

# CMSC 132: Object-Oriented Programming II

---



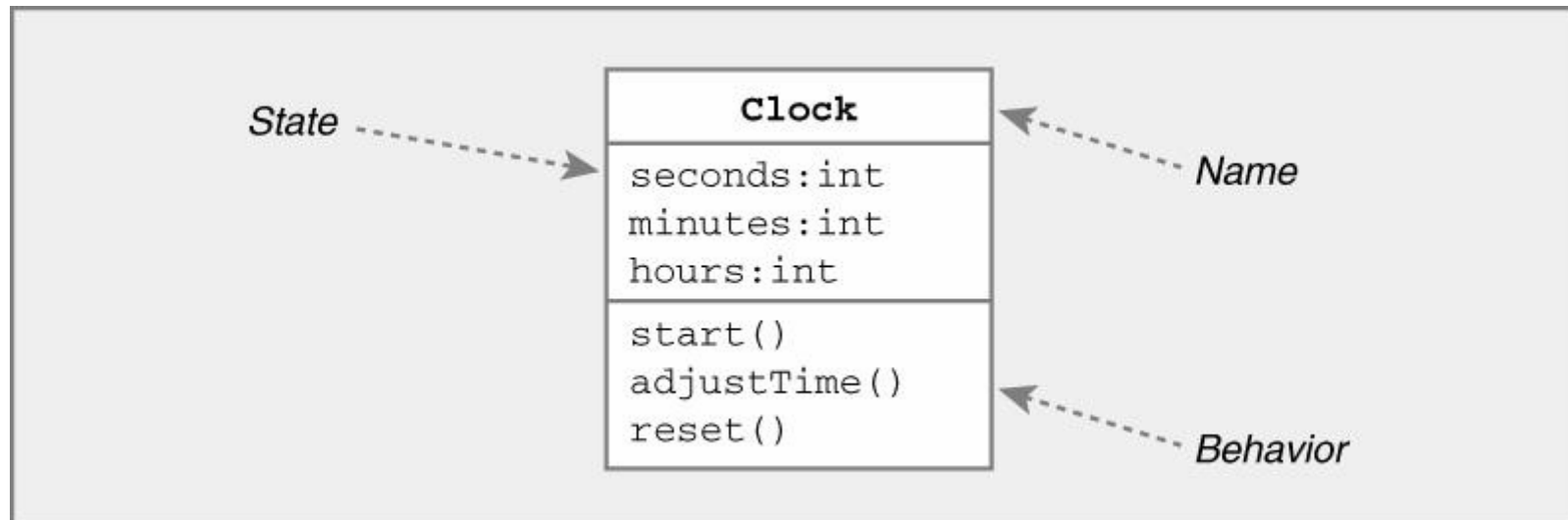
**Nelson Padua-Perez**  
**William Pugh**

**Department of Computer Science**  
**University of Maryland, College Park**





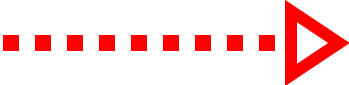
# UML Class Diagrams

- Represent the (static) structure of the system

■ General	In Java
■ Name	Name
■ State	Variables
■ Behavior	Methods

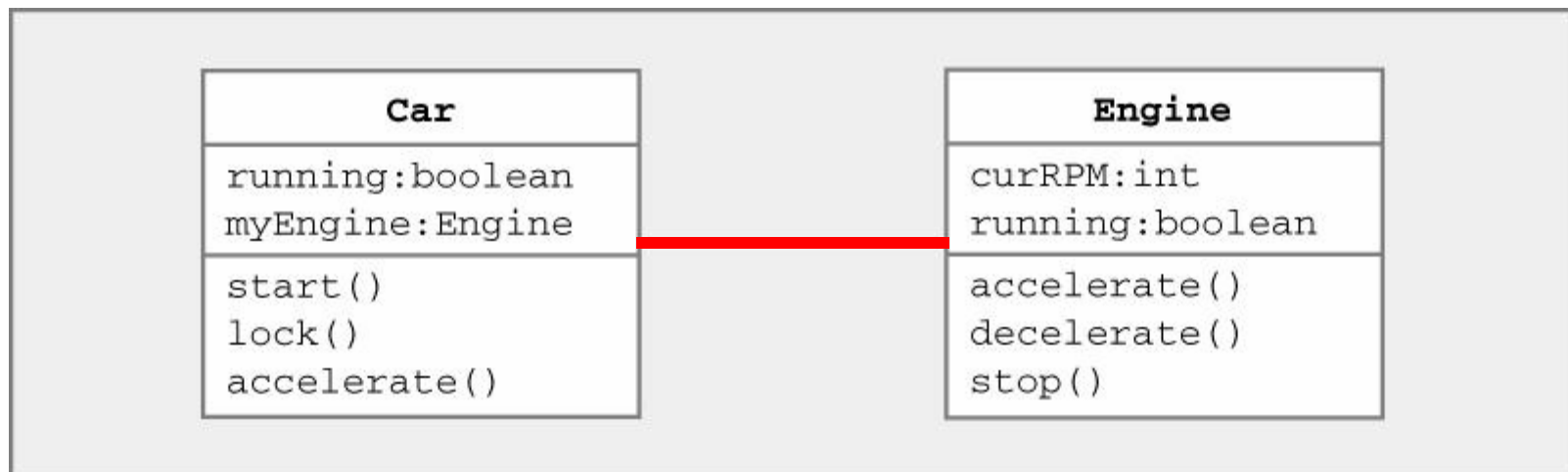


# Relationships Between Classes

- **Association**  OR 
  - Permanent, structural, “has a”
  - Solid line (arrowhead optional)
- **Dependency** 
  - Temporary, “uses a”
  - Dotted line with arrowhead
- **Generalization** 
  - Inheritance, “is a”
  - Solid line with open (triangular) arrowhead
- **Implementation** 
  - Dotted line with open (triangular) arrowhead

# Association

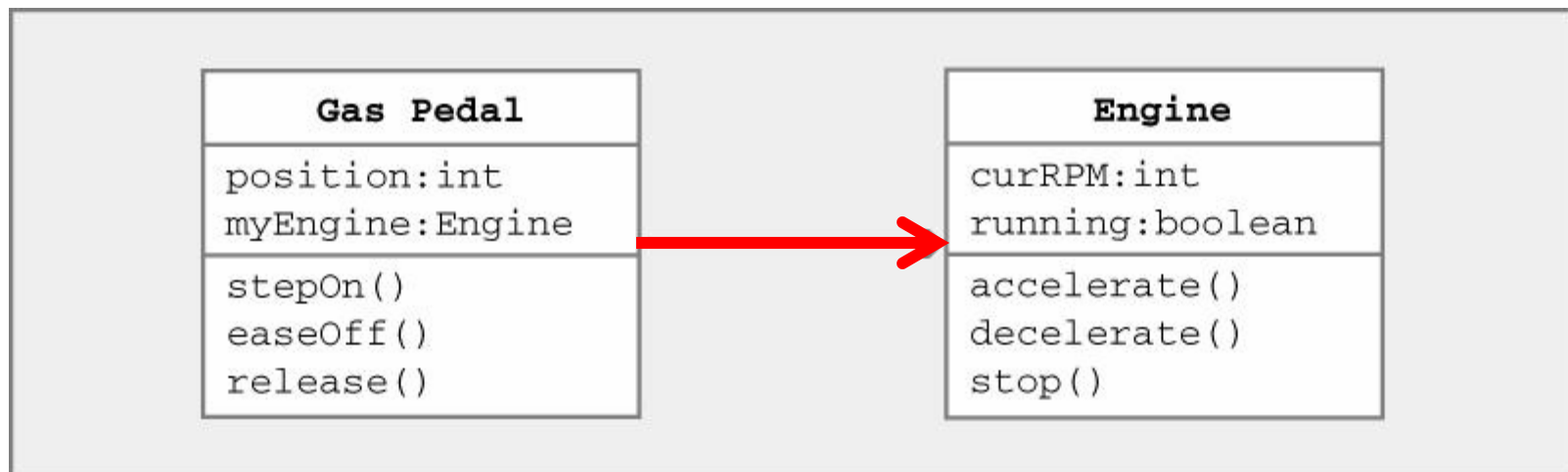
- Denotes permanent, structural relationship
- State of class A contains class B
- Represented by solid line (arrowhead optional)



