Software Engineering

Object-Oriented Analysis and Design
and
Modeling with UML

Assoc. Prof. Marenglen Biba
MSc in Computer Science, UoG-UNYT
Foundation Programme

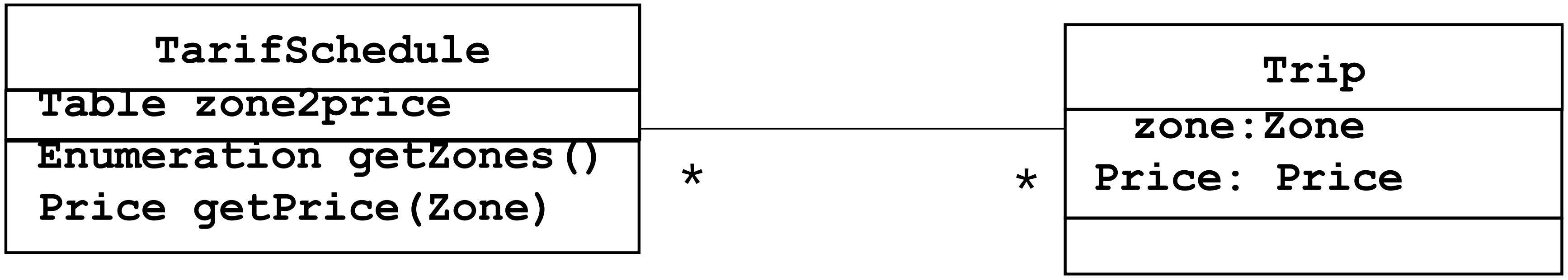
Material

- Get the material from
  - http://www.marenglenbiba.net/foundprog/
  - Sufficient for FP exam purposes

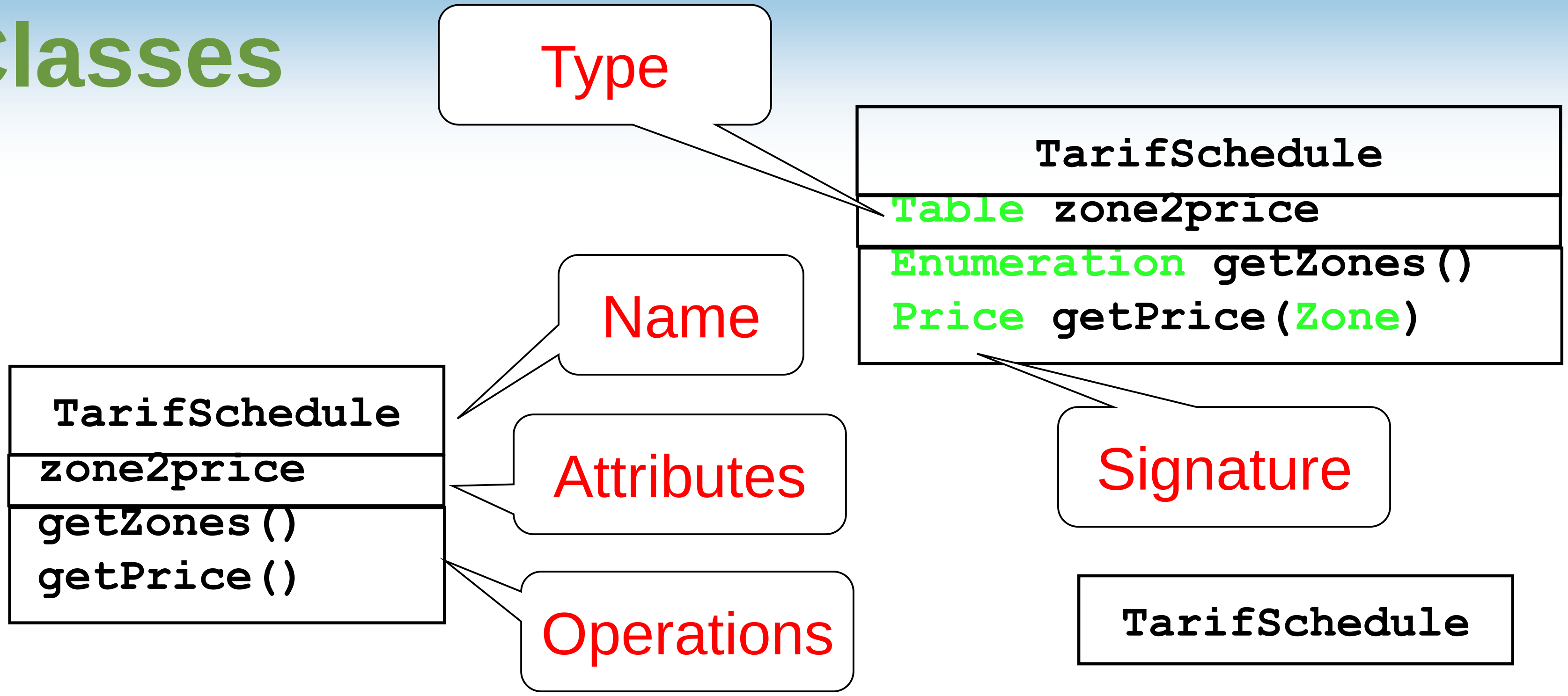
- Other useful material
  - I. Sommerville. Software Engineering (in library)
  - R. Pressman. Software Engineering: A Practitioner's Approach (in library)
  - *B. Bruegge & A. H. Dutoit.* Object-Oriented Software Engineering: Using UML, Patterns, and Java, 2nd Edition.

