COM343 Object Oriented Programming

Essence of Objects

Introduction

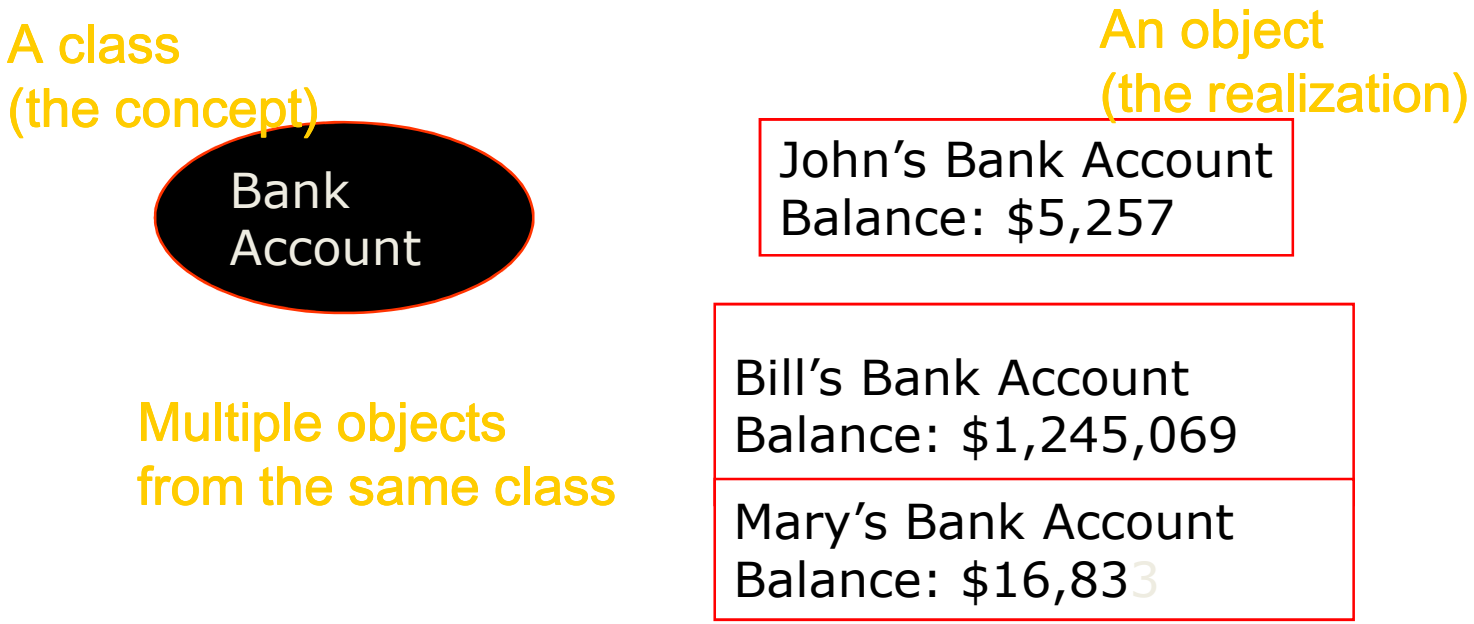
- An **object** is a software model of a real-world entity or concept
  - E.g. a temperature sensor, a color, a list, etc.
- Every object has a set of attributes such as value, state, etc.
  - Sensor can be active/inactive, it can have a value representing its reading, etc.
- Objects also provide facilities to modify their states
  - Sensor can be turned on/off

Introduction

- Objects have responsibilities
  - Carried out by providing services to other objects
- Often, it is better to think about an object in terms of its responsibilities instead of its attributes
  - Outside objects don't care how a sensor is implemented, but rather what services it provides

Classes

- A class represents a concept, and an object represents the embodiment of a class
- A class can be used to create multiple objects

