**Phần 1: giới thiệu**

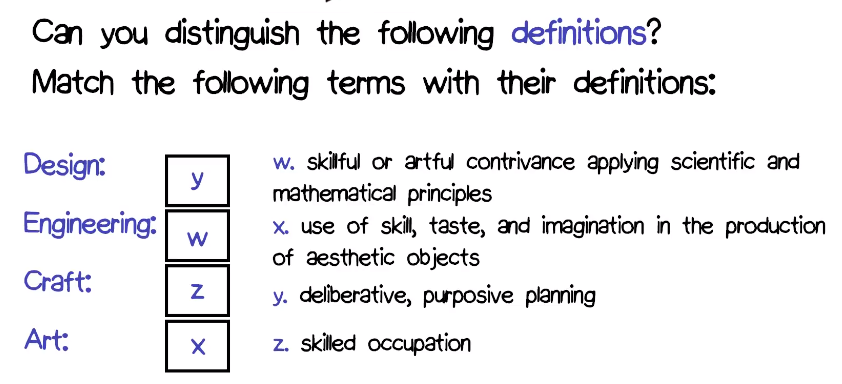
**Bài 1: giới thiệu**

Kiến trúc phần mềm của một chương trình máy tính là cấu trúc của các thành phần trong hệ thống đó. Kiến trúc phần mềm bao gồm các phần tử phần mềm, các thuộc tính và muốn quan hệ giữa chúng.

Kiến trúc phần mềm giúp việc đưa ra các quyết định ở mức **high-level** trở nên dễ dàng hơn và cho phép tái sử dụng các thành phần và các design pattern của các dự án phần mềm

**Bài 2: khái niệm thiết kế**

Design: là planning một cách có chủ ý, có chủ đích



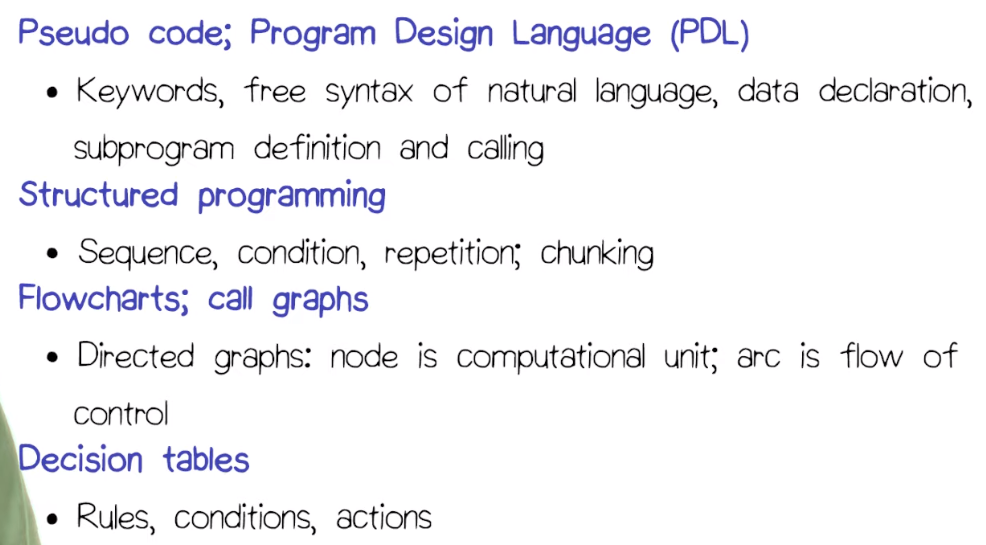
Điểm khác biệt giữa software design và programming:

- Scale

- Nhấn mạnh vào các Non-functional Requirements

Như vậy, **Software Design** là gì? Là process building một program trong khi vẫn thỏa mãn các vấn đề của functional requirements mà không làm vi phạm đến các non-functional constraints. Software Design được thường chia làm 2 giai đoạn:

1. Architectural Design: process của việc xác định và định hình trách nhiệm của các modules hay các components khác nhau của phần mềm trong việc đưa ra các hàng vi trong quá trình chạy của ứng dụng
2. Detail Design: process của việc dealing với các components riêng biệt mà chúng ta đã định hình trong architectural design



**Approaches to Software Design:**

- Method: là một chuỗi các bước nối tiếp nhau 1 cách có hệ thống trình bày một cách để giải quyết 1 vấn đề.

**Traditional Design Documentation:**

* Subcomponents:
  + Processes / activities
  + Data / Data flows
* Control Flow:
  + Control regime
* Performance
* Resources

**Design Rationale** (cơ sở lý luận của design)**:** lý do vì sao chúng ta lại coming up với design solution hiện tại. Design decisions là các lựa chọn rõ ràng chúng ta trade off giữa 2 non-functional aspects của một design, ví dụ giữa tốc độ và size

1 A / B

2 D / A

3 B

4 D / B

5 D

**--------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- ---------------**

**Phần 2: ngôn ngữ mô hình hóa thống nhất và ngôn ngữ ràng buộc đối tượng**

**Bài 3: đánh giá kiến thức về UML**

**Ngôn ngữ UML (ngôn ngữ mô hình hợp nhất UML)** là một ngôn ngữ để xác định, hình dung, xây dựng và ghi lại các khía cạnh của hệ thống phần mềm, cũng như mô hình nghiệp vụ và các hệ thống non-software khác. UML đại diện cho một tập hợp các minh họa thực tế dưới dạng kỹ thuật ở mức tốt nhất để chứng minh thành công trong việc mô hình hóa các hệ thống lớn và phức tạp.

