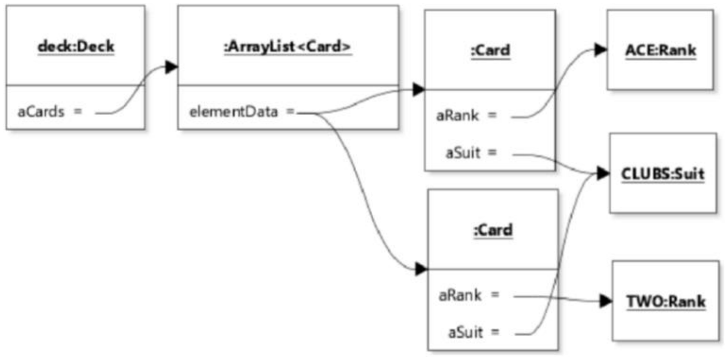
UML DIAGRAMS:

OBJECT DIAGRAMS

- An object and its reference are created whenever a ***new*** statement is executed
- **Object**: a rectangle (optional: with its name and type indicated by ***name: type***)
- **Fields**: (contained in the object)
  - Contain a primitive type
  - Or a value of reference type (another object) which is indicated by directed arrow

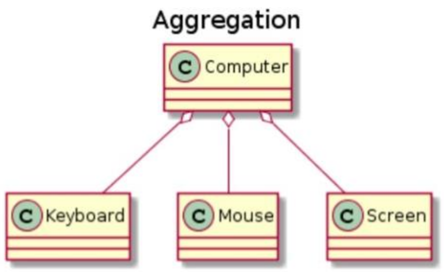


CLASS DIAGRAMS

Class/Interface

- It is a modeling error to have **both a field and an aggregation** to represent a given field
- Hard to indicate a class does not have certain member → use a note
- If method is static, add the keyword **static** before the method

Generalization, Realization, Aggregation, Dependency

- **Generalization** (泛化关系)
  - When "extends" is used (实线箭头 实心三角)
- **Realization** (实现关系)
  - When "implements" is used (虚线箭头 实心三角)
- **Aggregation** (聚合关系)
  - 表示整体由部分组成，但是整体和部分不是强依赖的，整体不存在了部分还是会存在。
  - 

- **Dependency** (依赖关系)
    - A 类是 B 类方法的局部变量;
    - A 类是 B 类方法当中的一个参数;
    - A 类向 B 类发送消息，从而影响 B 类发生变化。

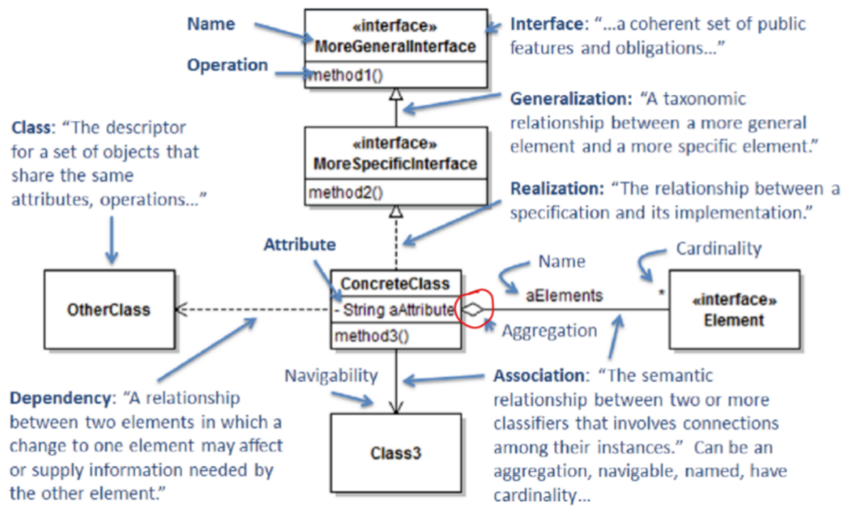
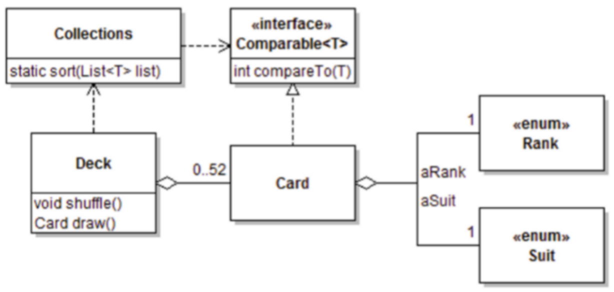
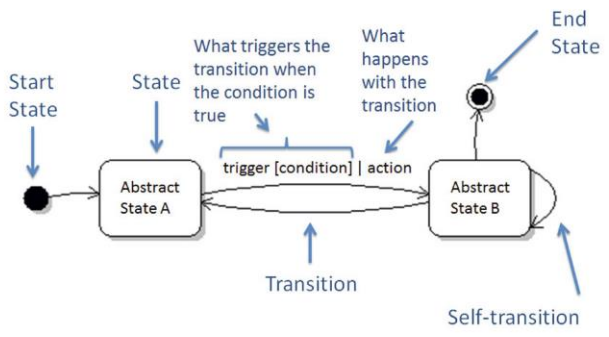

