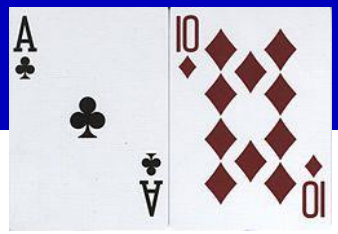
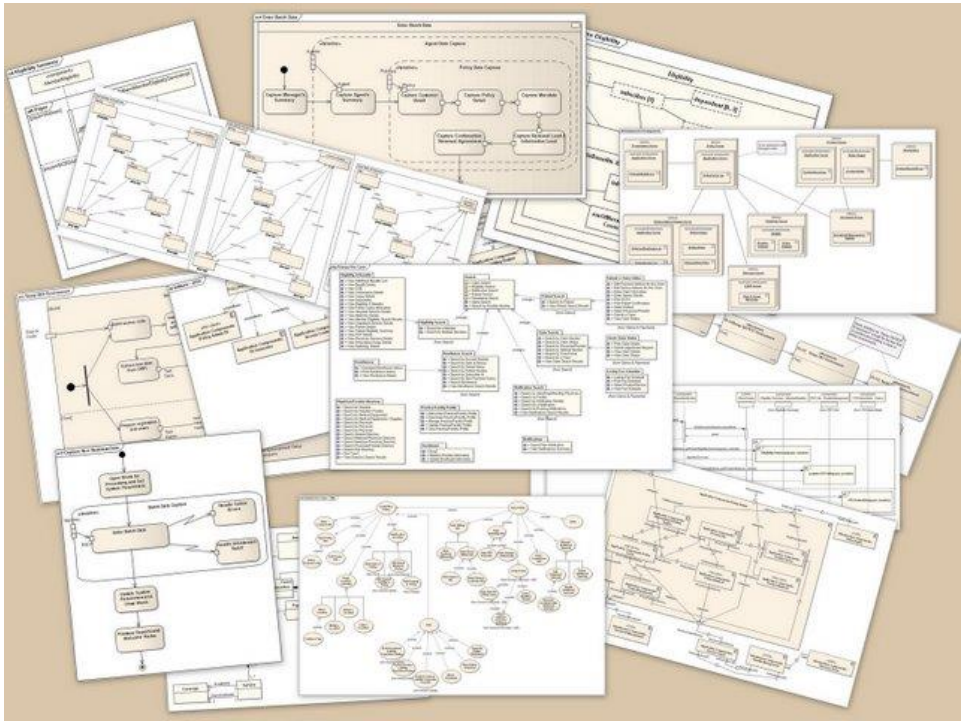
UML

UNIFIED MODELLING LANGUAGE

EXEMPLIFIED ON BLACKJACK





AGENDA

- Theoretically, before we write object-oriented code for solving a problem, we need to design an abstract model which depicts the essential features and hides all irrelevant details
- This is usually achieved by a bunch of Class Diagrams, Use-case diagrams, object diagrams, etc.
- The standard framework for doing it is called UML™:

Unified Modeling Language

- This framework deserves a full course of its own, but in this presentation we will exhibit some of its main ideas, exemplified on our Blackjack project

UML - Unified Modeling Language

- UML has nine kinds of modeling diagrams:
  - **Use-case diagrams**
  - **Class diagrams Object diagrams**
  - **Sequence diagrams**
  - **Collaboration diagrams**
  - **State-chart diagrams**
  - **Activity diagrams**
  - **Component diagrams**
  - **Deployment diagrams**
- The relation of **UML** to **OOP** is like **architect** to **builder**
  - One must have a clear architectural plan before constructing a physical building!
  - The plan should enable easy communication between both !

