Requirements Modeling

Requirements analysis

- specifies
  - software's operational characteristics
  - software's interface with other system elements
  - Establishes constraints that software must meet

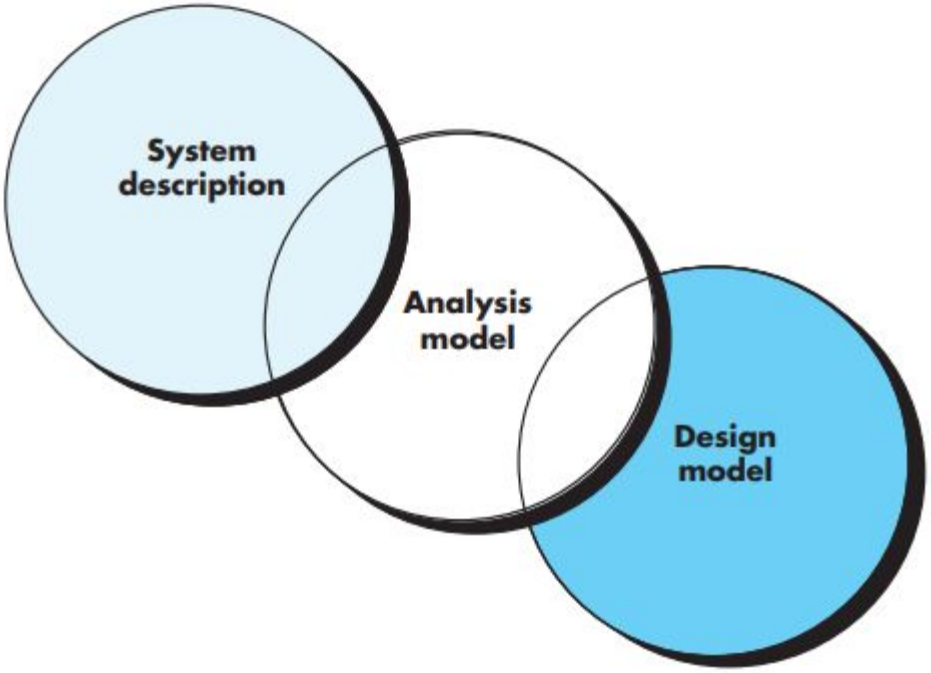
- Requirements analysis allows you to elaborate basic requirements established from
  - inception
  - Elicitation
  - Negotiation

  of requirement engineering

- requirements modeling action results in one or more of the following types of models
  - Scenario based models – requirements from the view of various actors
  - Data models – information domain for the problem
  - Class-oriented models – (class: attributes and operations), (collaboration of classes)
  - Flow-oriented models – transformation of data as it moves through the system
  - Behavioral models –behavior of the system as consequence of external events

- Models are translated
  - Architectural
  - Interface
  - component-level designs

The requirements model as a bridge between the system description and the design model



Overall Objectives and Philosophy

- your primary focus is on what, not how
  - What user interaction occurs in a particular circumstance?
  - what objects does the system manipulate?
  - what functions must the system perform?
  - what behaviors does the system exhibit?
  - what constraints apply?
- The analyst
  - should model what is known
  - use that model as the basis for design of the software increment

- The requirements model must achieve three primary objectives:

  (1) to describe what the customer requires,

  (2) to establish a basis for the creation of a software design,

  (3) to define a set of requirements that can be validated once the software is built

- all elements of the requirements model will be directly traceable to parts of the design model

Analysis Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain
- Each element of the requirements model should
  - add to an overall understanding of software requirements
  - provide insight into the information domain, function, and behavior of the system
- Delay consideration of infrastructure and other nonfunctional models until design
- Minimize coupling throughout the system
- Be certain that the requirements model provides value to all stakeholders
- Keep the model as simple as it can be

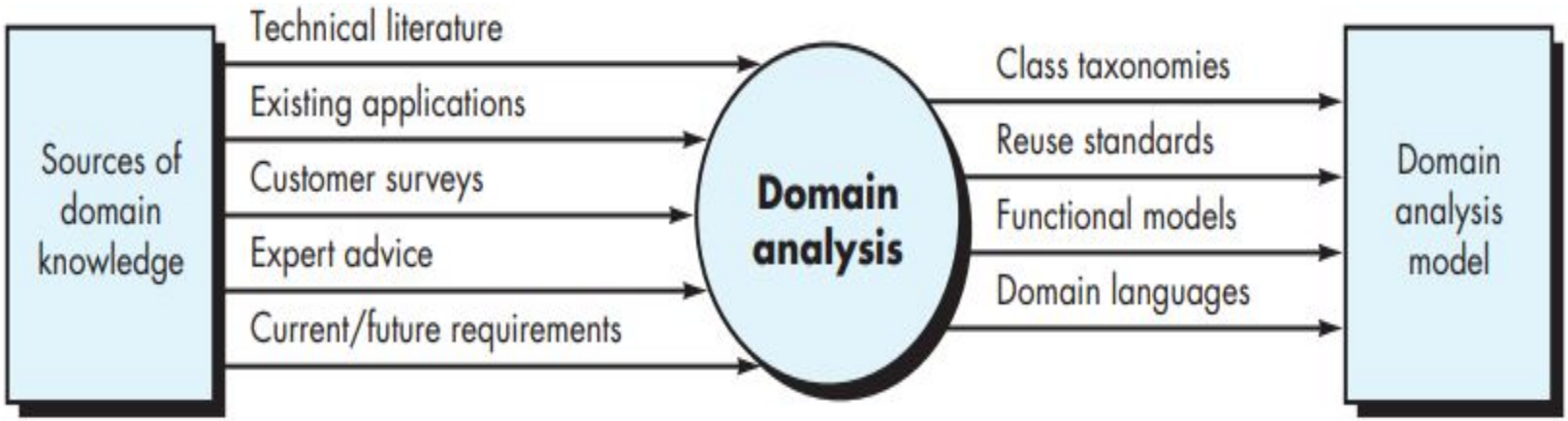
- If analysis patterns are defined and categorized in a manner that allows you
  - to recognize
  - apply them to solve common problems

the creation of the analysis model is expedited

- analysis patterns improves time-to-market and reduces development costs

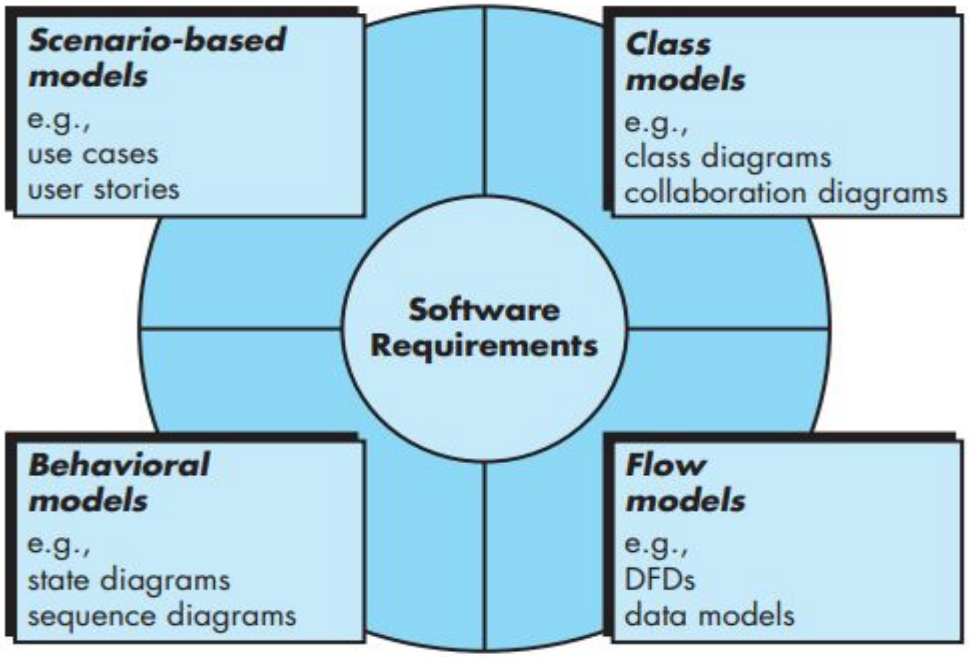
- Software domain analysis is
  - the identification
  - Analysis
  - specification

  of common requirements from a specific application domain
  - typically for reuse on multiple projects within that application domain



Requirements Modeling Approaches

- structured analysis
  - considers data and the processes that transform the data as separate entities
- object-oriented analysis
  - classes and the manner in which they collaborate with one another



Scenario-Based Modeling

- Helps to understand how end users interact with a system
- the creation of scenarios in the form of
  - use cases
  - activity diagrams
  - swimlane diagrams

Creating a Preliminary Use Case

- list the functions or activities performed by a specific actor
  - **Select camera** to view
  - **Request thumbnails** from all cameras
  - **Display camera views** in a PC window
  - **Control pan and zoom** for a specific camera
  - Selectively **record camera output**
  - **Replay camera output**
  - **Access camera surveillance** via the Internet

Use case narrative

- use cases are written first in an informal narrative fashion
- Function access camera surveillance via the Internet—display camera views (ACS-DCV)

Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)

Actor: homeowner

If I'm at a remote location, I can use any PC with appropriate browser software to log on to the *SafeHome Products* website. I enter my user ID and two levels of passwords and once I'm validated, I have access to all functionality for my installed *SafeHome* system. To access a specific camera view, I select "surveillance" from the major function buttons displayed. I then select "pick a camera" and the floor plan of the house is displayed. I then select the camera that I'm interested in. Alternatively, I can look at thumbnail snapshots from all cameras simultaneously by selecting "all cameras" as my viewing choice. Once I choose a camera, I select "view" and a one-frame-per-second view appears in a viewing window that is identified by the camera ID. If I want to switch cameras, I select "pick a camera" and the original viewing window disappears and the floor plan of the house is displayed again. I then select the camera that I'm interested in. A new viewing window appears.

