Object-Oriented Analysis and Design

Campus Batch 2011

Procedure-Oriented Programming

- Emphasis is on algorithms
- Large programs are divided into smaller programs known as functions
- Most of the functions share global data
- Data moves openly around the system from function to function
- Functions transform data from one form to another
- Employs top-down approach in program design
- Works great for smaller systems (<50,000 LOC) but fails for larger systems
- Complex data, behavioural relationships
- Increased coupling

Object-Oriented Programming

- OOP is a programming paradigm that focuses on the objects in a software system similar to that of real world.
- Objects communicate with each other by sending messages.

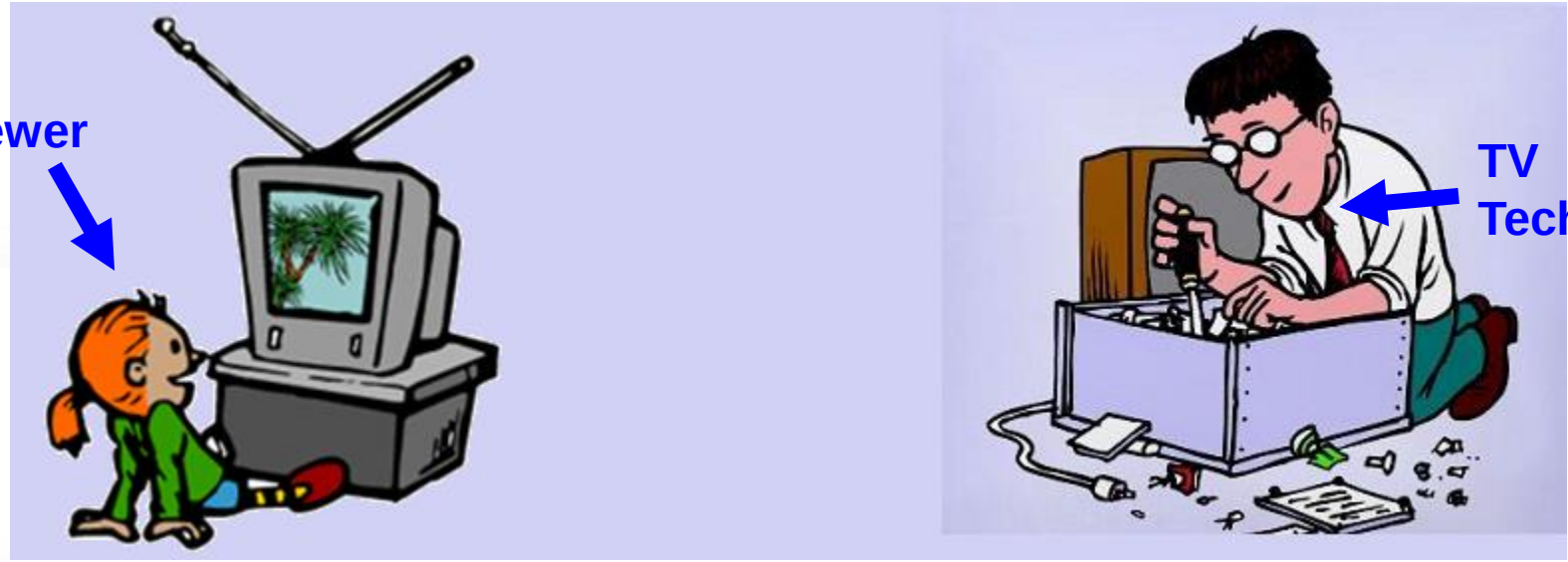
Interaction between objects

Identify Objects

Object Oriented Programming central concepts

- Abstraction
- Modularity
- Encapsulation
- Inheritance
- Polymorphism

Abstraction

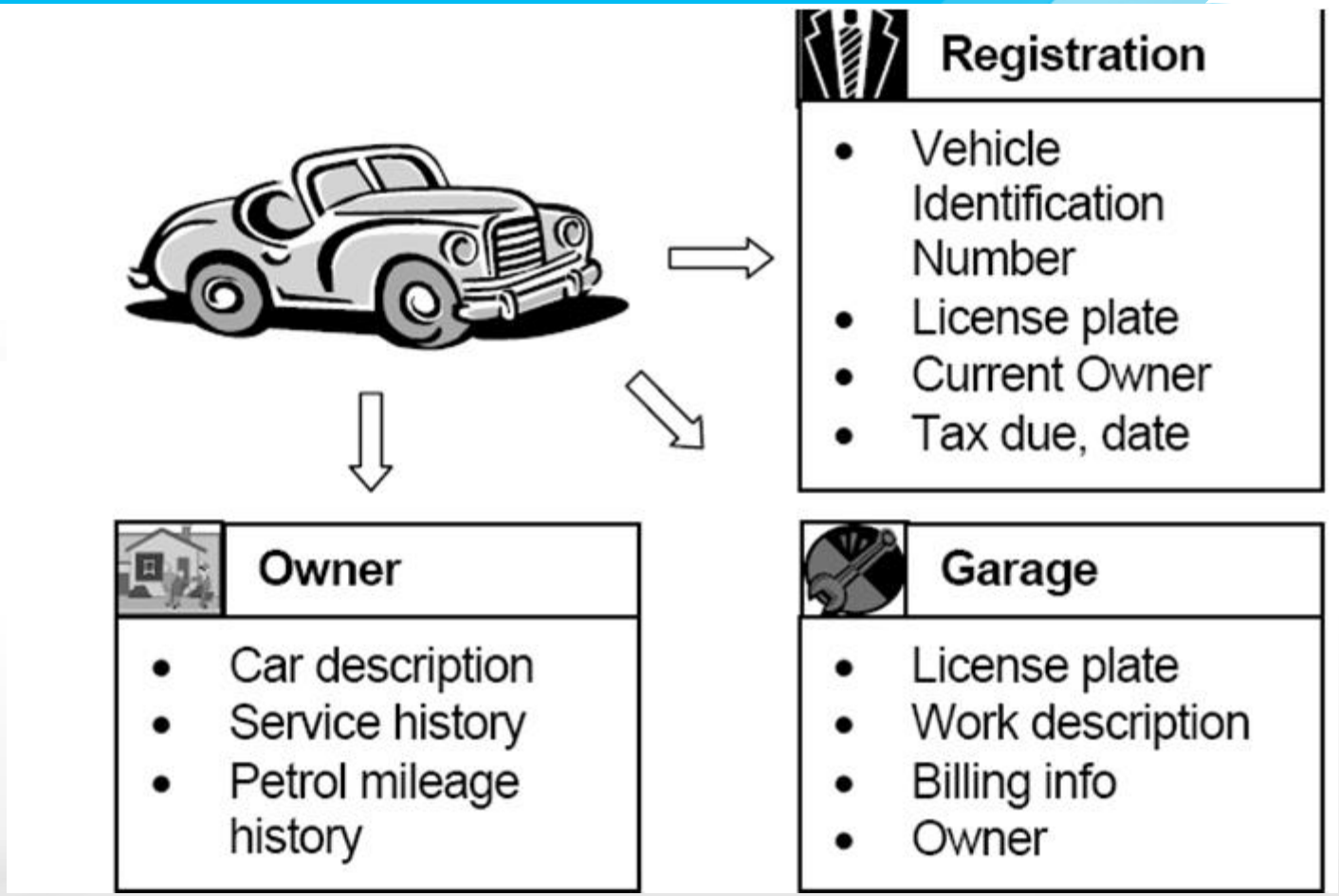


- Emphasize details that are significant to the viewer and suppress details that do not matter
- An abstraction denotes the essential characteristics of an object relative to the perspective of the viewer

Abstraction

- Abstraction is the process of capturing the essential while suppressing the detail
- An abstraction is encapsulated by a well-defined **interface**, which defines how the abstraction can be used
- A car's interface consists of steering wheel, accelerator, brake, etc.
  - We do not need to continually adjust the fuel/air mixture to drive the car - it is hidden from us.