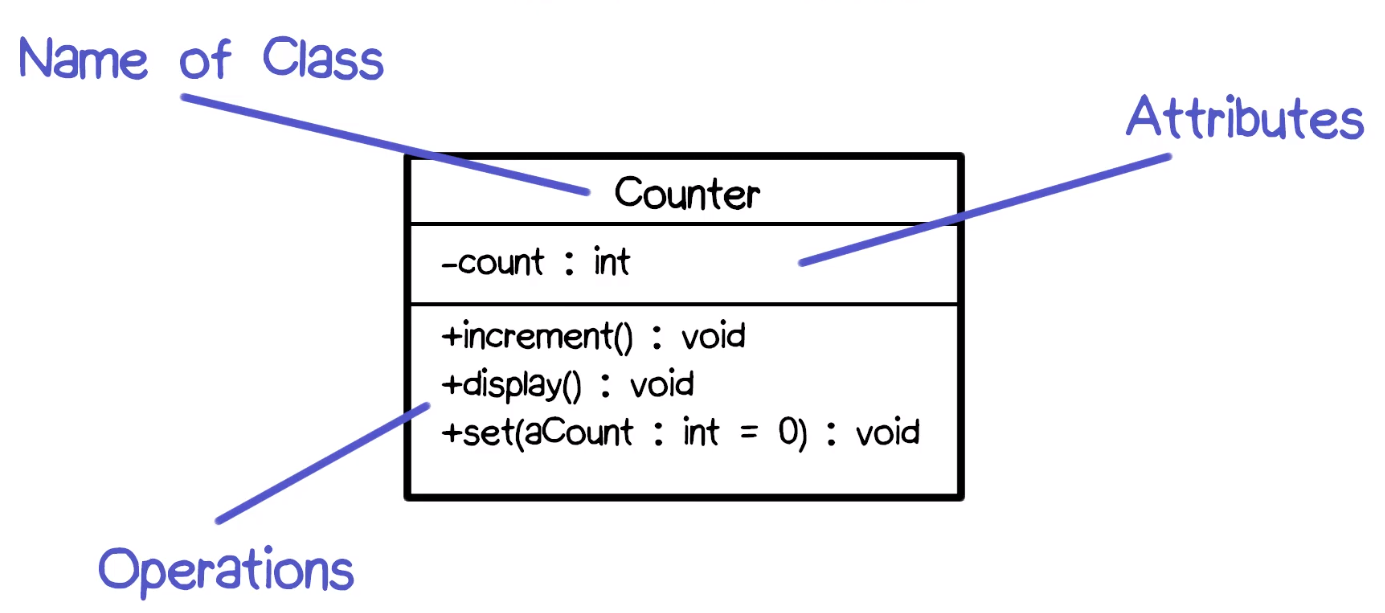
Các loại UML sẽ có một tính chất riêng biệt khác nhau, có loại giúp biểu diễn được thuật toán, có loại giúp hiểu được ngữ cảnh của hệ thống, có loại giúp nắm được tuần tự của hoạt động nghiệp vụ để biểu đạt bằng UML.

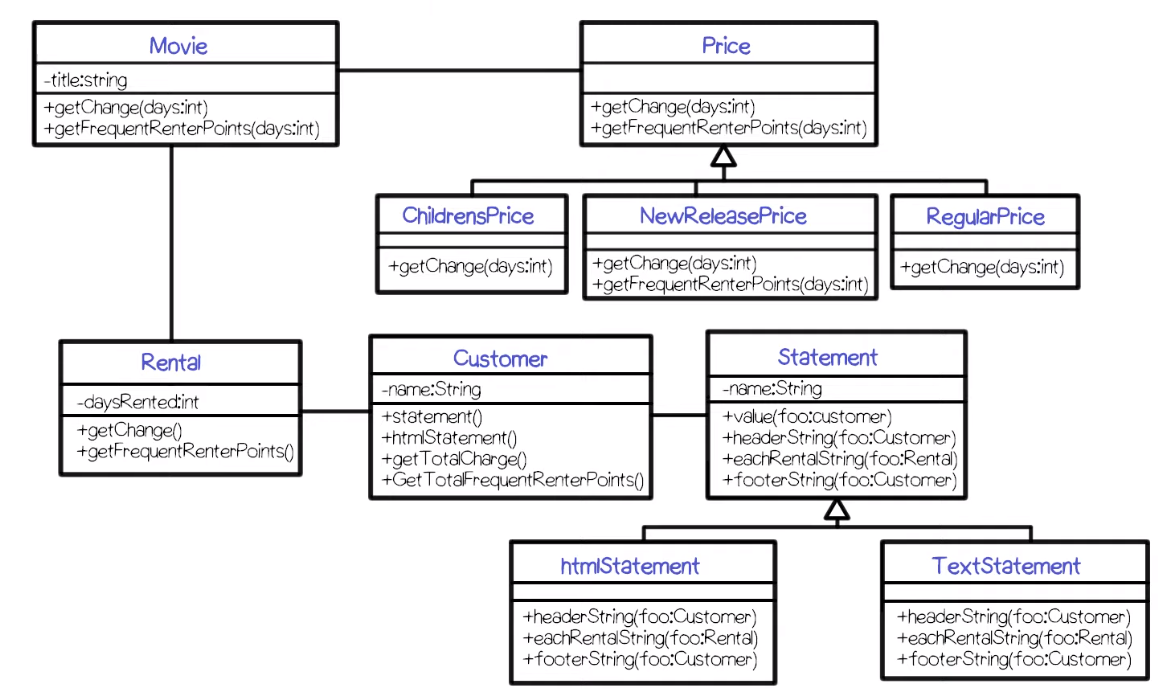
Các Diagrams có thể được chia làm 2 loại chính:

* Structural Diagrams: cho chúng ta thấy 1 phần hoặc toàn bộ hệ thống, luôn luôn static, và mối liên hệ giữa các thành phần này
* Behavioral Diagrams: tập trung vào executions của hệ thống

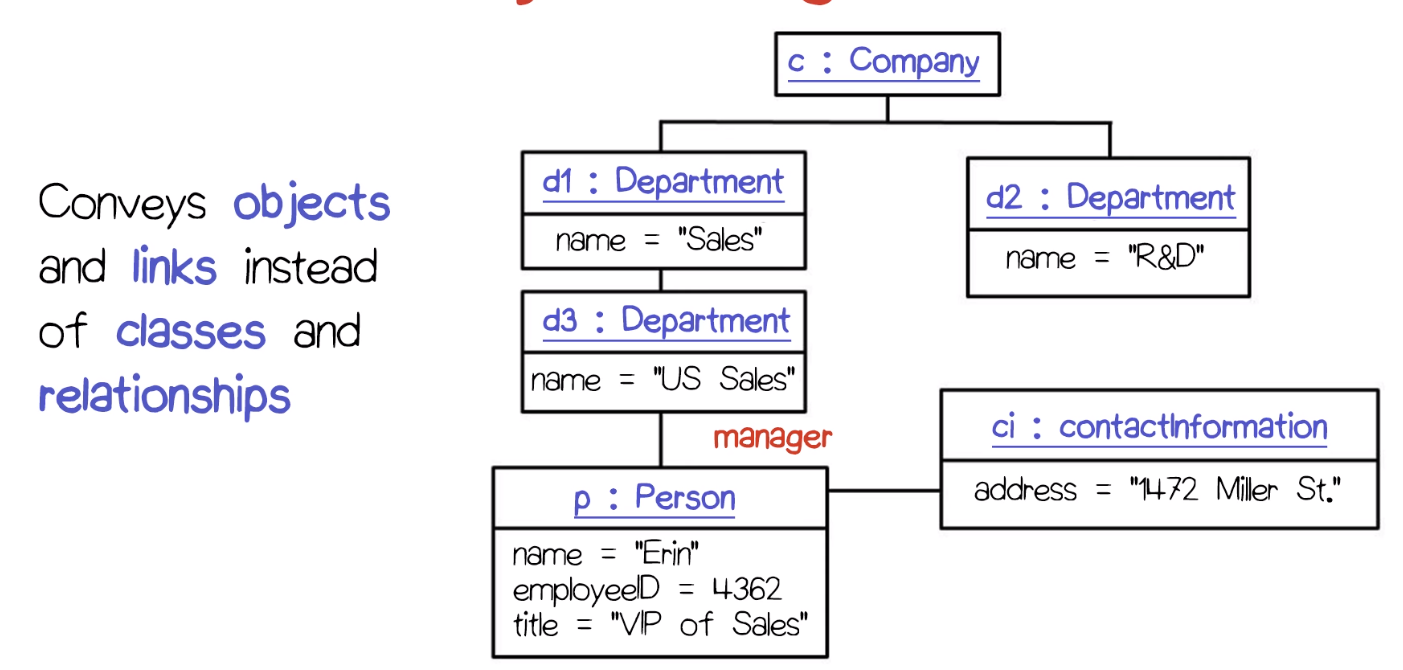
**ĐI QUAN TỪNG LOẠI UML DIAGRAM:**

1. UML Classes:
   1. Relationships:
      1. Dependency: X uses Y, sử dụng *dashed* hoặc *directed line*
      2. Asociations: X affects Y, sử dụng các line vô hướng
      3. Generalization: X is a kind of Y, sử dụng solid line với mũi tên

