Object Oriented Analysis and Design

Chapter 1:

Object Oriented Software Development Concepts

Chapter 1

Object Oriented Software Development Concepts

OBJECTIVES:

At the end of this chapter, you should be able to:

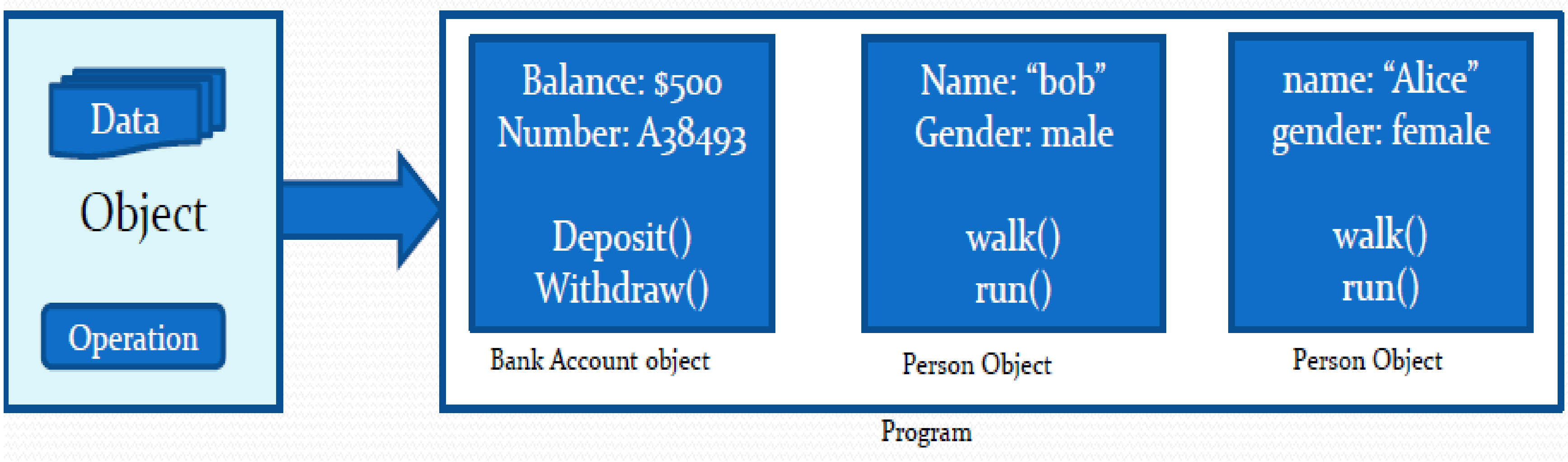
1. Differentiate the concept of procedural paradigm and object oriented paradigm
2. Analyze any scenario in OOAD perspective
3. Evaluate OOAD concepts
4. Analyze class and object relationship

What is an object?

- Every thing in our life is considered as an object
- Each object has its own existence and its own identity
- Object Orientation is intended to make thinking about programming close to thinking about real-world
- Object in computer program is a self-sufficient entity (It hosts data and operations)

What is an object?

- Think about computer program as collections of objects instead of a collection of data and operation.



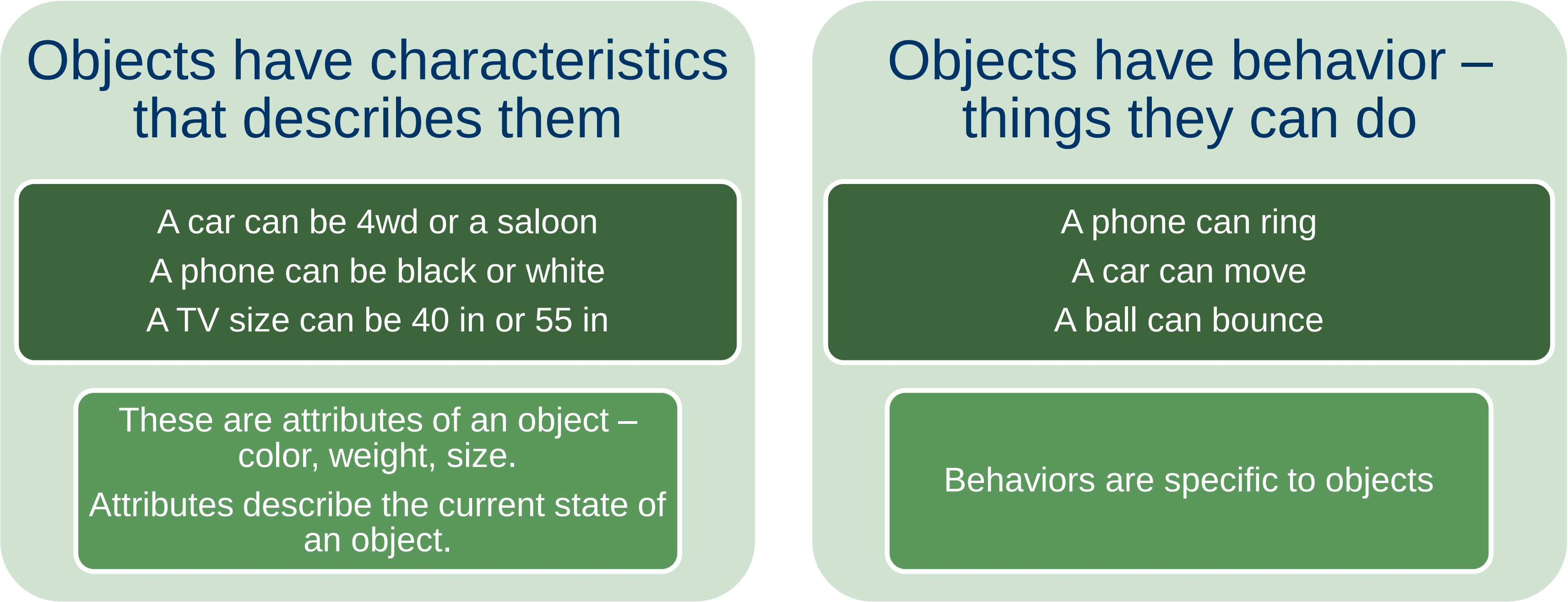
What is an object?

- Data or variables are called as attributes
- Operations are called methods (functions)

How do I know what is an object?

- Look for words which are nouns
- Try to put the word The before the word. (The date, The bank, etc.)
- Verbs would be behavior / methods of the object

What is an object?



Objects have characteristics that describes them

A car can be 4wd or a saloon

A phone can be black or white

A TV size can be 40 in or 55 in

These are attributes of an object – color, weight, size.

Attributes describe the current state of an object.

Objects have behavior – things they can do

A phone can ring

A car can move

A ball can bounce

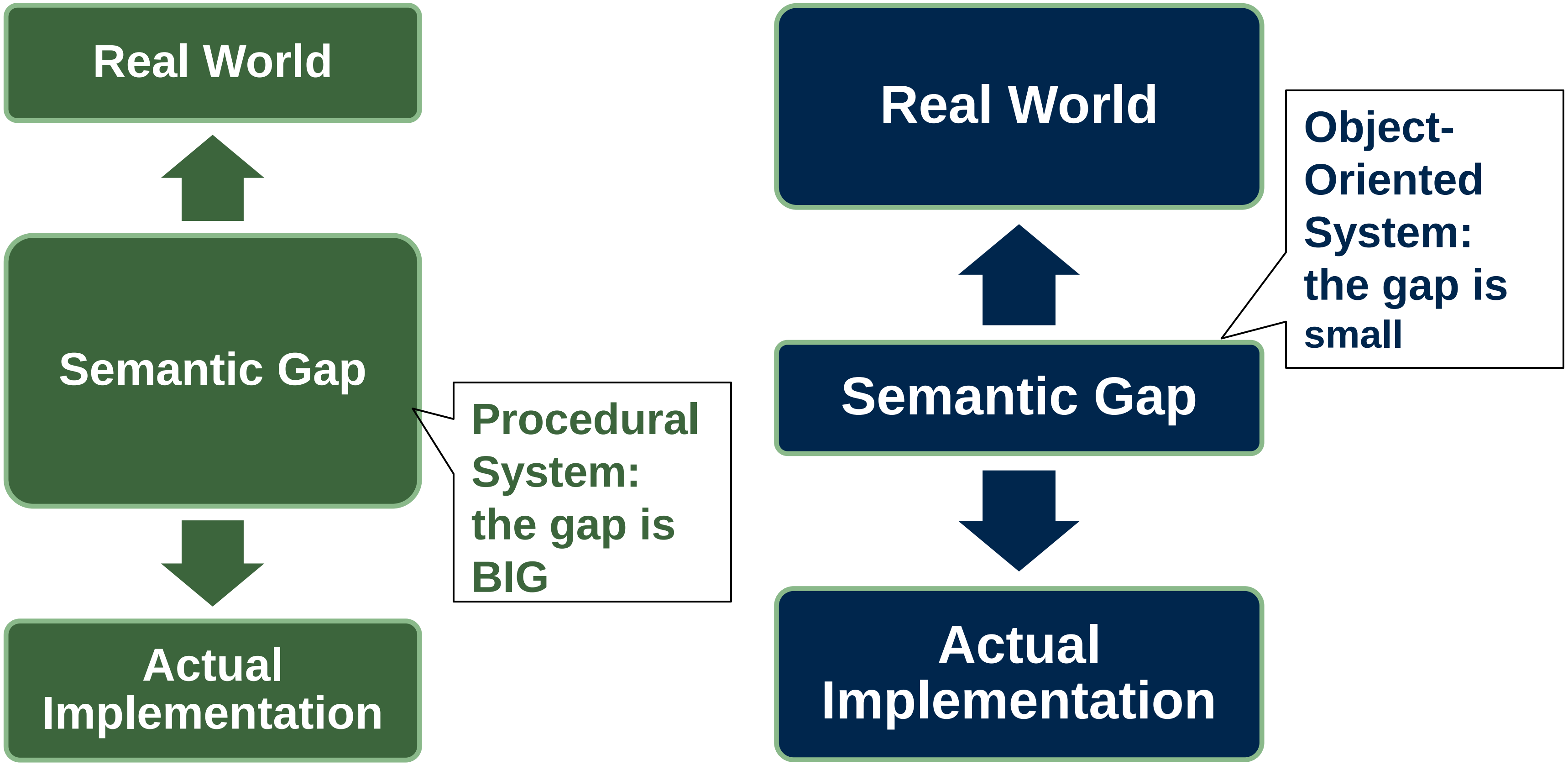
Behaviors are specific to objects

Object interacts with each other by exchanging messages and invoke methods (call getSalary() method to compute and print salary)

Examining Object Orientation

- Think about objects in the same way as a real world object.
- Object orientation is a different way to think about a system as compared to the traditional systems.
- The gap between what we perceive and what we model is called semantic gap.
- We need to make the semantic gap smaller to make:
    1. It easy to understand system requirement
    2. Alteration of software will be easy

Semantic Gap



Real World

Semantic Gap

Actual Implementation

Procedural System: the gap is BIG

Real World

Semantic Gap

Actual Implementation

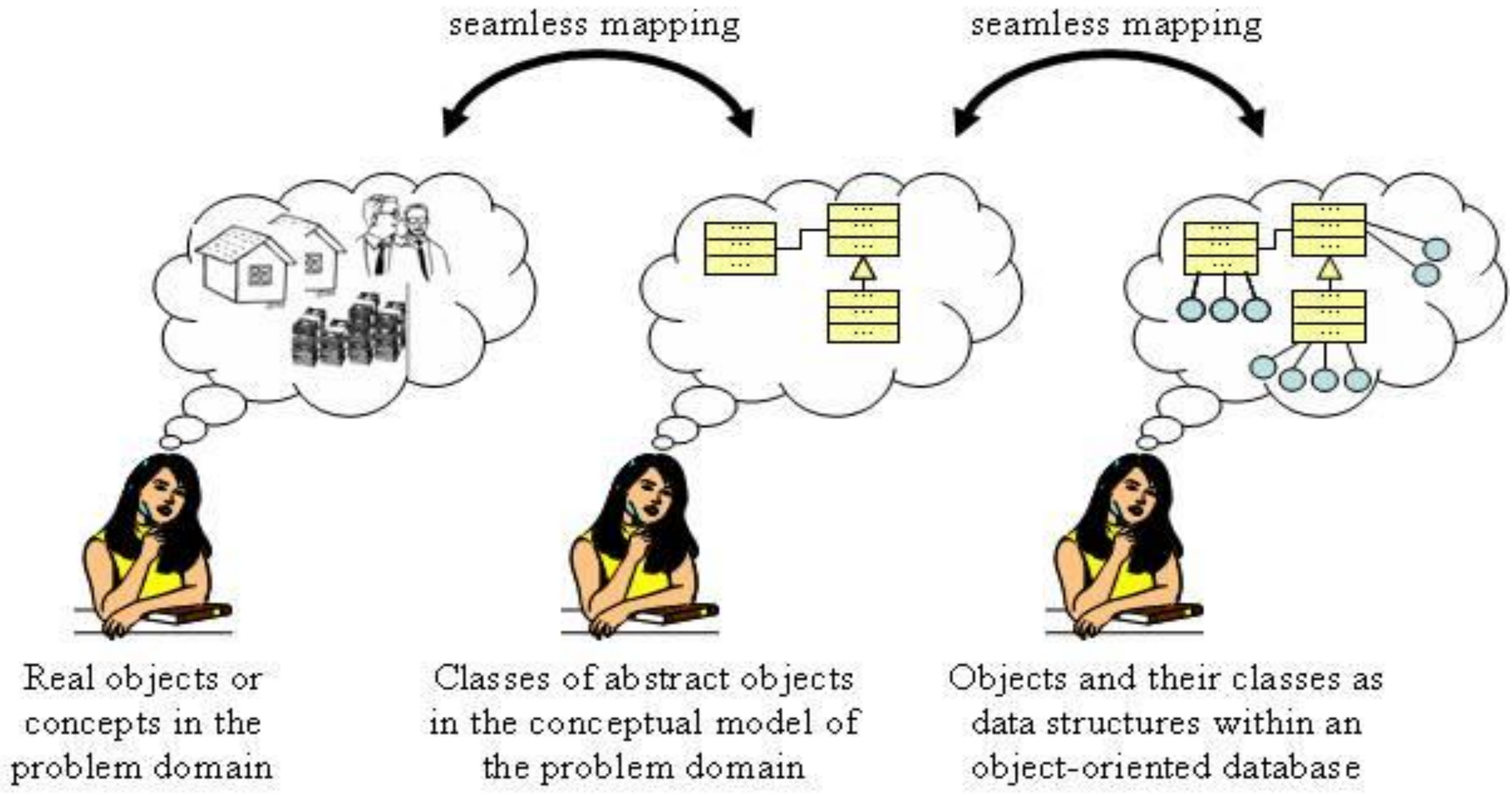
Object-Oriented System: the gap is small

Examining Object Orientation

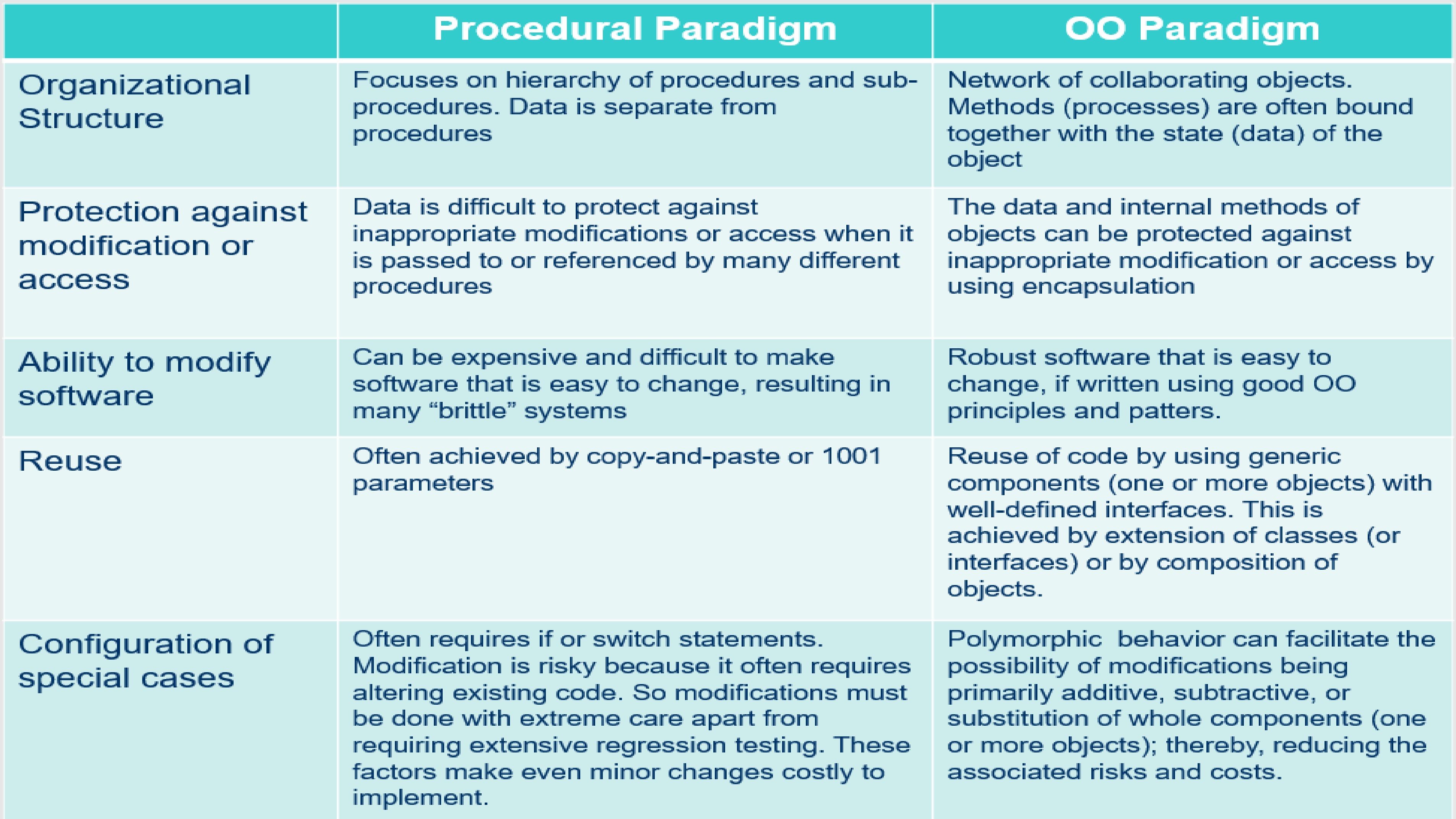
OO concepts affect the whole development process:

- Humans think in terms of **nouns (objects)** and **verbs (behaviors of objects)**
- With OOSD, both problem and solution domains are modeled using OO concepts.
- The Unified Modeling Language (UML) is a de facto standard for modeling OO software
- OO Languages bring the implementation closer to the language of mental models. The UML is a **good bridge between mental models and implementations.**

Examining Object Orientation


seamless mapping
seamless mapping
Real objects or concepts in the problem domain
Classes of abstract objects in the conceptual model of the problem domain
Objects and their classes as data structures within an object-oriented database

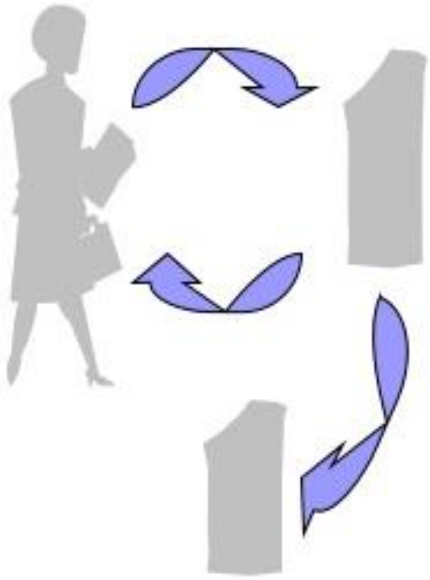
Comparing the Procedural and OO Paradigms



	Procedural Paradigm	OO Paradigm
Organizational Structure	Focuses on hierarchy of procedures and sub-procedures. Data is separate from procedures	Network of collaborating objects. Methods (processes) are often bound together with the state (data) of the object
Protection against modification or access	Data is difficult to protect against inappropriate modifications or access when it is passed to or referenced by many different procedures	The data and internal methods of objects can be protected against inappropriate modification or access by using encapsulation
Ability to modify software	Can be expensive and difficult to make software that is easy to change, resulting in many “brittle” systems	Robust software that is easy to change, if written using good OO principles and patters.
Reuse	Often achieved by copy-and-paste or 1001 parameters	Reuse of code by using generic components (one or more objects) with well-defined interfaces. This is achieved by extension of classes (or interfaces) or by composition of objects.
Configuration of special cases	Often requires if or switch statements. Modification is risky because it often requires altering existing code. So modifications must be done with extreme care apart from requiring extensive regression testing. These factors make even minor changes costly to implement.	Polymorphic behavior can facilitate the possibility of modifications being primarily additive, subtractive, or substitution of whole components (one or more objects); thereby, reducing the associated risks and costs.

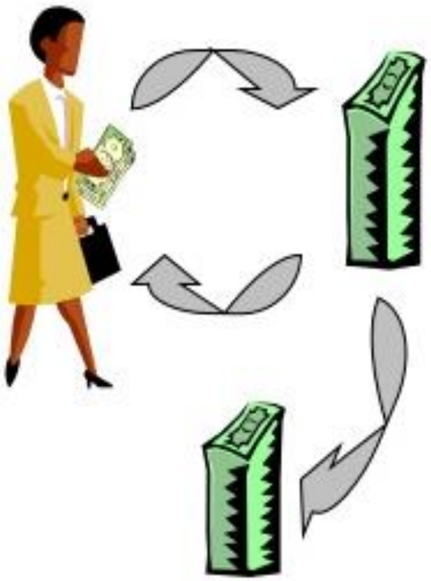
Procedural vs. Object Oriented

- Procedural



Withdraw, deposit, transfer

- Object Oriented



Customer, money, account

Why use Object Oriented?

- Most modern programming languages today are object oriented. (e.g. Java, C++, C#, etc.)
- Used in developing games, mobile apps, and other information system (IS).
- Allows your program to be:
  1. Modular
  2. Easy to maintain
  3. Easy to updated
  4. Easy to understand

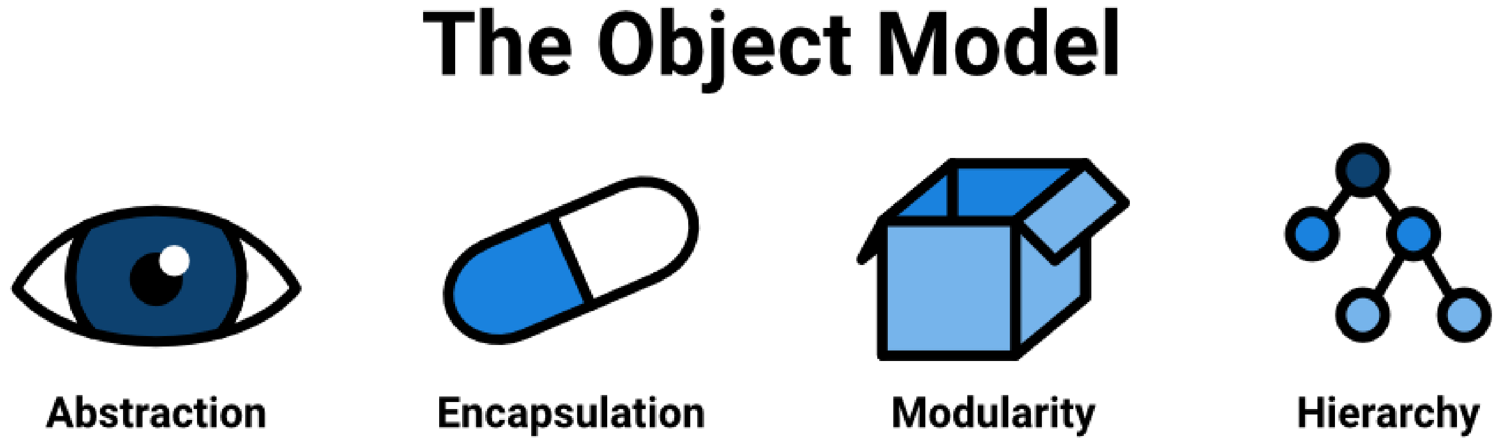
Click Link What is OOP? (You tube Link)

Object Model

For all things object-oriented, the conceptual frameworks is the object model

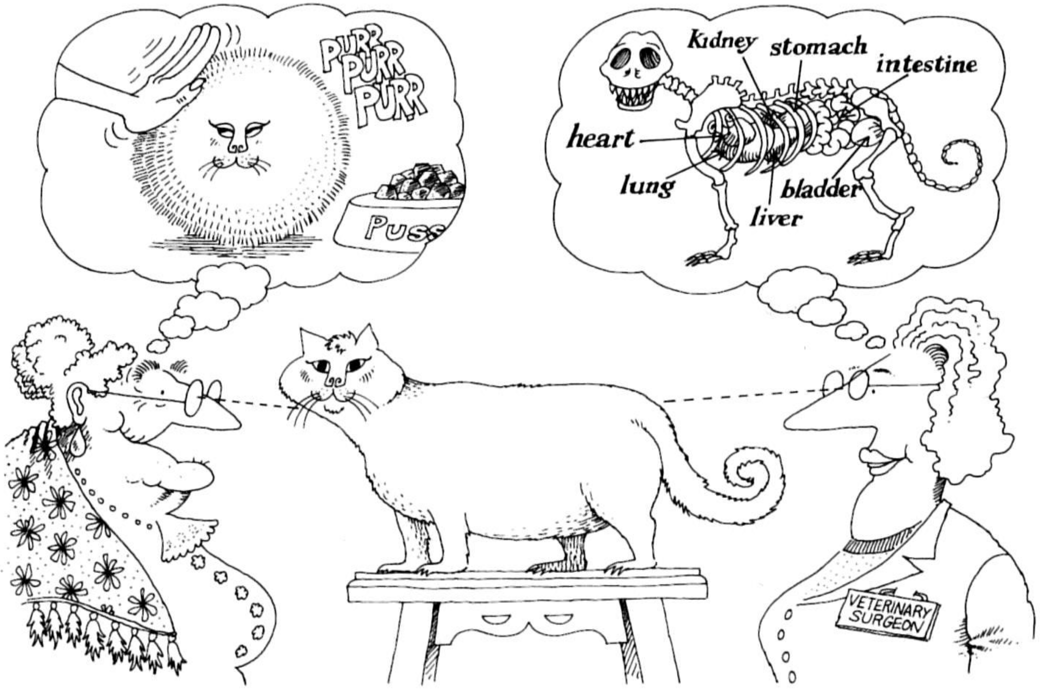
- **Major Elements of the Object Model**
    - ➢ Abstraction
    - ➢ Encapsulation
    - ➢ Modularity
    - ➢ Hierarchy
- **Minor Elements of the Object Model**
    - – Typing (language dependent – data typing)
    - – Concurrency (OS dependent)
    - – Persistence

Major Elements of



1. Abstraction

- Helps to deal with complexity by focusing on certain features and suppressing others.

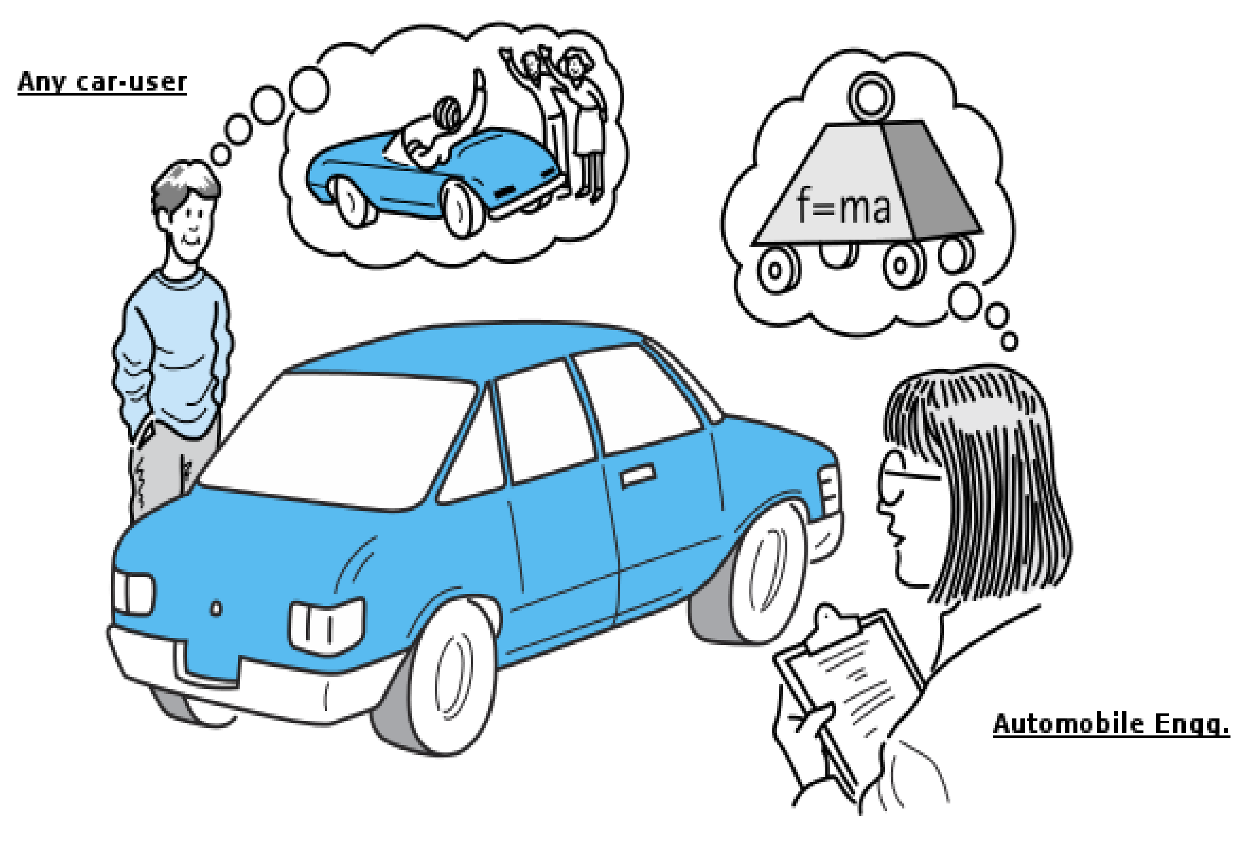


Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

- Denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crispy defined conceptual boundaries, relative to the perspective of the viewer
- Focus on interface (outside view)
- Separate behaviour from implementation

1. Abstraction

Focuses on the essential characteristics of some object, relative to the perspective of the viewer



An abstraction includes the essential details relative to the perspective of the viewer

1. Abstraction

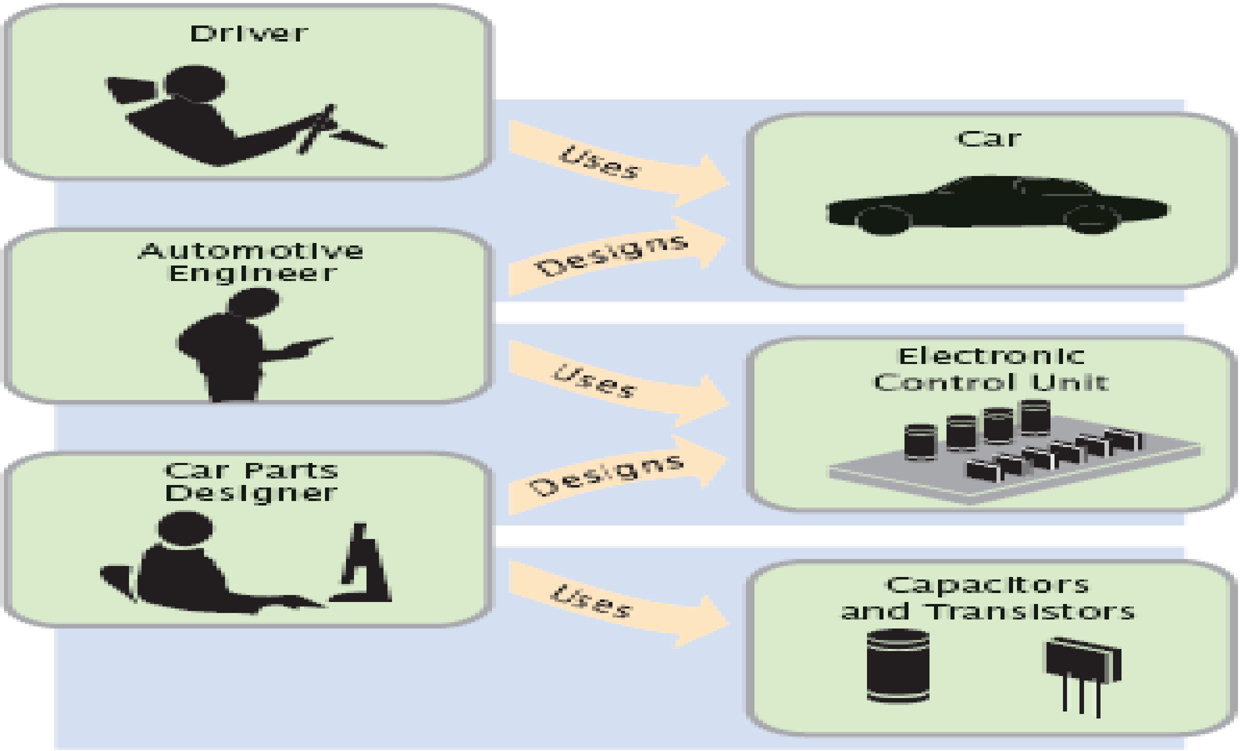
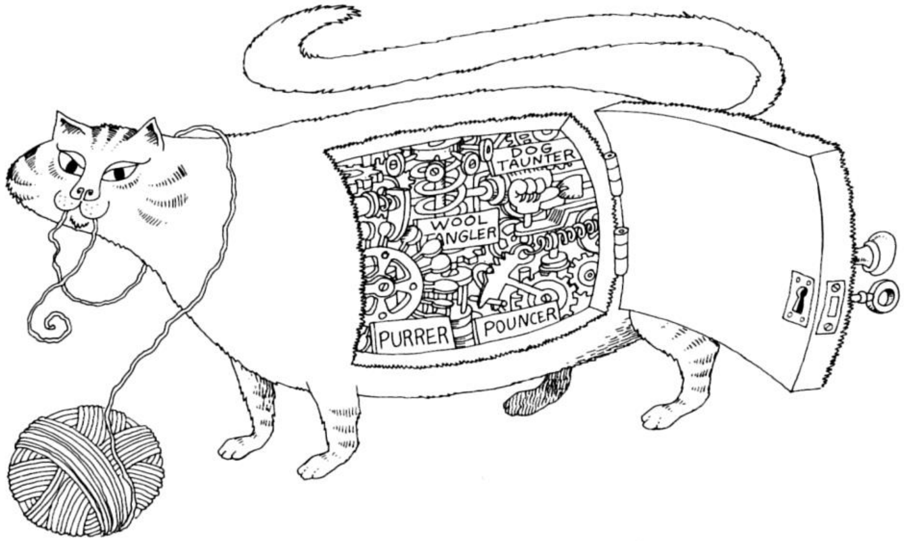


