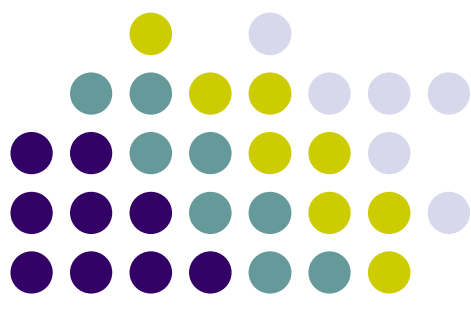
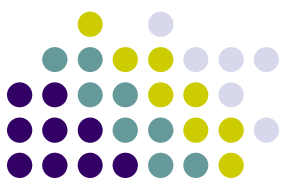


ICS 52: Introduction to Software Engineering

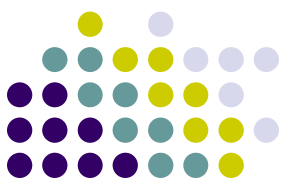
Instructor: Prof. Dan Frost
TA: Tiago Proenca
tproenca@ics.uci.edu
Oct. 29, 2008





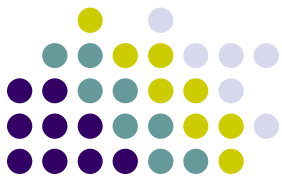
Abstractions

1. *Guidance*: an abstraction created up-front serves as roadmap for the next activity;
2. *Understanding*: an abstraction serves to explain the current state of the system at a given point in time.



UML Modeling

- General purpose programming language to create an abstract model of a system
- Modeling
 - Functional model
 - Use cases diagram (behavior diagram)
 - Object model
 - Class diagrams (what we are going to talk about, structure diagram)
 - Dynamic model
 - Sequence diagrams, state machines etc. (interaction diagram)



Class Diagrams

- A class diagram is a diagram showing a collection of classes and interfaces, along with the collaborations and relationships among classes and interfaces



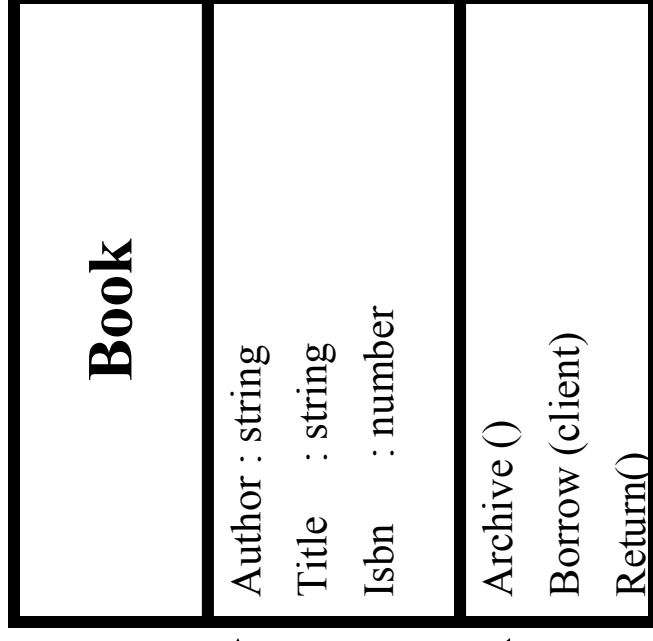
Class Diagrams

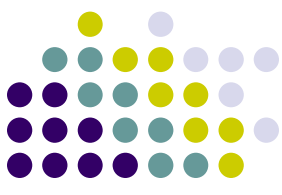
- ***Main Components:***
 - Class
 - Class name
 - Attribute
 - Operation
 - Relationships
 - Generalization
 - Association
 - Aggregation

Class Diagrams: class



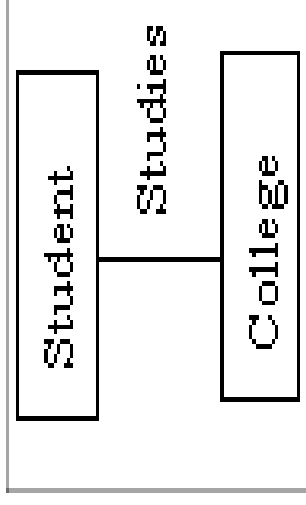
- Classes can have three parts
 - Name
 - Attributes (properties)
 - Operations (behavior)
- Classes can show visibility and types.



