SOFTWARE DESIGN

Software Architecture

× Software architecture is commonly used to describe the organization of software systems.
× An architecture is the definition of the key elements that constitute a system, their relationships, and any constraints.
× Generally speaking, architectures of software-based systems can be classified into several categories.

Software Architectures Categories:

Business Architecture

→ Described the structure of the business tasks performed by the organization
→ These are created by business analysts

Physical Architecture

→ Describes the structure of the hardware platform(s) or network(s) on which the system will operate
→ Physical architecture of a system

Logical Architecture

→ Describes the structure of the business and application objects
→ Is often part of an object-oriented view of a system
→ Is often created by an object analyst

Functional Architecture

→ Describes the structure of the potential use cases and the requirements from a user's point of view
→ Is often part of an object-oriented view of a system

Software Architecture

→ Describe the structure of the system into layers, such as OSI (Open systems interconnection) or layered architecture of UNIX operating system.
→ Decomposed to client and server

Technical Architecture

→ Describes the structure of the major interfaces in the software architecture
→ Elements include:
- Application programming interfaces (APIs)
- Middleware
- Database management systems
- Graphical User Interfaces
- Other glueware or bridgeware needed to interface components

→ Decisions to use COBRA, DCOM, Java RMI, or RPC or even cloud computing, with a protocol such as HTTP, SOAP, REST, or JSON, would be reflected in this type of system architecture

System Architecture

→ Describes the structure of the business, application, and technical objects and their relationships.
→ The description is often created by a systems analysts

Deployment Architecture

→ Describes the structure of the mapping of the system and technical architectures onto the physical architecture
→ It includes two views:
- A static view of the basic files that will be necessary
- A dynamic view of the concurrent processes and threads, and the mechanisms for their synchronization and communication.

→ A thread on a particular computer or to have an autonomous

Deployment Architecture

→ A thread on a particular computer or to have an autonomous agent performing

computation would be reflected in this type of architecture.

Software Architecture Importance

- Software architecture are an enabler for communication between stakeholders
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows

ARCHITECTURAL DESIGN

- The software must be placed into context
- The design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction
- A set of architectural archetypes should be identified
- An archetype is an abstraction (similar to a class) that represents one element of system behavior.
- The designer specifies the structure of the system by defining and refining software components that implement each archetype

DATA DESIGN (Data Modeling Concepts)

At the architectural level…

- Design databases to support the application architecture
- Design of methods for 'mining' the content of multiple databases
- Navigate through existing databases to extract information.
- Design of a data warehouse

ENTITY RELATIONSHIP DIAGRAM

Entity-Relationship Diagram (ERD) represents all data objects entered, stored, transformed, and produced within an application

Entity or Entity Types

- An ERD is consists of entities or data objects, data attributes and relationships with the entities.
- An **entity** or an **entity type** or a **data object** is a representation of something that has several different properties (attributes) that must be understood by software. It is anything that produce or consumes information.
- Entities:
  - **Entity instance** – person, place, object, event, concept (often corresponds to a row in a table)
  - **Entity Type** – collection of entities (often corresponds to a table)
- Example of entity or entity types
- **Attribute** – property or characteristic of an entity or relationship type (often corresponds to a field in a table)

Relationships

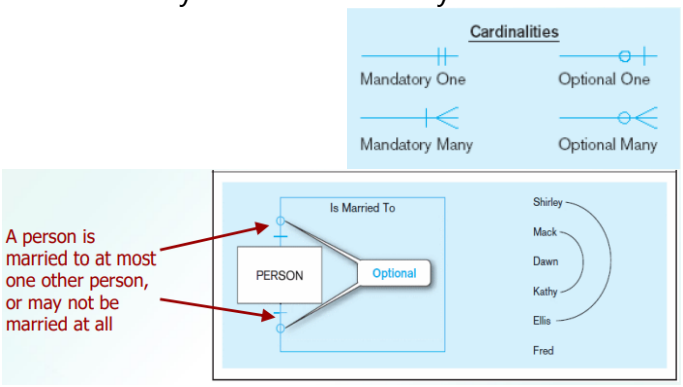
- **Relationship instance** – link between entities (corresponds to primary key-foreign key equivalencies in related tables)
- **Relationship Type** – category of relationship…link between entity types