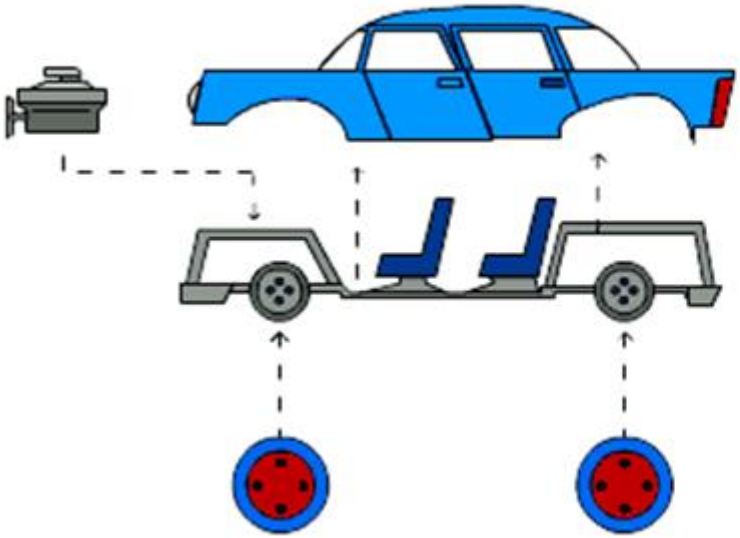
Abstraction

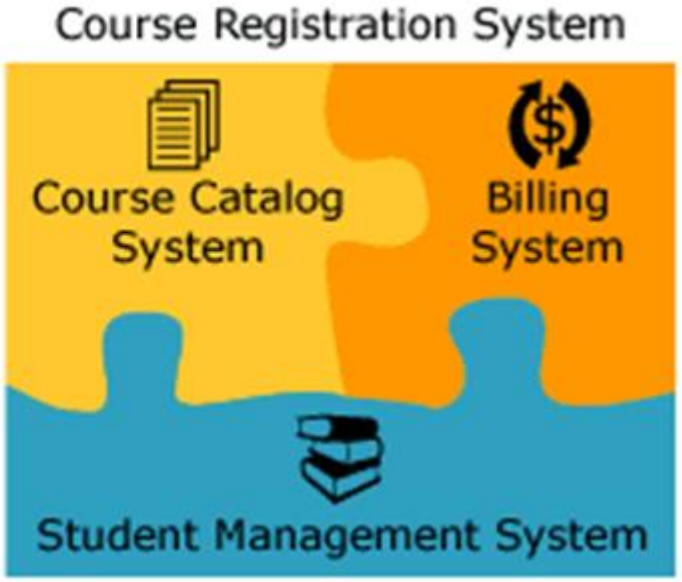


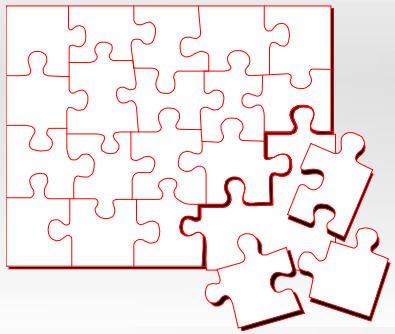
RTO office is not concerned about mileage of the car, service history . They are interested only about License plate, tax. Similarly Vehicle Mechanic is not interested about tax.

Modularity

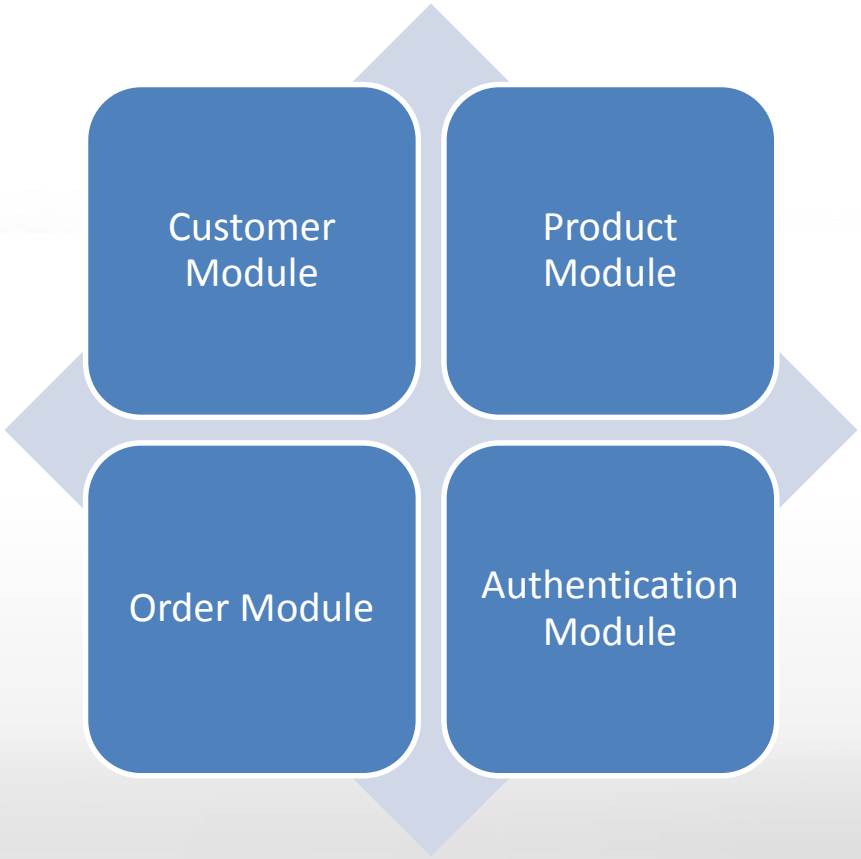
A program is partitioned into individual components to reduce the complexity







Modularity



Components in Online Shopping Application

Question

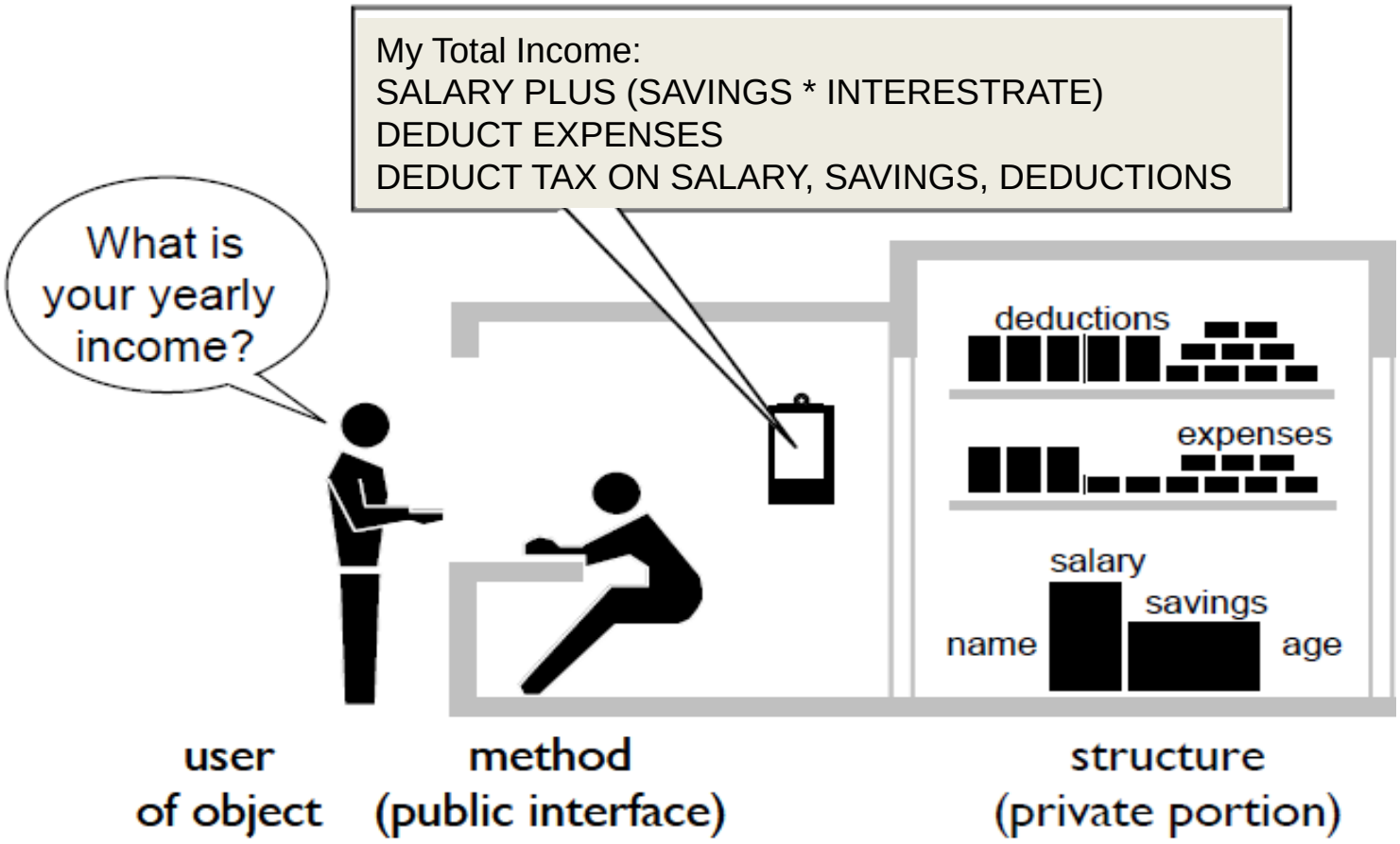
- Can you identify different modules for an Banking application.

- Can you identify different modules for Hotel management application.