Name
Operation
«interface» MoreGeneralInterface
method1()
Interface: "...a coherent set of public features and obligations..."
Generalization: "A taxonomic relationship between a more general element and a more specific element."
Class: "The descriptor for a set of objects that share the same attributes, operations..."
«interface» MoreSpecificInterface
method2()
Realization: "The relationship between a specification and its implementation."
Attribute
Name
Cardinality
OtherClass
ConcreteClass
- String aAttribute
method3()
aElements
*
«interface» Element
Aggregation
Navigability
Association: "The semantic relationship between two or more classifiers that involves connections among their instances." Can be an aggregation, navigable, named, have cardinality...
Dependency: "A relationship between two elements in which a change to one element may affect or supply information needed by the other element."
Class3

Fig. 3.1 Selected notation for class diagrams



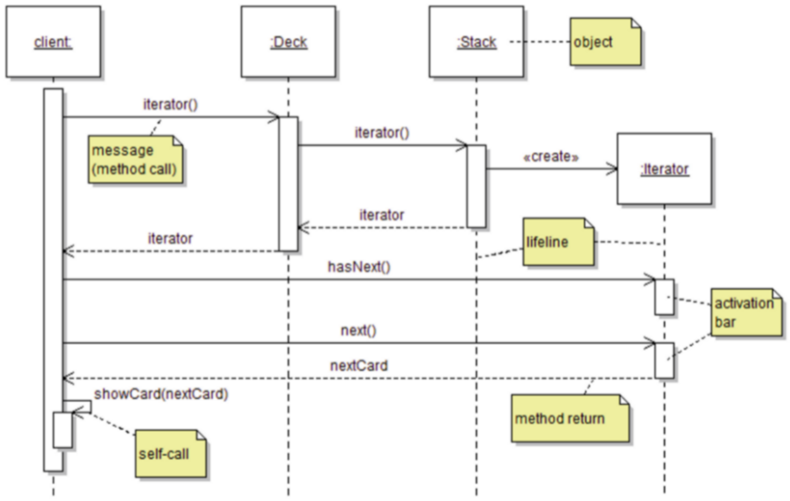
STATE DIAGRAMS

- UML state diagrams are useful to represent how objects can transition from one abstract state to another during their lifetime as a reaction to external events (typically, method calls)


Start State
State
What triggers the transition when the condition is true
What happens with the transition
End State
Abstract State A
trigger [condition] | action
Abstract State B
Transition
Self-transition

SEQUENCE DIAGRAMS

- sequence diagrams model the dynamic perspective on a software system.
- convention ***name: type*** as necessary.
- **object's lifeline**: dashed vertical line emanating from an object. (running from top to bottom) When objects are placed at the top of the diagram, they are assumed to exist at the beginning of the scenario being modeled.
- **Messages** are represented using a directed arrow from the caller object to the called object. Messages are typically labeled with the **method** that is called
- **return edges**


client:
:Deck
:Stack
object
iterator()
message (method call)
iterator()
«create»
:Iterator
iterator
iterator
lifeline
hasNext()
activation bar
next()
nextCard
showCard(nextCard)
method return
self-call