RELATIONSHIPS

- Relationships can have attributes
- Two entities can have more than one type of relationship between them (multiple relationships)
- **Associative Entity** – combination of relationship and entity

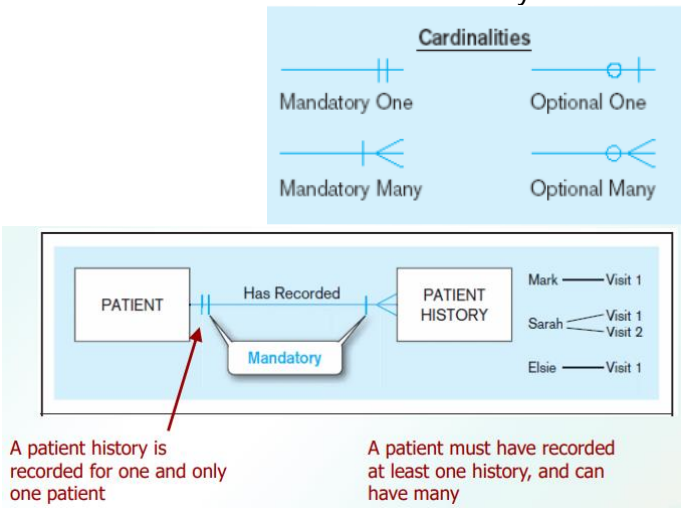
One-to-One (1:1)

→ Each entity in the relationship will have exactly one related entity



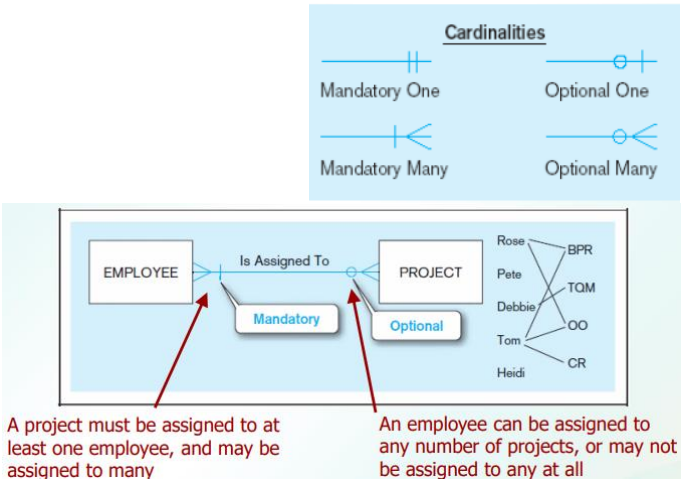
One-to-Many (1:M) or Many-to-One (M:1)

→ An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity.



Many-to-Many (M:M)

→ Entities on both sides of the relationship can have many related entities on the other side

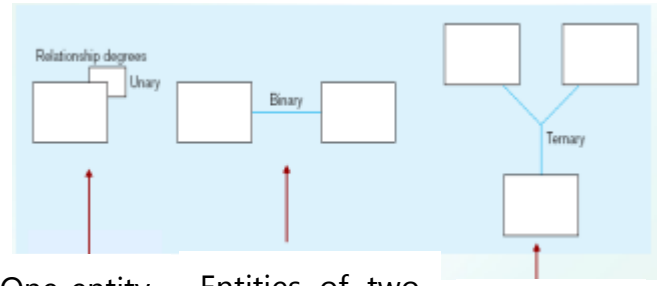


RELATIONSHIPS

× Degree of a relationship is the number of entity types that participate in it.

