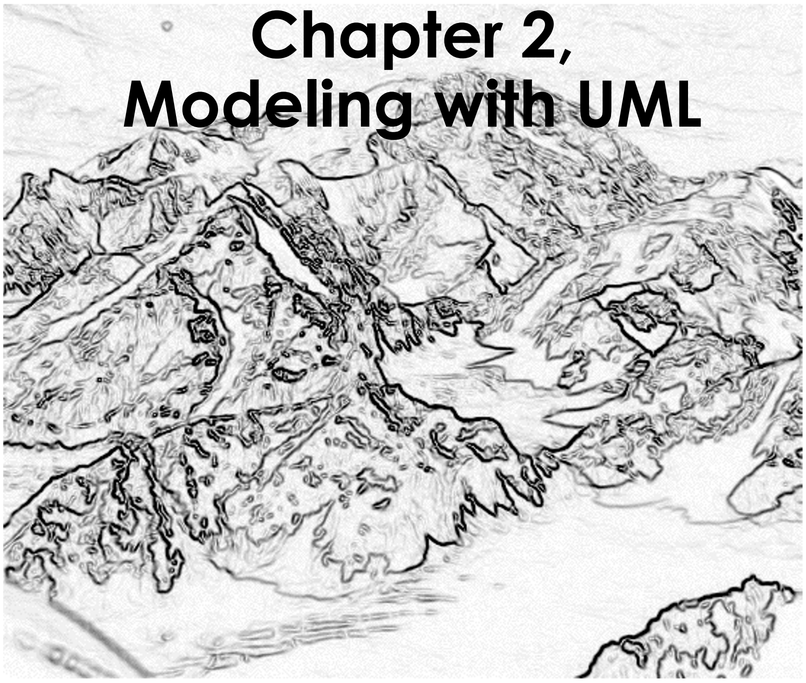
Object-Oriented Software Engineering
Using UML, Patterns, and Java



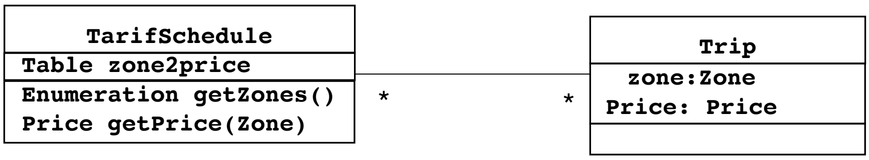
Chapter 2, Modeling with UML

UML Diagram Coverage

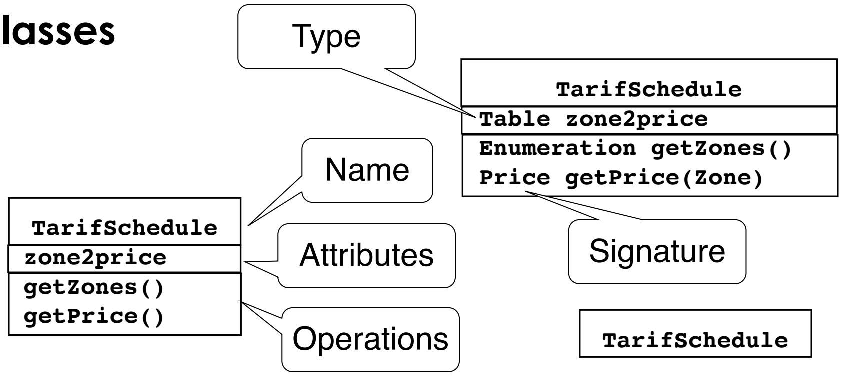
- Class diagrams
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object

Class Diagrams

- Class diagrams represent the structure of the system
- Used
  - during requirements analysis to model application domain concepts
  - during system design to model subsystems
  - during object design to specify the detailed behavior and attributes of classes.



Classes



- A ***class*** represents a concept
- A class encapsulates state ***(attributes)*** and behavior ***(operations)***

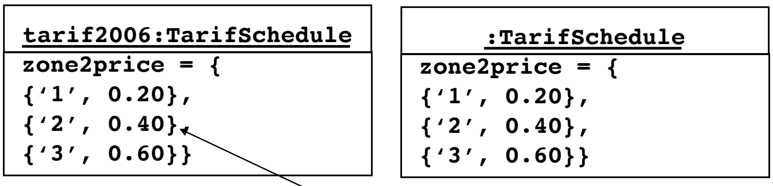
  Each attribute has a ***type***
  Each operation has a ***signature***

  The class name is the only mandatory information

Class vs Object

- **Class**
  - An abstraction modeling an entity in the application or solution domain
  - The class is part of the system model ("User", "Ticket distributor", "Server")
- **Object**
  - A specific instance of a class ("Joe, the passenger who is purchasing a ticket from the ticket distributor").

