# Classes: Concepts and Identification

**Daniel M. Berry**

# Introduction -1

**All your acquaintance with classes prior to this course has been as a device to implement information hiding and object orientation.**

**You have looked at them as documentation of code written in C++, Java, and possibly other languages.**

# Introduction -2

**You may have even looked at class diagrams as an expression of the architecture of a program either to be built or that has been built out of classes.**

**You may have looked at class diagrams as a notation in which to play with the architecture of a program in order to arrive at the best architecture.**

# Introduction -3

**It is better to work with the diagrams than raw code to arrive at the best architecture for two reasons:**

1. **Code is way too detailed to see the architecture in it; class diagrams are at a higher level of abstraction than is the code and are at the right level of abstraction in which to consider the architecture.**

# Introduction -4

2. **One can decide on possible architectures long before even considering code; in this case, a notation is needed**
   - **with which to express the architecture and**
   - **that is systematic enough that the meaning of the diagram and its implication on the final code are clear**

# Introduction -5

**Indeed, people who play with patterns to help find the best architecture use class diagrams as the medium in which to describe both the patterns and the architectures that are instances of these patterns.**

# Introduction -6

**However, the question remains, “How are class diagrams, and, indeed how are all of UML, used in requirements engineering to help arrive at a specification of requirements?”**

**This lecture tries to answer this question by considering some examples of deriving a class diagram from a problem description.**

# Introduction -7

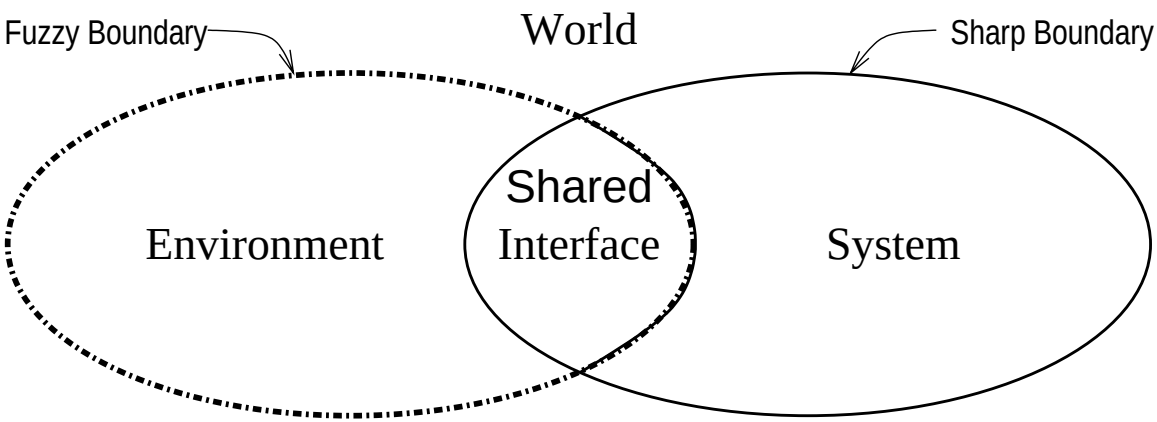
**Later, we will look at deriving scenarios and use cases from these same problem descriptions.**

**Still later, we will look at specifying user interfaces for some of these problem descriptions.**

# Example 2: Turnstile

**The city of Waterloo has decided to raise funds by instituting users fees for public parks. We need to implement a complete system of money collection, security, etc.**

# Dividing the World




**The Environment is the part of the World that is affected by the System.**

# Turnstile Requirements

***Informal requirements*:Collect $1 fee from each human park user on entry to park (no fee to leave).**

- **Ensure that no one may enter park without paying.**
- **Ensure that anyone who has paid may enter park.**

# Possible Solutions

***Solution #1*: Employ human fee collectors. Enforce security by instituting the Waterloo Park Militia, armed guards who make certain no one uses a park without paying a user fee.**

***Solution #2*: Use chain link fences for security, use turnstiles with automated coin collection. After some research, we find appropriate turnstile hardware, but it's brand new technology so we must create the embedded software system....**

# The Park World -1

**There is a barrier to enter a park.  A person inserts a coin, the turnstile unlocks, allowing the person to push the turnstile and enter the park.**

# The Park World -2a

| | |
|---|---|
| *environment* | **visitorS** |
| | **does insert of coinS to CoinSlot** |
| | **detects unlocking of Barrier** |
| | **does push of Barrier** |
| | **coinS** |
| | **fence** |
| | **p~~ersonS~~** |

# The Park World -2b

| | | |
|---|---|---|
| ***shared phenomena*** | **coinSlot** | **receives insert(denom)** |
| | | **receives inserted?()** |
| | | **does addCoin(denom) to TurnstileSystem** |
| | **Barrier** | **receives push()** |
| | | **receives unlock()** |
| | | **receives inRotation?()** |
| | | **receives lock() or locked?()** |
| | | **does addVisitor() to TurnstileSystem** |

# The Park World -2c

| | |
|---|---|
| *software system* | **TurnstileSystem** |

**does inserted?() to CoinSlot**
**does inRotation?() to Barrier**
**does unlock() to Barrier**
**does lock() or locked?() to Barrier**
**receives addCoin(denom)**
**receives addVisitor()**

# The Park World -3

**The software system and the environment interact via the shared phenomena, which may be both sensed and controlled by both the software system and the environment:**

# The Park World -4

- **The environment controls insertions of coins into coin slots.**
- **The software system senses coin insertion and then reacts by unlocking the turnstile.**