**Car and Engine classes know about each other**

# Associations w/ Navigation Information

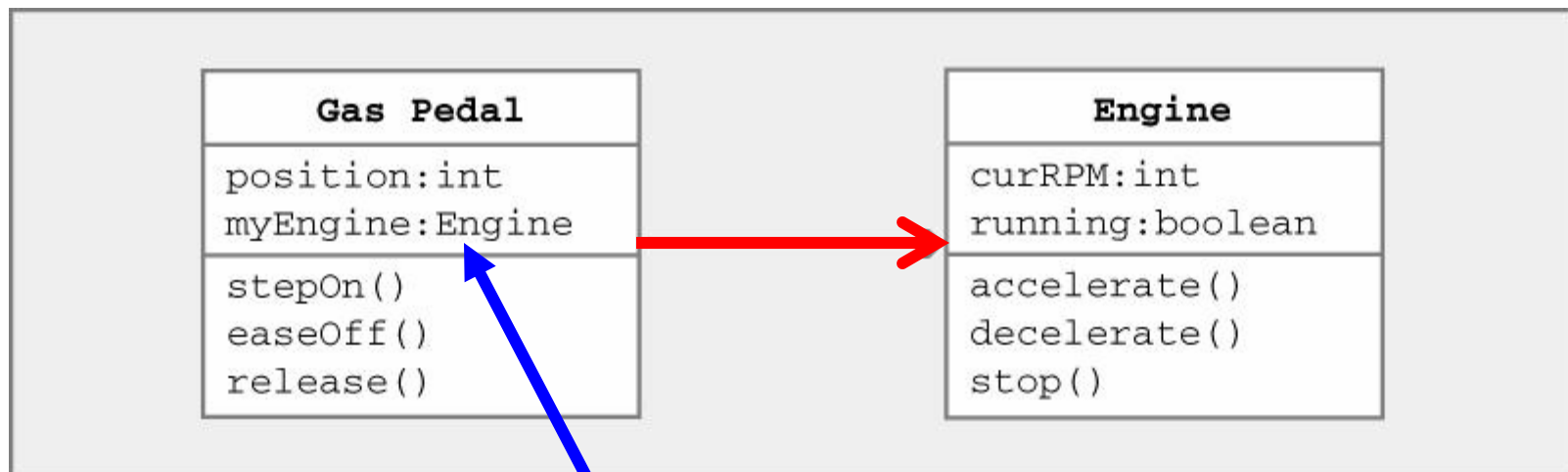
- Can indicate direction of relationship
- Represented by solid line with arrowhead



**Gas Pedal class knows about Engine class**  
**Engine class doesn't know about Gas Pedal class**

# Associations w/ Navigation Information

- Denotes “**has-a**” relationship between classes
- “**Gas Pedal**” **has a** “**Engine**”



**State of Gas Pedal class contains instance of Engine class ⇒ can invoke its methods**

# Multiplicity of Associations

- **Some relationships may be quantified**
- **Multiplicity denotes how many objects the source object can legitimately reference**
- **Notation**
  - \*           ⇒ 0, 1, or more
  - 5            ⇒ 5 exactly
  - 5..8        ⇒ between 5 and 8, inclusive
  - 5..\*        ⇒ 5 or more

# Multiplicity of Associations

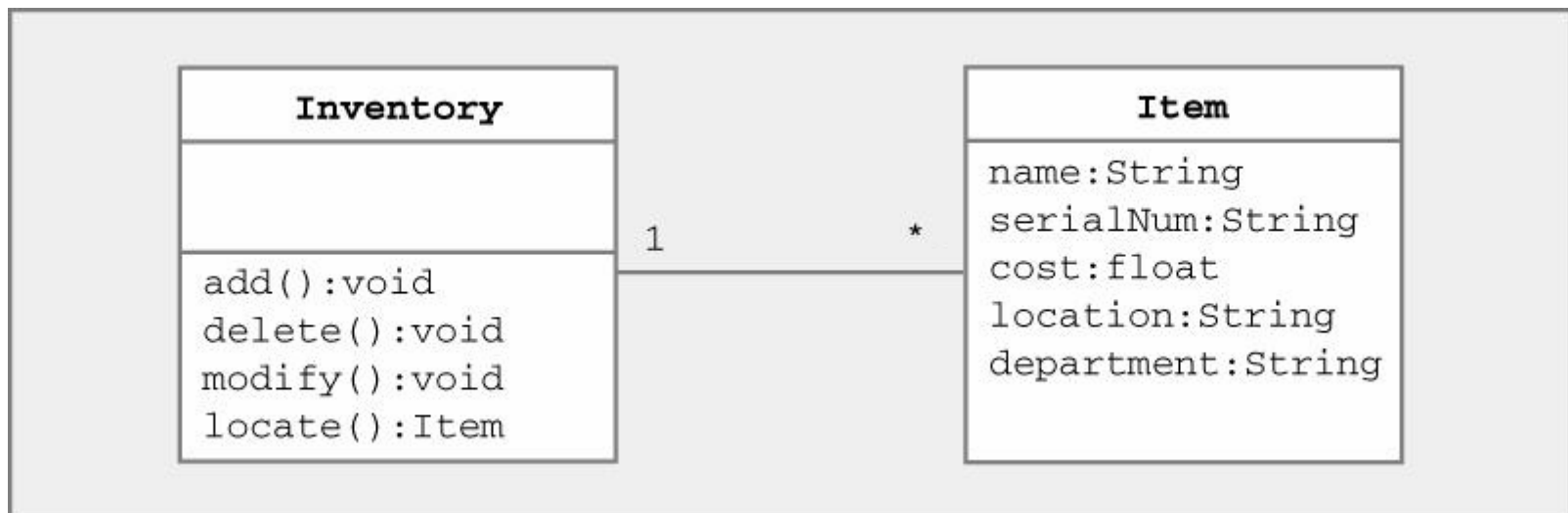
## ■ Many-to-one

- Bank has many ATMs, ATM knows only 1 bank



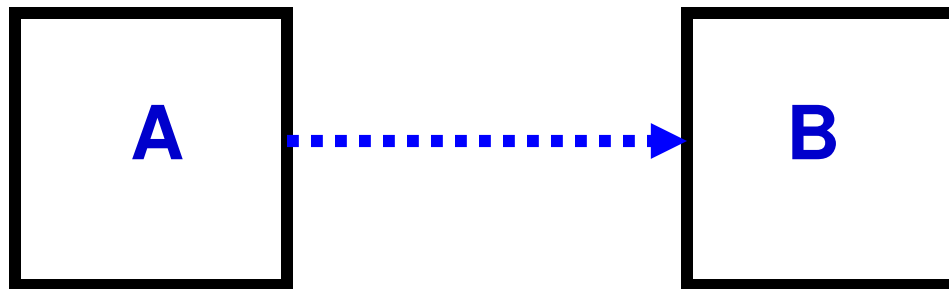
## ■ One-to-many

- Inventory has many items, items know 1 inventory



# Dependency

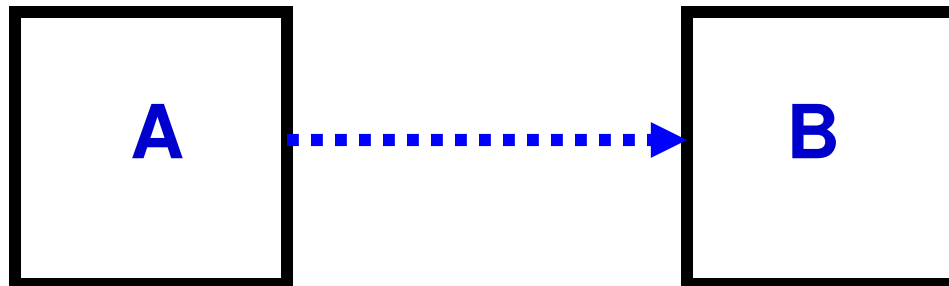
- Denotes **dependence** between classes
- Always directed (Class **A** depends on **B**)
- Represented by dotted line with arrowhead



