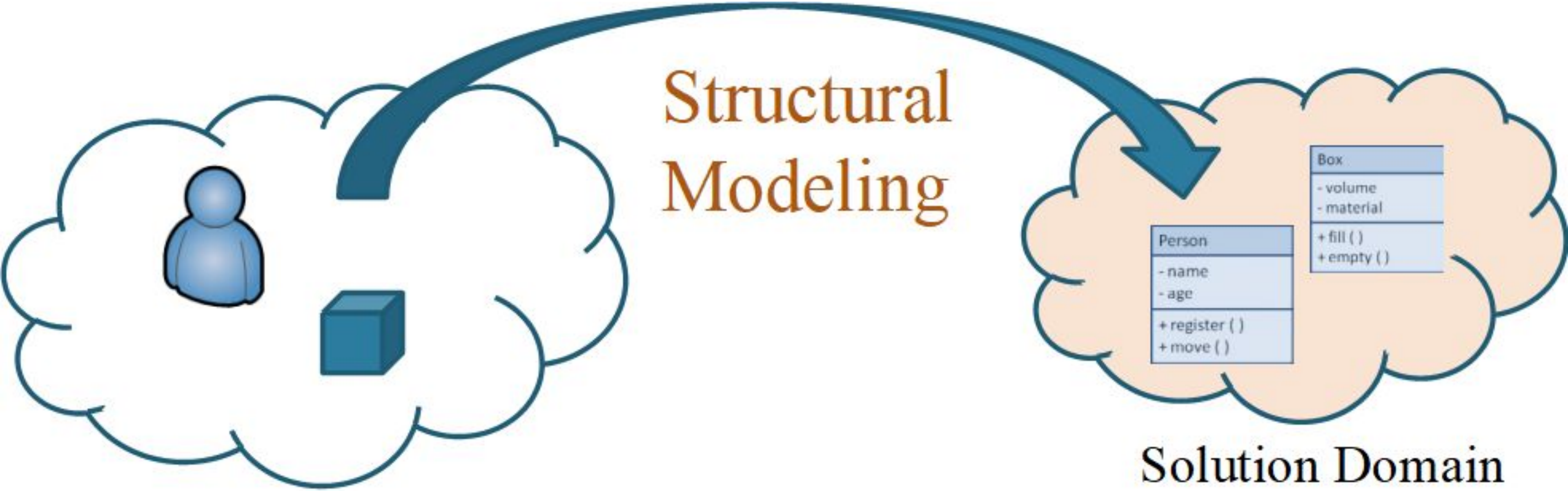
- Start conceptual, business-oriented
- **Refine** towards technology
    - Describe database, files, etc.

Create a **vocabulary** for users and analysts

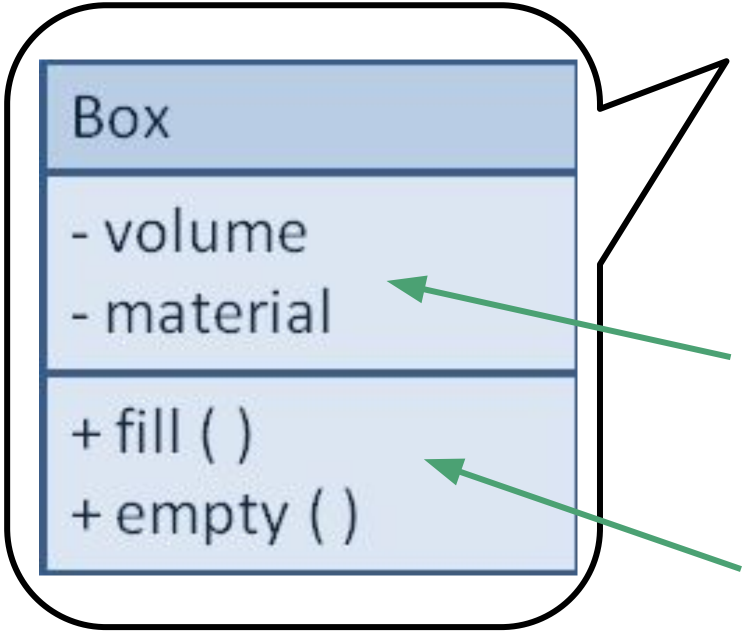
- Enables **effective communication**

Structural models

Discover key data in problem domain and **build structural model of objects**



Classes, attributes, operations



Classes

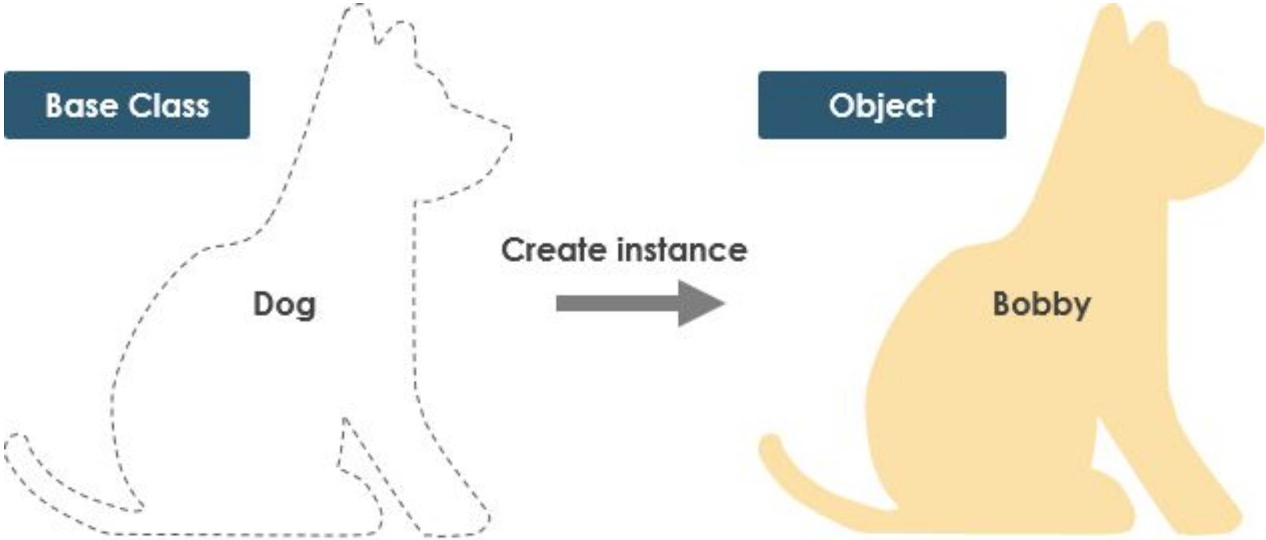
- Templates for instantiating people, objects, etc.
- Generally → SIMILAR objects!

Attributes

- Properties that **describe** state of class instance

Operations

- Actions/functions/methods that a class can perform



Base Class

Dog

Create instance

Object

Bobby

Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

Relationships

How classes relate to each other!

- Important for capturing dependencies, feature interactions, etc.

Generalization

- Enables inheritance of attributes and operations
- Represents relationships that are “a-kind-of”

Aggregation

- Relates parts to wholes or assemblies
- Represents relationships that are “a-part-of” or “has-parts”

Association

- Miscellaneous relationships between classes
- Usually a weaker form of aggregation

Object identification

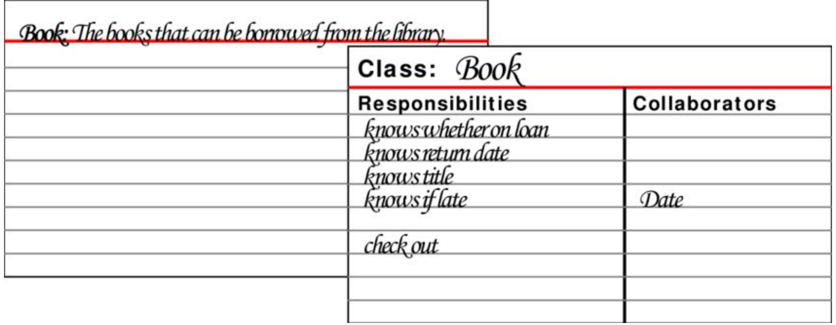
Analyze use cases

- Nouns ➜ suggest classes
- Verbs ➜ suggest methods
- (Rough outline for first iteration)

Brainstorming!

- Initial list of classes (objects)
- Attributes, operations, relationships can be added in future iterations

CRC cards



Book: The books that can be borrowed from the library.