# The Park World -5

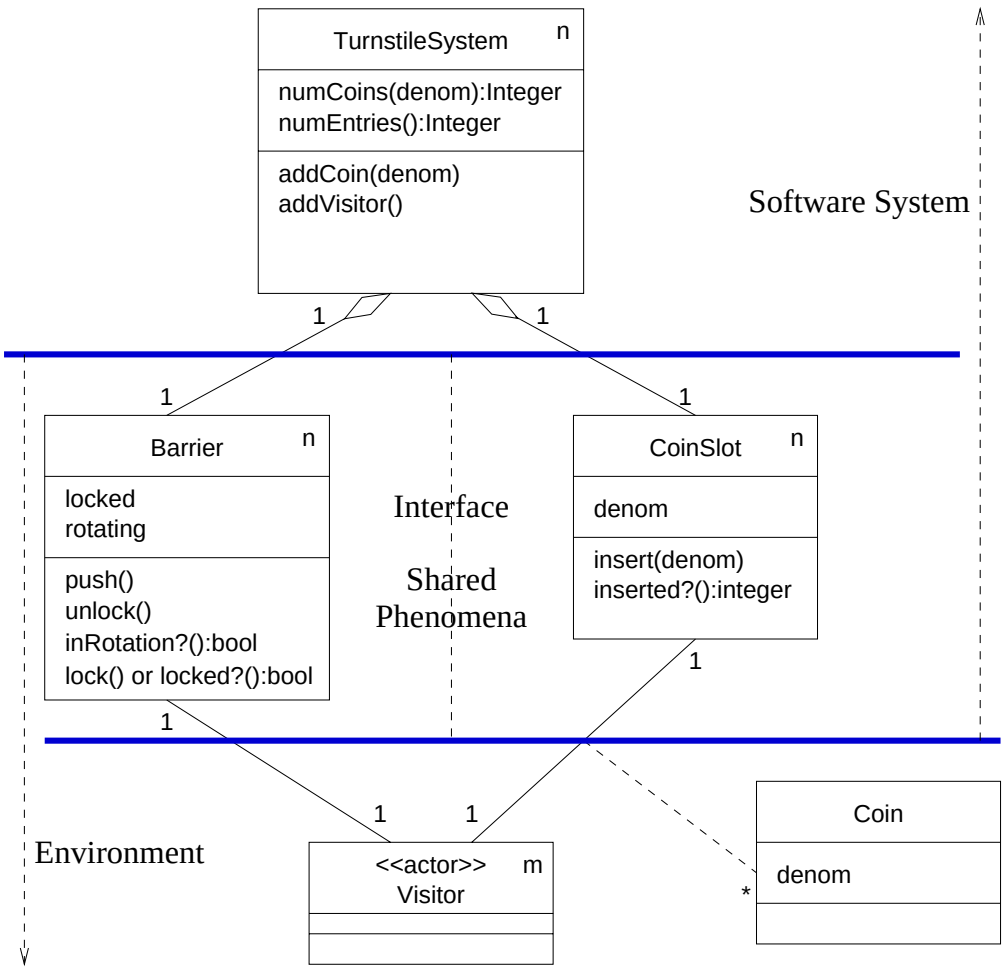
- **The environment senses that the turnstile is unlocked and a person can rotate the turnstile (to enter the park).**
- **The software system senses turnstile rotation and eventually either locks the turnstile, senses that the turnstile has locked, or assumes that the turnstile locks itself after rotation.**

# The Park World -6

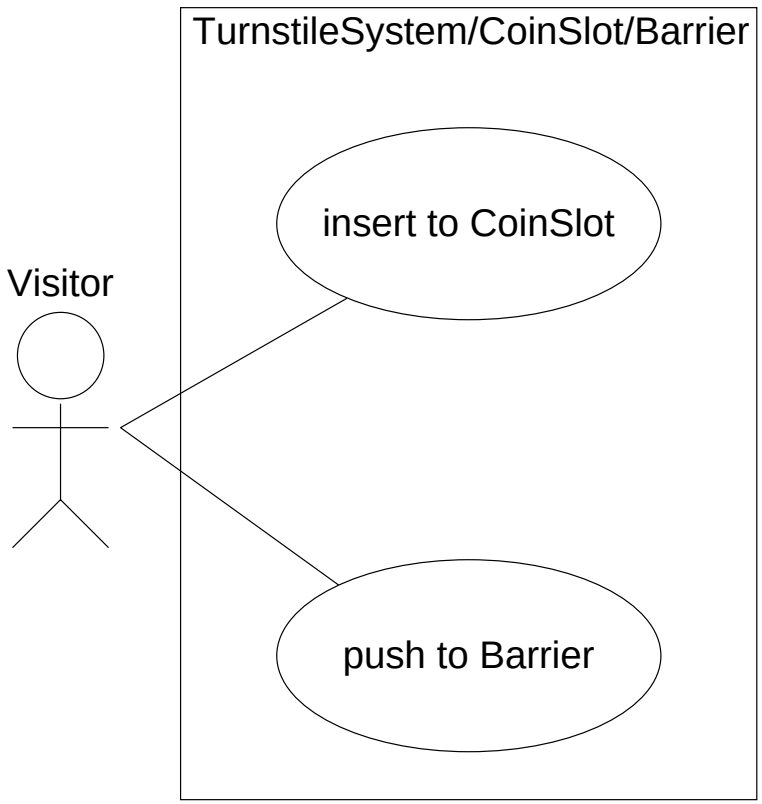
**What locks and unlocks and what is pushed and is rotated is the barrier, and this barrier together with the coin slot form the turnstile. The barrier and the coin slot together are the phenomena shared between the turnstile software system and the visitor in the environment.**

**Accordingly, we may construct a class diagram, enhanced from what you have seen in the tutorial on *Rational Rose*.**

# Turnstile Class Diagram



# Turnstile Use Cases



# Real World Example

**Now we consider how to build a necessarily incomplete model from the poor information you get from clients.**

# Example 3: Voting

**We now look at a system to automate voting that was motivated by what happened in the U.S. presidential elections in November 2000.**

