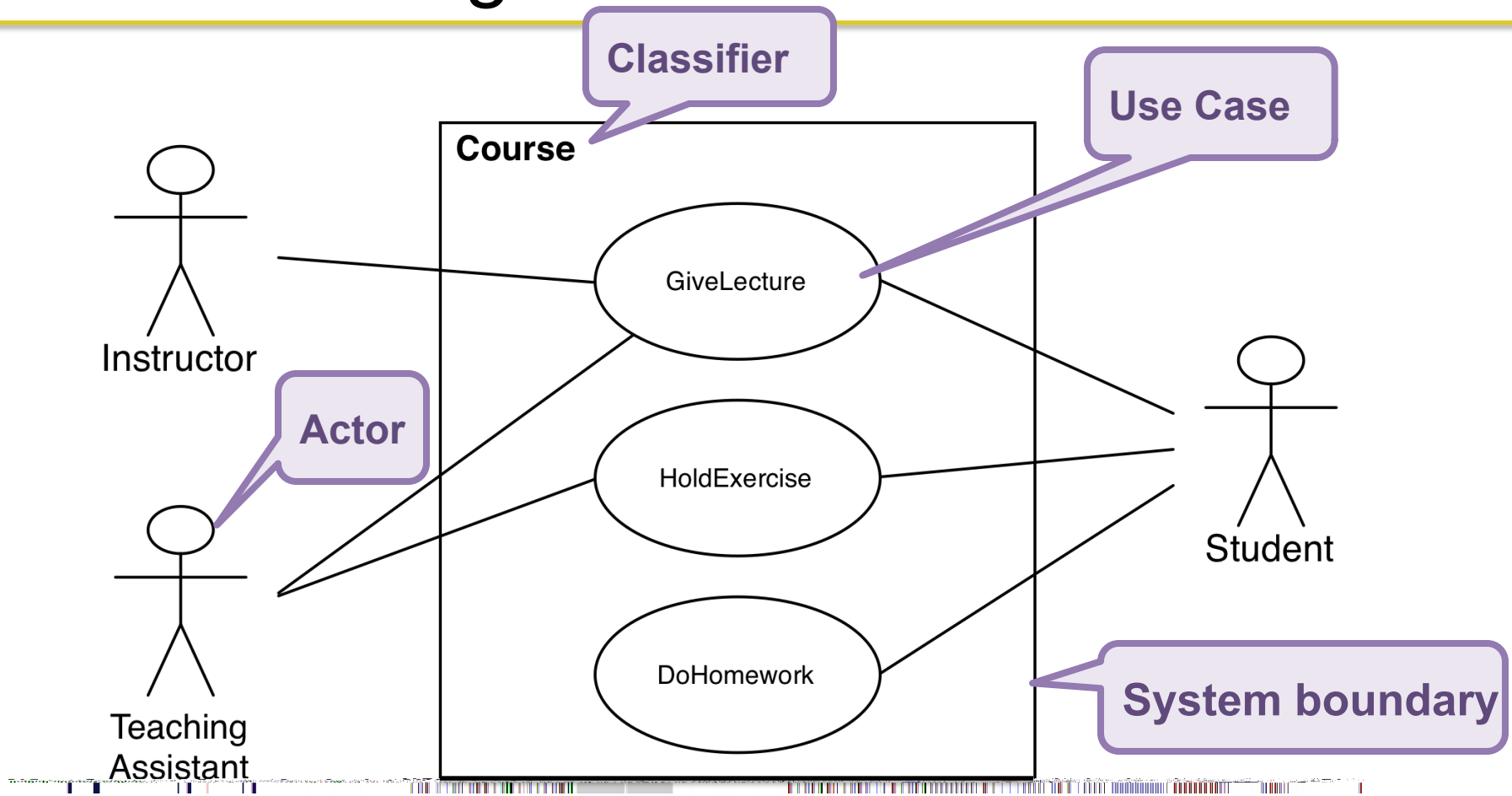
UML Summary

CSCI-4448 - Boese

University of Colorado **Boulder**

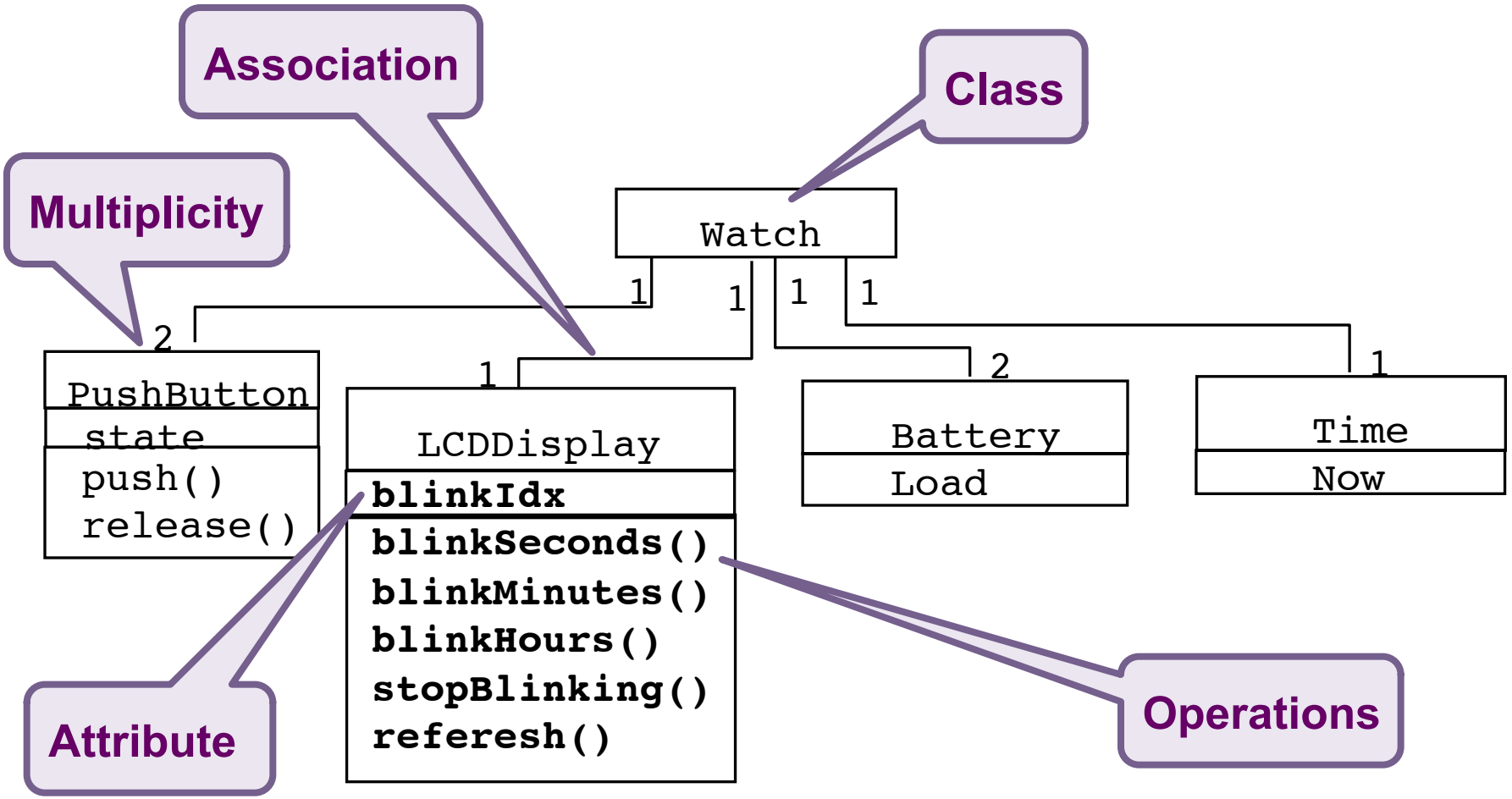
Diagram Review