Figure 1

Levels of Abstraction in Automotive Design

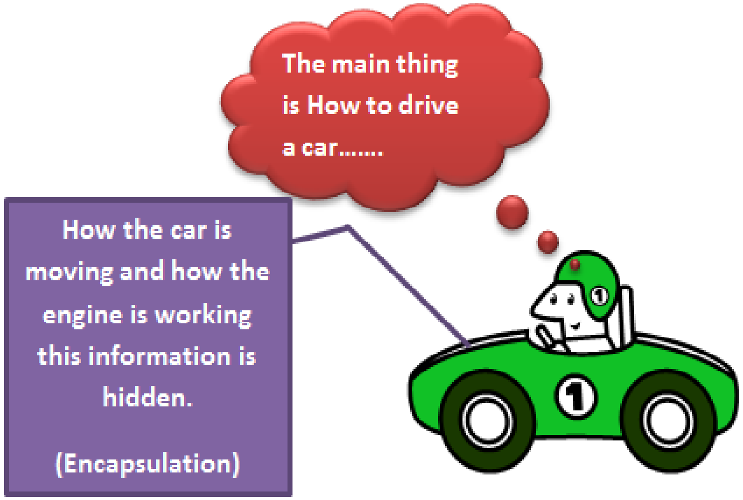
2. Encapsulation



Encapsulation hides the details of the implementation of an object.

- Complementary to abstraction
- Focuses on the implementation that give rise to behaviour of an object
- Also known as information hiding. Hides the details of the implementation (typically hides the structure of an object as well as the implementation of its method)
- Allows program changes to be reliably made with limited effort.

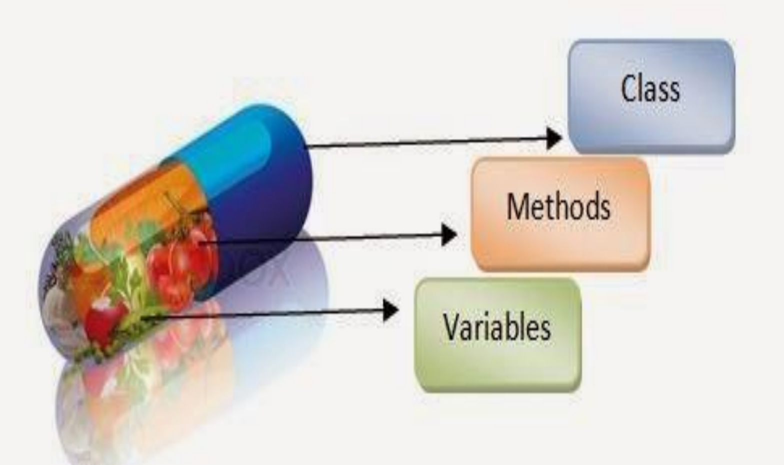
2. Encapsulation

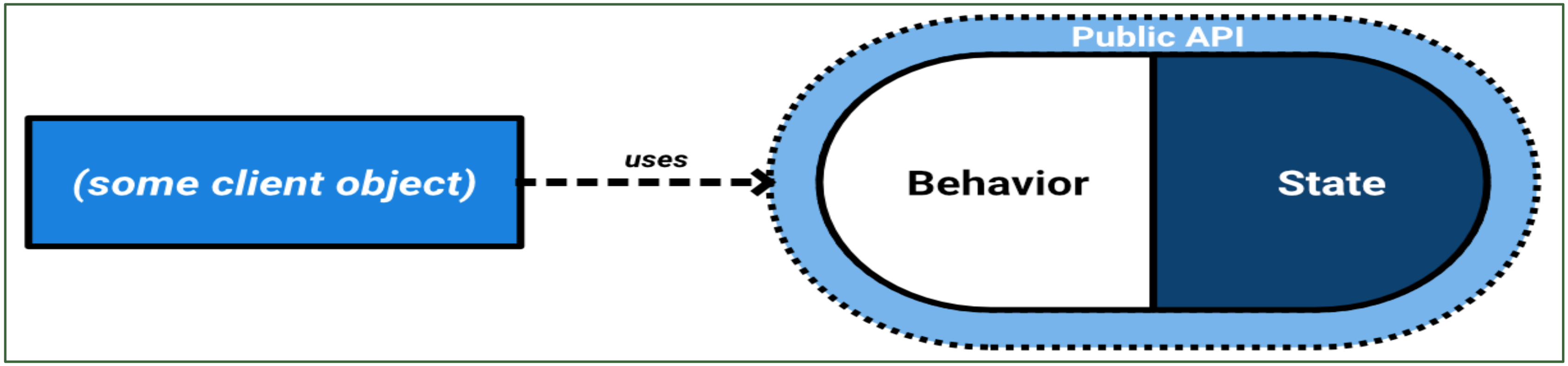


The main thing is How to drive a car…….

How the car is moving and how the engine is working this information is hidden.

(Encapsulation)





(some client object)

uses

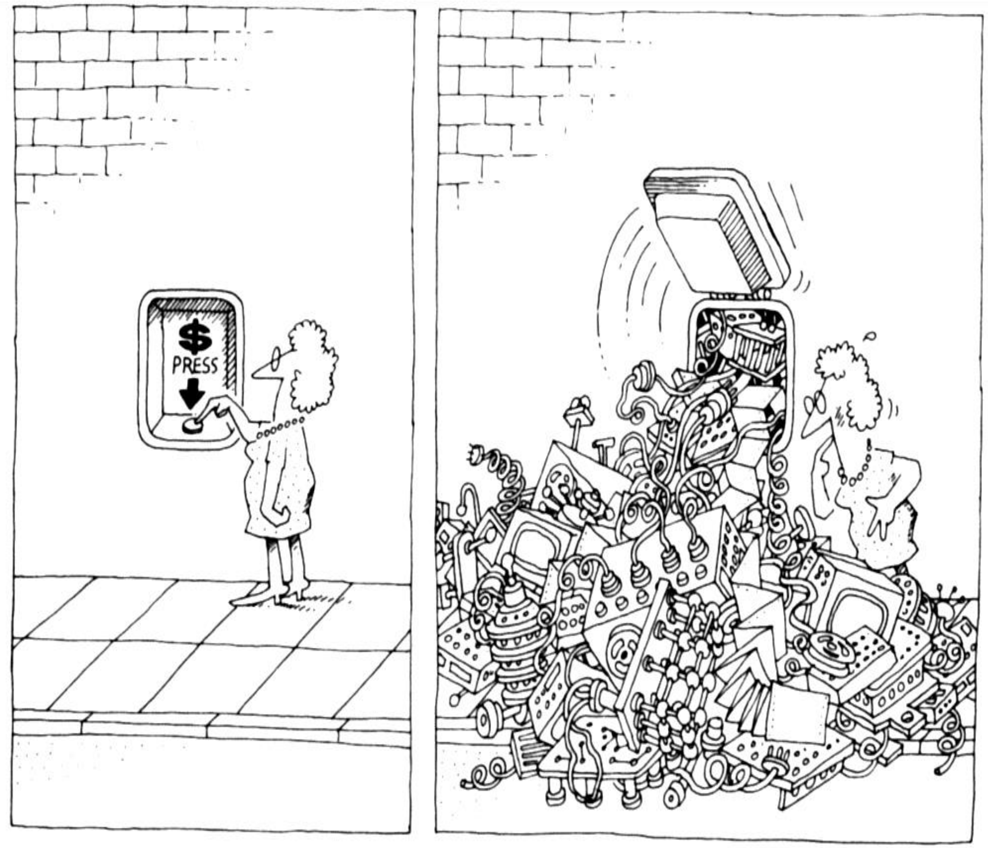
Public API

Behavior

State

Abstraction, Encapsulation and Software Design

- Interface should be simple providing the required behaviour.
- User is presented with high level abstract view. The detail of the implementation hidden from user.
- The designer may change the implementation keeping interface the same.



The task of the software development team is to engineer the illusion of simplicity.

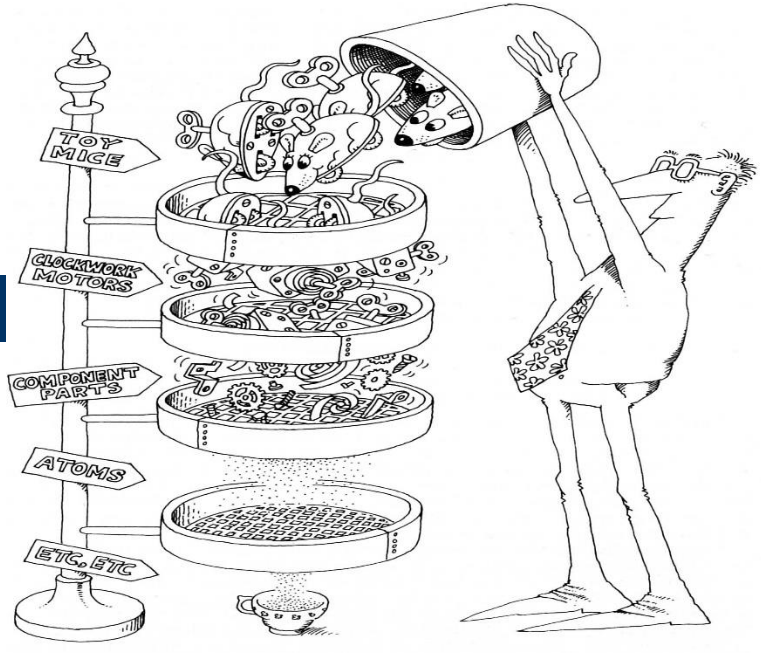
3. Modularity

- The act of partitioning a program into individual components to create a number of well defined, documented boundaries within the program.
- A common “Divide and conquer” approach
- Modules serves as the physical container in which we declare classes and objects of our logical design.
- Partitions a problem into sub-problems reduced complexity
- Modularity packages abstractions into discrete units
- In Java/C++ classes are the basic modules providing encapsulation and abstraction

Modularity packages abstractions into discrete units.

3. Modularity

4. Hierarchies



Abstractions form a hierarchy.

- A set of abstractions often forms a hierarchy
- Is a ranking or ordering of abstractions
- Two most important hierarchies in a complex system are its class structure (the “IS A” hierarchy) and its object structure (the “HAS A” or “PART OF” hierarchy)
- Interfaces and behaviours at each level
- Higher levels are more abstract
- Inheritance is the most important “is a” hierarchy and aggregation for “has a” or “part of” hierarchy.

Hierarchy: Inheritance (“is-a”)

- Inheritance is an essential element of object-oriented system
- It defines a relationship among classes, wherein one class shares the structure or behavior defined in one or more classes.
- Represents a hierarchy of abstractions in which a subclass inherits from one or more superclass.
- Typically, a subclass augments or redefines the existing structure and behavior of its super classes

Hierarchy: Aggregation (“part-of”)

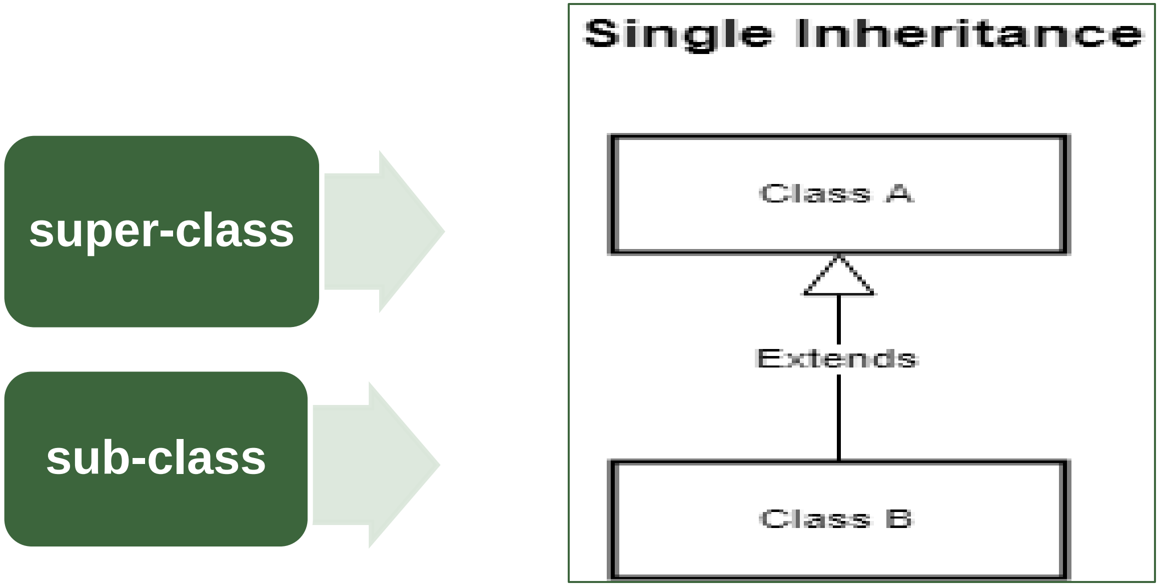
“Part of hierarchies describe aggregation relationships”

Consider the relationship between a garden and the plants in that garden. We can say that the plants “are part” of the garden

Aggregation permits the physical grouping of logically related structures and inheritance allows these common groups to be easily reused among different abstractions.