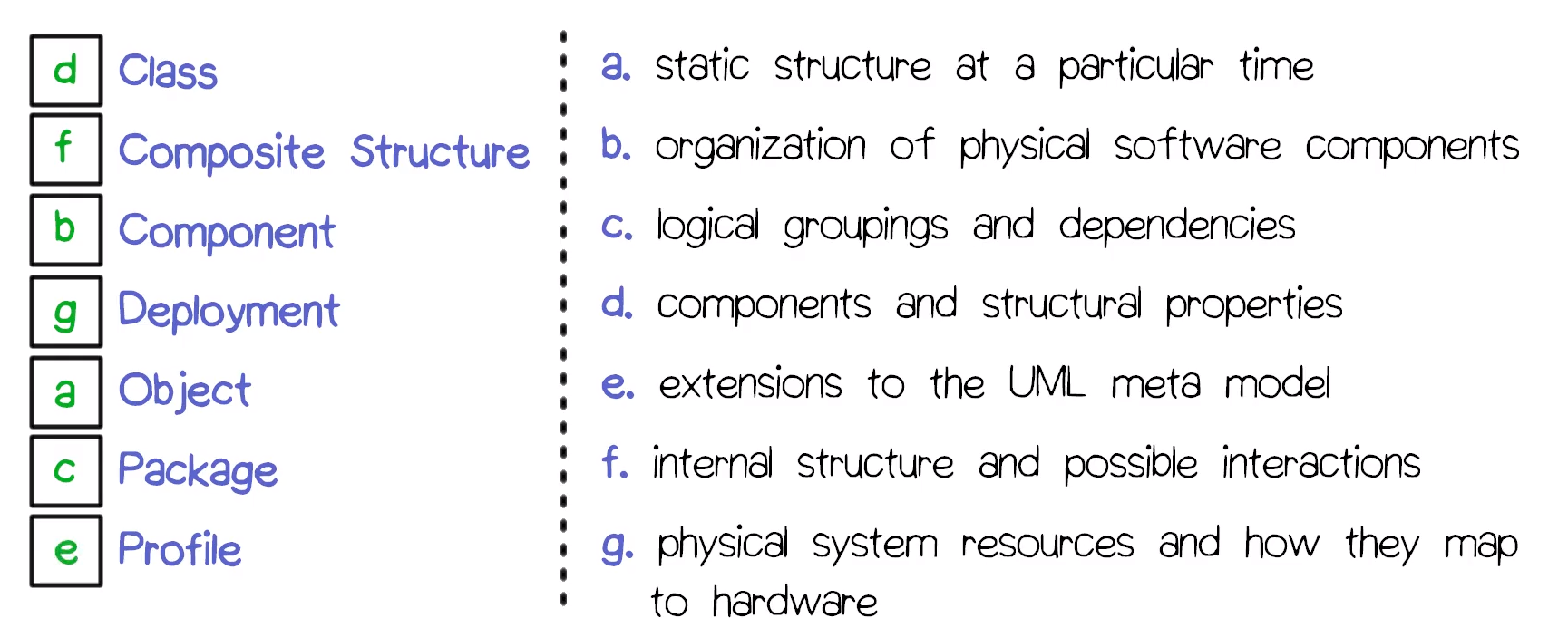




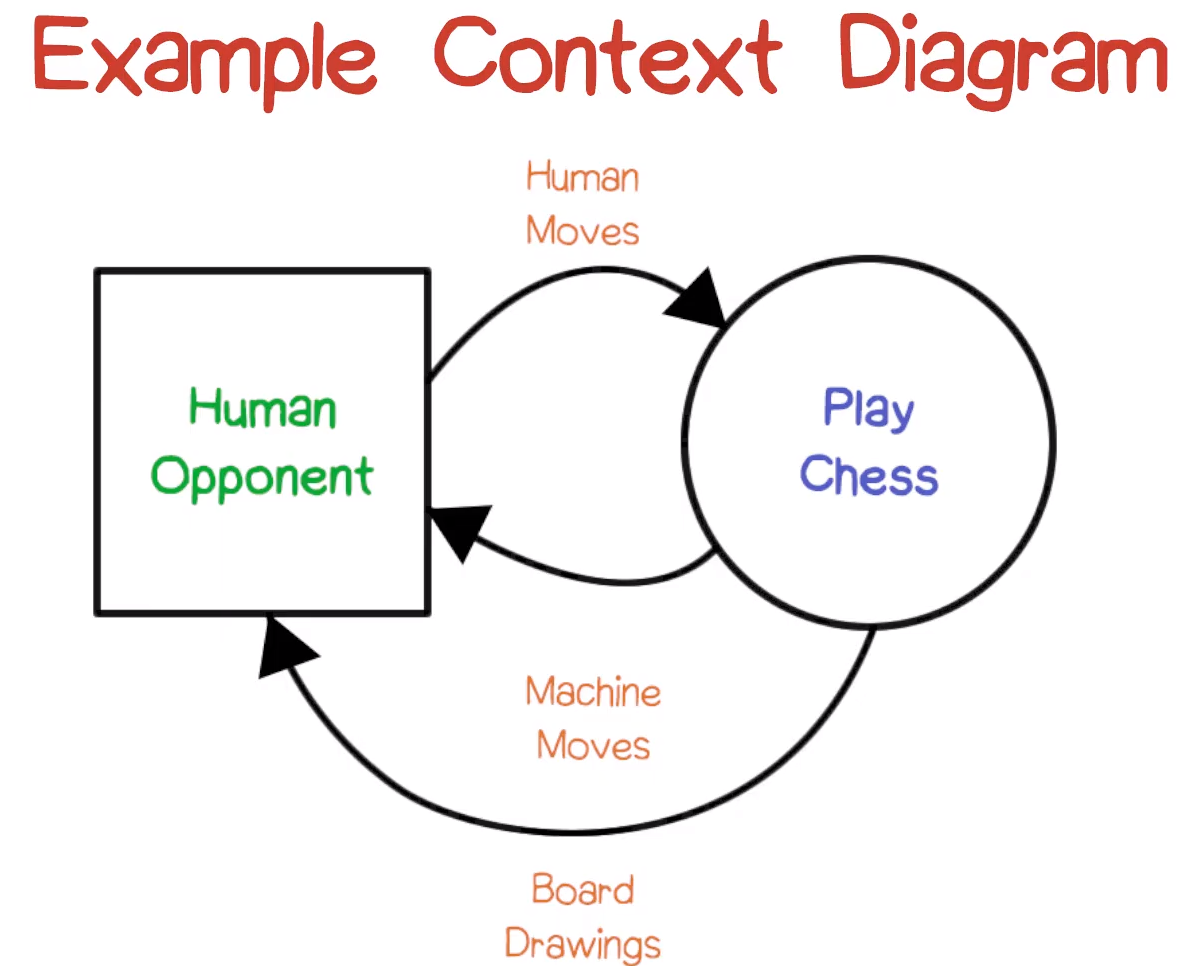
1. Object Diagram: related visually đối với class model diagram là object digram, thay vì mention đến classes, nó đề cập đến các instances được tạo nên từ Class.