Class: Book

Responsibilities	Collaborators
knows whether on loan	
knows return date	
knows title	
knows if late	Date
check out	

Index cards (physical or virtual)

- Document responsibilities / collaborations for a class

Responsibility

- Knowing ➜ what a class must **know** defined as attributes
- Doing ➜ what a class must **do** defined as operations (manifested later)

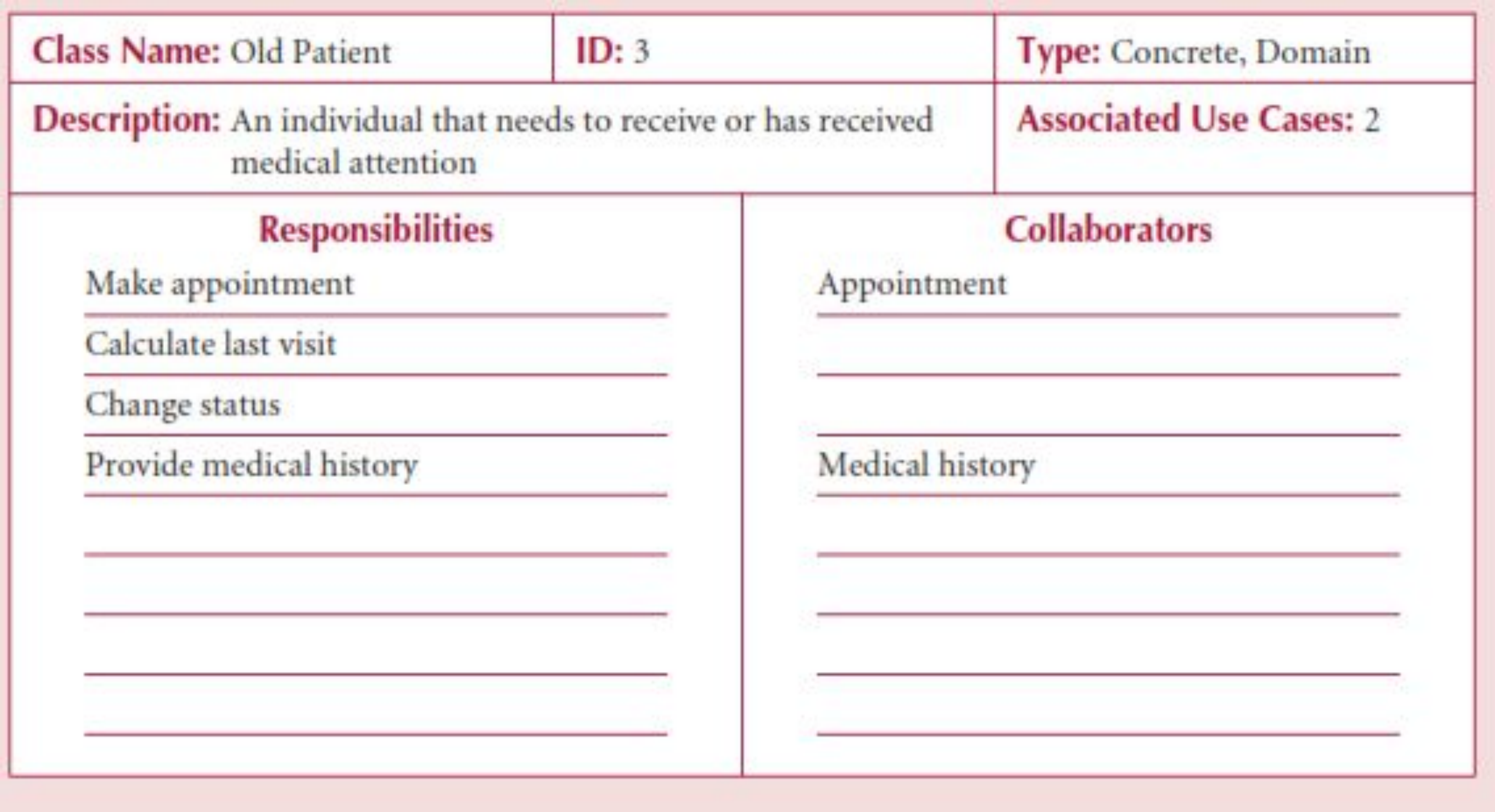
Collaboration

- Objects working **together** to service a request
  - Requestor (client)
  - Responder (server)

- Bound by a **contract**

(CRC → class responsibility collaboration card) https://www.youtube.com/watch?v=Bxgn6qJ-bYY

Front side (CRC)



Class Name: Old Patient

ID: 3

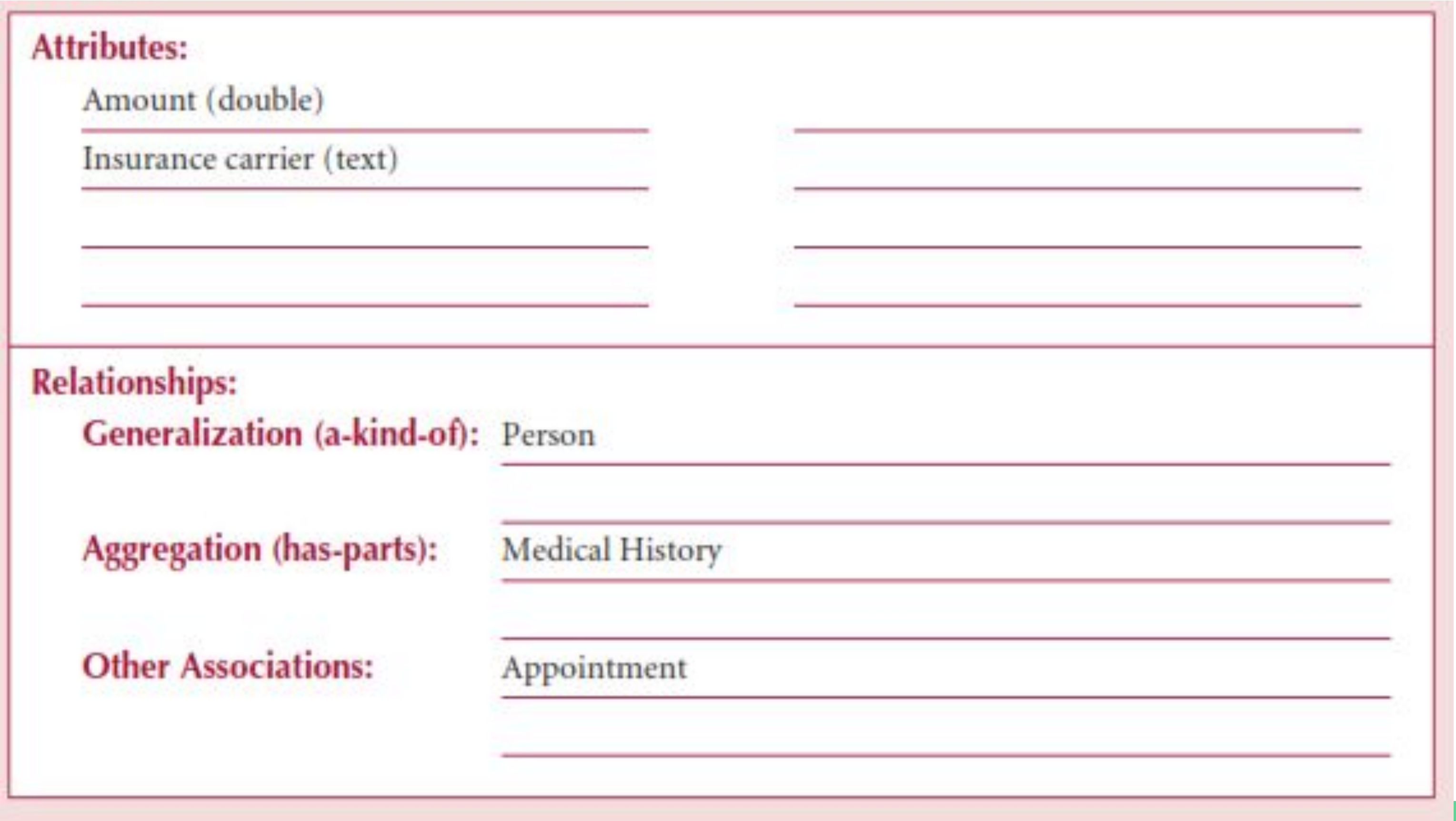
Type: Concrete, Domain

Description: An individual that needs to receive or has received medical attention

Associated Use Cases: 2

Responsibilities	Collaborators
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

Back side (CRC)



Attributes:

Amount (double)

Insurance carrier (text)

Relationships:

Generalization (a-kind-of): Person

Aggregation (has-parts): Medical History

Other Associations: Appointment

CRC cards / role playing

Exercise to help discover additional objects, attributes, relationships, operations

Team members perform roles associated with the actors and objects previously identified

CRC cards / role playing

Utilize activity diagrams to run through the steps in a scenario

- Identify an important use-case
- Assign roles based on actors and objects
- Team members perform each step in the scenario
- Discover and fix problems until a successful conclusion is reached
- Repeat for remaining use-cases

Why use CRC cards at all?