- Unary Relationship
- Binary Relationship
- Ternary Relationship

Unary Relationship

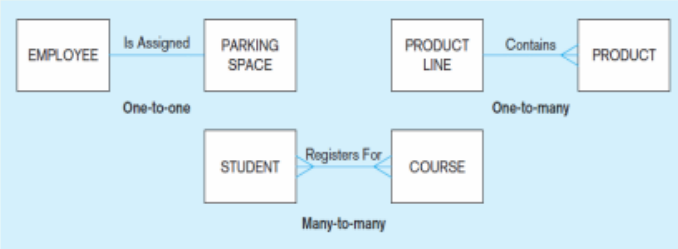


One entity related to another of the same entity type

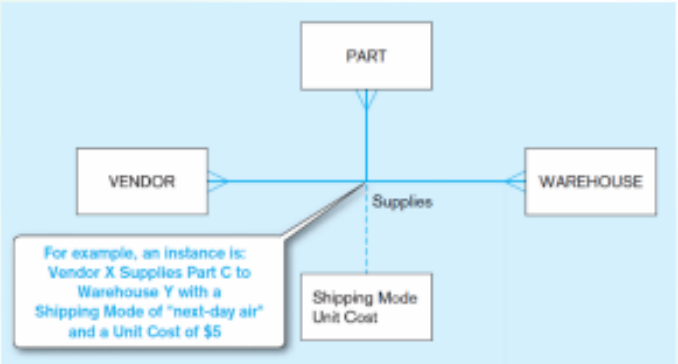
Entities of two different types related to each other

Entities of three different types related to each other

Binary Relationship



Ternary Relationship



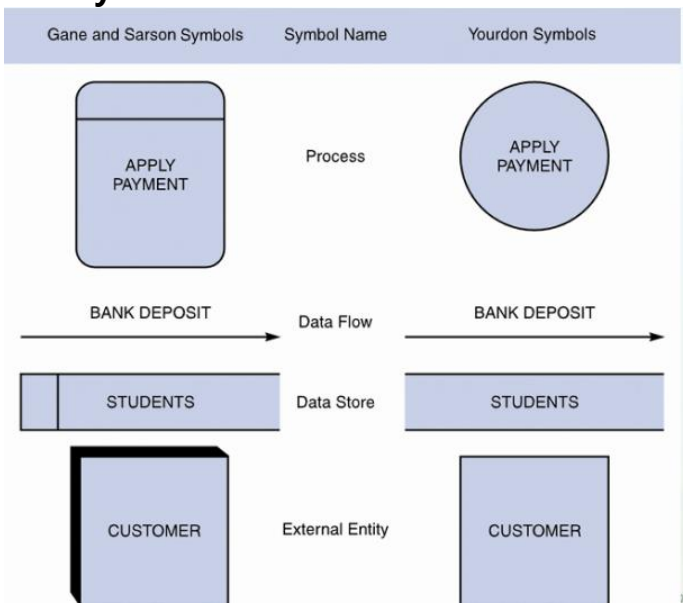
STRUCTURED DESIGN

- A mapping technique, called **structured design** is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture.
- The transition from information flow (represented as a DFD) to program structure is accomplished as part of a six step process.
- These six step processes are:
  1) the type of information flow is established,
  2) flow boundaries are indicated,
  3) the DFD is mapped into the program structure,
  4) control hierarchy is defined,
  5) the resultant structure is refined using design measures and heuristics, and
  6) the architectural description is refined and elaborated.