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

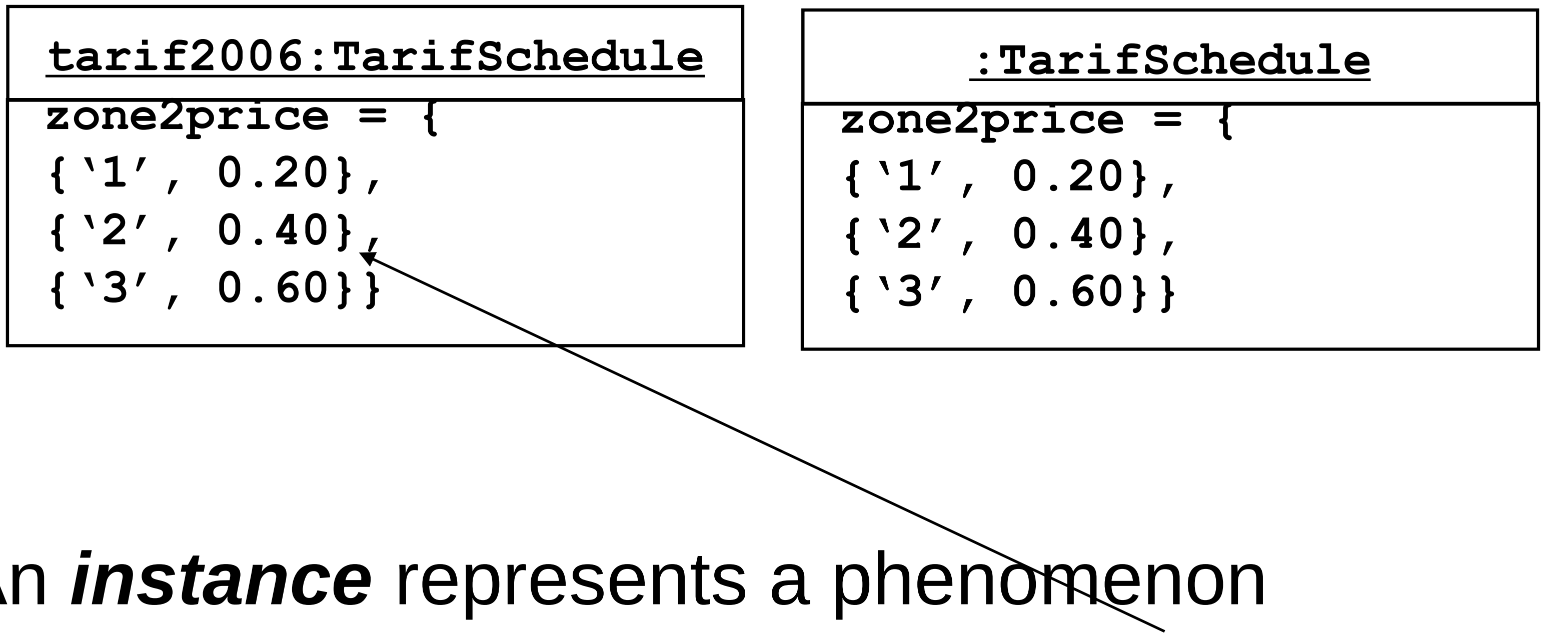


- A ***class*** represents a concept
- A class encapsulates state ***(attributes)*** and behavior ***(operations)***

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

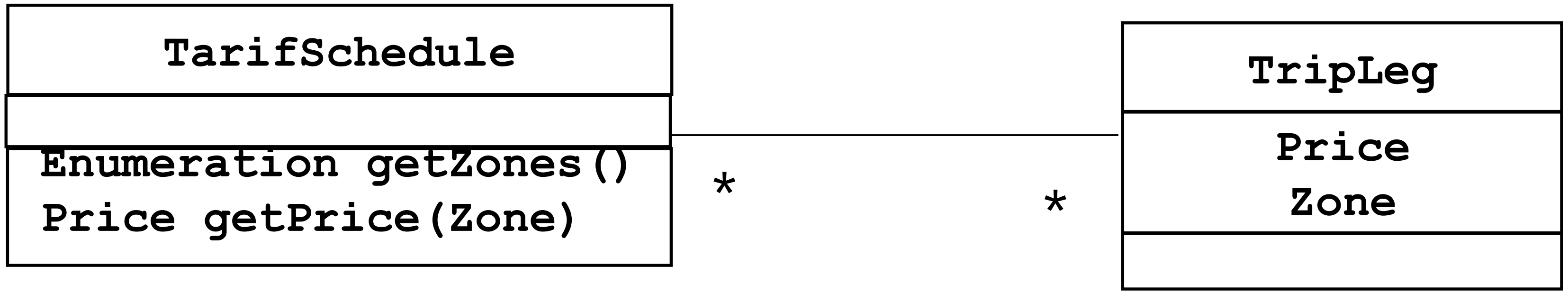
The class name is the only mandatory information

Instances



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

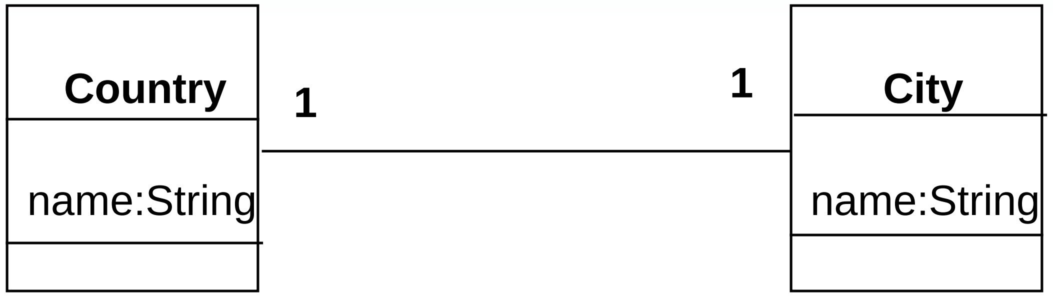
Associations



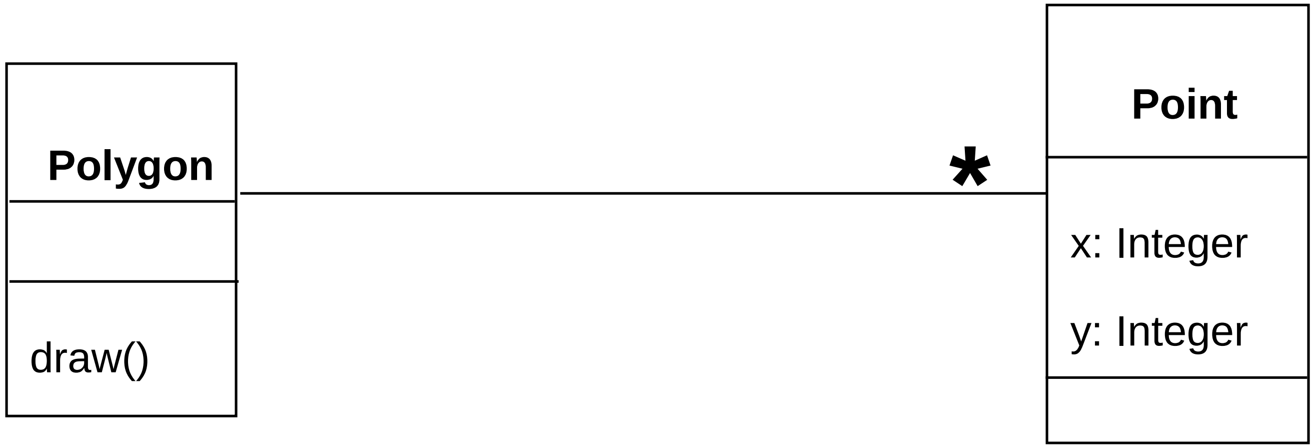
Associations denote relationships between classes