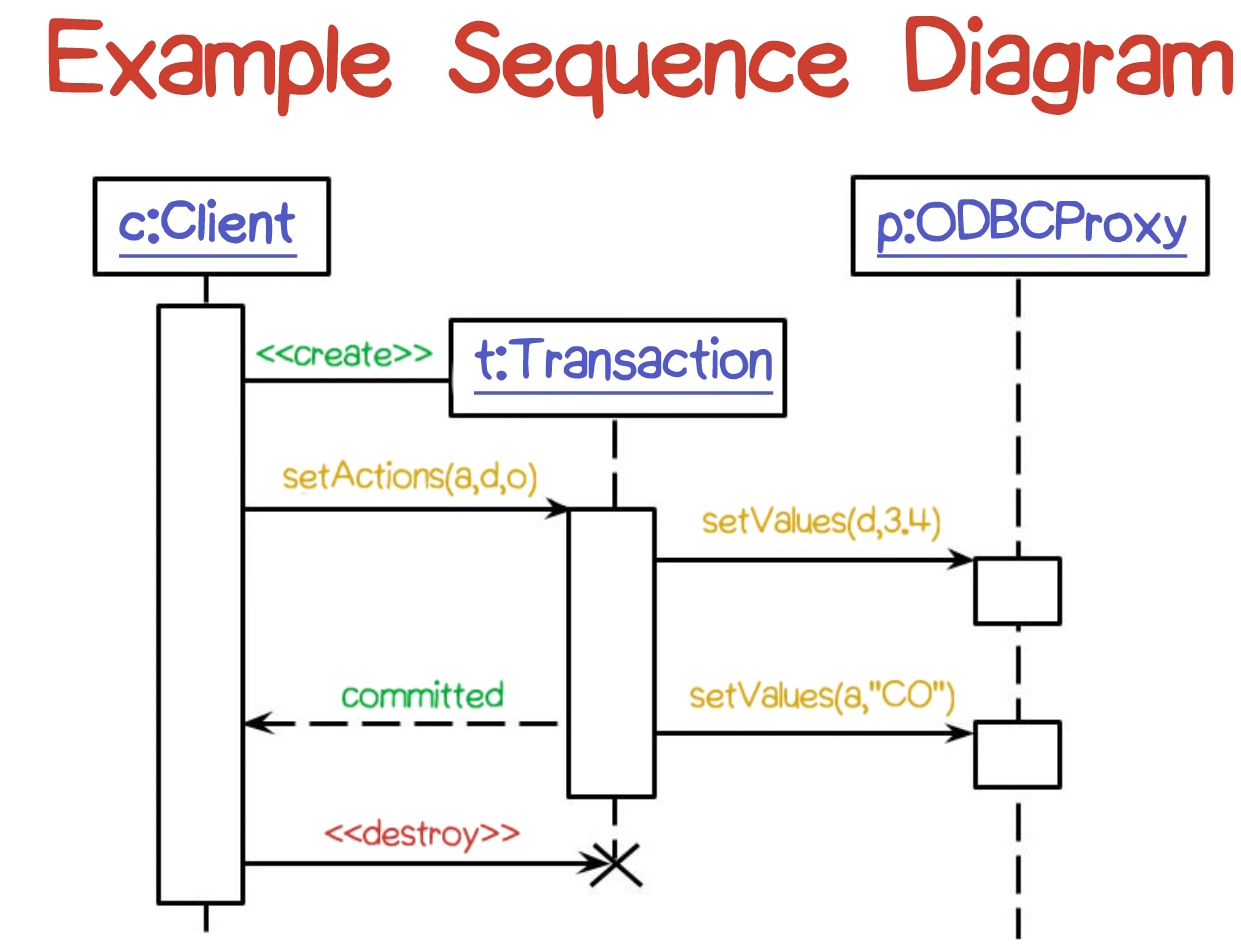
1. Component diagrams
   1. Là một static implementation view: được dùng để thể hiện sữ liên hệ giữa các components trong một hệ thống. Cụm từ **Component** được dùng để refer đến module of classes hiện thân cho một phần độc lập của system hoặc sub-system và thể hiện sự giao tiếp giữa chúng.