- A variation of a narrative use case presents the interaction as an ordered sequence of user actions

Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)

Actor: homeowner

1. The homeowner logs onto the *SafeHome Products* website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the "surveillance" from the major function buttons.
6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the "view" button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Refining a Preliminary Use Case

- A description of alternative interactions is essential for a complete understanding of the function
- each step in the primary scenario is evaluated by asking the following questions
  - Can the actor take some other action at this point?
  - Is it possible that the actor will encounter some error condition at this point? If so what might it be?
  - Is it possible that the actor will encounter some other behavior at this point? If so what might it be?
- Answers to these questions result in the creation of secondary scenarios

6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.

- ***Can the actor take some other action at this point?***
    - Referring to the free-flowing narrative, the actor may choose to view thumbnail snapshots of all cameras simultaneously
- ***Is it possible that the actor will encounter some error condition at this point?***
    - Floor plan with camera icons may have never been configured
    - selecting “pick a camera” results in an error condition: “No floor plan configured for this house”
- ***Is it possible that the actor will encounter some other behavior at this point?***
    - the system may encounter an alarm condition
    - providing the actor with a number of options relevant to the nature of the alarm
- Each of these is characterized as a use-case exception
    - Causing the system to exhibit somewhat different behavior
    - Should derive a complete set of exceptions for each use case

Additionally,

- Are there cases in which some “validation function” occurs during this use case?
- Are there cases in which a supporting function (or actor) will fail to respond appropriately?
- Can poor system performance result in unexpected or improper user actions?

Then

- The list of extensions developed
- an exception should be noted
- Solution to handle the exception should be presented

Writing a Formal Use Case

Use Case Template for Surveillance

Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)

Iteration: 2, last modification: January 14 by V. Raman.

Primary actor: Homeowner.

Goal in context: To view output of camera placed throughout the house from any remote location via the Internet.

Preconditions: System must be fully configured; appropriate user ID and passwords must be obtained.

Trigger: The homeowner decides to take a look inside the house while away.

Scenario:

1. The homeowner logs onto the *SafeHome Products* website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the "surveillance" from the major function buttons.
6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the "view" button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Exceptions:

1. ID or passwords are incorrect or not recognized—see use case **Validate ID and passwords.**
2. Surveillance function not configured for this system—system displays appropriate error message; see use case **Configure surveillance function.**
3. Homeowner selects "View thumbnail snapshots for all camera"—see use case **View thumbnail snapshots for all cameras.**
4. A floor plan is not available or has not been configured—display appropriate error message and see use case **Configure floor plan.**
5. An alarm condition is encountered—see use case **alarm condition encountered.**

Priority: Moderate priority, to be implemented after basic functions.

When available: Third increment.

Frequency of use: Infrequent.

Channel to actor: Via PC-based browser and Internet connection.

Secondary actors: System administrator, cameras.

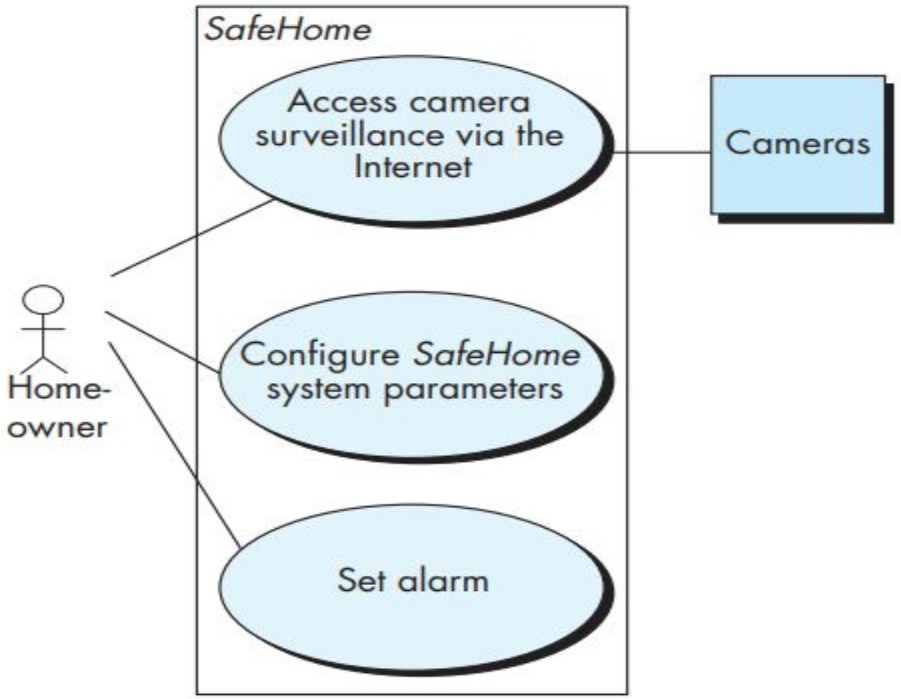
Channels to secondary actors:

1. System administrator: PC-based system.
2. Cameras: wireless connectivity.

Open issues:

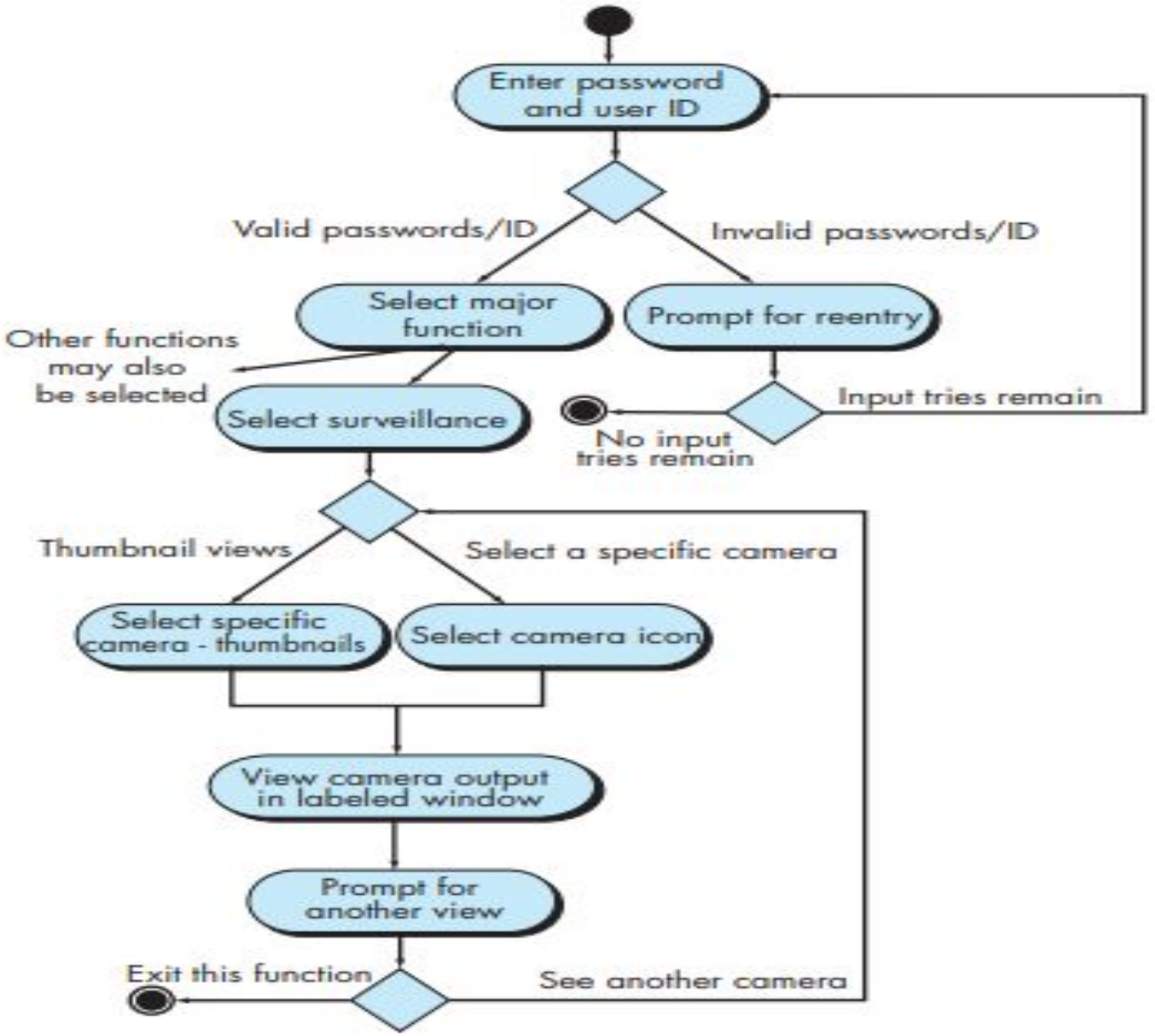
1. What mechanisms protect unauthorized use of this capability by employees of *SafeHome Products*?
2. Is security sufficient? Hacking into this feature would represent a major invasion of privacy.
3. Will system response via the Internet be acceptable given the bandwidth required for camera views?
4. Will we develop a capability to provide video at a higher frames-per-second rate when high-bandwidth connections are available?

