UML MODEL

- **What is UML?**

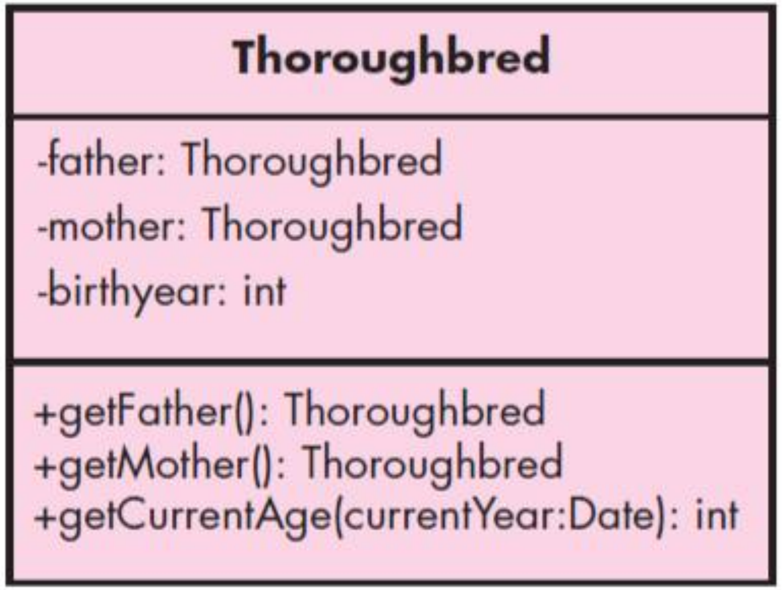
- The Unified Modeling Language (UML) is “a standard language for writing software blueprints.
- UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system”.
- In other words, just as building architects create blueprints to be used by a construction company, software architects create UML diagrams to help software developers build the software.
- If you understand the vocabulary of UML (the diagrams’ pictorial elements and their meanings), you can much more easily understand and specify a system and explain the design of that system to others.

• Class Diagrams:

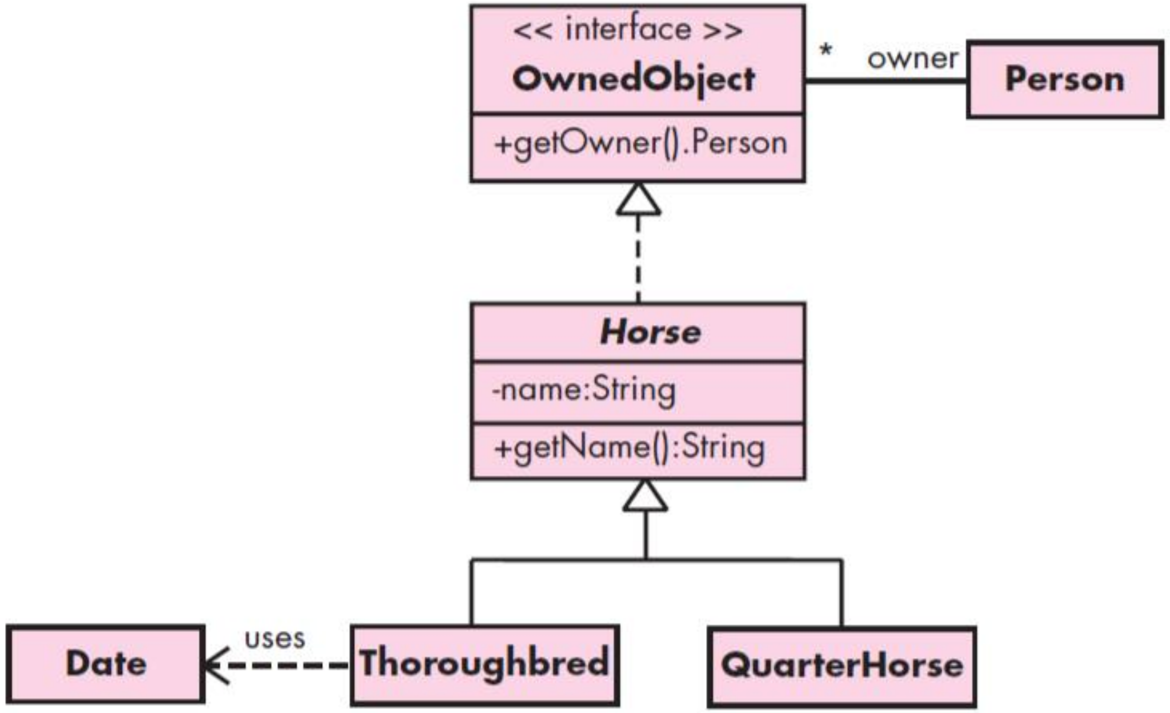
- To model classes, including their attributes, operations, and their relationships and associations with other classes, UML provides a class diagram.
- A class diagram provides a static or structural view of a system.
- It does not show the dynamic nature of the communications between the objects of the classes in the diagram.

- The main elements of a class diagram are boxes, which are the icons used to represent classes and interfaces.
- Each box is divided into horizontal parts.
- The top part contains the name of the class.
- The middle section lists the attributes of the class.
- An attribute refers to something that an object of that class knows or can provide all the time.
- Attributes are usually implemented as fields of the class, but they need not be.
- The third section of the class diagram contains the operations or behaviors of the class.
- An operation refers to what objects of the class can do. It is usually implemented as a method of the class

- Below Figure presents a simple example of a Thoroughbred class that models thoroughbred horses.
- It has three attributes displayed—mother, father, and birthyear.
- The diagram also shows three operations: getCurrentAge(), getFather(), and getMother().
- Each attribute can have a name, a type, and a level of visibility.
- The visibility is indicated by a preceding –, #, ~, or +, indicating, respectively, private, protected, package, or public visibility.



- An abstract class or abstract method is indicated by the use of italics for the name in the class diagram.
- See the Horse class in below Figure for an example.
- An interface is indicated by adding the phrase “«interface»” (called a stereotype) above the name.
- See the OwnedObject interface in below Figure.
- An interface can also be represented graphically by a hollow circle.



- Class diagrams can also show relationships between classes.