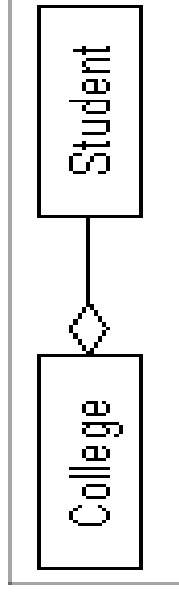


Class Diagram: Relationships

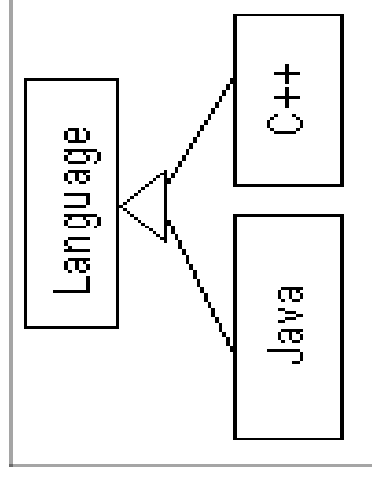
- Association



- Aggregation

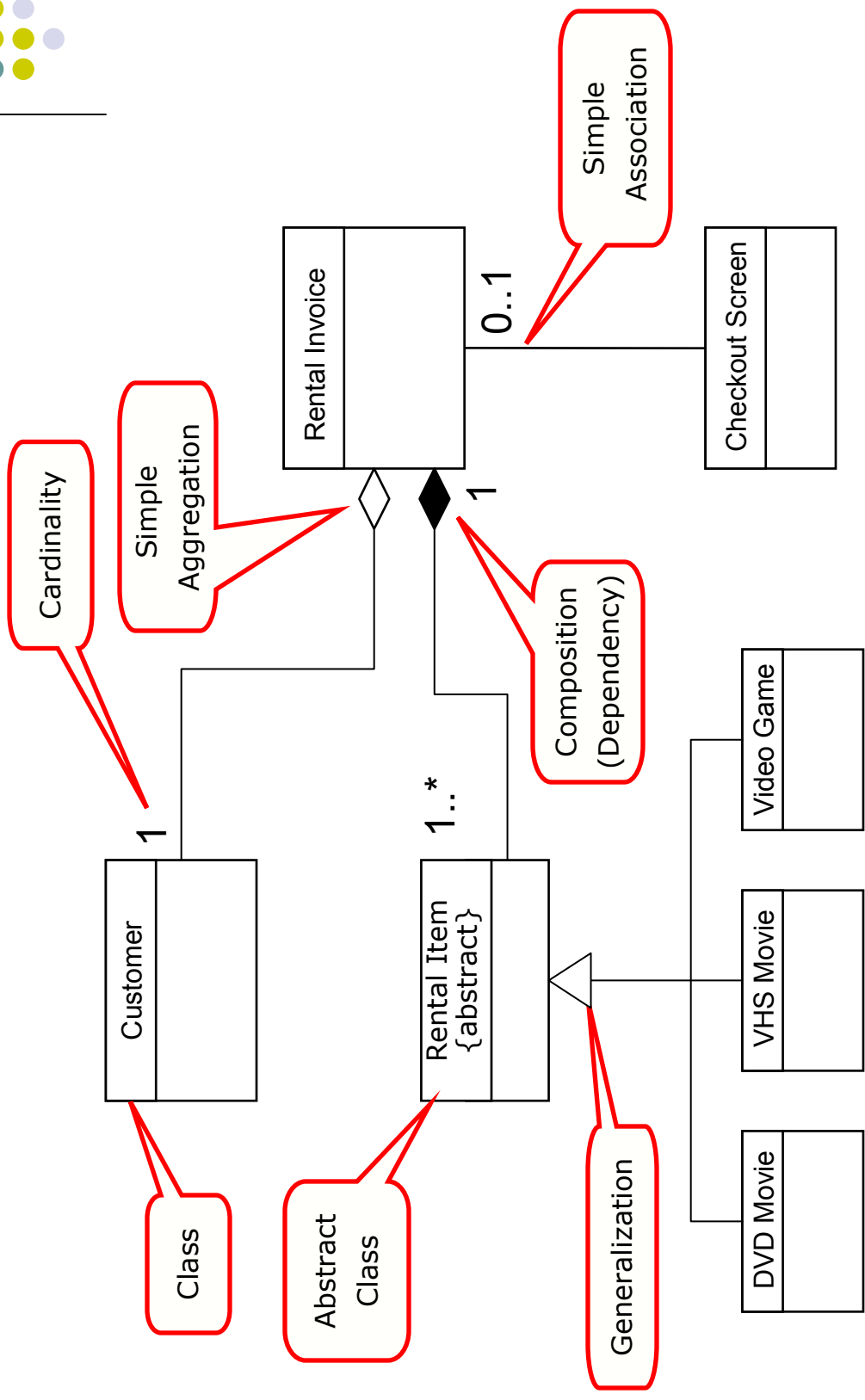


- generalization



