THE SOFTWARE PROCESS

A structured set of activities required to develop a software system.

Many different software processes but all involve:

- **Specification** – defining what the system should do;
- **Design and implementation (Development)** – defining the organization of the system and implementing the system;
- **Validation (Testing)** – checking that it does what the customer wants;
- **Maintenance & Evolution** – changing the system in response to changing customer needs.

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

MODELLING

A model is an abstraction of a system.

Abstraction allows us to ignore unessential details

Why building models?

- To reduce complexity
- To test the system before building it
- To communicate with the customer
- To document and visualize your ideas

SOME UML DIAGRAMS

Functional diagrams

- Describe the functionality of the system from the user's point of view. It describes the interactions between the user and the system. It includes use case diagrams.

Static diagrams

- Describe the static structure of the system: Classes, Objects, attributes, associations.

Dynamic diagrams:

- **Interaction diagrams**
  - Describe the interaction between objects of the system
- **State diagrams**
  - Describe the temporal or behavioral aspect of an individual object
- **Activity diagrams**
  - Describe the dynamic behavior of a system, in particular the workflow.

UML TOOLS

Some UML tools can generate code once UML diagram is completed.

Some UML tools are sketching tools.

Rational Rose is one of the most popular software for UML creation (IBM).

Bouml is an open source s/w. It supports python, C++, java.

Visio is a sketching tool.

STATIC DIAGRAMS

- *Class diagrams: show the classes and their relations.*
- *Object diagrams: show objects and their relations.*
- *Package diagrams: show how the various classes are grouped into packages to simplify complex class diagrams.*

CLASS MODEL

A ***class model*** captures the static structure of the system by characterizing

- the *classes and objects* in the system,
- the *relationships* among the objects and
- the *attributes* and *operations* for each class of objects

Class models are the *most important* OO models.

In OO systems we build the system around objects not functionality.

WHAT IS OO SOFTWARE?

Object-oriented software means that we organize software as a collection of discrete objects that incorporate both data structure and behavior.

The fundamental unit is the ***Object***.

An object has ***state (data)*** and ***behavior (operations)***.

In ***Structured programming***, data and operations on the data were separated or loosely related.

OBJECT ORIENTED “OO” CHARACTERISTICS

- Identity
- Classification
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Generics
- Cohesion
- Coupling

IDENTITY

An object can be ***concrete*** like a ***car***, a ***file***, …

An object can be ***conceptual*** like a ***feeling***, a ***plan***,…

Each object has its own identity even if two objects have exactly the same ***state.***

Omar’s car

Rana’s car

IDENTITY

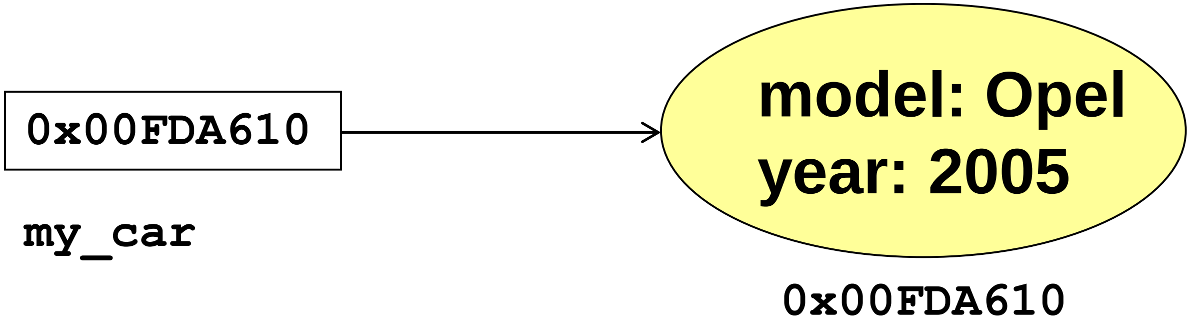
Object identity is the property by which each object can be identified and treated as a distinct software entity.

Each object has unique identity which distinguishes it from all its fellow objects. It is its memory address (or ***handle***) that is referred to by one or more ***object identifiers (variables)***.