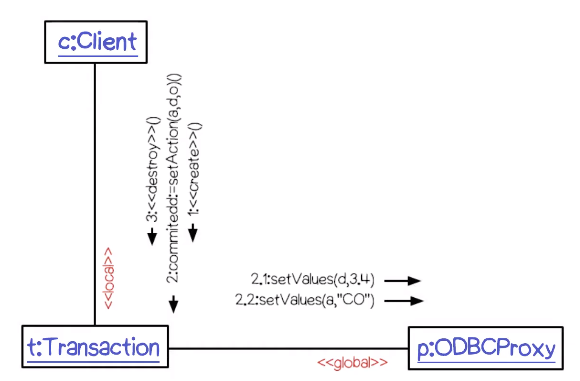
1. UML Diagrams:
   1. Một use-case là một chuỗi các hành động đi cùng với phản hồi của hệ thống, hay nói cách khác nó thể hiện how the system deals với một tương tác cụ thể
   2. Use-case diagram cực kỳ hữu dụng trong việc khơi gợi system requirements và tổ chức development activities
   3. Miêu tả mối quan hệ giữa các actors và use cases
2. Context Diagrams:
   1. KO PHẢI là một phần của UML nhưng cung cấp các chức năng rất hữu ích
   2. Dataflow diagrams cung cấp các cái nhìn khác nhau
   3. DFD miêu tả process, actorss và dataflow giữa chúng



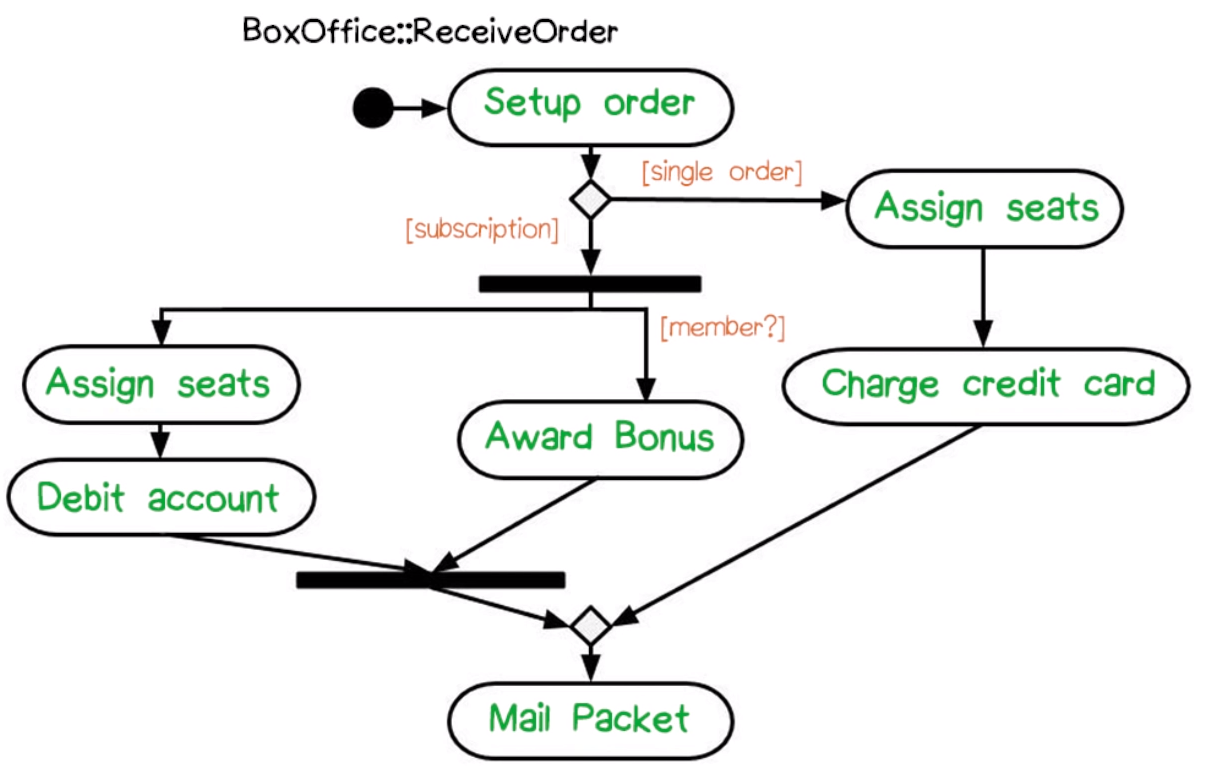
1. Sequence Diagram
   1. Có thể được dùng để miêu tả **MỘT** use case