Examples of Hierarchy

Single Inheritance - one class shares the structure and behavior defined in one class. A subclass derives from a single super-class

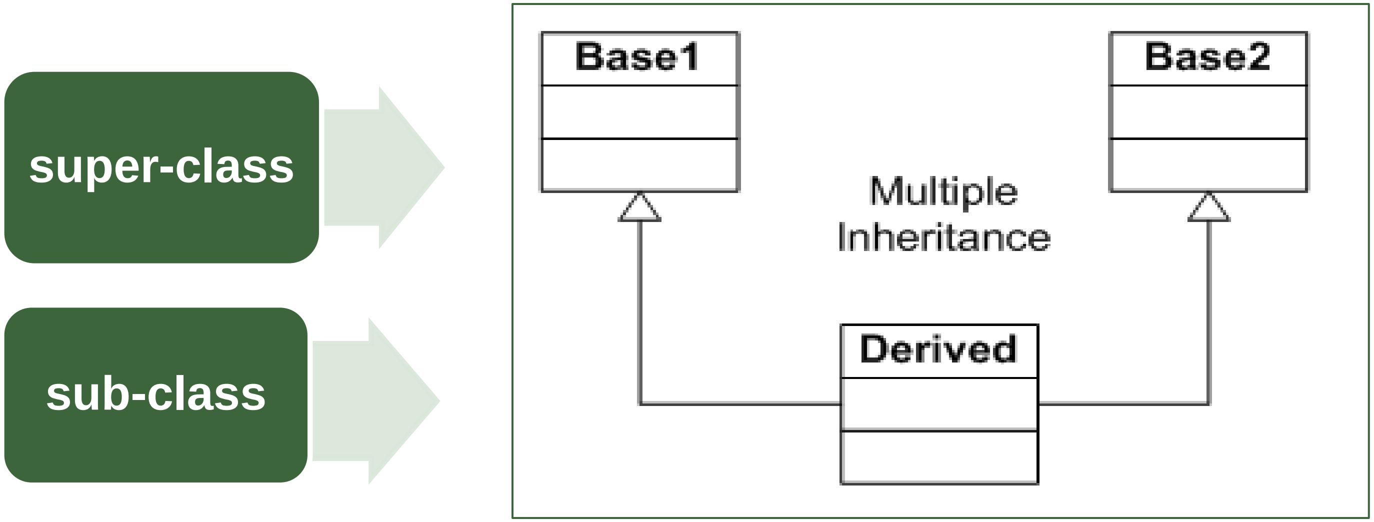


super-class

sub-class

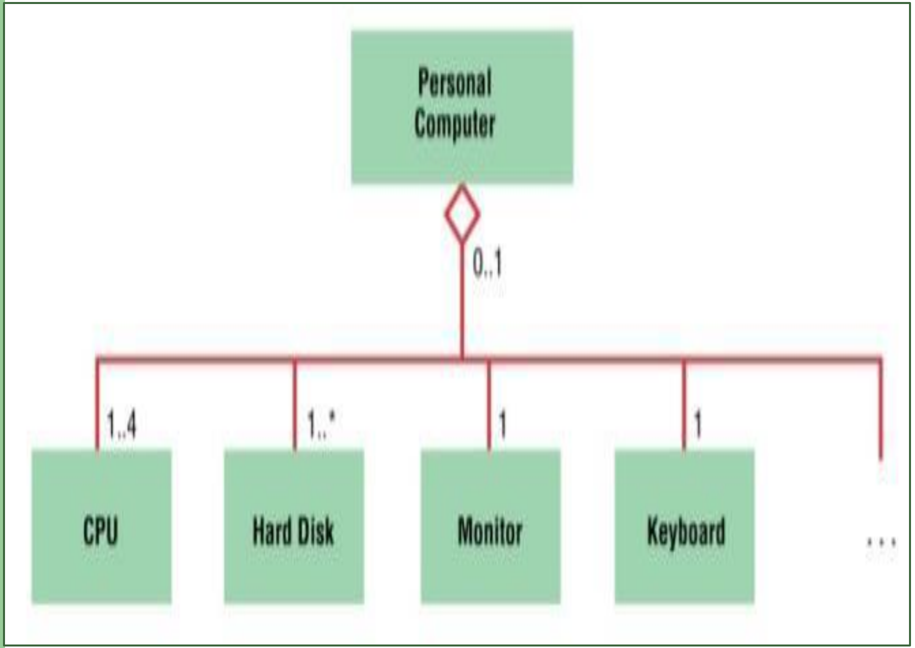
Examples of Hierarchy

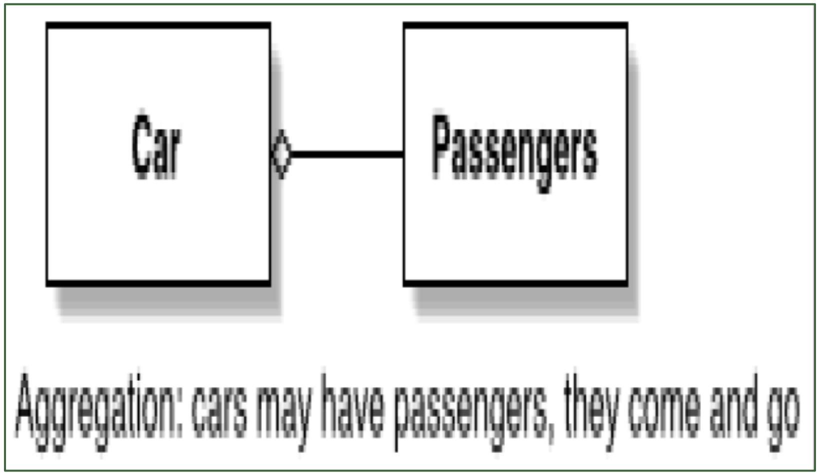
Multiple Inheritance - one class shares the structure and behavior defined in more than one class. A subclass derives from more than one super-class



Examples of Hierarchy

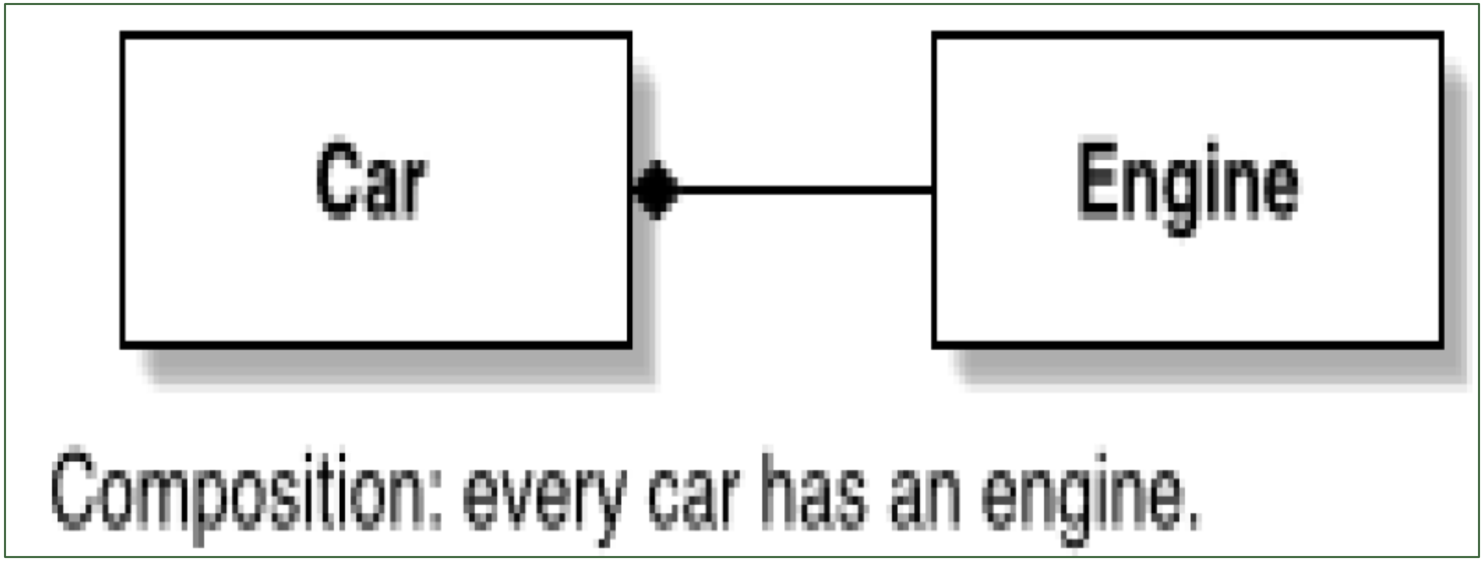
Aggregation- denotes a whole/part of hierarchy, with the ability to navigate from the whole (also called the aggregate) to its parts





Examples of Hierarchy

<u>Composition</u> – a stronger form of aggregation. The composite object has sole responsibility for the creation and destruction of all its parts.

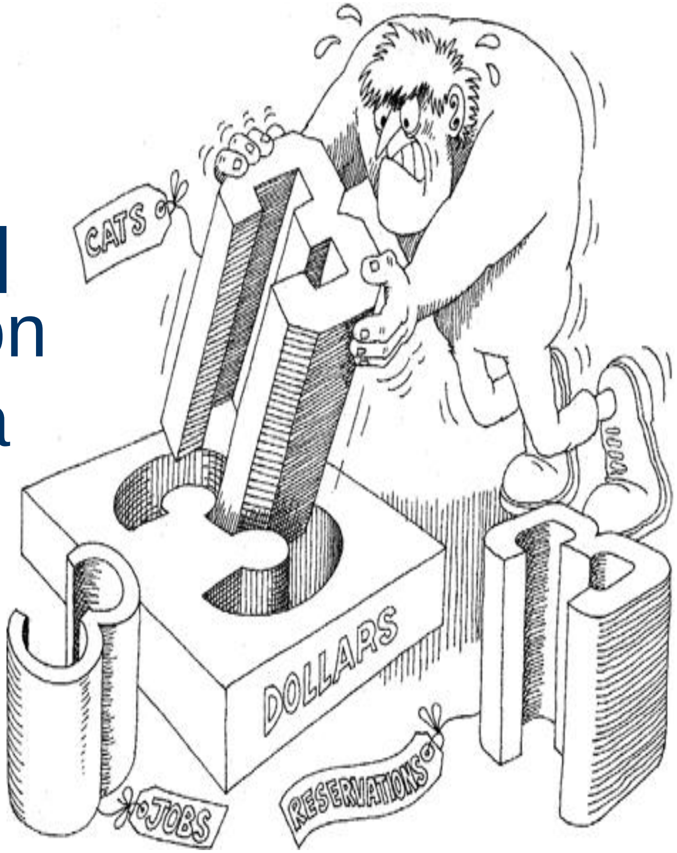


Composition: every car has an engine.

Minor Elements of The Object Model

1. Typing

It is the enforcement of the notion that an object is an instance of a single class or type; such that objects different types may not be interchanged, or at the most they may be interchange only in very restricted ways.



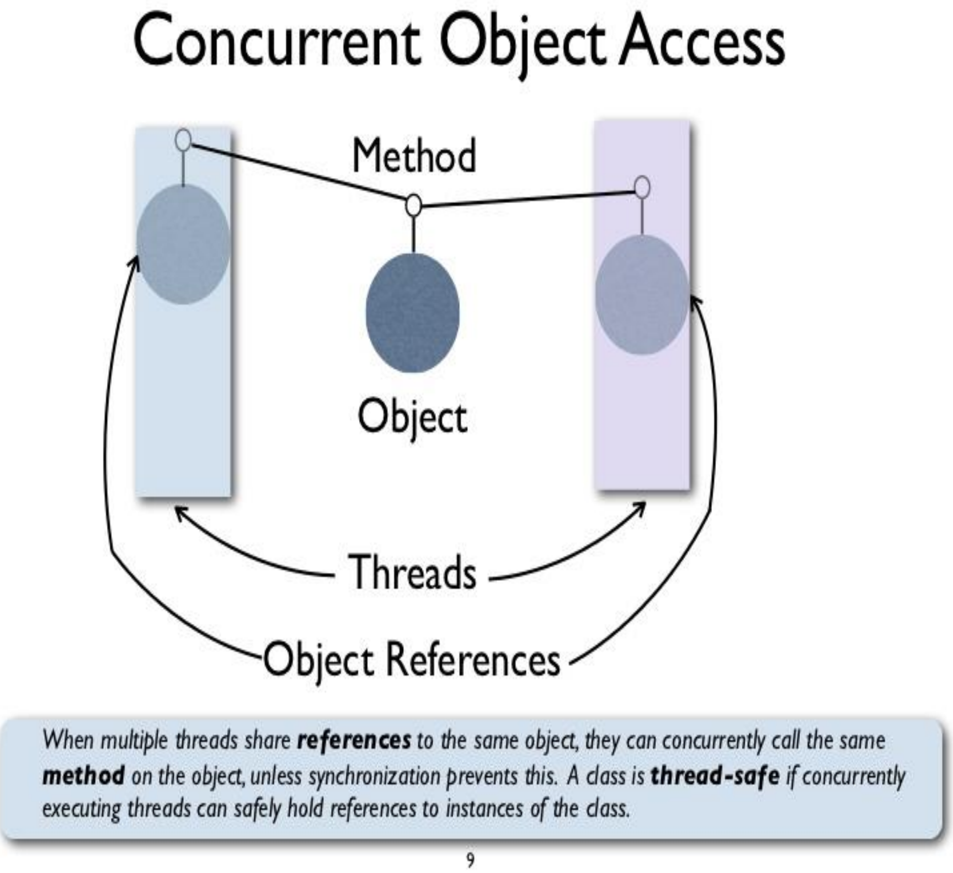
<u>Two Types of Typing</u>

Strong – operation is checked at compile time

Weak – operation is checked only at execution time

2. Concurrency

- Concurrency in OS allows performing multiple tasks or process simultaneously
- Means simultaneous execution of more than one thread.



Concurrent Object Access

When multiple threads share **references** to the same object, they can concurrently call the same **method** on the object, unless synchronization prevents this. A class is **thread-safe** if concurrently executing threads can safely hold references to instances of the class.

9

3. Persistance

- An object occupies a memory space and exists for a particular period of time.
- The property by which an object continues to exists even after its creator ceases.

The Nature of An Object

What Is and What Isn’t an Object?

- A tangible entity that exhibits some-well defined behavior
- From human perspective an object is any of the following:

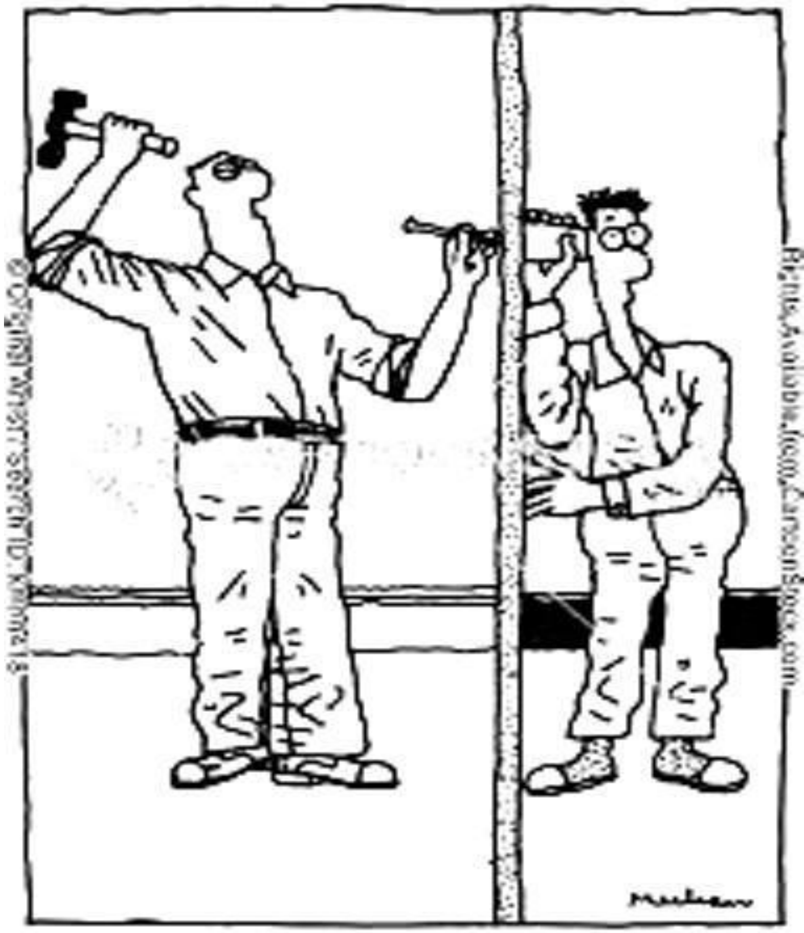
1. A tangible and/or visible thing
2. Something that may be comprehended intellectually
3. Something toward which thought or action is directed

What is an Object?

- “An object has **state**, **behavior**, and **identity**; the structure and behavior of similar objects are defined in their common class.” The term instance and object are interchangeable.
- Some things are not objects, but are attributes; e.g. age, color.
  - Attributes may be *properties* of some object

What is an Object?

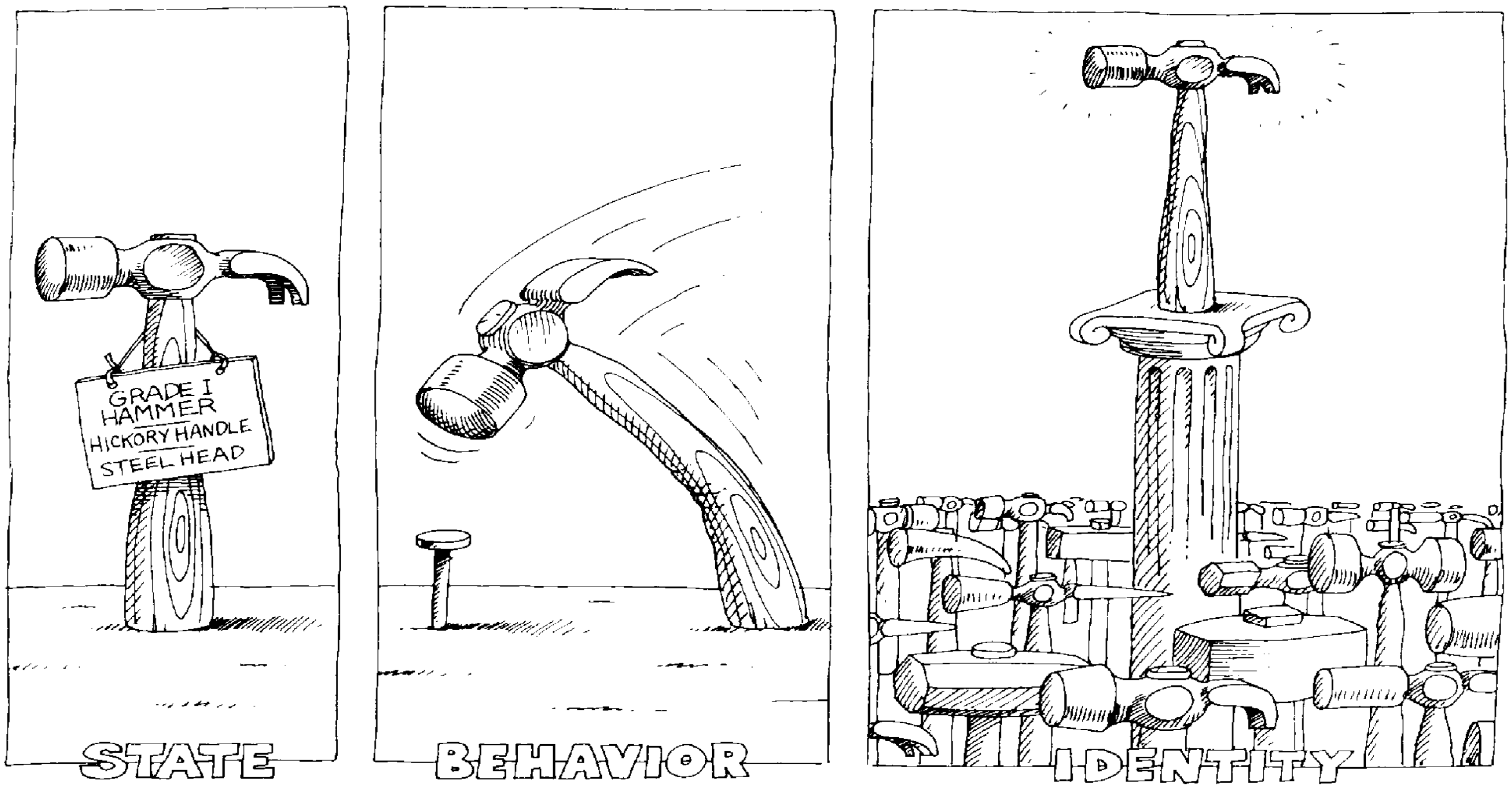
A person holding a hammer tends to see everything in the world as a nail, so the developer with an object-oriented mindset begins to think that everything in the world is an object which is wrong since some things are distinctly not objects.