Use case diagrams



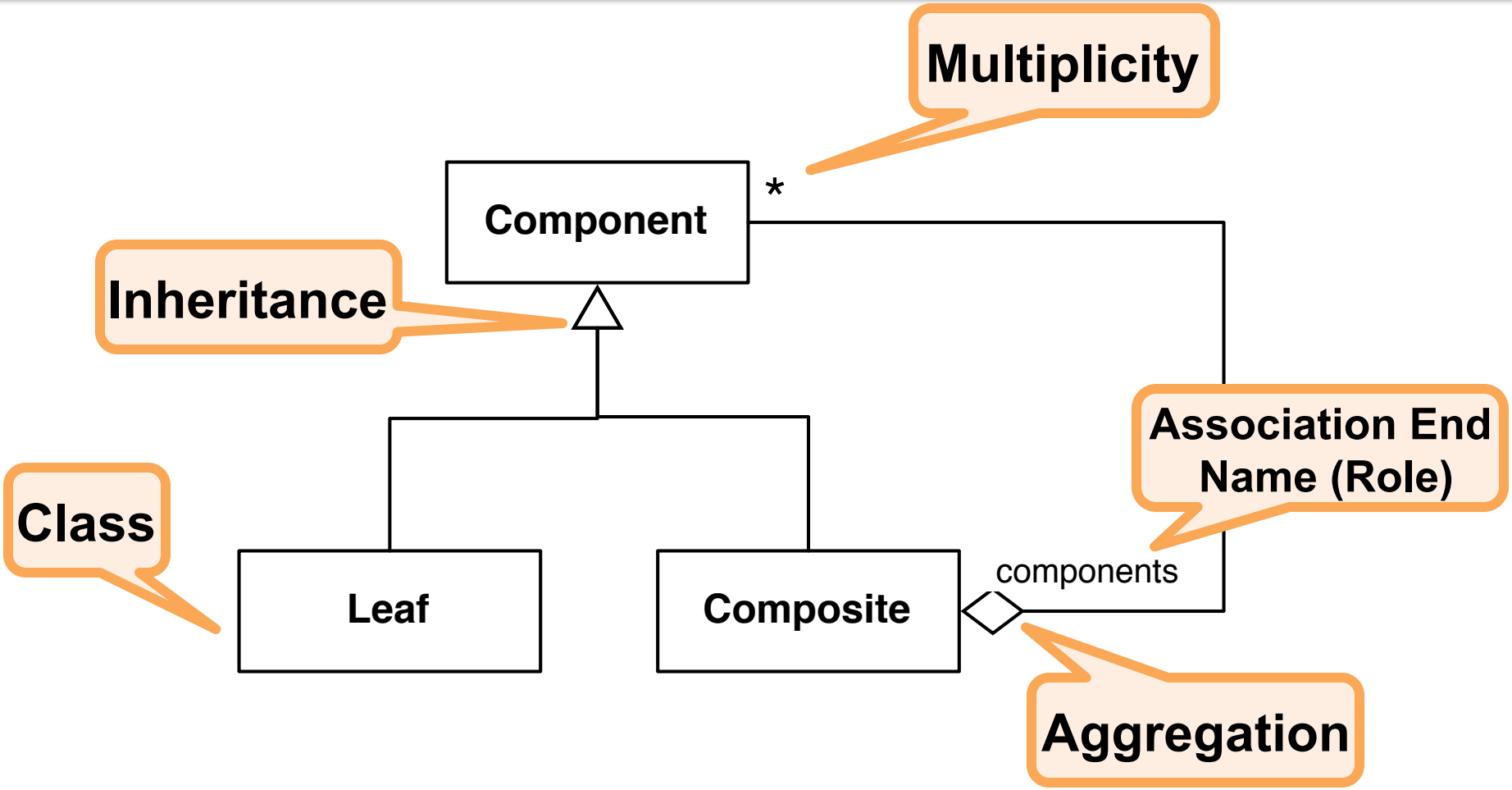
Use case diagrams represent the functionality of the system from the user's point of view