# Voting Problems -1

**We have all seen the problems that Florida and the U.S. had in the 2000 presidential elections to get an accurate count of the votes. These counting problems are caused at least partially by the use of ancient punched-card based voting machines. If a voter does not punch out a chad (the piece of the card that makes a hole) completely, his or her vote is not counted by the counting machines that read the card ballots, looking for holes signifying votes.**

# Voting Problems -2

**Yet we all know how easy it is to vote at these WWW sites that conduct informal polls. These sites have a much better potential of accurately counting a vote that has been cast. Perhaps we should install an electronic voting system, allowing people to vote via the internet at election web sites.**

# Voting Problems -3

**Of course, there is now a whole new set of security problems caused by the poor security of the internet. Can you imagine if hackers managed to break into the election web site and changed the votes.**

**If we decide that security is a problem, we can forget about using the internet and provide each polling place with an intranet and workstations to replace the ancient voting machines.**

# Voting Problems -4

**On the other hand, if the security can be assured, voting by the internet would eliminate the need for absentee ballots, provided that internet access were universal enough, say as much as a telephone or television. We are not there yet, but are getting there.**

# Voting Problems -5

**Recall that an absentee ballot is a ballot filled out and snail mailed in by a voter who cannot be present in the polling place during the time the polls are open, e.g., because of illness, a religious holiday, or being out of town.**

# Voting Problems -6

**Below is described the requirements for one particular electronic voting system, namely *Sensus* by Lorrie F. Cranor.**

**Before we get into it, please understand the role of the requirements engineer (RE).**

# Role of the RE -1

**He or she is often called in to work with problems that are totally new to him or her. The problem description uses vocabulary unfamiliar to him or her.**

**It is the job of the requirements engineer to begin to form a model of the described problem so that he or she can use the model to identify what he or she does not understand and to ask questions of the client.**

# Role of the RE -2

**Very often this initial model is formed in ignorance. The requirements engineer identifies the nouns, verbs, adjectives, and adverbs of the problem description and uses them as the names of classes, operations, attributes, and nonfunctional requirements in the model formed in ignorance.**

**The thingamajig snarkles the doodad.**

**The thingamajig snarkles the doodad.**

Nouns:

Verbs:

Questions:

Exceptions:

**The thingamajig snarkles the doodad.**

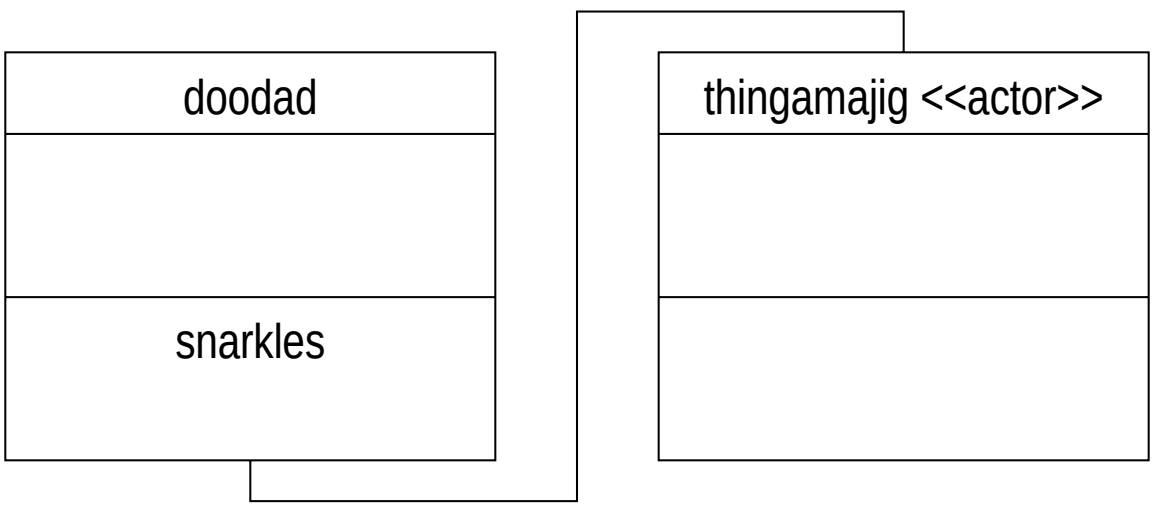
Nouns: thingamajig
doodad

Verbs: snarkle

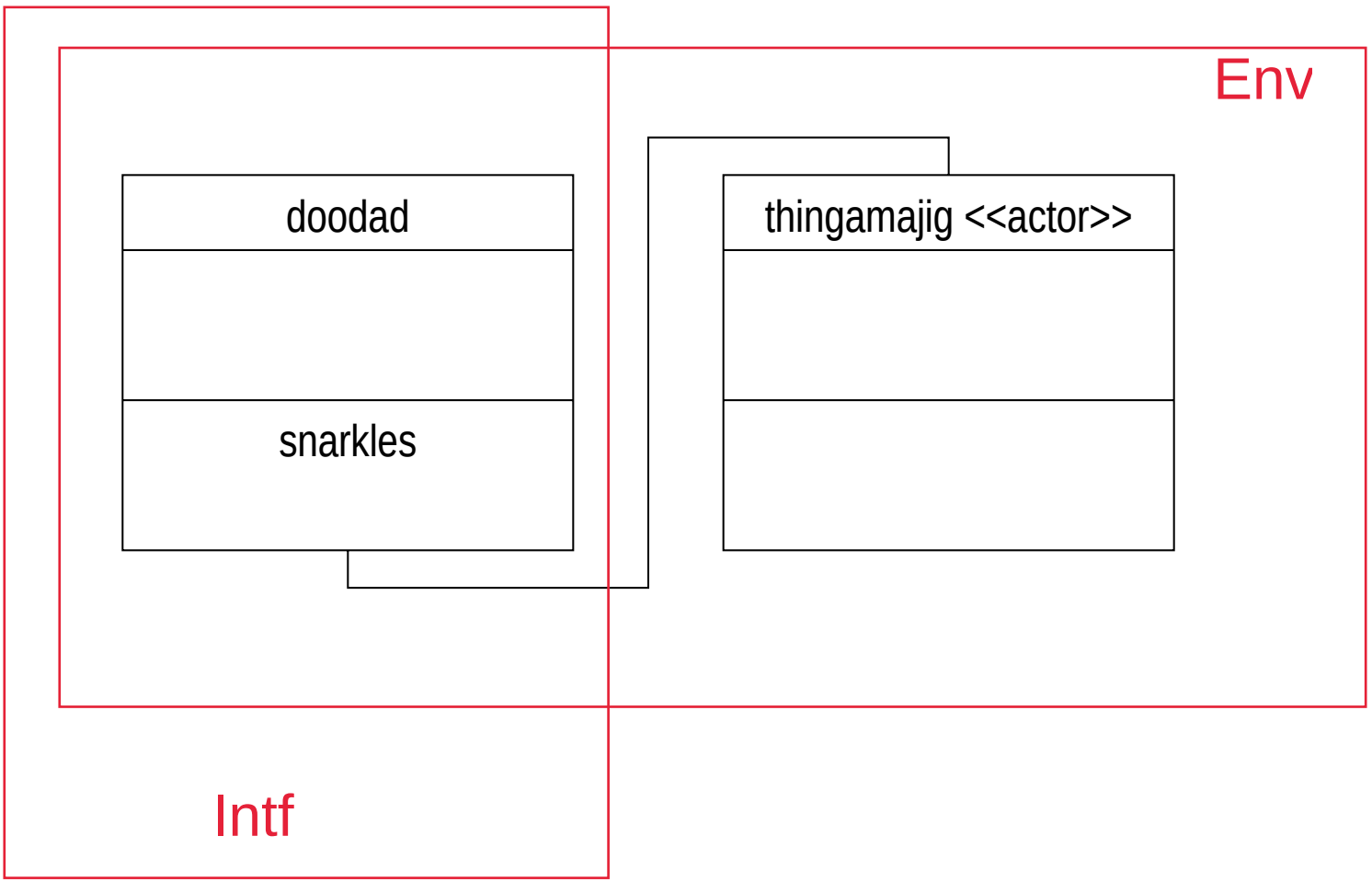
Questions: What is a thingamajig?
What is a doodad?
What is snarkling?

Exceptions: Can snarkling ever fail in any reason that we can check for?