Generalization:

- A class that is a subclass of another class is connected to it by an arrow with a solid line for its shaft and with a triangular hollow arrowhead.
- The arrow points from the subclass to the superclass.
- In UML, such a relationship is called a generalization.
- For example, in above Figure , the Thoroughbred and QuarterHorse classes are shown to be subclasses of the Horse abstract class.

Realization:

- An arrow with a dashed line for the arrow shaft indicates implementation of an interface.
- In UML, such a relationship is called a realization.
- For example, in above Figure, the Horse class implements or realizes the OwnedObject interface.

• Association:

- An association between two classes means that there is a structural relationship between them.
- Associations are represented by solid lines.
- An association has many optional parts. It can be labeled, as can each of its ends, to indicate the role of each class in the association.
- For example, in above Figure, there is an association between OwnedObject and Person in which the Person plays the role of owner.
- Arrows on either or both ends of an association line indicate navigability.
- Also, each end of the association line can have a multiplicity value displayed.
- An association with an arrow at one end indicates one-way navigability.
- The arrow means that from one class you can easily access the second associated class to which the association points, but from the second class, you cannot necessarily easily access the first class.
- An association with no arrows usually indicates a two-way association.

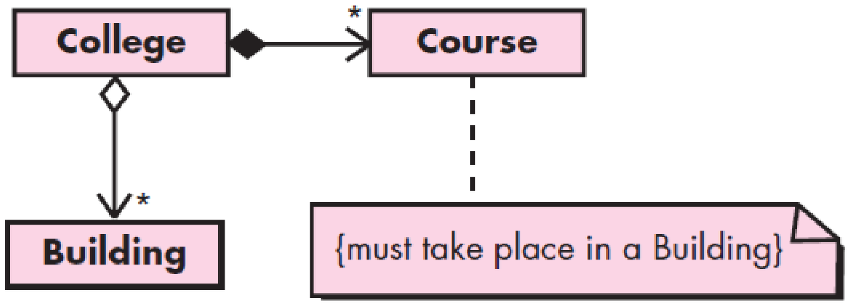
- The multiplicity of one end of an association means the number of objects of that class associated with the other class.
- A multiplicity is specified by a nonnegative integer or by a range of integers.
- A multiplicity specified by “0..1” means that there are 0 or 1 objects on that end of the association.
- A multiplicity specified by “1..*” means one or more, and a multiplicity specified by “0..*” or just “*“ means zero or more.

- ## Aggregation:

- An aggregation is a special kind of association indicated by a hollow diamond on one end of the icon.
- It indicates a “whole/part” relationship, in that the class to which the arrow points is considered a “part” of the class at the diamond end of the association.

- ## Composition:

- A composition is an aggregation indicating strong ownership of the parts.
- In a composition, the parts live and die with the owner because they have no role in the software system independent of the owner.
- See below Figure for examples of aggregation and composition.
- Another common element of a class diagram is a note. It can have arbitrary content (text and graphics) and is similar to comments in programming languages.



USE-CASE DIAGRAMS

• What is Use-Case?

- Use cases and the UML use-case diagram help you determine the functionality and features of the software from the user's perspective.
- It specify how the users will interact or behave with the system or software.

- To give you a feeling for how use cases and use-case diagrams work, lets create some for a software application for managing digital music files, similar to Apple's iTunes software. Some of the things the software might do include:

➢ Download an MP3 music file and store it in the application's library.

➢ Capture streaming music and store it in the application's library.

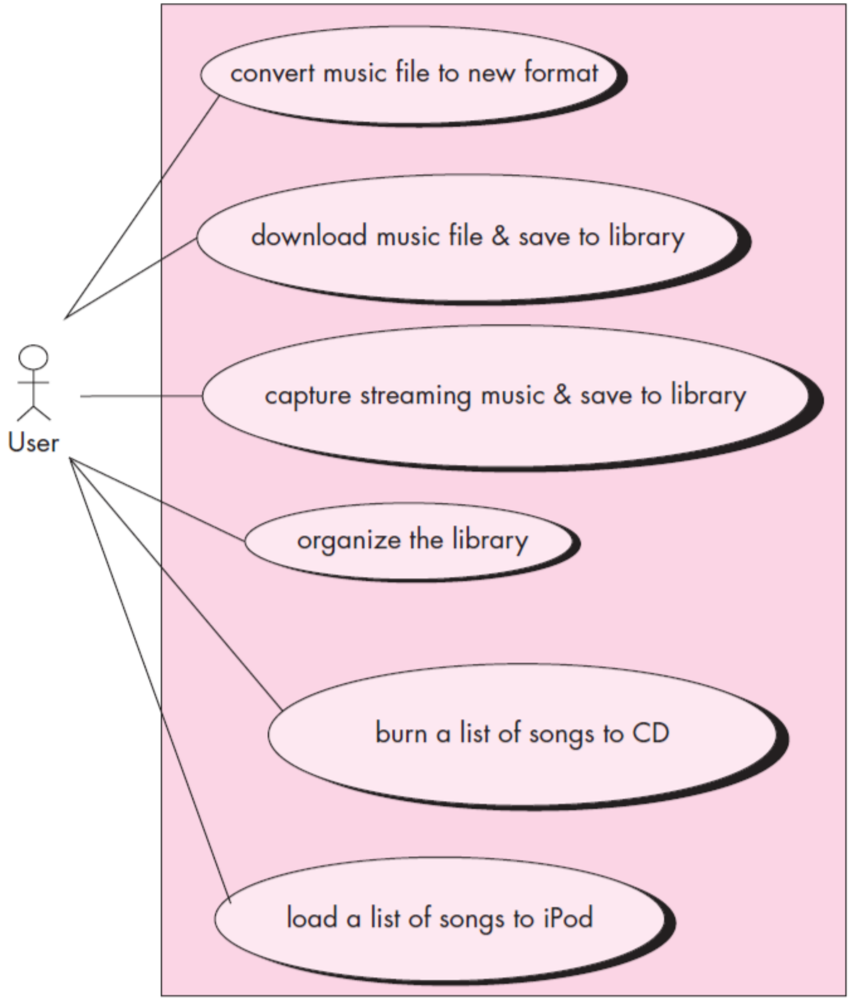
➢ Manage the application's library (e.g., delete songs or organize them in playlists).

➢ Burn a list of the songs in the library onto a CD.