Class diagrams



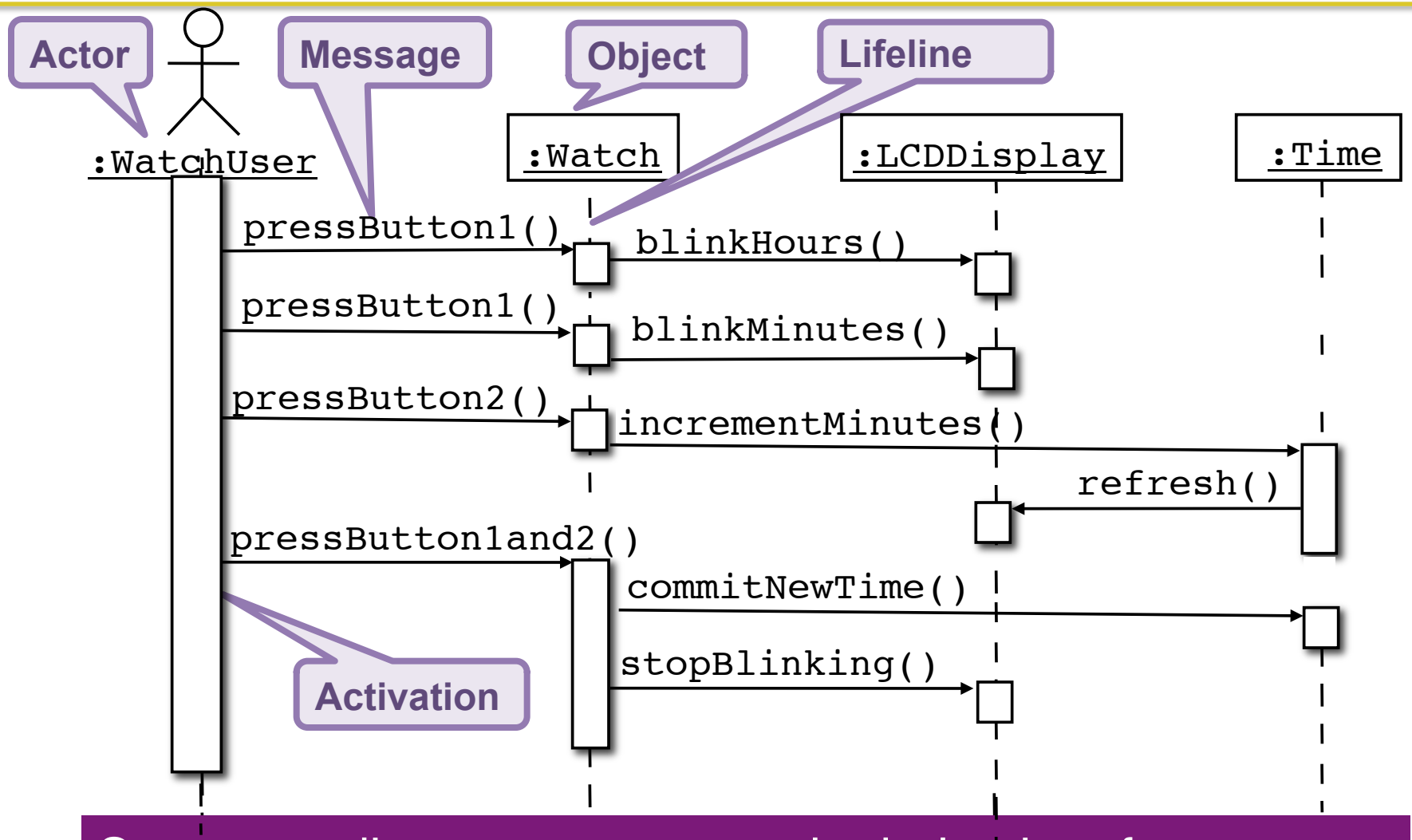
Class diagrams represent the structure of the system