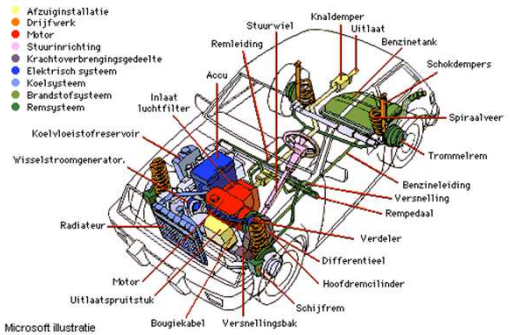


OO System

- Any object-oriented software system will have the following:
  - Abstraction with objects
  - Encapsulated classes
  - Communication via messages
  - Object lifetime
  - Class hierarchies
  - Polymorphism

Abstraction

- An *abstraction* hides (or ignores) unnecessary details
- denotes the essential properties of an object
- One of the fundamental ways in which we handle complexity
  - Objects are abstractions of real world entities
- Programming goal: choose the right abstractions



Abstraction

A car consists of four wheels an engine, accumulator and brakes.

Multiple Abstractions

A single thing can have multiple abstractions

Example: a protein is…

- a sequence of amino acids
- a complicated 3D shape (a fold)
- a surface with “pockets” for ligands

Choosing Abstractions

Abstractions can be about

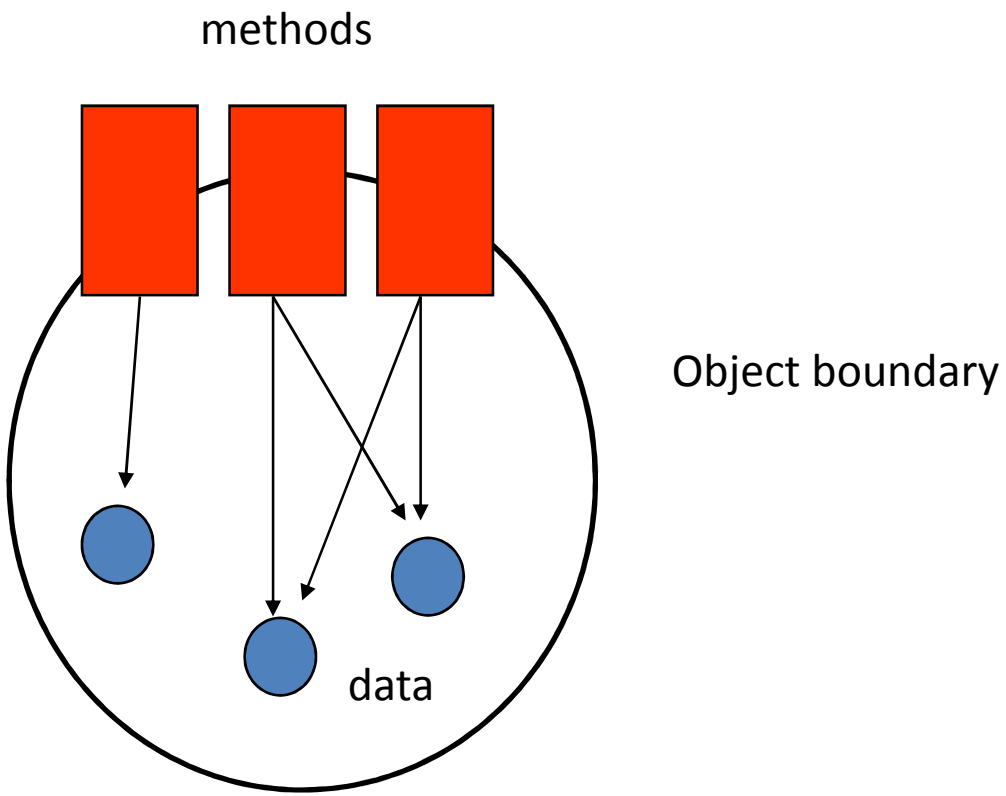
- tangible things (a vehicle, a car, a map) or
- intangible things (a meeting, a route, a schedule)

An example:

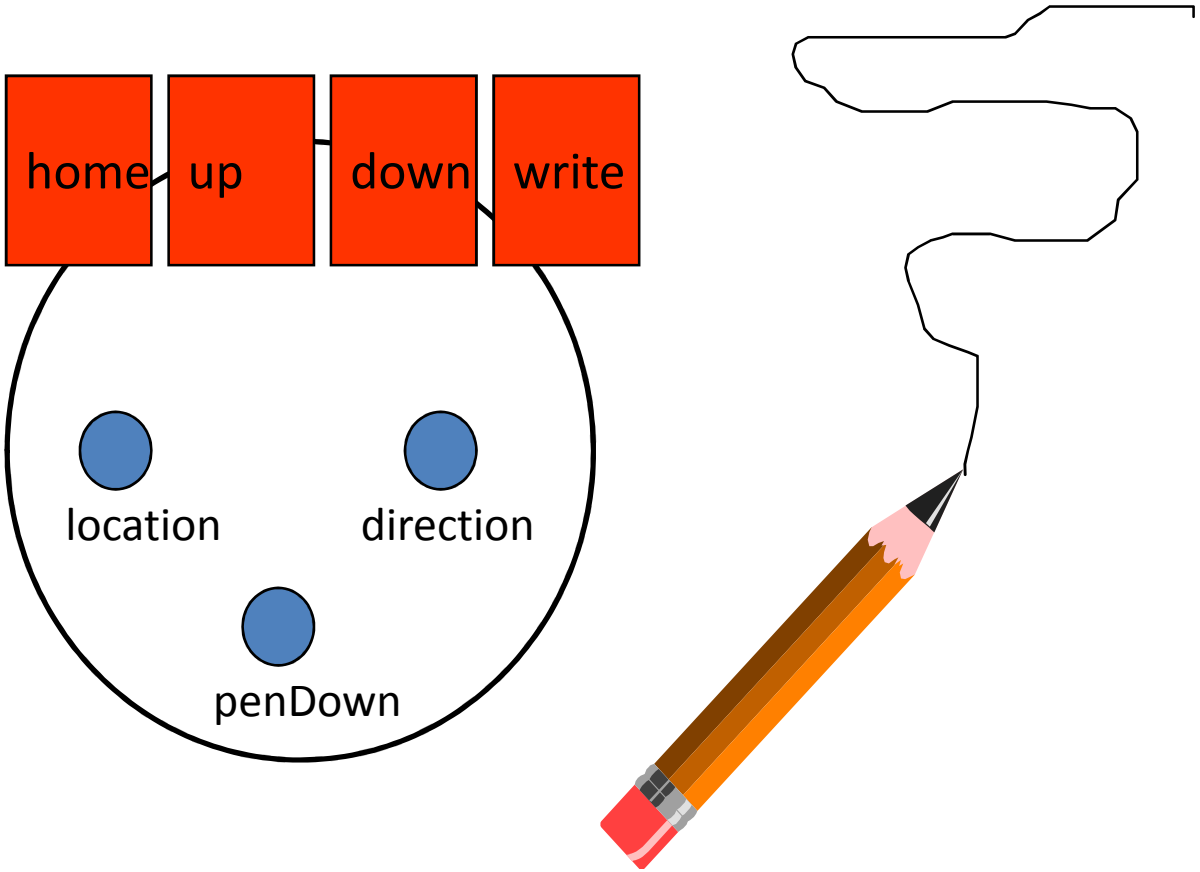
- Abstraction name: light
- Light's wattage (i.e.,energy usage)
- Light can be on or off

- There are other possible properties (shape, color, socket size, etc.), but we have decided those are less essential
- The essential properties are determined by the problem