➢ Load a list of the songs in the library onto an iPod or MP3 player.

➢ Convert a song from MP3 format to AAC format and vice versa.

- In this diagram, the stick figure represents an actor that is associated with one category of user (or other interaction element).
- Complex systems typically have more than one actor.
- In the use-case diagram, the use cases are displayed as ovals. The actors are connected by lines to the use cases that they carry out.



Developing Use-Cases:

- In essence, a use case tells a stylized story about how an end user (playing one of a number of possible roles) interacts with the system under a specific set of circumstances.
- The story may be narrative text, an outline of tasks or interactions, a template-based description, or a diagrammatic representation.
- Regardless of its form, a use case depicts the software or system from the end user's point of view.
- The first step in writing a use case is to define the set of "actors" that will be involved in the story.

- Actors:

- Actors are the different people (or devices) that use the system or product within the context of the function and behavior that is to be described.
- Actors represent the roles that people (or devices) play as the system operates.
- An actor is anything that communicates with the system or product and that is external to the system itself.
- Every actor has one or more goals when using the system.

- ## Primary Actors:

- Primary actors interact to achieve required system function and derive the intended benefit from the system. They work directly and frequently with the software.

- ## Secondary Actors:

- Secondary actors support the system so that primary actors can do their work.

- Once actors have been identified, use cases can be developed. Jacobson suggests a number of questions that should be answered by a use case:

- Who is the primary actor, the secondary actor(s)?
- What are the actor’s goals?
- What preconditions should exist before the story begins?
- What main tasks or functions are performed by the actor?
- What exceptions might be considered as the story is described?
- What variations in the actor’s interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

- Recalling basic SafeHome requirements, we define four actors:

1) homeowner (a user)
2) setup manager (likely the same person as homeowner, but playing a different role)
3) sensors (devices attached to the system)
4) monitoring and response subsystem (the central station that monitors the SafeHome home security function).

- For the purposes of this example, we consider only the homeowner actor.
- The homeowner actor interacts with the home security function in a number of different ways using either the alarm control panel or a PC:

➢ Enters a password to allow all other interactions.

➢ Inquires about the status of a security zone.

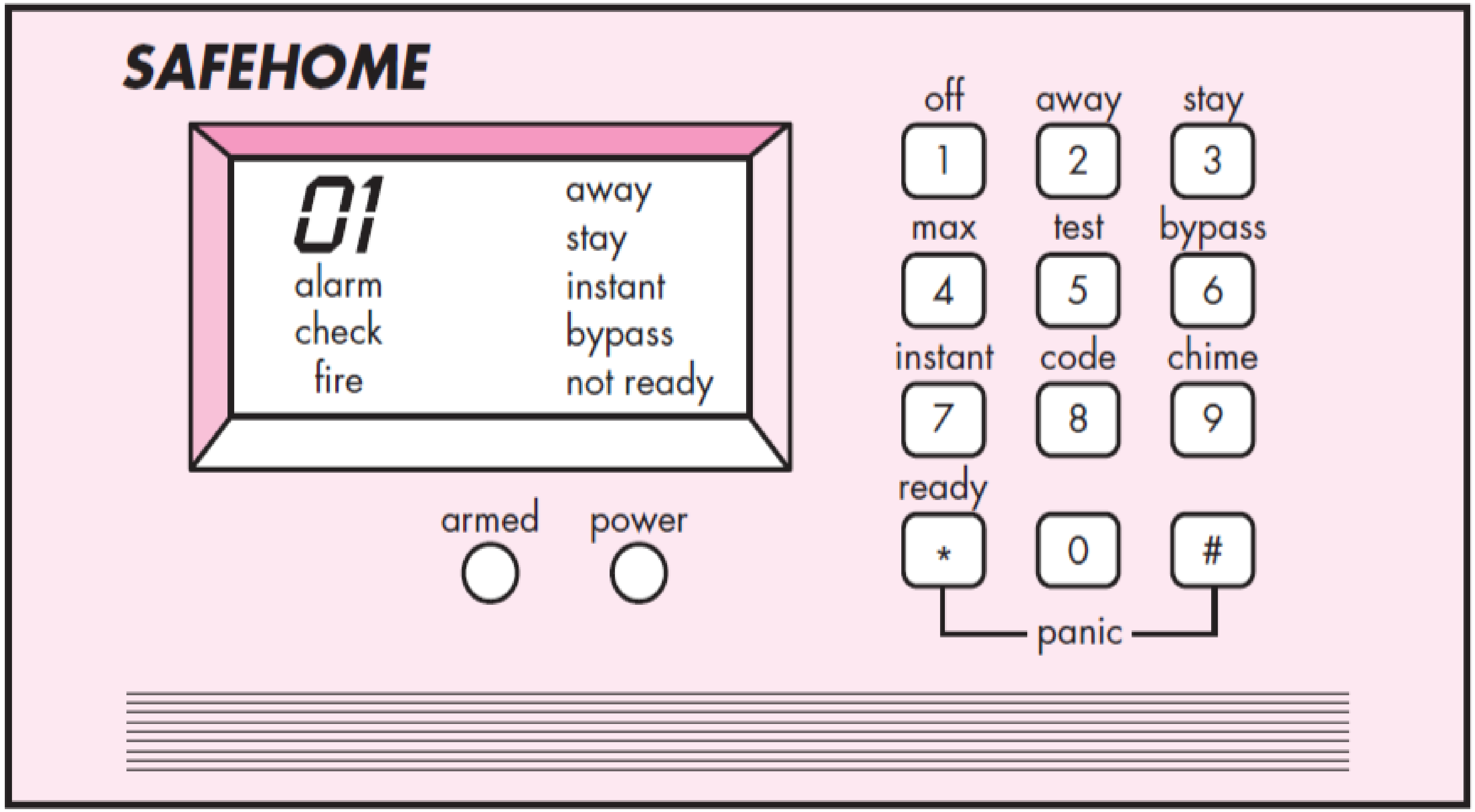
➢ Inquires about the status of a sensor.

➢ Presses the panic button in an emergency.