UML - Basics

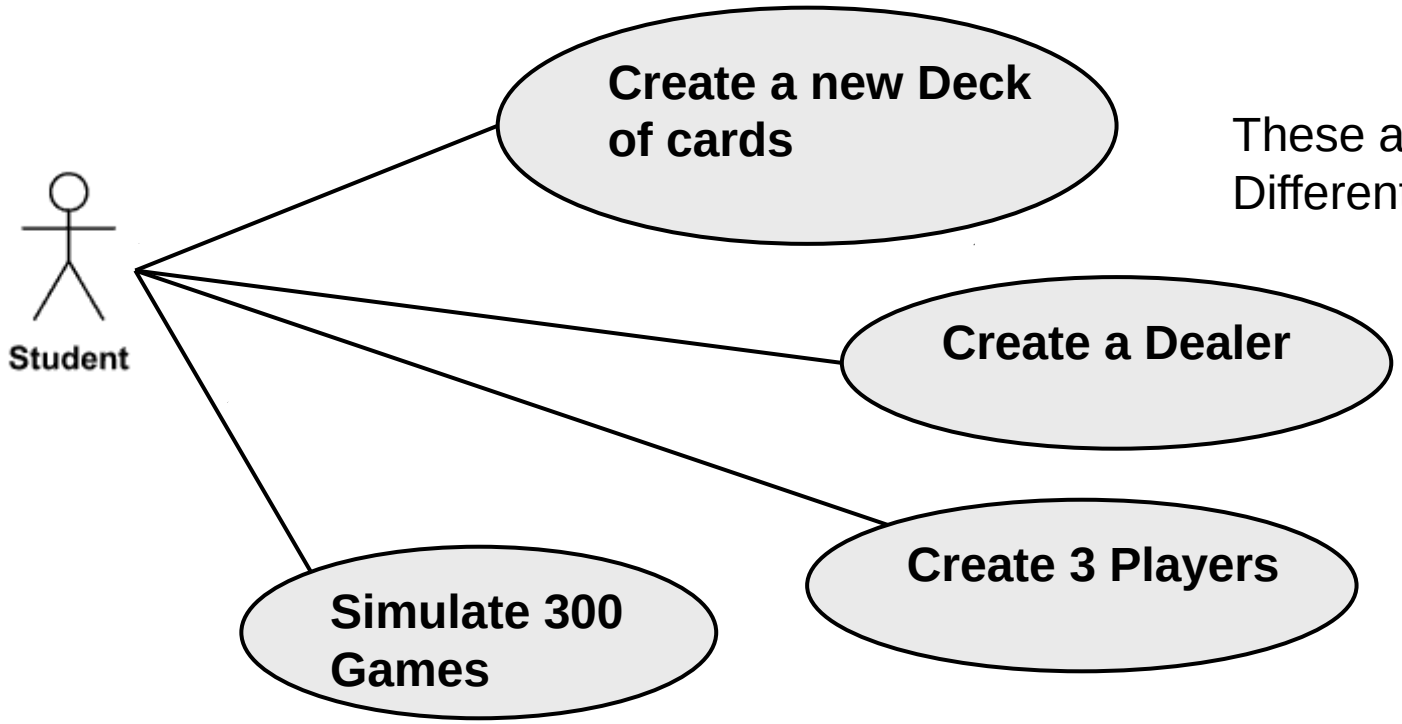
- As of today, **UML** has become the standard modelling language for software analysts, architects, and programmers
- It enables managers, clients, and programmers (in one or several teams) to communicate efficiently when designing or refactoring software systems, even if some of the participants are not professional programmers
- **UML** is best tuned to object-oriented methodologies, therefore some familiarity with **OOP** is required before learning **UML**
- A **UML** model is an abstract system which represents the objects and the relations in our problem domain
- A **UML** model consist of abstract items (“objects”) that own attributes and methods
- **UML** objects can interact with each other by “activating” each other (one object can ask another object to invoke one of his method, or simply getting one of his attributes)

UML - Basics

- A **class** is the **architectural plan** for objects of its kind
- A class consists of **attributes** (or **data members**) and **behaviors** (also called **methods**)
- Objects are also knows as **instances** of a class
- Each object of a class, contains a private set of all the attributes, which forms its **"state"** (the values of these attributes may change a lot during the object lifetime)
- The class is a static entity that is always there, while its objects may come and go (construct/destruct) during the program life
- Classes are related by several types of **association** such as inheritance, composition, and other kinds of reference of one class by another

Use-Case Diagrams

- One way to describe what a **system** does is to list many of its usage scenarios
- A **scenario** is an example of how someone is using the system (**actor**)