Preliminary use-case diagram for the *SafeHome* system



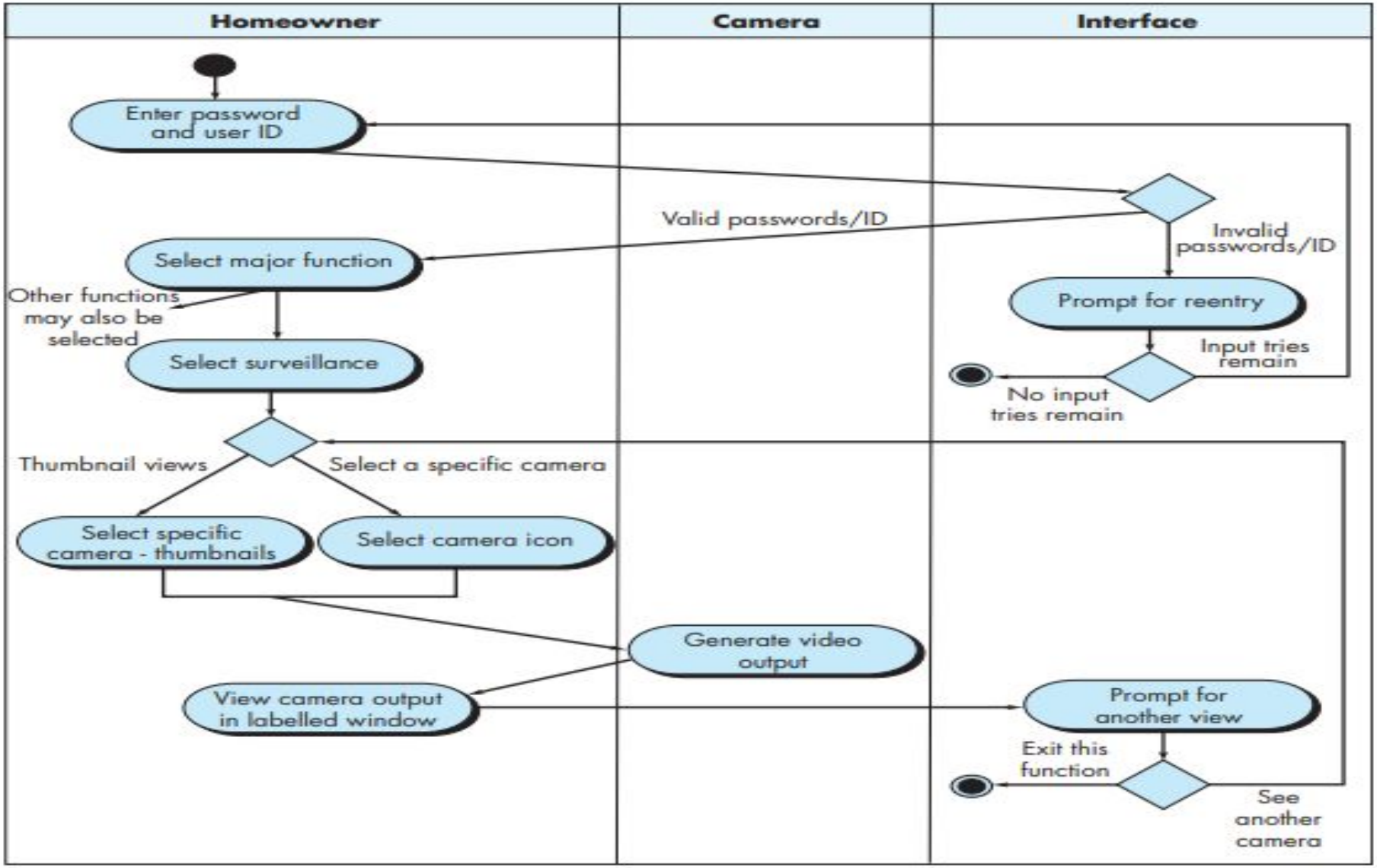
- If the description is unclear, the use case can be misleading or ambiguous
- A use case focuses on
  - functional and behavioral requirements
  - inappropriate for nonfunctional requirements

UML Models That Supplement the Use Case



Activity diagram for Access camera surveillance via the Internet—display camera views function.

Swimlane diagram for Access camera surveillance via the Internet—display camera views function



Data Modeling Concepts

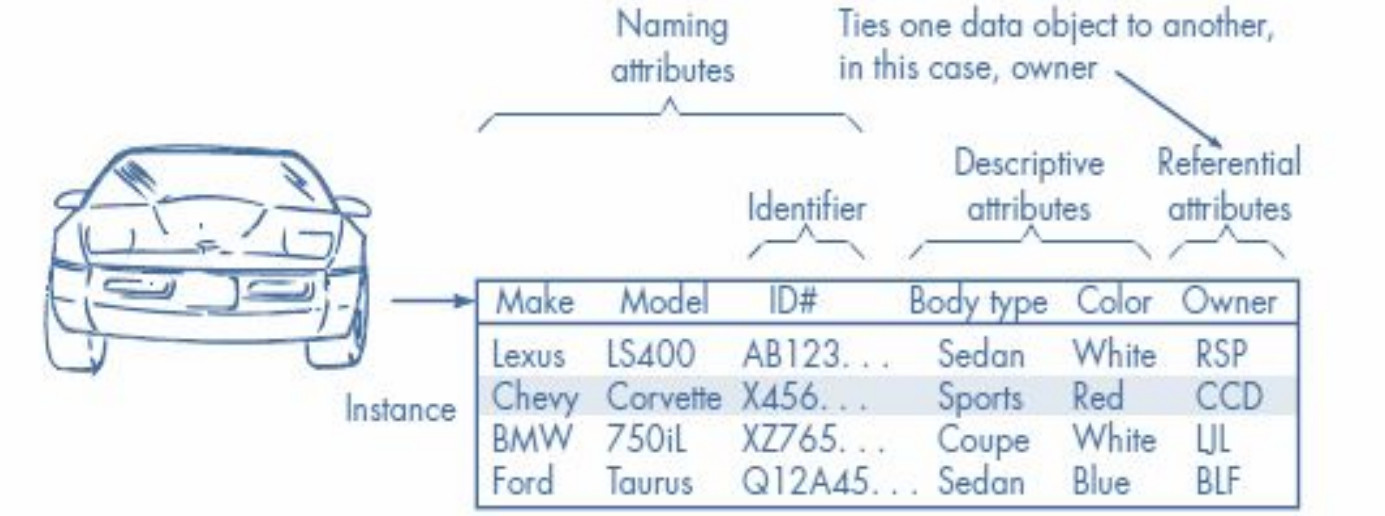
- all data objects that are processed within the system, the relationships between the data objects
- entity-relationship diagram (ERD) represents all data objects
  - Entered
  - Stored
  - transformed,
  - produced within an application

Data Objects

- A data object is a representation of composite information that must be understood by the system

- Data object can be an
  - External entity (anything that produces or consumes information)
  - a thing (report or a display)
  - an occurrence (e.g., a telephone call)
  - or event (e.g., an alarm),
  - a role (e.g., salesperson),
  - an organizational unit (e.g.,accounting department),
  - a place (e.g., a warehouse),
  - or a structure (e.g., a file).

- A data object encapsulates data only – not the functions

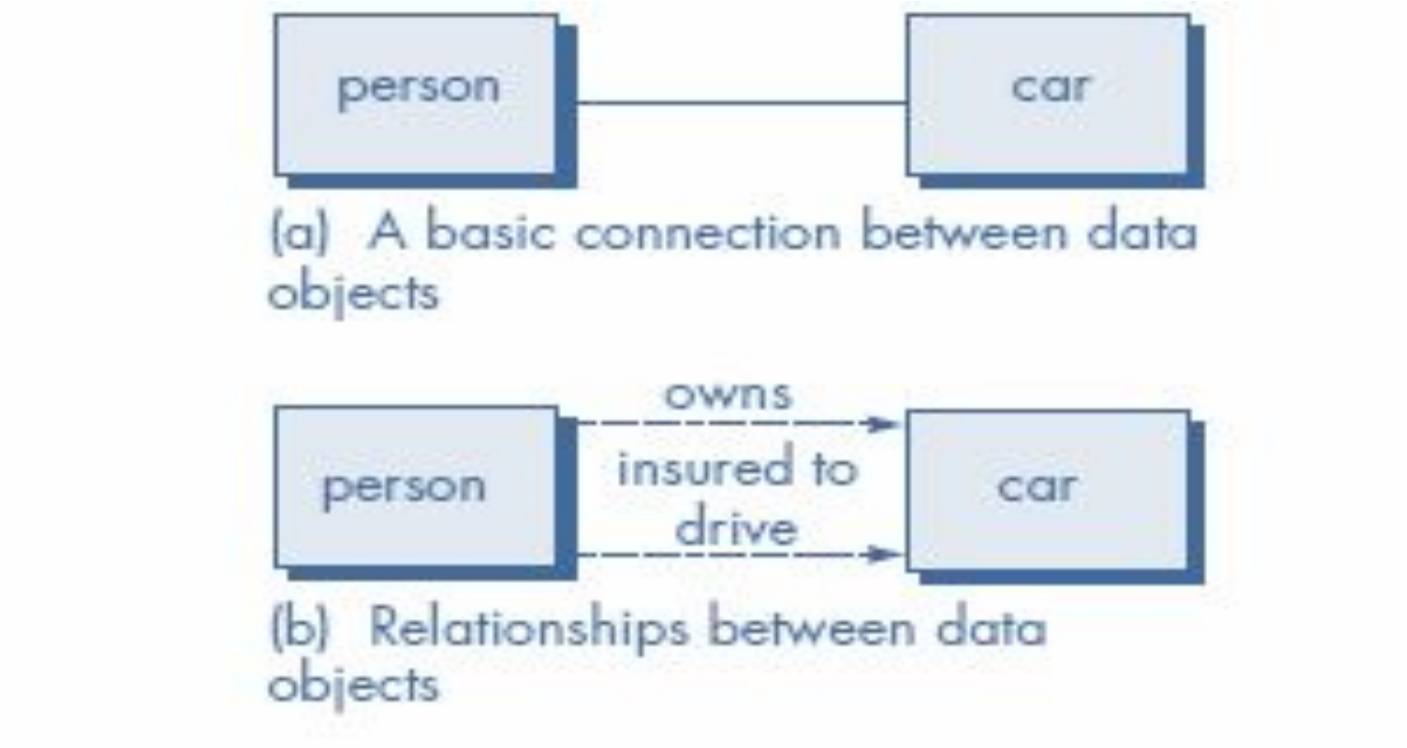


Data Attributes

Data attributes define the properties of a data object and take on one of three different characteristics.
They can be used to
(1) name an instance of the data object,
(2) describe the instance, or
(3) make reference to another instance in another table