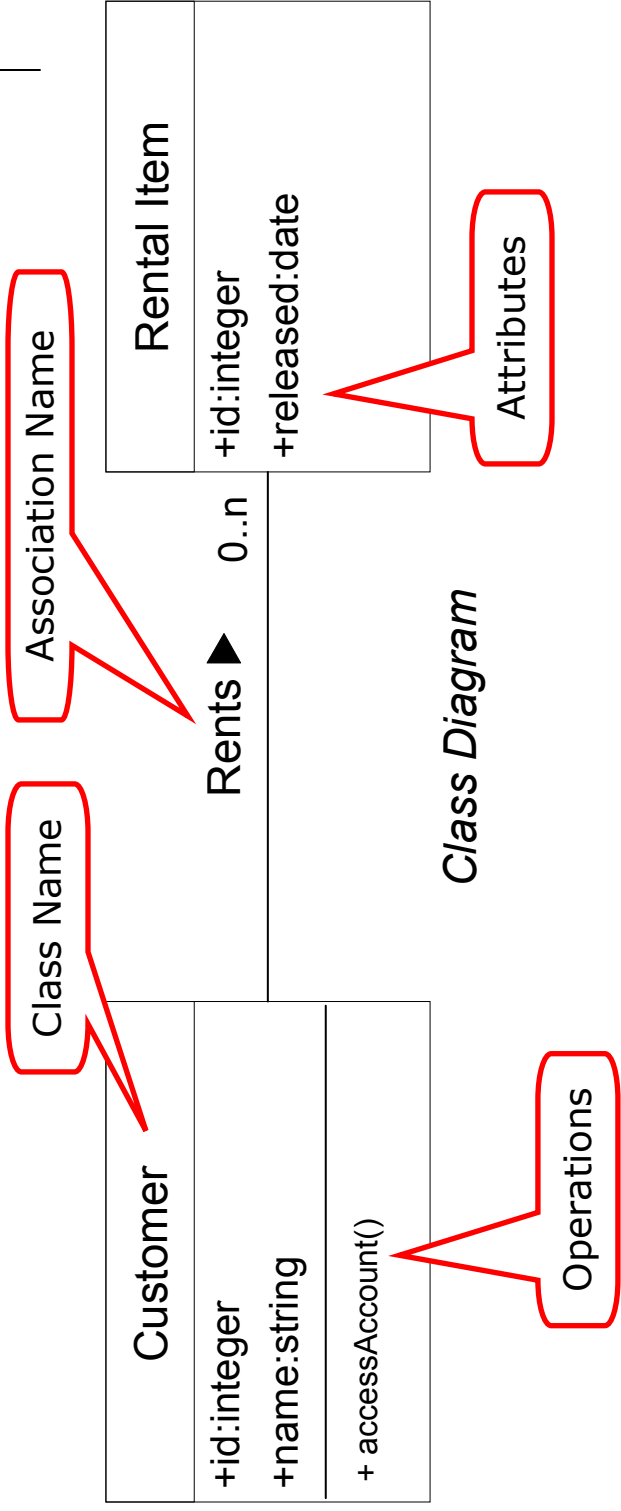


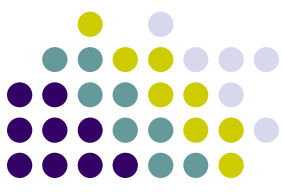
Class Diagrams



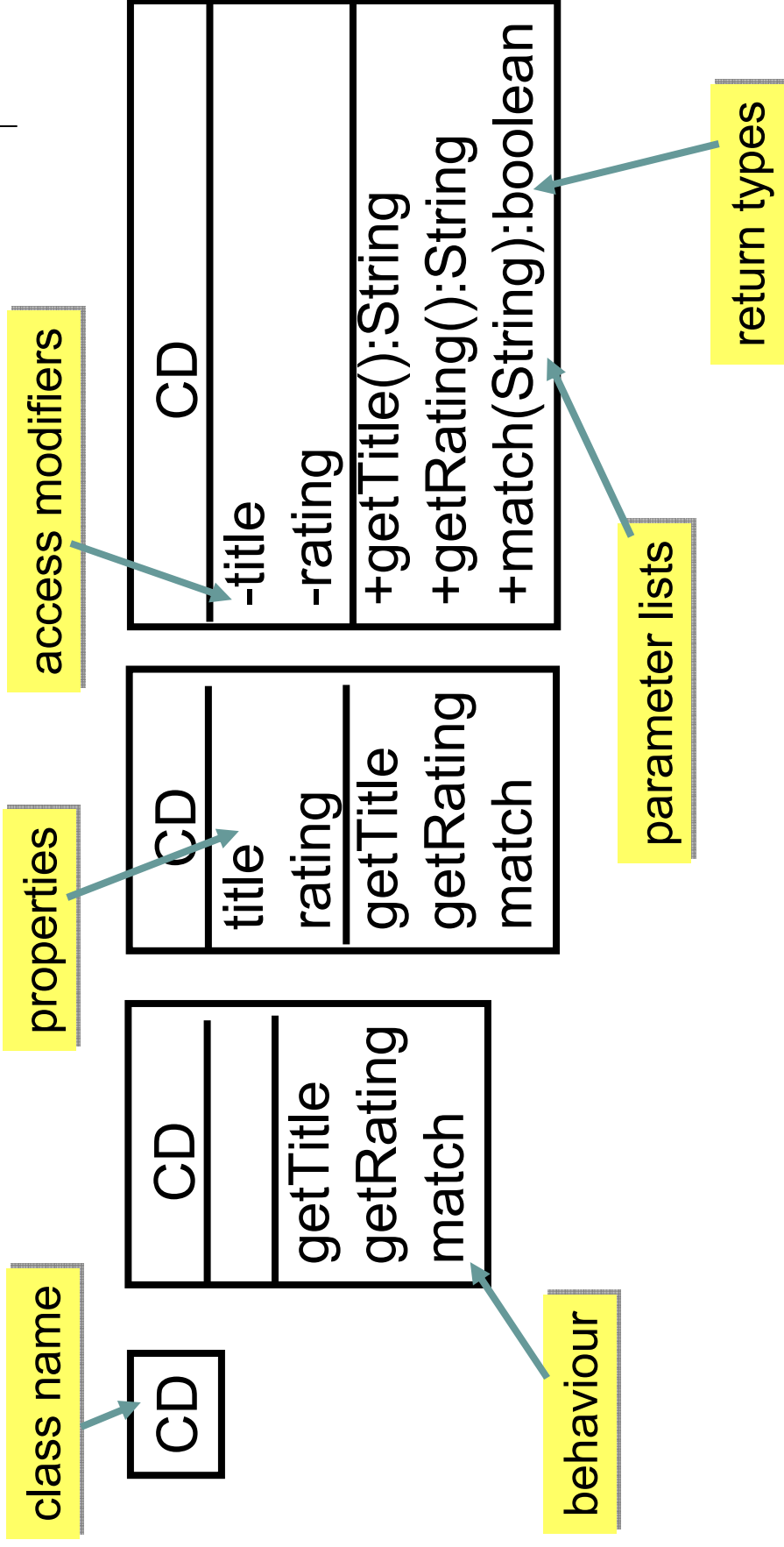


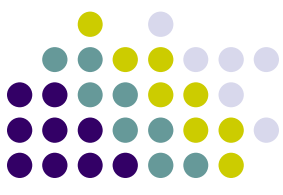
Class Diagrams