© Original Artist search ID: kmnn218

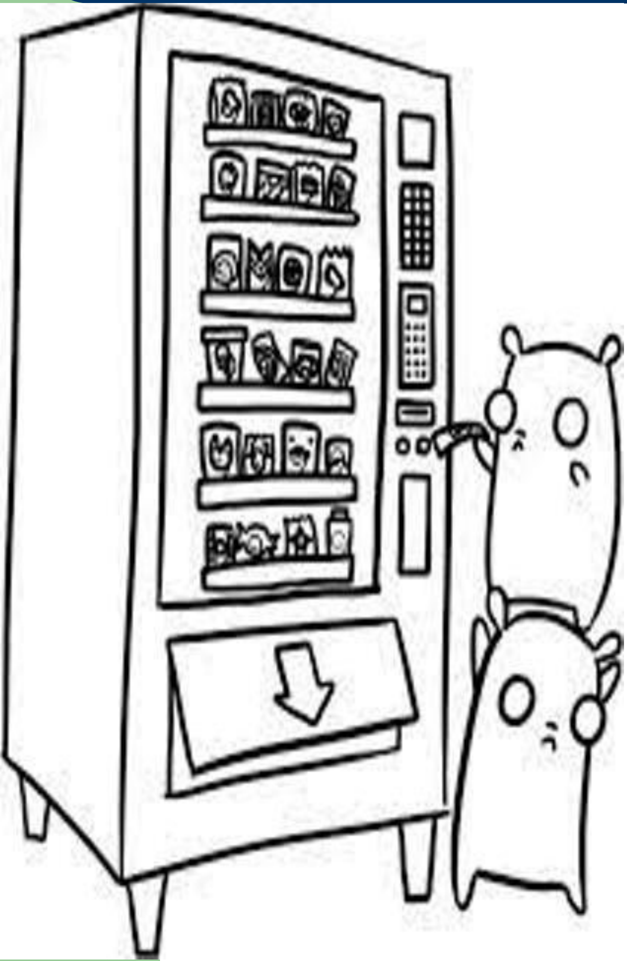
Rights Available from CartoonStock.com

What is an Object?



An object has state, exhibits some well-defined behavior, and has a unique identity.

State



Vending machine is in a state (waiting for money). After the money is inserted, the vending machine will move to another state (waiting for the user to push a button to make a selection).

The order in which one operates on the object is important and the reason for this event-and-time dependent behavior is the existence of state within the object.

State

- The state of object encompasses all of the properties of the object (usually static) plus the current values (usually dynamic) of each of these properties.
- The state of an object consists of a set of data fields (its properties) with their current values.
  - The state represents the cumulative results of an object’s behavior.
- A property is an inherent characteristic, trait, or quality that distinguishes one kind of object from another (i.e. its class).

State

- All properties have some value.
  - May be a simple quantity.
    - E.g. a person’s net worth is Rs 25000.
  - Values are usually dynamic.
    - E.g. a person’s net worth may go up or down.
  - Occasionally a value is static.
    - E.g. a car’s serial number.
- The state of an object changes in response to events (i.e. a message sent to it), or by acting autonomously.

Behavior

- “Behavior is how an object acts and reacts in terms of its state changes and message passing.”
- When you send a message to an object, you actually invoke a method (execute some code).
- Invoking a method will cause certain well-defined behavior, and may change the object’s state.
  - The behavior may depend on the object’s current state
- The behavior of an object is embodied in the sum of its operations

Identity

- “Identity is that property of an object which distinguishes it from all others.”
- Every instance of a class (object) has its own memory to hold its state.
- An object retains its identity even if some or all of the values of variables or definitions of methods change over time.

Several Forms of Identity

- **Value**: A data value is used for identity (primary key of a tuple in a relational value)
- **Name**: A user-supplied name is used for identity (filename in a file system)
- **Built-in**: A notion of identity is built-into the data model or programming language and no user-supplied identifier is required.

Identity

- An object has both state and behavior : the state defines the object whereas the behavior defines what the object does
- Objects are the real time entity which are created through their template, their class.
- By real time entity, I mean they would occupy a memory space in which they will save the values of its attributes. These values create a state of an object. Since Object occupy a space in memory so they have unique address in memory. This become the identity of an object.

Identity

For example : Car is a class, which has model and color as its attributes. They may have behavior like turnRight(), turnLeft().

Still they don't occupy a space in memory.

When I say My car ,It will be an object of class Car since it will have some values for model like 2014 and color like red.

This will occupy some memory to save these values.

Operation

An operation denotes a service that a class offers to its client. In practice a client typically performs five kind of operations on an object.

- The three most common kind of operations:
  - **Modifier:** alters the state of an object.

    Example: karan.setNetWorth(25000);
  - **Selector:** accesses the state of an object, but does not alter it. Example: age = s.getAge();
  - **Iterator:** permits all parts of an object to be accessed in some well defined order.

Operation

- Two other Kinds of Operations

  - Constructor: creates an object and initializes its state.

  - **Destructor:** destroys an object (frees its memory).

  Example:

```
class X {
 public:
              X();     // Constructor for class X
              ~X();   // Destructor for class X
              };
```

Roles and Responsibilities

All of the methods associated with a particular objects comprise its protocol which comprises the entire static and dynamic view of the object

It is useful to divide this protocol into logical groupings of behavior. These collections denotes the roles that an object can play

A **role** is a mask that an object wears and so defines a contract between an abstraction and its clients.

Objects Can Play Many Different Roles

- The **responsibilities** of an object are all the services it provided for all the of the contracts it supports.
- The state and behavior of an object collectively define the roles that an object may play in the world.
- The roles played by objects are dynamic yet can be mutually exclusive.

Relationships Among Objects

- Objects contribute to the behavior of a system by collaborating with one another.
- Two Kinds of Object Relationship

    1. Links
    2. Aggregation

Links

- Defined as a physical or conceptual connection between objects.
- An object collaborates with other objects through its links to these objects.
- As a participant in a link, an object may play these roles

1. **Controller**: can operate on another object but not operated on by other object
2. **Server**: doesn’t operate on another object, it is only operated on by other object
3. **Proxy:** can both operate and be operated. Usually created to represent real-world objects

Aggregation Link (“has a”)

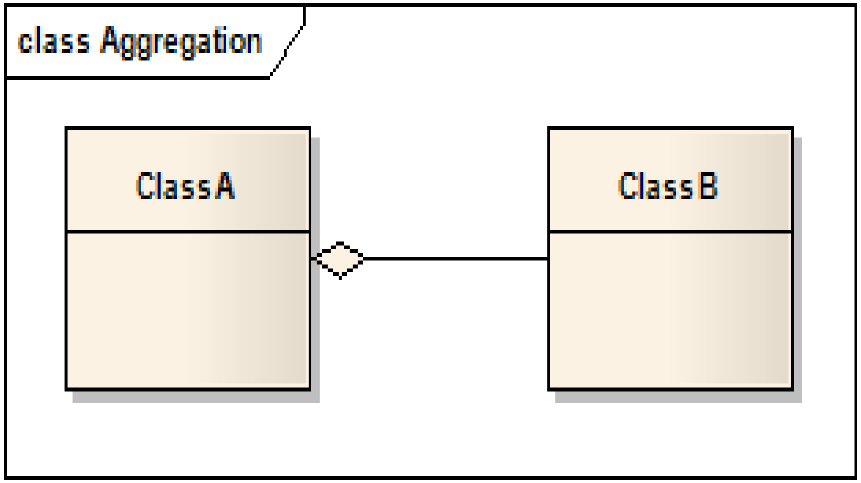
- In cases where there’s a part-of relationship between two classes, we can be more specific and use the aggregation link instead of the association link, highlighting that a class can also be aggregated by other classes in the application.
- Aggregation is therefore also known as shared association

The part (Class B)

may be independent

of the whole (Class A)

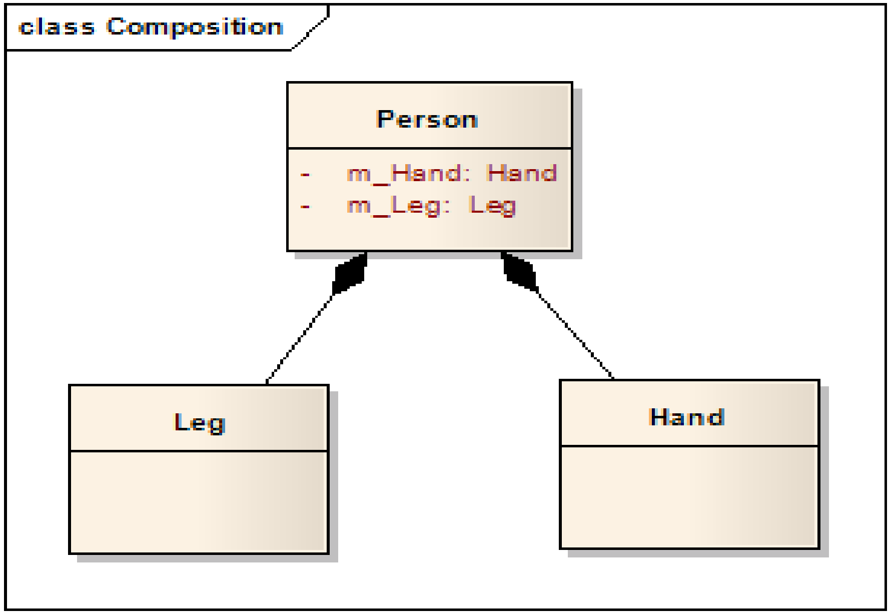
but the whole requires the part



Composition Link (“part of”)

- Specific and use the composition link in cases where in addition to the part-of relationship (for example between Class A and Class B), there is strong lifecycle dependency between the two (stronger version of aggregation).
- When Class A is destroyed, then Class B will also be destroyed

A stronger form of aggregation where the part is created and destroyed with the whole



Aggregation vs Composition

Aggregation implies a relationship where the child can exist independently of the parent.

Example: Class (parent) and Student (child).

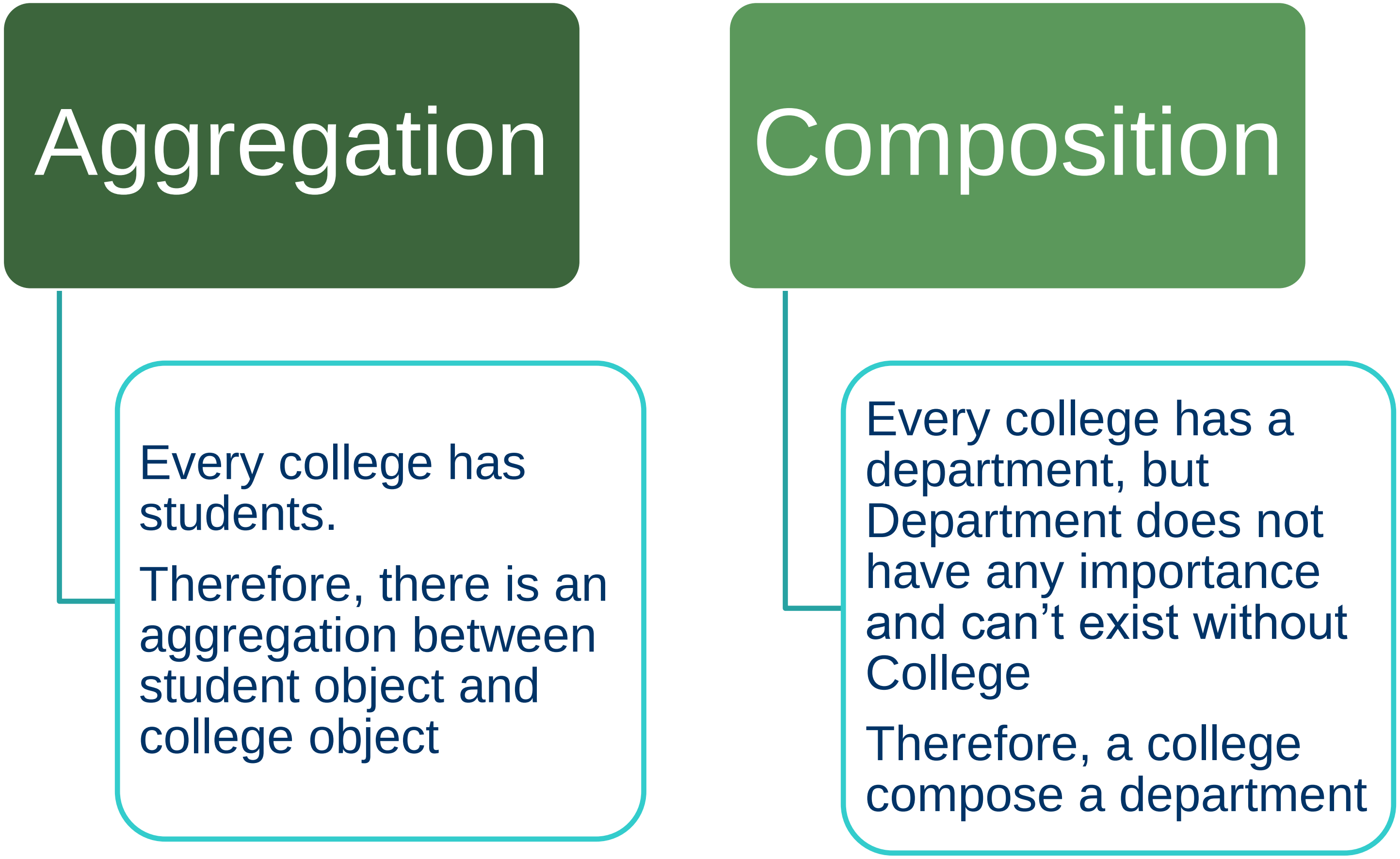
Delete the Class and the Students still exist.

Composition implies a relationship where the child cannot exist independent of the parent.

Example: House (parent) and Room (child).

Rooms don't exist separate to a House.

Aggregation vs Composition



Aggregation

Every college has students.

