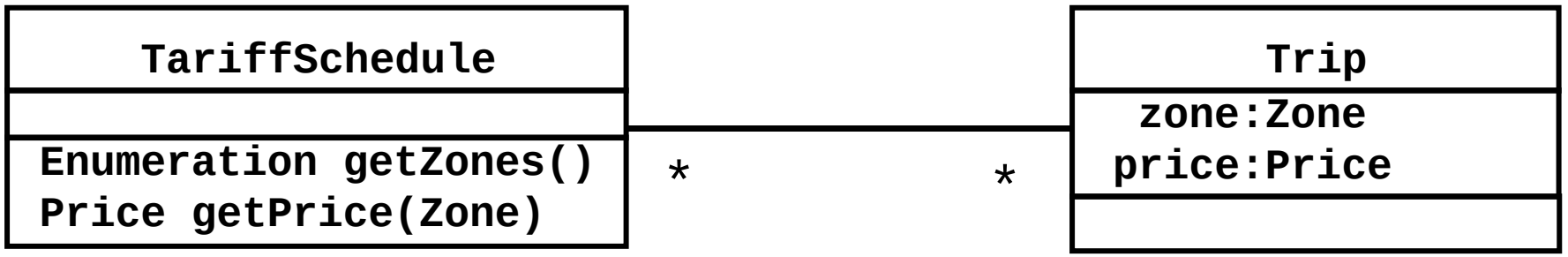
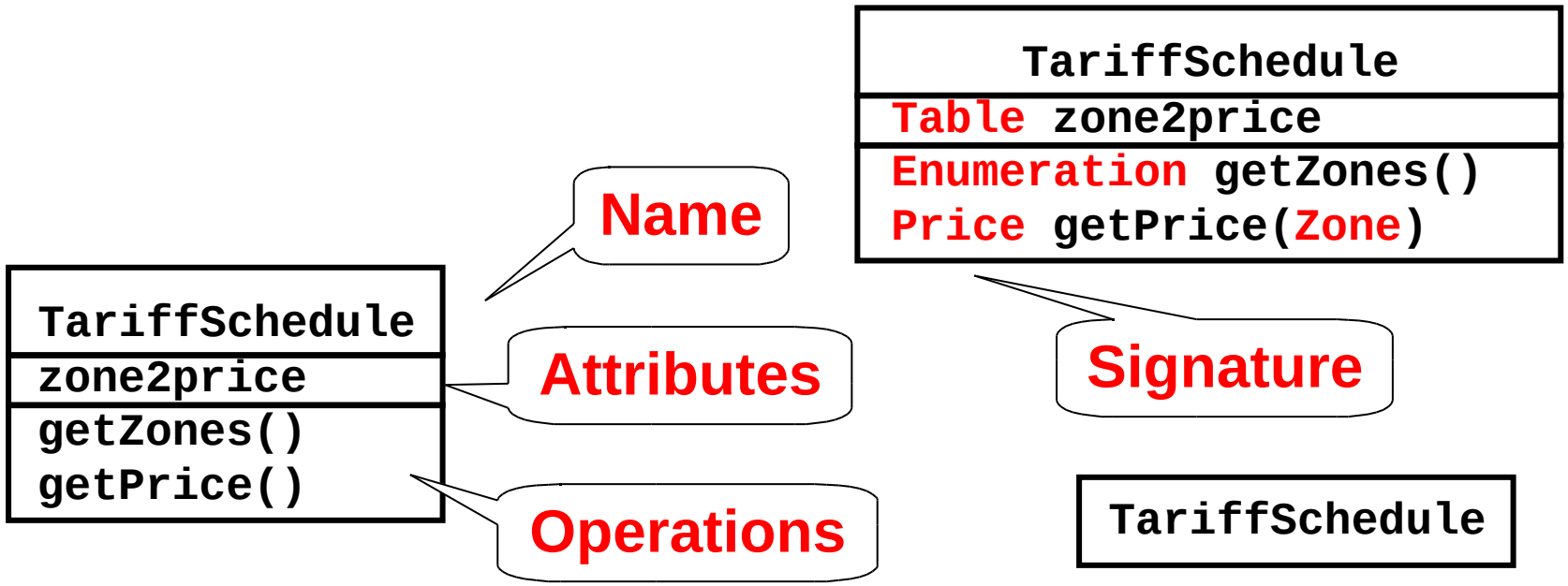
# *Class Diagrams*