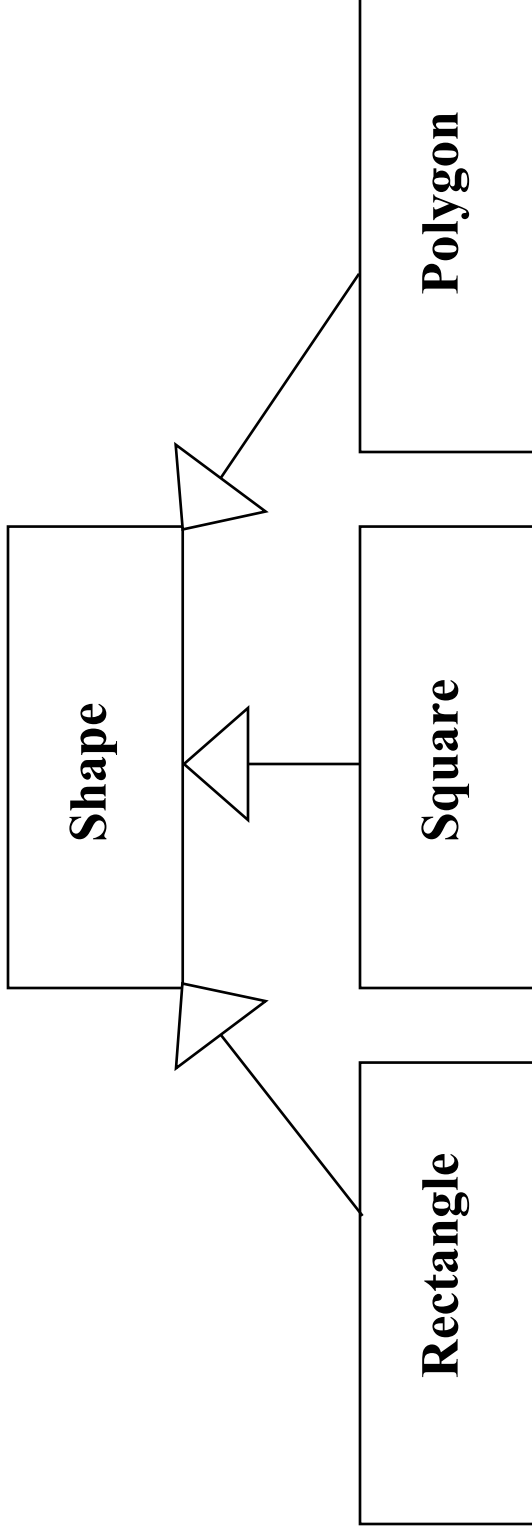
Class Diagrams

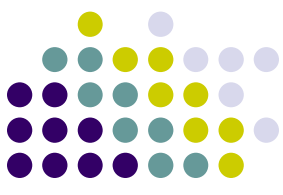




Class Diagram Relationships

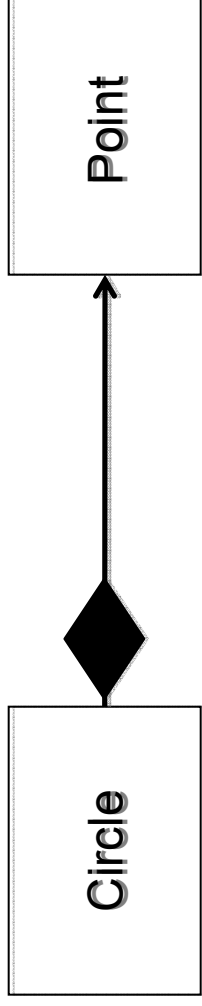
- **Generalization**