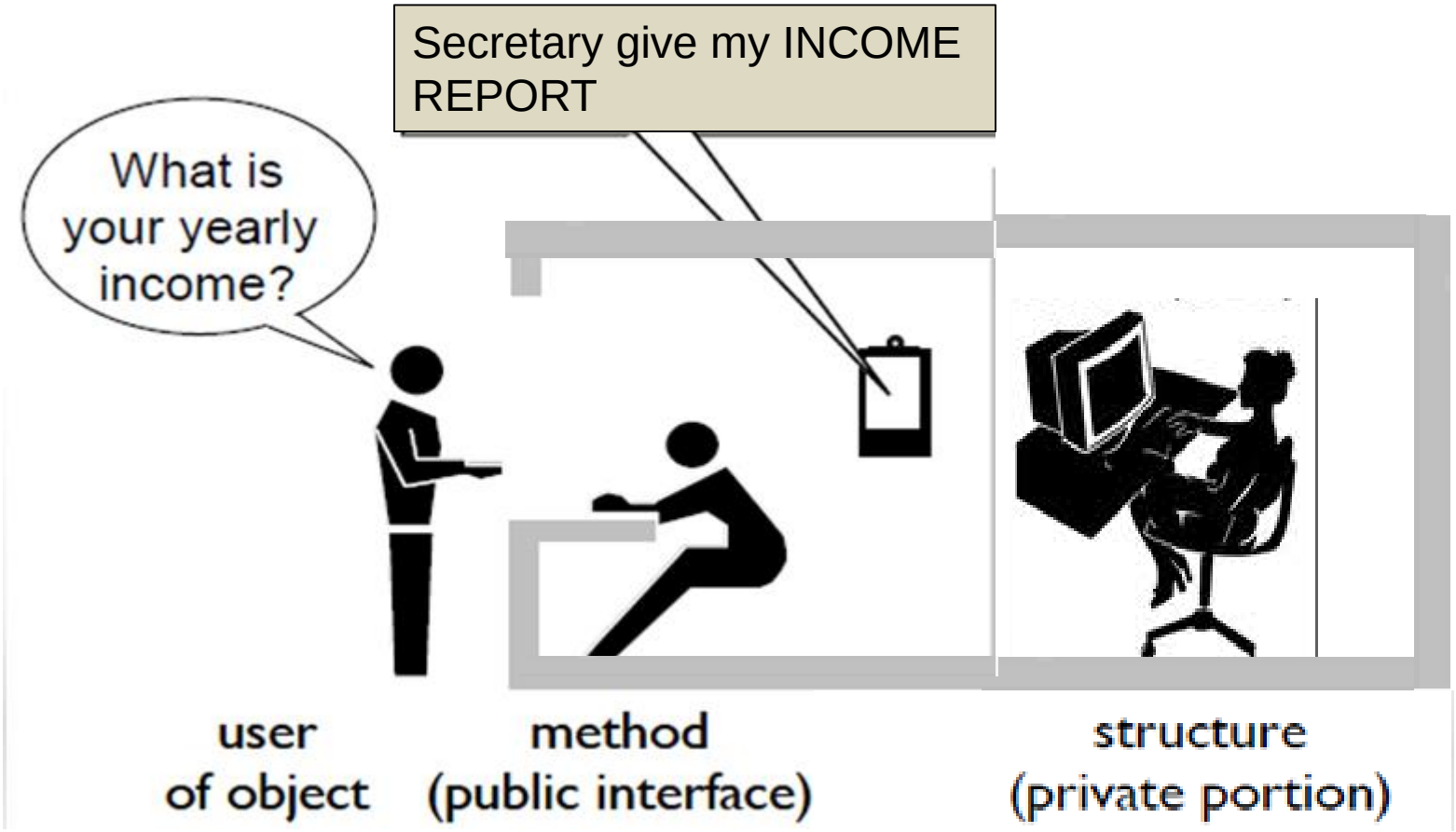
Encapsulation

- Programming languages enforce abstraction via encapsulation
- Encapsulation is an **information hiding** mechanism that hides detailed internal information about an abstraction
- The mechanical devices of your car are encapsulated (hidden) by the chassis of the car
- This insulation of the data from direct access by the program is called data hiding or information hiding
- Eliminates direct dependencies on the implementation

Encapsulation



Encapsulation

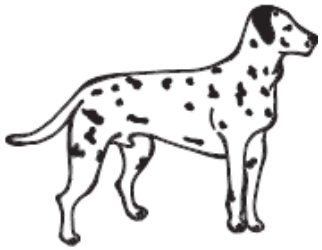


Encapsulation for builders and users

	Builder	User
Goal	Create a reusable, maintainable code with an understandable interface	Quickly get and use a code
Approach	Hide structure and implementation details from user.	Examine and use operations which are exposed through an interface.
Benefits	May change structure and algorithms without affecting user	Understandable interface; can be used without fear of "breaking" object
Costs	Reusability must be carefully planned	Access to data through operations may be slower than direct access

Objects

- What type of Objects are they?
- What are the different states of these objects?
- What is the common behavior among them?



Characteristics

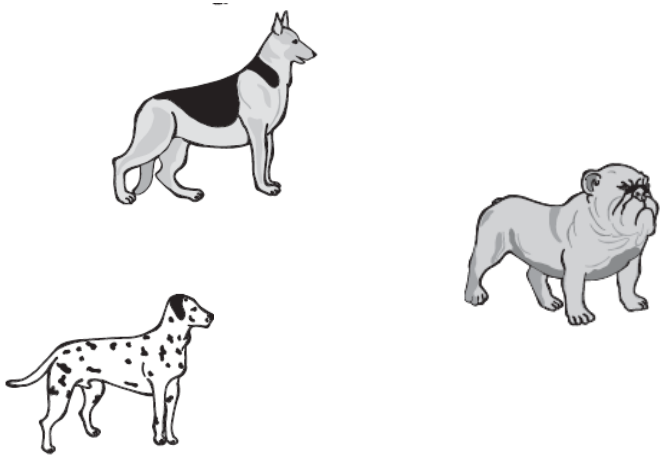
- All the objects are of type Dog
- Every dog has the following states name, breed and size.
- Every dog does the following: barks, eats, sleeps and wags its tail.
- A class defines constituent members which enable class instances to have *state* and *behavior*.

state

behavior

Dog
size breed name
bark eat sleep wagTail

Class



Objects

Objects

- What type of Objects are they?
- What are the different states of these objects?
- What is the common behavior among them?

Characteristics

- All the objects are of type car.
- Every car has a name, chassis number, registration number.
- Every car does : start , stop, turn , accelerate, etc.

Car
model chassisNo regNo
start stop turn accelerate

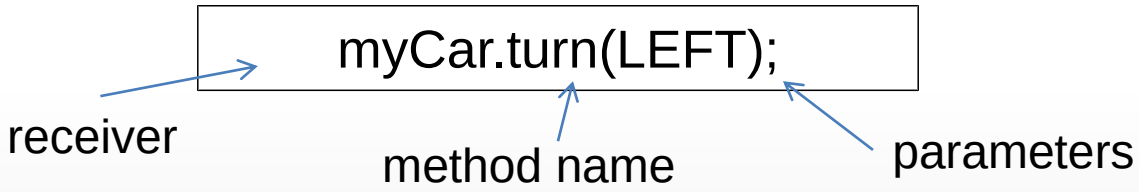
state

behavior

Class

Methods and Messages

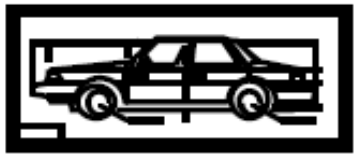
- A ***message is the request you send to an object in order to get it*** to do something: perform an action, return a value, etc.
- A message generally has three parts:
  - The receiver : the object receiving the message.
  - The method name: what we want the object to execute.
  - The parameters: any arguments that the message takes.



- A ***method is the piece of code which is called to perform the*** activity requested by the message

```
public void turn( DIRECTION side){
        // code
}
```

An OO Complete picture

	Real World	Software
	 blueprint	Car model chassisNo regNo start stop turn accelerate
Objects / instances	 car	myCar myDadsCar myFriendCar
State	 seat steering wheel tires	myCar: model: Skoda Rapid, chassisNo: AE122011, regNo: KA-50-5667 myDadsCar: model: Santro, chassisNo : LK922T3, regNo: KA-04-8222

Behavior	 dashboard	myCar.start( );
		myCar.shiftGear();
		myCar.accelerate( );
		myCar.turn(LEFT);
		myDadsCar.start( );
		myDadsCar.stop( );

Relationship between objects

- Objects can be related with one another by having
  - Association
    - Association means two classes are connected (related) in some way
  - Aggregation
    - Aggregation indicates a relationship between a whole and its parts.
  - Composition
    - Composition indicates that each part may belong to only one whole part.
  - Inheritance
    - Inheritance enables programmers to define an "is-a" relationship between classes.