?

Group work

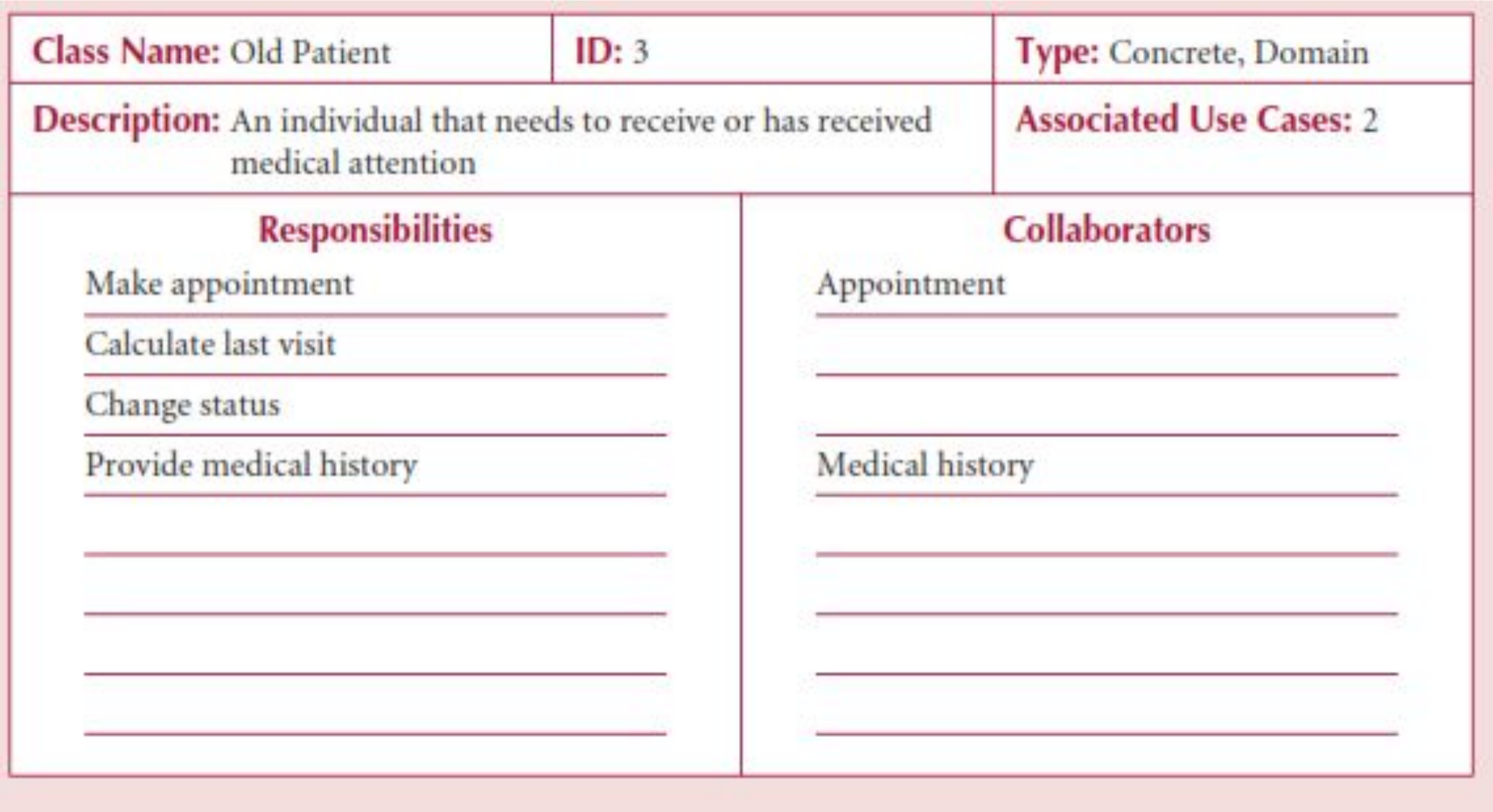
Pick **one use case diagram** and create **one** CRC for one of your use cases

(Just the front)

Be prepared to present it for

constructive criticism

(You'll turn it in tomorrow night with a rough class diagram to support it, made later on)



Class Name: Old Patient

ID: 3

Type: Concrete, Domain

Description: An individual that needs to receive or has received medical attention

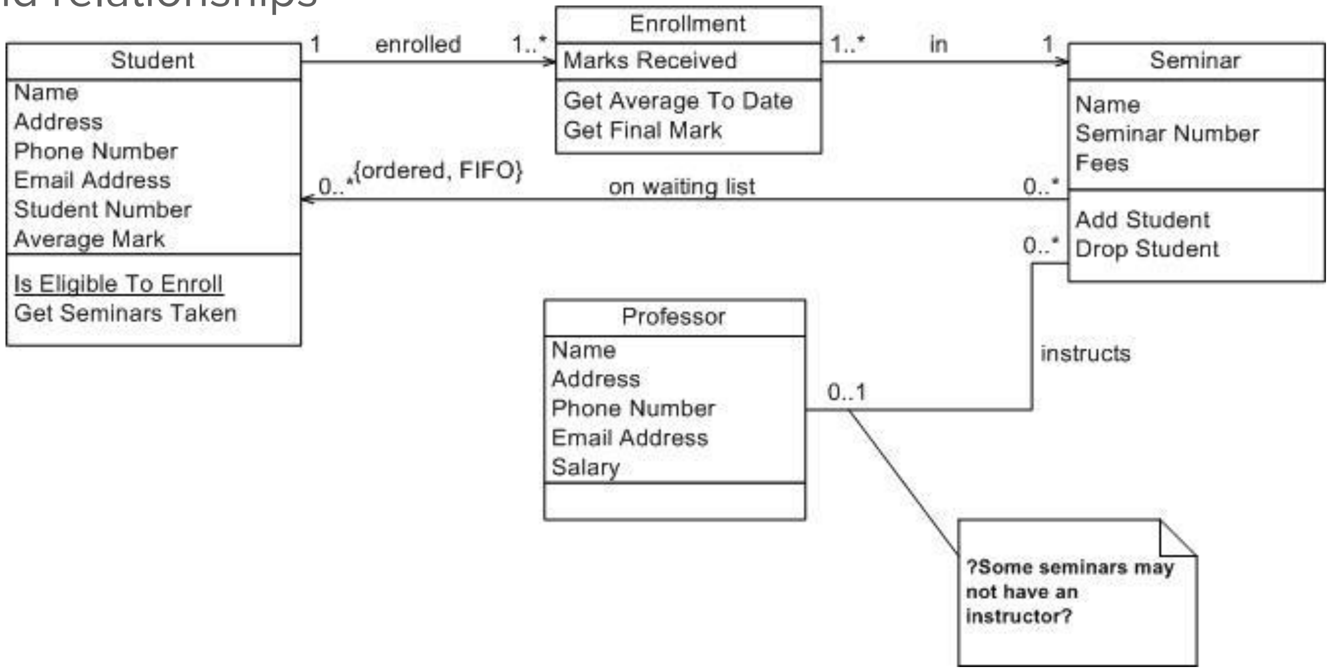
Associated Use Cases: 2

Responsibilities	Collaborators
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

https://echeung.me/crcmaker/

Class diagrams

Illustrates classes and relationships



Helpful ➔ https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Class diagrams