 - A relationship between a general thing and a more specific kind of that thing.
 - It is **a-kind-of** relationship

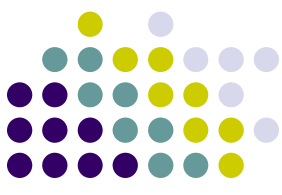




Class Diagram Relationships

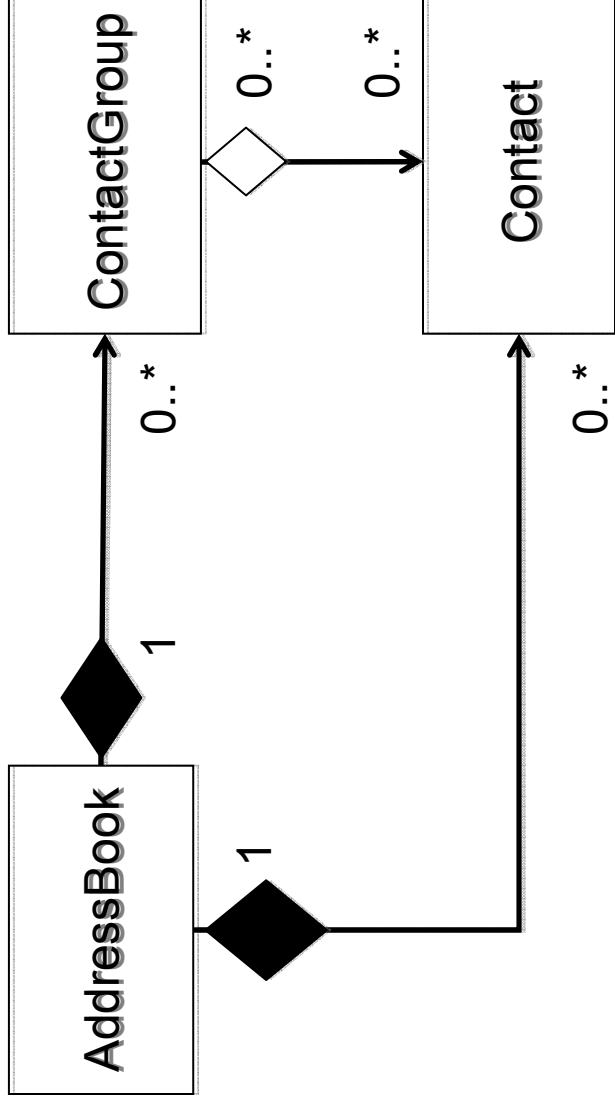
- **Composition**
 - Combine simple objects into more complex objects
 - It's a **whole/part** relationship.
 - Diamond filled means the Point lifecycle depends upon Circle.