Data Flow Diagram

- A **Data Flow Diagram (DFD)** is a graphical representation of the "flow" of data through an information modeling its process aspects.
- A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored.
- The DFD takes an input-process-output view of a system. Data objects are represented by labeled arrows, and
- Transformations are represented by circles (also called bubbles).
- The DFD is presented in a hierarchical fashion.
- Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

DFD Symbols



Creating a Set of DFD

Here are simple guidelines

1. the level 0 data flow diagram should depict the software/system as a single bubble;
2. primary input and output should be carefully noted;

3. Refinement should begin by isolating candidate processes, data objects, and data stores to be represented at the next level;
4. all arrows and bubbles should be labeled with meaningful names;
5. information flow continuity must be maintained from level to level, and
6. one bubble at a time should be refined

Before we proceed to diagram 0, lets discuss what SafeHome does

- 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…

UNIFIED MODELING LANGUAGE

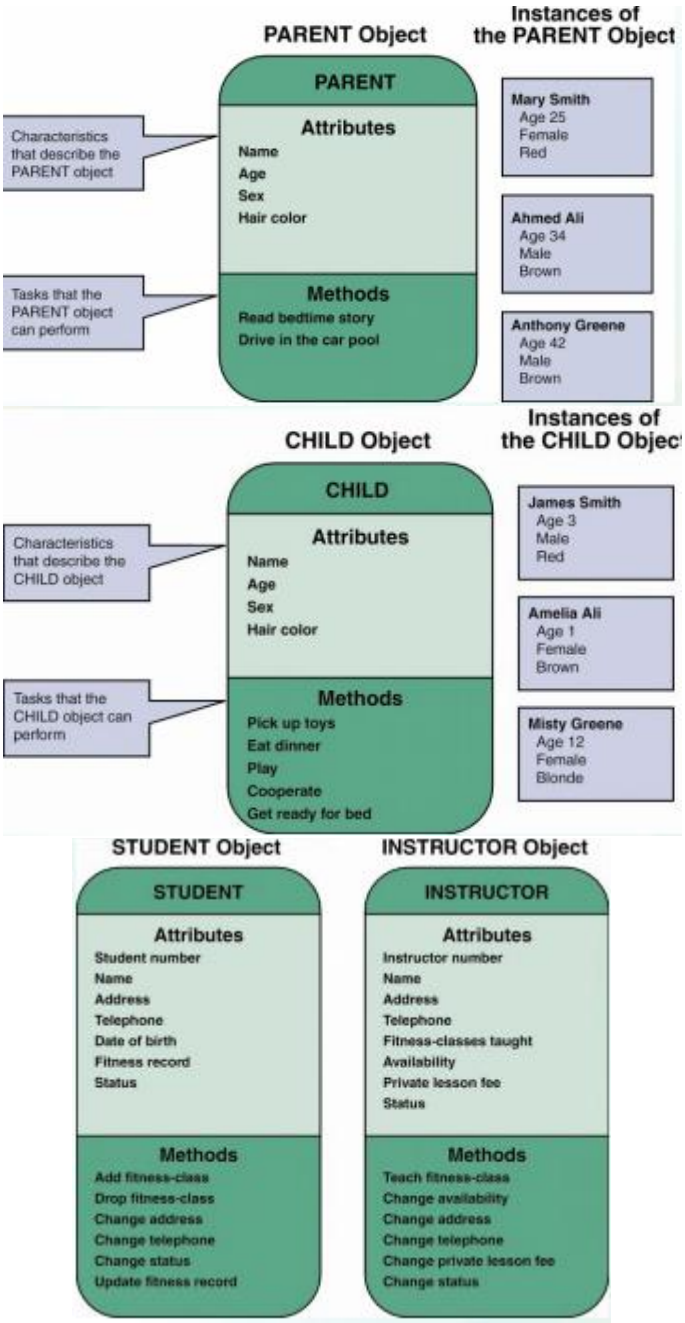
- Unified Modeling Language (UML) is "a standard language for writing blueprints. UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system"
- Just as building architects create blueprints
- Software architects create UML diagrams

Two main types of diagrams in the UML

1. **Structural Diagram** – are used to describe the relationship between classes
2. **Behavioral Diagram** – can be used to describe the interaction between people and the thing we refer to as a use case, or how the actors use the system.