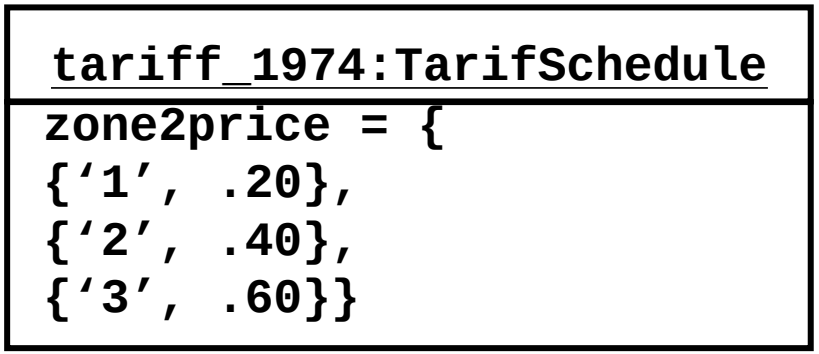

- Class diagrams represent the structure of the system.
- Class diagrams are used
  - **during requirements analysis to model problem domain concepts**
  - **during system design to model subsystems and interfaces**
  - **during object design to model classes.**

# *Classes*




- A ***class*** represent 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*

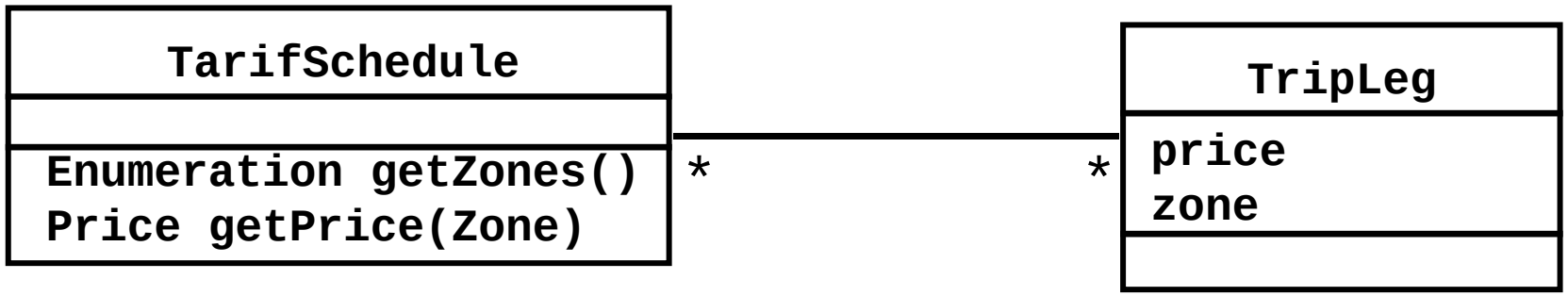



- An ***instance*** represents a phenomenon.
- The name of an instance is underlined and can contain the class of the instance.
- The attributes are represented with their ***values***.