Therefore, there is an aggregation between student object and college object

Composition

Every college has a department, but Department does not have any importance and can’t exist without College

Therefore, a college compose a department

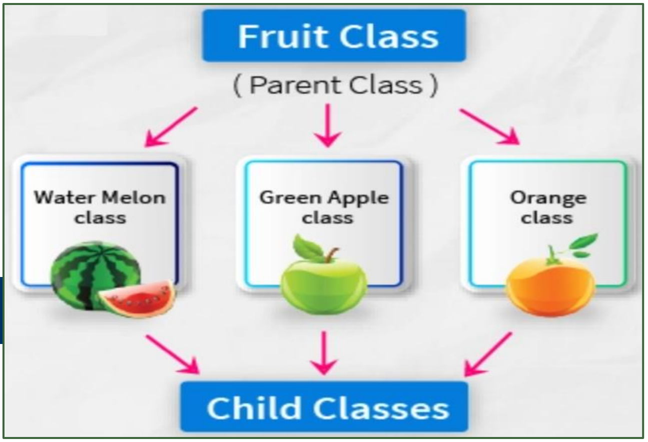
The Nature Of A Class

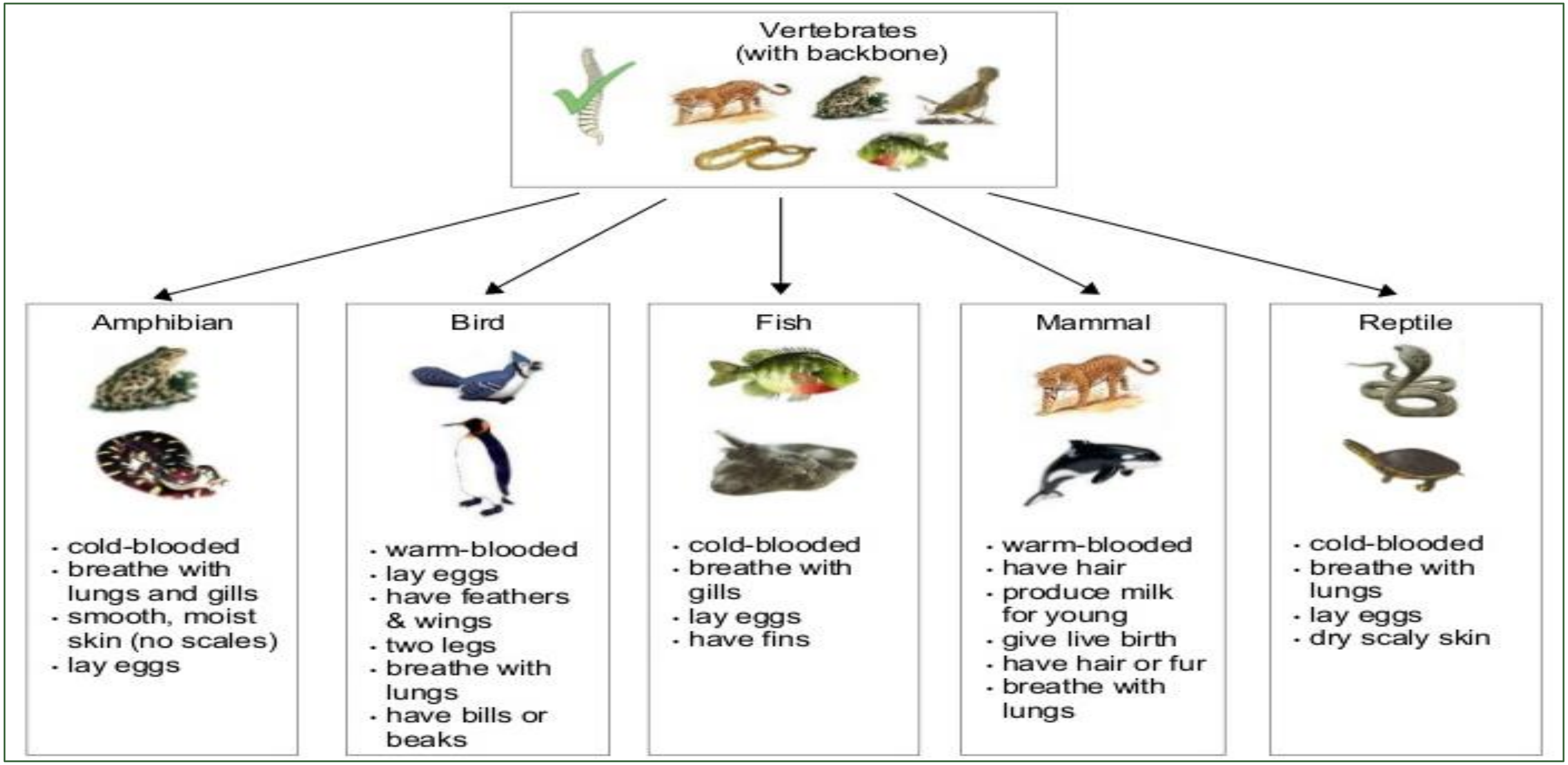
- The concept of a class and an object are tightly interwoven, for we cannot talk about an object without regards for its class. However, there are important differences between these two terms

What is a class?

- A group, set, or kind marked by common attributes; a group division, distinction, or rating based on quality, degree of competence, or condition. (from Webster)
- “A class is a set of objects that share a common structure and a common behavior.”
- A class is an abstraction, a way of classifying similar objects.
- An object is an instance of a class, a concrete entity that exists in time and space.

What is a class?





Amphibian

- cold-blooded
- breathe with lungs and gills
- smooth, moist skin (no scales)
- lay eggs

Bird

- warm-blooded
- lay eggs
- have feathers & wings
- two legs
- breathe with lungs
- have bills or beaks

Fish

- cold-blooded
- breathe with gills
- lay eggs
- have fins

Mammal

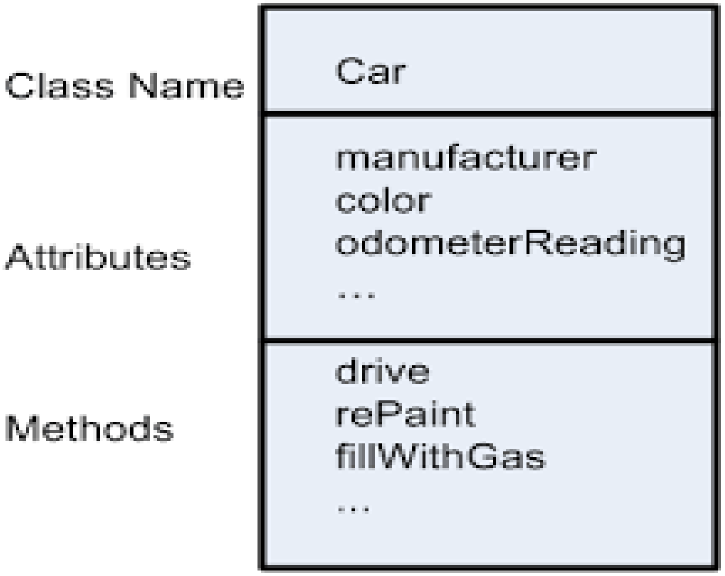
- warm-blooded
- have hair
- produce milk for young
- give live birth
- have hair or fur
- breathe with lungs

Reptile

- cold-blooded
- breathe with lungs
- lay eggs
- dry scaly skin

UML: Class Diagram

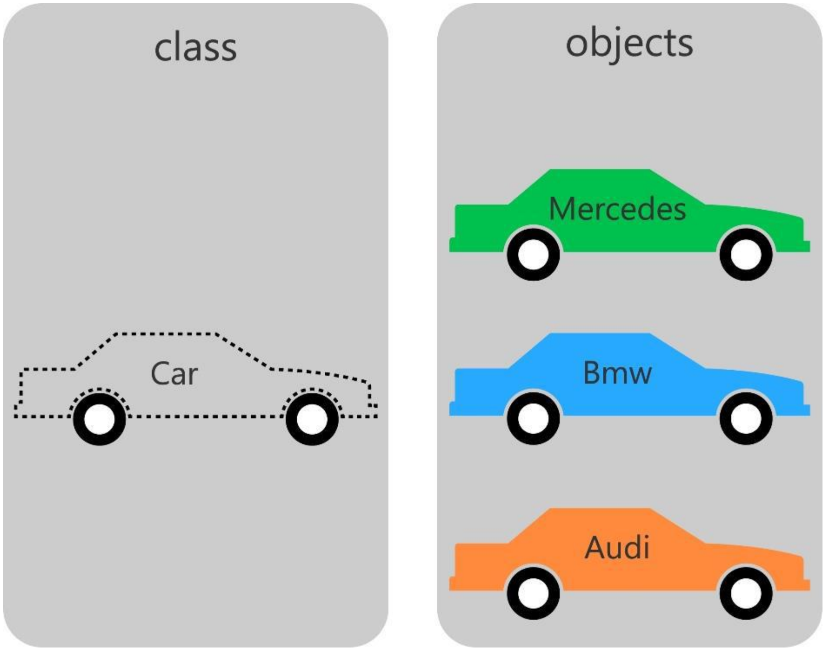
- Class is represented as a RECTANGLE with three (3) compartments
- Objects can participate in relationships with objects of the same class



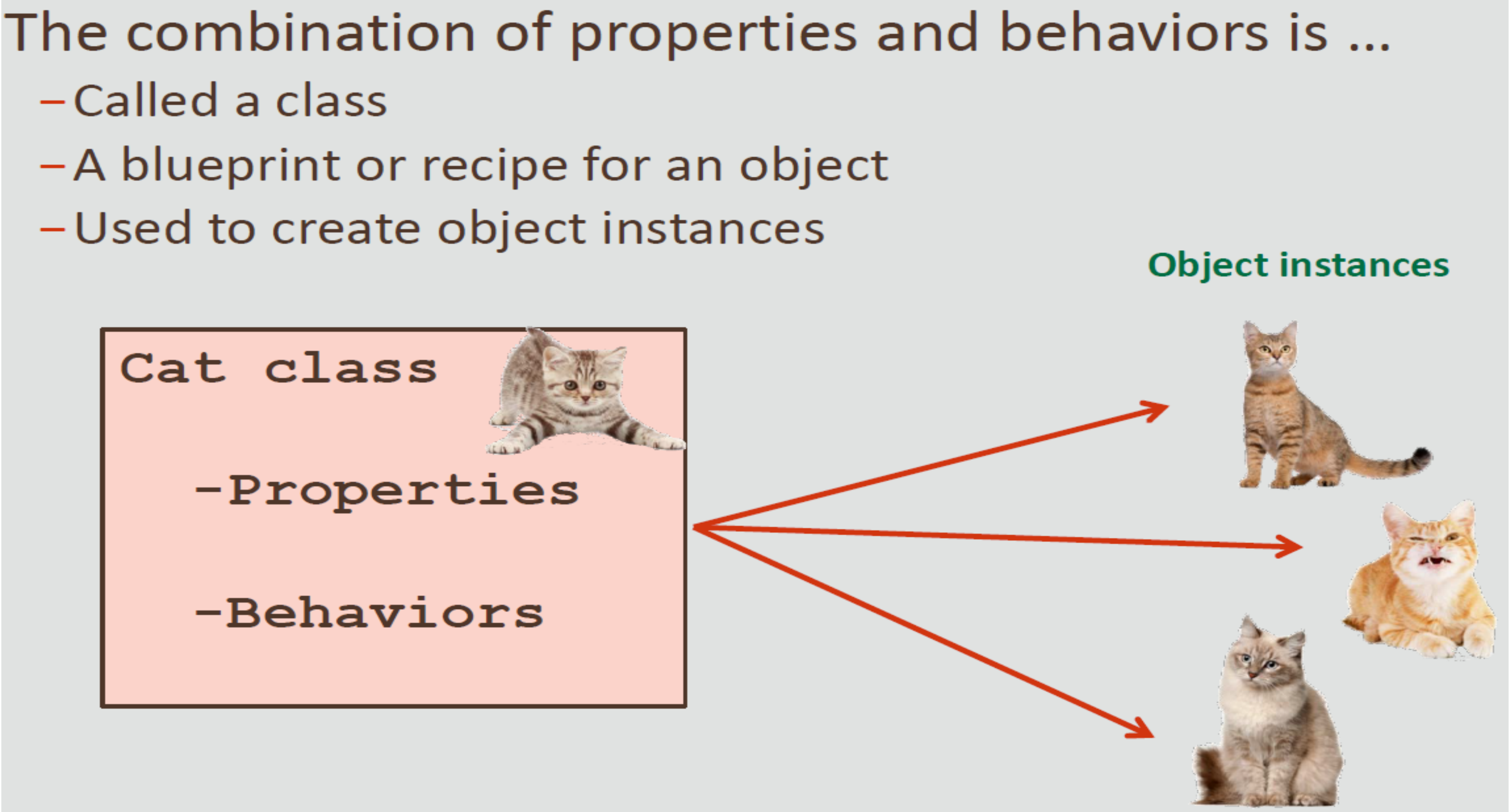
Class Name | Car

Attributes | manufacturer, color, odometerReading, ...

Methods | drive, rePaint, fillWithGas, ...



UML: Class Diagram



The combination of properties and behaviors is …

- Called a class
- A blueprint or recipe for an object
- Used to create object instances

Cat class

-Properties

-Behaviors

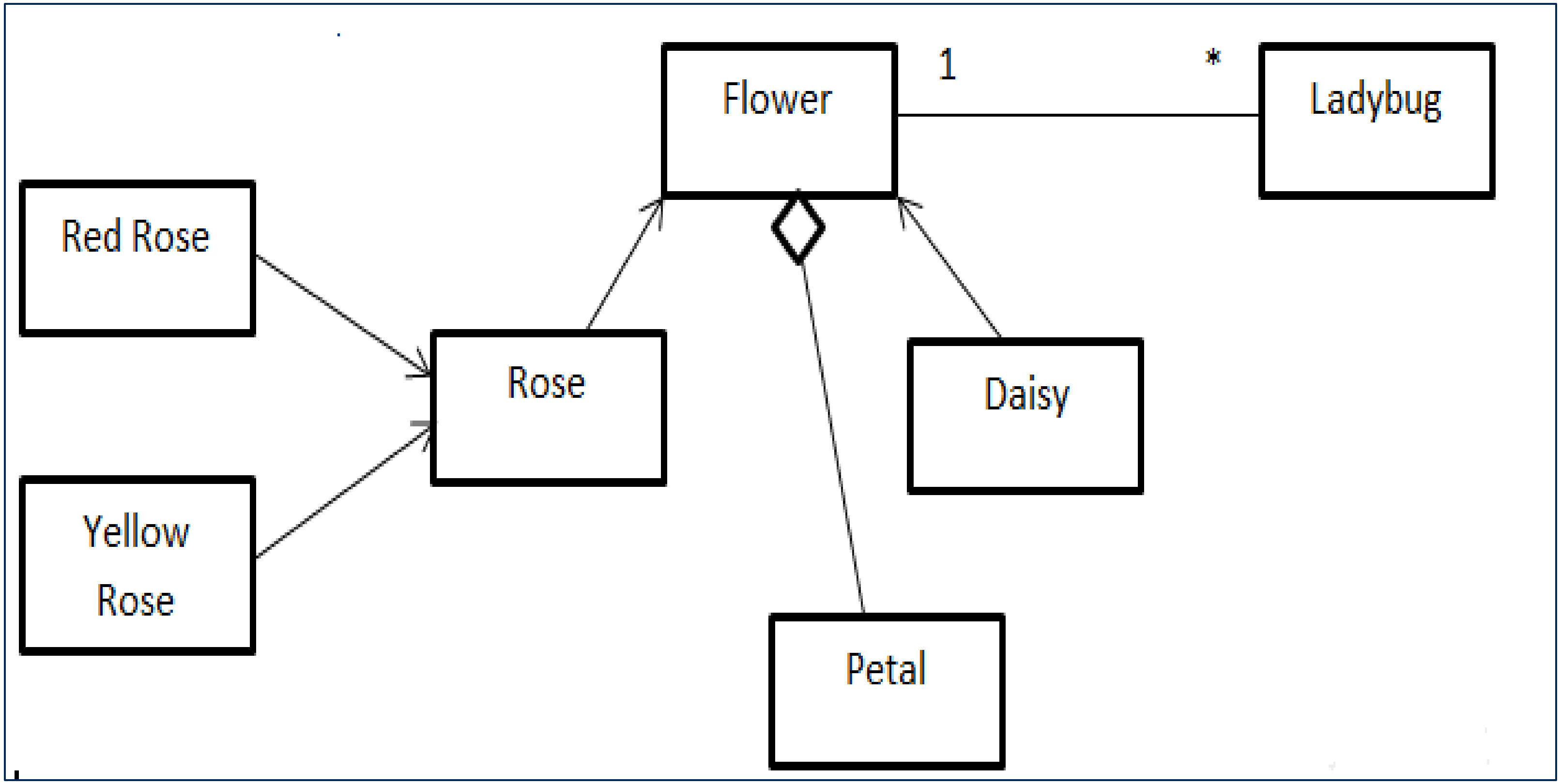
Object instances

Relationship Among Classes

Consider for a moment the similarities and differences among the following classes of objects: flowers, daisies, red roses, yellow roses, petals, and ladybugs.