Instances

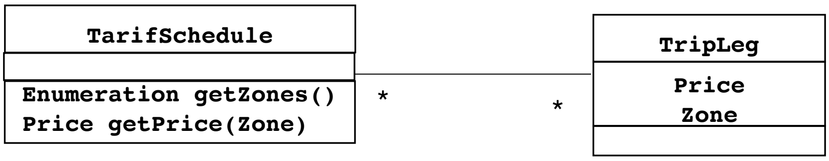


```
tarif2006:TarifSchedule
zone2price = {
{'1', 0.20},
{'2', 0.40},
{'3', 0.60}}
```

```
:TarifSchedule
zone2price = {
{'1', 0.20},
{'2', 0.40},
{'3', 0.60}}
```

- An ***instance*** represents a phenomenon
- The attributes are represented with their ***values***
- The name of an instance is underlined
- The name can contain only the class name of the instance (anonymous instance)

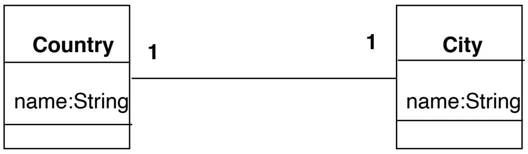
Associations



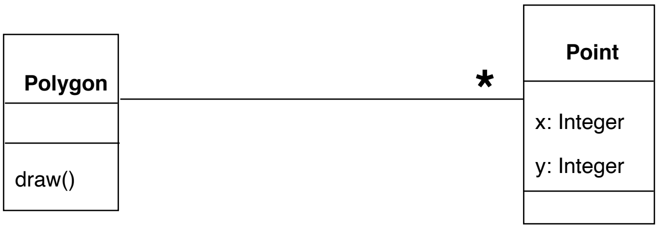
Associations denote relationships between classes

The multiplicity of an association end denotes how many objects the instance of a class can legitimately reference.

1-to-1 and 1-to-many Associations

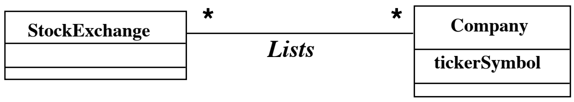


1-to-1 association



1-to-many association

Many-to-many Associations


StockExchange

* Lists *

Company

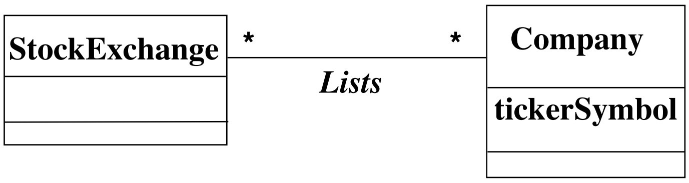
tickerSymbol

- A stock exchange lists many companies.
- Each company is identified by a ticker symbol

From Problem Statement To Object Model

Problem Statement: A stock exchange lists many companies. Each company is uniquely identified by a ticker symbol

Class Diagram:


StockExchange

* Lists *

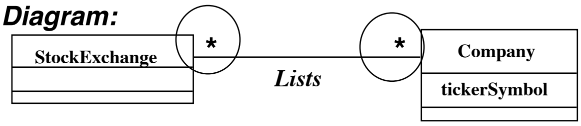
Company

tickerSymbol

From Problem Statement to Code

Problem Statement : A stock exchange lists many companies. Each company is identified by a ticker symbol

Class Diagram:



Java Code

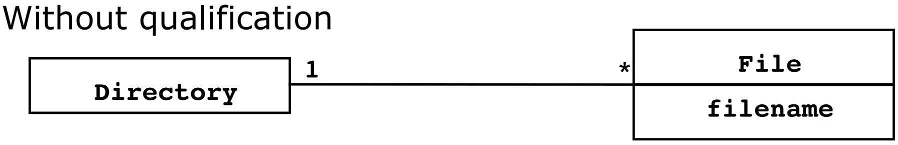
```
public class StockExchange
{
 private Vector m_Company = new Vector();
};

public class Company
{
 public int m_tickerSymbol;
 private Vector m_StockExchange = new Vector();
};
```

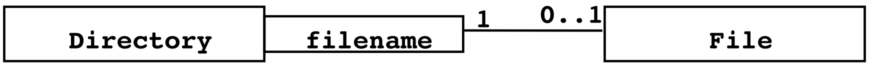
Associations are mapped to Attributes!