# The thingamajig snarkles the doodad.

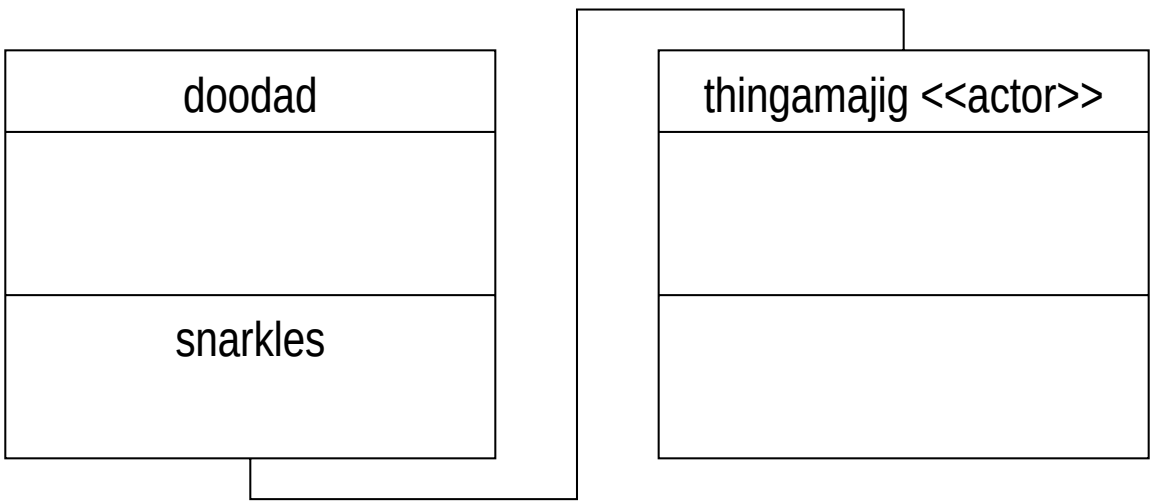


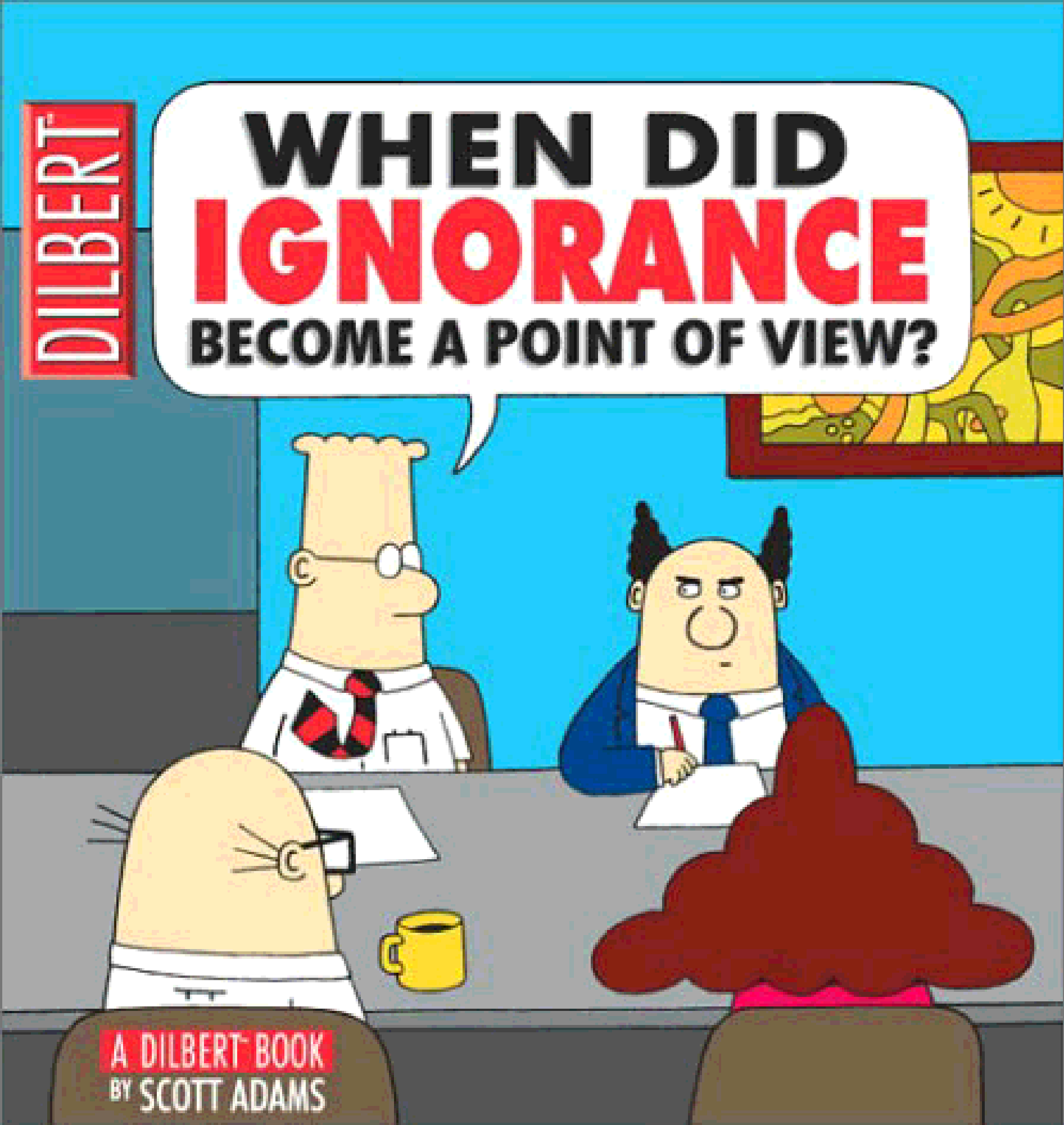
# The thingamajig snarkles the doodad.



# Encoding Ignorance

**The thingamajig snarkles the doodad.**




DILBERT
WHEN DID
IGNORANCE
BECOME A POINT OF VIEW?
A DILBERT BOOK
BY SCOTT ADAMS

# Role of the RE -3

**That is, the requirements engineer is skilled enough in modeling that he or she can take the words of the problem description and put them in the right places in the model, so as to end up with an intelligible model, even though he or she does not understand the words.**

# Role of the RE -4

**So we operate in this way on the problem description provided by the authors of *Sensus*.**

# Sensus Modules

***Sensus has four main modules:***

- ***Registrar — The registrar registers voters prior to an election.***
- ***Pollster — The pollster acts as a voters' [sic] agent, presenting human readable ballots to a voter, collecting the voter's responses to ballot questions, performing cryptographic functions on the voter's behalf, obtaining necessary validations and receipts, and delivering ballots to the ballot box....***

***The pollster is the only component of the Sensus system that voters must trust completely; voters concerned about the privacy of their ballots may want to install personal copies of the pollster on trusted machines.***

- ***Validator — The validator ensures that only registered voters can vote, and that only one ballot is counted for each registered voter.***
- ***Tallier — The tallier tallies the results of the election or survey.*** **[The word "tallier" should be read as "tally-er".]**

# Registering to Vote

***Before registering to vote, a voter must obtain a voter identification number, token, and registration address from the election administrators.***

***You may begin the registration process by running the pollster module. This is generally done by invoking the* sensus *command.***

***The pollster module will display a menu of options. Select the "register to vote" option.***

***The pollster will generate a public/private key pair for you and then prompt you for your identification number, token, and the registration address.***

***The pollster will prepare a registration request on your behalf and submit it to the registrar. If all goes well, the pollster will collect an acknowledgment from the registrar within a few seconds. Then, the pollster will prompt you for a file name for saving your registration information. Select a name you will remember, as you will need to tell the pollster***

*the name of your registration file every time you vote. If you are registered with more than one election authority, make sure you store your registration information in separate files. All Sensus files will be stored in your* **.sensus** *directory; if you do not have one, the pollster will create one for you.*

# Marking Your Ballot

***Before you can mark a ballot, you must obtain the unvoted ballot for the election and place it in your* .sensus *directory. You must also be registered to vote in that election.***

***Start by running the pollster module as you did when you registered to vote.***

***If you would like to review the ballot before you mark it, select "view ballot questions and instructions" from the pollster menu.***

***When you are ready to mark your ballot, select "mark ballot" from the pollster menu.***

***The pollster will prompt you for the name of the ballot and your registration file name.***

***The pollster will then display the ballot questions one at a time along with instructions for responding to each question.***