IDENTITY

```
car  my_car = new car(“Opel”, 2005);
```

The object handle is 0x00FDA610 is referenced by an object identifier (object variable) my_car



0x00FDA610

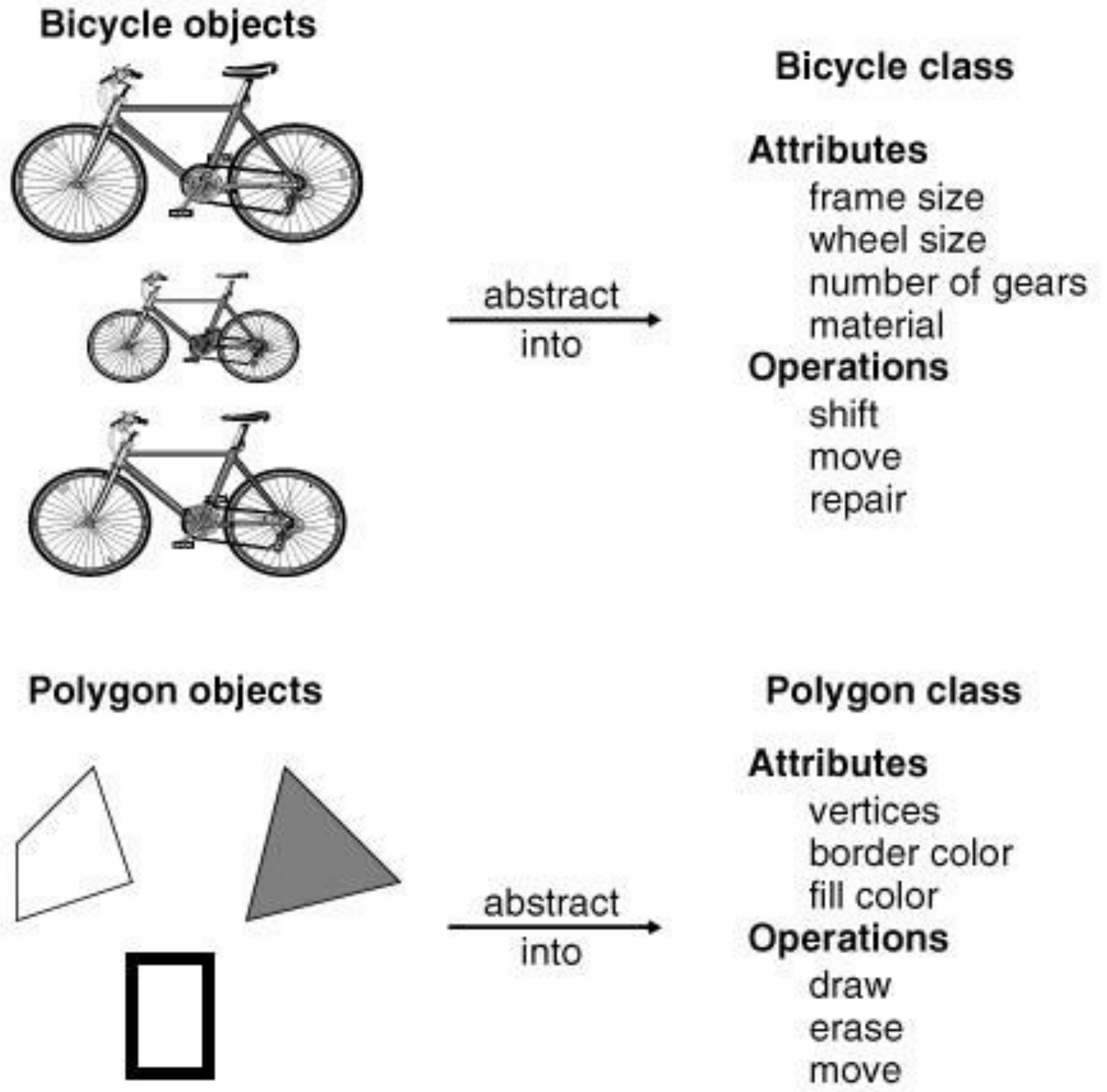
my_car

model: Opel
year: 2005

0x00FDA610

CLASSES AND OBJECTS

- Each object is an *instance* of a class
- Each object has a reference to its class (knows which class it belongs to)



Bicycle objects

Bicycle class

Attributes
frame size
wheel size
number of gears
material

Operations
shift
move
repair

Polygon objects

Polygon class

Attributes
vertices
border color
fill color

Operations
draw
erase
move

Figure 1.2 Objects and classes. Each class describes a possibly infinite set of individual objects.

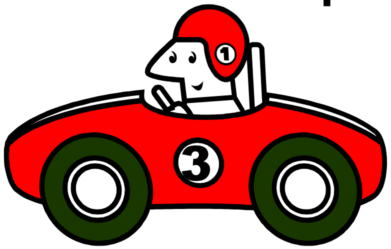
CLASSIFICATION

Classification means that objects with the same data structure (***attributes***) and behavior (***operations***) belong to the same ***class***.