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

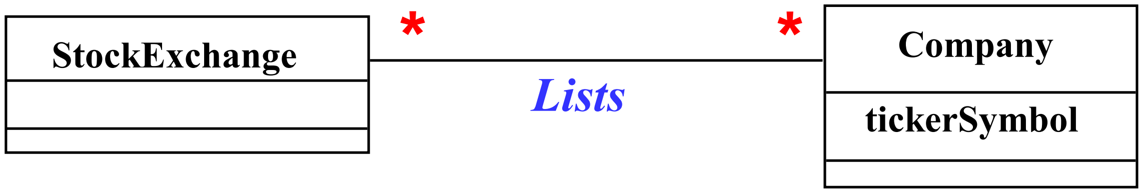


1-to-1 association



1-to-many association

Many-to-many Associations

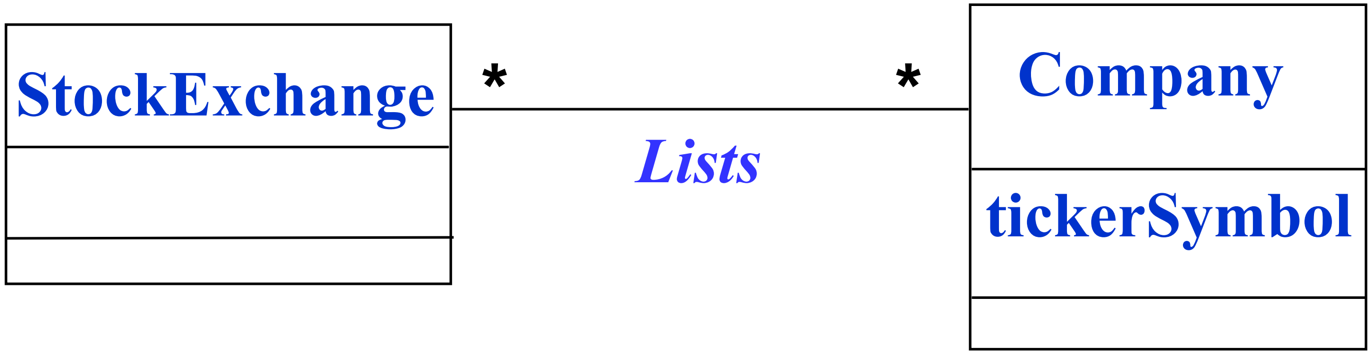


- A stock exchange lists many companies.
- Each company is identified by a ticker symbol

From Problem Statement To Object Model

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

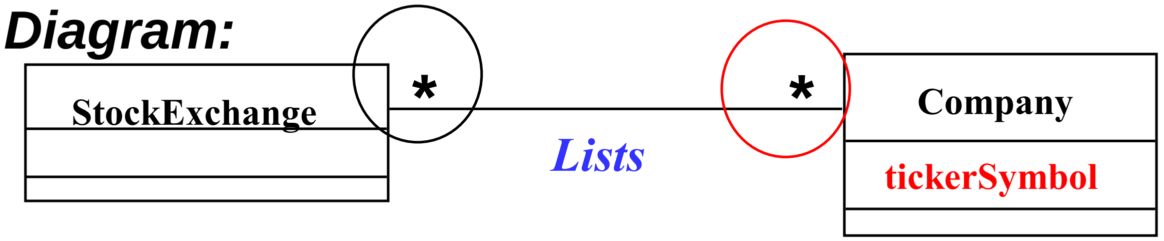
Class Diagram:



From Problem Statement to Code

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

Class Diagram:



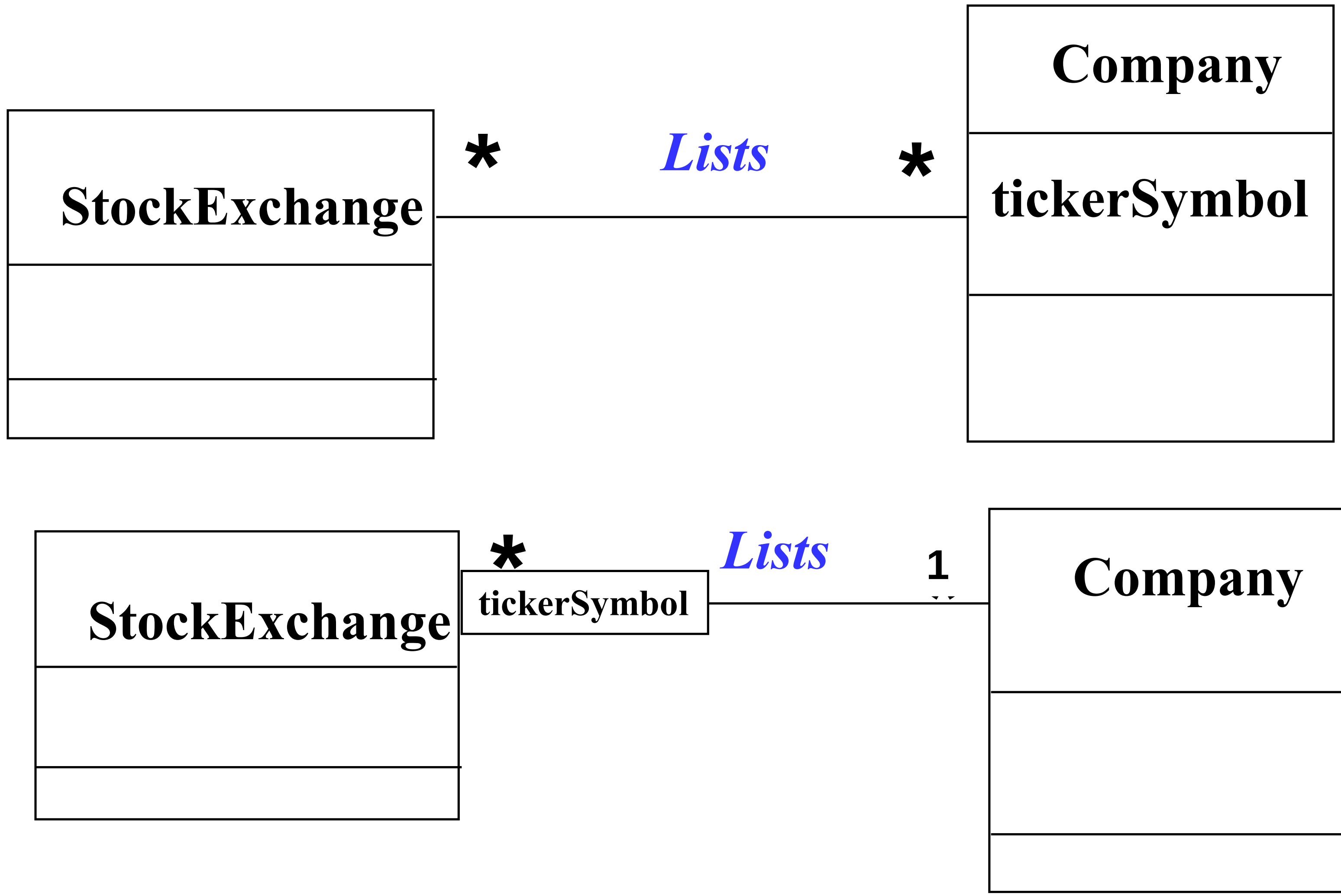
Java Code

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

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

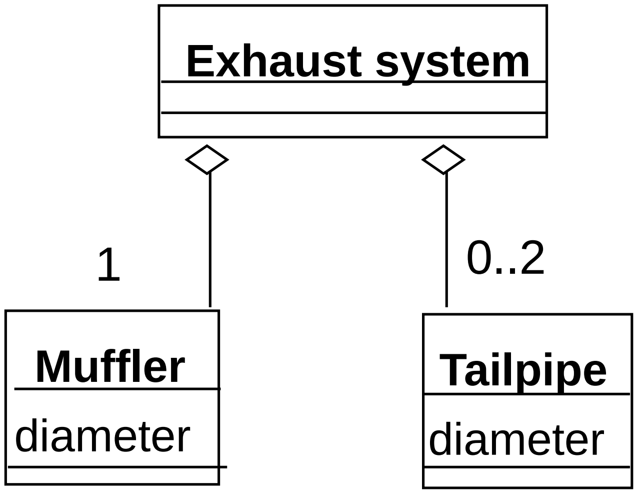
Associations are mapped to Attributes!