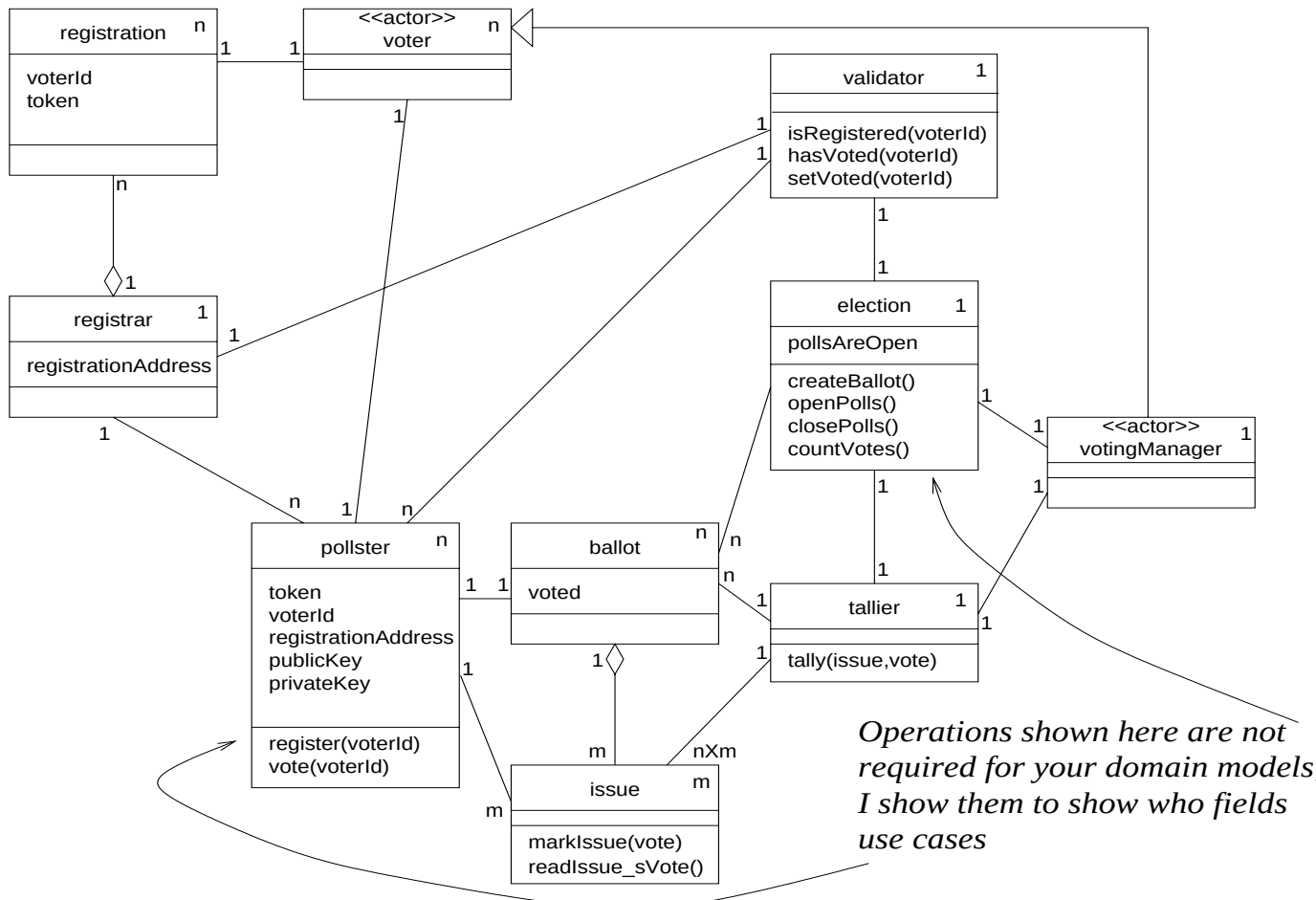
***[Why not display before, when viewing?]***

***If you change your mind or make a mistake marking your ballot, you can remark your ballot. At this time it is not possible to change your Response to some ballot questions without remarking your entire ballot.***