Association

- Association means that two classes are connected (related) in some way.
- Information about one class is linked to data about other class.
  - e.g. Information about payroll object is associated or linked to the data about employee object.
- One class works with another to perform a task (Collaboration)
  - One class acts upon the other class for e.g. Shark eats the gold fish
- Multiplicity shows the number of objects that can participate in a relationship.
  - One class occupies one classroom.
  - One teacher teaches many classes.
  - One teacher teaches many students.
  - One student learns from many teachers

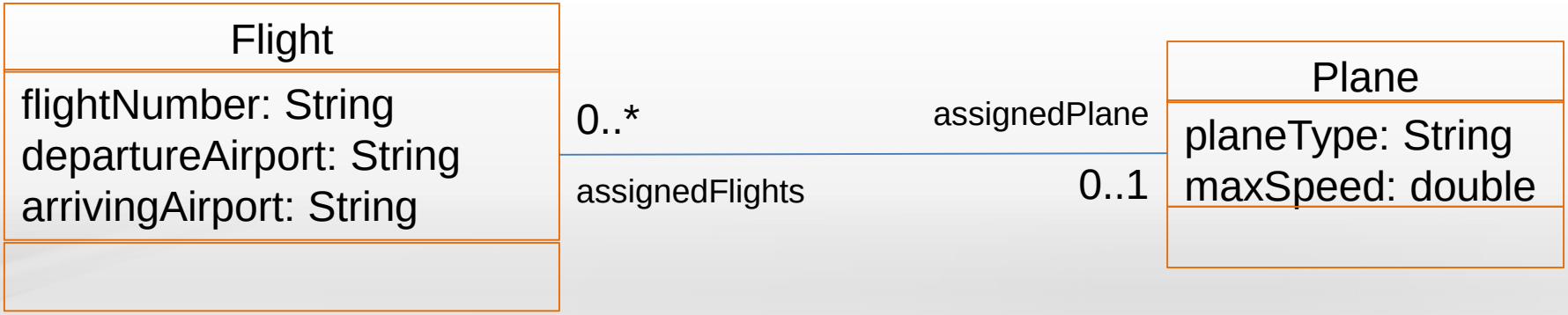
Association

- An Airline company has many planes like Boeing 747, light aircraft, etc
- The company need to schedule flights between places.
- The Company has to associate a plane to a flight.
- Possibility is not all planes will be associated and there may not be any plane available for a given flight.

- How to model the relation between flight and Plane?

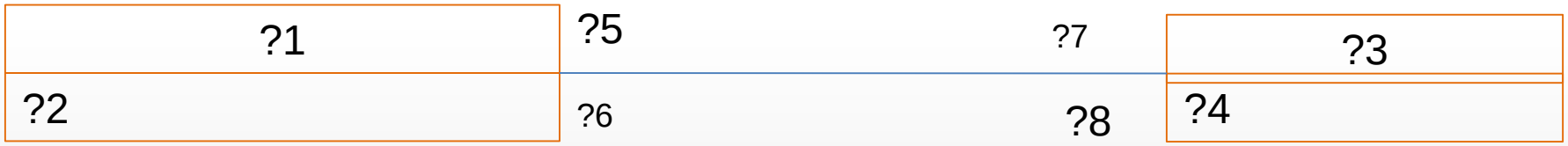
Association

- Plane takes on the role of "assignedPlane”
- Flight takes on the role of "assignedFlights“
- Flight can have one instance of a Plane associated with it or no Planes associated with it
- Plane instance can be associated either with no flights (e.g., it's a brand new plane) or with up to an infinite number of flights.



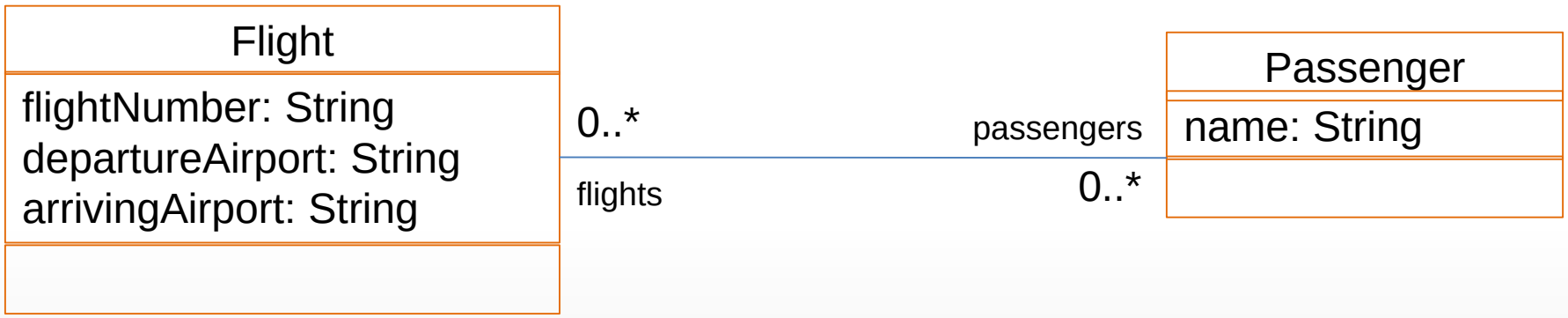
Quiz

- An Organization has many departments and in each department many employees may work.
- A employee inducted may not be assigned any department
- Also a new department might be formed and hence no employees might have been assigned



Association

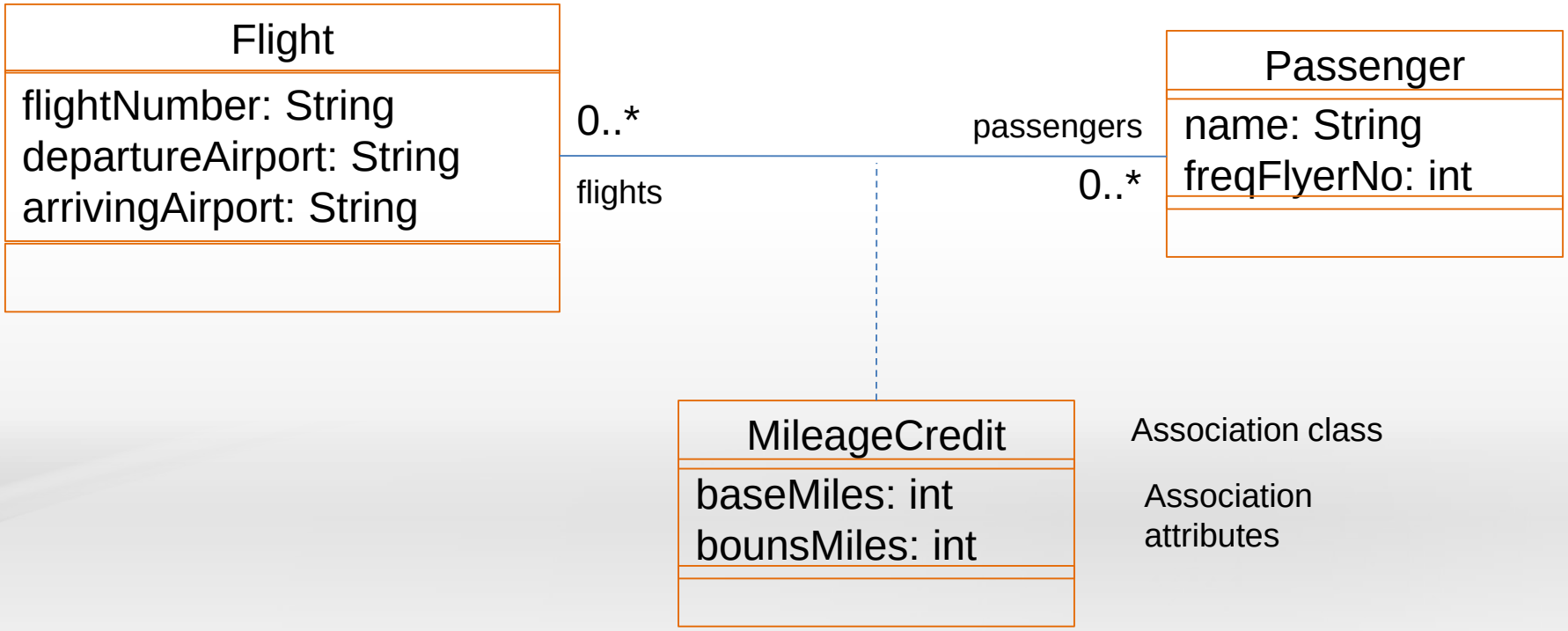
- A Flight has many passengers who fly frequently, association can be represented as shown below.



- Where to store additional attributes like bonus miles credited for the passenger for every base miles covered.

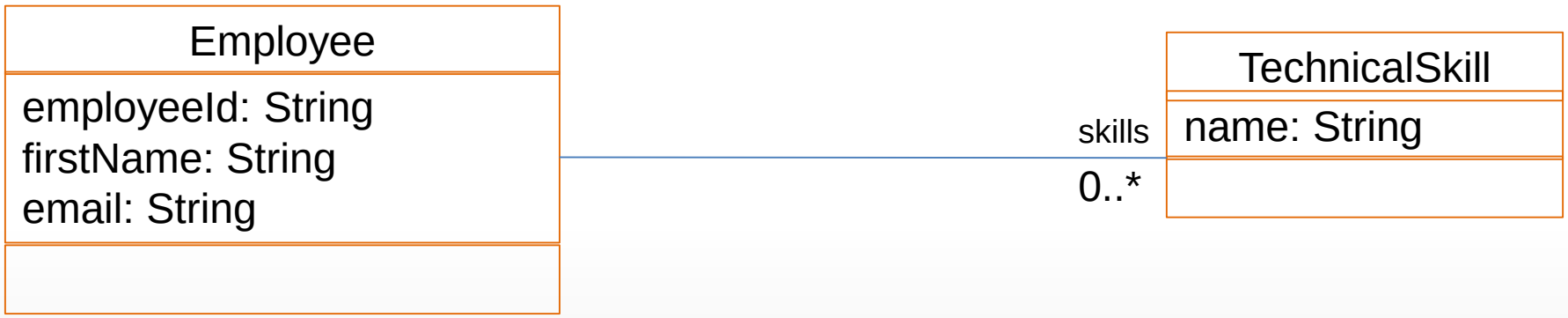
Association class

- Since bonus miles and base miles are not applicable as attributes of Flight or Passenger.
- Association classes can be used to add attributes, operations and constraints to associations.