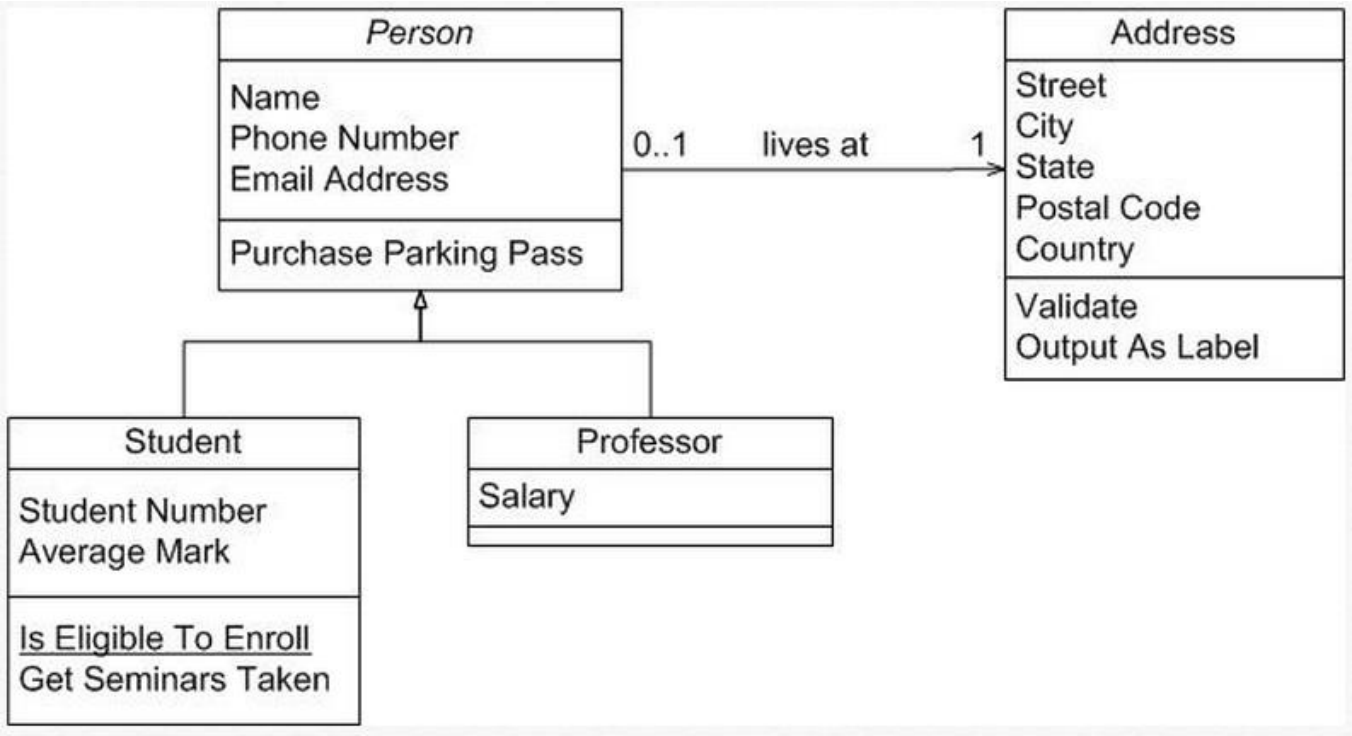
These are 4 Different use cases

Use-Case Diagrams Goals

- **Collect system requirements**
  The more use cases we have the more we know what are our system requirements
- **Use cases are great source for regression tests**
  Almost every use case can serve as a basis for one or more regression tests
- **Communication**
  Use cases are a great basis for discussions and brainstorming, between managers, tool architects, clients, developers, testers, and company accountants

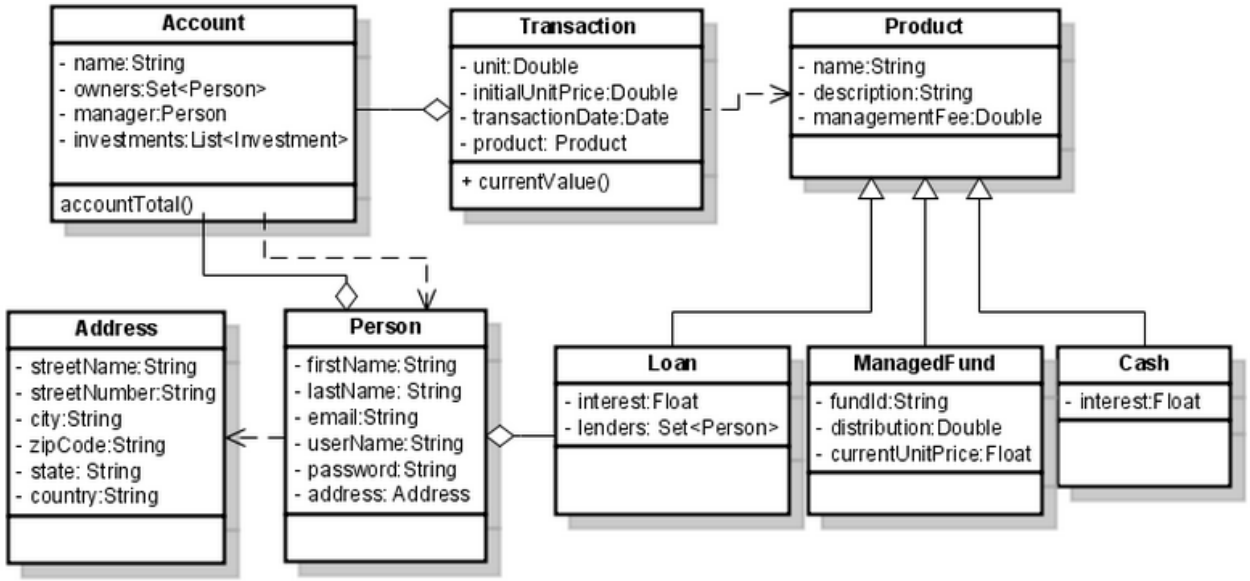
Class Diagram

- Should display all the system classes and their relations

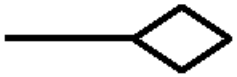


Class Diagram

- Every class in the system is represented by a box with three parts:
  - **Class name**
  - **Data members**
  - **Methods**
- Names of abstract classes are in *italics* (*Person*)
- A relation between two classes is designated by a connecting line with (normal or dashed), a special arrow, and labels (adornments)