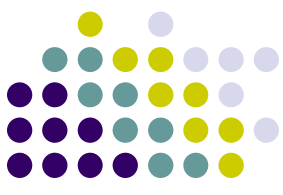




Class Diagram Relationships

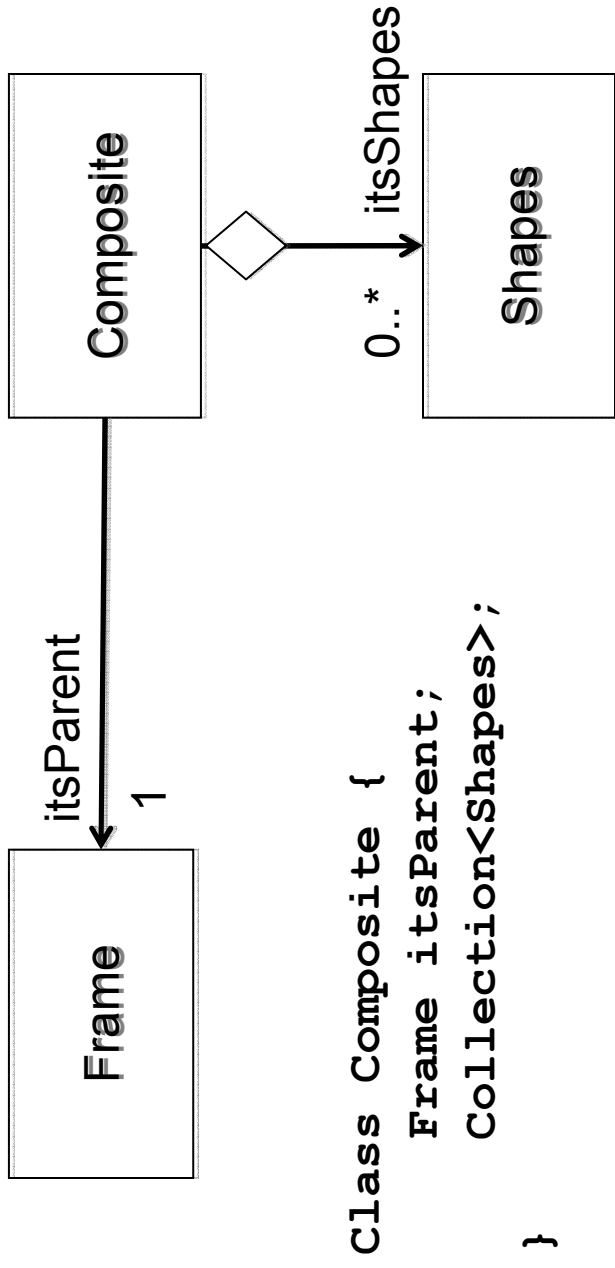
- Aggregation
 - Combine simple objects into more complex objects
 - It's a **weak whole/part** relationship.
 - Diamond not filled means the Contact lifecycle does not depends upon ContactGroup.

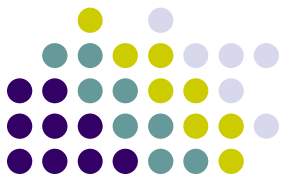




Class Diagram Relationships

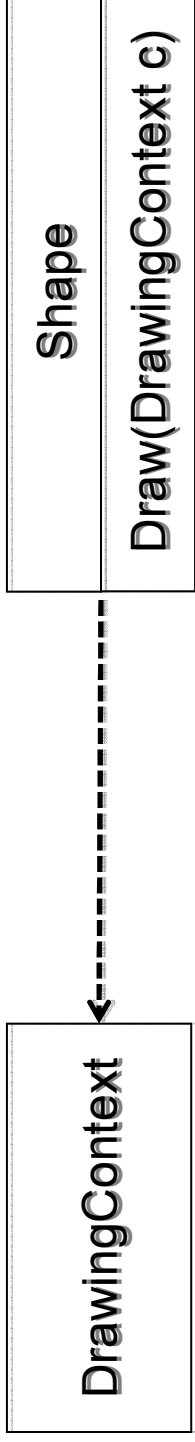
- Association
 - Specifies that one object is connected with other;
 - It's not a whole/part relationship;
 - Weak relationship;