Qualification: Another Example

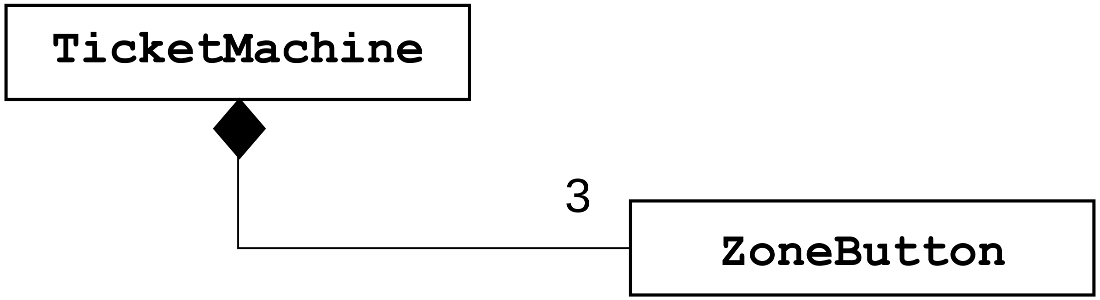


Aggregation

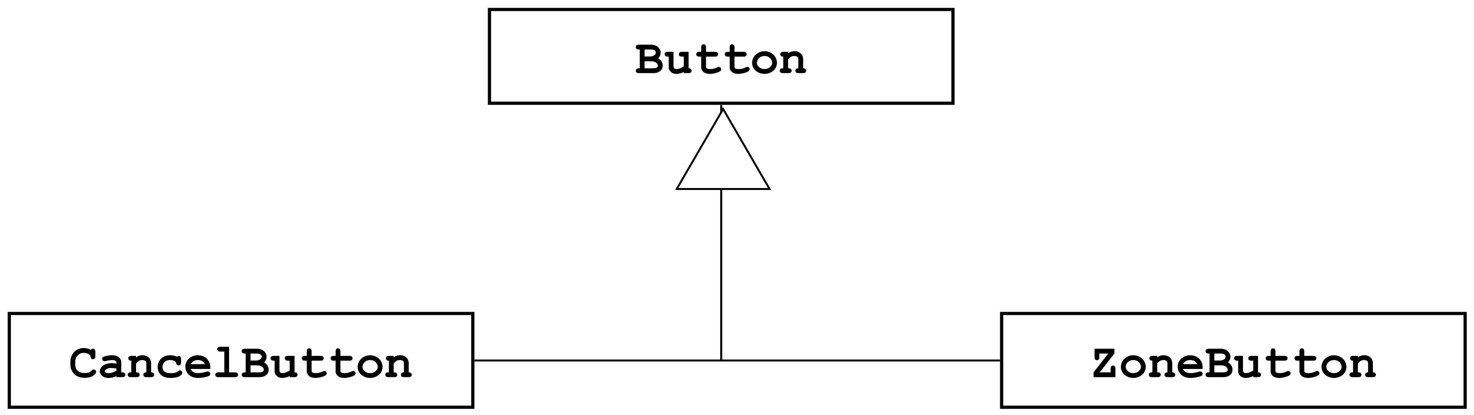
- An *aggregation* is a special case of association denoting a "consists-of" hierarchy
- The *aggregate* is the parent class, the components are the children classes



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 own ("the whole controls/destroys the parts")



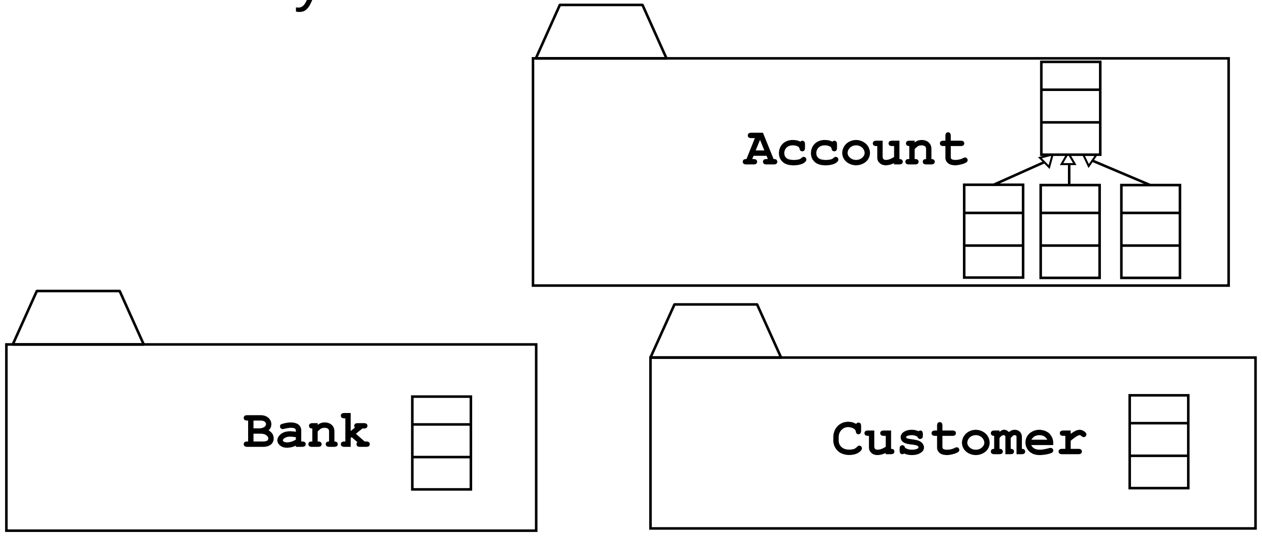
Inheritance



- *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.**

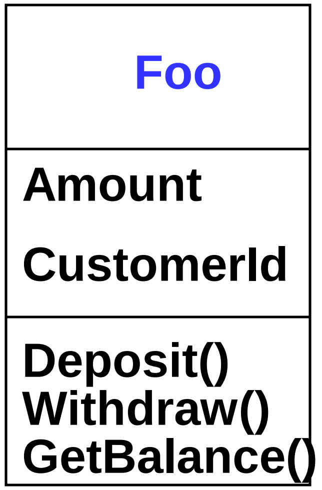
Packages

- 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.