- A daisy is a kind of flower
- A rose is a (different) kind of flower
- Red roses and yellow roses are both roses
- A petal is a part of both flowers
- Ladybugs eat certain pest such as aphids, which may be infesting flowers

Sample Class Diagram

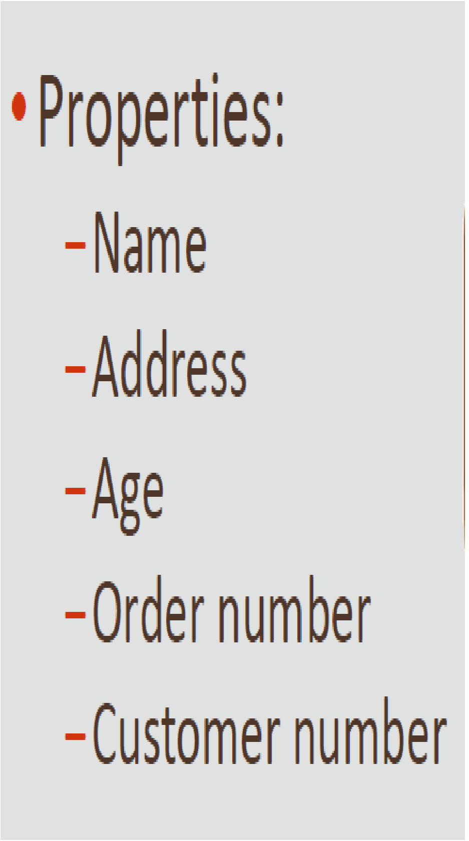


Activity: Class Diagram

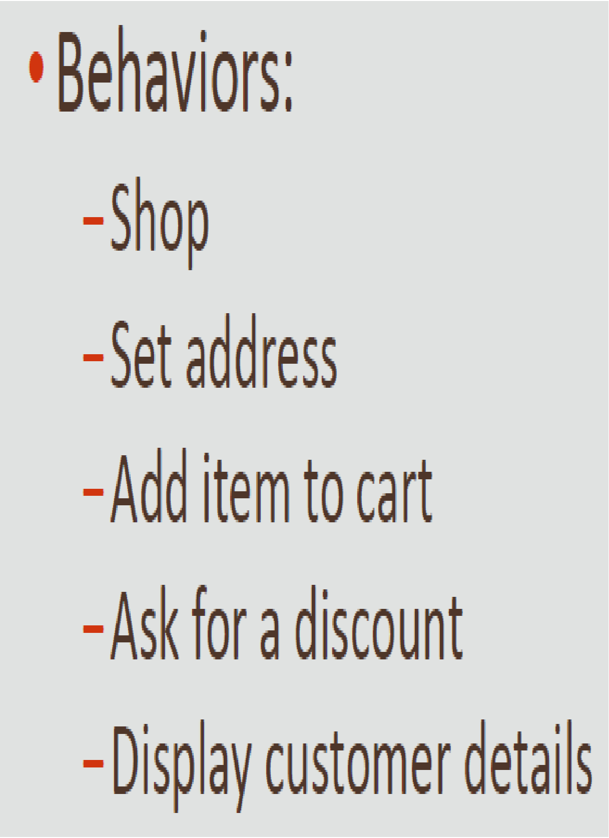
Customer

What could be the properties / attributes / data?

What could be the behavior / method?



- Properties:
  - Name
  - Address
  - Age
  - Order number
  - Customer number

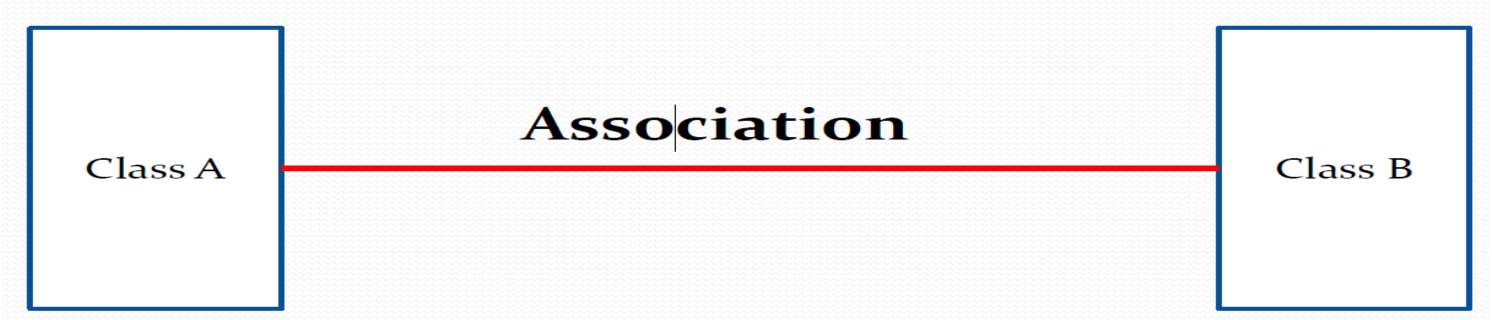


- Behaviors:
  - Shop
  - Set address
  - Add item to cart
  - Ask for a discount
  - Display customer details

Relationship Among Class

1. Association

- The most general but also the most semantically weak.
- The identification of association among classes is often an activity of analysis and early design.
- A relationship between object classes. It is depicted as a solid line between classes.



Two Types of Association

1. **Semantics**

- means naming the role each class plays in relationship with the other.
- This is sufficient during the analysis of a problem, at which time we need only to identify such dependencies.

2. **Multiplicity**

In practice there are three common kinds of multiplicity across an association:

(1)One-to-one (2)One-to-many (3)Many-to-many

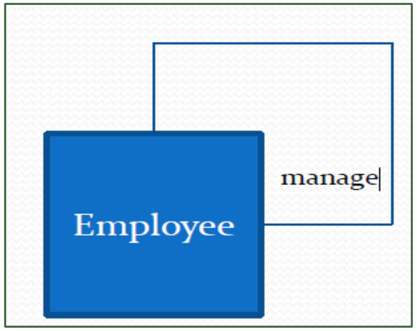
Relationship Among Class

Degree of Association – denotes the number of classes involved in a connection.

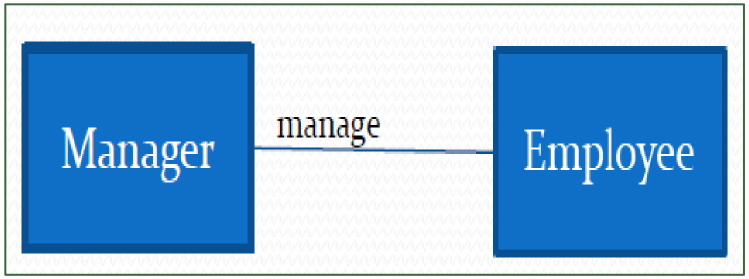
Unary relationship – connects objects of the same class

Binary relationship – connects objects of two classes

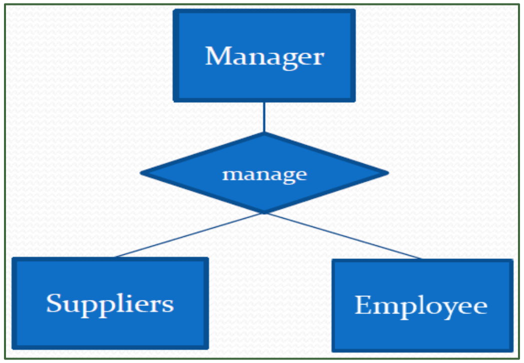
Ternary relationship - connects objects of three or more classes



Unary



Binary



Ternary

Relationship Among Class

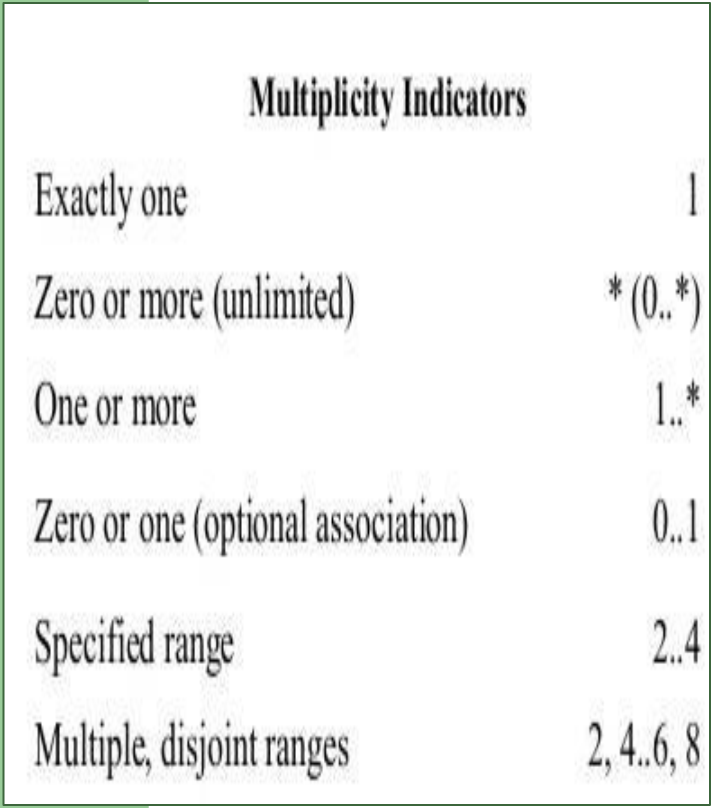
<u>Multiplicity</u>– the number of classes/objects that can participate in the association

One-to-one: single object A is associated with a single object B

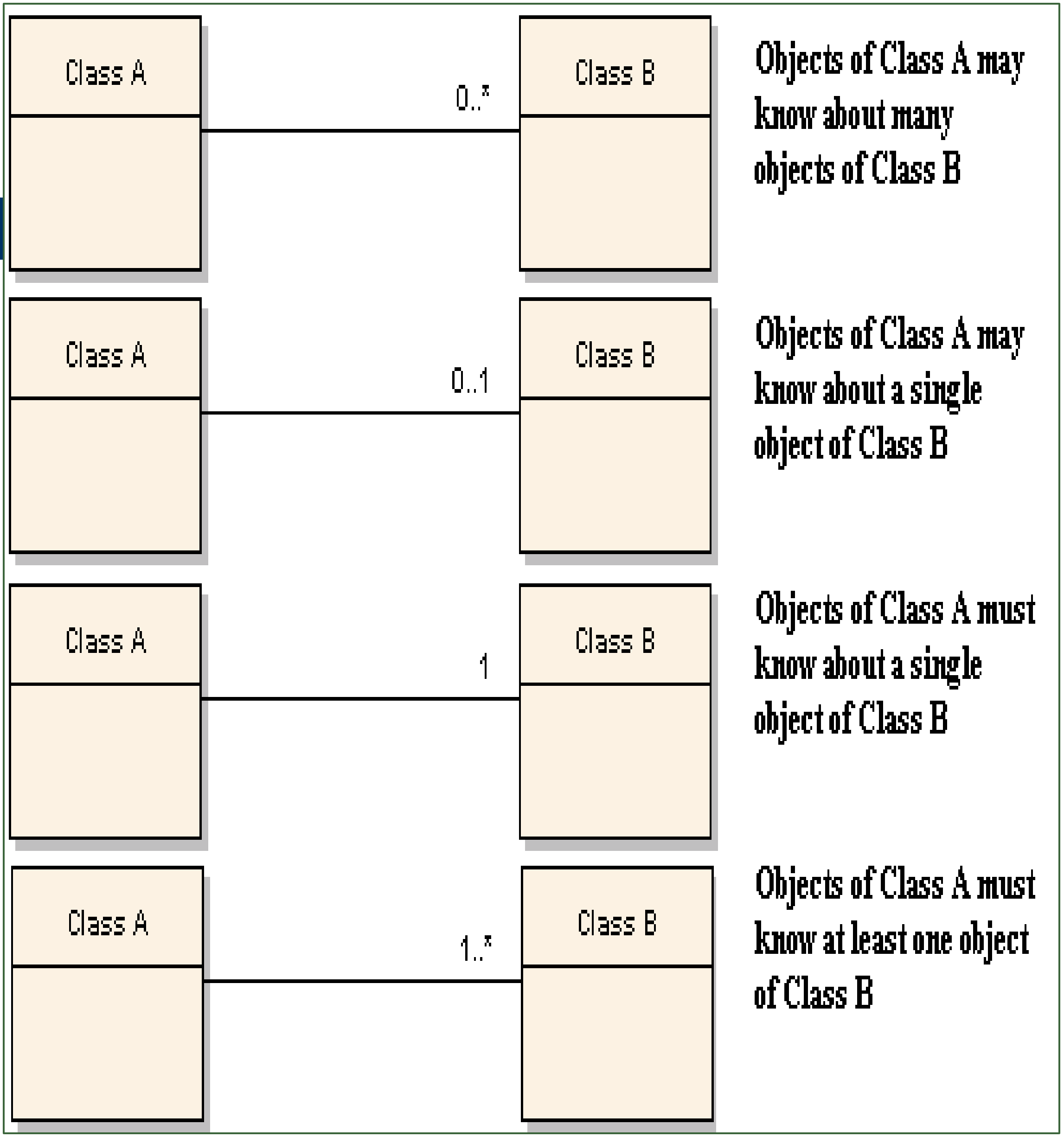
One-to-many: single object A is associated with a many object B

Many-to-many: a single object A maybe associated with many object B and conversely a single object B may be associated with many object A.

Multiplicity



Multiplicity Indicators	
Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8



Objects of Class A may know about many objects of Class B

Objects of Class A may know about a single object of Class B

Objects of Class A must know about a single object of Class B

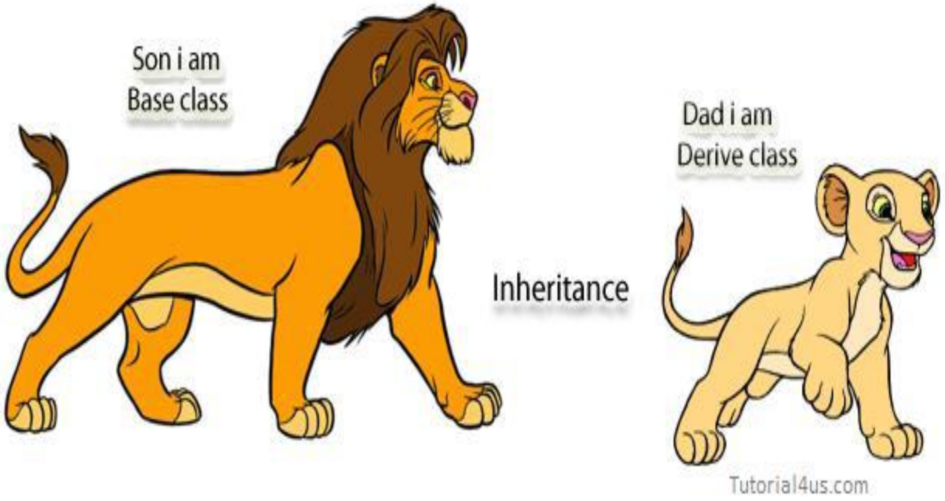
Objects of Class A must know at least one object of Class B

Relationship Among Classes

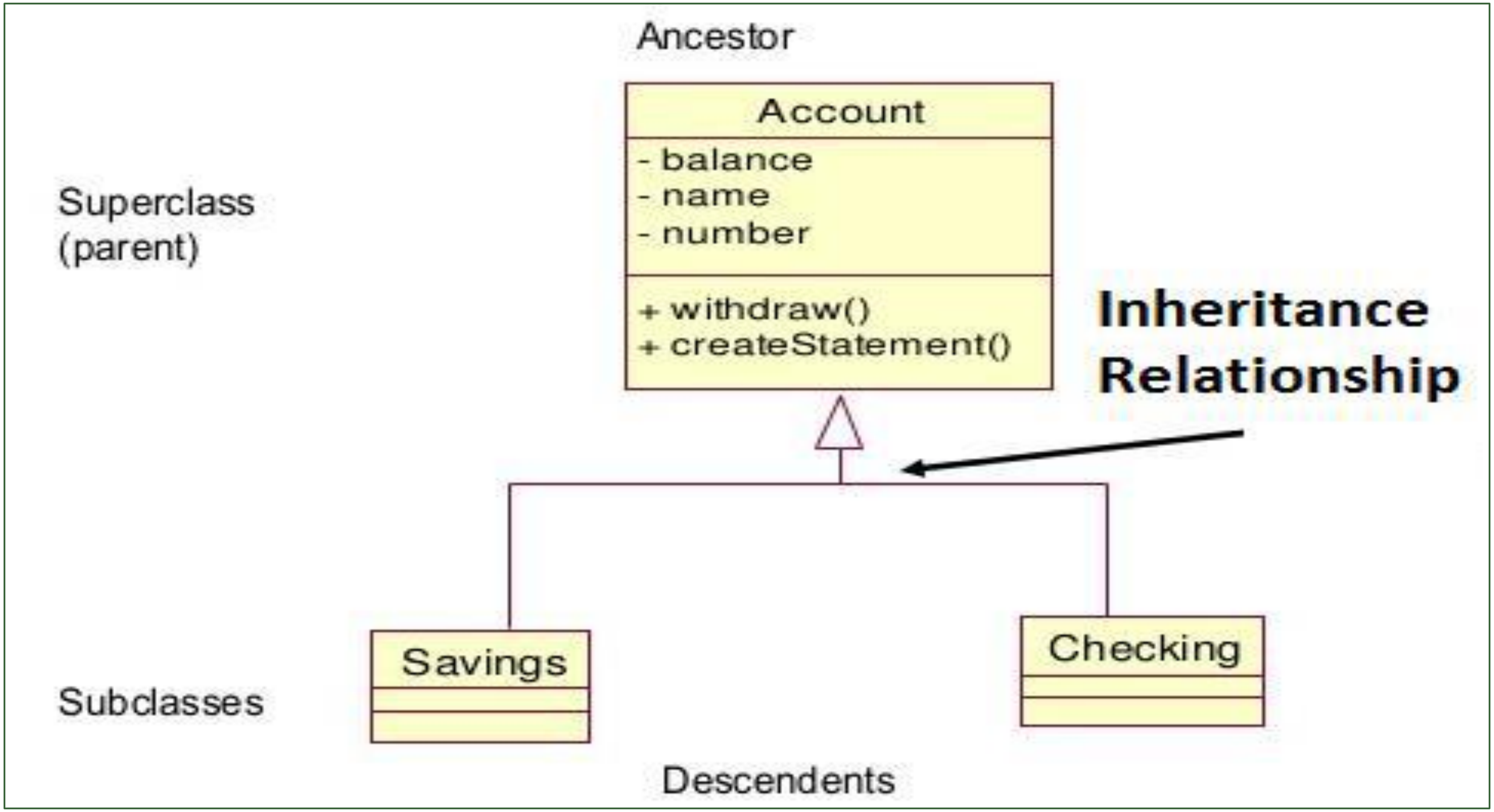
1. Inheritance

- The most semantically interesting of these concrete relationships, exists to express generalizations/specializations relationships.