Qualifiers

Without qualification

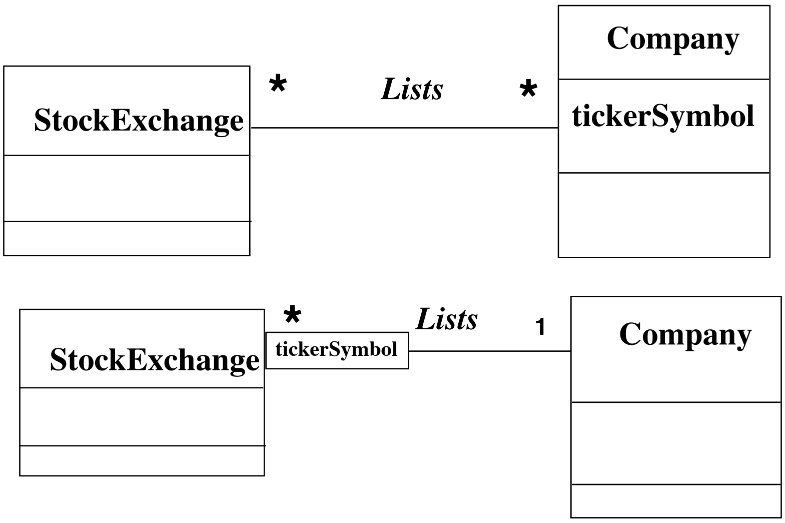


With qualification



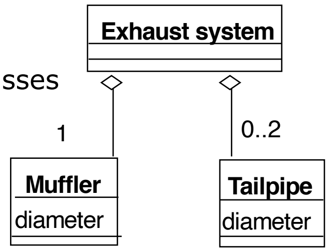
- Qualifiers can be used to reduce the multiplicity of an association

Qualification: Another Example

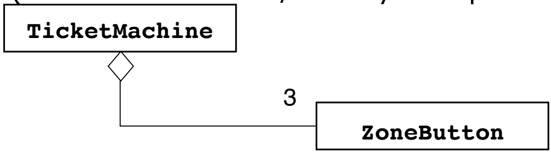


Aggregation

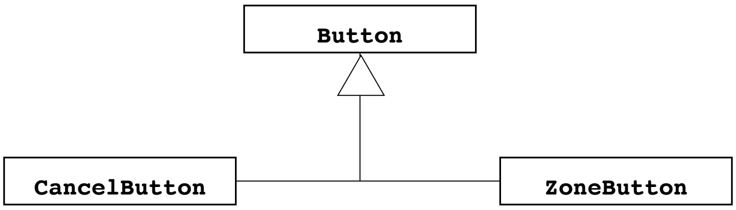
- An *aggregation* is a special case of association denoting a "consists-of" hierarchy
- The *aggregate* is the parent class, the components are the children classes



A solid diamond denotes *composition*: A strong form of aggregation where the *life time of the component instances* is controlled by the aggregate. That is, the parts don't exist on their won ("the whole controls/destroys the parts")



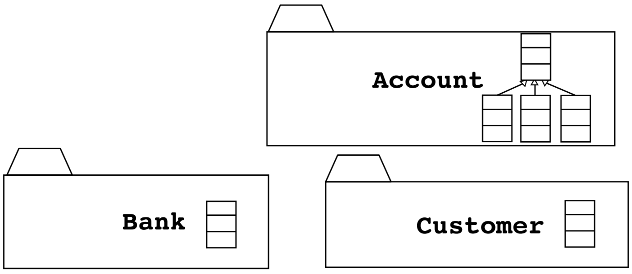
Inheritance



- *Inheritance* is another special case of an association denoting a “kind-of” hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The **children classes** inherit the attributes and operations of the **parent class.**

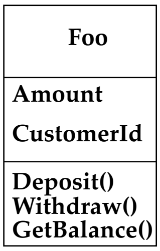
Packages

- 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.