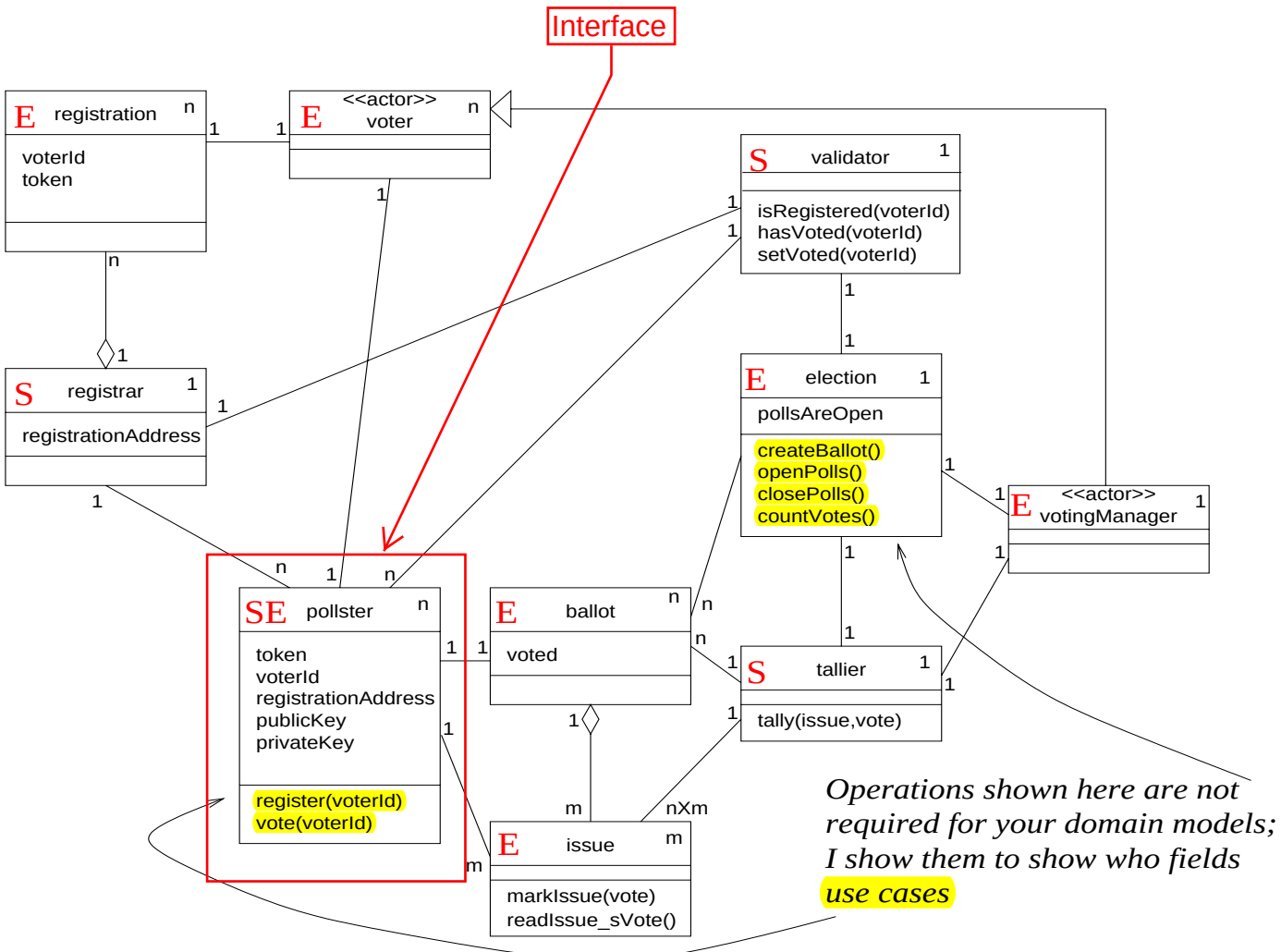
***When you have finished marking your ballot, the pollster will prompt you to continue the voting process. By answering yes at each of the prompts, you can authorize the pollster to complete the entire voting process on your behalf immediately. This process usually takes a few minutes. If you do not want to***

***complete the process right away, you can exit from the pollster program and run it again later to pick up where you left off.***