A depends on B

# Dependency

- Caused by class methods
- Method in Class **A** temporarily “**uses a**” object of type Class **B**
- Change in Class **B** may affect class **A**




A uses object of class B

# Dependency

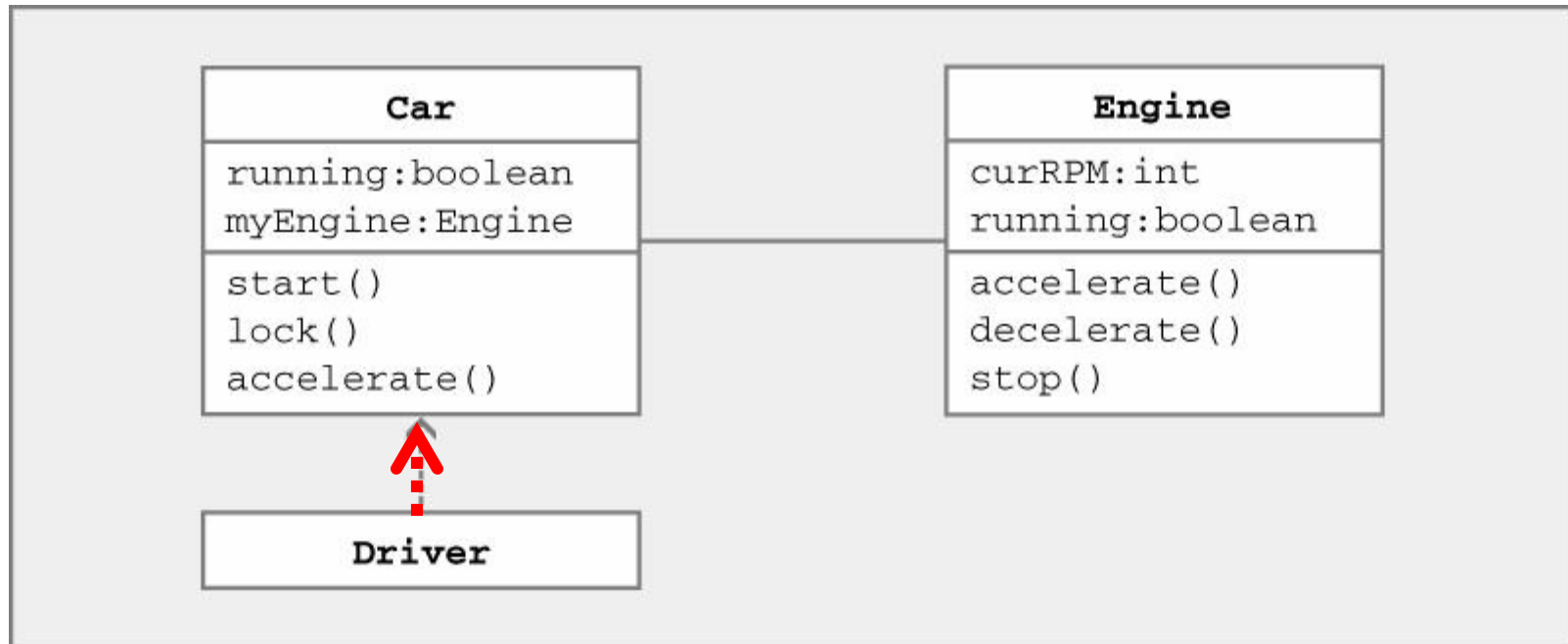
- Dependence may be caused by
  - Local variable
  - Parameter
  - Return value
- Example

```
Class A {  
    B Foo(B x) {  
        B y = new B();  
        return y;  
    }  
}
```

```
Class B {  
    ...  
    ...  
    ...  
}
```



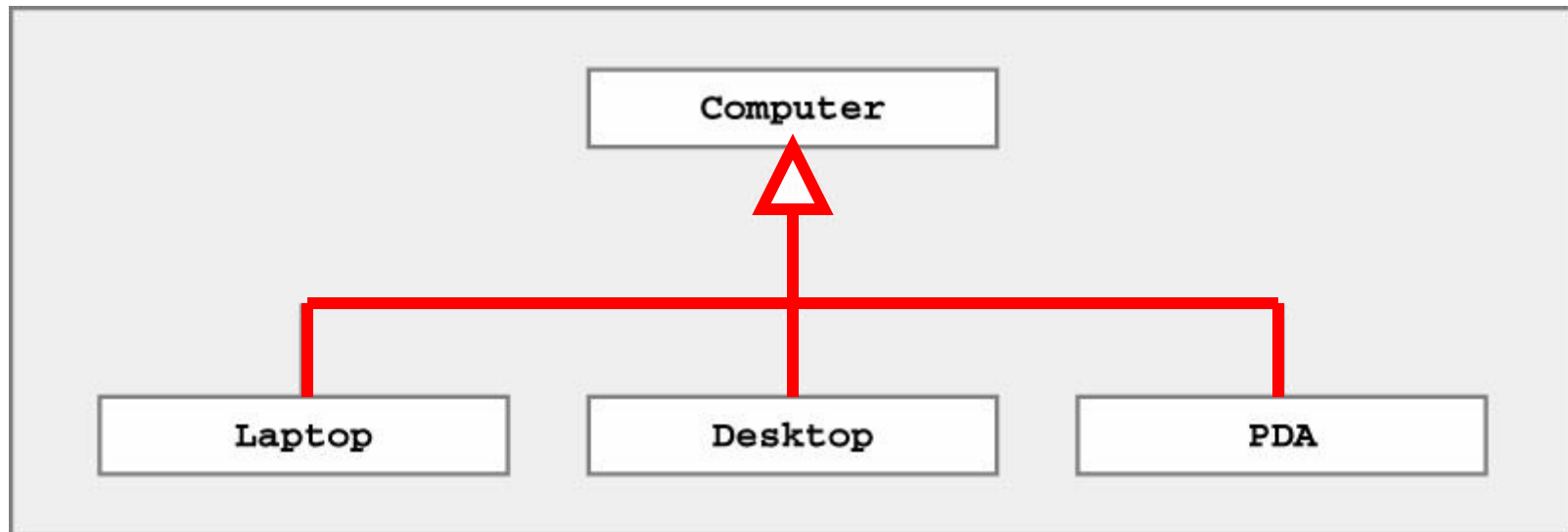
# Dependency Example



**Class Driver depends on Class Car**

# Generalization

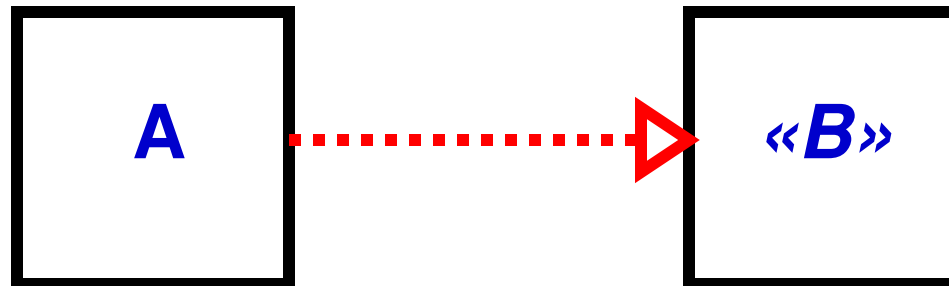
- Denotes **inheritance** between classes
- Can view as “**is-a**” relationship
- Represented by line ending in (open) triangle