Object Modeling in Practice



Foo

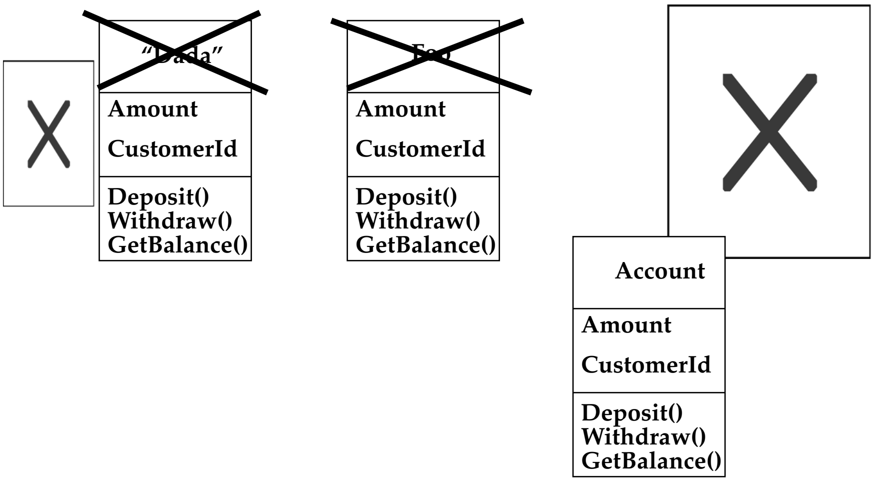
Amount
CustomerId

Deposit()
Withdraw()
GetBalance()

Class Identification: Name of Class, Attributes and Methods

Is Foo the right name?

Object Modeling in Practice: Brainstorming



~~“Dada”~~

Amount
CustomerId

Deposit()
Withdraw()
GetBalance()

~~Foo~~

Amount
CustomerId

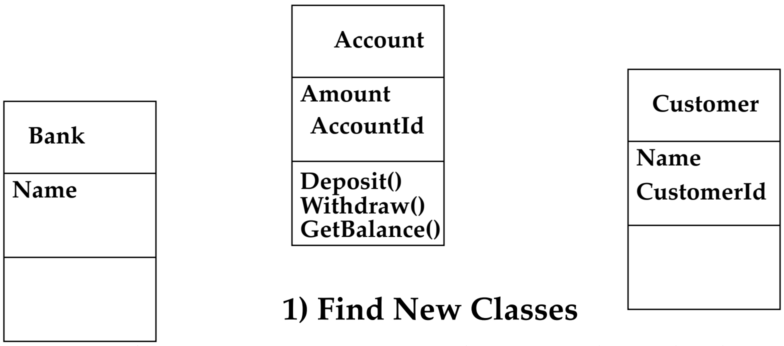
Deposit()
Withdraw()
GetBalance()

Account

Amount
CustomerId

Deposit()
Withdraw()
GetBalance()

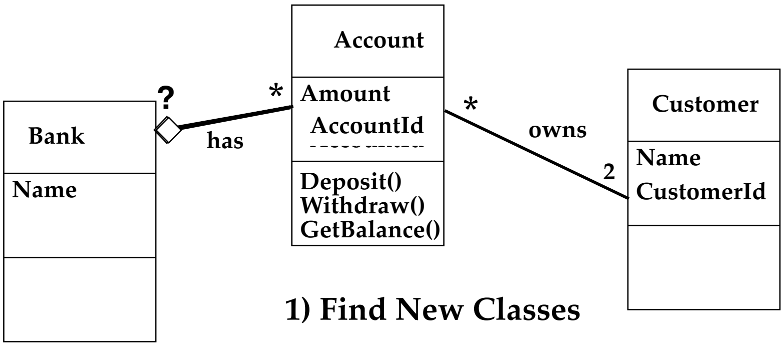
Object Modeling in Practice: More classes