Object-Oriented Model

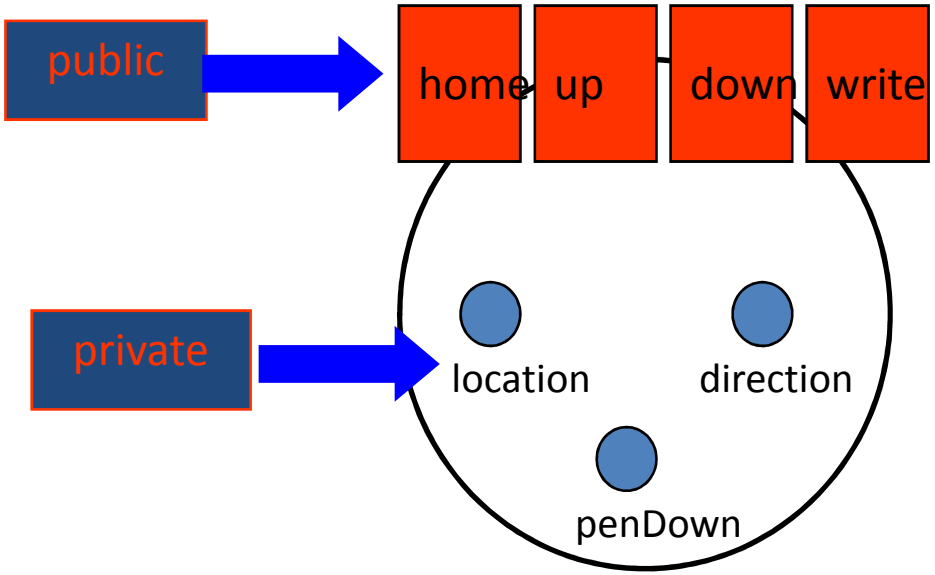


Example: Pencil



Encapsulation

- the data belonging to an object is hidden, so variables are *private*
- methods are *public*
- we use the public methods to change or access the private data
- No dependence on implementation



Programming Implications

- Encapsulation makes programming easier
  - As long as the contract is the same, the client doesn't care about the implementation

- In Java, as long as the method signatures are the same, the implementation details can be changed
  - In other words, I can write my program using simple implementations; then, if necessary, I can replace some of the simple implementations with efficient implementations

Interaction via Messages

- To accomplish useful tasks, objects need to interact with each other
- Interaction can be between objects of the same class or objects from different classes
- This is done by sending messages (calling methods) to others to pass info or request action

Object Lifetime

- All objects have a lifetime.
- They are
  - created and initialized as they are needed during program execution, (**instantiation**)
  - exist and carry out their functions,
  - and eventually destroyed (**garbage collection** in Java)
- Many objects from the same class can exist
- Every object has a unique identity
  - Java uses **references** to keep track of individual objects

Class Hierarchies

- In OOD, classes are arranged into hierarchies which model and describe relationships among classes
  - Association
    - Person – company
  - Aggregation (or composition)
    - GUI contains buttons, sliders, boxes, etc.
  - Inheritance
    - Mountain bike is a specialized case of a more general bike class

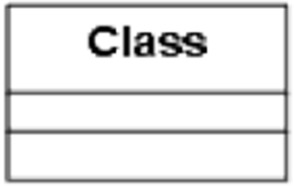
Polymorphism

- When inheritance is used, the specialized class usually extends some of the behaviors of the generalized class
- Name used to define the behavior will be same but the behavior will be somewhat different
- It is important that an object uses the correct behavior, and polymorphism allows this to happen automatically and seamlessly (via **dynamic binding**)
- More on this later.

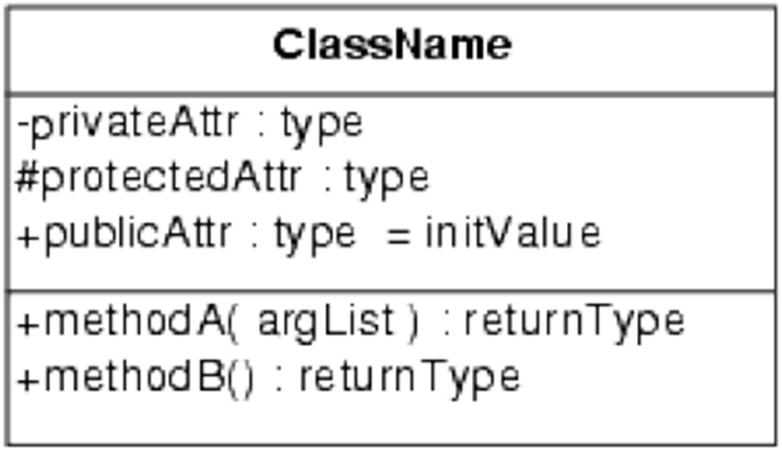
Properties of OO Systems

- If a system doesn't include abstraction, encapsulation, messages, lifetime, hierarchies, and polymorphism, then it isn't OO, even if it is written in Java, C#, C++, or some other OO language.