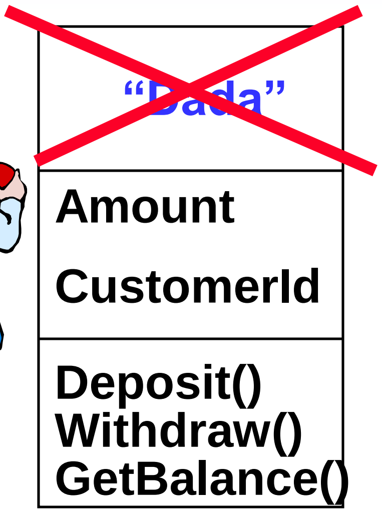
Object Modeling in Practice

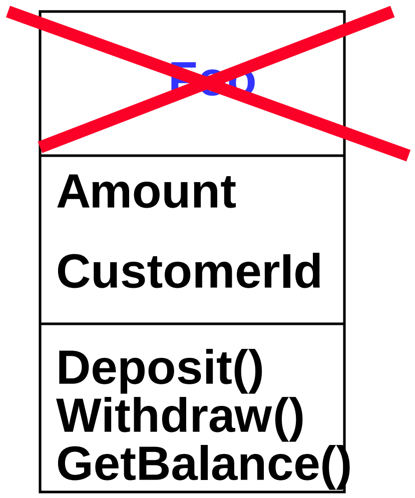


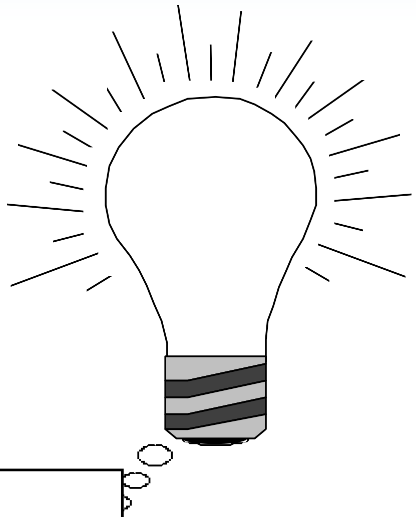
Class Identification: Name of Class, Attributes and Methods

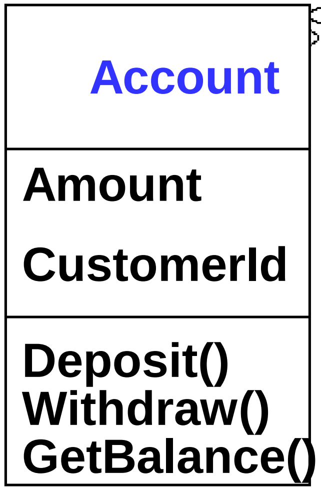
Is Foo the right name?

Object Modeling in Practice: Brainstorming



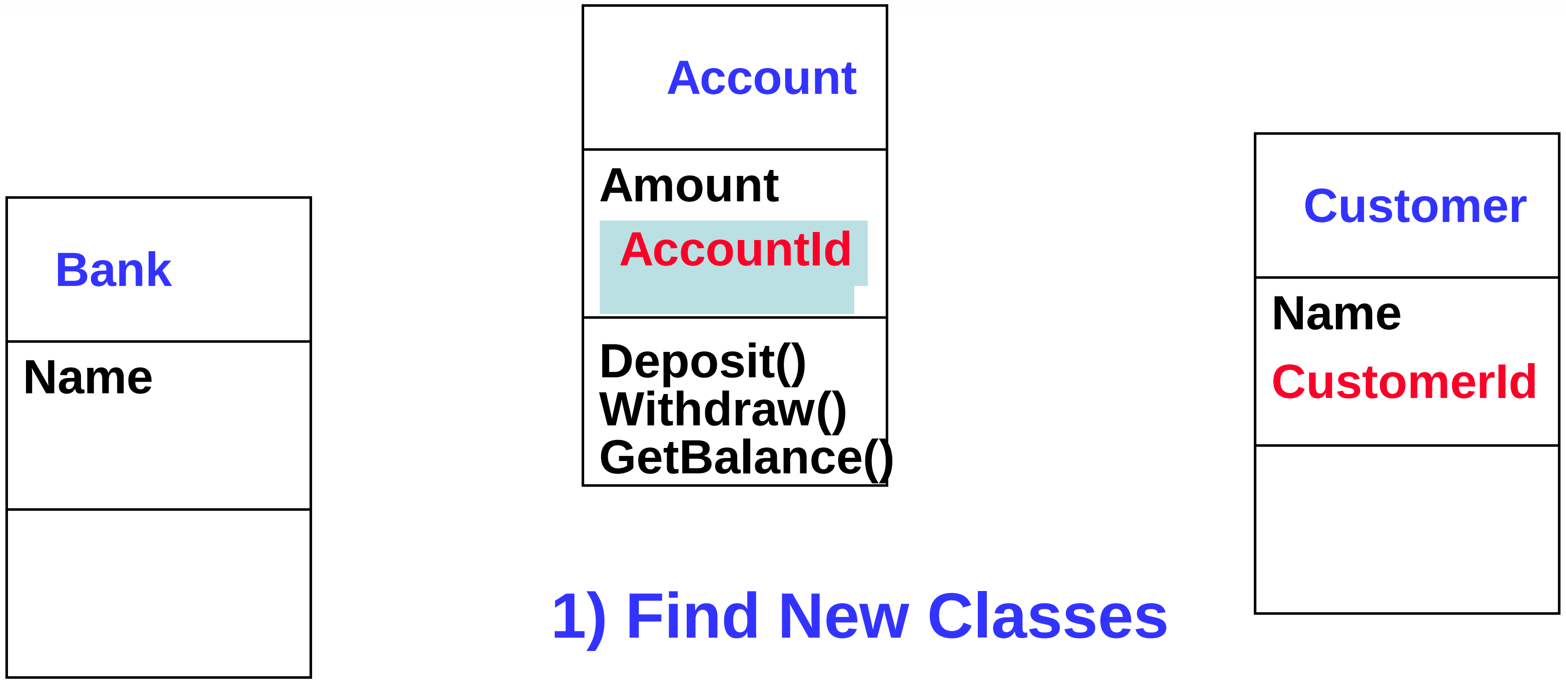






Is Foo the right name?