Review of Class Diagrams



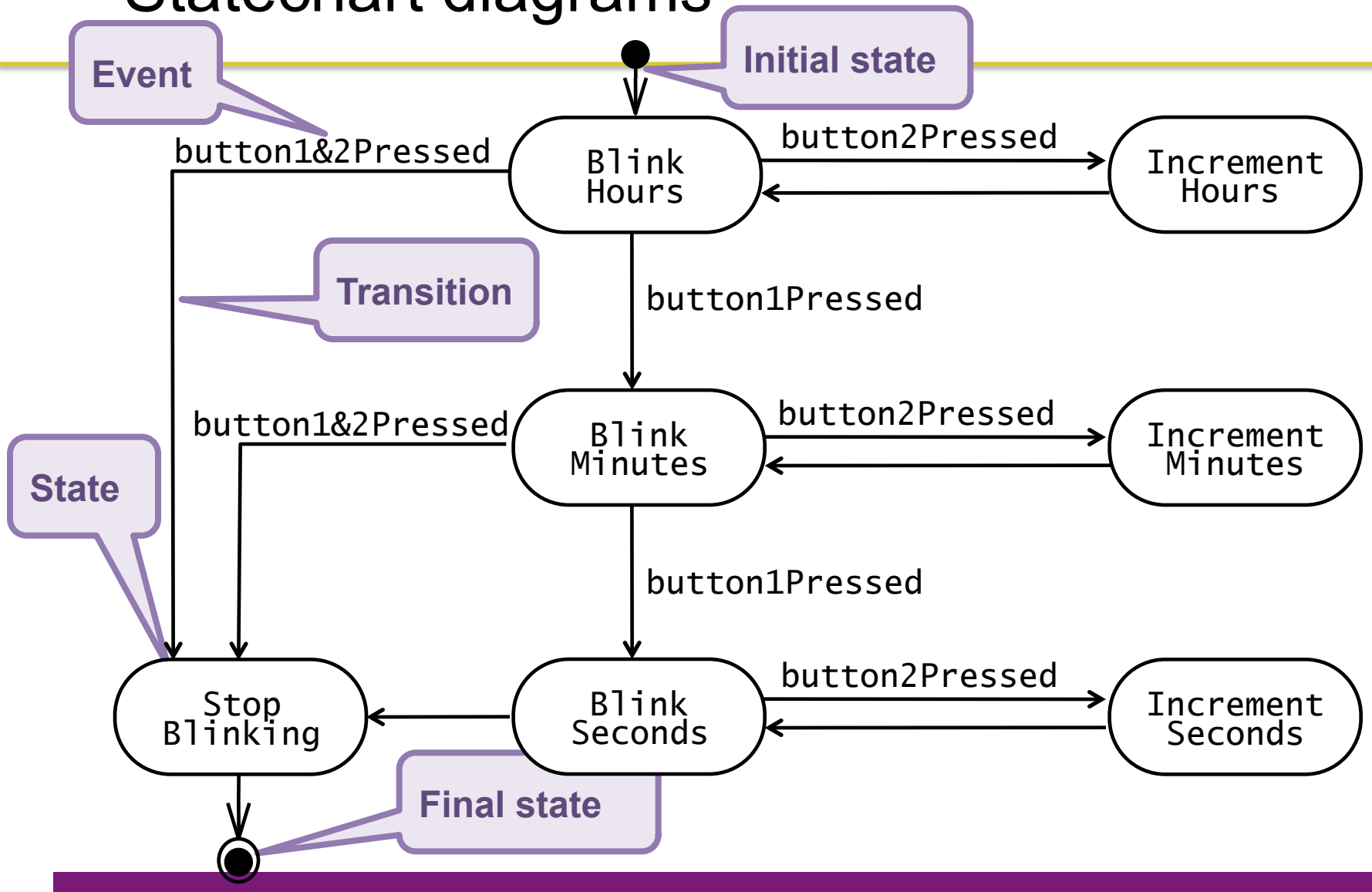
Class diagrams represent the structure of the system

Sequence diagram



Sequence diagrams represent the behavior of a system as messages ("interactions") between *different objects*