- It all begins with the construction of a model.
- A **model** is an abstraction of the underlying problem. The **domain** is the actual world from which the problem comes.
- Models consist of **objects** that interact by sending each other **messages**. Think of an object as "alive." Objects have things they know (**attributes**) and things they can do (**behaviors** or **operations**). The values of an object's attributes determine its **state**.
- **Classes** are the "blueprints" for objects. A class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity. Objects are **instances** of classes.

OBJECTS CONCEPTS



Attributes

- If objects are similar to nouns, attributes are similar to adjectives that describe the characteristics of an object
- Some objects might have a few attributes; others might have dozens state

Methods

- A method defines specific tasks that an object can perform
- Just as objects are similar to nouns and attributes are similar to adjectives, methods resemble verbs that describe what and how an object does something

USE CASE

USE CASE DIAGRAM

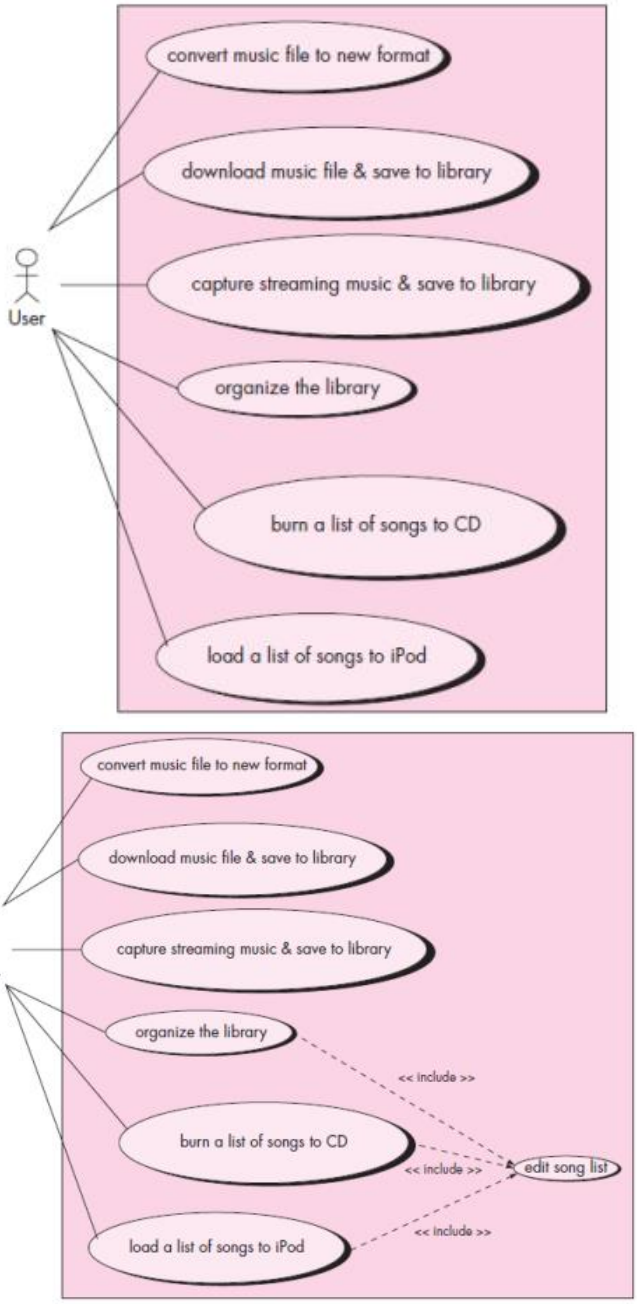
- **Use case diagrams** describe what a system does from the standpoint of an external observer. The emphasis is on what a system does rather than how.
- Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system. Here is a scenario for a managing user digital files.

Use Case Diagram Example

- Let us say, there is 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.
- A use case describes how a user interacts with the system by defining the steps

required to accomplish a specific goal (e.g., burning a list of songs onto a CD).

× A use case provides a big picture of the functionality of the system.



SEQUENCE DIAGRAM

× 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.