Association class

- Employee and Skill association can be represented as shown below:

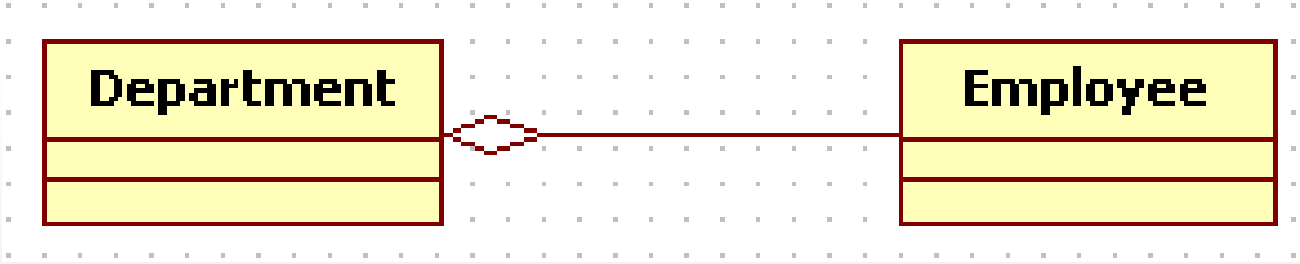


- How do you keep track of additional details like employee's skill level and experience working in that particular skill?

- **Aggregation**
  - Aggregation indicates a relationship between a whole and its parts.
  - *Aggregation* can occur when a class is a collection or container of other objects, but where the contained classes do not have a strong *life cycle dependency* on the container—essentially,
  - if the container is destroyed, its contents are not destroyed.

Aggregation

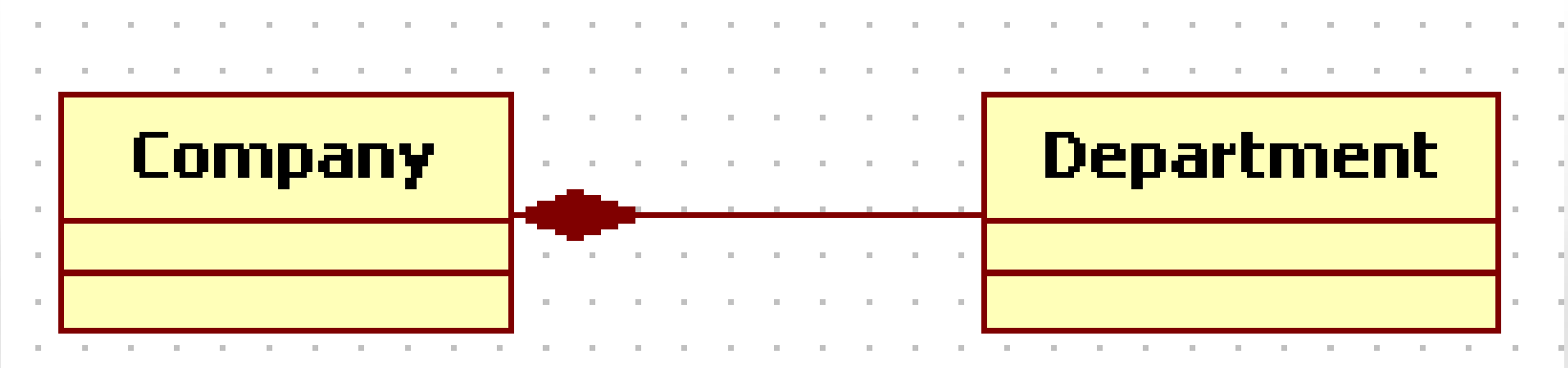
- Department have a aggregation of Employees.
  - Department does not have the lifetime responsibility for the Employee
  - An Employee can work in more than one department



- **Composition**
  - Composition is a strong form of aggregation.
  - In this kind of relationship, each part may belong to only one whole.
  - The part is not shared with any other whole.
  - In a composition relationship, when the whole is destroyed, its parts are destroyed as well.

Composition

- The composite (Company) is responsible for the creation and destruction of the component (Department) parts.
- An Department object may only be part of one composite (Company). If the composite object (Company) is destroyed, all the component parts (Departments) must be destroyed.

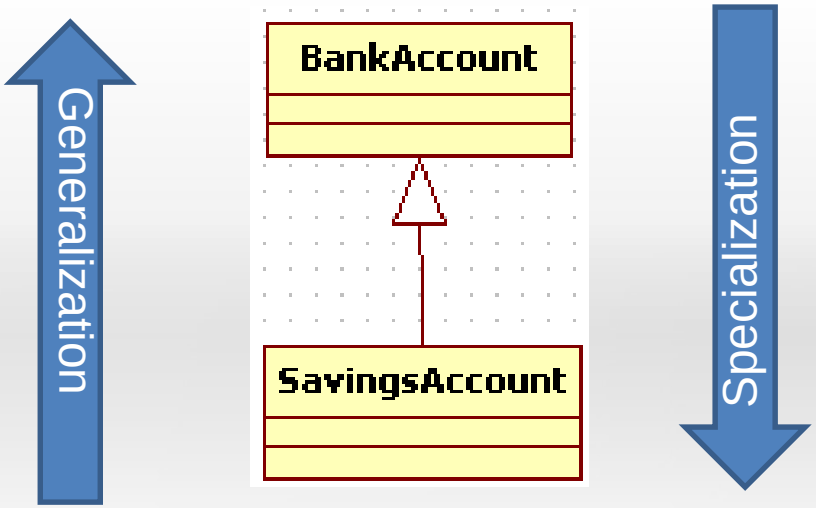


Answer this

- Can you identify the relation between them
  - BookShelf and Books
  - Components in a Television Set
  - Orchestra is made up of Musicians
  - Building is made up of rooms

Inheritance (is-A Relationship)

- **Inheritance** is a way to group and reuse code by creating objects which can be based on previously created objects.
- Inheritance enables programmers to define an "is-a" relationship between a class and a more specialized version of that class.
- E.g. If a SavingsAccount extends BankAccount. SavingsAccount is-A BankAccount.



Generalization

Different Products

Mobile
productNumber description price cameraDetails memoryCapacity

Television
productNumber description price screenSize screenType

Refrigerator
productNumber description price capacity frostFree

Inheritance Hierarchy

Generalization

Specialization

Product
productNumber: int description: String price: double

Generalization Relationship

Mobile
cameraDescription: String memoryCapacity: String

Television
screenSizeInInches: int screen Type: String

productNumber : 544
description: G Android Mobile
price: 25000.55
cameraDescription: 3 MP
memoryCapacity “: 16 GB

productNumber : 122
description: ABC TV
price: 65000.55
screenSizeInInches: 42
screenType : LCD

Answer this

- How do you generalize an Rectangle and Circle.
- How do you generalize an Bitmap, GIF and JPEG.

When Inheritance is required.

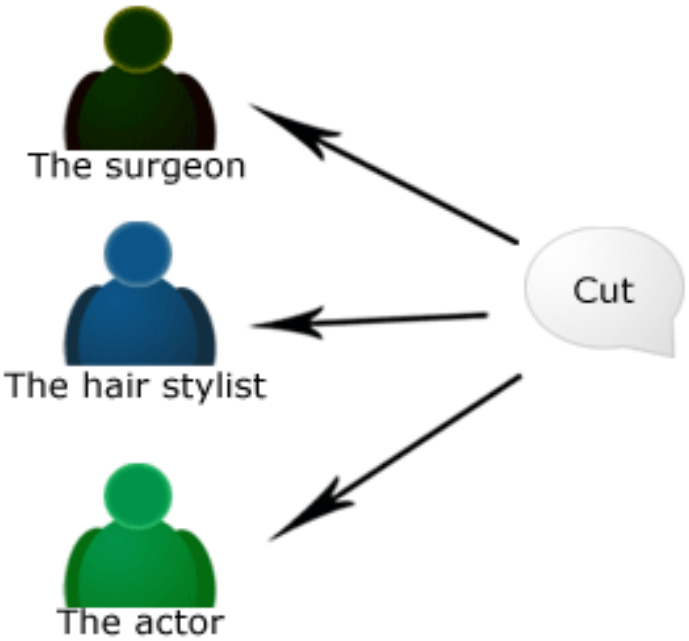
- Use inheritance only when all of the following criteria are satisfied:
  - A subclass expresses "is a special kind of" and not "is a role played by a“.
  - An instance of a subclass never needs to become an object of another class (transmute)
  - A subclass extends, rather nullifies, the responsibilities of its super class.
  - A subclass does not extend the capabilities of what is merely a utility class
  - For a class in the actual Problem Domain, the subclass specializes a role, transaction or device

Polymorphism

Polymorphism is the ability of different objects to respond in their own unique way to the same message

Polymorphism

How do they react if someone says “CUT” to these people?