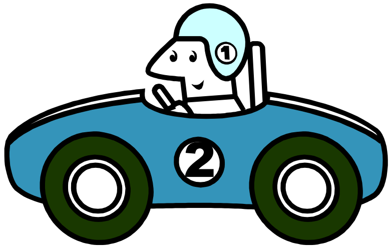
A class is an ***abstraction*** that describes the properties important for an application

The choice of classes is arbitrary and application-dependent.

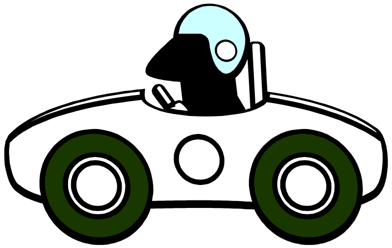
Mina’s car



Ali’s car

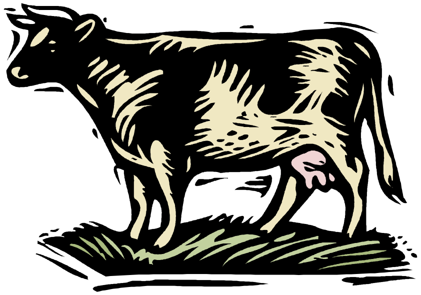


Samir’s car



A Car

CLASSIFICATION



Objects in a class share a common ***semantic*** purpose in the system model.

Both car and cow have *price* and *age*.

If both were modeled as pure *financial assets*, they both can belong to the same class.

If the application needs to consider that:

- Cow eats and produces milk
- Car has speed, make, manufacturer, etc.

then model them using separate classes.

So the semantics depends on the application

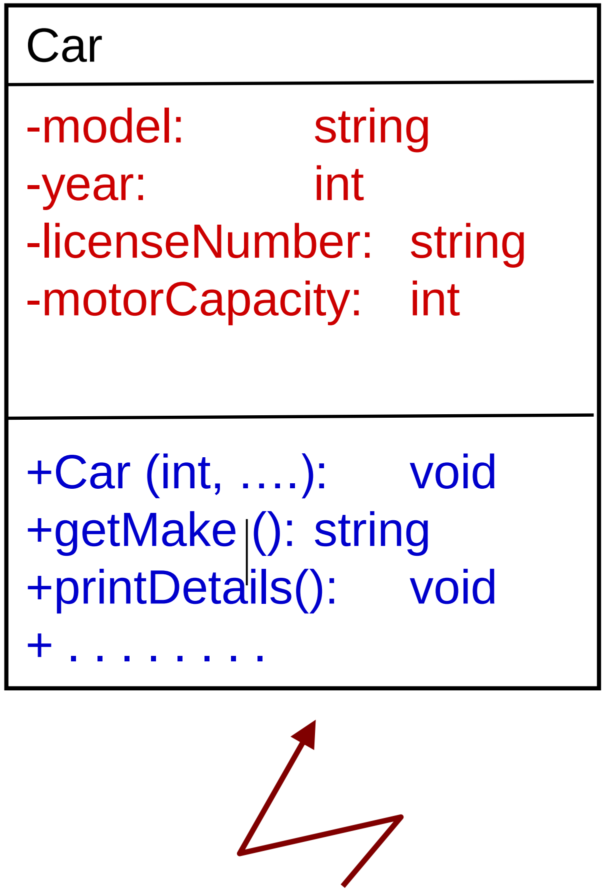
ABSTRACTION

Abstraction is the selective examination of certain aspects of a problem.

Abstraction aims to isolate the aspects that are important for some purpose and suppress the unimportant aspects.

The purpose of abstraction determines what is important and what is not.