# Class Diagram

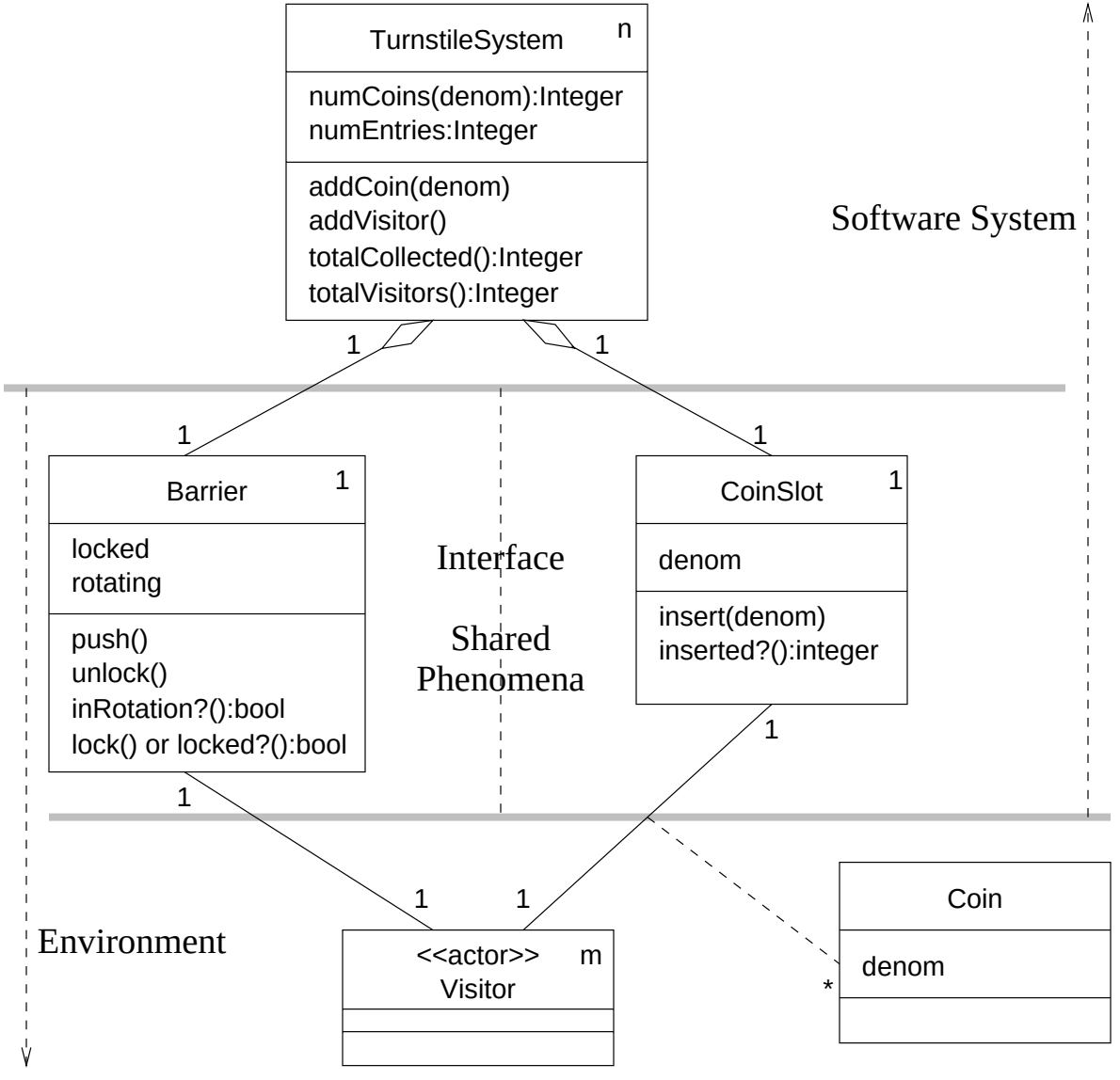


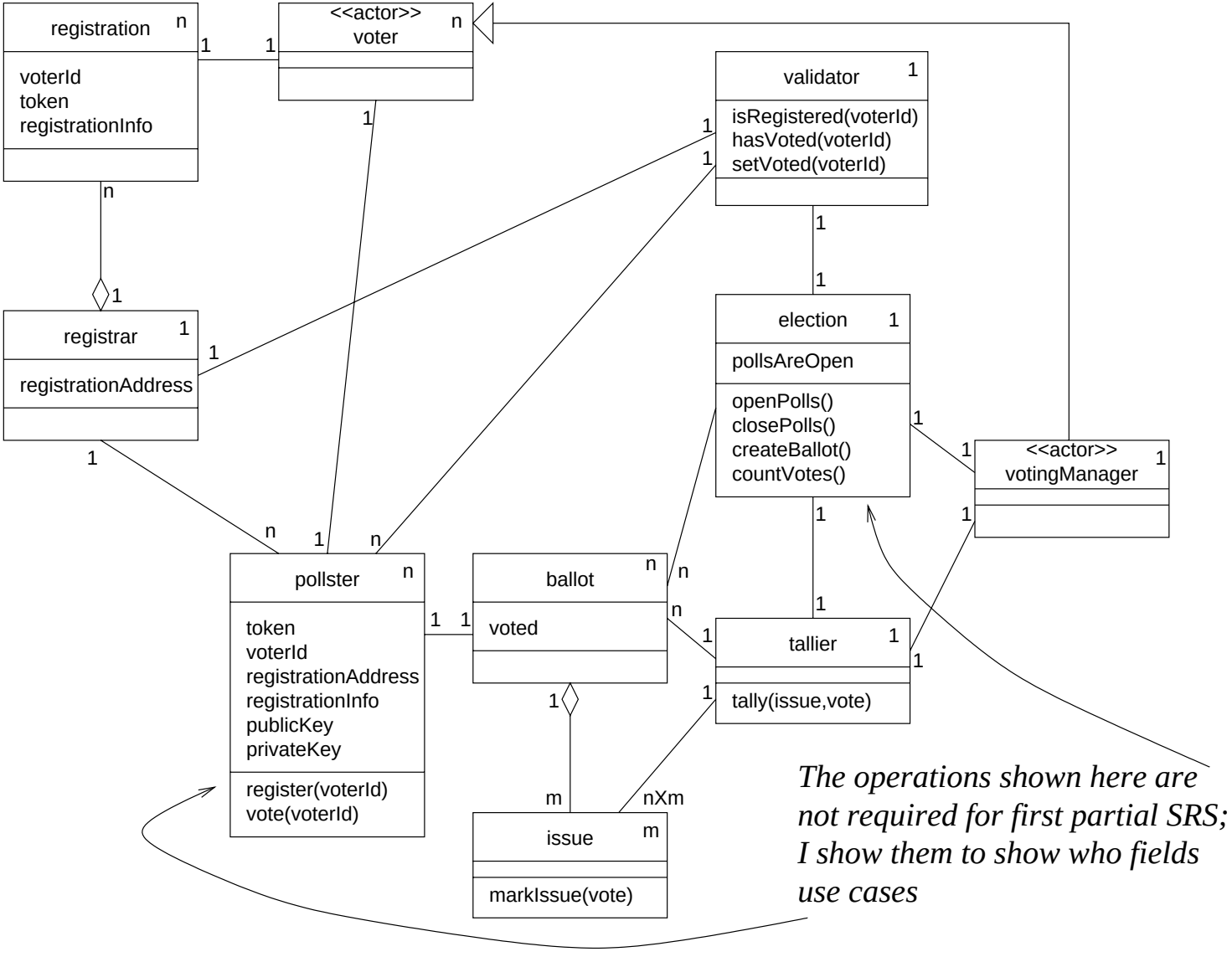
# Class Diagram & World Model



# Scenarios and Use Cases

**In the next lecture, we will develop use cases from the scenarios given for registering to vote and for voting.**


TurnstileSystem
n
numCoins(denom):Integer
numEntries:Integer
addCoin(denom)
addVisitor()
totalCollected():Integer
totalVisitors():Integer
Software System
1
1
1
1
Barrier
1
locked
rotating
push()
unlock()
inRotation?():bool
lock() or locked?():bool
Interface
Shared
Phenomena
CoinSlot
1
denom
insert(denom)
inserted?():integer
1
1
1
1
Environment
<<actor>>
Visitor
m
Coin
denom
*


registration
n
voterId
token
registrationInfo
<<actor>>
voter
n
validator
1
isRegistered(voterId)
hasVoted(voterId)
setVoted(voterId)
registrar
1
registrationAddress
election
1
pollsAreOpen
openPolls()
closePolls()
createBallot()
countVotes()
<<actor>>
votingManager
1
pollster
n
token
voterId
registrationAddress
registrationInfo
publicKey
privateKey
register(voterId)
vote(voterId)
ballot
n
voted
tallier
1
tally(issue,vote)
issue
m
markIssue(vote)
nXm
The operations shown here are not required for first partial SRS; I show them to show who fields use cases