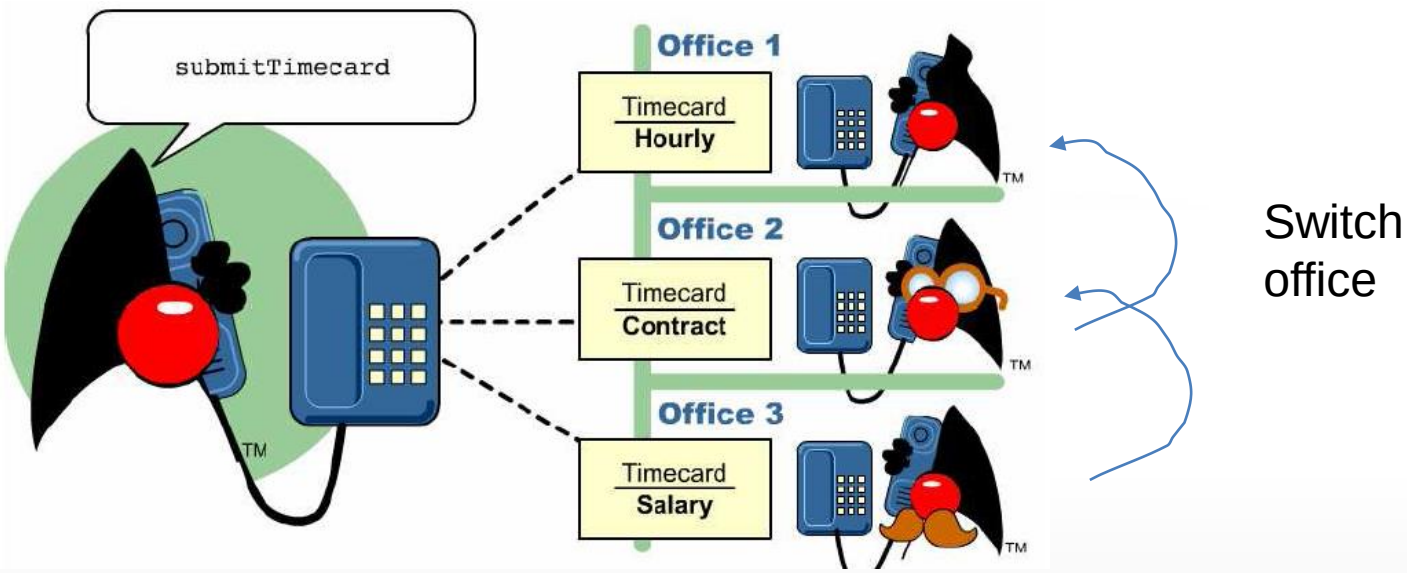


The Surgeon would begin to make an incision (slit).

The hair stylist would begin to cut someone’s hair.

The actor would abruptly stop acting the current scene, awaiting directorial guidance.

Polymorphism

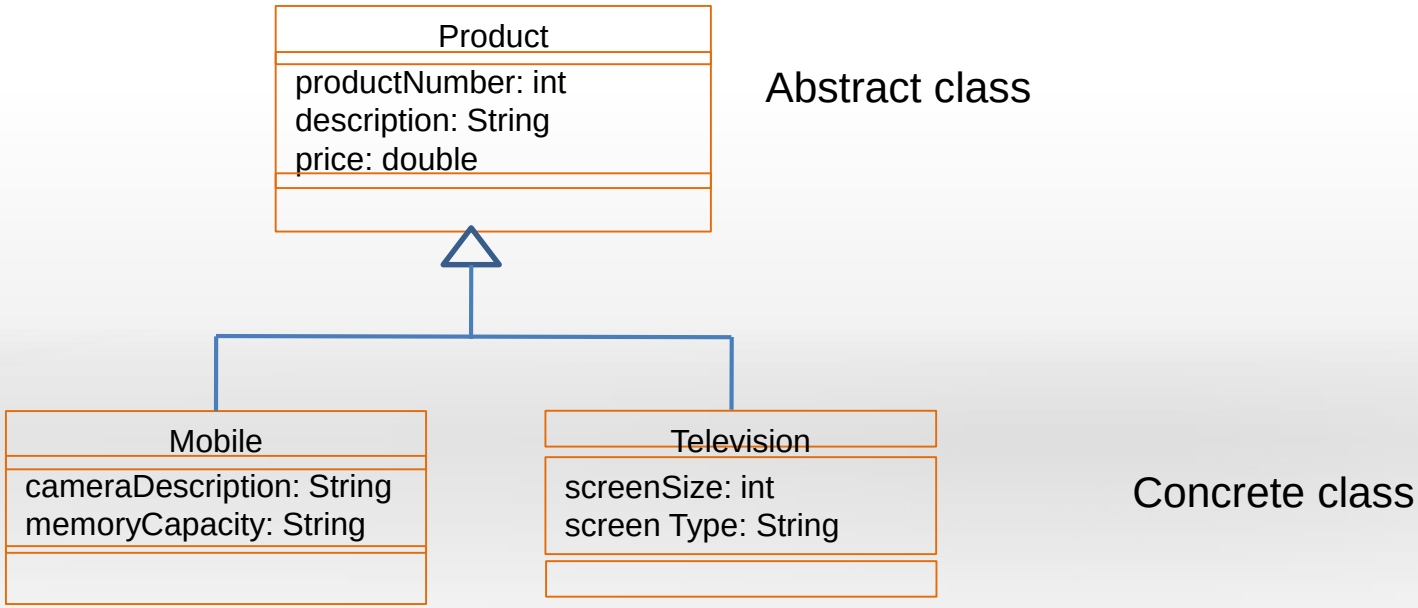


Employee in office 1 is on salary and in office 2 is working hourly basis…
The employees in each office follow their own submitTimecard() method.

When employees switch office, the manager can still place the same call, knowing that the employees will follow their own submitTimeCard() implementation.

Abstract Class

- An ***Abstract Class*** is a class that provides common behavior across a set of subclasses, but is not itself designed to have instances of its own.
- An abstract class is designed as a template for other classes to follow by dictating behavior that must be implemented by its subclasses



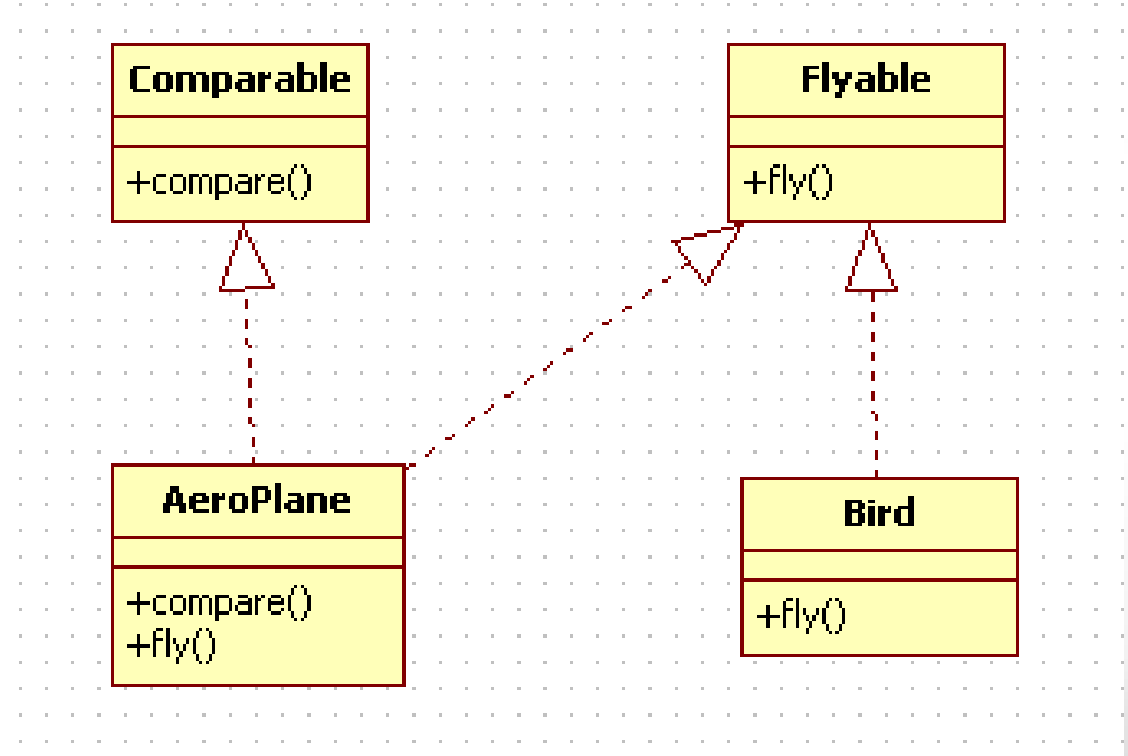
Interface

- In the field of computer science, an **interface** is a tool and concept that refers to a point of interaction between components, and is applicable at the level of both hardware and software.