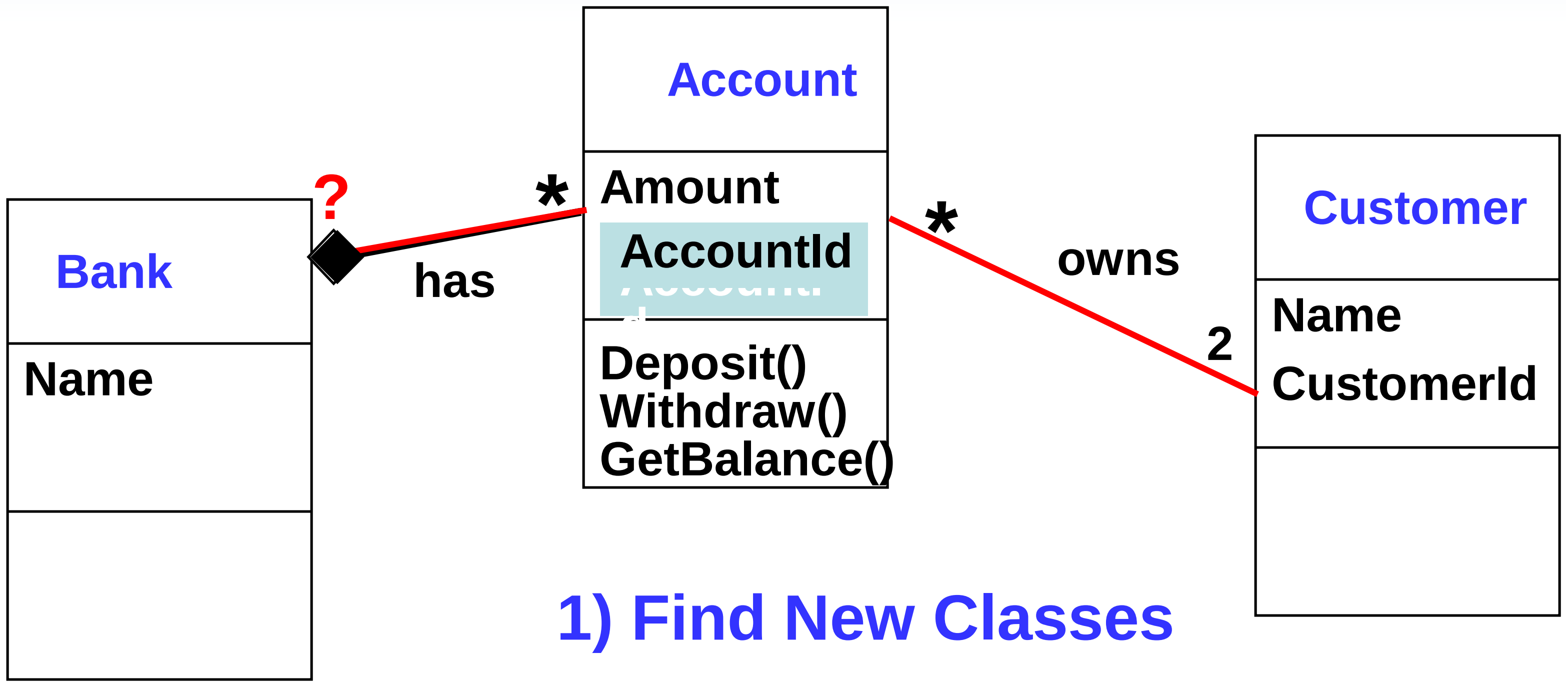
Object Modeling in Practice: More classes