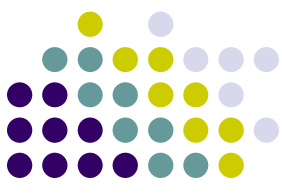


Class Diagram Relationships

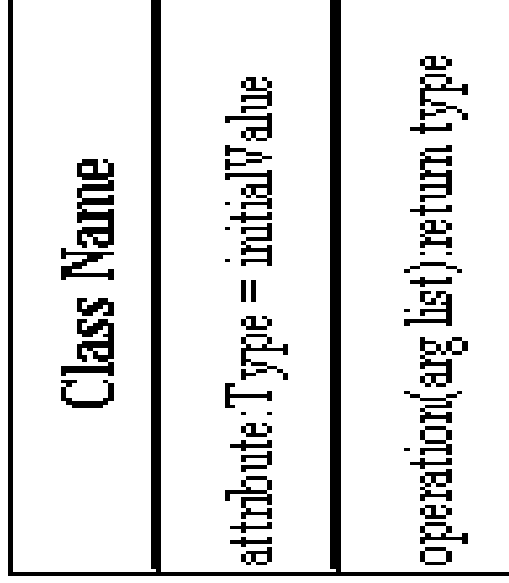
- **Dependency**
 - Very weak dependency;
 - Not implemented with member variables at all;
 - E.g., parameter variable, local variable



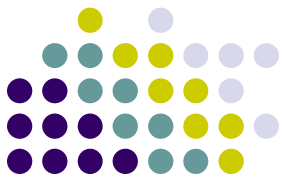
Basic Class Diagram Symbols and Notations



- **Classes**
 - Illustrate classes with rectangles divided into compartments.
 - Place the name of the class in the first partition (centered, bolded, and capitalized)
 - list the attributes in the second partition
 - write operations into the third.



Basic Class Diagram Symbols and Notations

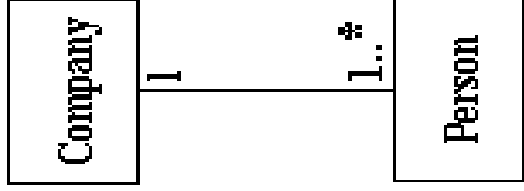
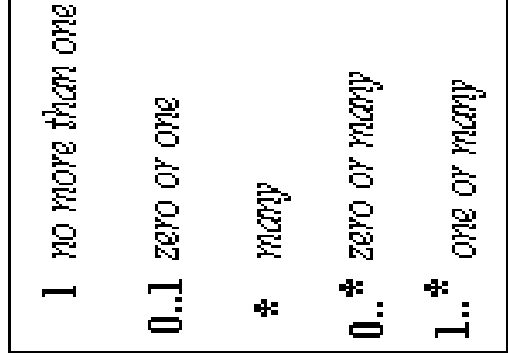
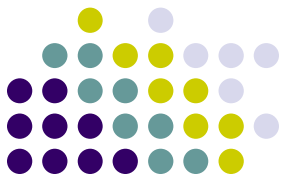


+	<i>public</i>
-	<i>private</i>
#	<i>protected</i>

Class Name
- attribute
- attribute
+ operation
+ operation
+ operation

- **Visibility**
 - Use visibility markers to signify who can access the information contained within a class.
 - Private visibility hides information from anything outside the class partition.
 - Public visibility allows all other classes to view the marked information.
 - Protected visibility allows child classes to access information they inherited from a parent class.

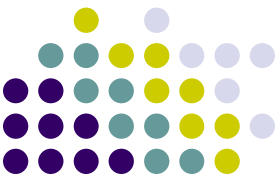
Basic Class Diagram Symbols and Notations



- **Multiplicity (Cardinality)**
 - Place multiplicity notations near the ends of an association.
 - These symbols indicate the number of instances of one class linked to *one* instance of the other class.
 - For example, one company will have one or more employees, but each employee works for one company only.

LibFiles Example

(On the white board)





References

- Van der Westhuizen, C., Chen, P.H., van der Hoek, A.: Emerging design: New roles and uses for abstraction. In: ROA '06: Proceedings of the 2006 International Workshop on Role of Abstraction in Software Engineering, New York, NY, USA, ACM (2006) 23–28