Review of UML Diagrams:

- **Uses of UML:**
    - As a sketch:
    - Very selective - informal and dynamic.
      I.e. Can be used to sketch a high level view of the system.
    - Forward engineering - Describes the concepts we need to implement.
    - Reverse engineering - Explains how parts of the code work.
    - As a blueprint:
    - Should be complete and describes the system in detail.
    - Forward engineering - Model as a detailed specification for the programmer.
    - Reverse engineering - Model as a code browser.
    - Tools provide both forward and reverse engineering to move back and forth between the program and the code.
    - As a programming language:
    - UML diagrams can be automatically compiled into working code using sophisticated tools, such as Visual Paradigm.
- **Things to Model:**
    - Structure of the code:
    - Code dependencies.
    - Components and couplings.
    - Behaviour of the code:
    - Execution traces.
    - State machine models of complex objects.
    - Function of the code:
    - What function does it provide to the user?

Interaction Diagrams:

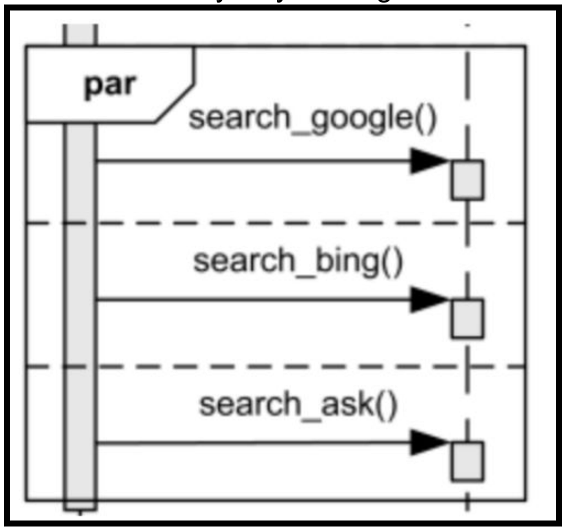
- **Interaction diagrams** describe how a group of objects collaborate in some behavior. They commonly contain objects, links and messages.
- Objects communicate with each other through function/method calls called **messages**.
- An interaction is a set of messages exchanged among a set of objects in order to accomplish a specific goal.
- Interaction diagrams:
    - Are used to model the dynamic aspects of a system.
    - Aid the developer visualize the system as it is running.
    - Are storyboards of selected sequences of message traffic between objects.
- After class diagrams, interaction diagrams are possibly the most widely used UML diagrams.
- A **lifeline** represents a single participant in an interaction. It describes how an instance of a specific classifier participates in the interaction. A lifeline represents a role that an instance of the classifier may play in the interaction.
- A **message** is the vehicle by which communication between objects is achieved. A function/method call is the most common type of message. The return of data as a result of a function call is also considered a message.
- A message may result in a change of state for the receiver of the message.
- The receipt of a message is considered an instance of an event.
- Interactions model the dynamic aspects of a system by showing the message traffic between a group of objects. Showing the time-ordering of the message traffic is a central ingredient of interactions.
- Graphically, a message is represented as a directed line that is labeled.
- The **sequence diagram** is the most commonly used UML interaction diagram. Typically a sequence diagram captures the behavior of a group of objects in a single scenario.

Other types of interaction diagrams include communication diagrams and timing diagrams. Out of those three, sequence diagrams are preferred for their simplicity.

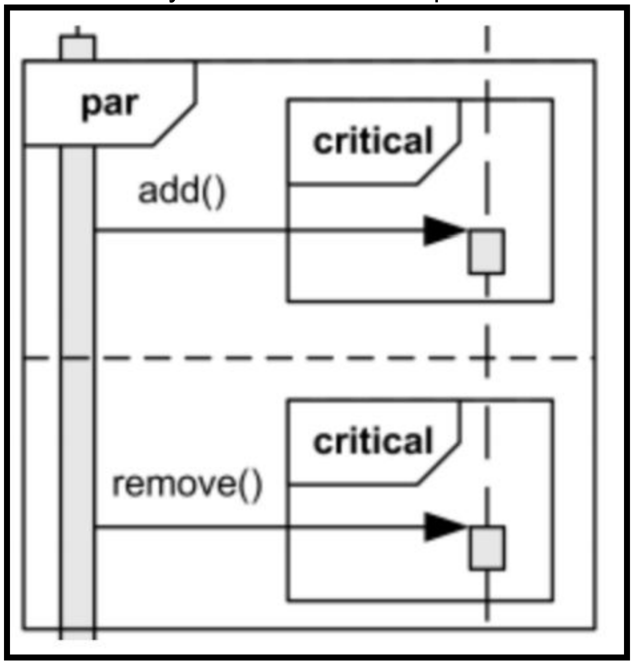
- Interaction Frame Operators:

Operator	**Name**	**Meaning**
Opt	Option	An operand is executed if the condition is true. (E.g. If-else)
Alt	Alternative	The operand, whose condition is true, is executed. (E.g. Switch)
Loop	Loop	It is used to loop an instruction for a specified period.
Break	Break	It breaks the loop if a condition is true or false, and the next instruction is executed.
Ref	Reference	It is used to refer to another interaction.
Par	Parallel	All operands are executed in parallel.
Region	Critical Region	Only 1 thread can execute this frame at a time.
Neg	Negative	Frame shows an invalid interaction.
Sd	Sequence Diagram	(Optional) Used to surround the whole diagram.

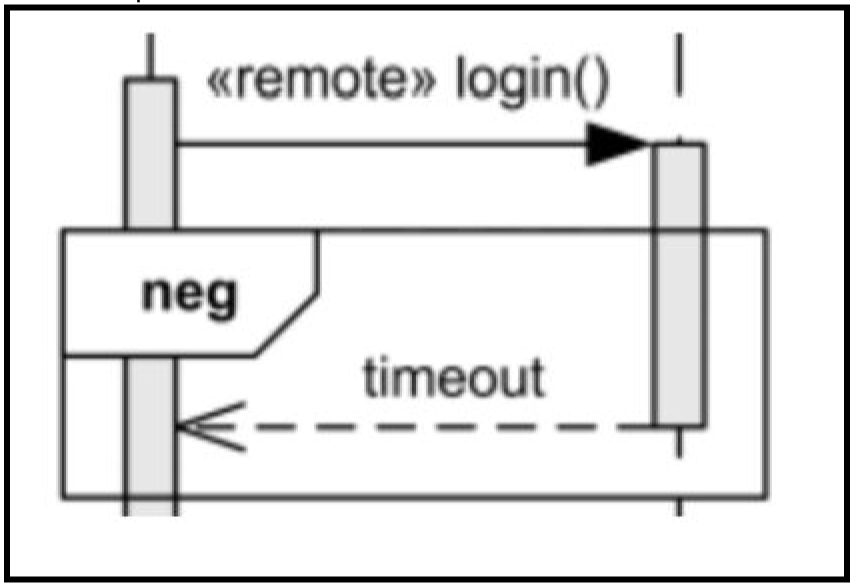
- **Parallel Example:** The interaction operator par defines potentially parallel execution of behaviors of the operands of the combined fragment. Different operands can be interleaved in any way as long as the ordering imposed by each operand is preserved.



- **Region Example:** The interaction operator region defines that the combined fragment represents a critical region. A critical region is a region with traces that cannot be interleaved by other occurrence specifications on the lifelines covered by the region.