1) Find New Classes

2) Review Names, Attributes and Methods

Object Modeling in Practice: Associations



1) Find New Classes

2) Review Names, Attributes and Methods

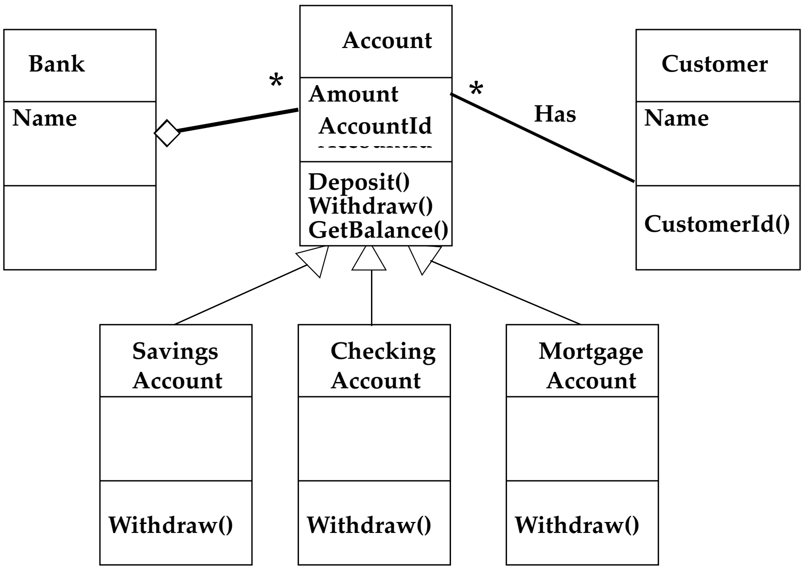
3) Find Associations between Classes

4) Label the generic assocations

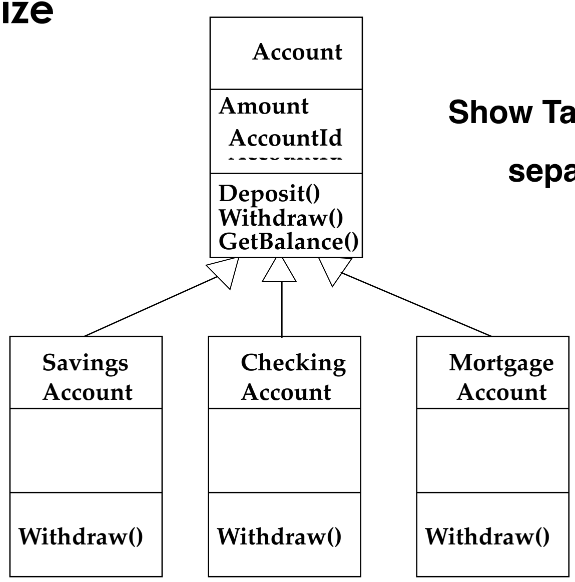
5) Determine the multiplicity of the assocations

6) Review associations

Practice Object Modeling: Find Taxonomies

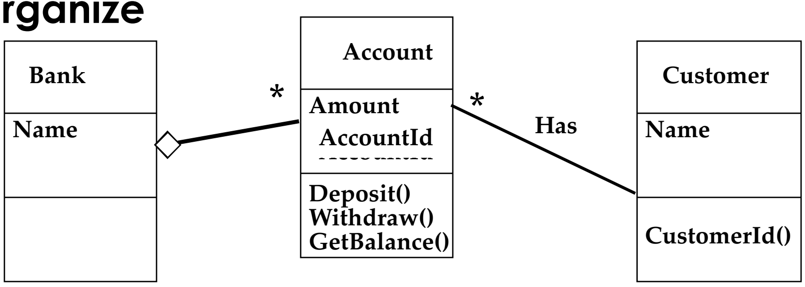


Practice Object Modeling: Simplify, Organize



Show Taxonomies separately

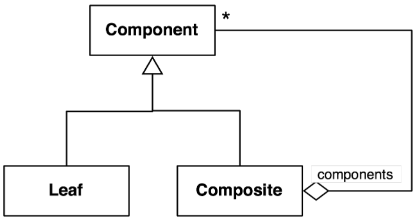
Practice Object Modeling: Simplify, Organize



Use the 7+-2 heuristics

or better 5+-2!