# *Actor vs. Instances*

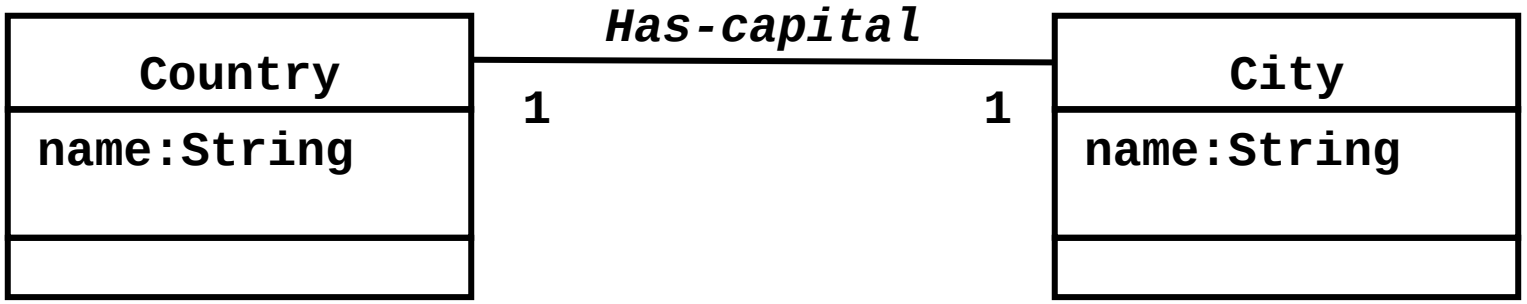
- What is the difference between an actor and a class and an instance?
- Actor:
  - **An entity outside the system to be modeled, interacting with the system (“Pilot”)**
- Class:
  - **An abstraction modeling an entity in the problem domain, inside the system to be modeled (“Cockpit”)**
- Object:
  - **A specific instance of a class (“Joe, the inspector”).**

# *Associations*

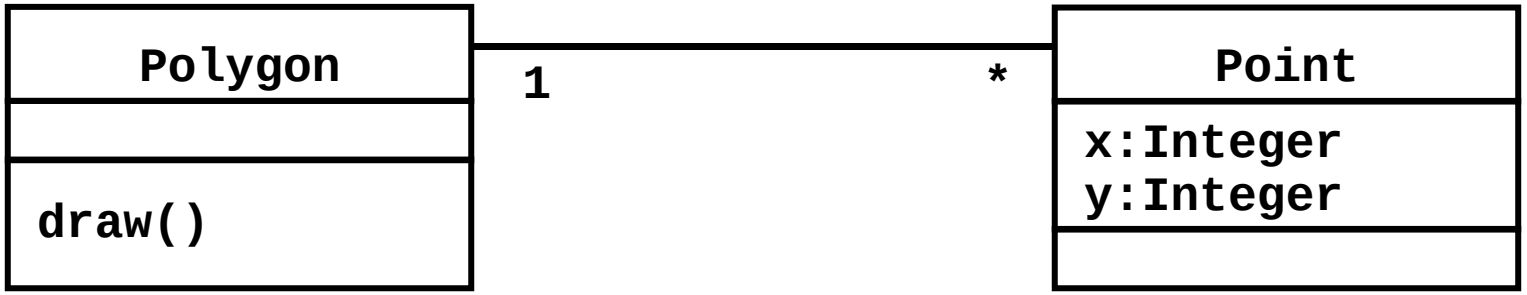



- Associations denote relationships between classes.
- The multiplicity of an association end denotes how many objects the source object can legitimately reference.