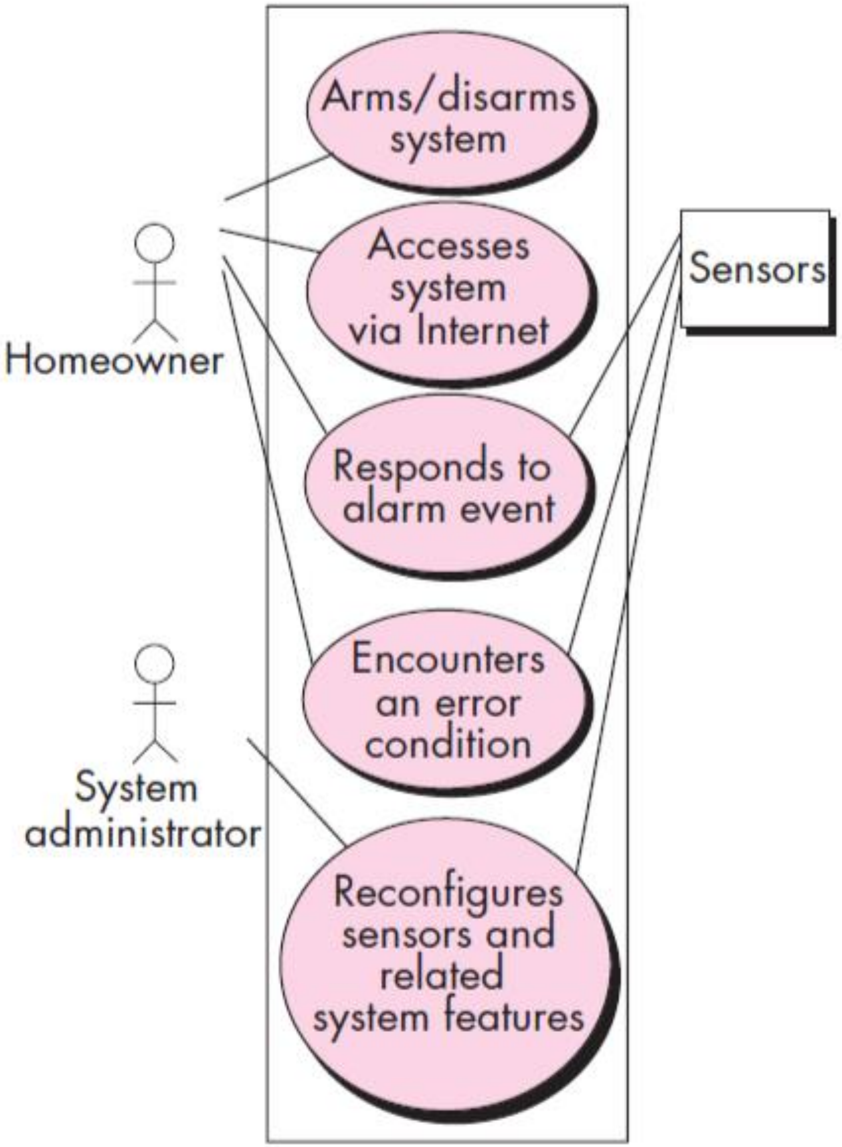
➢ Activates/deactivates the security system.

- Considering the situation in which the homeowner uses the control panel, the basic use case for system activation follows:

1) The homeowner observes the SafeHome control panel to determine if the system is ready for input. If the system is not ready, a not ready message is displayed on the LCD display, and the homeowner must physically close windows or doors so that the not ready message disappears. [A not ready message implies that a sensor is open; i.e., that a door or window is open.]
2) The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.
3) The homeowner selects and keys in stay or away (see below Figure) to activate the system. Stay activates only perimeter sensors (inside motion detecting sensors are deactivated). Away activates all sensors.
4) When activation occurs, a red alarm light can be observed by the homeowner.


SAFEHOME
01
alarm
check
fire
away
stay
instant
bypass
not ready
armed
power
off
away
stay
1
2
3
max
test
bypass
4
5
6
instant
code
chime
7
8
9
ready
*
0
#
panic

- Below is the UML Use-Case Diagram for SafeHome Home Security Function:



- The basic use case presents a high-level story that describes the interaction between the actor and the system.
- In many instances, uses cases are further elaborated to provide considerably more detail about the interaction. For example, Cockburn suggests the following template for detailed descriptions of use cases:

- **Use case**: InitiateMonitoring
- **Primary actor**: Homeowner.
- **Goal in context**: To set the system to monitor sensors when the homeowner leaves the house or remains inside.
- **Preconditions**: System has been programmed for a password and to recognize various sensors.
- **Trigger**: The homeowner decides to "set" the system, i.e., to turn on the alarm functions.

Scenario:

1. Homeowner: observes control panel
2. Homeowner: enters password
3. Homeowner: selects "stay" or "away"
4. Homeowner: observes red alarm light to indicate that SafeHome has been armed.

Exceptions:

1. Control panel is not ready: homeowner checks all sensors to determine which are open; closes them.
2. Password is incorrect (control panel beeps once): homeowner reenters correct password.
3. Password not recognized: monitoring and response subsystem must be contacted to reprogram password.
4. Stay is selected: control panel beeps twice and a stay light is lit; perimeter sensors are activated.
5. Away is selected: control panel beeps three times and an away light is lit; all sensors are activated.

Priority: Essential, must be implemented

When available: First increment

Frequency of use: Many times per day

Channel to actor: Via control panel interface

Secondary actors: Support technician, sensors

Channels to secondary actors:

Support technician: phone line

Sensors: hardwired and radio frequency interfaces

Open issues:

1. Should there be a way to activate the system without the use of a password or with an abbreviated password?
2. Should the control panel display additional text messages?
3. How much time does the homeowner have to enter the password from the time the first key is pressed?
4. Is there a way to deactivate the system before it actually activates?

Use cases for other homeowner interactions would be developed in a similar manner. It is important to review each use case with care. If some element of the interaction is ambiguous, it is likely that a review of the use case will indicate a problem.

Developing a High-Level Use-Case Diagram

The scene: A meeting room, continuing the requirements gathering meeting

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator: We've spent a fair amount of time talking about *SafeHome* home security functionality. During the break I sketched a use case diagram to summarize the important scenarios that are part of this function. Take a look.

(All attendees look at Figure 5.2.)

Jamie: I'm just beginning to learn UML notation.[14] So the home security function is represented by the big box with the ovals inside it? And the ovals represent use cases that we've written in text?