Class Relation Types (1)

- **Association** 
  - An association is a link connecting two classes
  - Indicates that (at least) one of the two related classes makes reference to the other class
  - One class is using the other class in order to perform its work
- **Aggregation** 
  - A special type of association
  - Aggregation means that one class is contained as a member (or element) in the other class
  - The first class is called a **member** (or **element**) and the other class is called a **Container**
  - **Examples:**
    - In blackjack, a **Card** is an element of **Deck**
      A **Player** object is an element of a **Game** object
    - **List** is a container of its **members**

Class Relation Types (2)

- **Composition**
  - A stronger type of aggregation in which the element is an essential part of the container (in such case the term **"component"** is used)
  - Usually a data member that persist thru all the life cycle of the container
  - In Blackjack, a **Hand** object is a strong component of a **Player** object
  - The string object **Player.name** is an essential part of **Player**
  - Any <u>data member</u> in a class is a component of that class
  - A Blackjack Card object is not a strong component of a Deck object, since at any time it can be removed from the **Deck** and move to a Player **Hand**
- **Generalization**
  - The classical **is-a** relationship (*inheritance*) in which one class is a **superclass** of the other
  - In Blackjack, the **Dealer** class is a superclass of the **Player** class
  - This is more a relation between classes (than their objects)
  - An object of the second class **is** also an object of the first class!

Association Types

	Bidirectional Association
	Unidirectional Association
	Aggregation Element/Container relationship
	Generalization Element/Container relationship
	Generalization Element/Container relationship