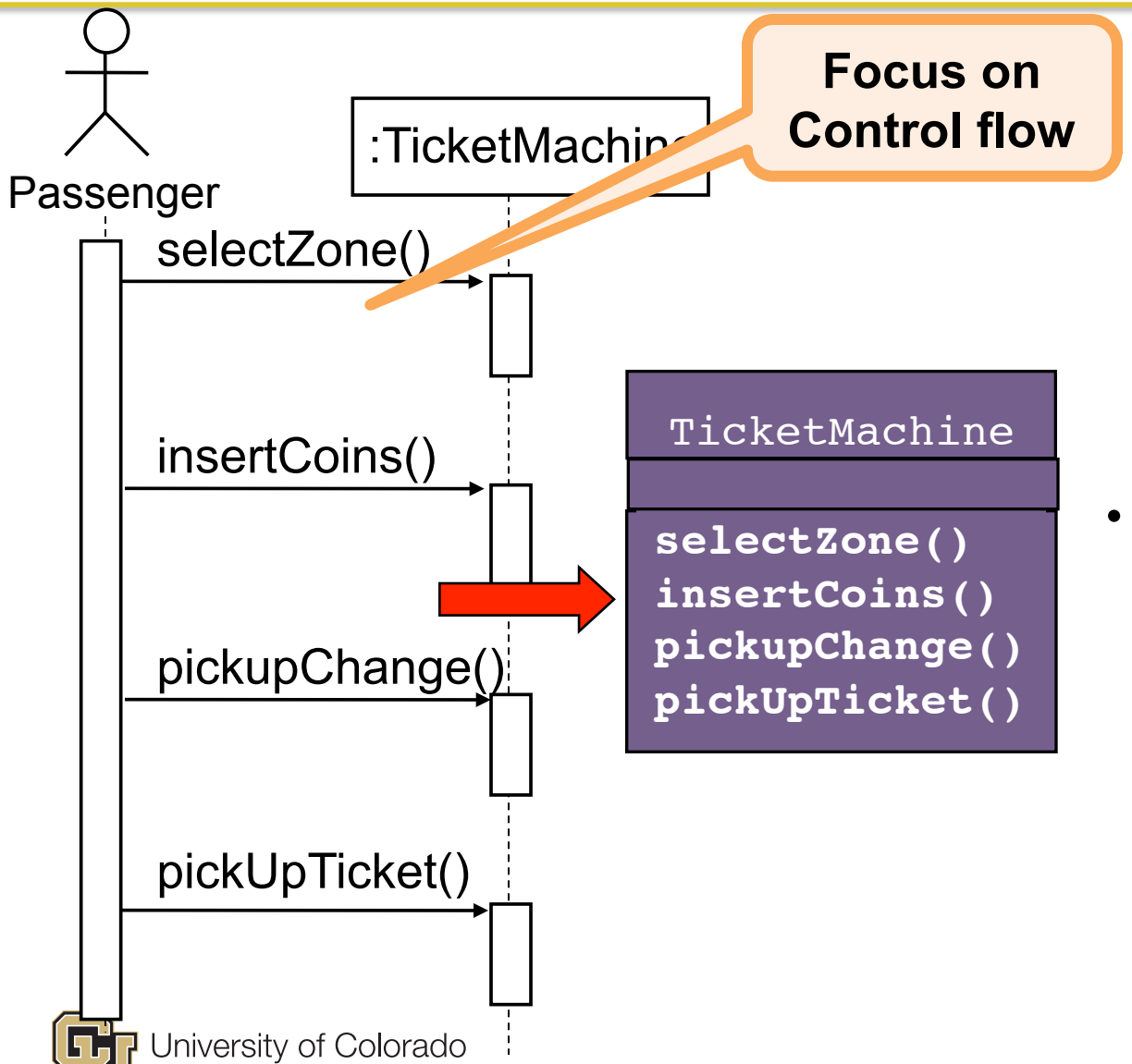
Statechart diagrams



Represent behavior of *a single object* with interesting dynamic behavior.

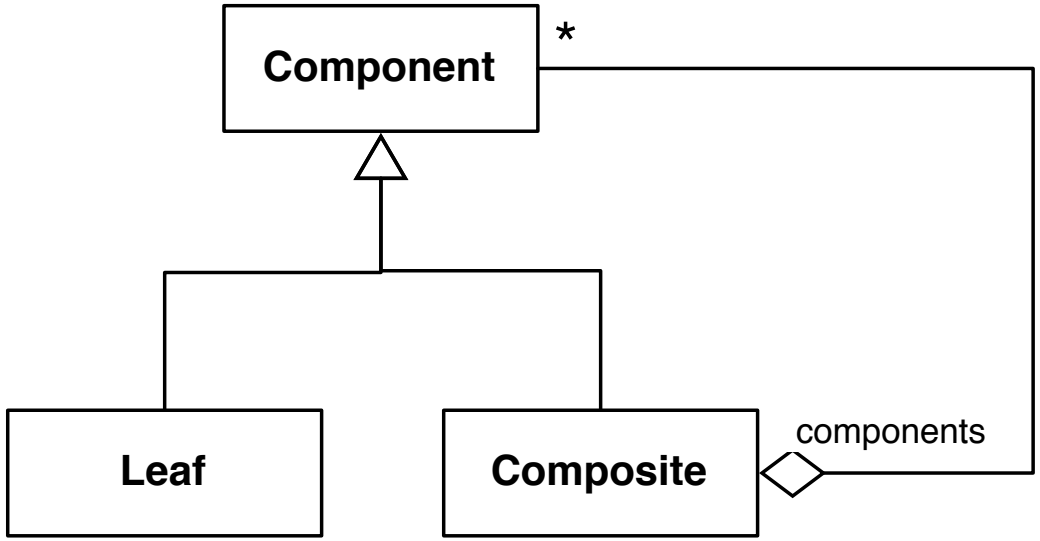
Relations between diagrams

Use Cases – Sequence D. – Class D.



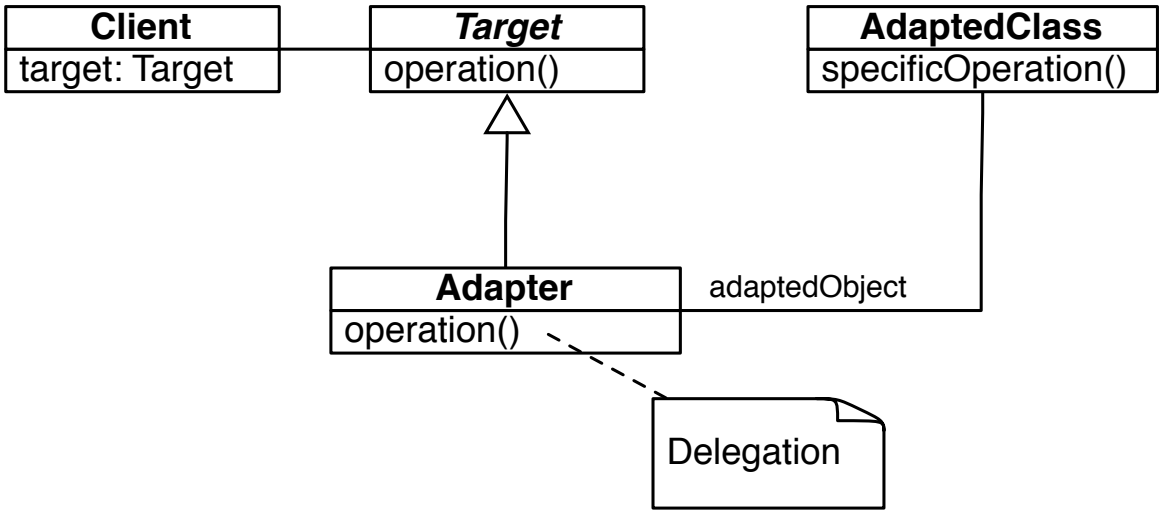
- Used during analysis
  - To refine use case descriptions
  - Find additional objects ("participating objects")
- Used during system design
  - Refine subsystem interfaces