× Sequence diagrams can be used to show the interactions in one use case or in one scenario of the software system.

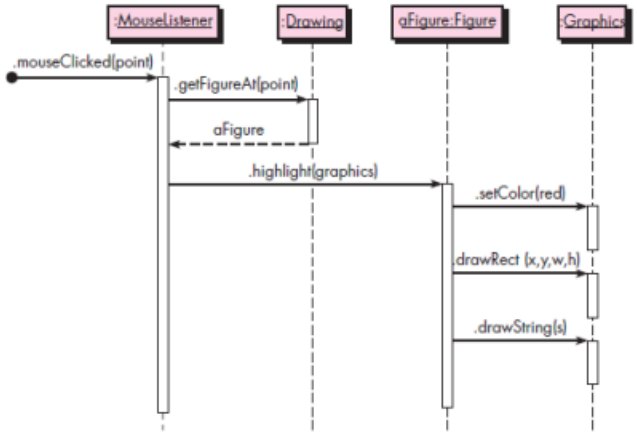


Figure 2.19
Example of sequence diagram of a drawing program

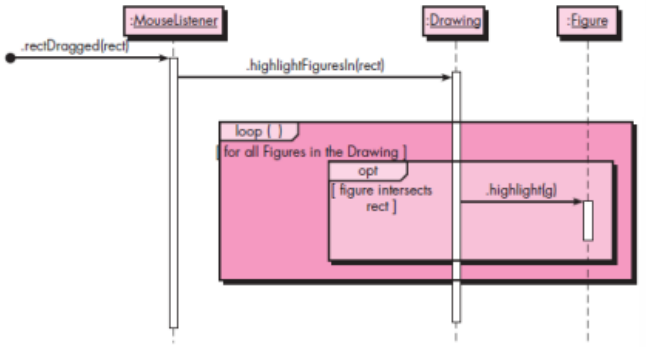


Figure 2.20
Example of sequence diagram of a drawing program with loops

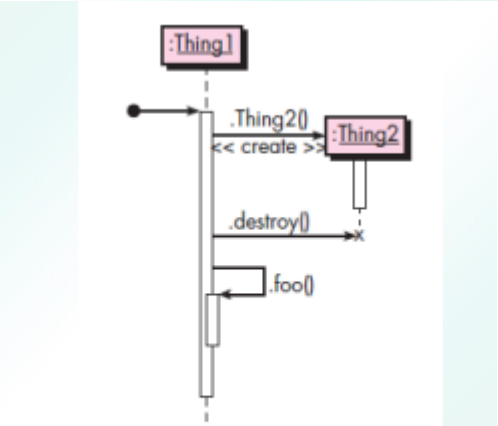


Figure 2.21
Special features included in a sequence diagram

SEQUENCE DIAGRAM: Another Example

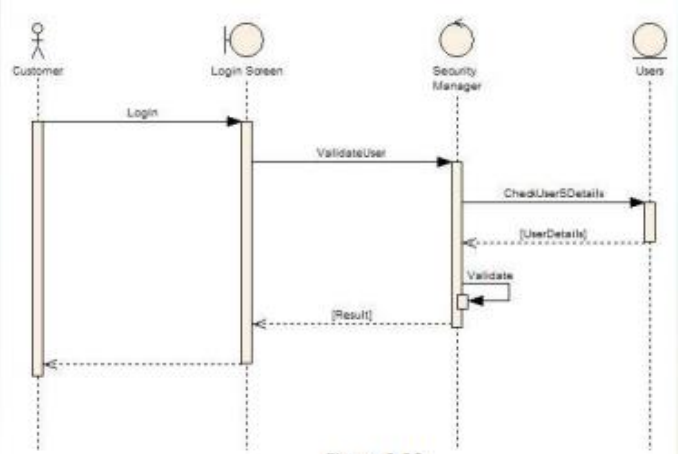
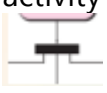
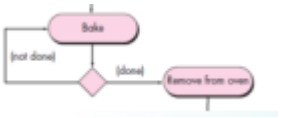