http://www2.cs.uidaho.edu/~jeffery/courses/383/lecture.html

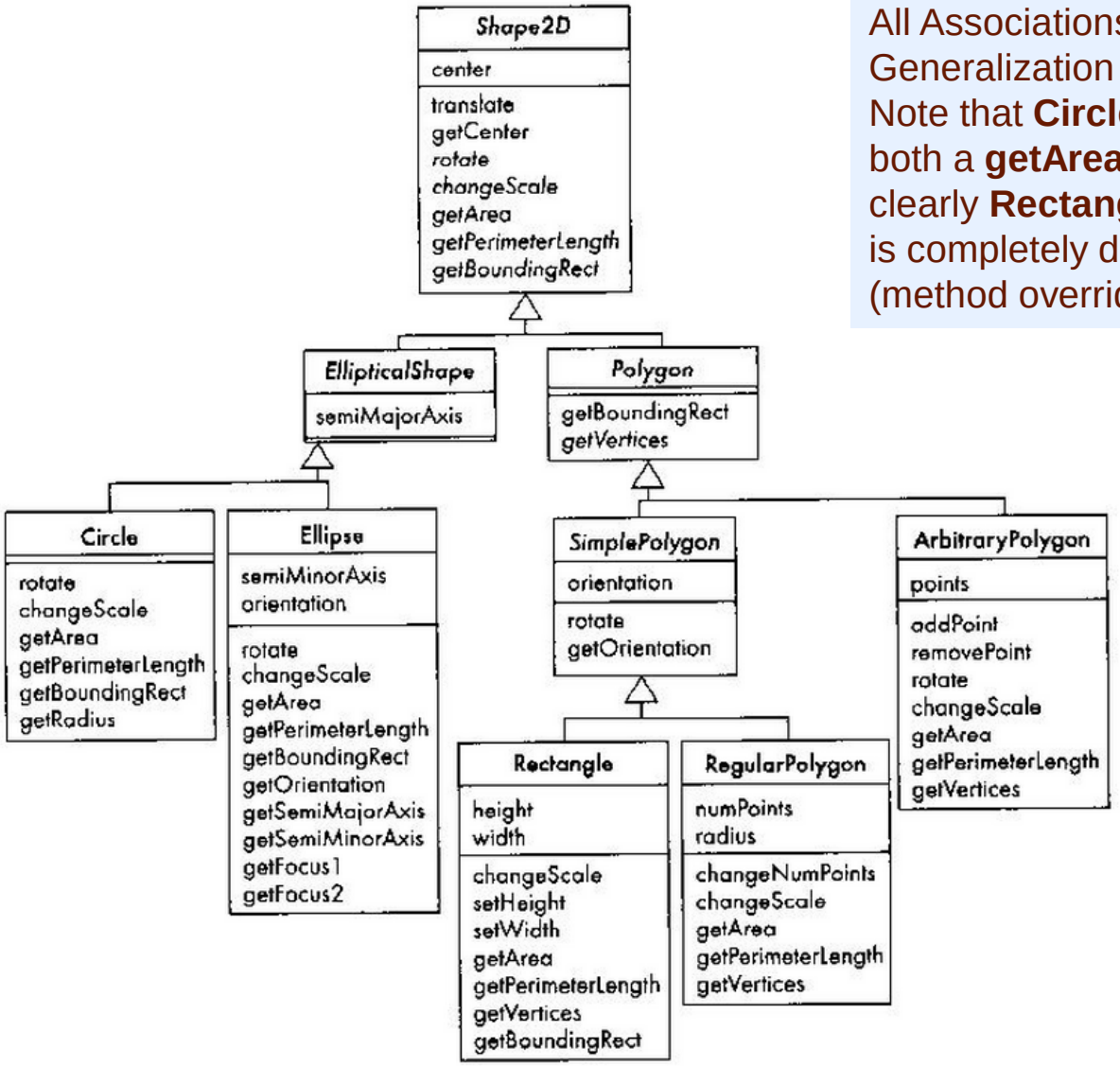
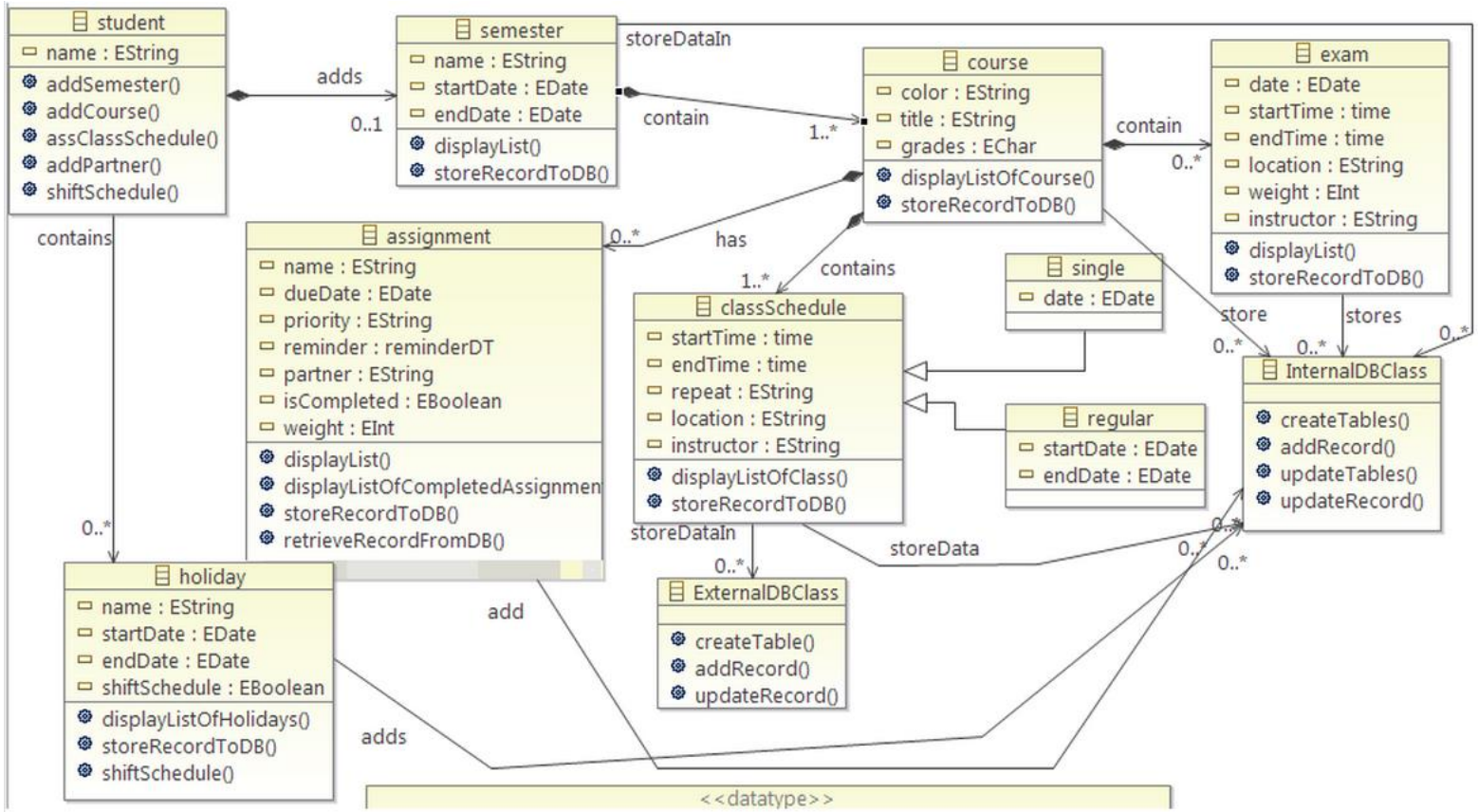