Code Generation from UML to Java I

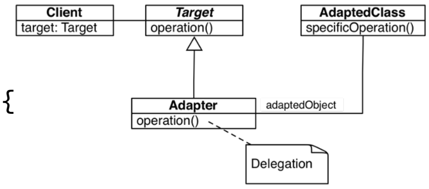


```
public class Component{  }

public class Leaf extends
  Component{  }

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

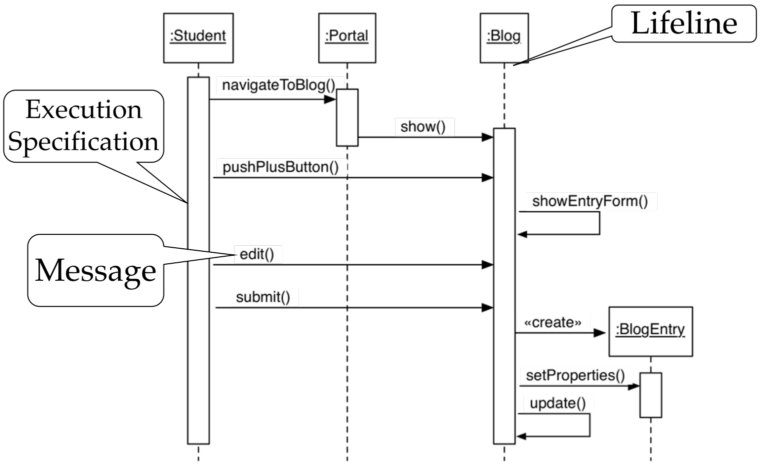
Code Generation from UML to Java II



```
public abstract class Target{
  public … operation(); }

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

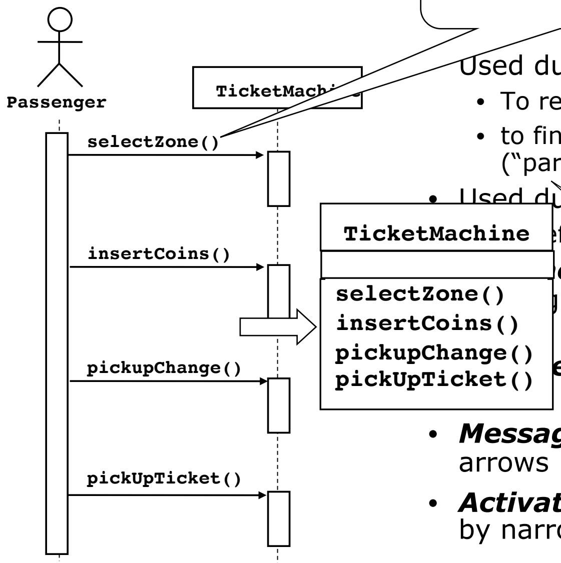
Sequence diagram: Basic Notation



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

Sequence Diagrams

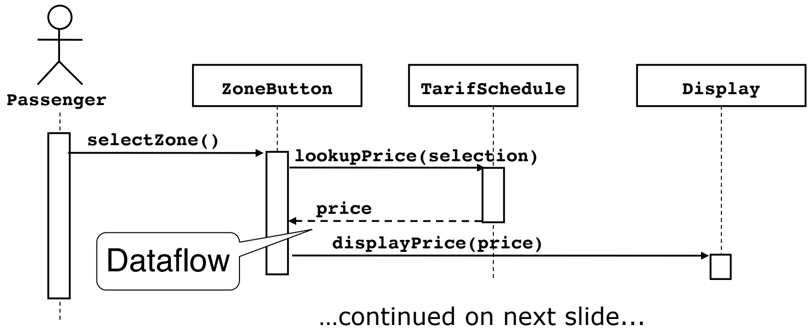
Focus on Controlflow



- Used during analysis
  - To refine use case descriptions
  - to find additional objects (“participating objects”)
- Used during system design
  - ...efine ...bsystem interfaces
- ...ces ar... ...gles. Act...
- ...es are represented by ... lines
- ***Messages*** are represented by arrows
- ***Activations*** are represented by narrow rectangles.

Messages -> Operations on participating Object

Sequence Diagrams can also model the Flow of Data

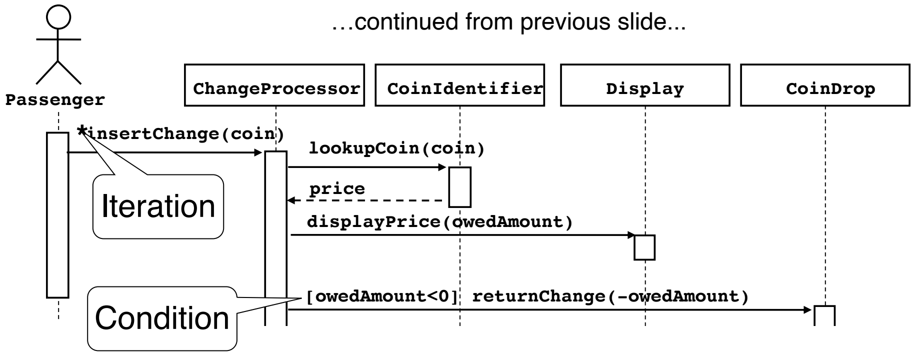


...continued on next slide...