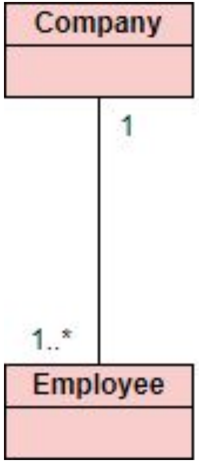
(Generally) static model

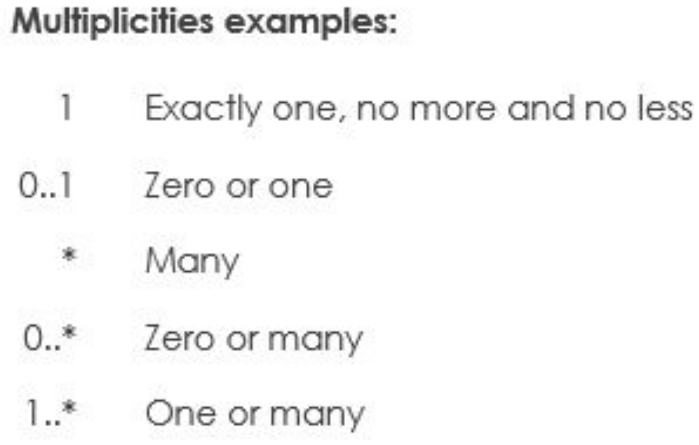
Classes

- Objects within system
- Stores/manages:
    - Attributes
    - Methods

Relationships

- Associations between classes
- "Lines" between classes
- Multiplicity ➜ *n* objects associated with *m* other objects





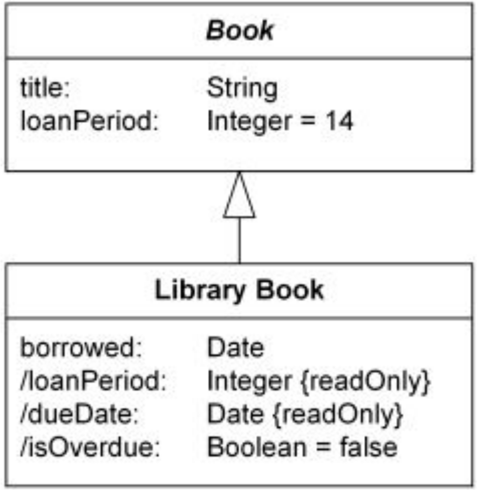
Multiplicities examples:

1	Exactly one, no more and no less
0..1	Zero or one
*	Many
0..*	Zero or many
1..*	One or many

Class diagrams ➜ Attributes

Properties:

- E.g., **Person**: last name, first name, address, etc.
- Can be derived:
    - Preceded with slash (/)
    - E.g., age derived from birth


Book
title: String
loanPeriod: Integer = 14
Library Book
borrowed: Date
/loanPeriod: Integer {readOnly}
/dueDate: Date {readOnly}
/isOverdue: Boolean = false

Visibility:

- Restricts access for consistency
- Public (+) ➜ visible to **all** classes
- Private (-) ➜ visible **only** to instance of defined class
- Protected (#) ➜ visible **only** to instance of defined class **and** its descendents

Class diagrams ➜ Operations

Common operations **not shown**

- Create/delete instance
- Get/set value

Types of operations

- Constructor ➜ creates an object
- Query ➜ retrieves object information
- Update ➜ changes some value
- Destructor ➜ destroys (delete/remove) an object

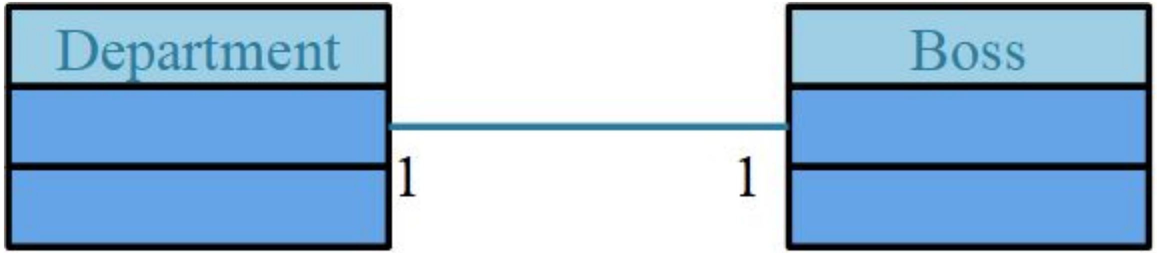
Class diagrams ➜ Relationships

Associations between classes

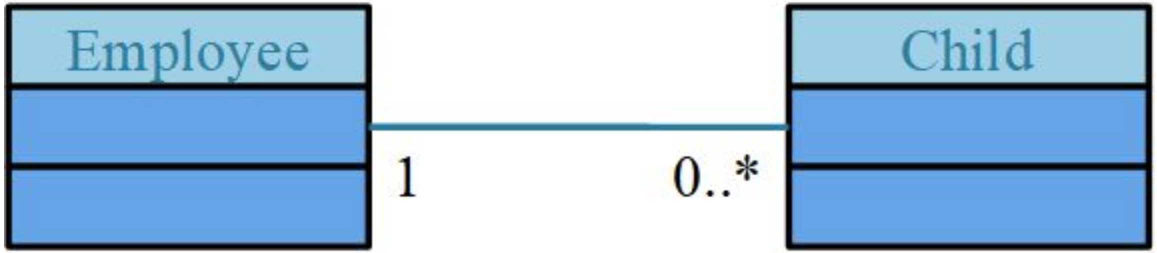
- Line labeled with relationship name
- *May be* directional (triangle)
    - Patient schedules an appointment

Classes can be self-related (e.g., employees and managers are both people)

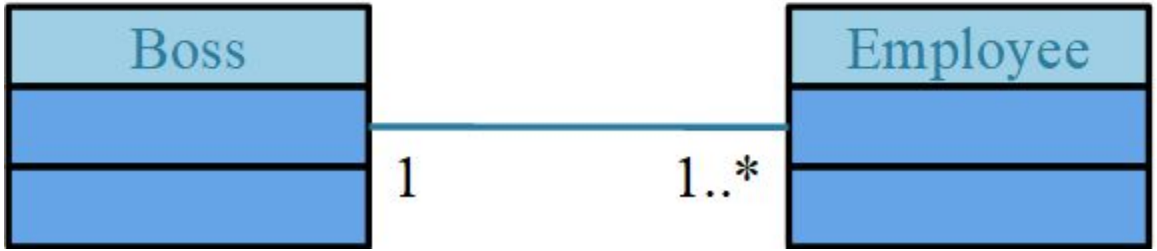
Multiplicity: how many of each class are related to each other



Exactly one:
A department has one and only one boss



Zero or more:
An employee has zero to many children



One or more:
A boss is responsible for one or more employees

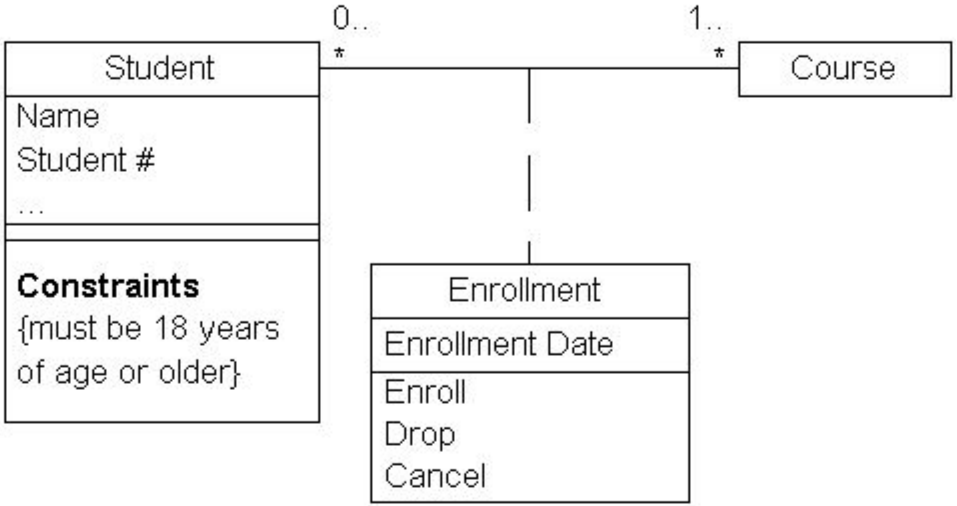
Association classes

Common with many-to-many relationships

Used when attributes about relationship needs recording

- Students related to courses
    - Grade class provides attribute to describe relationship
- Illness related to symptoms
    - Treatment class provides attribute to describe as well