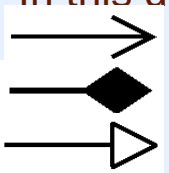
Figure 2.8 A hierarchy of shapes showing polymorphism and overriding

All Associations in this diagram are of type Generalization (inheritance)
Note that **Circle** and **Rectangle** classes have both a **getArea** method (polymorphism), but clearly **Rectangle.getArea()**
is completely different than **Circle.getArea()** (method override)



http://shikhaandroid.files.wordpress.com/2012/07/class-diagram.png

In this diagram we have three types of associations



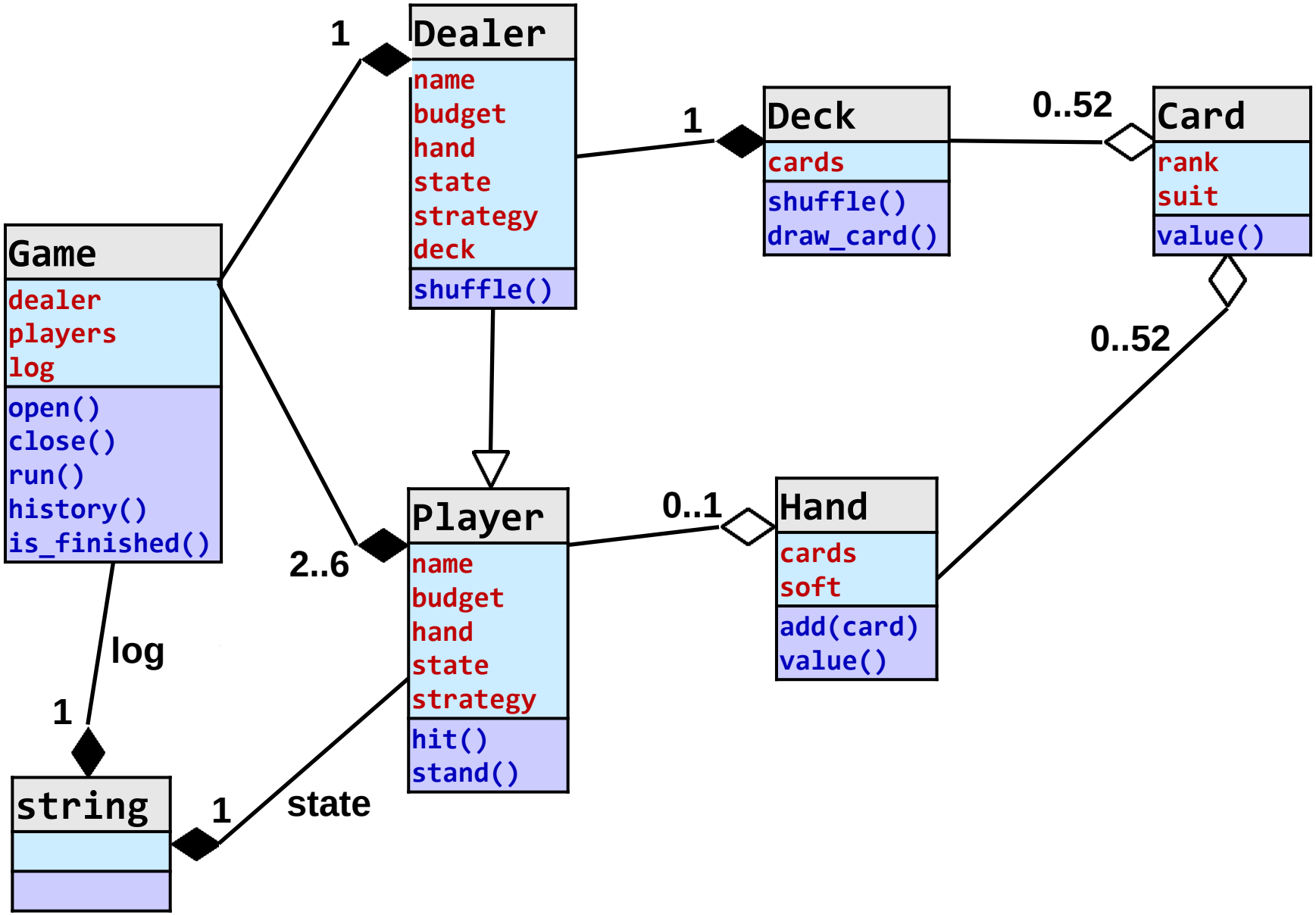
Unidirectional Association

Composition

Aggregation

Note that some association lines have two arrows!