ABSTRACTION

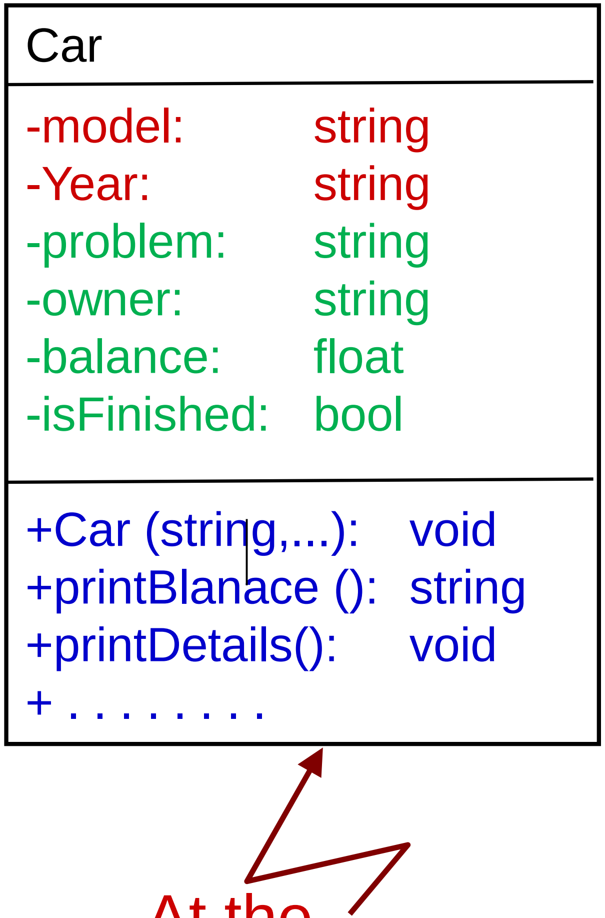


Car

-model: string
-year: int
-licenseNumber: string
-motorCapacity: int

+Car (int, ….): void
+getMake (): string
+printDetails(): void
+ . . . .

In Department of Motor Vehicles



Car

-model: string
-Year: string
-problem: string
-owner: string
-balance: float
-isFinished: bool

+Car (string,...): void
+printBlanace (): string
+printDetails(): void
+ . . . .

At the mechanic

ENCAPSULATION

Encapsulation separates the external aspects of an object, that are accessible to other objects, from the internal implementation details that are hidden from other objects.

Encapsulation reduces ***interdependency*** between different parts of the program.

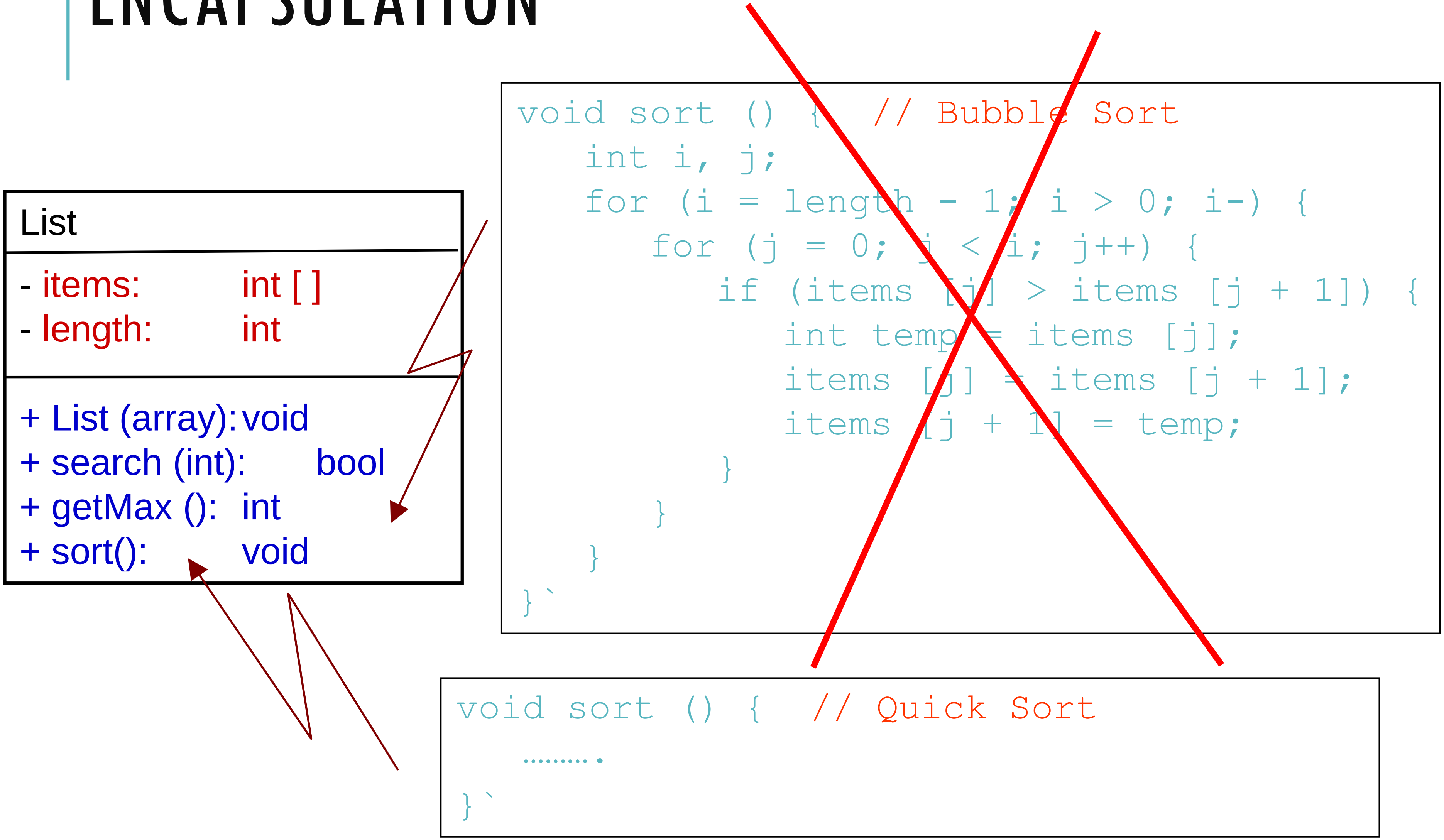
You can *change the implementation* of a class (to enhance performance, fix bugs, etc) *without affecting* the applications that use objects of this class.