Code Generation from UML to Java




```
public class Component {  }
public class Leaf extends Component {  }
public class Composite extends Component
{
    private Collection<Component> components;
    …
}
```

Code Generation from UML to Java




```
public abstract class Target
{
    public … operation();
}
public class Adapter extends Target
{
    private AdaptedClass adaptedObject;
    public … operation()
    {
        adaptedObject.specificOperation();
    }
```

Comparing diagrams

Differences Between Diagrams

- **Activity Diagrams** are an important tool whenever business workflow is important
- Ultimately, **use cases** usually provide the authoritative view of the bunches of requirements, especially when their scope is automation of business activities.
- Which one or both?
  - Picture thinkers vs. text thinkers
  - May need both techniques for some teams, but try to avoid redundancy
  - When an activity diagram gets very detailed, it may be time to switch to text.

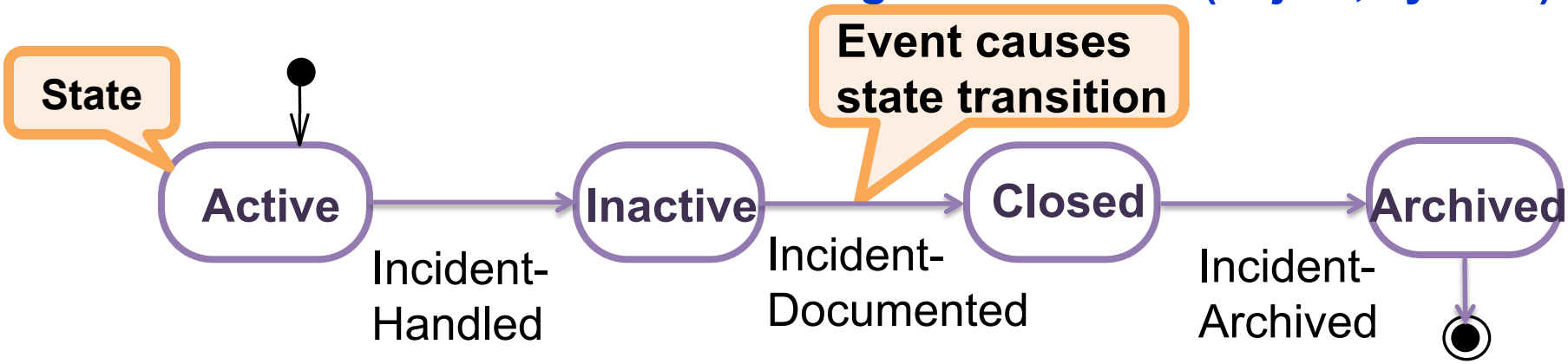
Differences Between Diagrams

- Use Case diagrams are fundamentally different from Sequence diagrams or flow charts because they do not make any attempt to represent the order or number of times that the systems actions and sub-actions should be executed.
- UCDs do not indicate data flows (shown in sequence diagrams).
- Each use-case is described further by textual document and by scenarios developed using UML sequence diagrams

Activity Diagram vs. Statechart Diagram

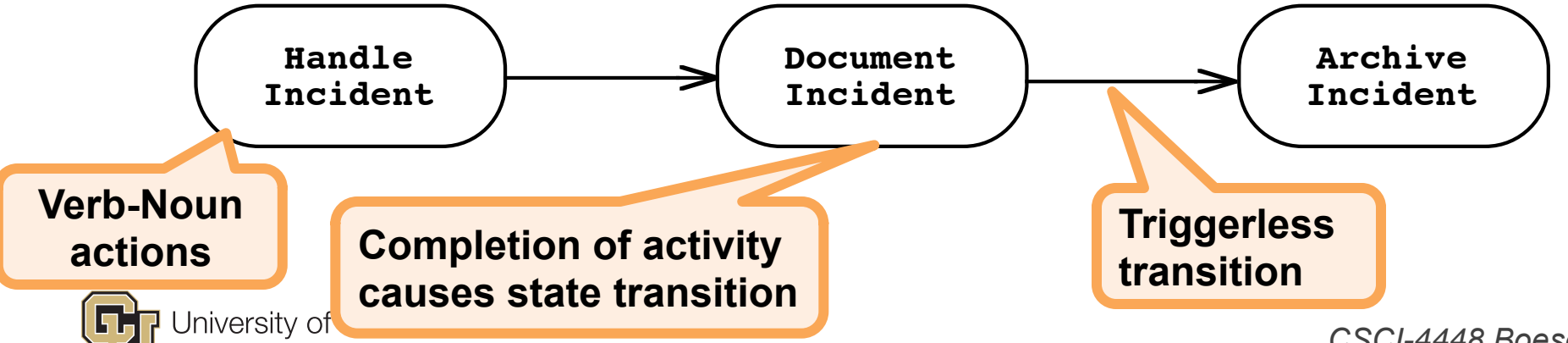
Statechart Diagram for Incident

Focus on the set of attributes of a single abstraction (object, system)