**Laptop, Desktop, PDA inherit  
state & behavior from Computers**

# Implementation

- Denotes class **implements** Java interface
- Represented by dotted line ending in (open) triangle



A implements interface B

# UML Examples

- **Read UML class diagram**
- **Try to understand relationships**
- **Examples**
  - **Pets & owners**
  - **Computer disk organization**
  - **Banking system**
  - **Home heating system**
  - **Printing system**

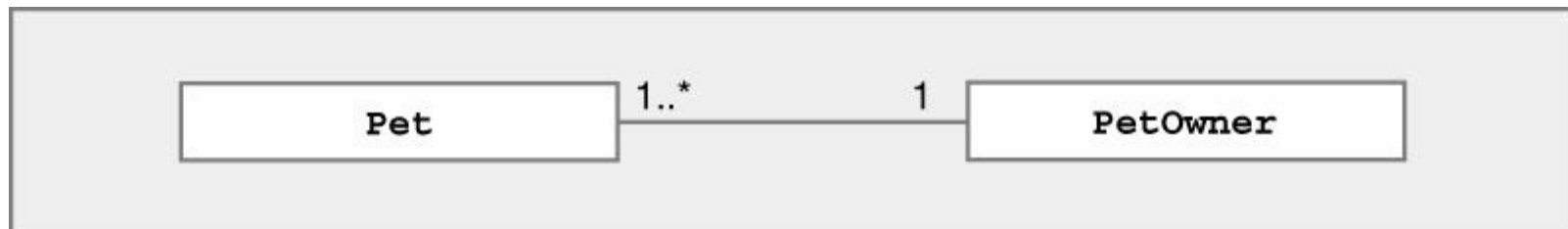
# UML Example – Veterinary System

- Try to read & understand UML diagram