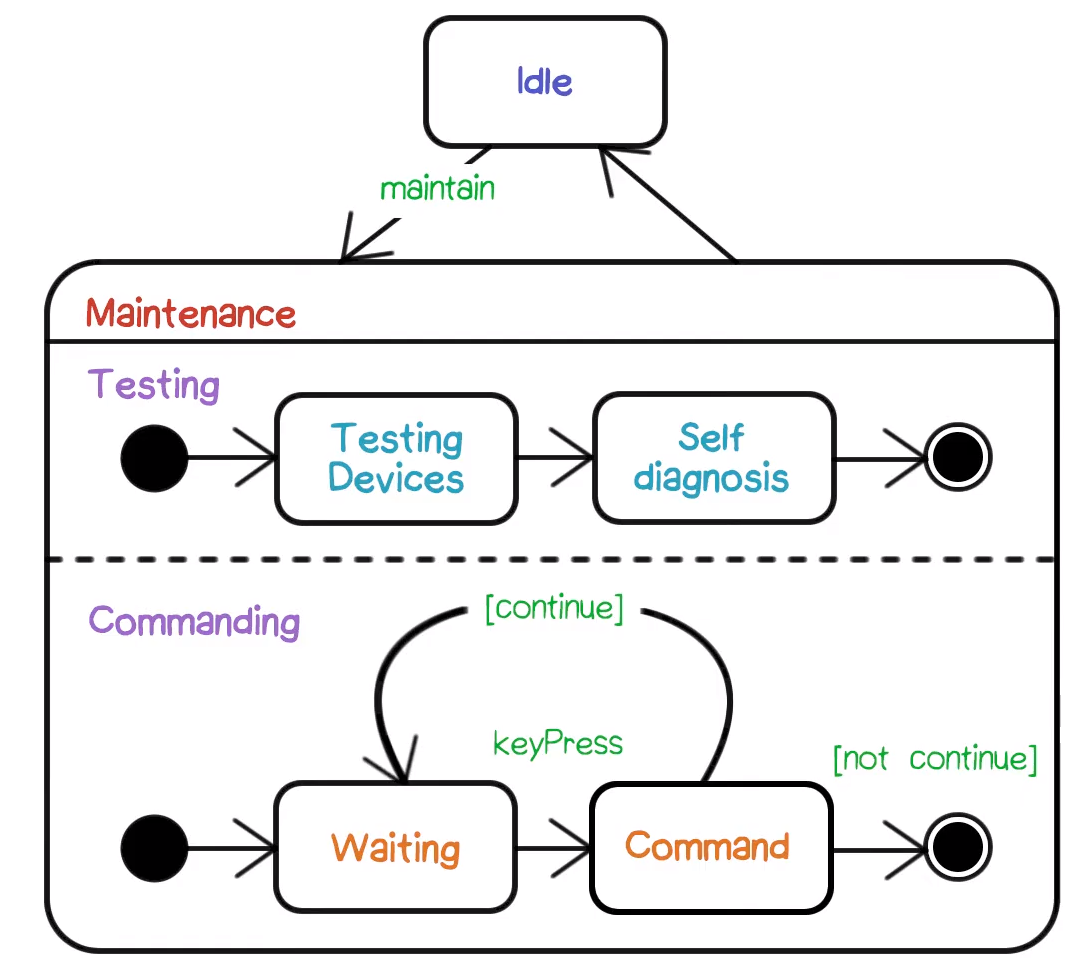
1. Communication diagram:
   1. Là object diagram được chú thích với thứ tự của các tương tác thay cho các links (đường nối)
   2. Về mặt ngữ nghĩa thì nó tương đương với Sequence diagram