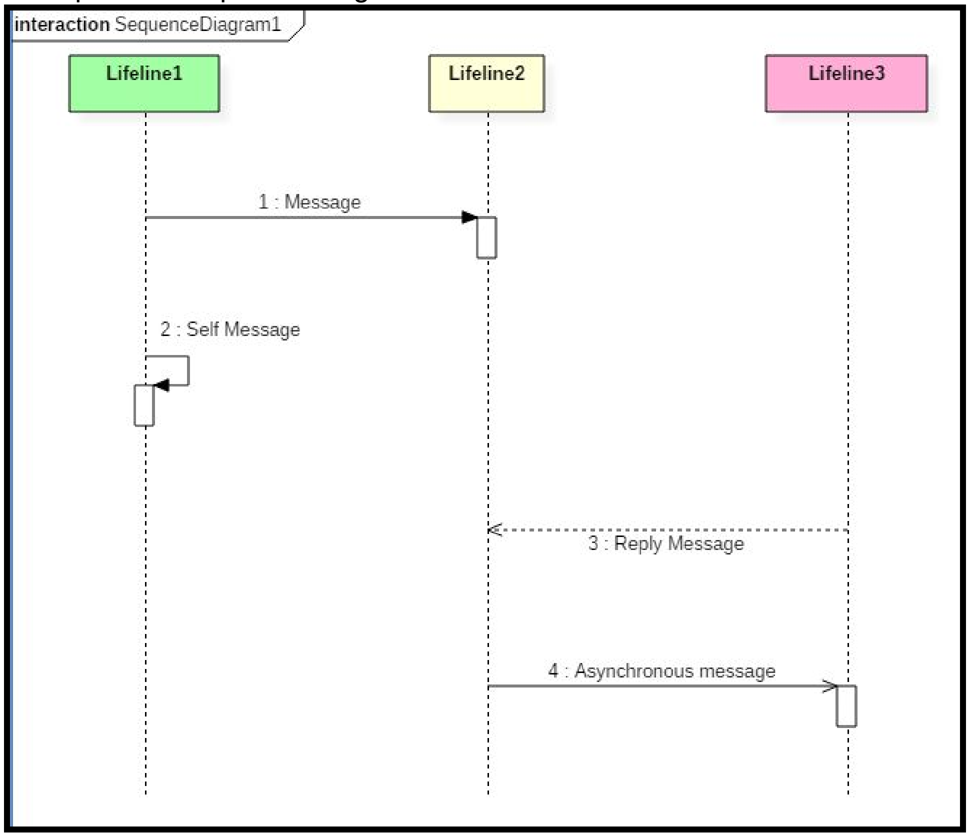


- **Negative Example:** The interaction operator neg describes a combined fragment of traces that are defined to be negative (invalid). Negative traces are the traces which occur when the system has failed. All interaction fragments that are different from the negative are considered positive, meaning that they describe traces that are valid and should be possible.

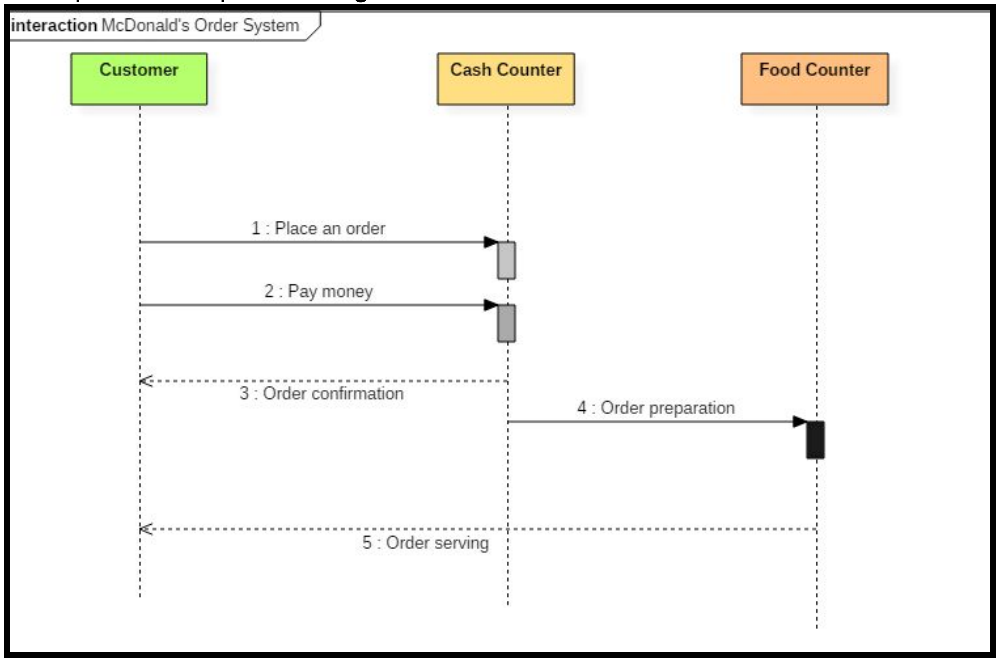


Sequence Diagrams:

- **Introduction:**
- A **sequence diagram** depicts interactions between objects in a sequential order. The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system. The sequence diagram shows the interaction between two lifelines as a time-ordered sequence of events.
- A sequence diagram shows an implementation of a scenario in the system. Lifelines in the system take part during the execution of a system.
- In a sequence diagram, a lifeline is represented by a vertical bar.
- A message flow between two or more objects is represented using a vertical dotted line which extends across the bottom of the page.
- Sequence diagrams are built around an X-Y axis.
- Objects are aligned at the top of the diagram, parallel to the X axis.
- Messages travel parallel to the X axis.
- Time passes from top to bottom along the Y axis.
- Sequence diagrams most commonly show relative timings, not absolute timings.
- Links between objects are implied by the existence of a message.
- Example of a sequence diagram:



- Example of a sequence diagram:



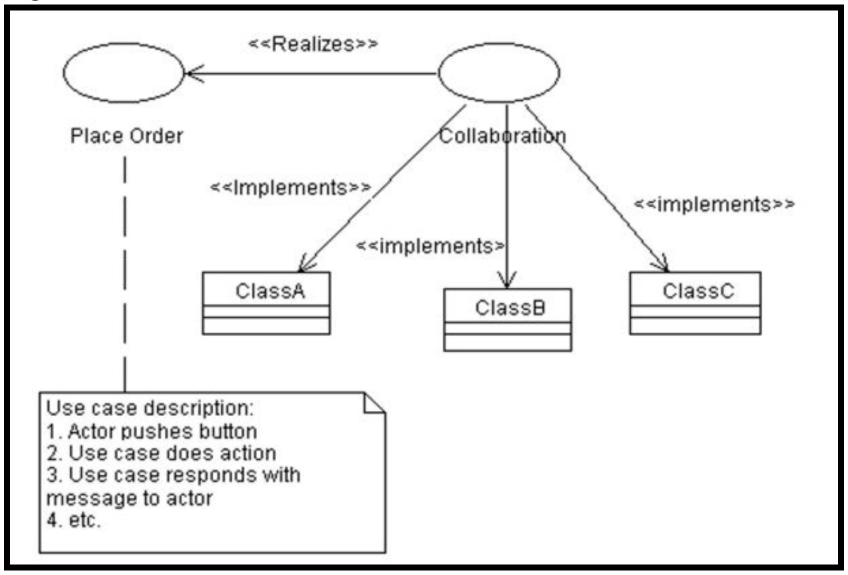
- **Benefits of a sequence diagram:**
- Sequence diagrams are used to explore any real application or a system.
- Sequence diagrams are used to represent message flow from one object to another object.
- Sequence diagrams are easy to maintain and generate.
- Sequence diagrams can be easily updated according to the changes within a system.
- Sequence diagrams allow both reverse and forward engineering.
- **Drawbacks of a sequence diagram:**
- Sequence diagrams can become complex when too many lifelines are involved in the system.
- If the order of message sequence is changed, then incorrect results are produced.
- Each sequence needs to be represented using different message notation, which can be a little complex.
- The type of message decides the type of sequence inside the diagram.
- **When to use sequence diagrams:**
    1. <u>Comparing Design Options:</u>
    - Shows how objects collaborate to carry out a task.
    - Graphical form shows alternative behaviours.
    2. <u>Assessing Bottlenecks:</u>
    3. <u>Explaining Design Patterns:</u>
    - Enhances structural models.
    - Good for documenting behaviour of design features.
    4. <u>Elaborating Use Cases:</u>
    - Shows how the user expects to interact with the system.
    - Shows how the user interface operates.
- **Modelling Control Flow By Time:**
- Determine what scenarios need to be modeled.
- Identify the objects that play a role in the scenario.

- Lay the objects out in a sequence diagram left to right, with the most important objects to the left.
  Most important in this context means objects that are the principle initiators of events.
- Draw in the message arrows, top to bottom.
  Adorn the message as needed with detailed timing information.
- **Style Guide for Sequence Diagrams:**
  1. Spatial Layout:
  - Strive for left-to-right ordering of messages.
  - Put proactive actors on the left.
  - Put reactive actors on the right.
  2. Readability:
  - Keep diagrams simple.
  - Don't show obvious return values.
  - Don't show object destruction.
  3. Usage:
  - Focus on critical interactions only.
  4. Consistency:
  - Class names must be consistent with class diagram.
  - Message routes must be consistent with navigable class associations.