ENCAPSULATION

Data hiding. information from within the object cannot be seen outside the object.

Implementation hiding. implementation details within the object cannot be seen from the outside.

ENCAPSULATION



List

- items: int [ ]
- length: int

+ List (array): void
+ search (int): bool
+ getMax (): int
+ sort(): void

```
void sort () {   // Bubble Sort
    int i, j;
    for (i = length - 1; i > 0; i-) {
        for (j = 0; j < i; j++) {
            if (items [j] > items [j + 1]) {
                int temp = items [j];
                items [j] = items [j + 1];
                items [j + 1] = temp;
            }
        }
    }
} `
```

```
void sort () {   // Quick Sort
    ……….
} `
```

INHERITANCE

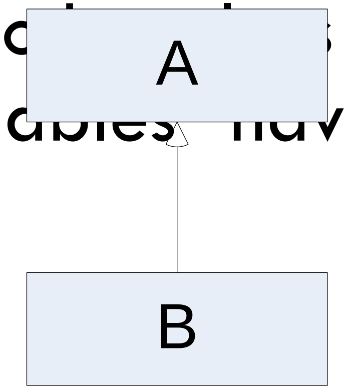
Inheritance is the sharing of ***features*** (attributes and operations) among classes based on a hierarchical relationship.

A ***superclass*** (also ***parent*** or ***base***) has general features that ***sublcasses*** (***child*** or ***derived***) inherit. and may refine some of them.

Inheritance is one of the strongest features of OO technology.

INHERITANCE

Inheritance is the facility by which objects of a class (say B) may use the methods and variables that are defined only to objects of another class (say A), as if these methods and variables have been defined in class B



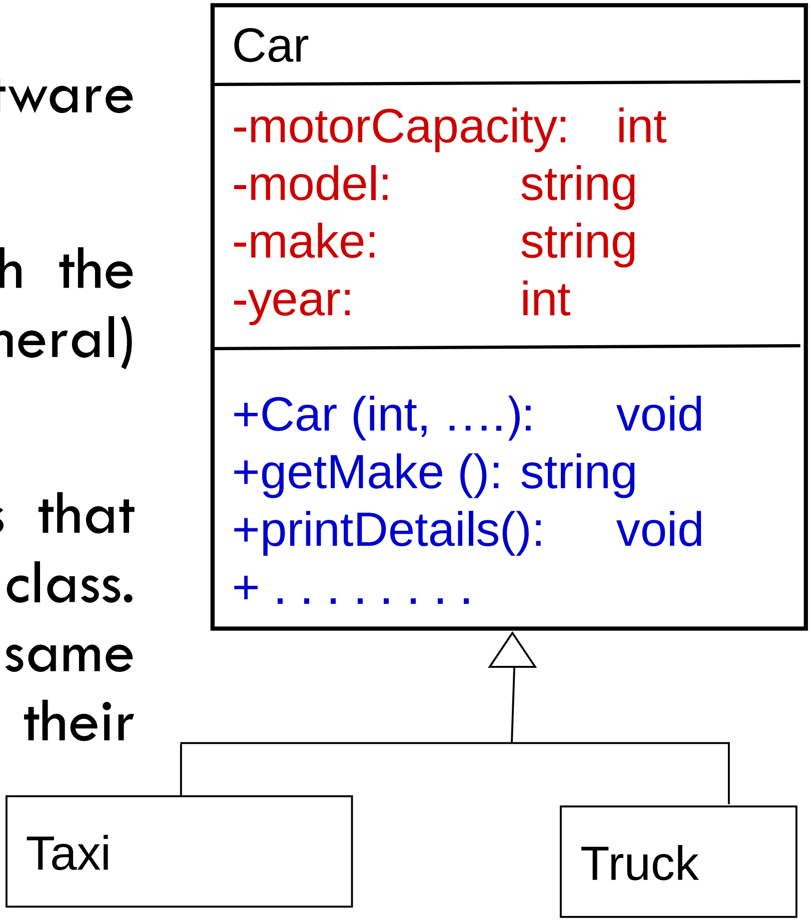
Inheritance is represented as shown in ***UML*** notation.