Figure 2.22
Another example of sequence diagram

ACTIVITY DIAGRAM

- 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 like a flowchart except that an activity diagram can show concurrent flows.
- Components:
  - 

 - **action node** (rounded rectangle) – tasked performed by the software system
  - 

 - **flow of control** (arrows) – arrow between two action nodes indicate that the first node will be performed following the second node.
  -  - **initial node** (solid black dot) – starting point
  -  - **final node** (a black dot surrounded by a black circle) – end of activity
  -  - **separation of activities** (fork) – consists of one arrow pointing to it and two or more arrows pointing out.
  -  - **synchronization of concurrent flow of control** (join) – a horizontal black bar with two or more incoming arrows and one outgoing arrow
  - 

 - **decision node** (diamond with incoming arrow and two or more outgoing arrows) – corresponds to a branch in the flow control based on a condition. Each outgoing arrow is labeled with a **guard** (a condition inside square brackets)

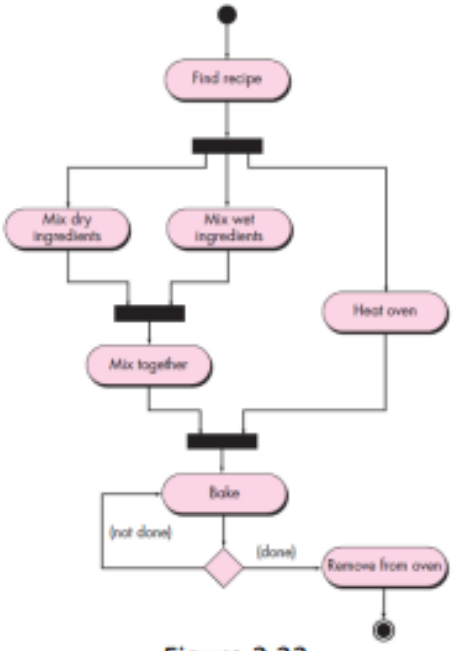
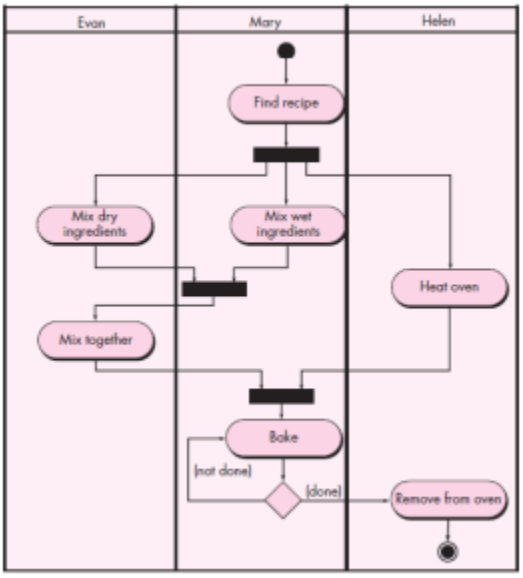


Figure 2.23

- Often, the exact division of labor does not matter. But if you do want to indicate how the actions are divided among the participants, you can decorate the activity diagram with **swimlanes**.
- Swimlanes, as the name implies, are formed by dividing the diagram into

strips or "lanes," each of which corresponds to one of the participants.



- An Activity Diagram resembles a horizontal flowchart that shows the actions and events as they occur. Activity diagrams show the order in which the actions take place and identify the outcomes.
- **Diagram Purpose**
  - Activity Diagram is typically used for modeling the logic captured in a specific use case in a use case diagram. Activity diagram can also be used to model a specific Actor's workflow within the entire system. Activity diagram can also be used independent of use cases for other purposes such as to model business process of a system, to model detailed logic of business rules etc. Activity diagram shows all potential sequence flows in an activity.