# UML Example – Veterinary System

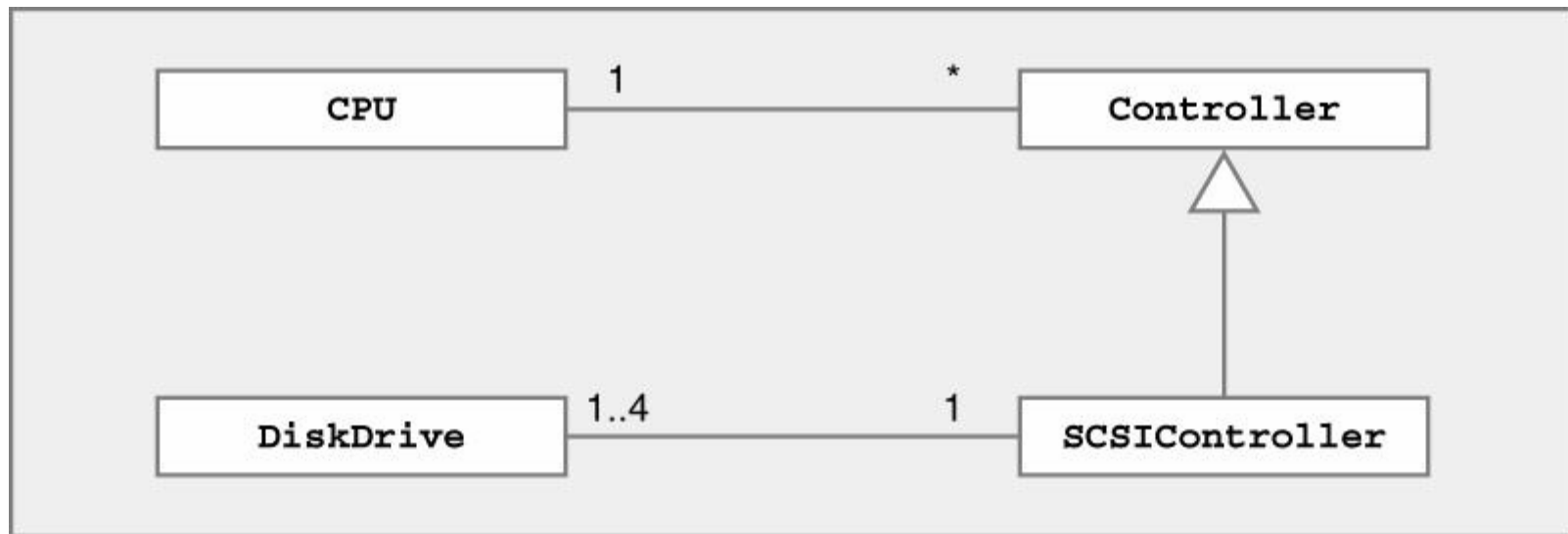
- Try to read & understand UML diagram



- 1 or more Pets associated with 1 PetOwner

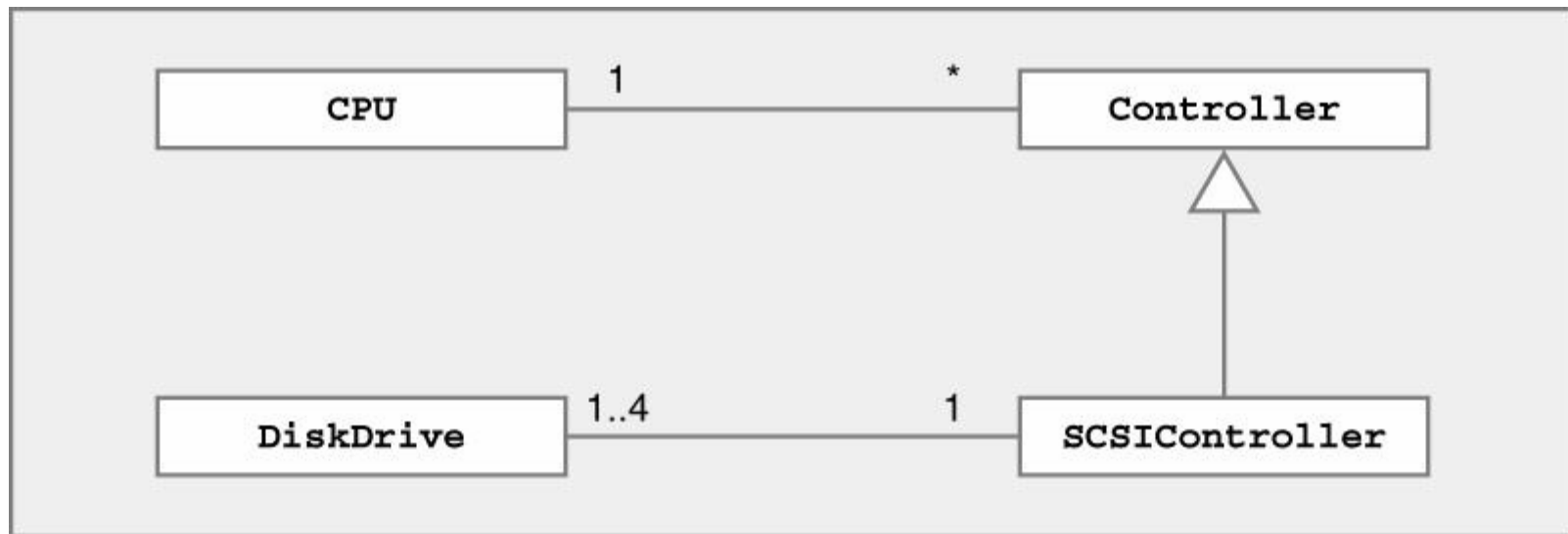
# UML Example – Computer System

- Try to read & understand UML diagram



# UML Example – Computer System

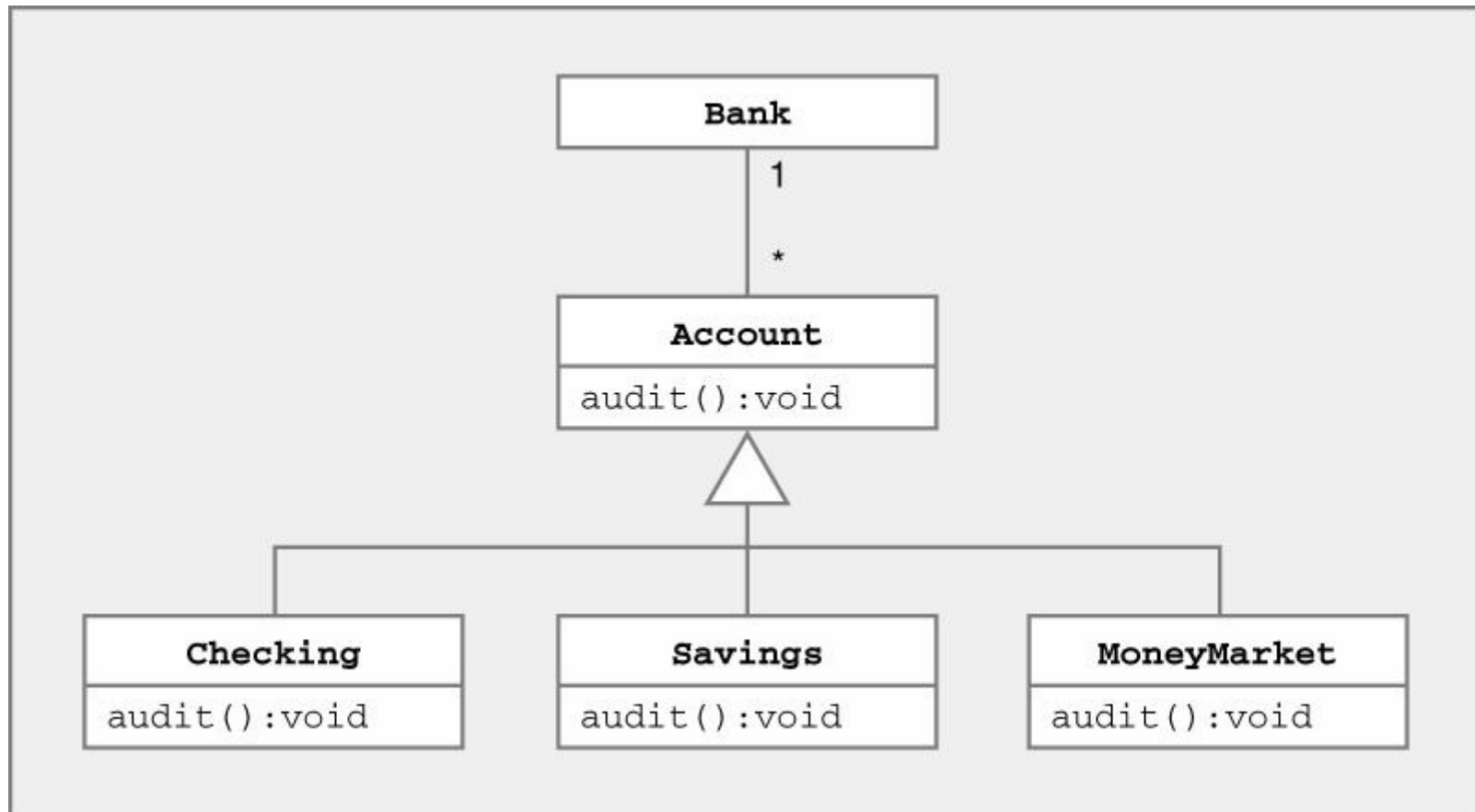
- Try to read & understand UML diagram



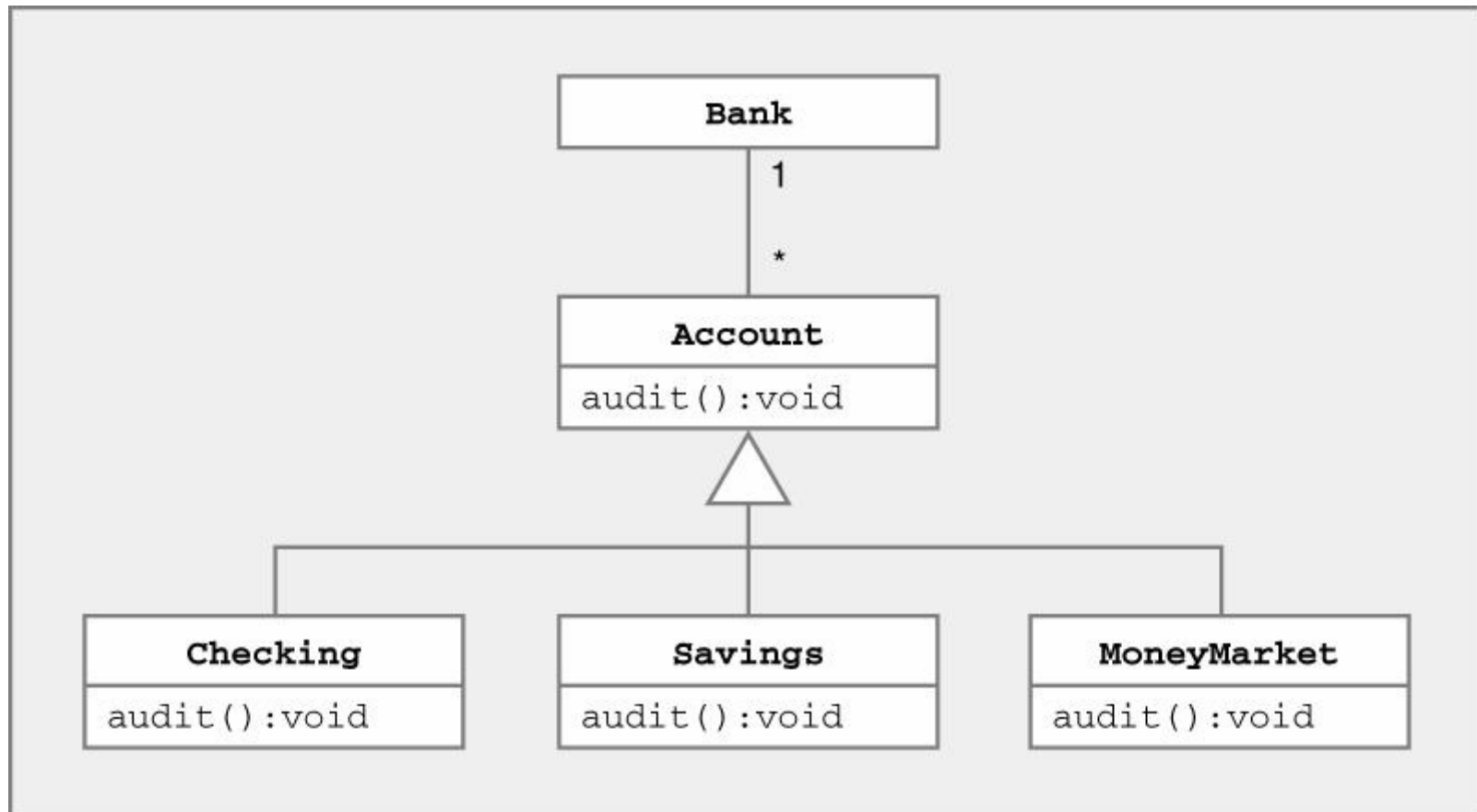
- 1 CPU associated with 0 or more Controllers
- 1-4 DiskDrives associated with 1 SCSIController
- SCSIController is a (specialized) Controller