Blackjack Class Diagram (suggestion 0.1)

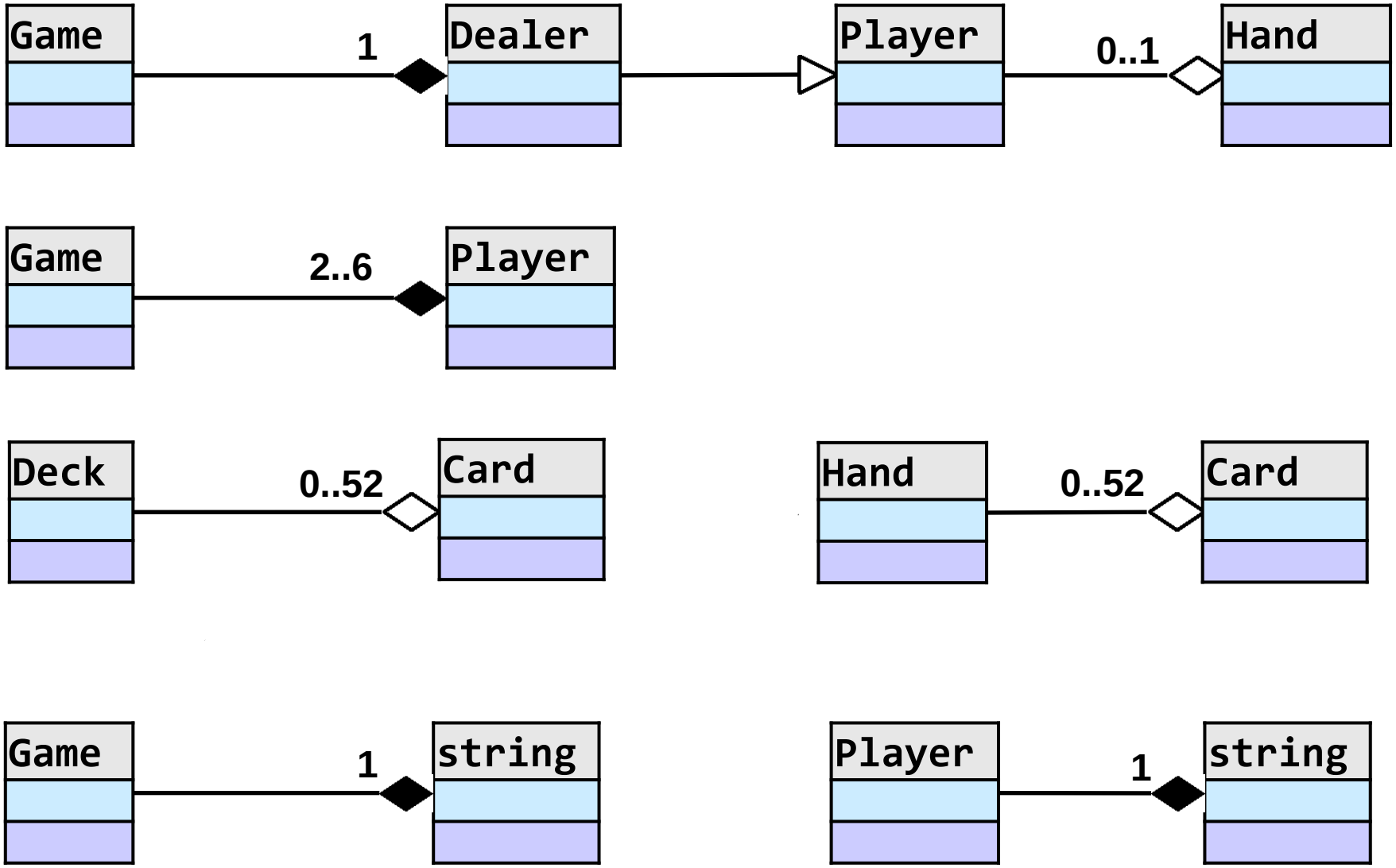


Multiplicity and Adornments

- UML association line may contain
  - An optional *arrowhead* that specifies the association type
  - Optional label at each end of line which specifies the *multiplicity* of instances of that entity (the potential number of objects that may exist in the association)
  - At each end of the line, we can add a short label ("*adornment*") which details the kind of association

Multiplicity	Meaning
5	Five instances
0..1	None or One instance
i..j	i to j instances
i..*	i or more instances (no upper limit)
0..*	Any number of instances (including none)