Relationships

- Data objects are connected to one another in different ways



- A person *owns* a car.
- A person *is insured to drive* a car.

Class-Based Modeling

- The elements of a class-based model include
  - classes and objects,
  - attributes,
  - operations,
  - classresponsibility-collaborator (CRC) models,
  - collaboration diagrams, and
  - Packages

Identifying Analysis Classes

- identify classes by examining the usage scenarios
- Do a “grammatical parse”
- Determine each noun or noun phrase
- enter it into a simple table
- Synonyms should be noted
- If the class (noun) is required to implement a solution
  - Solution space
- a class is necessary only to describe a solution
  - Problem space

Analysis classes

- External entities (other systems, devices, people) that produces or consumes information
- Things (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
- Occurrences or events (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation
- Roles (e.g., manager, engineer, salesperson) played by people who interact with the system
- Organizational units (e.g., division, group, team) that are relevant to an application

- Places (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system

- Structures (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects

For example, Budd [Bud96] suggests a taxonomy of classes that includes

- Producers (sources) and consumers (sinks) of data,
- data managers,
- view or observer classes, and
- helper classes.

The SafeHome security function *enables* the homeowner to *configure* the security system when it is *installed*, *monitors* all sensors *connected* to the security system, and *interacts* with the homeowner through the Internet, a PC, or a control panel.

During installation, the SafeHome PC is used to *program* and *configure* the system. Each sensor is assigned a number and type, a master password is programmed for *arming* and *disarming* the system, and telephone number(s) are *input* for *dialing* when a sensor event occurs.

When a sensor event is *recognized*, the software *invokes* an audible alarm attached to the system. After a delay time that is *specified* by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, *provides* information about the location, *reporting* the nature of the event that has been detected. The telephone number will be *redialed* every 20 seconds until telephone connection is *obtained*.

The homeowner *receives* security information via a control panel, the PC, or a browser, collectively called an interface. The interface *displays* prompting messages and system status information on the control panel, the PC ,or the browser window. Homeowner interaction takes the following form . . .

Potential Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity

Characteristics of potential classes

- Retained information
- Needed services
- Multiple attributes
- Common attributes
- Common operations
- Essential requirements

Potential Class	Characteristic Number That Applies
homeowner	rejected: 1, 2 fail even though 6 applies
sensor	accepted: all apply
control panel	accepted: all apply
installation	rejected
system (alias security function)	accepted: all apply
number, type	rejected: 3 fails, attributes of sensor
master password	rejected: 3 fails
telephone number	rejected: 3 fails
sensor event	accepted: all apply
audible alarm	accepted: 2, 3, 4, 5, 6 apply
monitoring service	rejected: 1, 2 fail even though 6 applies

- some of the rejected potential classes will become attributes for those classes that were accepted
    - number and type are attributes of Sensor
    - master password and telephone number may become attributes of System

Specifying Attributes

- attributes describe and define a class in the context of a problem space

- For an instance,
  - you want to track baseball statistics for professional baseball players

    Player:
    - name,
    - position,
    - batting average,
    - fielding percentage,
    - years played,
    - and games played

    - In the context of professional baseball pension system of a player the attributes would be

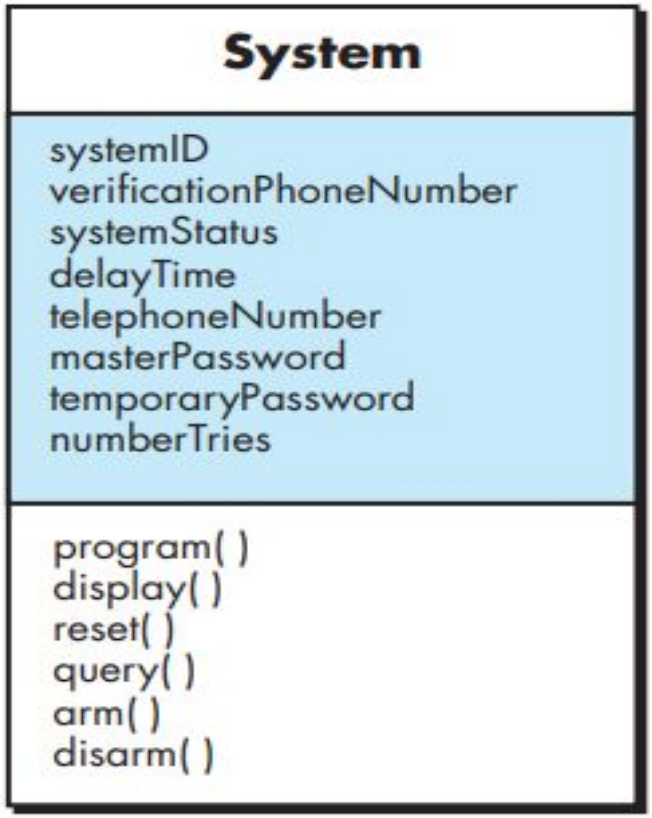
    Player:
    - average salary,
    - credit toward full vesting,
    - pension plan options chosen,
    - Mailing address

System class

identification information = system ID + verification phone number + system status

alarm response information = delay time + telephone number

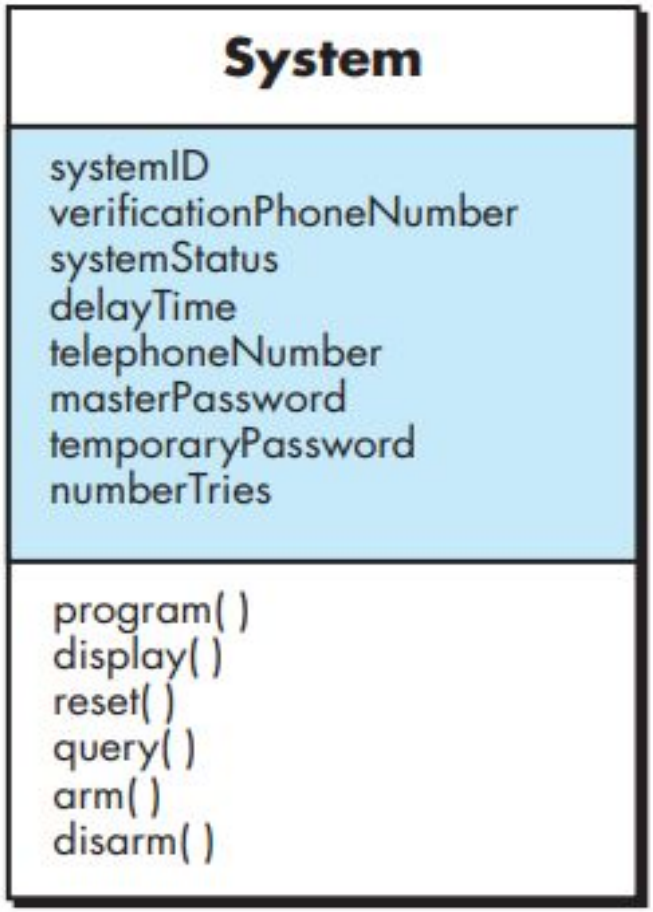
activation/deactivation information = master password + number of allowable tries + temporary password