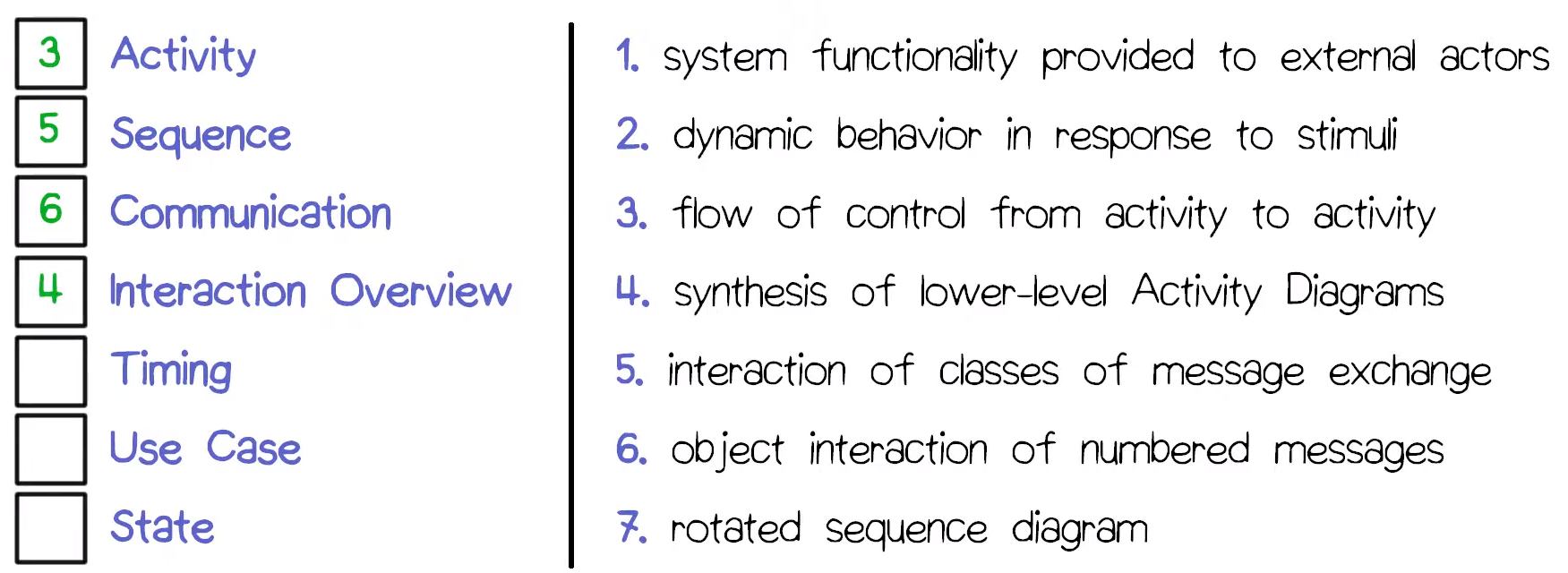


1. Activity Diagram:
   1. Là một biến thể của state machine trong đó các states có thể đồng thời diễn ra.
   2. Các transitions thông thường được triggered bởi sự kết thúc của các hoạt động (activity completion)
   3. Thường được dùng để model workflow, process đồng bộ hóa và đồng thời



1. Interaction Overview Diagram + Timing diagram: ít dùng, tham khảo thôi.
2. State diagram:
   1. Là diagram cực kỳ mạnh mẽ nhưng cũng là loại phức tạp nhất trong số các UML behavior diagram, còn được gọi là state charts
   2. Extended finite state machine, có chức năng thể hiện tập hợp lịch sử, concurrency, broadcasting events …





**1 D**

**2 E**

**3 B**

**4 A + C**

**5 A + B + E / A + E /**

**6 D**

**--------------- --------------- --------------- ---------------**

**Bài 4: ngôn ngữ ràng buộc đối tượng**

**-** Trong mô hình hóa hệ thống với UML, ta có thể sử dụng ngôn ngữ ràng buộc đối tượng (OCL) để đặc tả chính xác các phần tử của hệ và các ràng buộc chặt chẽ giữa các mối quan hệ, giới hạn phạm vi của mô hình hệ thống cho phù hợp với điều kiện ràng buộc thức tế.

(***Keywords***: Side effect, Invariant. Precondition, Postcondition, Parameter, Primitive types, Navigation OCL, Navigation multiplicity OCL, Tuple expression, Enumerations, Message expression)

1. OCL (object constraint language) là gì?
   1. Là một phần của UML
   2. Là một ngôn ngữ đặc tả, nó là những khai báo, được định kiểu rõ (strongly typed) và cho phép bạn chỉ định các chức năng chi tiết của những thuộc tính hệ thống
   3. Có thể nói PCL bao gồm những ràng buộc + tập hợp các classes + UML diagram navigation
2. Tại sao chúng ta lại cần OCL?
   1. UML vẫn có những giới hạn trong việc thể hiện những gì mong muốn
   2. Có những trường hợp chúng ta cần phải tóm lược cụ thể hơn về chi tiết của các chức năng
   3. OCL mở rộng (extended) UML với
      1. Class bất biến (Class invariants)
      2. Operation pre & post conditions
      3. Guard on state-machine transitions
3. OCL overview:
   1. Tập trung vào khai báo chứ không phải procedural
      1. Là pure expression language
      2. Không có các assignment statements hoặc các side effects
   2. Strongly typed
      1. Built-in types và các types đã được định nghĩa trong UML diagram
   3. Cơ chế cấp cao nhất là constraint:
      1. Có thể coi như là những lời khẳng định chính thức về các thuộc tính của hệ thống
4. Syntax:
   1. context <identifier> <constrainType>: <Boolean expression>
5. Invariants:
   1. Statement of a property luôn luôn đúng.
   2. Express key system requirements.
   3. **inv** keyword
   4. Ví dụ: *context LargeCompany inv: numberOfEmployees > 50*
6. Role of Invarriants:
   1. Properties của dữ liệu trong một database mô tả liệu Database đó có tốt không.
   2. OCL invariant constraint đóng vai trò này trong system specifications
7. Pre and Post conditions:
   1. Được sử dụng để express chính xác ý nghĩa của một UML operations
   2. **Pre condition** diễn tả những điều phải đúng để operation có thể diễn ra
   3. **Post condition** ngược lại là thể hiện những gì chắc chắn sẽ đúng sau khi operation được hoàn thành, hay nói cách khác nó là resultss sau khi executing operation đó.

1 C

2 A B D E /

3 C / B /

4 A B D / A C D /

5 A / B