# *1-to-1 and 1-to-Many Associations*



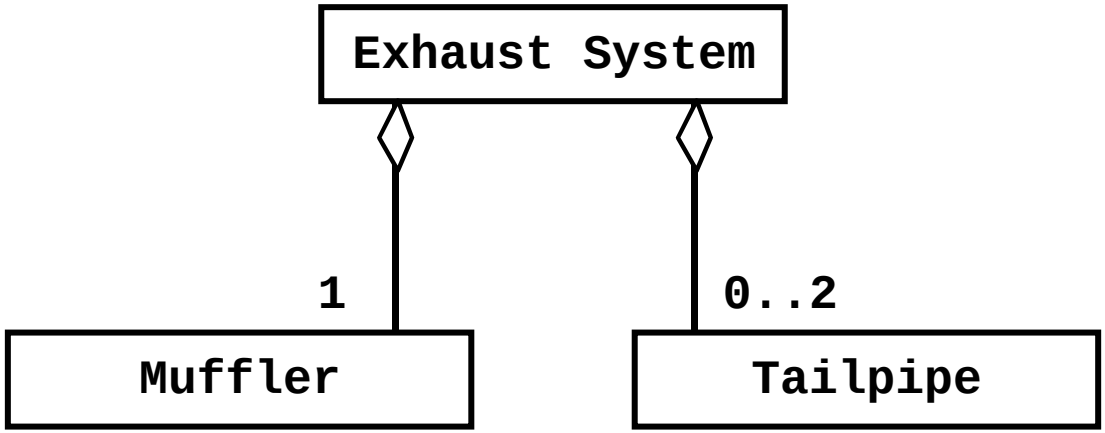

**1-to-1 association**




**1-to-many association**

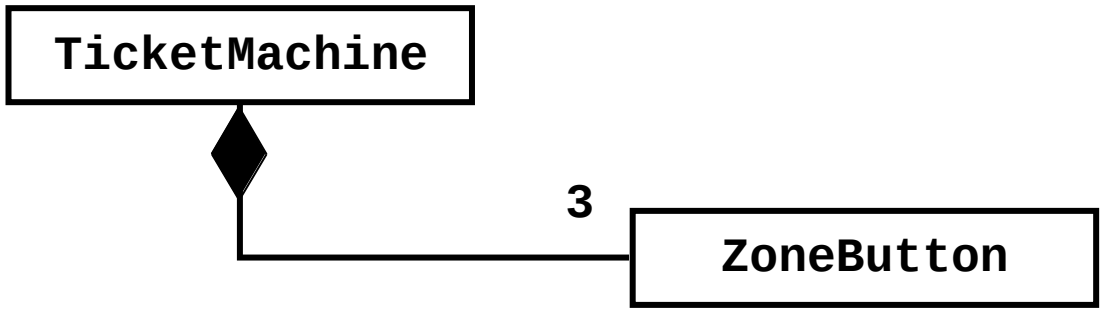
# *Aggregation*

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

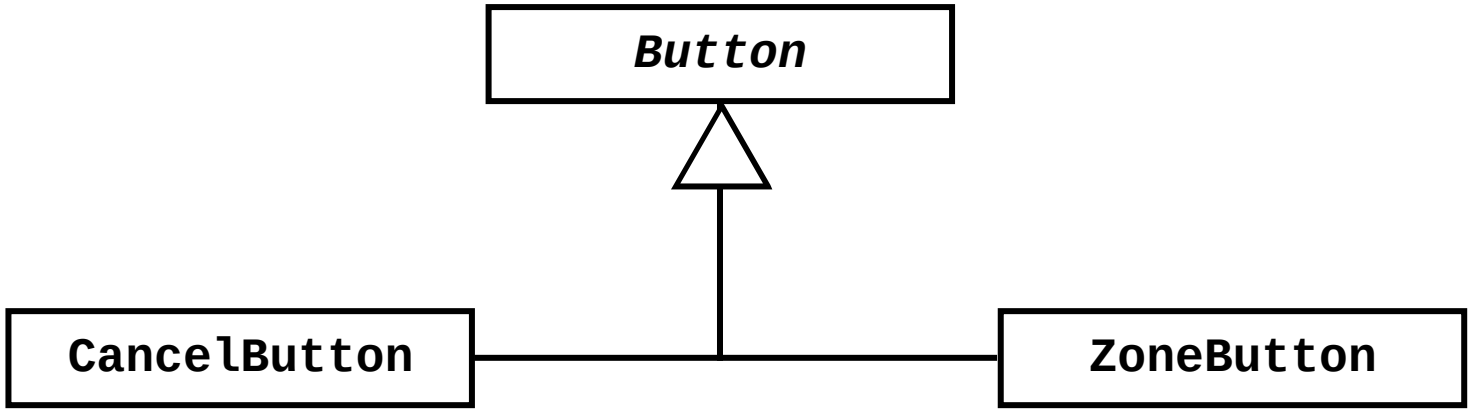


# *Composition*

- A solid diamond denote ***composition***, a strong form of aggregation where components cannot exist without the aggregate.



# *Generalization*




- Generalization relationships denote inheritance between classes.
- The children classes inherit the attributes and operations of the parent class.
- Generalization simplifies the model by eliminating redundancy.

# *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 {
    public Vector m_Company = new Vector();
};
public class Company {
    public int m_tickerSymbol;
    public Vector m_StockExchange = new Vector();
};
```