Activity Diagram for Incident

Focus on dataflow in a system



State Machine vs. Activity Diagram

State Machine

- Models behavior from the perspective of an object
  - *React* to various events, regardless of the source
- Component-level view of the system

Activity Diagram

- Describes interactions among the system and several actors
  - May include several objects / subsystems
- High-level / business logic view of the system

State Machine vs. Sequence Diagram

State Machine

- Demonstrates how individual objects make some decision
- Provides information on state of object
- Object may be involved in *multiple* sequence diagrams
  - Ensure non-conflicting behavior

Sequence Diagram

- Demonstrates how individual objects interact to achieve a goal
  - Focus on messages and communication
- Provides no information on object states
- Provides information on control
  - Loops, decisions, etc.

Verify and Validate

Model Validation and Verification

- Verification is an equivalence check between the transformation of two models
- Validation is the comparison of the model with reality
  - Validation is a critical step in the development process.
  - Requirements should be validated with the client and the user.
  - Techniques: Formal and informal reviews (Meetings, requirements review)
- Requirements validation involves several checks
  - Correctness, Completeness, Ambiguity, Realism

Checklist for a Requirements Review

- Is the model correct?
  - A model is correct if it represents the client's view of the system
- Is the model complete?
  - Every scenario is described
- Is the model consistent?
  - The model does not have components that contradict each other
- Is the model unambiguous?
  - The model describes one system, not many
- Is the model realistic?
  - The model can be implemented

Examples for Inconsistency and Completeness Problems

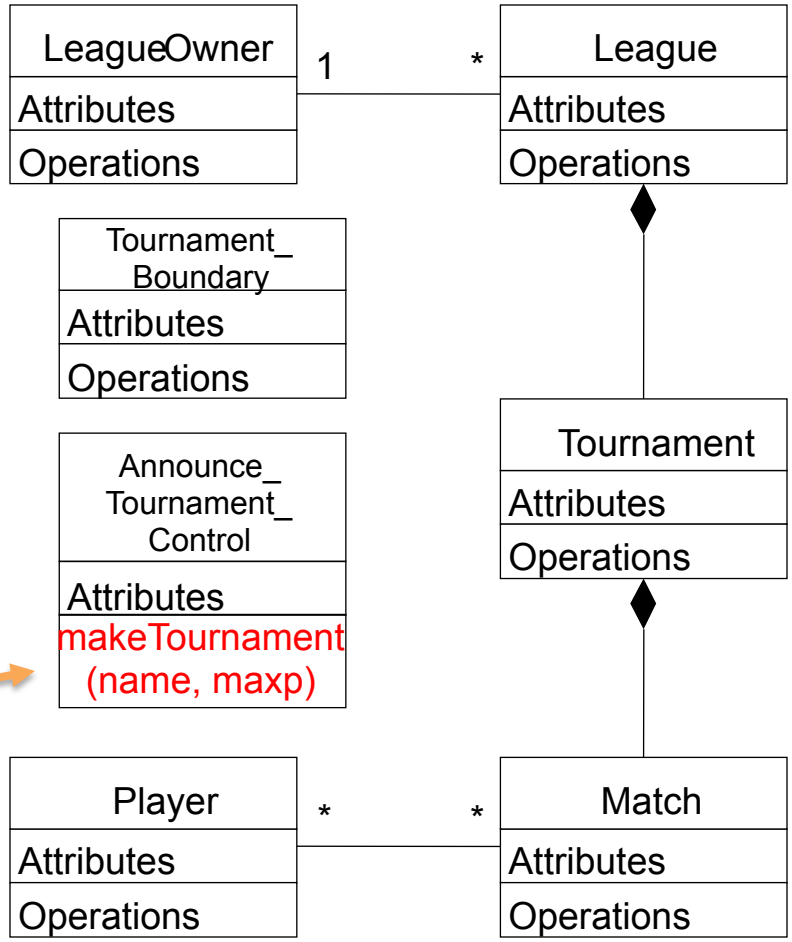
- Inconsistency
  - Classes with the same name but different meanings
  - Different spellings in different diagrams
    - *Class*
    - *Attribute*
    - *Method*
- Omissions in diagrams
  - Class
  - Attribute
  - Method
  - Missing use case for provided methods
  - Classes that are disconnected from associated classes
  - Identify dangling associations (“pointing to nowhere”)