HOW TO USE INHERITANCE?

Inheritance helps building software incrementally:

First; build classes to cope with the most straightforward (or general) case,

Second; build the special cases that inherit from the general base class. These new classes will have the same features of the base class plus their own.



Car

-motorCapacity: int
-model: string
-make: string
-year: int

+Car (int, ….): void
+getMake (): string
+printDetails(): void
+ . . . .

Taxi

Truck

POLYMORPHISM

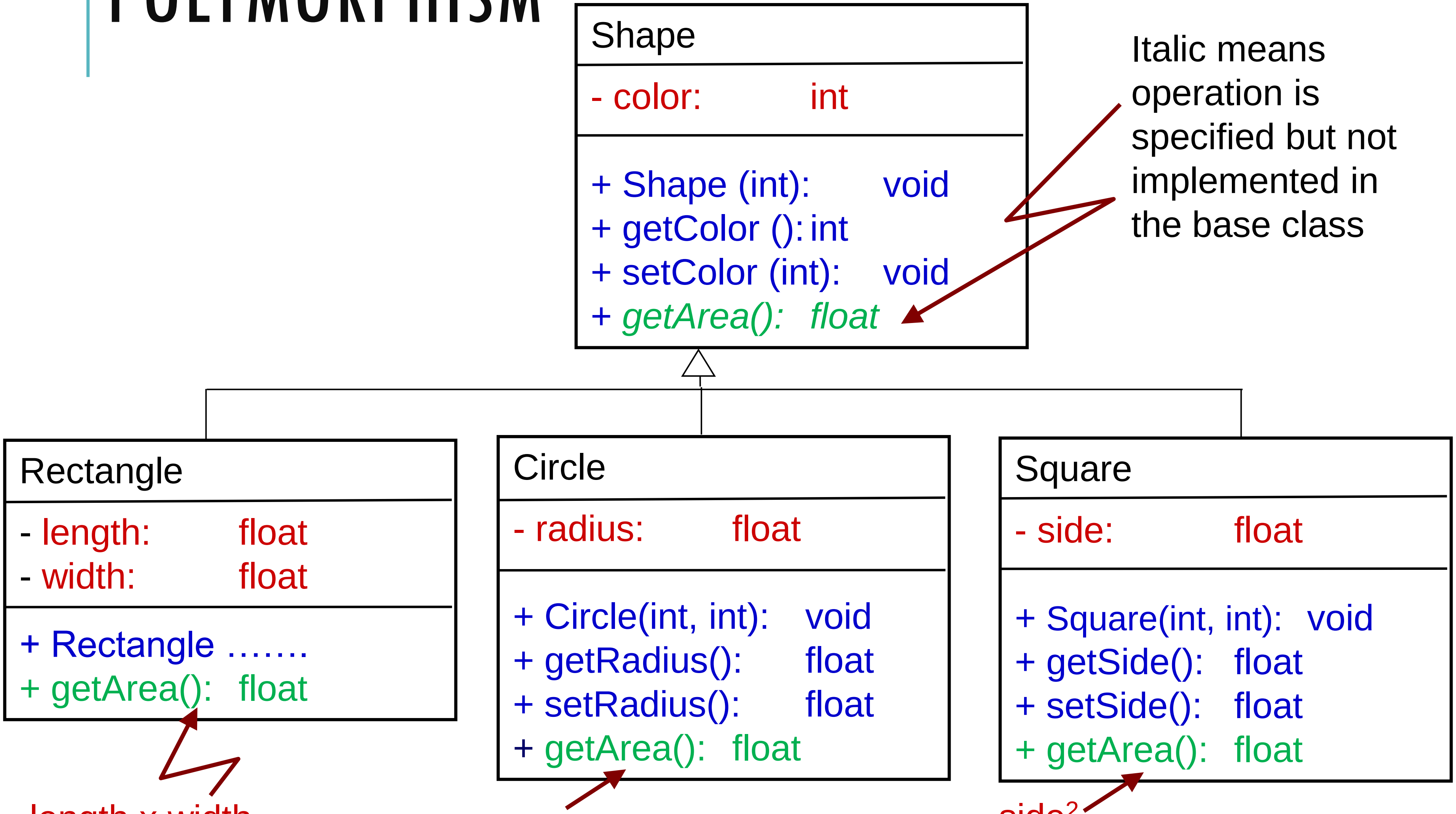
Polymorphism means that the same operation may behave differently for different classes.