Facilitator: Yep. And the stick figures represent actors—the people or things that interact with the system as described by the use case . . . oh, I use the labeled square to represent an actor that's not a person . . . in this case, sensors.

Doug: Is that legal in UML?

Facilitator: Legality isn't the issue. The point is to communicate information. I view the use of a humanlike stick figure for representing a device to be misleading. So I've adapted things a bit. I don't think it creates a problem.

Vinod: Okay, so we have use-case narratives for each of the ovals. Do we need to develop the more detailed template-based narratives I've read about?

Facilitator: Probably, but that can wait until we've considered other *SafeHome* functions.

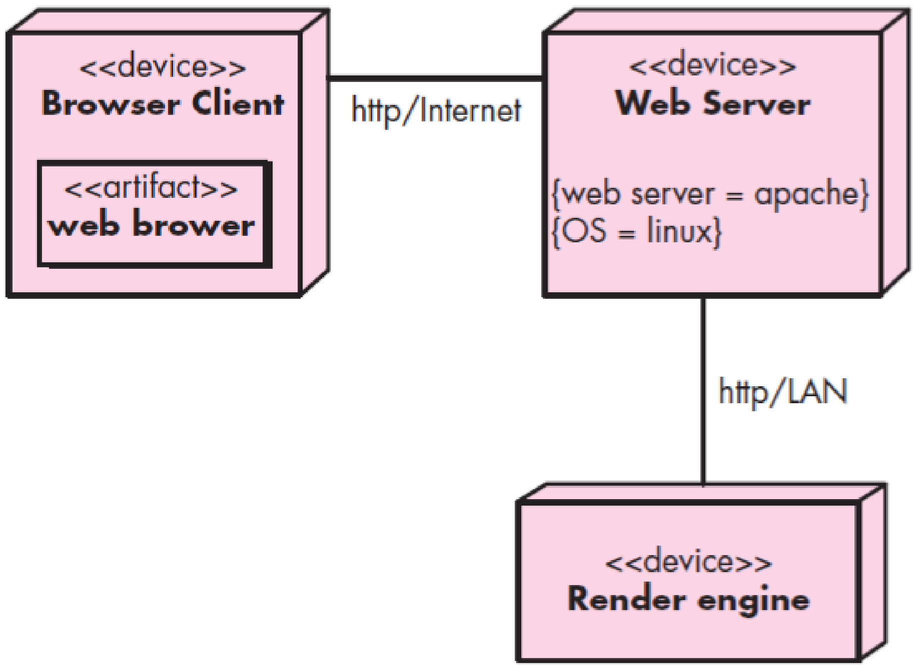
Marketing person: Wait, I've been looking at this diagram and all of a sudden I realize we missed something.

Facilitator: Oh really. Tell me what we've missed.

(The meeting continues.)