Class Dependency

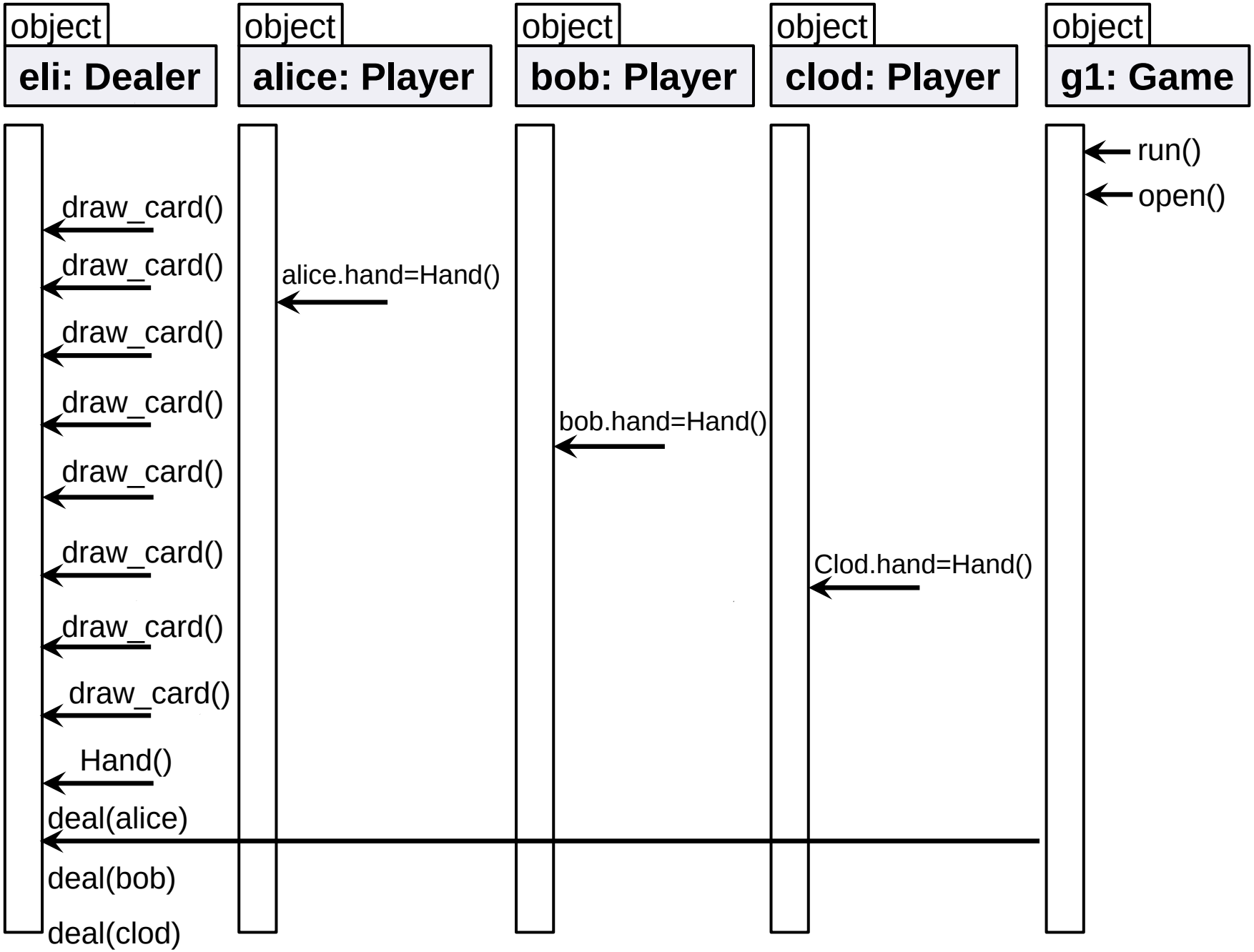


Class Dependency

- A class A is **dependent** on class B, if A is using B in one of the above relationships
- Practically, it means that class B must be implemented first, before A can do any work at all
- Dependency relations are extremely important (particularly for managers) in order to have a tidy work plan
- In real life projects, classes are usually developed by several developers, and class dependency is crucial for planning work timelines

Sequence Diagram

- While class diagram depicts a static view of our system, a **sequence diagram** is a dynamic view of the system in action
- A sequence diagram models a control flow scenario of the system arranged in a time sequence
  - Time flows from top to bottom
  - The objects involved in the scenario appear from left to right according to when they take part in the message sequence
- It consists of several objects that interact with each other within part or full life cycle (birth and death of objects)
- Sequence diagrams are associated with use case realization within the UML model of the system under development.
- Sequence diagrams are sometimes called **event diagrams**, **event scenarios**



No more space …

Resources for further study

- http://edn.embarcadero.com/article/31863
- http://en.wikipedia.org/wiki/Class_diagram
- http://en.wikipedia.org/wiki/Unified_Modeling_Language
- http://en.wikipedia.org/wiki/List_of_UML_tools