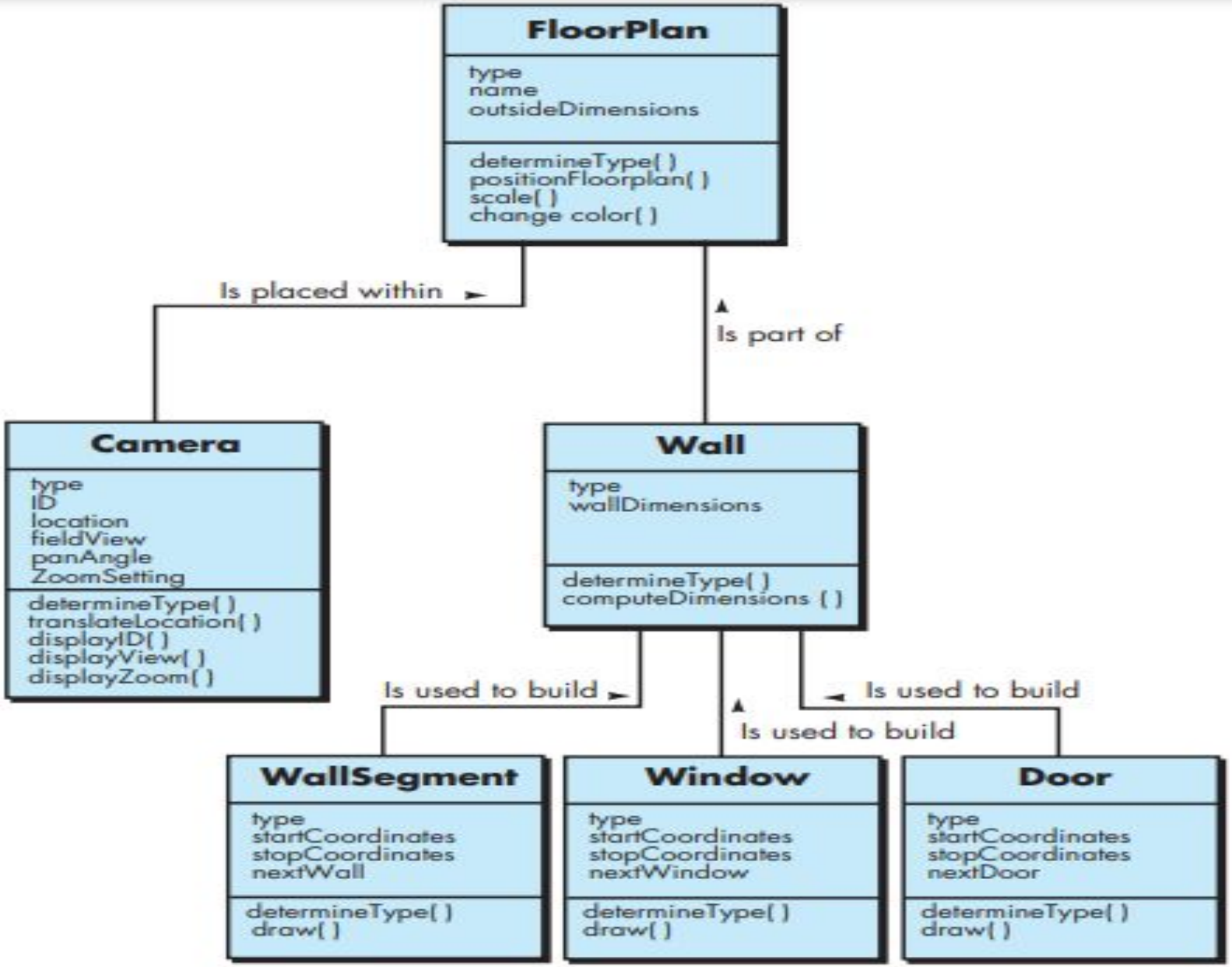


Defining Operations

- Operations define the behavior of an object
- (1) Operations manipulate data in some way (e.g., adding, deleting, reformatting, selecting),
- (2) operations perform a computation,
- (3) operations inquire about the state of an object
- (4) operations monitor an object for the occurrence of a controlling event

master password is *programmed* for *arming and disarming* the system

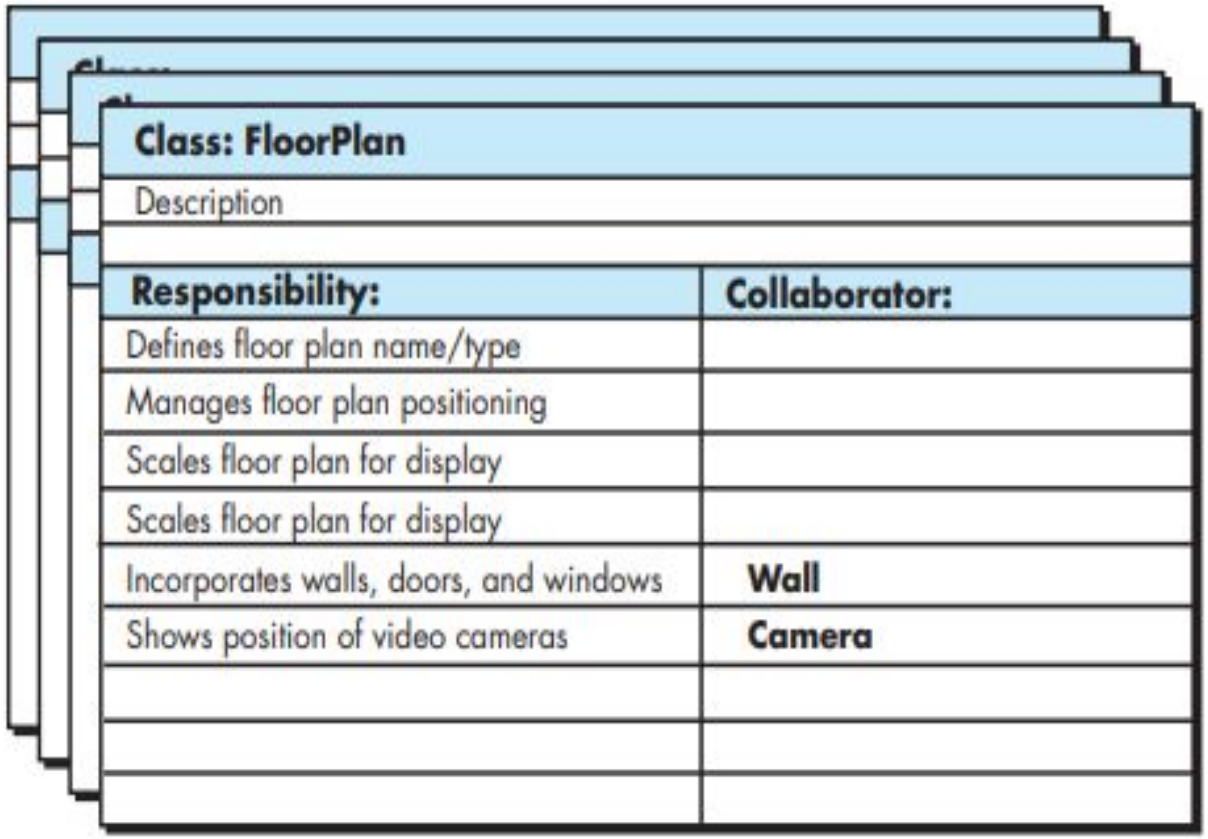



FloorPlan
type
name
outsideDimensions
determineType( )
positionFloorplan( )
scale( )
change color( )
Is placed within
Is part of
Camera
type
ID
location
fieldView
panAngle
ZoomSetting
determineType( )
translateLocation( )
displayID( )
displayView( )
displayZoom( )
Wall
type
wallDimensions
determineType( )
computeDimensions ( )
Is used to build
Is used to build
Is used to build
WallSegment
type
startCoordinates
stopCoordinates
nextWall
determineType( )
draw( )
Window
type
startCoordinates
stopCoordinates
nextWindow
determineType( )
draw( )
Door
type
startCoordinates
stopCoordinates
nextDoor
determineType( )
draw( )

Class-Responsibility-Collaborator (CRC) Modeling

- Identifies and organizes the classes that are relevant to system or product requirements
- A CRC model is a collection of standard index cards that represent classes
- The cards are divided into three sections
- Along the top of the card you write the name of the class
- In the body of the card you list the class responsibilities on the left
- List the collaborators on the right

- CRC model may make use of actual or virtual index cards
- Responsibilities are the attributes and operations that are relevant for the class
- responsibility is “anything the class knows or does”
- Collaborators classes provide information needed to complete a responsibility
- collaboration implies either a request for information or a request for some action



Class: FloorPlan	
Description	
Responsibility:	**Collaborator:**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Scales floor plan for display	
Incorporates walls, doors, and windows	**Wall**
Shows position of video cameras	**Camera**

Classes

taxonomy of class types

- Entity classes, also called model or business classes
    - extracted directly from the statement of the problem
    - represent things that are to be stored in a database
    - persist throughout the duration of the application
- Boundary classes are used to create the interface
    - user sees and interacts with as the software
    - Presents entity objects to users
- Controller classes
    - manage a “unit of work” from start to finish