# UML Example – Banking System

- Try to read & understand UML diagram



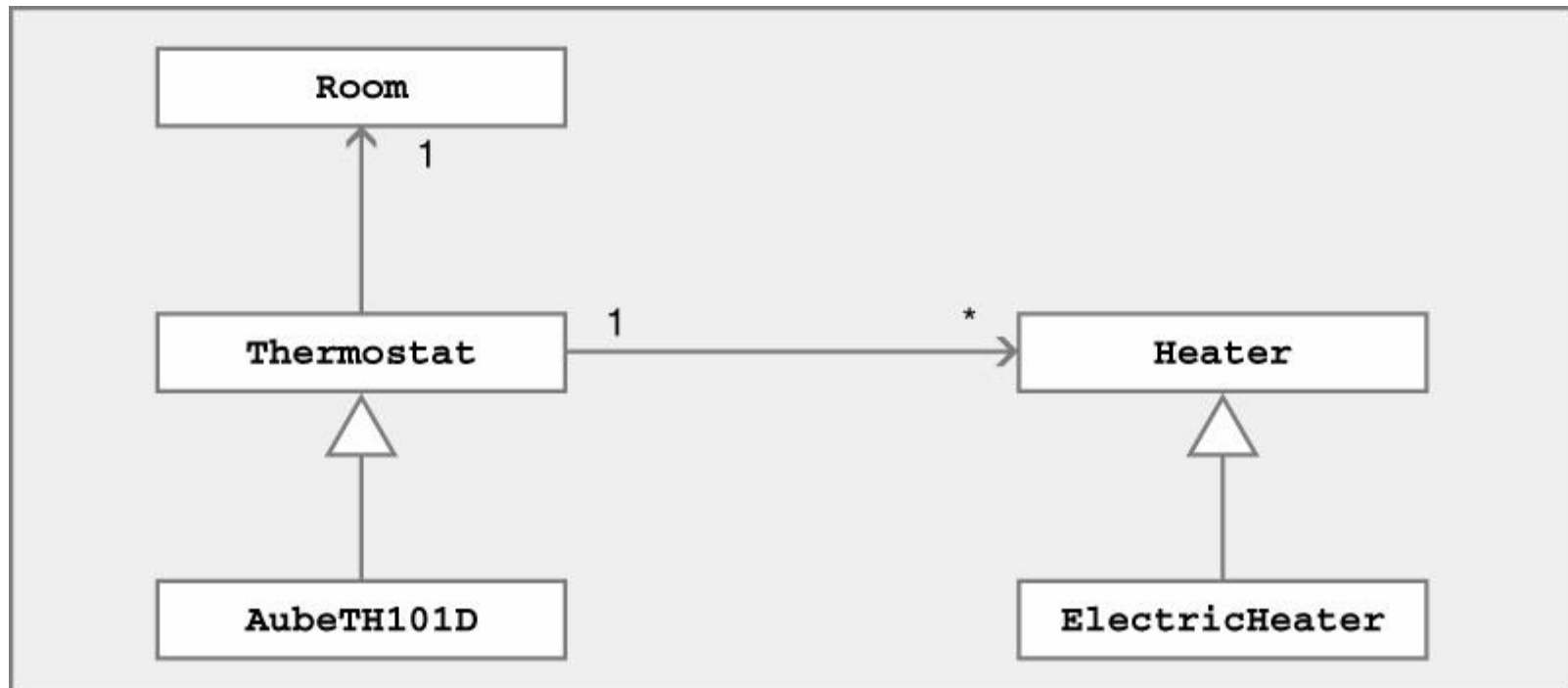
# UML Example – Banking System



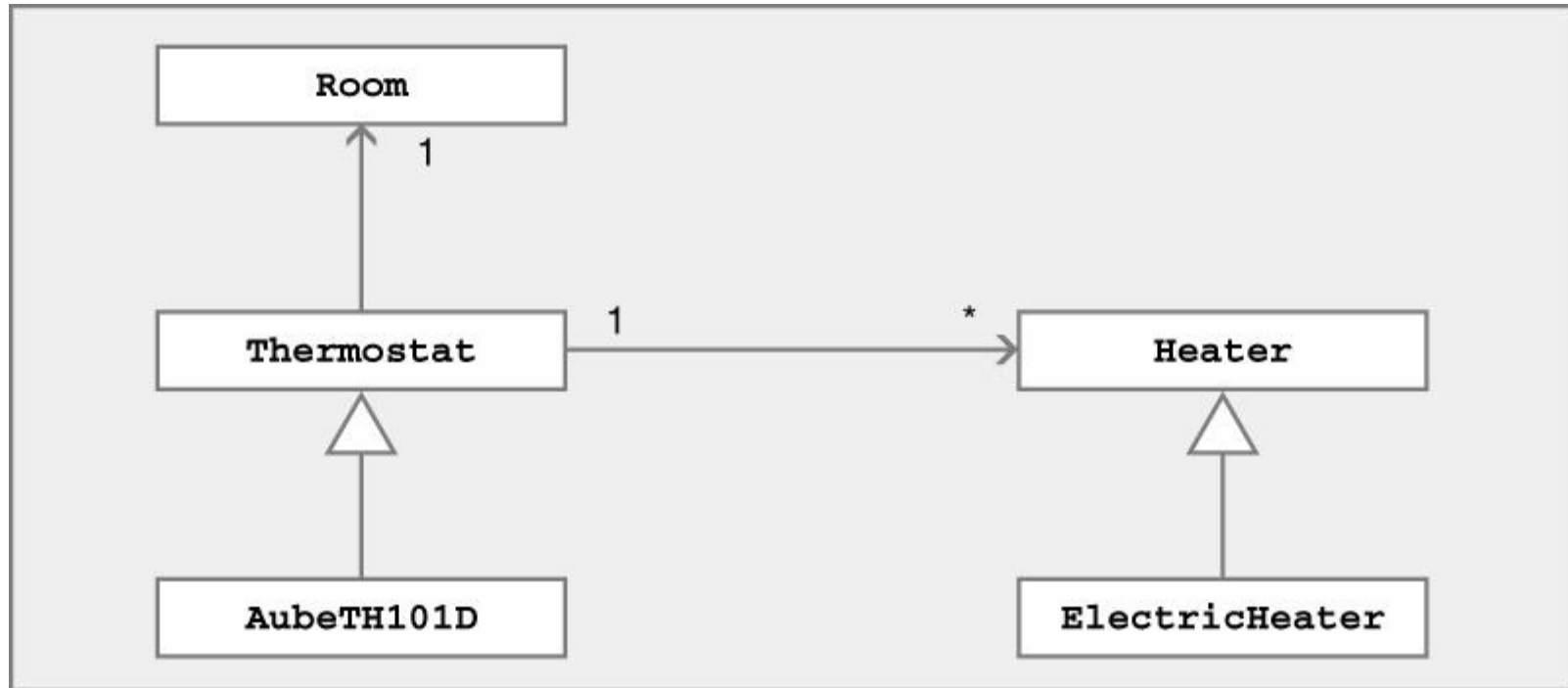
- **1 Bank associated with 0 or more Accounts**
- **Checking, Savings, MoneyMarket are Accounts**