Class Diagrams

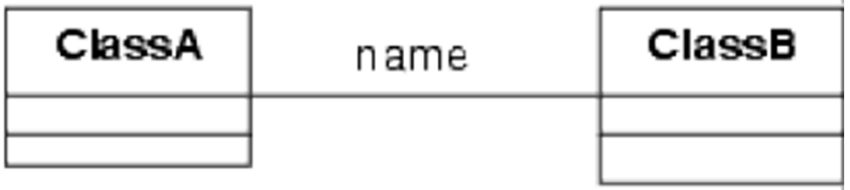


Simple Class View

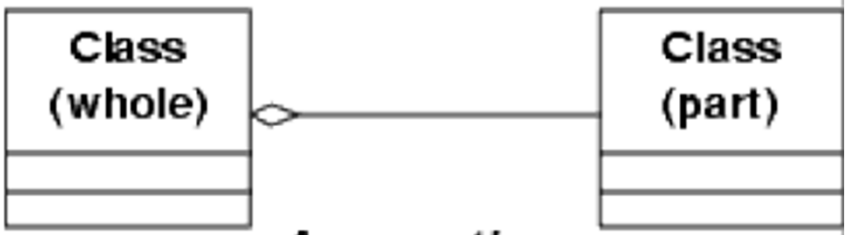


Detailed Class View

Class Hierarchies



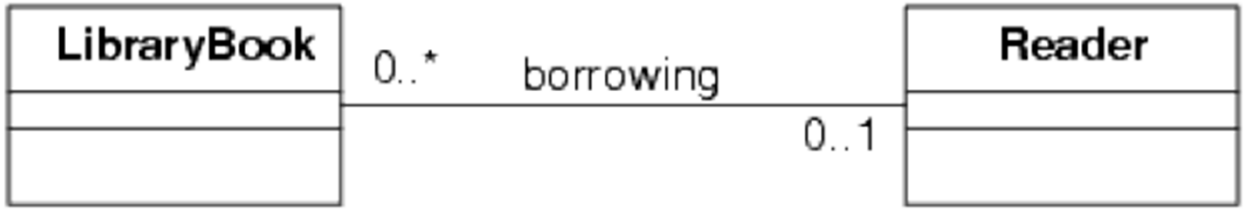
Association



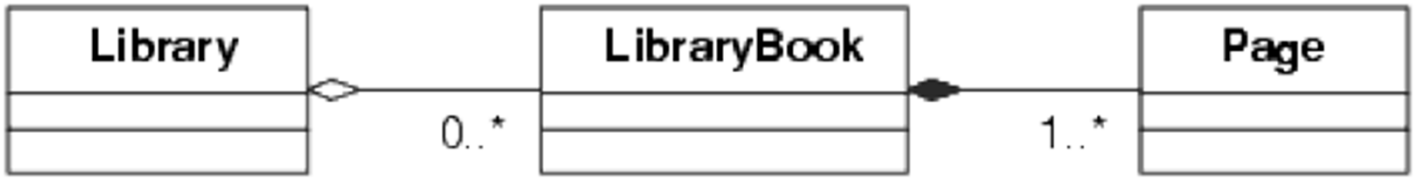
Aggregation

Composition

Associations & Hierarchies

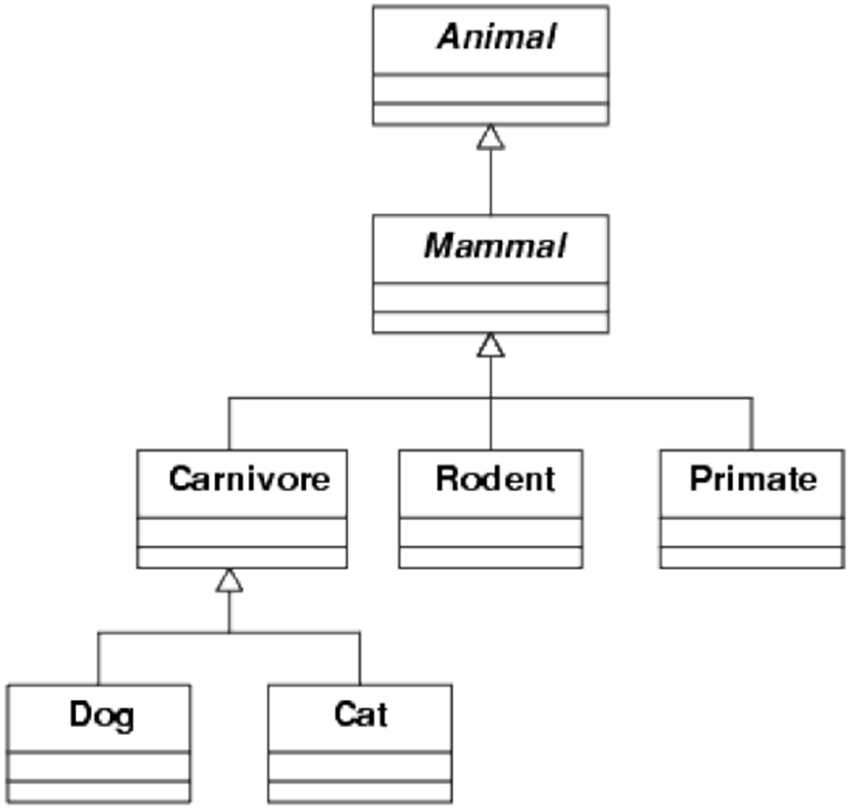


- Associations show relationship between different, independent classes
- Associations can have **multiplicity**
  - 0 .. *, 0 .. 1, 0 .. 4, etc.
- Two common structures in hierarchies: **whole/part** and **generalization/specialization**
- **Whole/part** is a **has-a** (or **part-of**) relationship
  - Library has books which have pages (**aggregation** and **composition**)
- A library has books but books come and go. There can be an empty library without any books. But there cannot be a book without any pages.