Sample association



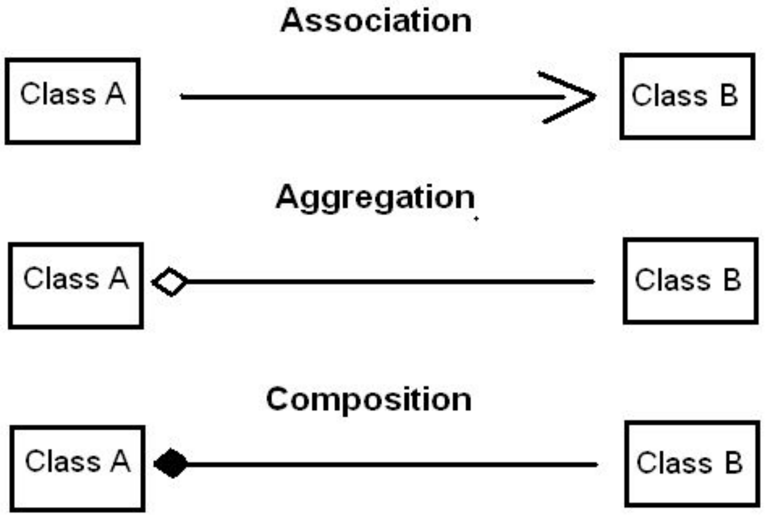
Generalization/Aggregation associations

Generalization denotes *inheritance*

- Properties/operations of *superclass* also valid for subclass
- Solid line with hollow arrow pointing at superclass

Aggregation: **logical** "part-of" relationship

Composition: **physical** "part-of" relationship



Generalization (*is a kind of):*

- Cardinal *is a kind of* Bird
- Truck *is a kind of* Land Vehicle (which is *a kind of* Vehicle)

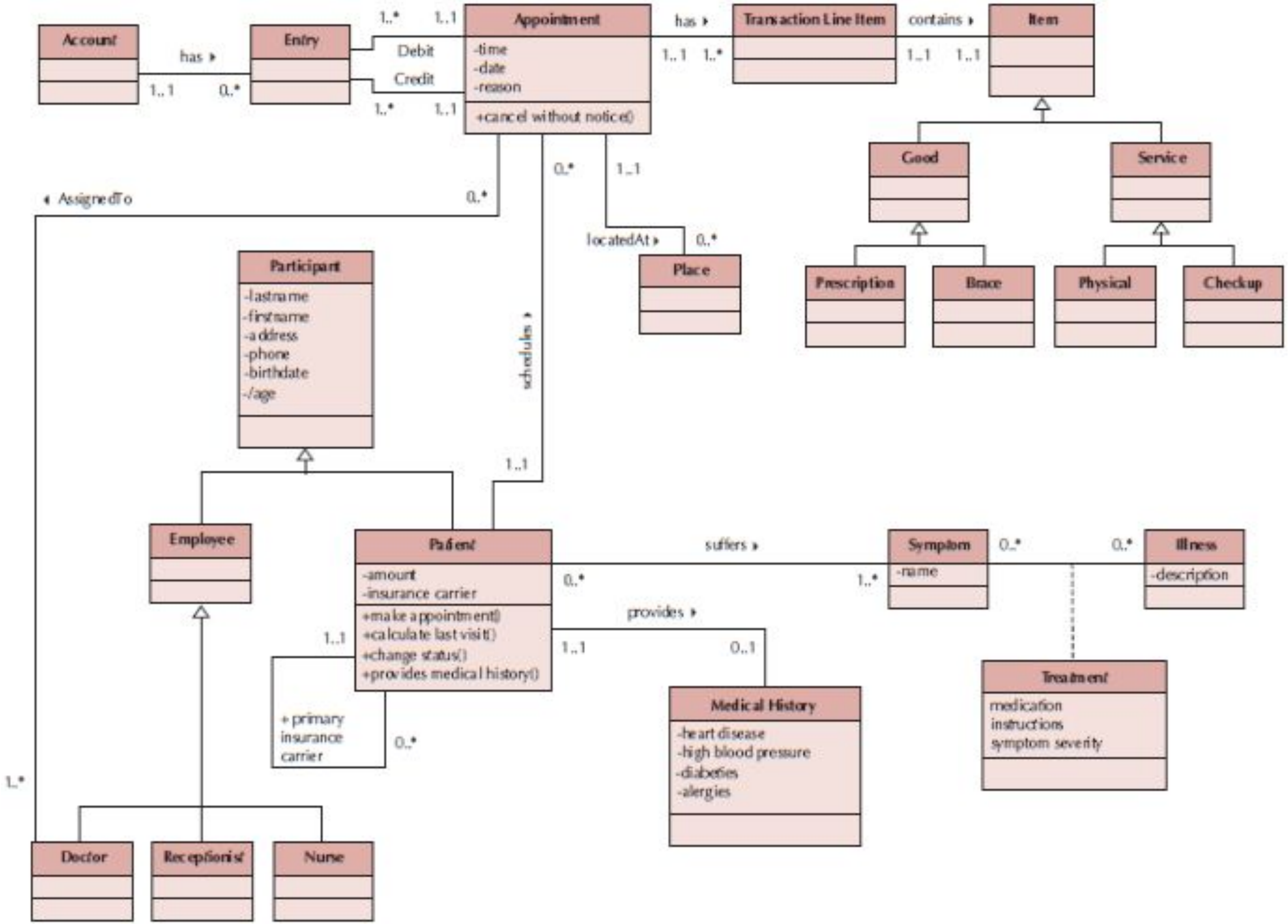
Aggregation (**logical** *is a part of*):

- Wheel instance *is a part of* a Vehicle instance

Composition (**physical** *is a part of*):

- Door instance *is a part of* only a single Vehicle instance

Sample CD

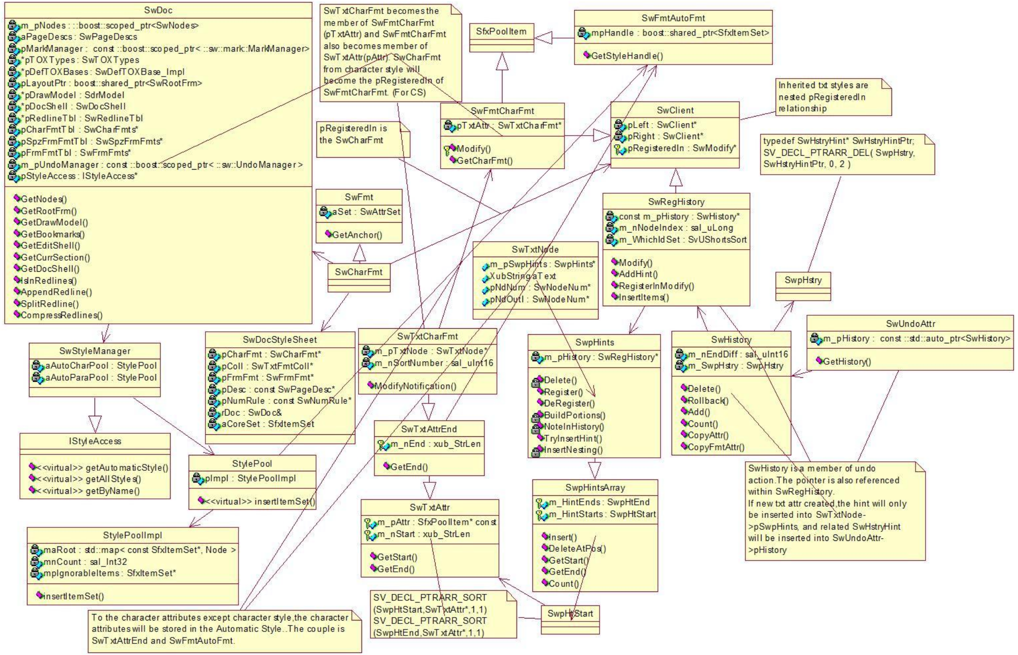

Account
has ▸
1..1
0..*
Entry
1..*
1..1
Debit
Credit
1..*
1..1
Appointment
-time
-date
-reason
+cancel without notice()
has ▸
1..1
1..*
Transaction Line Item
contains ▸
1..1
1..1
Item
Good
Service
Prescription
Brace
Physical
Checkup
◂ AssignedTo
0..*
0..*
1..1
locatedAt ▸
0..*
Place
schedules ▸
1..1
Participant
-lastname
-firstname
-address
-phone
-birthdate
-/age
Employee
Patient
-amount
-insurance carrier
+make appointment()
+calculate last visit()
+change status()
+provides medical history()
1..1
+ primary
insurance
carrier
0..*
suffers ▸
0..*
1..*
Symptom
-name
0..*
0..*
Illness
-description
provides ▸
1..1
0..1
Medical History
-heart disease
-high blood pressure
-diabeties
-alergies
Treatment
medication
instructions
symptom severity
1..*
Doctor
Receptionist
Nurse

Simplifying class diagrams!