A subclass may inherit
the structure
and behavior
of its super class



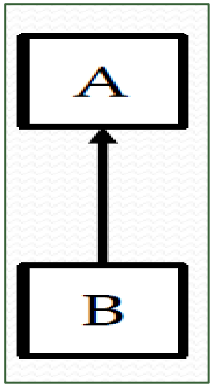
Relationship Among Classes



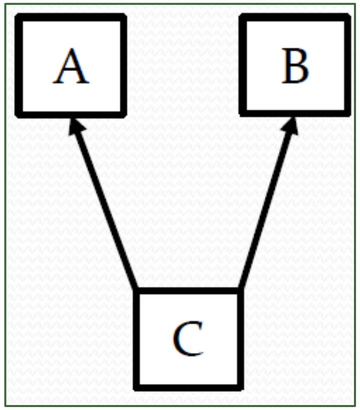
Types of Inheritance

- **Single Inheritance:** a subclass derives from one super-class
- **Multiple Inheritance:** a subclass derives from more than one super-class
- **Multilevel Inheritance:** a subclass derives from a super-class which in turn is derived from another super-class

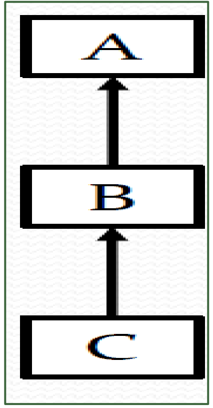
Single



Multiple



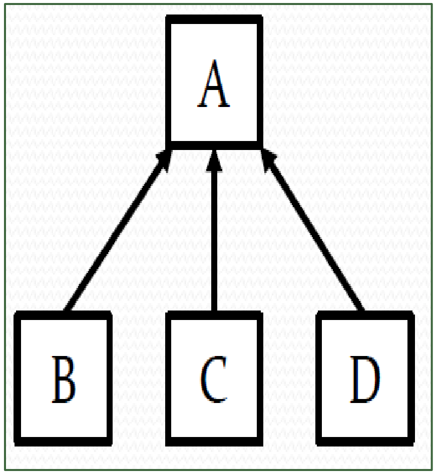
Multilevel

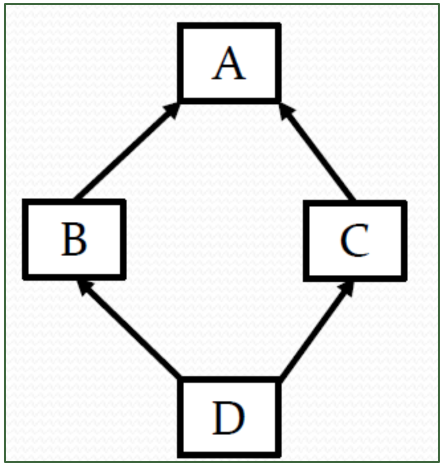


Types of Inheritance

- **Hierarchical Inheritance:** a class has a number of subclasses each of which may have subsequent subclasses continuing for a number of levels, so as to form a tree structure.
- **Hybrid Inheritance:** a combination of multiple and multilevel inheritance.

Hierarchical





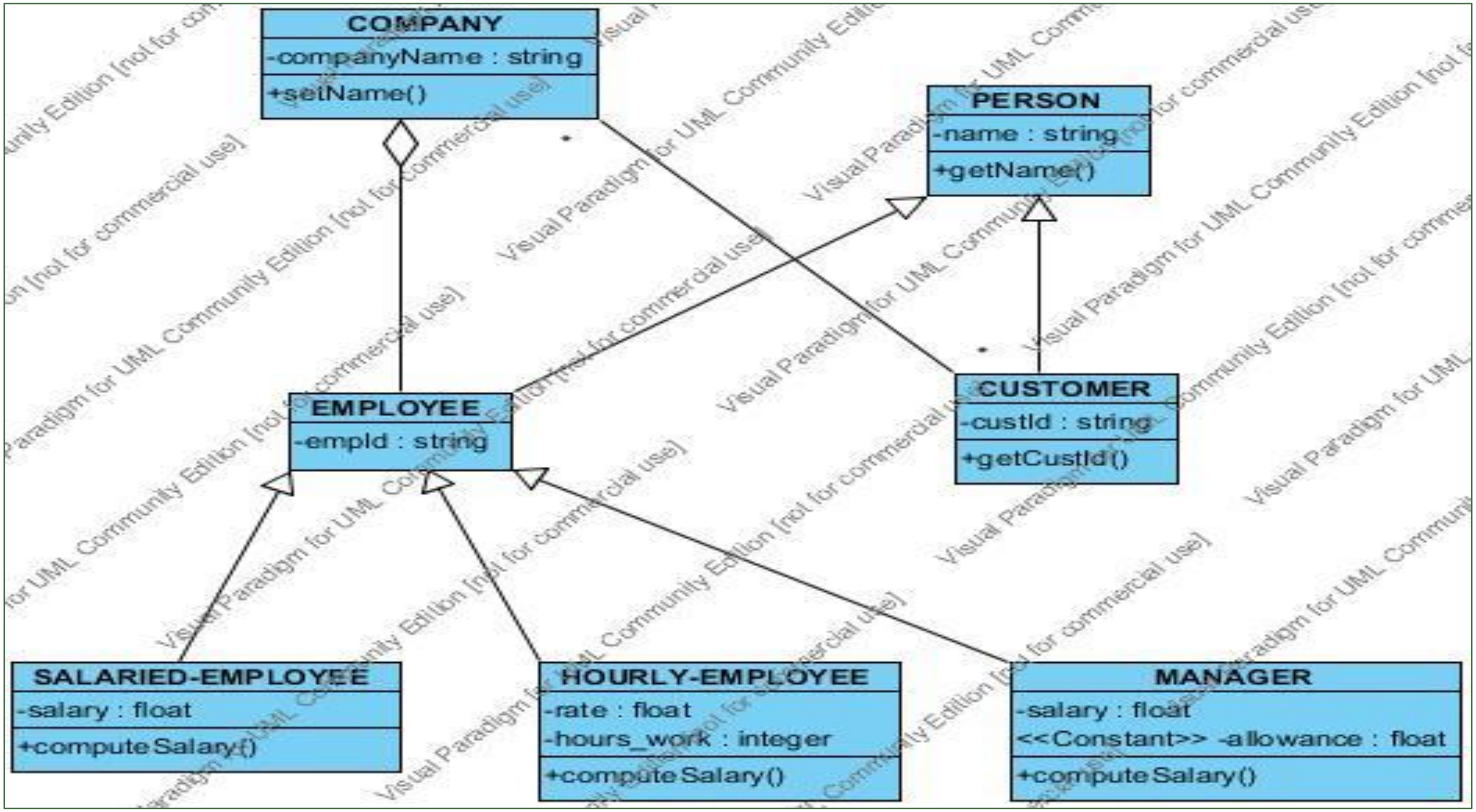
Hybrid

Sample Problem

A company has several employees and pays them on a weekly basis. The company has three types of employees: salaried-employee, who are paid a fixed weekly salary regardless of the number of hours worked; hourly-employee, who are paid by the hour and receive overtime pay; and managers who are paid a fixed salary plus allowance. The company caters to many customers.

- Create a base class named Person and allow the appropriate classes to inherit from this base class.
- Identify classes and its properties and behavior and draw the Class diagram.

Answer to Sample Problem1



COMPANY
-companyName : string
+setName()

PERSON
-name : string
+getName()

EMPLOYEE
-empId : string

CUSTOMER
-custId : string
+getCustId()

SALARIED-EMPLOYEE
-salary : float
+computeSalary()

HOURLY-EMPLOYEE
-rate : float
-hours_work : integer
+computeSalary()

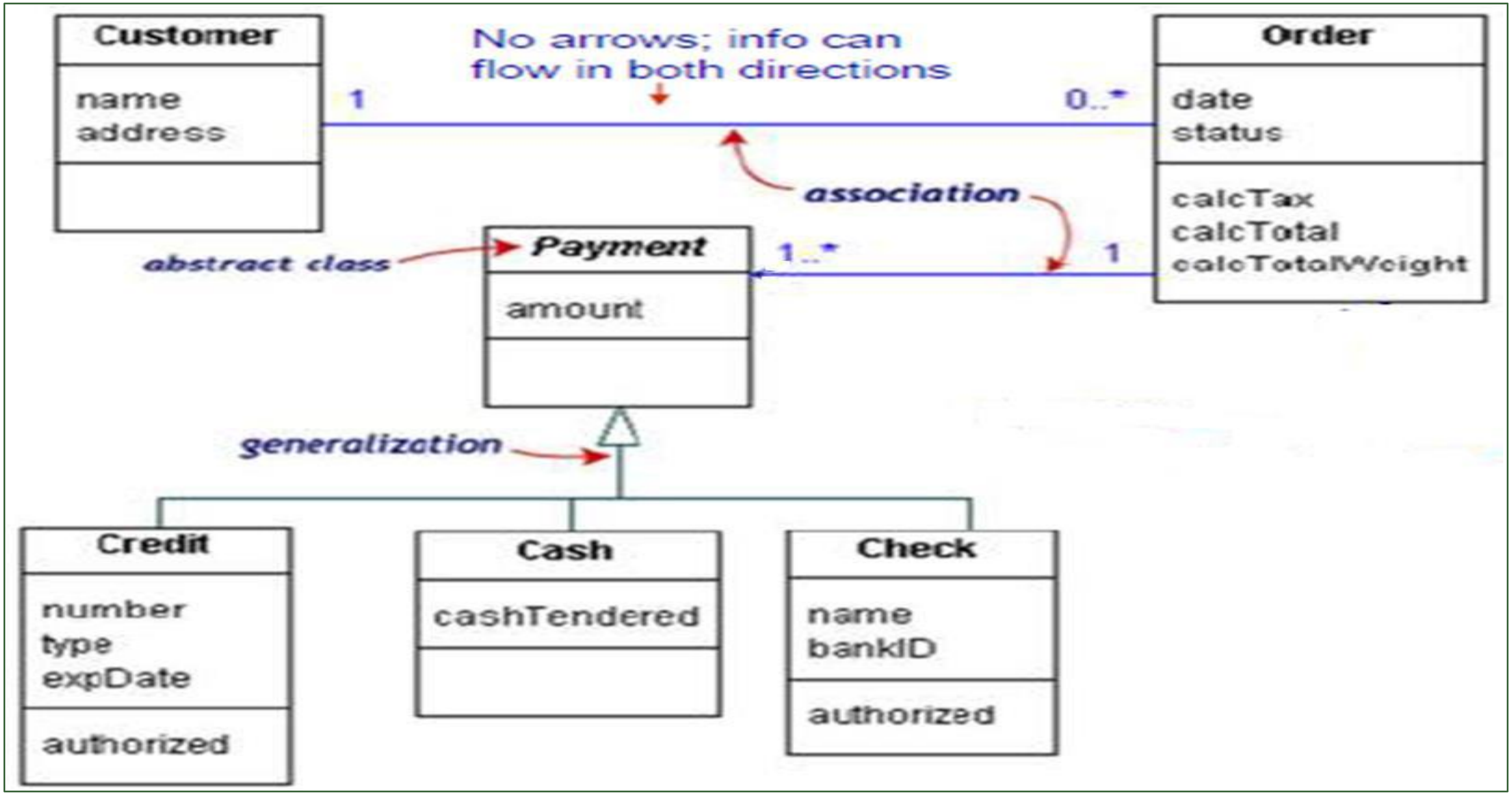
MANAGER
-salary : float
<<Constant>> -allowance : float
+computeSalary()

Sample Activity

A customer orders in a restaurant. The payment to the order will be based on the amount of the order. If the amount of the order is less than or equal to100 OMR, the customer will pay cash. If the amount is more than 100 OMR but less than or equal to 300 OMR, the customer will pay using credit card (VISA). If the amount is more than 300 OMR, the customer will issue a check

- Create a base class named Order and allow the appropriate classes to inherit from this base class.
- Identify classes and its properties and behavior and draw the Class diagram.

Answer to Sample Activity

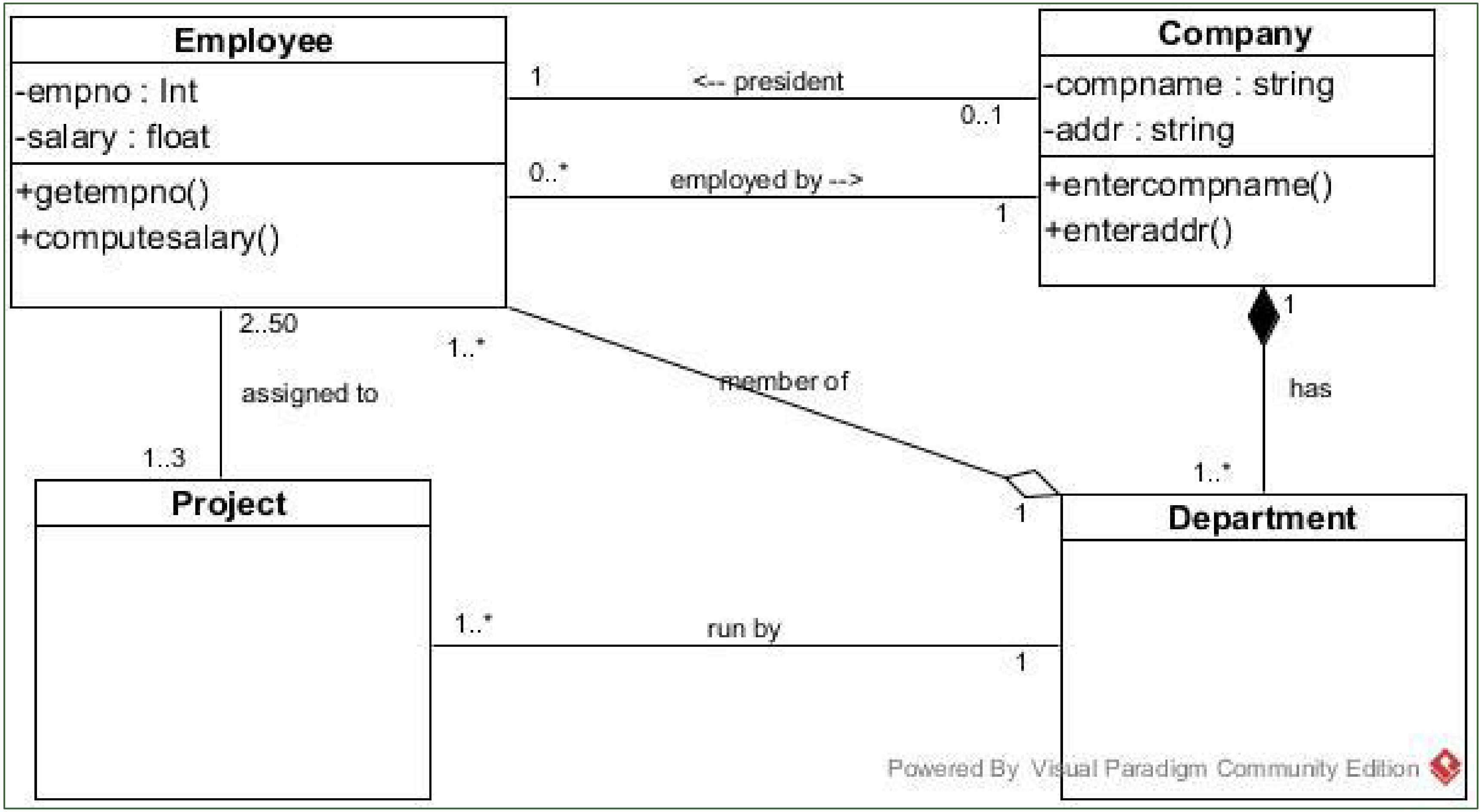


Sample Activity

Companies employ employees (who can only work for one company) and consist of one or more departments. Each company has a single president, who is an employee. Departments have employees as members and run projects (one or more.) Employees can work in 1 to 3 projects, while a project can have 2 to 50 assigned employees.
You may assume that companies have a name and address, while employees have an employee number and salary

- Identify classes and its properties and behavior and draw the Class diagram.

Answer to Sample Activity



End Of Presentation

Thank You!