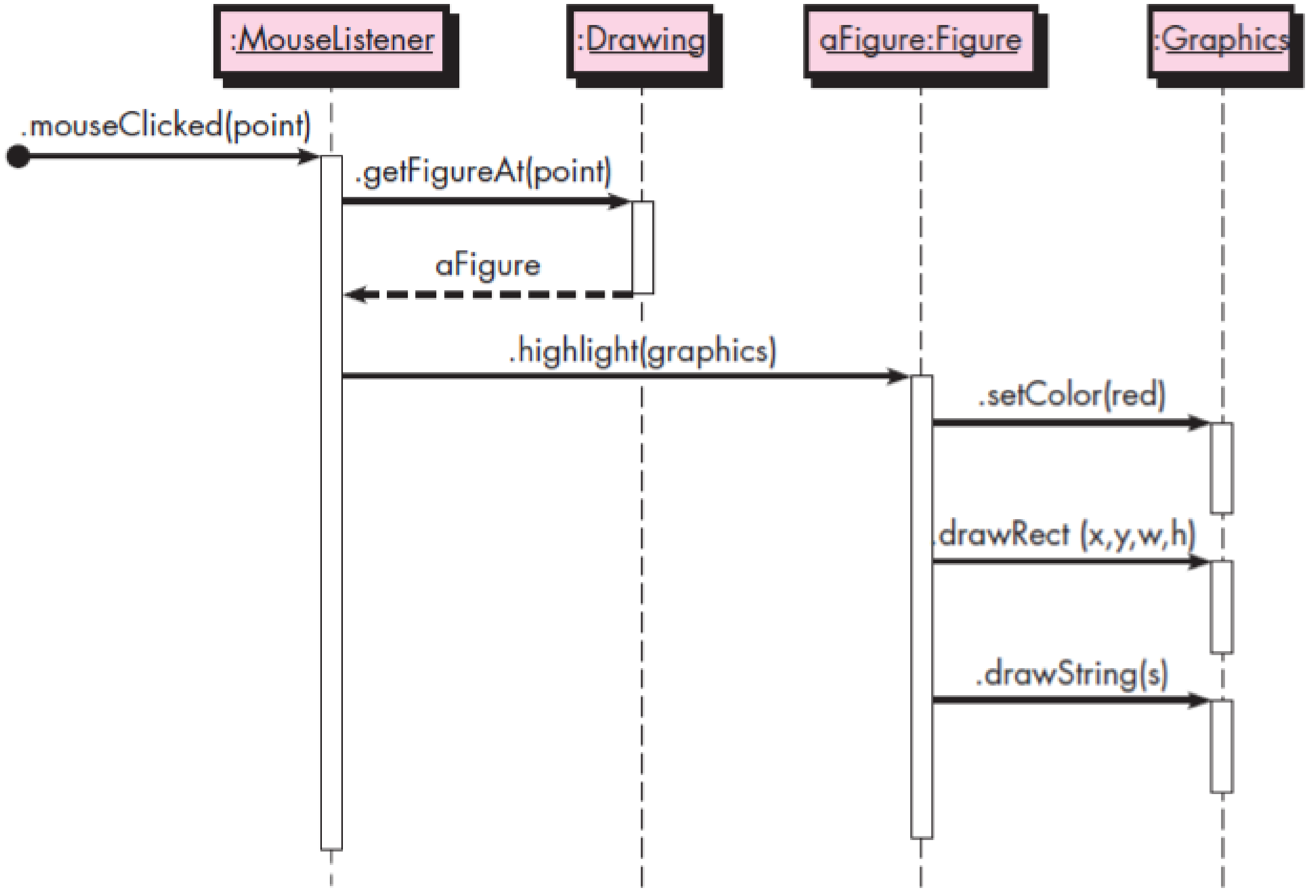
DEPLOYMENT DIAGRAMS

- A UML deployment diagram focuses on the structure of a software system and is useful for showing the physical distribution of a software system among hardware platforms and execution environments.
- For example, you are developing a Web-based graphics-rendering package. Users of your package will use their Web browser to go to your website and enter rendering information. Your website would render a graphical image according to the user's specification and send it back to the user. Because graphics rendering can be computationally expensive, you decide to move the rendering itself off the Web server and onto a separate platform. Therefore, there will be three hardware devices involved in your system: the Web client (the users' computer running a browser), the computer hosting the Web server, and the computer hosting the rendering engine.



SEQUENCE DIAGRAMS

- In contrast to class diagrams and deployment diagrams, which show the static structure of a software component, a **sequence diagram** is used to show the dynamic communications between objects during execution of a task.
- It shows the temporal order in which messages are sent between the objects to accomplish that task.
- One might use a sequence diagram to show the interactions in one use case or in one scenario of a software system.



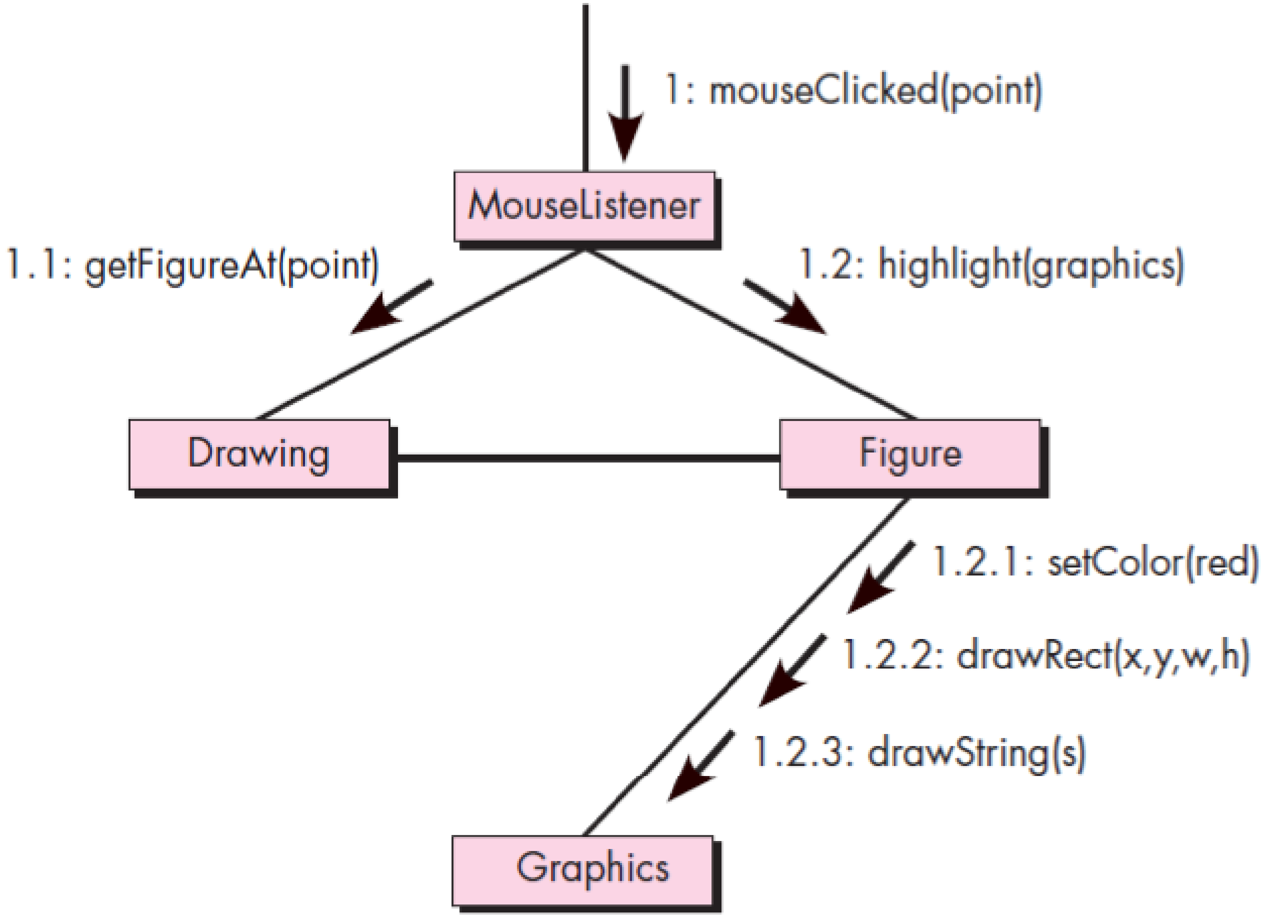
- In above Figure, you see a sequence diagram for a drawing program.
- The diagram shows the steps involved in highlighting a figure in a drawing when it has been clicked.
- Each box in the row at the top of the diagram usually corresponds to an object, although it is possible to have the boxes model other things, such as classes.
- If the box represents an object (as is the case in all our examples), then inside the box you can optionally state the type of the object preceded by the colon.
- You can also precede the colon and type by a name for the object, as shown in the third box in Figure.
- Below each box there is a dashed line called the lifeline of the object.
- The vertical axis in the sequence diagram corresponds to time, with time increasing as you move downward.

- A sequence diagram shows method calls using horizontal arrows from the caller to the callee, labeled with the method name and optionally including its parameters, their types, and the return type.
- For example, in Figure, the MouseListener calls the Drawing's getFigureAt() method.
- When an object is executing a method (that is, when it has an activation frame on the stack), you can optionally display a white bar, called an activation bar, down the object's lifeline.
- In Figure, activation bars are drawn for all method calls.
- The diagram can also optionally show the return from a method call with a dashed arrow and an optional label.
- A black circle with an arrow coming from it indicates a found message whose source is unknown or irrelevant.