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

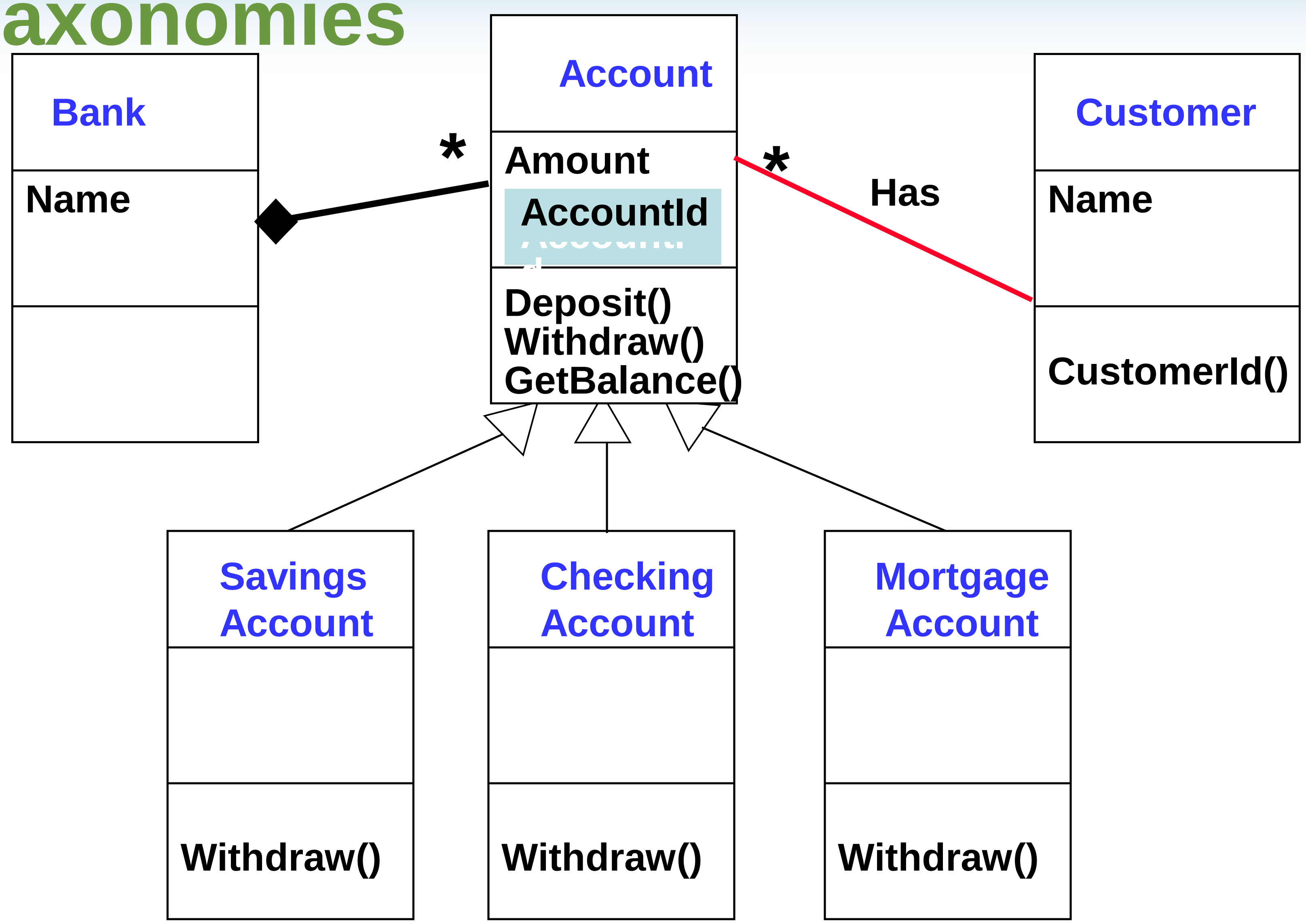
3) Find Associations between Classes

4) Label the generic assocations

5) Determine the multiplicity of the assocations

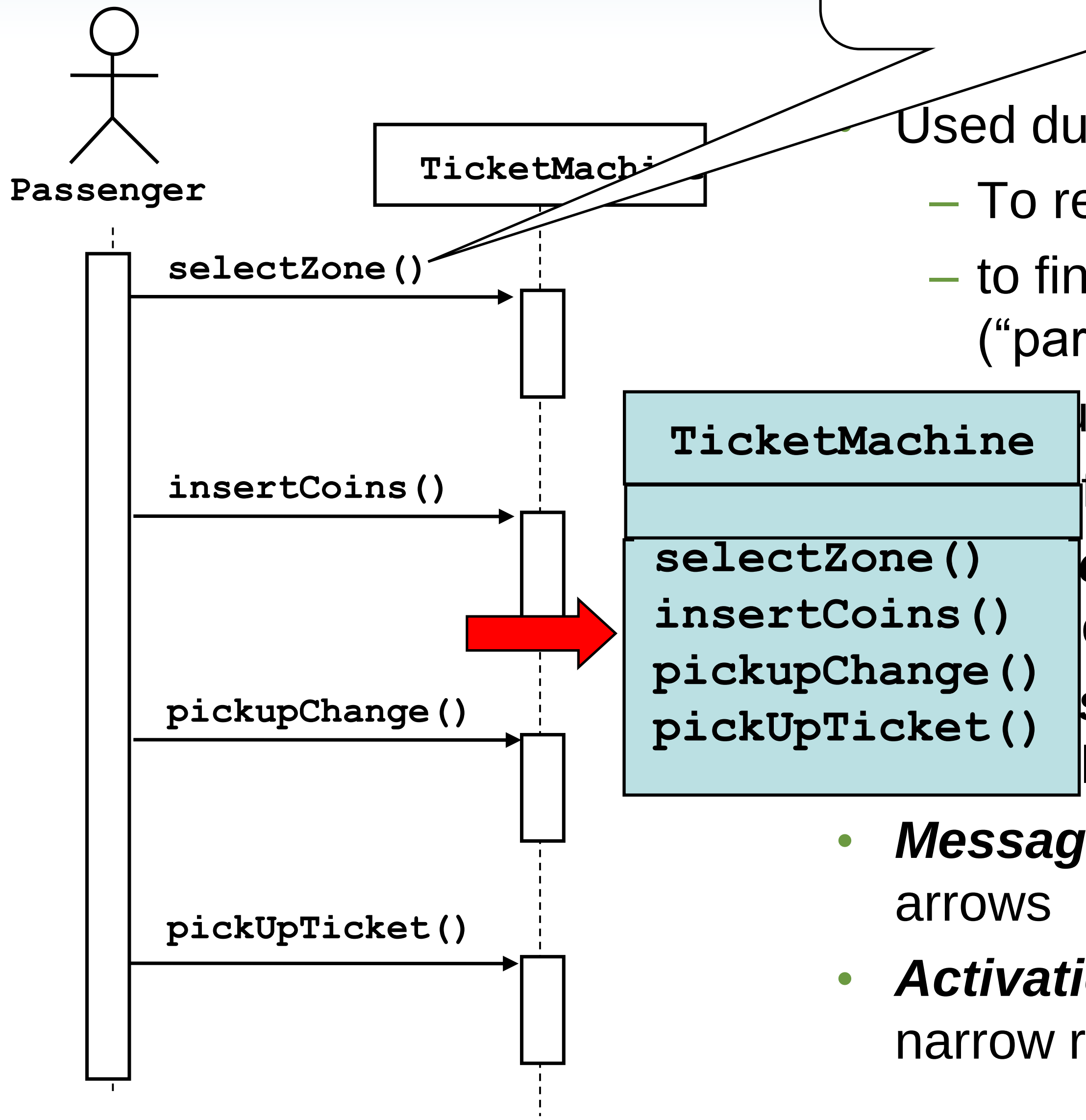
6) Review associations

Practice Object Modeling: Find Taxonomies



Sequence Diagram

Focus on Controlflow

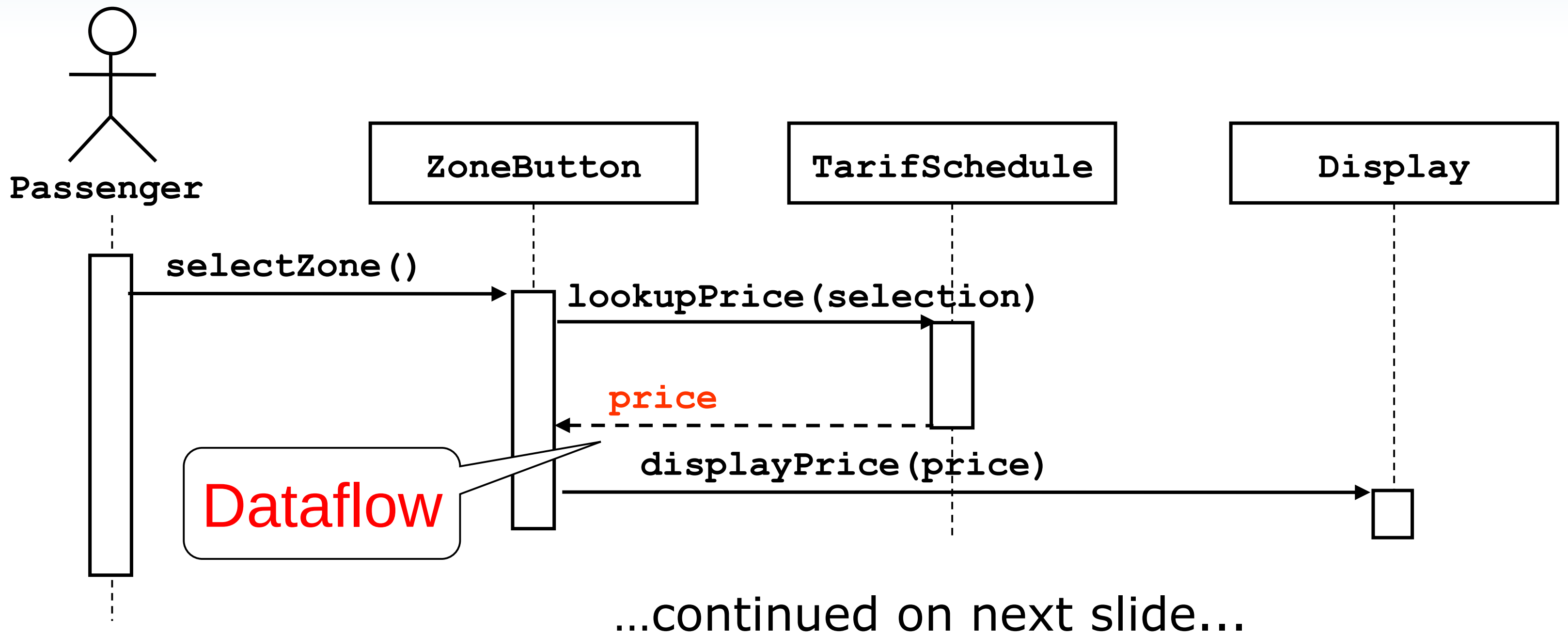


- Used during analysis
  - To refine use case descriptions
  - to find additional objects (“participating objects”)
- uring system design
  - fine subsy
- ***es*** are re
  es. ***Actor***
- ***s*** are repres d by
  lines

Messages -> Operations on participating Object

- ***Messages*** are represented by arrows
- ***Activations*** are represented by narrow rectangles.