# UML Example – Home Heating System

- Try to read & understand UML diagram



# UML Example – Home Heating System



- Each Thermostat has 1 Room
- Each Thermostat associated with 0 or more Heaters
- ElectricHeater is a specialized Heater
- AubeTH101D is a specialized Thermostat

# UML Class Diagrams ↔ Java

- Different representation of **same** information
  - Name, state, behavior of class
  - Relationship(s) between classes
- Practice deriving one from the other
  - Accurately depicting relationship between classes

# UML → Java : Veterinary System

## ■ UML



## ■ Java



# UML → Java : Veterinary System

## ■ UML



## ■ Java

```
class Pet {
    PetOwner myOwner;
}
```

// 1 owner for each pet

```
class PetOwner {
    Pet [ ] myPets;
}
```

// multiple pets for each owner



# Java → UML : Veterinary System

## ■ Java

```
class Pet {  
    PetOwner myOwner;           // 1 owner for each pet  
}  
class PetOwner {  
    Pet [ ] myPets;             // multiple pets for each owner  
}
```



## ■ UML

# Java → UML : Veterinary System

## ■ Java

```
class Pet {  
    PetOwner myOwner;           // 1 owner for each pet  
}
```

```
class PetOwner {  
    Pet [ ] myPets;           // multiple pets for each owner  
}
```



## ■ UML



# UML Class Diagrams ↔ Java

## ■ UML



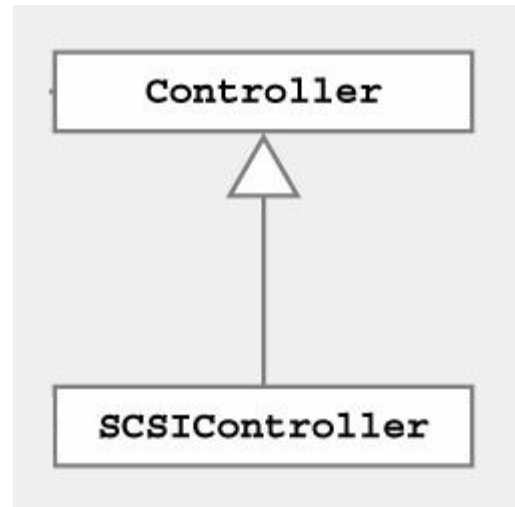
## ■ Java

```
class Pet {
    PetOwner myOwner;           // 1 owner for each pet
}
class PetOwner {
    Pet [ ] myPets;           // multiple pets for each owner
}
```



# UML → Java : Computer System

## ■ UML

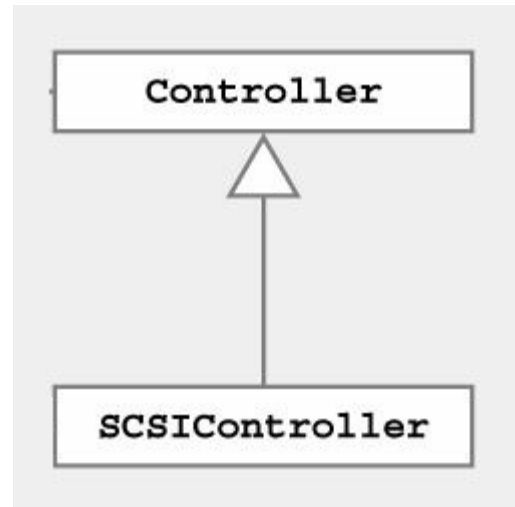


## ■ Java



# UML → Java : Computer System

## ■ UML

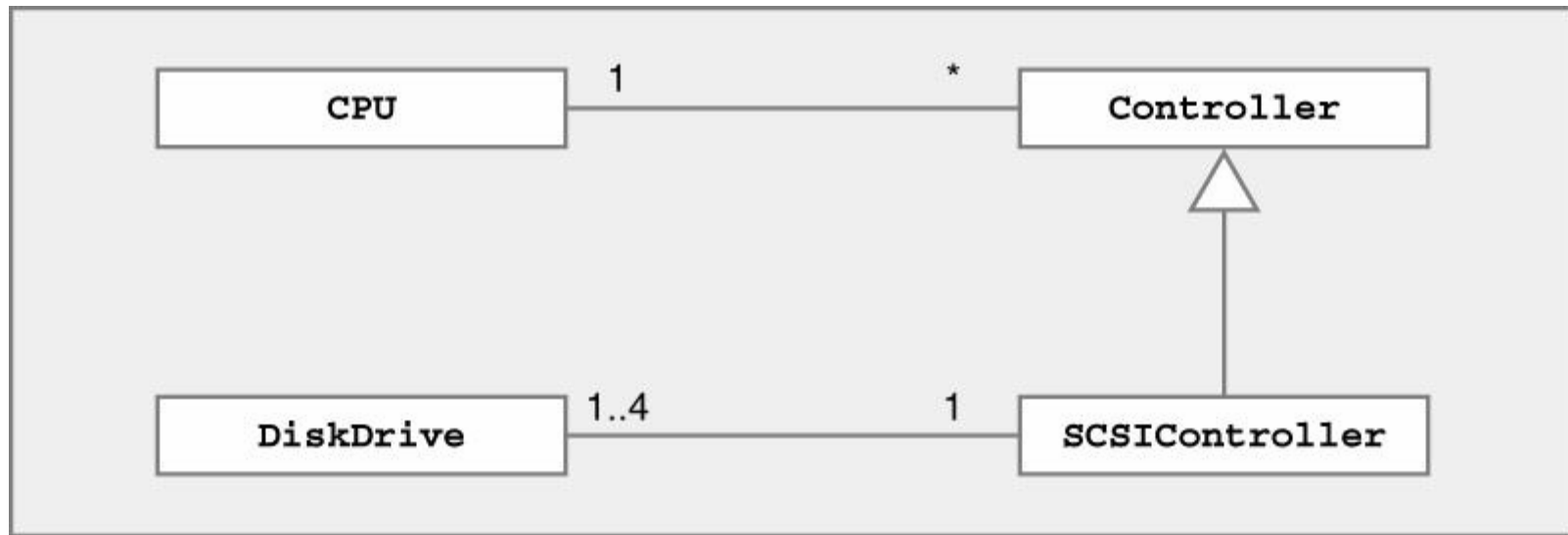


## ■ Java

`class Controller {`  
`}`  
`class SCSCIController extends Controller {`  
`}`

# UML → Java : Computer System

## ■ UML



## ■ Java

- Design code using all available information in UML...