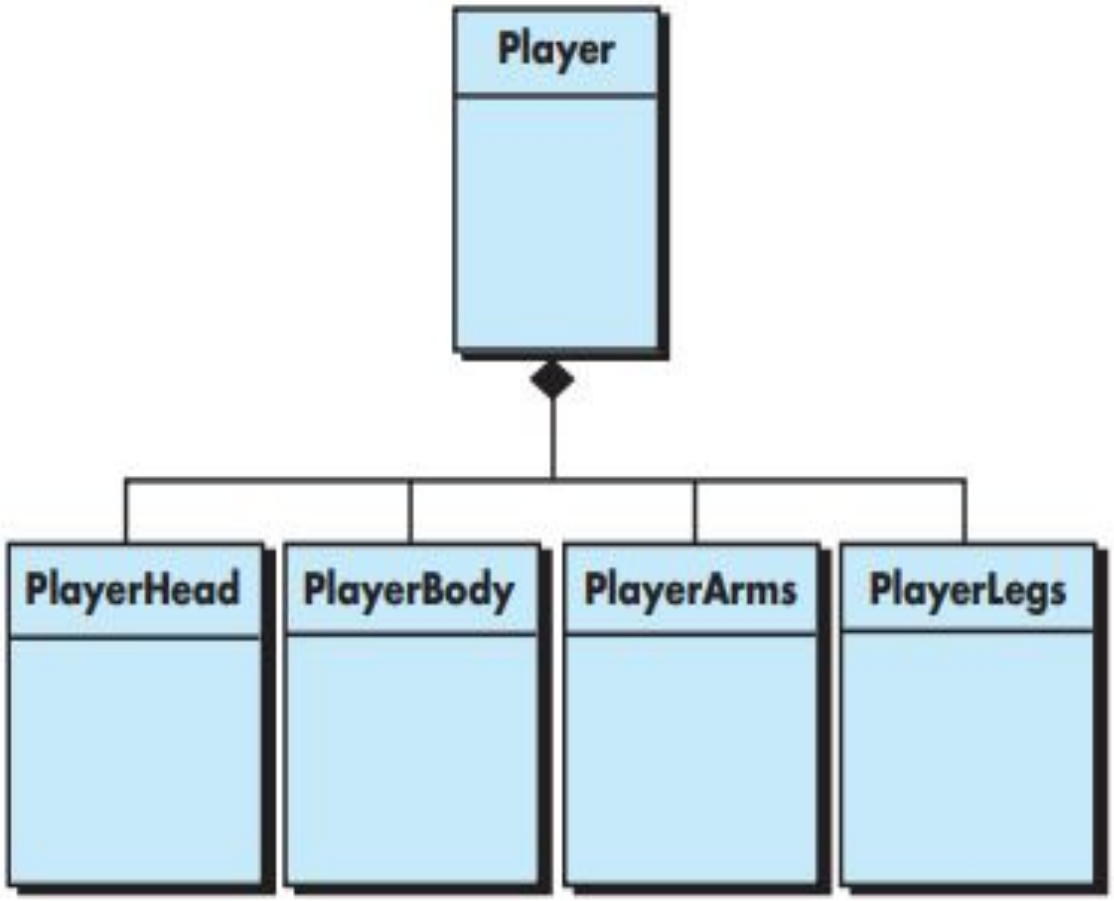
Responsibilities.

- System intelligence should be distributed across classes
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class
- Information about one thing should be localized with a single class
    - not distributed across multiple classes
- Responsibilities should be shared among related classes, when appropriate

Collaborations.

- A class can use its own operations to manipulate its own attributes,
- a class can collaborate with other classes
- To help in the identification of collaborators you can examine three different generic relationships between classes
    - (1) the is-part-of relationship,
    - (2) the has-knowledge-of relationship,
    - (3) the depends-upon relationship.
- All classes that are part of an aggregate class is in part-of relationship
- When one class must acquire information from another class, the has-knowledge of relationship is established
- depends-upon relationship implies that two classes have a dependency that is not achieved by has-knowledge-of or is-part-of

A composite aggregate class



determine-sensor-status() responsibility has –knowledge of responsibility

- the collaborator class name is recorded on the CRC model index card
- Index card contains a list of responsibilities and the corresponding collaborations
- When a complete CRC model has been developed
  - stakeholders can review the model using the following approach

1. All participants in the review are given a subset of the CRC model index cards
2. Cards that collaborate should be separated
3. All use-case scenarios should be organized into categories
4. The review leader reads the use case deliberately
5. As the review leader comes to a named object
   - passes a token to the person holding the corresponding class index card

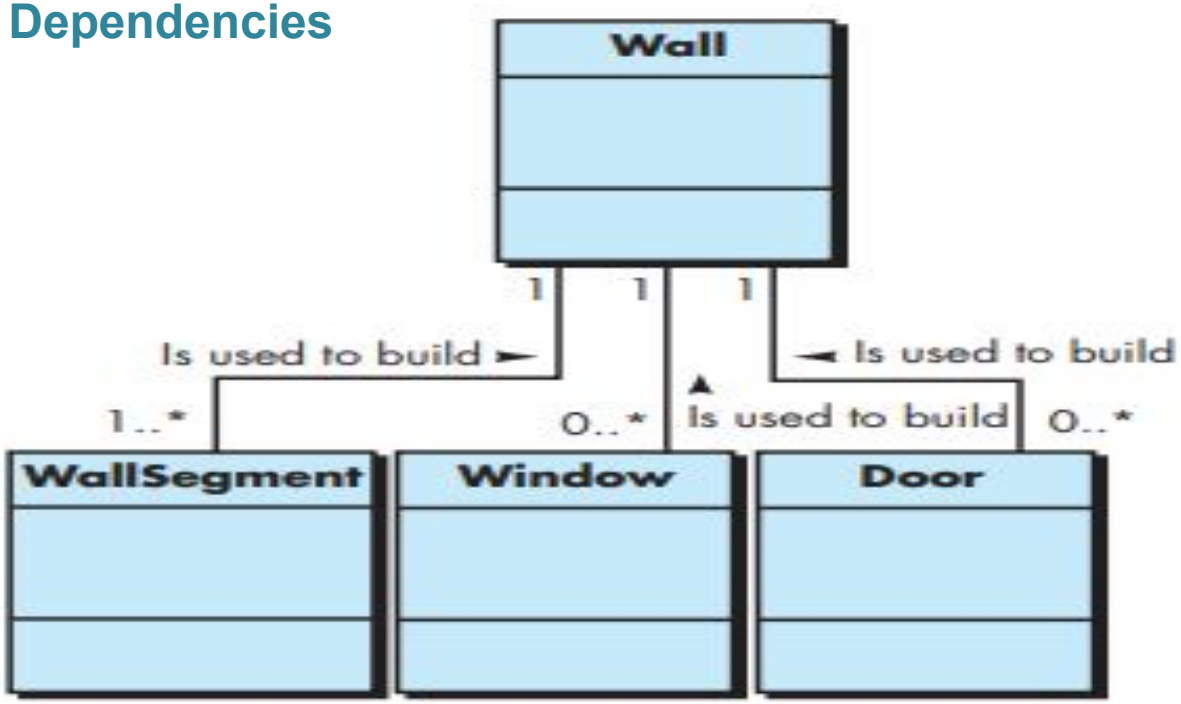
The homeowner observes the SafeHome control panel to determine if the system is ready for input. If the system is not ready, the homeowner must physically close windows/doors so that the ready indicator is present. [A not-ready indicator implies that a sensor is open, i.e., that a door or window is open.]

When the review leader comes to "control panel," in the use case narrative, the token is passed to the person. holding the ControlPanel index card. The phrase "implies that a sensor is open" requires that the index card contains a responsibility that will validate this implication (the responsibility determine-sensor-status() accomplishes this). Next to the responsibility on the index card is the collaborator Sensor. The token is then passed to the Sensor object.

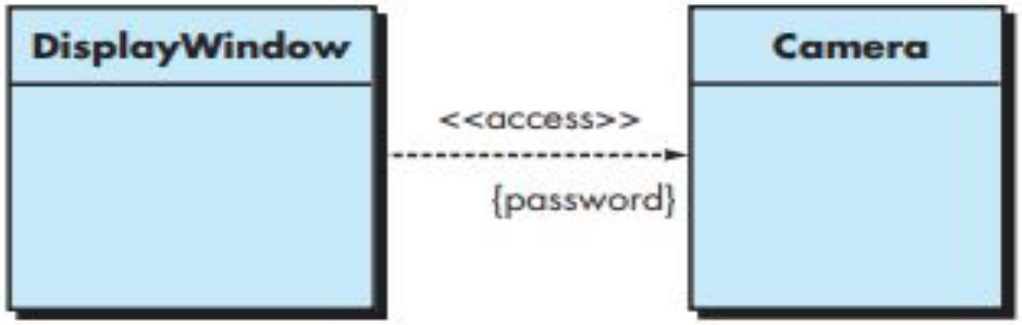
- When the token is passed, the holder of the card is asked
  - to describe the responsibilities noted on the card
- If the responsibilities and collaborations noted on the index cards cannot accommodate the use case
  - modifications are made to the cards