Different spellings in different UML diagrams

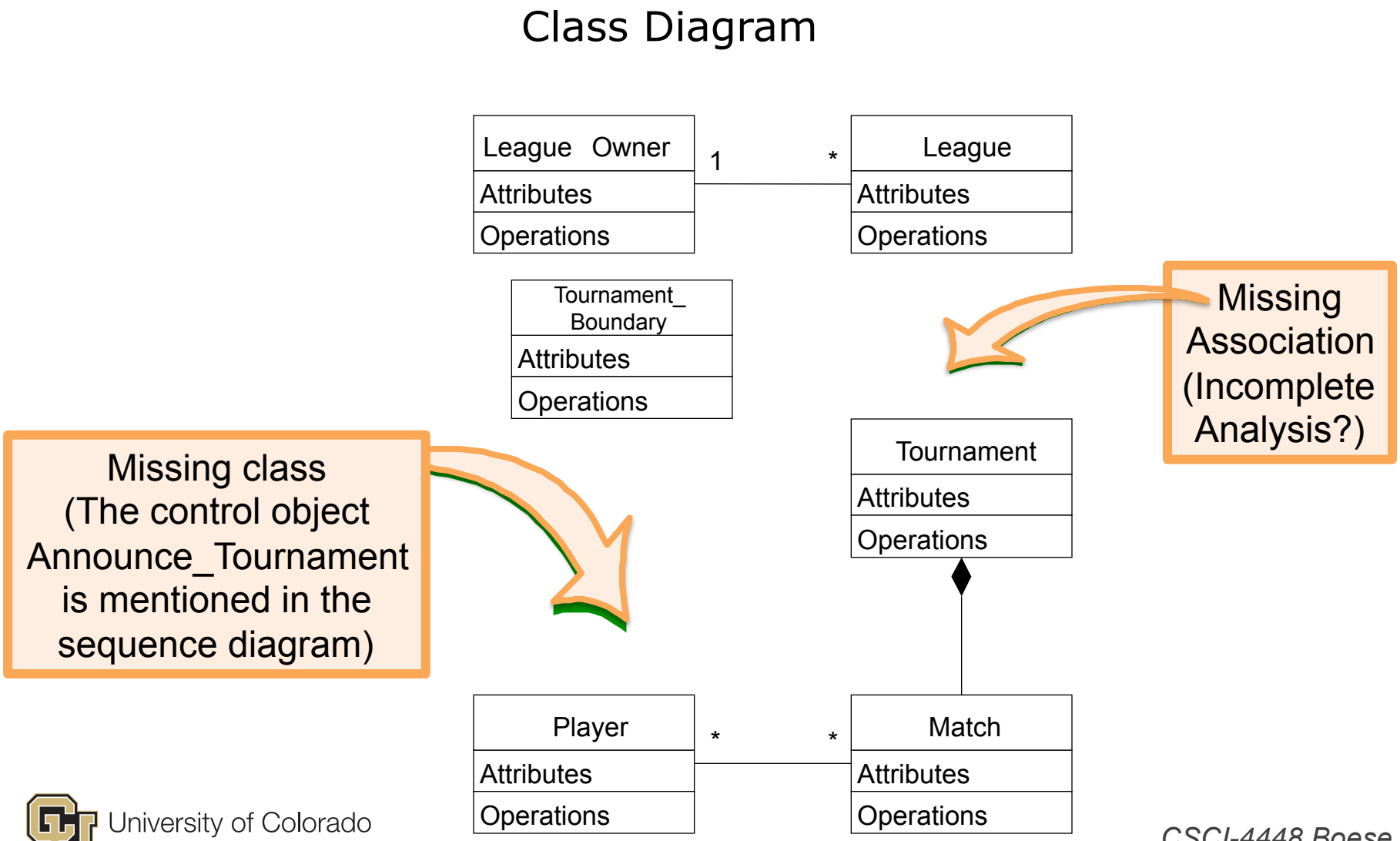
UML Sequence Diagram

Different spellings
in different models
for the same operation

UML Class Diagram



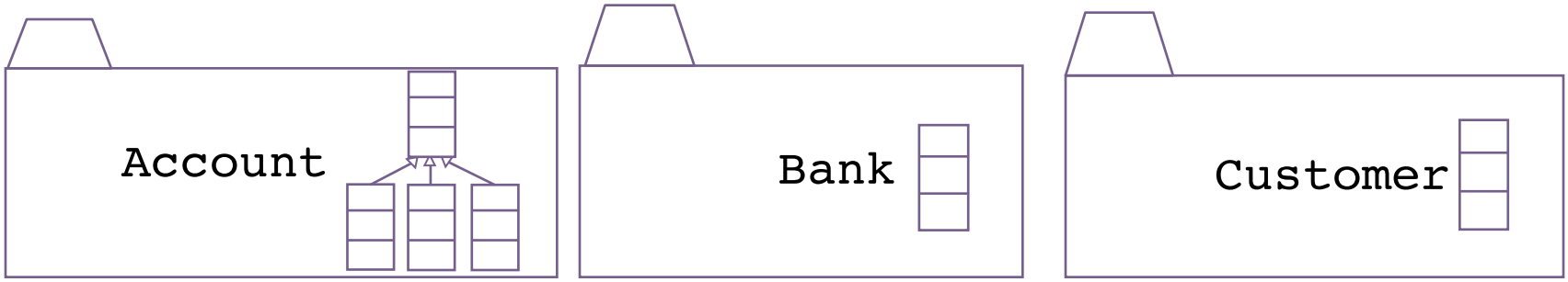
Omissions in some UML Diagrams



Packages

Packages

- **Packages** help you to organize UML models to increase their readability
- We can use the UML package mechanism to organize classes into subsystems



- Any complex system can be decomposed into subsystems, where each subsystem is modeled as a package.

```
package com.succeedInEveryWay.Account;
public class Business
{ …
```