- Example:
  - **Universal Serial Bus** (**USB**) is an interface that can be used to connect computer peripherals such as mouse, keyboards, digital cameras, printers, USB Pen Drives, etc
  - VGA input port interface of LCD Projector can be used to connect to VGA output port of Computer, DVD player, Handy Camera, etc.

Realization

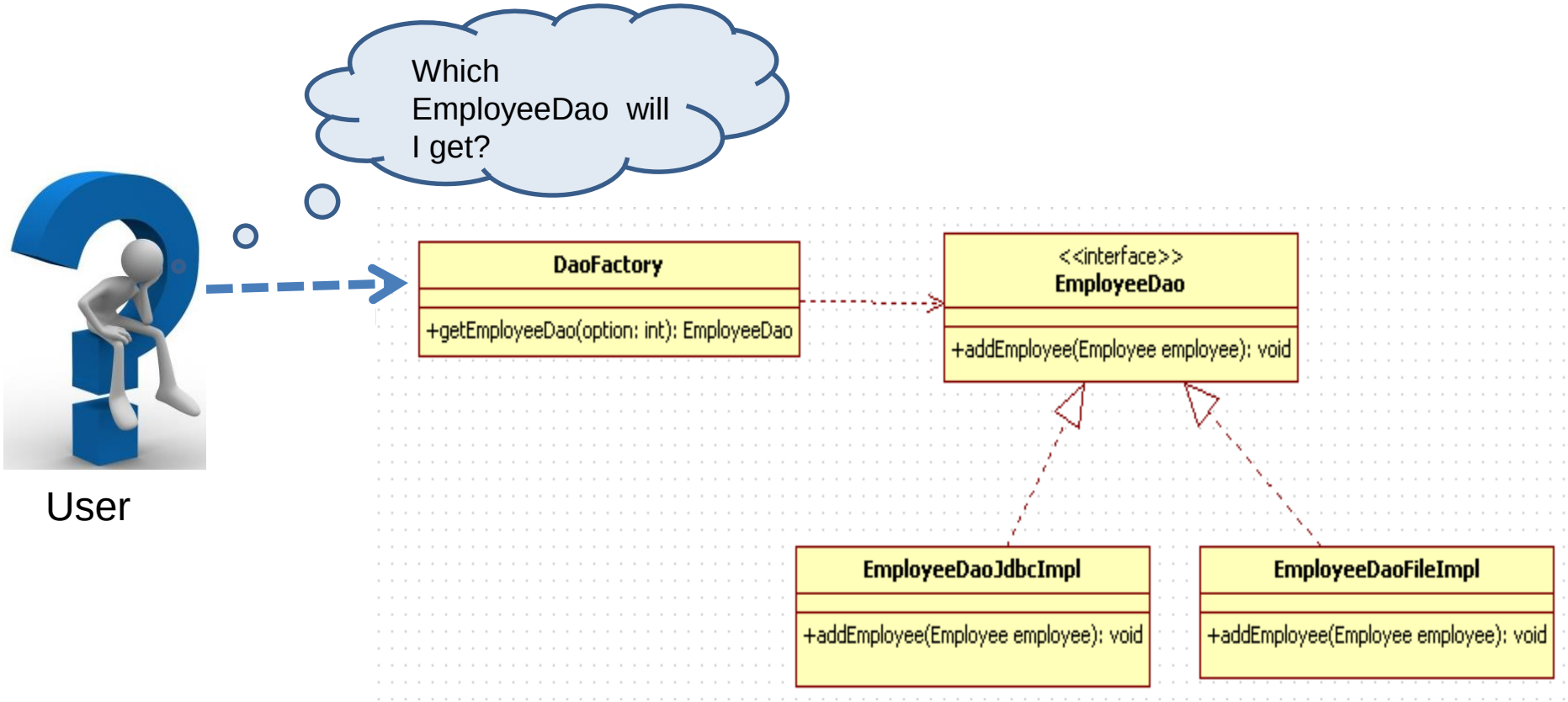
- Methods of interface are realized by implementing class



A class can implement multiple interfaces.
AeroPlane class realizes two interfaces Comparable and Flyable.

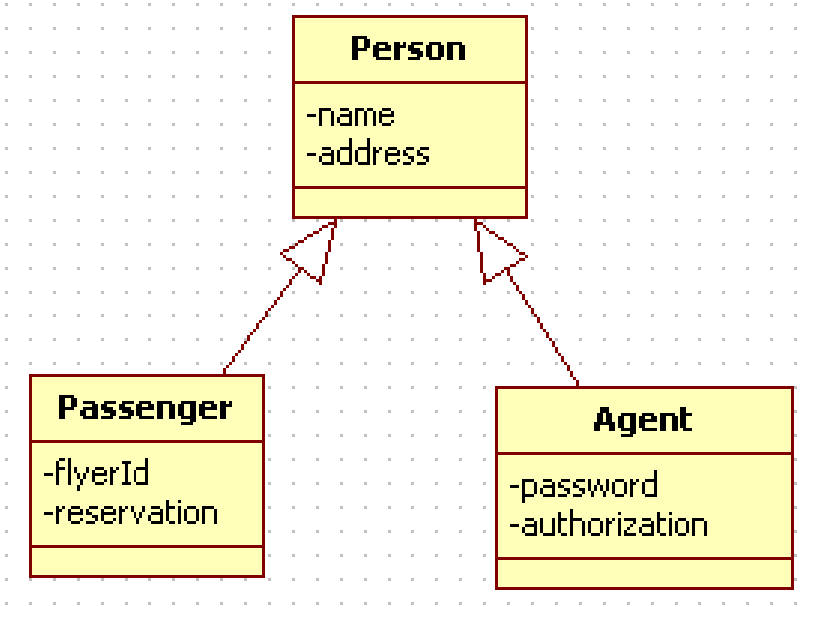
Why use Interfaces?

- To separate (decouple) the specification available to the user from implementation.



Inheritance / Composition Example

- Approach 1



1."Is a special kind of" not "is a role played by a"

Fail. A passenger is a role a person plays. So is an agent.

2.Never needs to transmute

Fail. A instance of a subclass of Person could change from Passenger to Agent to Agent Passenger over time

3.Extends rather than overrides or nullifies

Pass.

4.Does not extend a utility class

Pass.

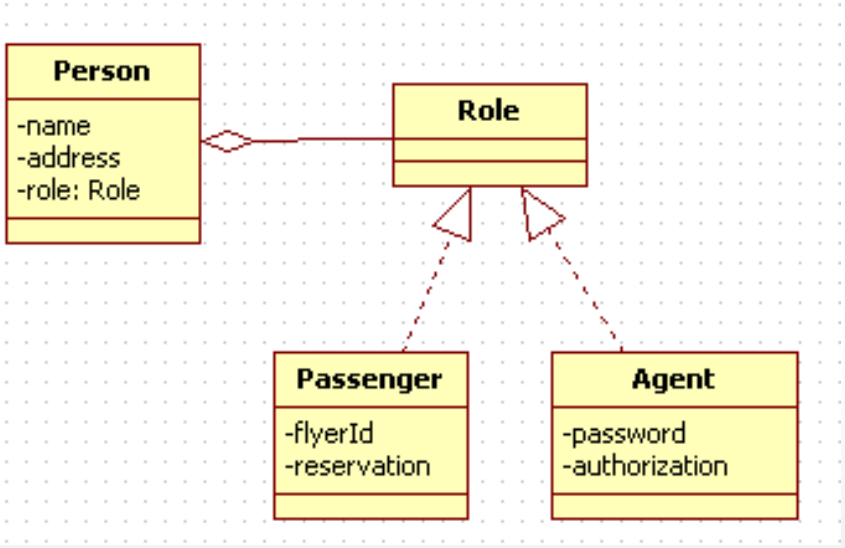
5.Within the Problem Domain, specializes a role, transaction or device

Fail. A Person is not a role, transaction or device.

Inheritance does not fit here!

Inheritance / Composition Example

- Approach 2



1."Is a special kind of" not "is a role played by a".

Pass. Passenger and Agent are special kinds of Roles.

2.Never needs to transmute.

Pass. Passenger object stays a Passenger; same is true for Agent.

3.Extends rather than overrides or nullifies.

Pass.

4.Does not extend a utility class.