Associations and Dependencies

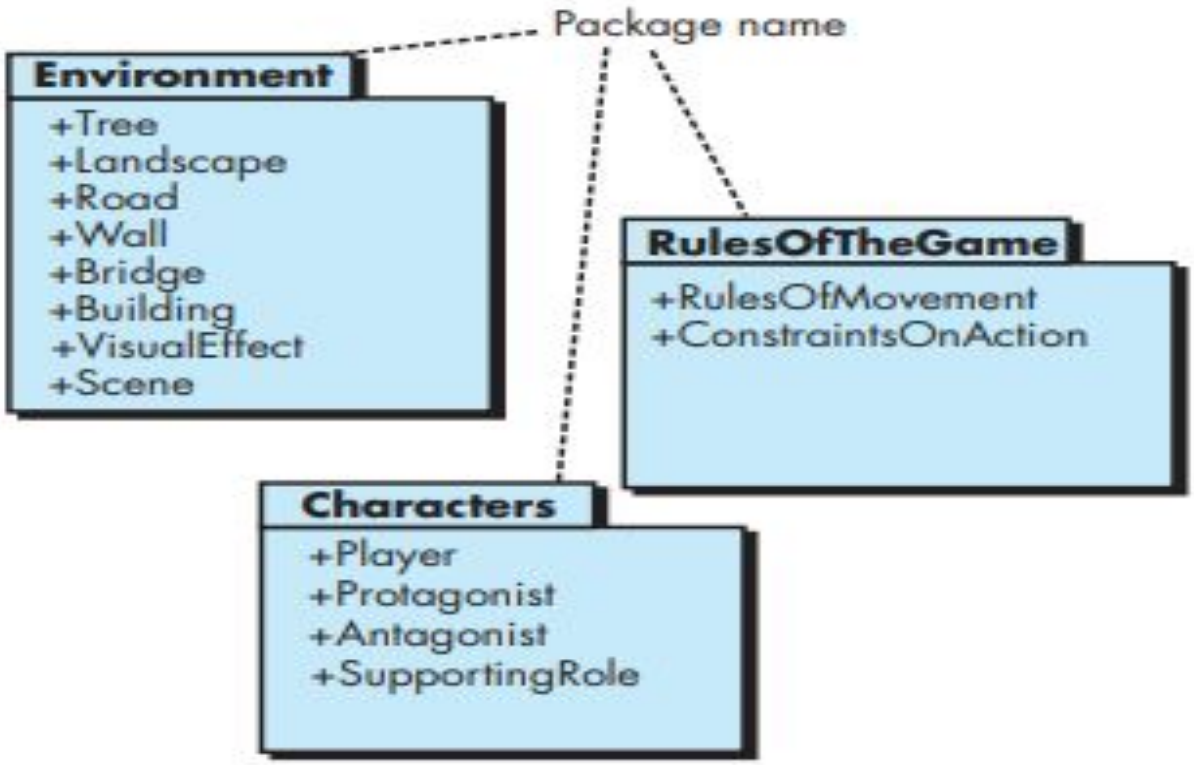
Multiplicity

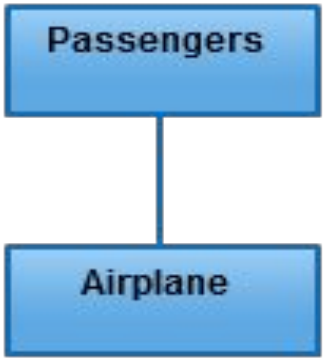


Dependencies

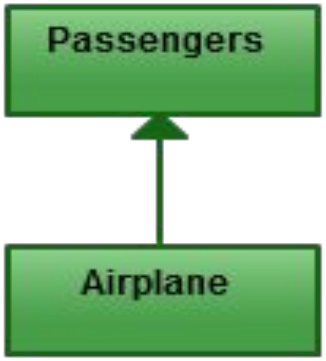


Packages

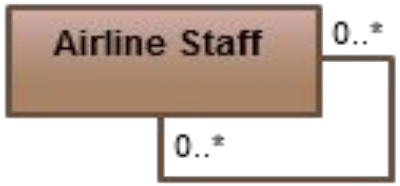

Package name
Environment
+Tree
+Landscape
+Road
+Wall
+Bridge
+Building
+VisualEffect
+Scene
RulesOfTheGame
+RulesOfMovement
+ConstraintsOnAction
Characters
+Player
+Protagonist
+Antagonist
+SupportingRole