Fully-populated class diagrams can be horrendously complex

Simplify:

- Show only concrete classes
- View mechanism shows subset of classes
- Packages show aggregations of classes (or any UML element)



SwTxtCharFmt becomes the member of SwFmtCharFmt (pTxtAttr) and SwFmtCharFmt also becomes member of SwTxtAttr(pAttr). SwCharFmt from character style will become the pRegisteredIn of SwFmtCharFmt. (For CS)

pRegisteredIn is the SwCharFmt

Inherited txt styles are nested pRegisteredIn relationship

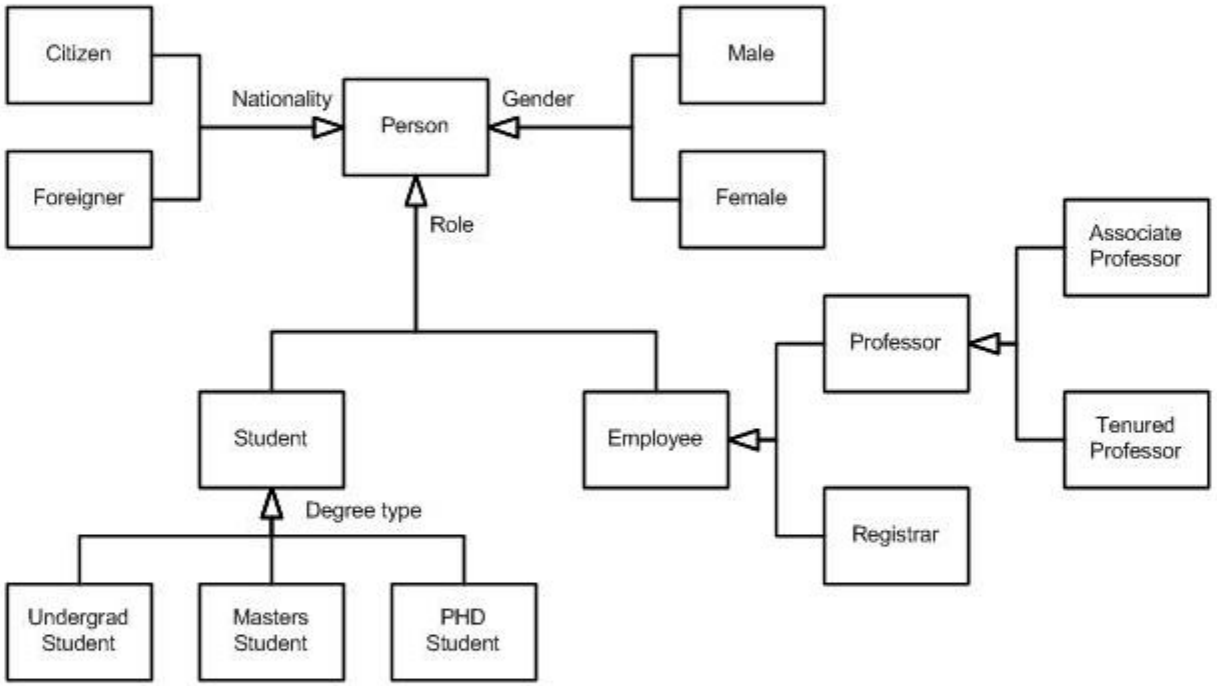
typedef SwHstryHint* SwHstryHintPtr;
SV_DECL_PTRARR_DEL(SwpHstry, SwHstryHintPtr, 0, 2)

SwHistory is a member of undo action.The pointer is also referenced within SwRegHistory.
If new txt attr created,the hint will only be inserted into SwTxtNode->pSwpHints, and related SwHstryHint will be inserted into SwUndoAttr->pHistory

To the character attributes except character style,the character attributes will be stored in the Automatic Style..The couple is SwTxtAttrEnd and SwFmtAutoFmt.

SV_DECL_PTRARR_SORT
(SwpHtStart,SwTxtAttr*,1,1)
SV_DECL_PTRARR_SORT
(SwpHtEnd,SwTxtAttr*,1,1)

Disqus



http://www.agilemodeling.com/artifacts/classDiagram.htm

Let's create an example *together*

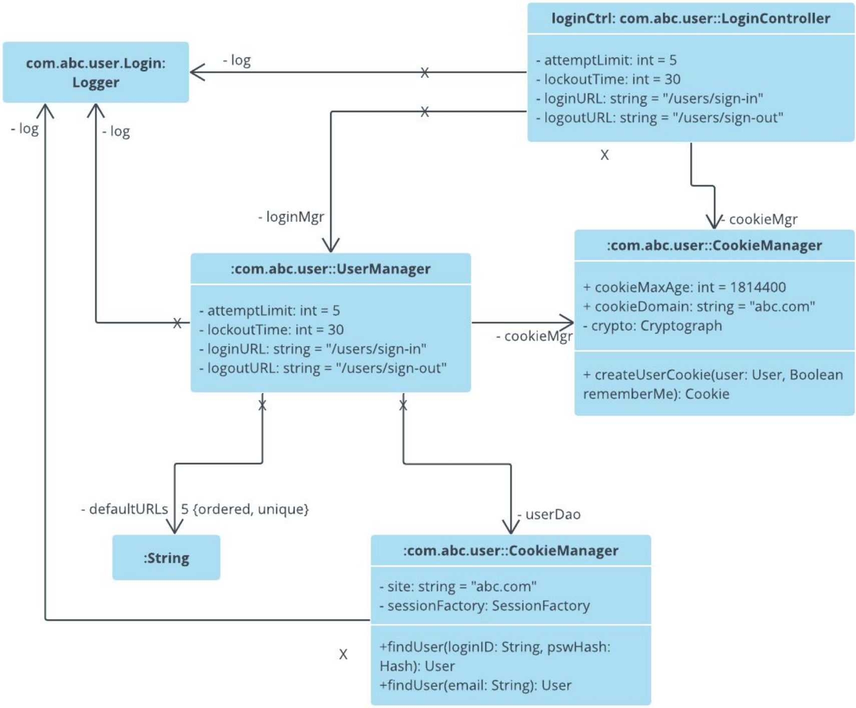
Object diagrams

Class diagrams with **instantiated** classes

- Doctor class ➔ Dr. Fredericks (ha)
- Attributes have **values**

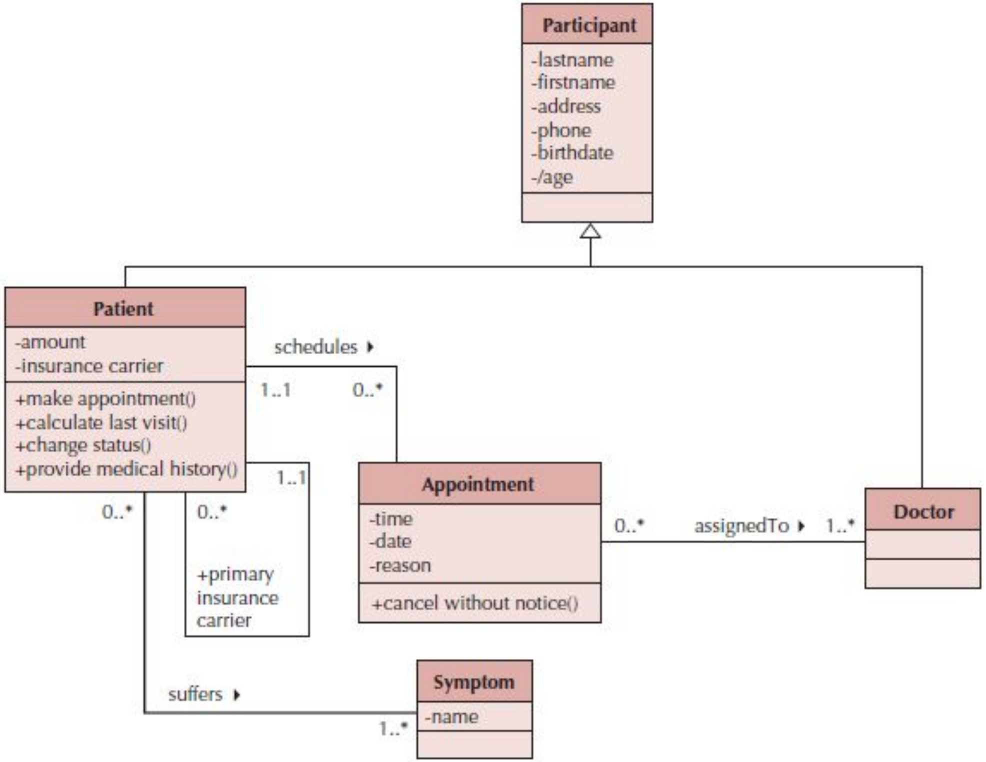
Discover additional attributes/relationships/etc.