- The source of an arrow indicates the activation which sent the message
- Horizontal dashed arrows indicate data flow, for example return results from a message

Sequence Diagrams: Iteration & Condition

…continued from previous slide...

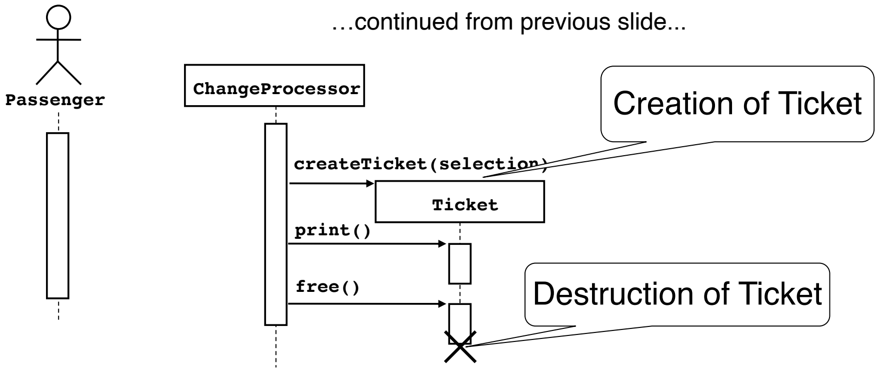


…continued on next slide...

- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name

Creation and destruction

…continued from previous slide...



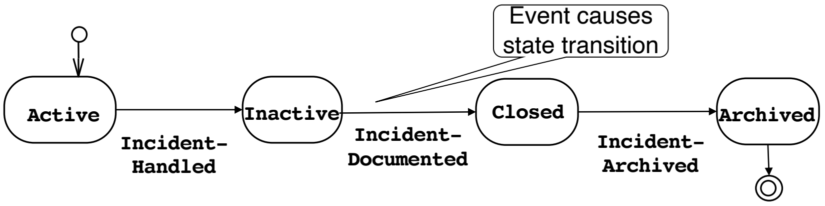
- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
  - In garbage collection environments, destruction can be used to denote the end of the useful life of an object.

Sequence Diagram Properties

- UML sequence diagram represent *behavior in terms of interactions*
- Useful to identify or find missing objects
- Time consuming to build, but worth the investment
- Complement the class diagrams (which represent structure).

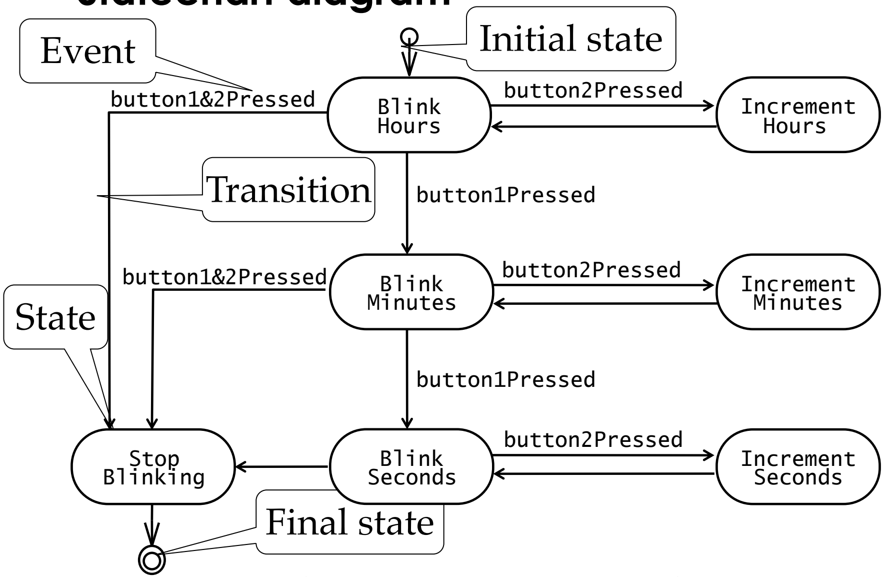
Statechart Diagram

Statechart Diagram for Incident
Focus on the set of attributes of a single abstraction (object, system)



Event causes state transition

Statechart diagram



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