- You should now be able to understand the task that above Figure is displaying.
- An unknown source calls the mouseClicked() method of a MouseListener, passing in the point where the click occurred as the argument.
- The MouseListener in turn calls the getFigureAt() method of a Drawing, which returns a Figure.
- The MouseListener then calls the highlight method of Figure, passing in a Graphics object as an argument.
- In response, Figure calls three methods of the Graphics object to draw the figure in red.

COMMUNICATION DIAGRAMS OR COLLABORATION DIAGRAMS

- The UML communication diagram (called a “collaboration diagram” in UML) provides another indication of the temporal order of the communications but emphasizes the relationships among the objects and classes instead of the temporal order.
- A communication diagram, illustrated in below Figure, displays the same actions shown in the sequence diagram.



- In a communication diagram the interacting objects are represented by rectangles.
- Associations between objects are represented by lines connecting the rectangles.
- here is typically an incoming arrow to one object in the diagram that starts the sequence of message passing.
- That arrow is labeled with a number and a message name.
- If the incoming message is labeled with the number 1 and if it causes the receiving object to invoke other messages on other objects, then those messages are represented by arrows from the sender to the receiver along an association line and are given numbers 1.1, 1.2, and so forth, in the order they are called.
- If those messages in turn invoke other messages, another decimal point and number are added to the number labeling these messages, to indicate further nesting of the message passing.

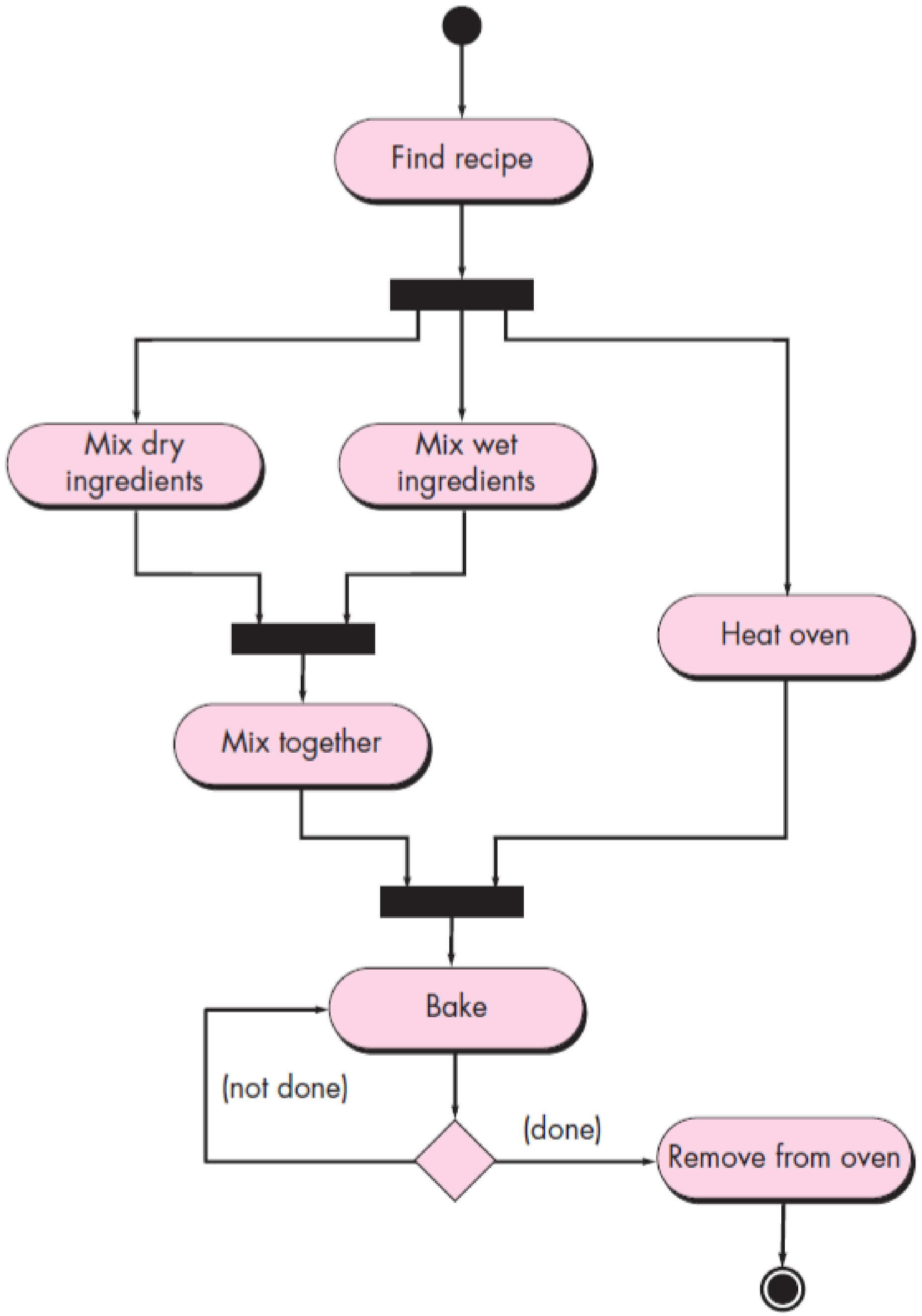
- In Figure, you see that the mouseClicked message invokes the methods getFigureAt() and then highlight().
- The highlight() message invokes three other messages: setColor(), drawRect(), and drawstring().
- The numbering in each label shows the nesting as well as the sequential nature of each message.

ACTIVITY DIAGRAMS

- A UML Activity Diagram depicts the dynamic behavior of a system or part of a system through the flow of control between actions that the system performs.
- It is similar to a flowchart except that an activity diagram can show concurrent flows.

- The main component of an activity diagram is an action node, represented by a rounded rectangle, which corresponds to a task performed by the software system.
- Arrows from one action node to another indicate the flow of control.
- That is, an arrow between two action nodes means that after the first action is complete the second action begins.
- A solid black dot forms the initial node that indicates the starting point of the activity.
- A black dot surrounded by a black circle is the final node indicating the end of the activity.

- Below Figure shows a UML Activity Diagram showing how to bake a cake.