Association



Directed Asscoation



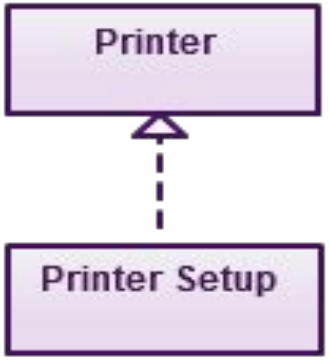
Reflexive Assciation

Multiplicity

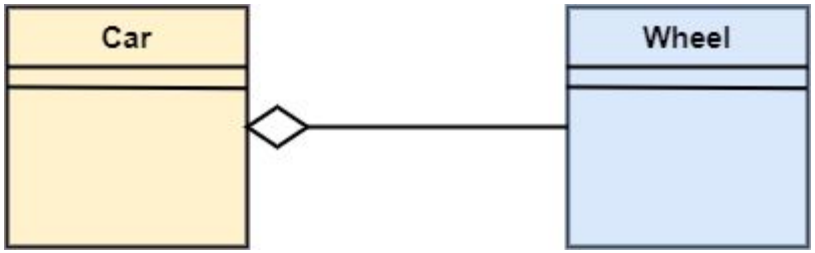
Aggregation

Composition

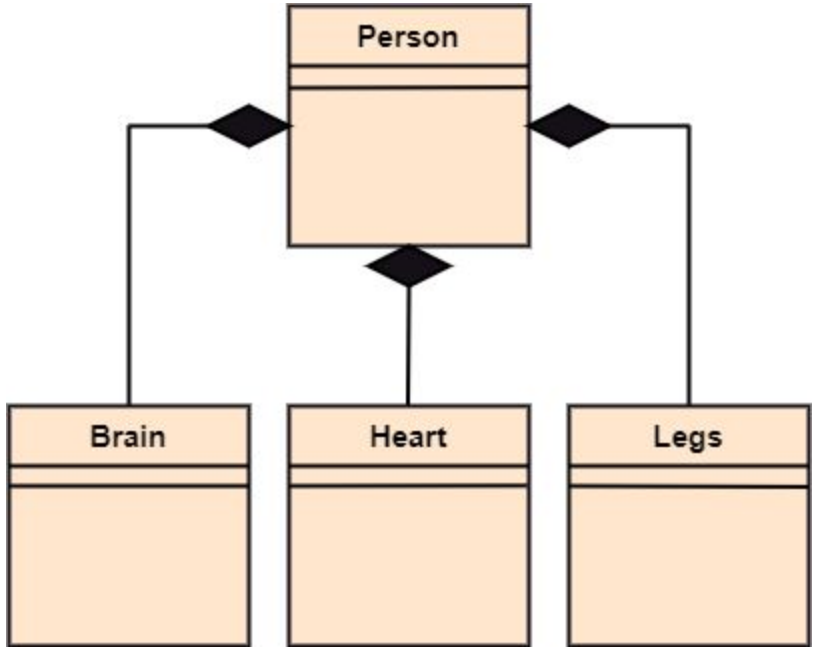
Inheritance



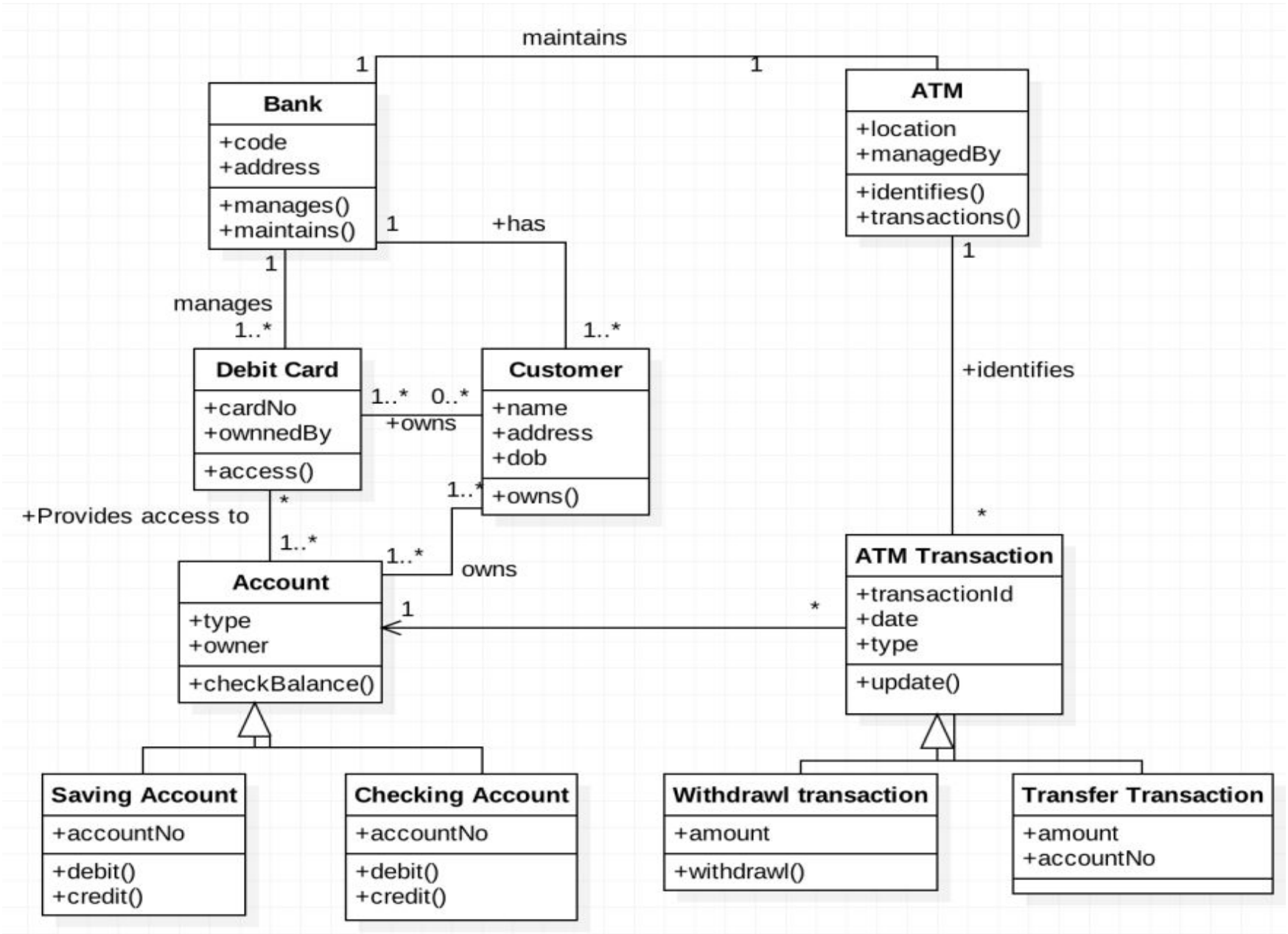
Realization



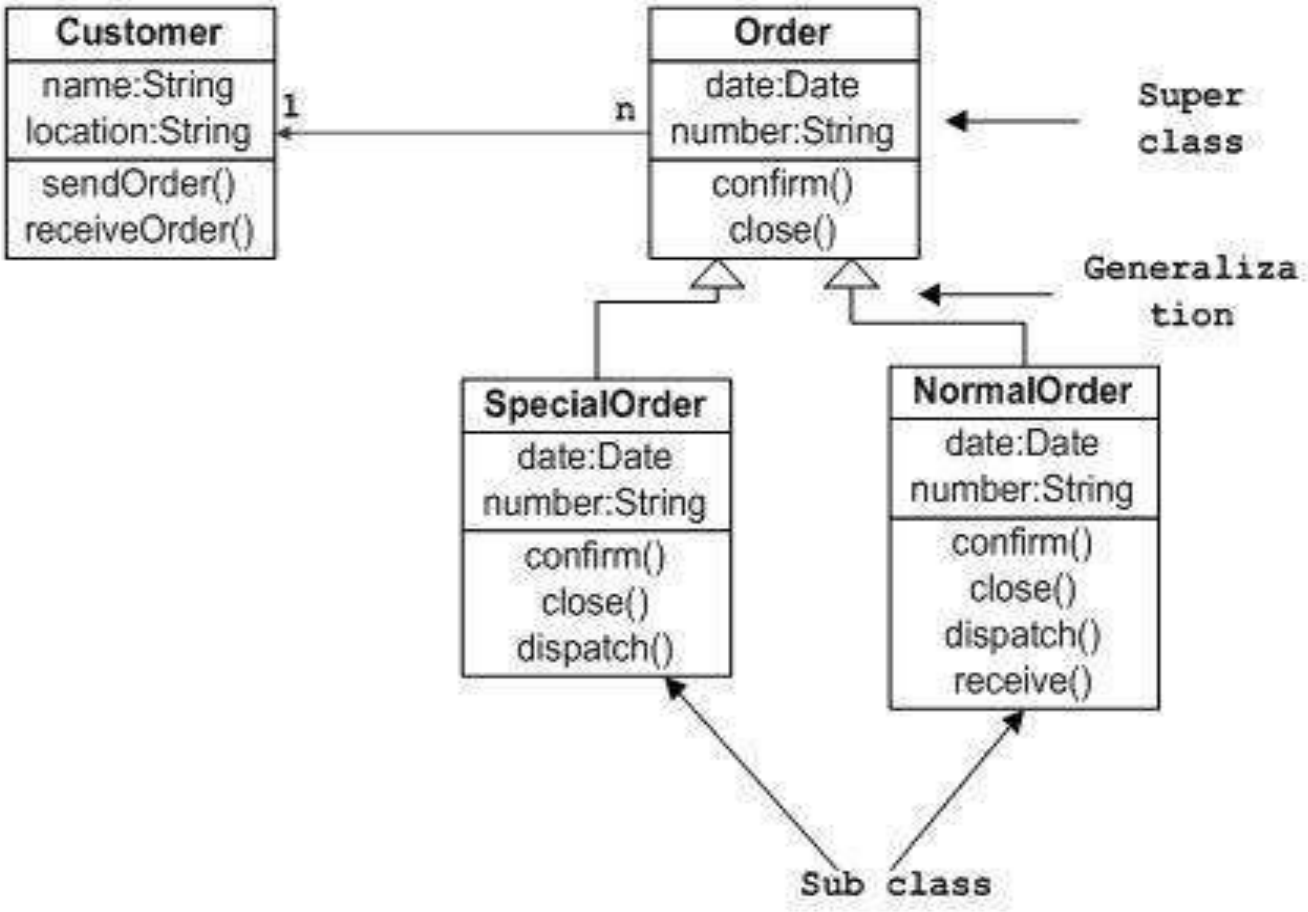
Aggregation

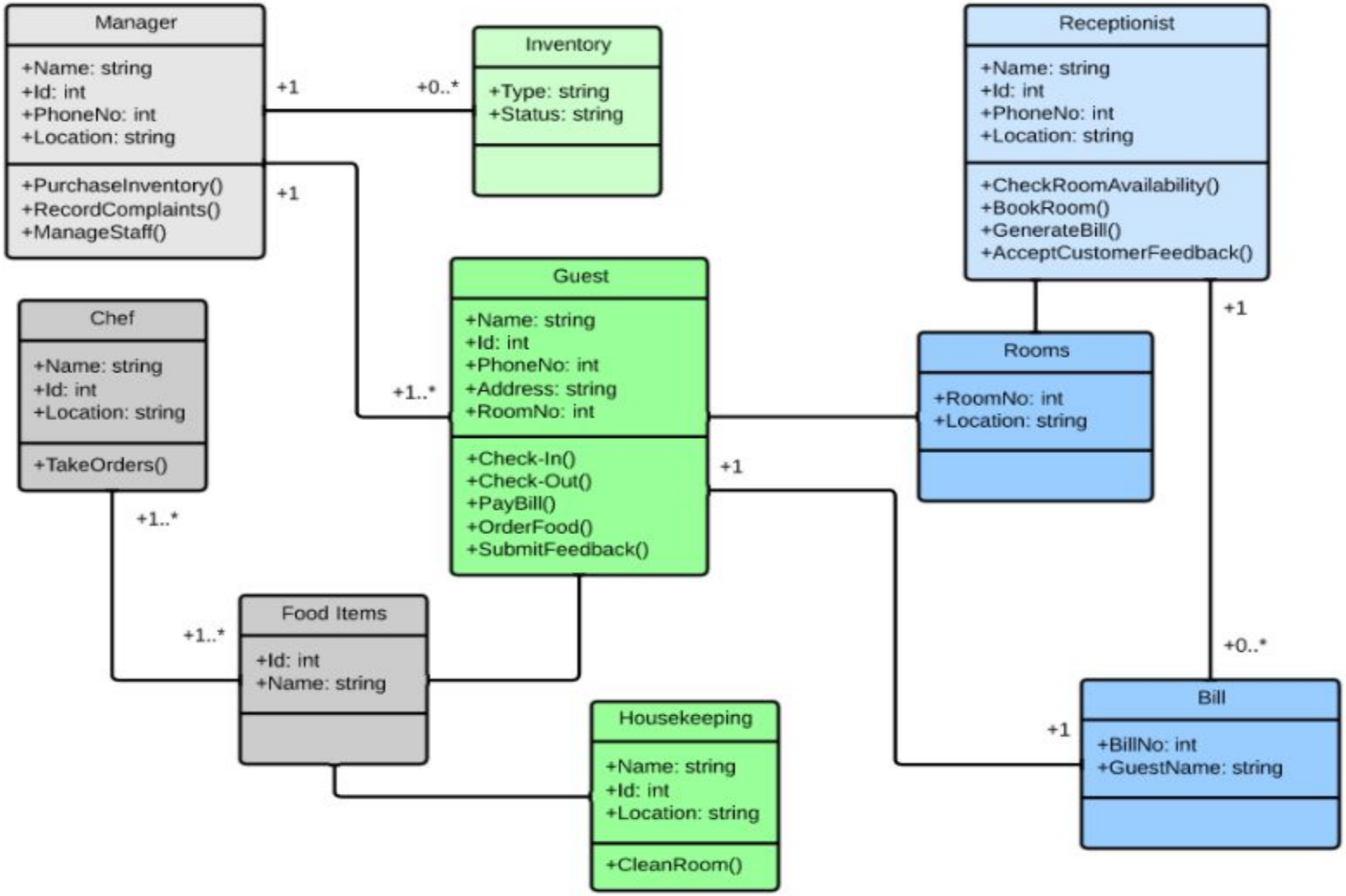


Composite


maintains
1
1
Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedBy
+identifies()
+transactions()
1
+has
1
1
manages
1..*
1..*
1
Debit Card
+cardNo
+ownnedBy
+access()
1..*
0..*
+owns
Customer
+name
+address
+dob
+owns()
+identifies
1..*
*
+Provides access to
1..*
1..*
owns
*
Account
+type
+owner
+checkBalance()
1
*
ATM Transaction
+transactionId
+date
+type
+update()
Saving Account
+accountNo
+debit()
+credit()
Checking Account
+accountNo
+debit()
+credit()
Withdrawl transaction
+amount
+withdrawl()
Transfer Transaction
+amount
+accountNo

Sample Class Diagram




Manager
+Name: string
+Id: int
+PhoneNo: int
+Location: string
+PurchaseInventory()
+RecordComplaints()
+ManageStaff()
Inventory
+Type: string
+Status: string
Receptionist
+Name: string
+Id: int
+PhoneNo: int
+Location: string
+CheckRoomAvailability()
+BookRoom()
+GenerateBill()
+AcceptCustomerFeedback()
Guest
+Name: string
+Id: int
+PhoneNo: int
+Address: string
+RoomNo: int
+Check-In()
+Check-Out()
+PayBill()
+OrderFood()
+SubmitFeedback()
Rooms
+RoomNo: int
+Location: string
Chef
+Name: string
+Id: int
+Location: string
+TakeOrders()
Food Items
+Id: int
+Name: string
Housekeeping
+Name: string
+Id: int
+Location: string
+CleanRoom()
Bill
+BillNo: int
+GuestName: string
+1
+0..*
+1..*