- Also find out what has been misplaced or misconfigured!



loginCtrl: com.abc.user::LoginController
- attemptLimit: int = 5
- lockoutTime: int = 30
- loginURL: string = "/users/sign-in"
- logoutURL: string = "/users/sign-out"
com.abc.user.Login: Logger
- log
- log
- log
x
x
x
- loginMgr
- cookieMgr
:com.abc.user::UserManager
- attemptLimit: int = 5
- lockoutTime: int = 30
- loginURL: string = "/users/sign-in"
- logoutURL: string = "/users/sign-out"
x
- cookieMgr
:com.abc.user::CookieManager
+ cookieMaxAge: int = 1814400
+ cookieDomain: string = "abc.com"
- crypto: Cryptograph
+ createUserCookie(user: User, Boolean rememberMe): Cookie
x
x
- defaultURLs
5 {ordered, unique}
- userDao
:String
:com.abc.user::CookieManager
- site: string = "abc.com"
- sessionFactory: SessionFactory
+findUser(loginID: String, pswHash: Hash): User
+findUser(email: String): User
x

Example



Participant

-lastname
-firstname
-address
-phone
-birthdate
-/age

Patient

-amount
-insurance carrier

+make appointment()
+calculate last visit()
+change status()
+provide medical history()

schedules ▸

1..1

0..*

Appointment

-time
-date
-reason

+cancel without notice()

1..1

0..*

0..*

+primary insurance carrier

0..*

assignedTo ▸

1..*

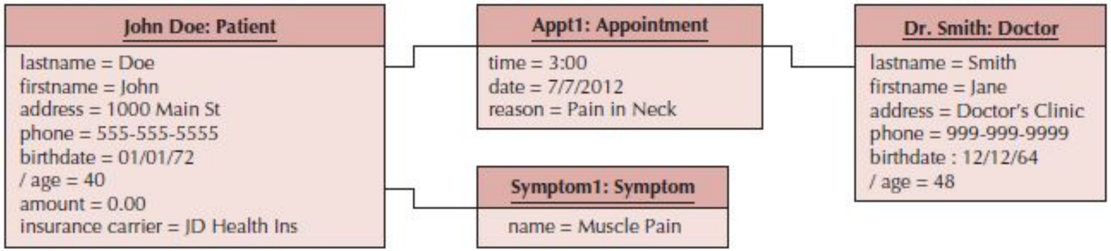
Doctor

suffers ▸

1..*

Symptom

-name



John Doe: Patient

lastname = Doe
firstname = John
address = 1000 Main St
phone = 555-555-5555
birthdate = 01/01/72
/ age = 40
amount = 0.00
insurance carrier = JD Health Ins

Appt1: Appointment

time = 3:00
date = 7/7/2012
reason = Pain in Neck

Symptom1: Symptom

name = Muscle Pain

Dr. Smith: Doctor

lastname = Smith
firstname = Jane
address = Doctor's Clinic
phone = 999-999-9999
birthdate : 12/12/64
/ age = 48

Let's create *an example together*

Seven steps to structural models

1. Create CRC Cards
2. Review CRC Cards and identify missing objects, attributes, operations and/or relationships
3. Role-play the CRC cards—look for breakdowns & correct; create new cards as necessary
4. Create the class diagram
5. Review the class diagram—remove unnecessary classes, attributes, operations and/or relationships
6. Incorporate patterns
7. Review and validate the model

More group work!

Take that CRC card you made earlier and:

1) Make a rough class diagram out of it

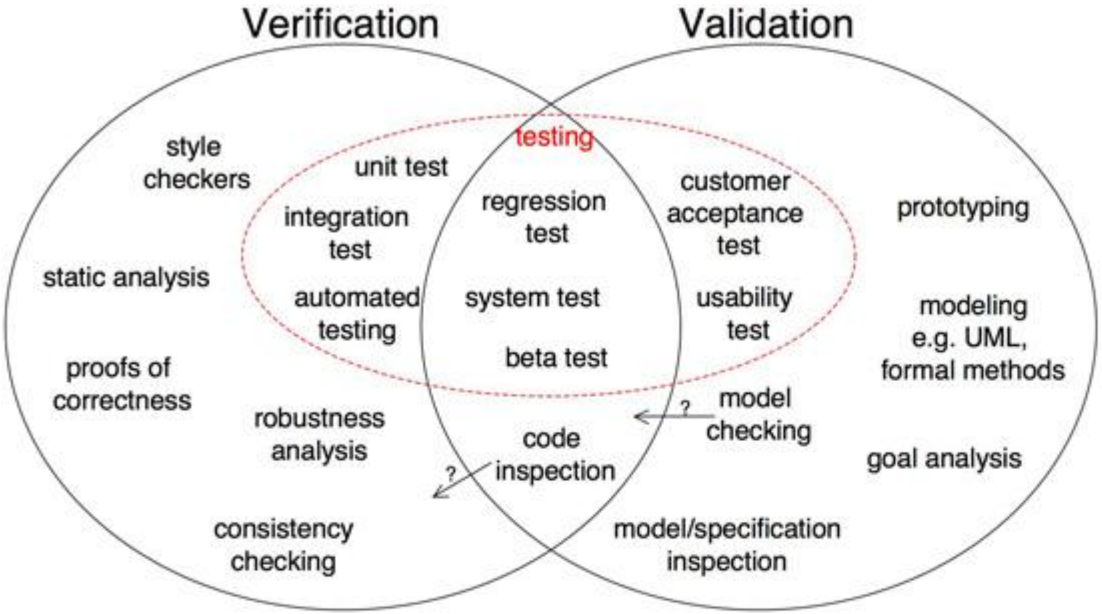
2) Create an object model to demonstrate it working

Turn in these materials by tomorrow night at midnight
➔ Keyword -- rough (these don't have to be perfect, we'll formalize some soon)

Verifying and validating the model

Analyst presents to developers and users

- Walks through the model
- Provides explanations and reasoning behind each class



Verifying and validating the model

Rules

1. Each CRC card is associated with a class
2. Responsibilities on the front of the card are included as operations on the class diagram
3. Collaborators on the front of the card imply a relationship on the back of the card
4. Attributes on the back of the card are listed as attributes on the class diagram
5. Attributes on the back of the CRC card each have a data type (e.g., salary implies a number format)
6. Relationships on the back of the card must be properly depicted on the class diagram
   a. Aggregation/Association
   b. Multiplicity
7. Association classes are used only to include attributes that describe a relationship