Sequence Diagrams can also model the Flow of Data

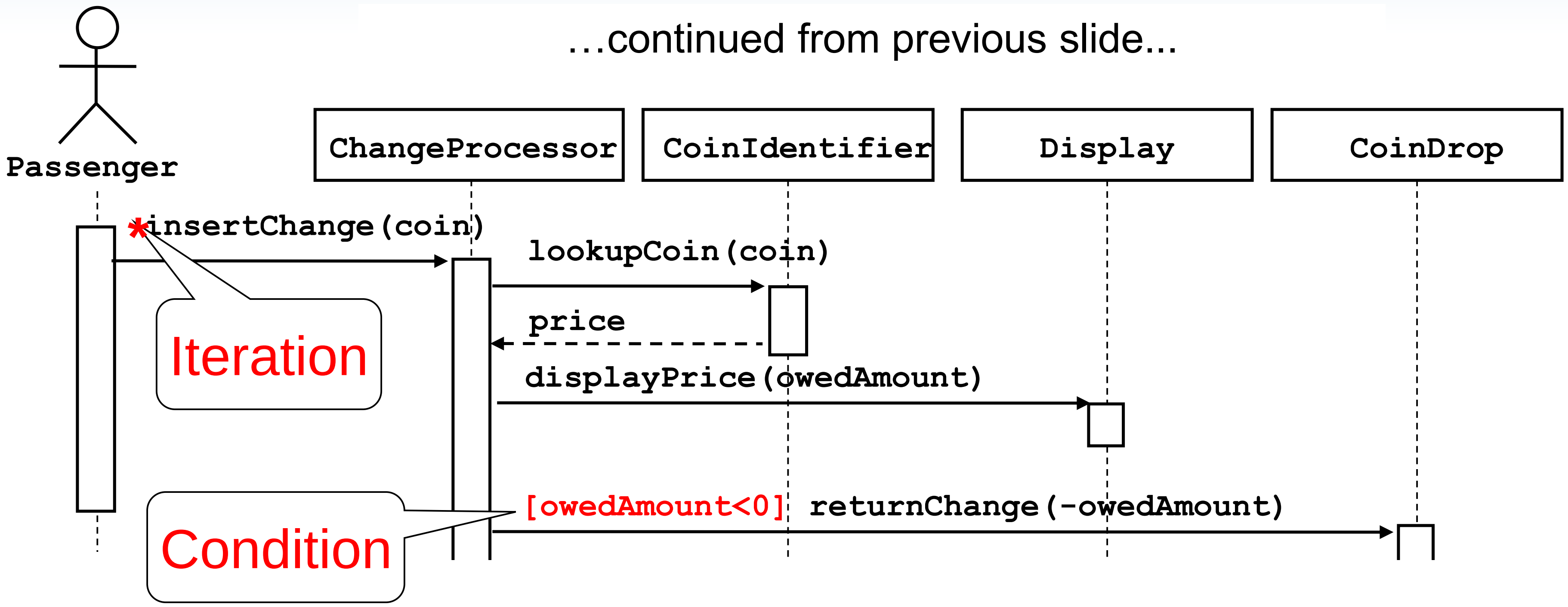


…continued on next slide...

- 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

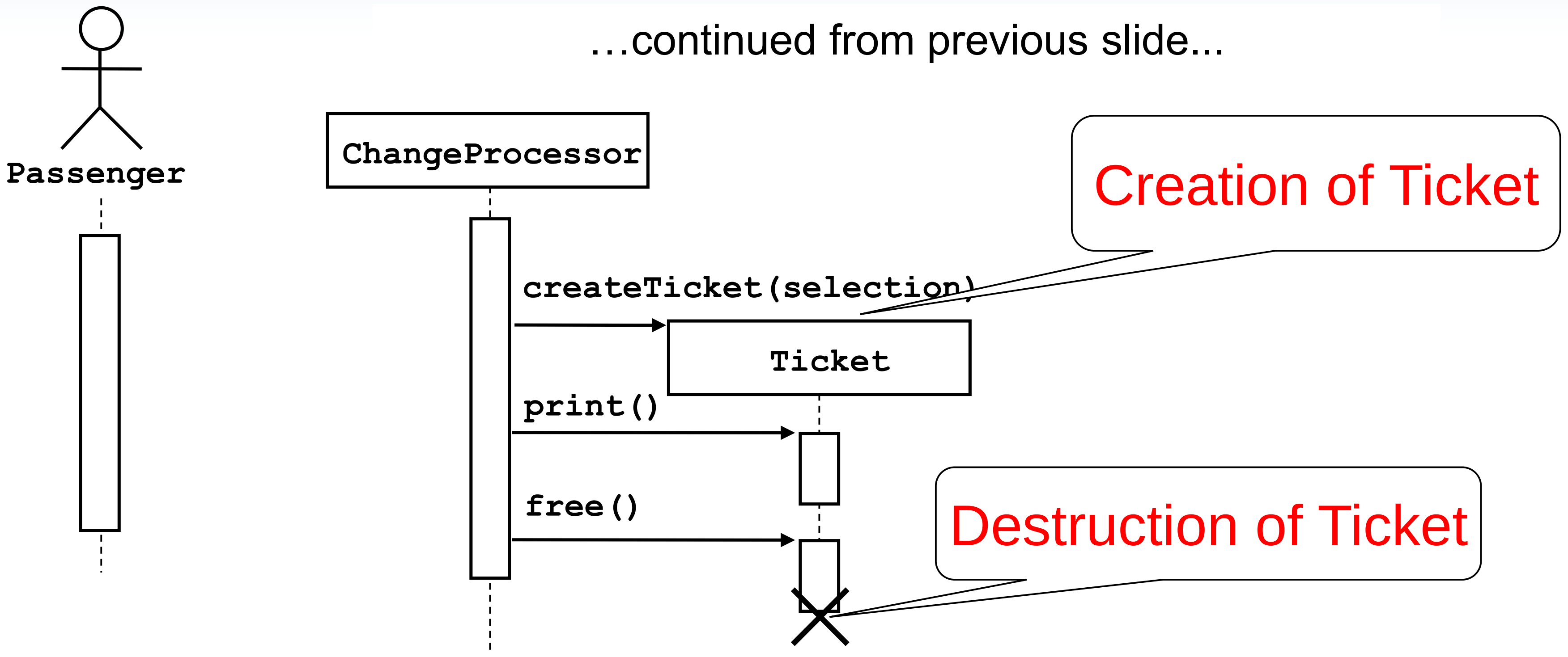
…continued from previous slide...



...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



- 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.

Sample code for the diagram

```
public class Machine{
Display ds = new Display();
private ZoneButton zb = new ZoneButton(ds);
private TarifSchedule tf = new TarifSchedule();

public static void main(String[] args){
int selection = zb.selectZone();
double price = tf.lookupPrice(selection);
zb.sendPrice(price);
}

public class ZoneButton{
private Display ds;
public ZoneButton(Display ds){
this.ds = ds;
}
public void sendPrice(double price){
ds.displayPrice(price);
}
```

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).

UML Summary

- UML provides a wide variety of notations for representing many aspects of software development
  - Powerful, but complex
- UML is a powerful language
  - Can be misused to generate unreadable models
  - Can be misunderstood when using too many exotic features
- We concentrated on a few notations:
  - Functional model: Use case diagram
  - Object model: class diagram
  - Dynamic model: sequence diagrams, statechart and activity diagrams.

Lab Session on UML

- Class diagrams