# UML → Java : Computer System

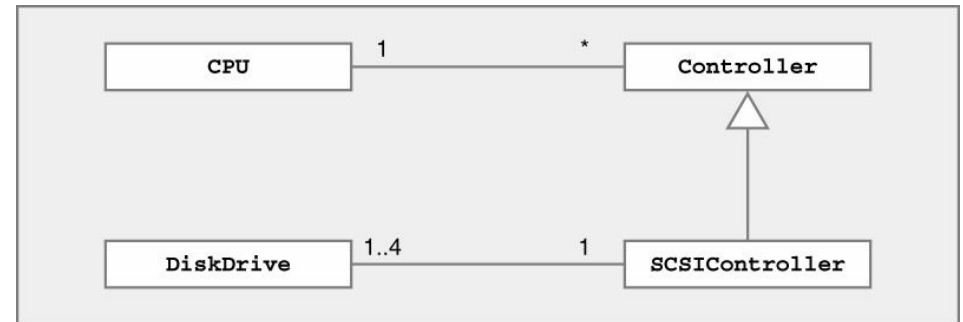
## ■ Java

```
class CPU {  
    Controller [ ] myCtrls;  
}
```

```
class Controller {  
    CPU myCPU;  
}
```

```
class SCSIController extends Controller {  
    DiskDrive [ ] myDrives = new DiskDrive[4];  
}
```

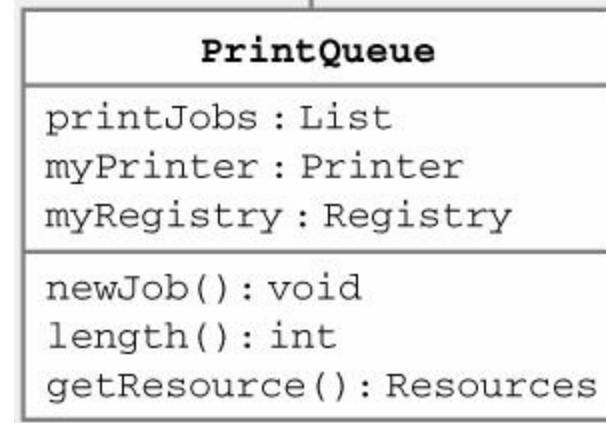
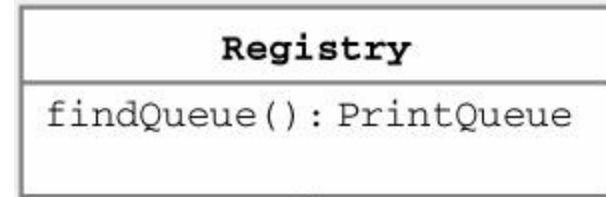
```
Class DiskDrive {  
    SCSIController mySCSI;  
}
```



# Java → UML : Printing System

## ■ Java

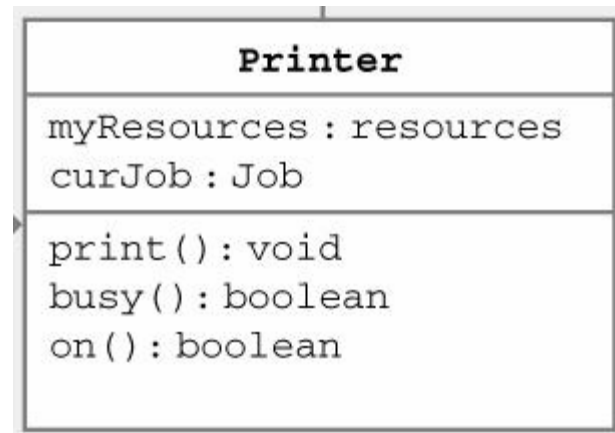
```
class Registry {  
    PrintQueue findQueue();  
}  
class PrintQueue {  
    List printJobs;  
    Printer myPrinter;  
    Registry myRegistry;  
    void newJob();  
    int length();  
    Resources getResource();  
}
```



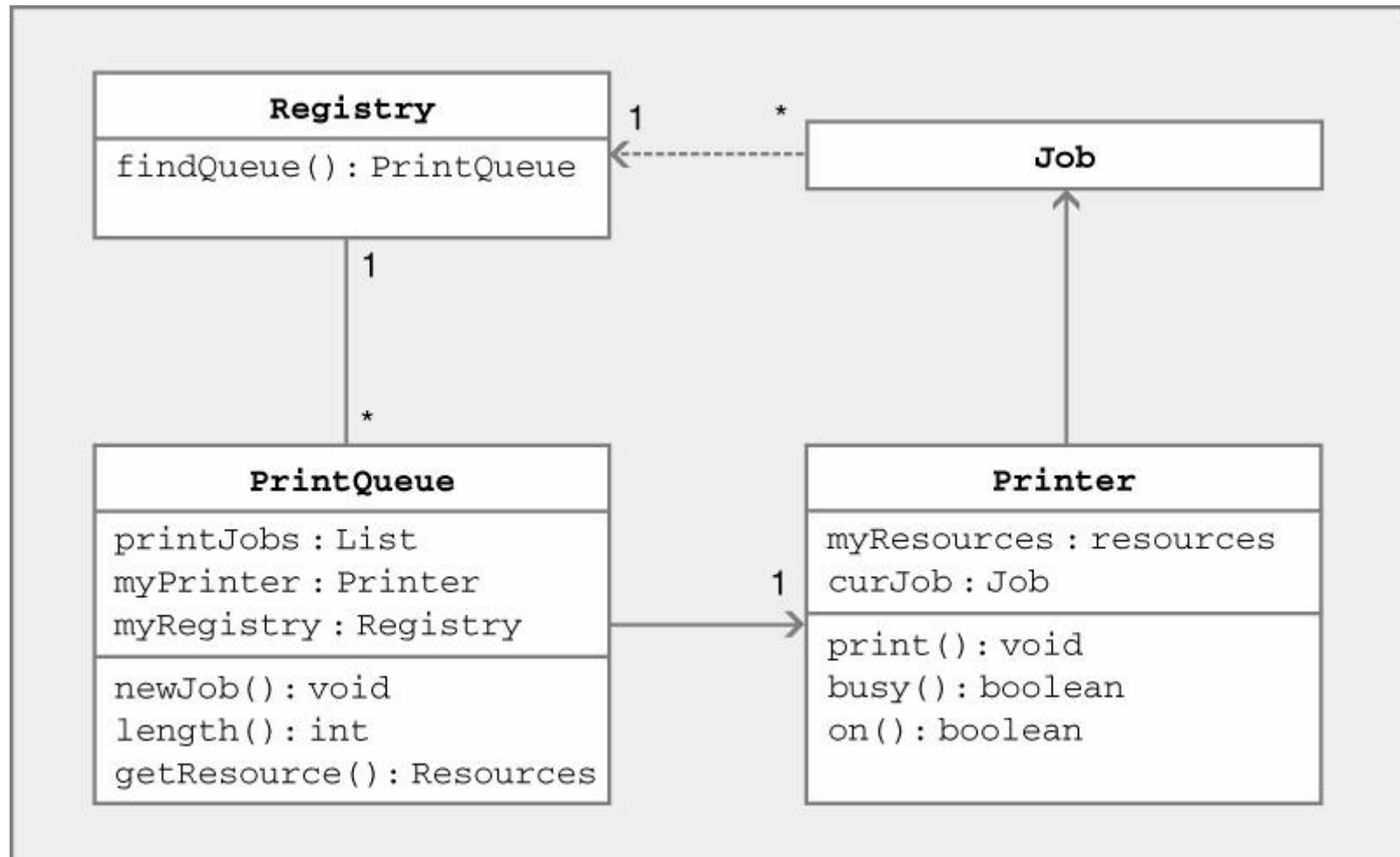
# Java → UML : Printing System

## ■ Java

```
Class Printer {  
    Resources myResources;  
    Job curJob;  
    void print();  
    boolean busy();  
    boolean on();  
}  
class Job {  
    Job(Registry r) {  
        ...  
    }  
}
```



# Java → UML : Printing System



# UML Summary

- Graphics modeling language
- Visually represents design of software system
- We focused on **class diagrams**
  - Contents of a class
  - Relationship between classes
- You should be able to
  - Draw UML class diagram given Java code
  - Write Java code given UML class diagram