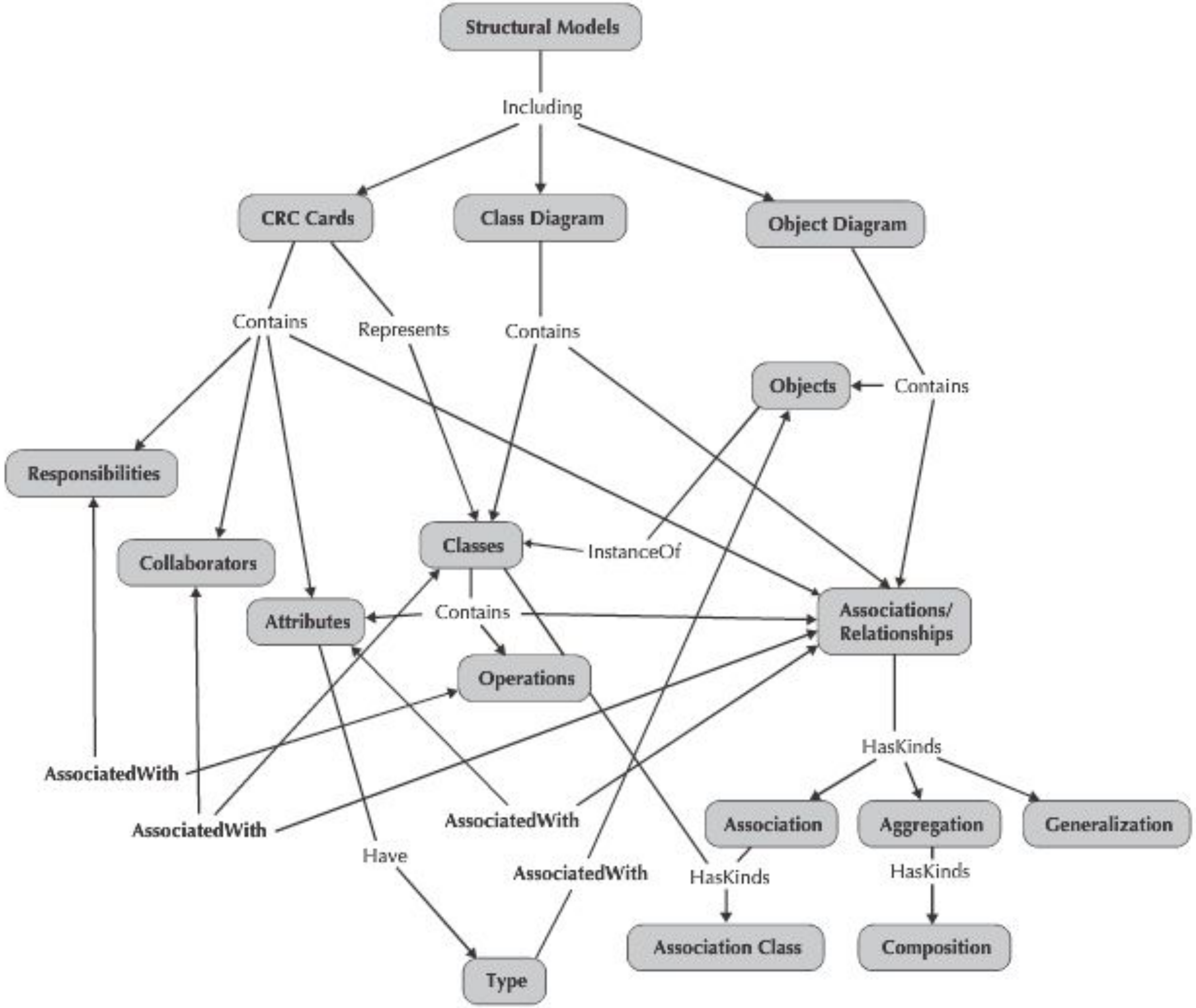


FIGURE 5-26 Interrelationships among Structural Models

Interlude :: Gantt Charts, Burn-Down/Burn-Up

Seems we forgot to go into these, especially important as *you'll be selecting one and doing it for your midterm presentations!*

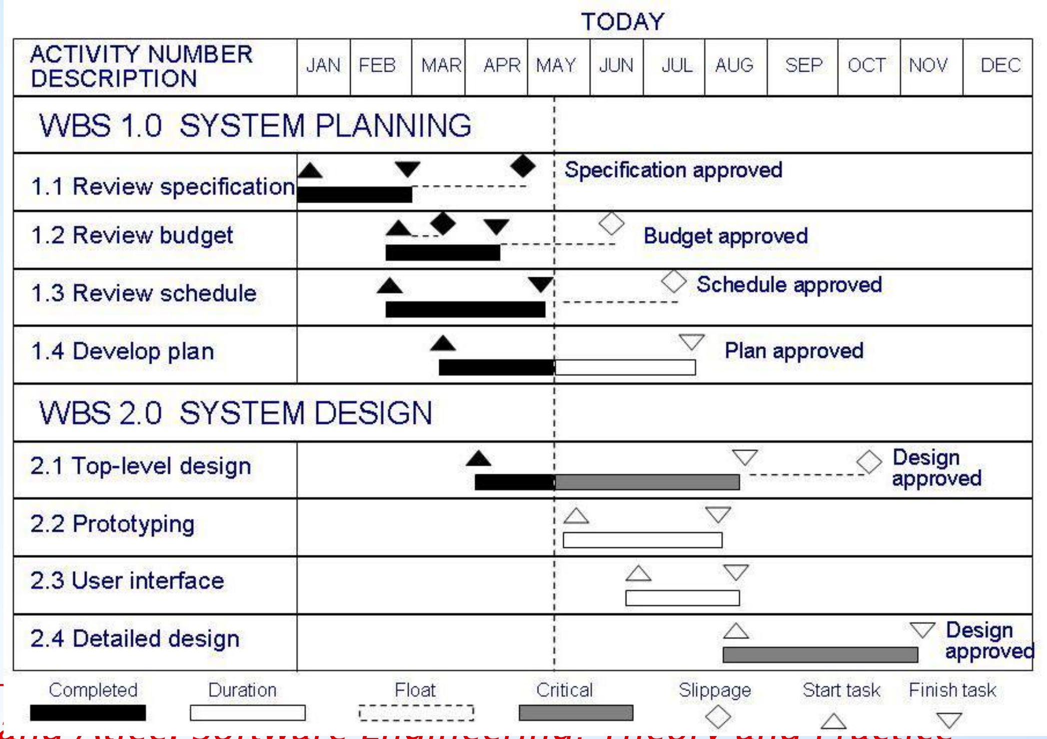
Which will be October TBDth!

There are details in Blackboard under 'Term Project'

3.1 Tracking Progress

Tools to Track Progress: Gantt Chart

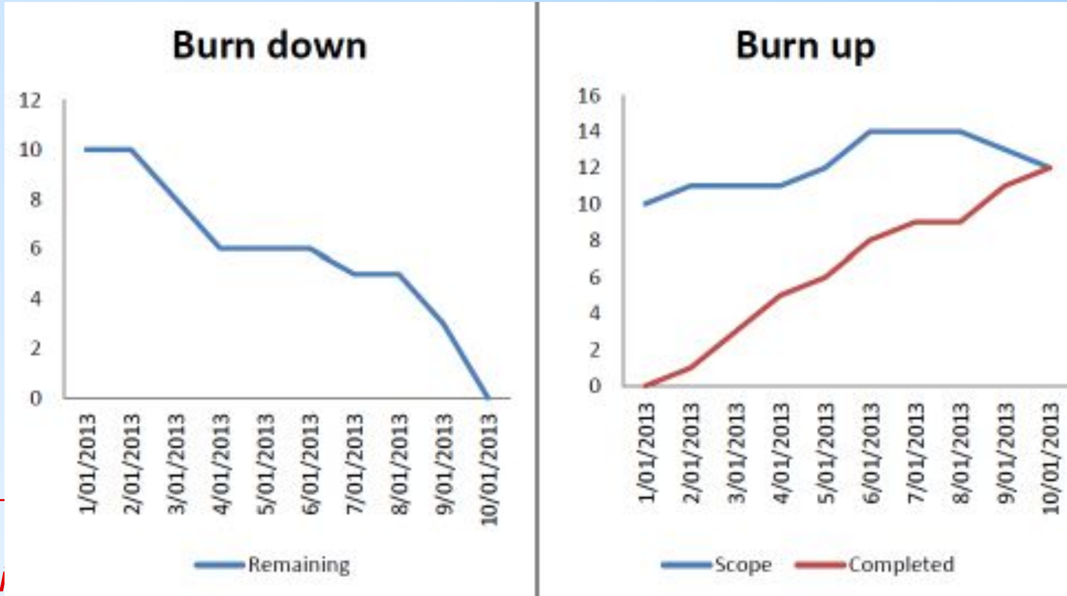
- Activities shown in parallel (shows task concurrency)



Wot about Agile?

Gantt charts are so *passé*

Let's burn up and burn down


Burn down
1/01/2013
2/01/2013
3/01/2013
4/01/2013
5/01/2013
6/01/2013
7/01/2013
8/01/2013
9/01/2013
10/01/2013
Remaining
Burn up
1/01/2013
2/01/2013
3/01/2013
4/01/2013
5/01/2013
6/01/2013
7/01/2013
8/01/2013
9/01/2013
10/01/2013
Scope
Completed