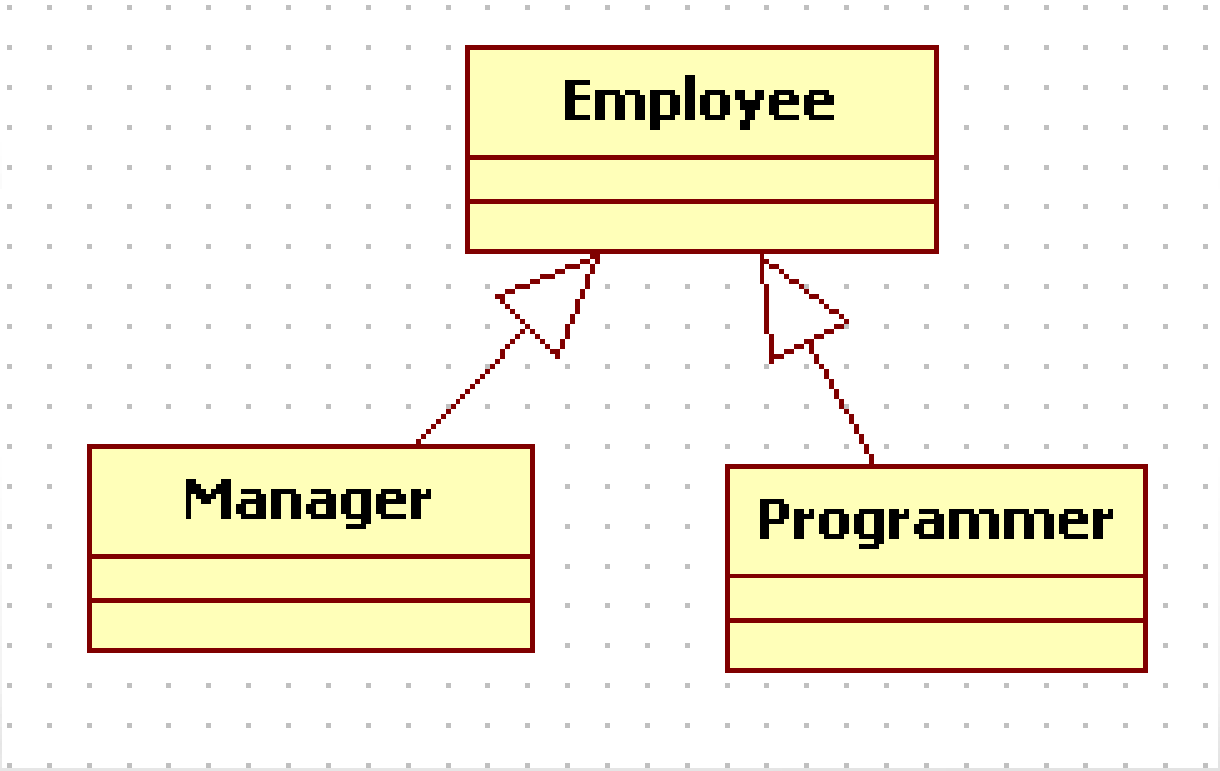
Pass.

5.Within the Problem Domain, specializes a role, transaction or device.

Pass. A Role is a type of Role..

Inheritance is OK here!

Quiz: Is this class hierarchy OK. explain



Answer this

- Real-world objects contain _____ and _____.
- A software object's state is stored in _____.
- A software object's behavior is exposed through _____.
- Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as _____.
- A blueprint for a software object is called a _____.
- A collection of methods with no implementation is called an _____.

Case Study – Library Management System

- A university library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals. The system must keep track of when books and journals and returned, enforcing the rules described above.

- Nouns and noun phrases: University, library, book, journal, copy, short term loan, library member, week, member of library, item, time, member of staff, system, rule.

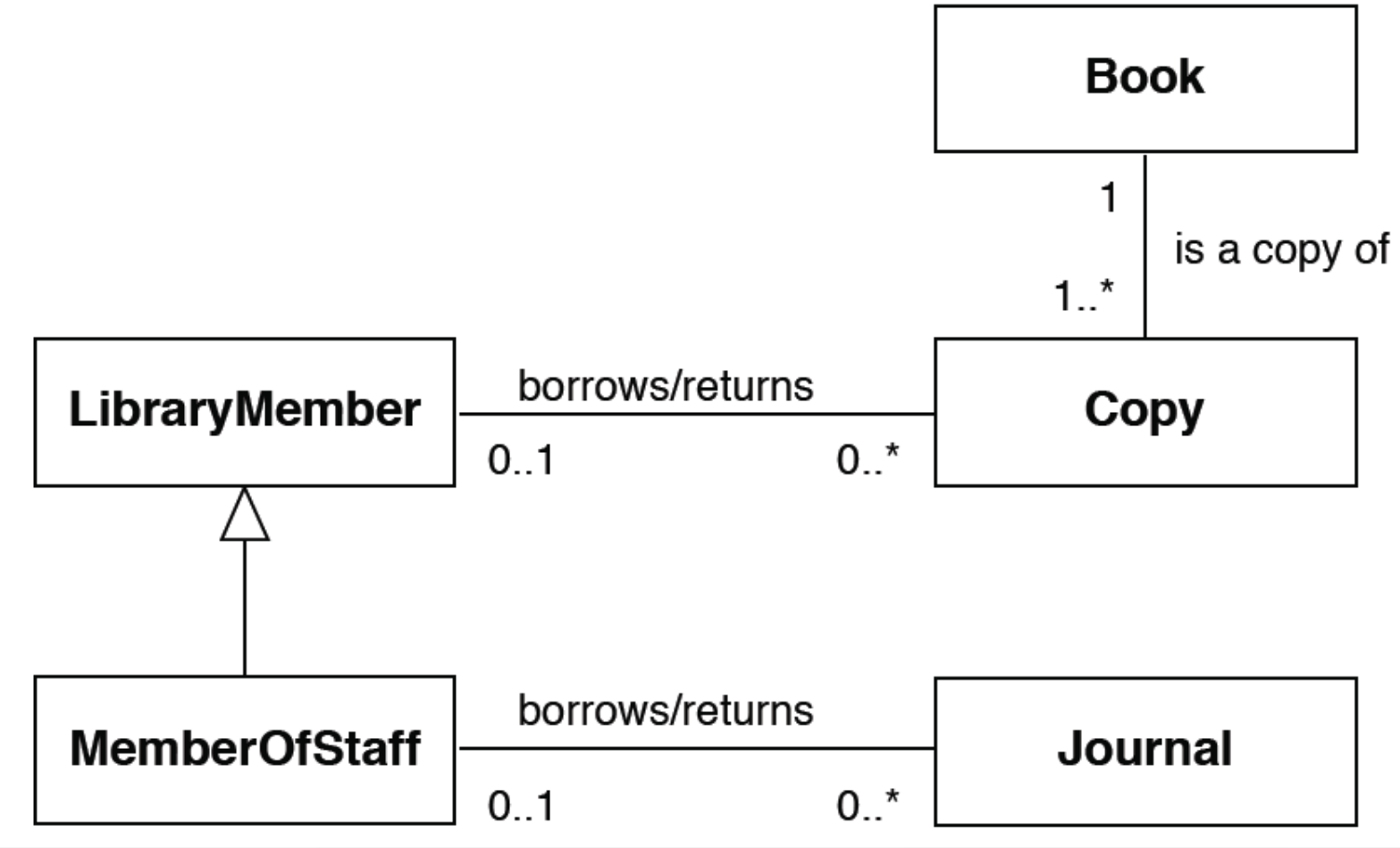
Case Study : Bad Classes

- **Bad Classes**
  - **university**: irrelevant (outside the scope of the system)
  - **library**: irrelevant (outside the scope of the system, but sometimes you need a "main object")
  - **short term loan** operation (but this may be questioned, since the loans should be recorded)
  - **week** attribute (a measure of something [time])
  - **member of library** redundant (same as library member)
  - **item** vague (means book or journal)
  - **time** attribute (or irrelevant)
  - **system** irrelevant (or redundant, same as library)
  - **rule** irrelevant (but sometimes it is useful to have "business rules objects", if the rules are subject to frequent changes)

Case Study : Good Classes

- Good Classes
  - **Book** Records information about the book (ISBN, author, number of copies, ...)
  - **Journal** Same for journal (but there is only one copy of each journal)
  - **Copy** (of book) Records information about each copy of a book (e.g., a bar code number which is read when the copy is borrowed)
  - **LibraryMember** Anybody using the library (actually a computer representation of the user, necessary since his name/address/.../number of items borrowed is needed in the system). The member object will carry out actions on behalf of the "real" user member of staff A user with special privileges

Case Study : Class Diagram



Explore More!!

Never let your curiosity die!

Advanced Topic

- UML
  - http://edn.embarcadero.com/article/31863
  - http://www.visualcase.com/tutorials/class-diagram.htm
  - http://www.cragsystems.co.uk/uml_tutorial/
  - http://seqcc.icarnegie.com/content/SSD/SSD3/4.3.0.0/normal/pg-class-dsgn/pg-dsgnng-class/pg-rltnshps-class/pg-rltnshps-class-cn.html
  - http://sourcemaking.com/uml
- Design Patterns
  - http://www.oodesign.com/factory-method-pattern.html