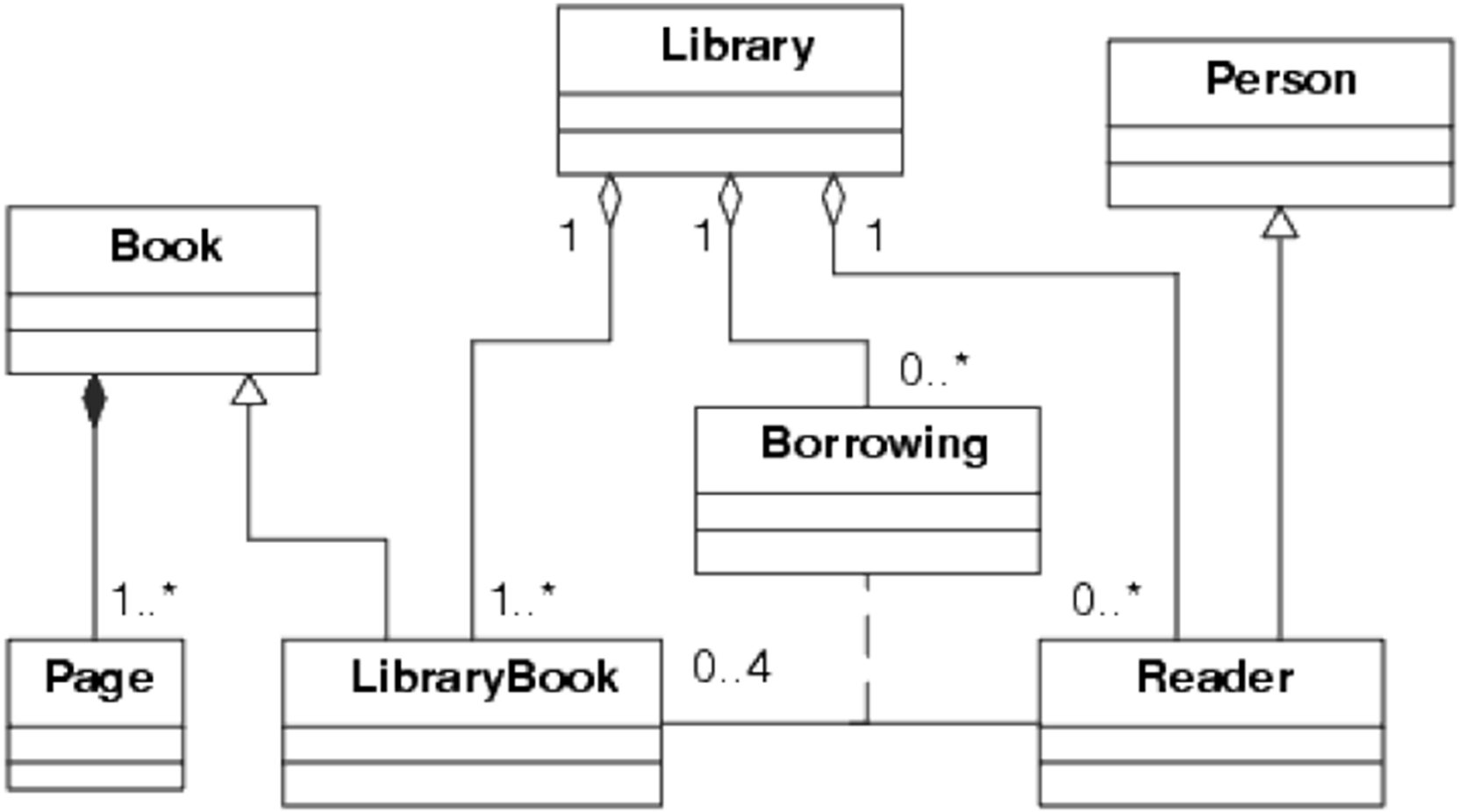


Associations & Hierarchies (2)

- Generalization/specialization is characterized by **is-a** relationship
  - A dog is a carnivore, a carnivore is a mammal, and so on
- **Inheritance:** a new **subclass** is **derived** from an existing parent **superclass**
- Topmost class is called the **root class**
- In Java, a subclass can inherit from only one superclass (**single inheritance**)
- In others, **multiple inheritance** is possible
  - Java makes up for this with **interfaces**



An Example – Putting it Altogether



Other OO Concepts

- **Abstract classes**
  - In a class hierarchy, it is common to have classes that will never have any instances
  - They are only there to be inherited from.
  - E.g. Animal class
  - Opposite of **concrete classes**

Other OO Concepts

- **Visibility** of attributes and methods
  - 4 levels
    - **Public.** Public attributes and methods are visible from anywhere (+ in UML)
    - **Private.** Visible only to members of the given class (- in UML)
    - **Protected.** Visible to the class and its subclasses (# in UML)
    - **Package (default).** Visible to classes in the same package (# in UML)
- Attributes are commonly declared private (and thus direct access is prevented)
- General convention is to use **setter & getter** methods (accessors and mutators) to access or modify attributes

Other OO Concepts

- **Class vs. Instance**
  - 2 types of attributes or methods can exist
    - **Instance attributes/methods.** Those that relate to instances of the class (each object)
    - **Class attributes/methods.** Those which apply to the class as a whole (**static** keyword)

Other OO Concepts

- **Container.** a class whose instances are collections of other objects
  - Lists, stacks, queues, …
- **Iterator.** Objects commonly used with container classes which are used to access every element in order
- **Mix-in.** a class (an interface in Java) that is used to define a single behavior
- **Callback.** A method which is called when an event has taken place

Other OO Concepts

- **Listener.** An object that responds to events. A listener typically invokes callback methods
- **link.** A reference to another class. Used to build associations between classes
- **this.** Also called **self.** A reference to the current object.