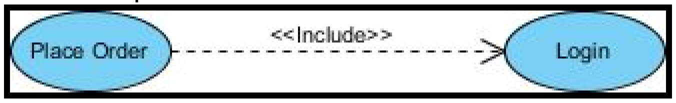
Use Case Diagrams:

- **Introduction:**
- A **use case diagram** is the primary form of system/software requirements for a new software program underdeveloped.
- Use cases specify the expected behavior (what), and not the exact method of making it happen (how).
- A key concept of use case modeling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behavior in the user's terms by specifying all externally visible system behavior.
- Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.
- A use case:
  - Specifies the behavior of a system or some subset of a system.
  - Is a set of scenarios tied together by a common user goal.
  - Does not indicate how the specified behavior is implemented, only what the behavior is.
  - Performs a service for some user of the system.
  - A user of the system is known as an **actor**.
  - During the analysis phase, facilitates communication between the customer, users of the system and the developers of the system.
- A use case represents a functional requirement of the system. A requirement:
  - Is a design feature, property, or behavior of a system.
  - States what needs to be done, but not how it is to be done.
  - Is a contract between the customer and the developer.
  - Can be expressed in various forms, including use cases.
- In brief, the purposes of use case diagrams are as follows:
  - Used to gather the requirements of a system.
  - Used to get an outside view of a system.
  - Identify the external and internal factors influencing the system.
  - Show the interaction among the requirements of the **actors**.

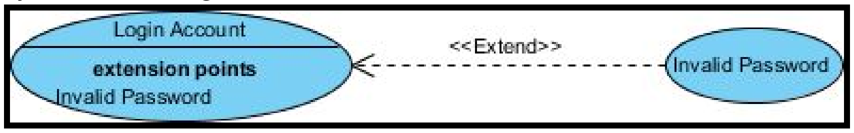
- **Actors** can be a human user, some internal applications, or may be some external applications. It is a user of the system.
- An actor:
    - Is a role that the user plays with respect to the system.
    - Is associated with one or more use cases.
    - Does not have to be human.
    - Is a user of the system.
    - Is most typically represented as a stick figure of a person labeled with its role name. Note that role names should be nouns.
    - May exist in a generalization relationship with other actors in the same way as classes may maintain a generalization relationship with other classes.
- **Note:** Use cases only show the relationships between actors, the systems and use cases and does not show the order in which the steps are performed to achieve the goals of each use case.
- Use cases are a technique for capturing the functional requirements of a system. Use cases work by describing the typical interactions between the users of a system and the system itself, providing a narrative of how the system is used.
- Use case development process:
    1. Develop multiple scenarios.
    2. Distill the scenarios into one or more use cases where each use case represents a functional requirement.
    3. Establish associations between the use cases and actors.
- A use case is graphically represented as an oval with the name of its functionality written inside. The functionality is always expressed as a verb or a verb phrase.
- A use case may exist in relationships with other use cases much in the same way as classes maintain relationships with other classes.
- As stated earlier, a use case by itself does not describe the flow of events needed to carry out the use case. The flow of events can be described using informal text, pseudocode, or activity diagrams.
  I.e. May use a note to attach flow of events documentation to a use case.
  Be sure to address exception handling (error conditions) when describing flow of events.
  The amount to detail you need in a use case depends on the amount of risk in that use case.
  E.g.



- **Relationships Between Use Cases:**
- A use case may have a relationship with other use cases.
- Generalization between use cases is used to extend the behavior of a parent use case.
- An **<<include>> relationship** between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base.
  Note: Sometimes the **<<uses>>** stereotype is used instead of <<include>>.
  When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.



- An **<<extend>> relationship** between use cases means that the base use case implicitly incorporates the behavior of another use case at a location specified indirectly by the extending use case.



- Extended behavior is optional behavior, while included behavior is required behavior. I.e. Extended means "may use" while include/uses means "will use".
- Extend occurs when one use case adds a behaviour to a base use case while include occurs when one use case invokes another.
- **Actor Classes:**
- Identify classes of actors.
- Actors inherit use cases from the class.
- **Describing Use Cases:**
- For each use case, a flow of events document, written from the actor's point of view, describes what the system must provide to the actor when the use case is executed.
- Typical contents:
  - How the use case starts and ends.
  - Normal flow of events.
  - Alternate flow of events.
  - Exceptional flow of events.
- Documentation style:
  - Activity Diagrams - Good for business process.
  - Collaboration Diagrams - Good for high level design.
  - Sequence Diagrams - Good for detailed design.
- **Finding Use Cases:**
- Noun phrases may be domain classes.
- Verb phrases may be operations and associations.
- Possessive phrases may indicate attributes.
- **For each actor, ask the following questions:**

1. What functions does the actor require from the system?
2. What does the actor need to do?
3. Does the actor need to read, create, destroy, modify or store information in the system?
4. Does the actor have to be notified about events in the system?
5. Does the actor need to notify the system about something?
6. What do these events require in terms of system functionality?
7. Could the actor's daily work be simplified or made more efficient through new functions provided by the system?