Find recipe
Mix dry ingredients
Mix wet ingredients
Heat oven
Mix together
Bake
(not done)
(done)
Remove from oven

- Above Figure shows a sample activity diagram involving baking a cake.
- The first step is finding the recipe.
- Once the recipe has been found, the dry ingredients and wet ingredients can be measured and mixed and the oven can be preheated.
- The mixing of the dry ingredients can be done in parallel with the mixing of the wet ingredients and the preheating of the oven.

Fork:

- A fork represents the separation of activities into two or more concurrent activities.
- It is drawn as a horizontal black bar with one arrow pointing to it and two or more arrows pointing out from it.
- Each outgoing arrow represents a flow of control that can be executed concurrently with the flows corresponding to the other outgoing arrows.
- These concurrent activities can be performed on a computer using different threads or even using different computers.

Join:

- A join is a way of synchronizing concurrent flows of control.
- It is represented by a horizontal black bar with two or more incoming arrows and one outgoing arrow.
- The flow of control represented by the outgoing arrow cannot begin execution until all flows represented by incoming arrows have been completed.
- In Figure, we have a join before the action of mixing together the wet and dry ingredients.
- This join indicates that all dry ingredients must be mixed and all wet ingredients must be mixed before the two mixtures can be combined.
- The second join in the figure indicates that, before the baking of the cake can begin, all ingredients must be mixed together and the oven must be at the right temperature.

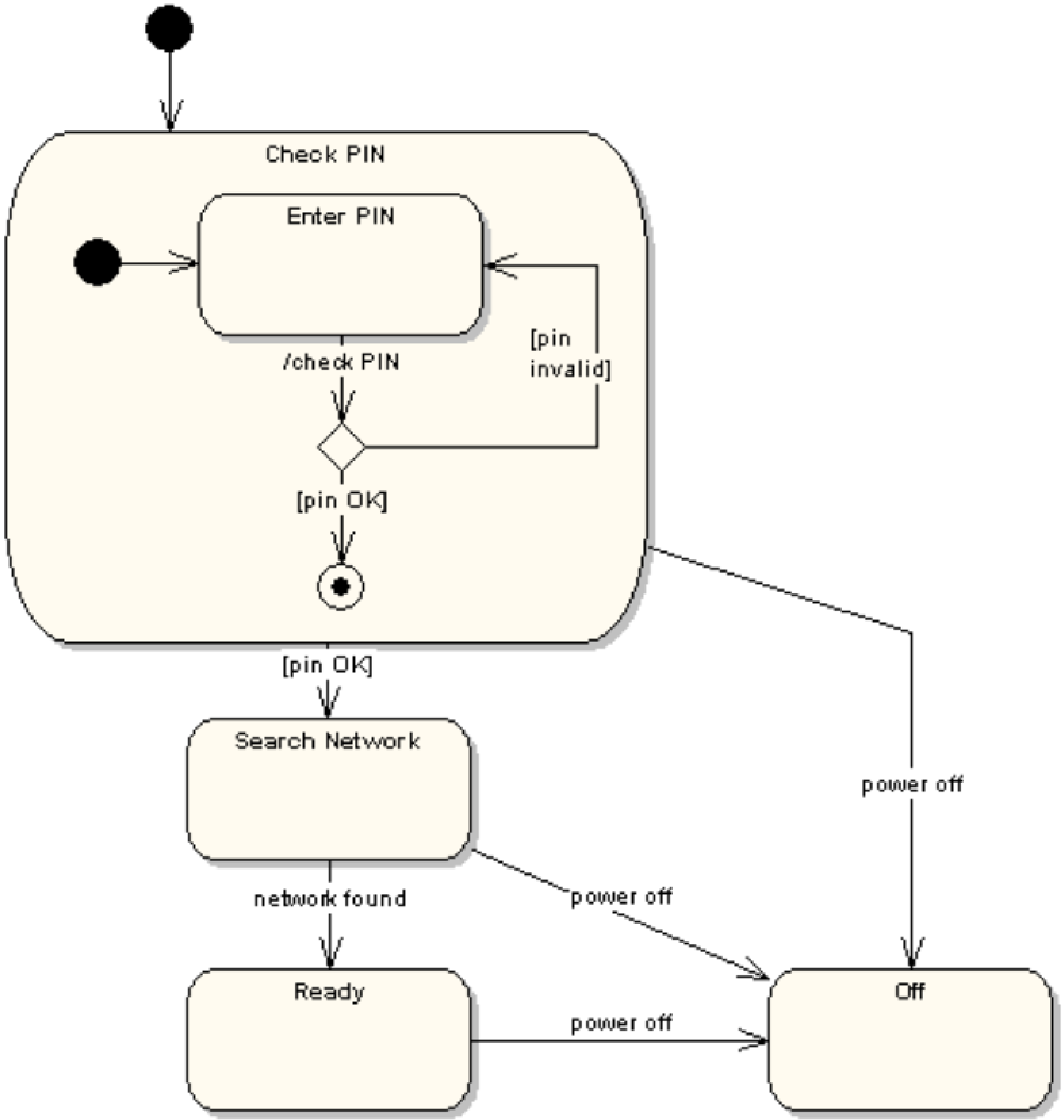
Decision:

- A decision node corresponds to a branch in the flow of control based on a condition.
- Such a node is displayed as a white triangle with an incoming arrow and two or more outgoing arrows.
- Each outgoing arrow is labeled with a guard (a condition inside square brackets).
- The flow of control follows the outgoing arrow whose guard is true.
- It is advisable to make sure that the conditions cover all possibilities so that exactly one of them is true every time a decision node is reached.
- Figure shows a decision node following the baking of the cake.
- If the cake is done, then it is removed from the oven.
- Otherwise, it is baked for a while longer.

STATE DIAGRAMS

- The behavior of an object at a particular point in time often depends on the state of the object, that is, the values of its variables at that time.
- As a trivial example, consider an object with a Boolean instance variable.
- When asked to perform an operation, the object might do one thing if that variable is true and do something else if it is false.

- A UML State Diagram models an object's states, the actions that are performed depending on those states, and the transitions between the states of the object.


Check PIN
Enter PIN
/check PIN
[pin
invalid]
[pin OK]
[pin OK]
Search Network
power off
network found
power off
Ready
power off
Off