An ***operation*** is a procedure or a transformation that the object performs or is subject to.

An implementation of an operation by a specific class is called a ***method***.

Because an OO operation is polymorphic, it may have more than one method for implementing it, each for a different class.

POLYMORPHISM



Shape

- color: int

+ Shape (int): void
+ getColor (): int
+ setColor (int): void
+ *getArea(): float*

Italic means operation is specified but not implemented in the base class

Rectangle

- length: float
- width: float

+ Rectangle …….
+ getArea(): float

Circle

- radius: float

+ Circle(int, int): void
+ getRadius(): float
+ setRadius(): float
+ getArea(): float

Square

- side: float

+ Square(int, int): void
+ getSide(): float
+ setSide(): float
+ getArea(): float

length x width

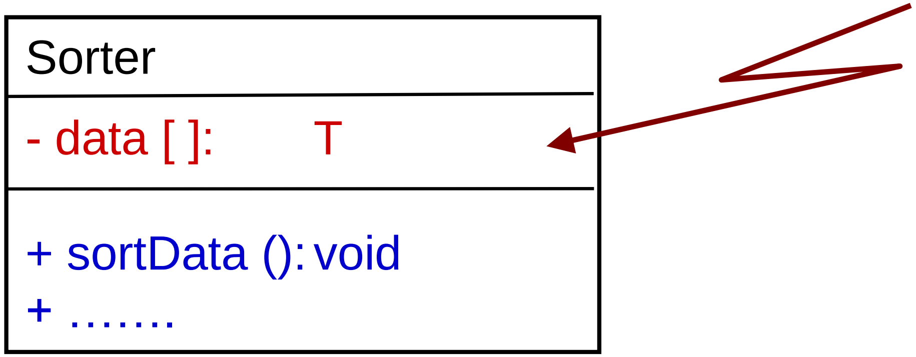
Π x radius^2

side^2

GENERICS

Generic Class ; the data types of one or more of its attributes are supplied at run-time (at the time that an object of the class is instantiated).

This means that the class is ***parameterized***, i.e., the class gets a parameter which is the name of another type.



Sorter

- data [ ]: T

+ sortData (): void
+ …….

T will be known at runtime ..

CLASS MODEL

A ***class model*** captures the static structure of the system by characterizing

- the *classes and objects* in the system,
- the *relationships* among the objects and
- the *attributes* and *operations* for each class of objects

Class models are the *most important* OO models

In OO systems we build the system around objects not functionality

OBJECTS

Objects often appear as ***proper nouns*** in the problem description or discussion with the customer.

Some object correspond to ***real world entities*** (BUE, MIT, Omar's car)

Some objects correspond to ***conceptual entities*** (the formula for solving an equation, binary tree, etc.)

The choice of objects depends on the analyst's judgment and the problem in hand. There can be *more than one correct representation*.

CLASS

An object is an ***instance of*** a class

A class describes a group of objects with the same

- Properties (attributes)
- Behavior (operations)
- Kinds of relationships

Person, *Company and Window* are all classes

Classes often appear as ***common nouns*** and ***noun phrases*** in problem description and discussion with customers or users

CLASS MODEL

Provides a graphical notation for modeling classes and their relationships, thereby describing possible objects.

Class diagram is useful for:

- *Designing and Implementing the programs.*

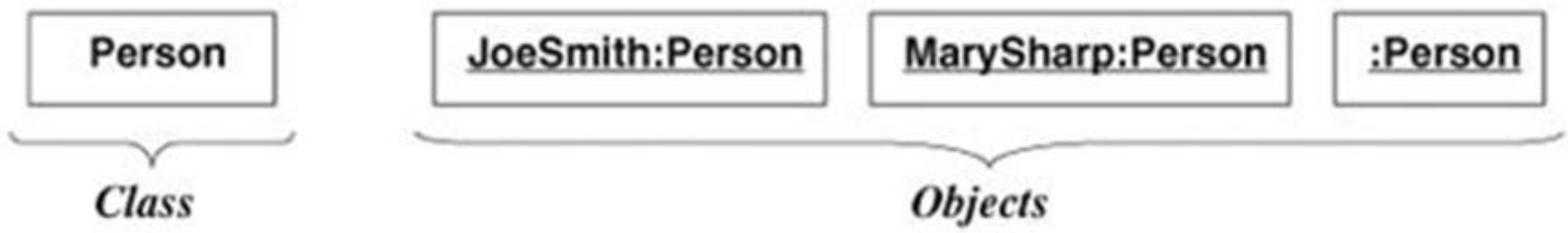
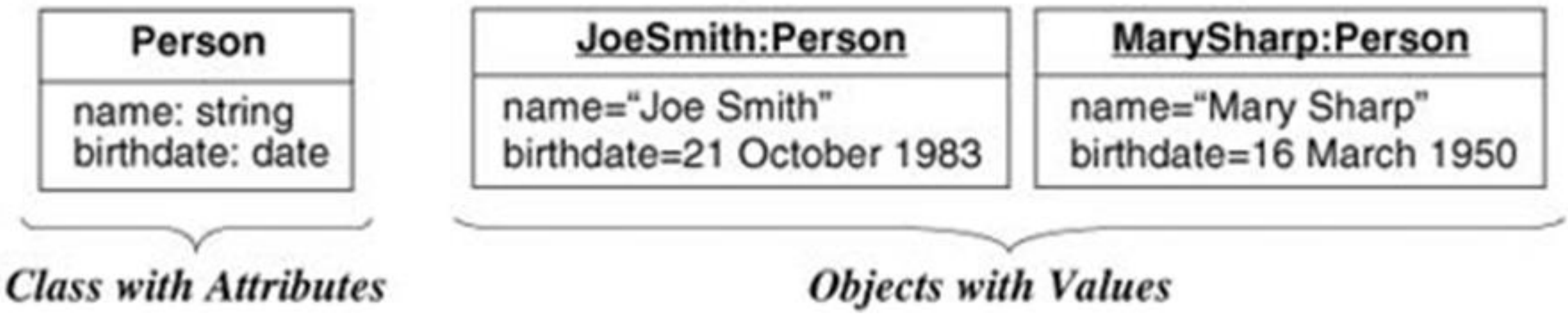


Figure 3.1 A class and objects. Objects and classes are the focus of class modeling.

VALUES AND ATTRIBUTES



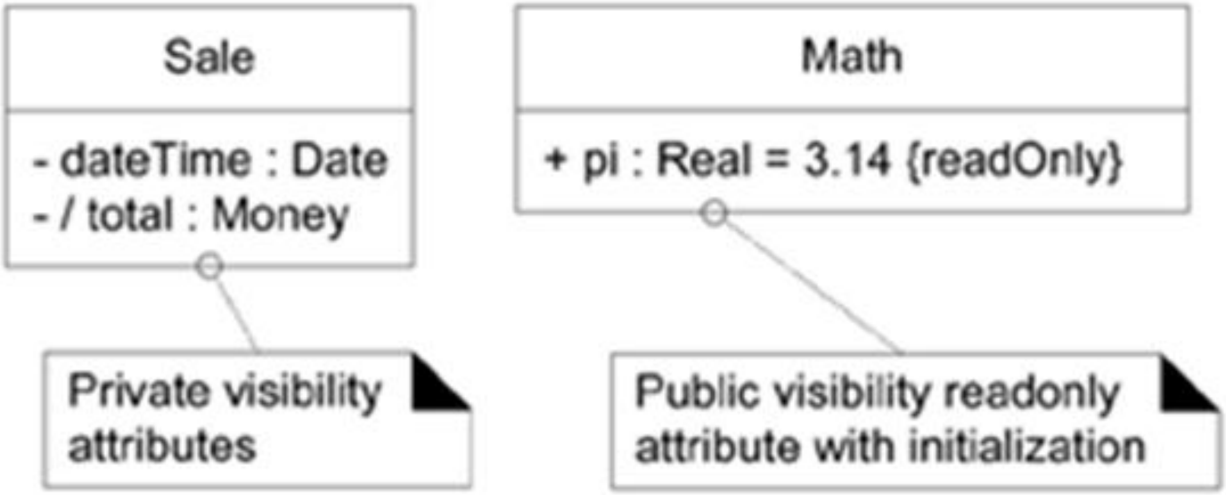
An ***attribute*** is a named property of class that describes a value held by each object of that class.

Attribute name is unique per class.

Several classes may have the same attribute name.

A ***value*** is a piece of data assigned to an attribute.

MORE ON ATTRIBUTES



Sale

- dateTime : Date
- / total : Money

Private visibility attributes

Math

+ pi : Real = 3.14 {readOnly}

Public visibility readonly attribute with initialization

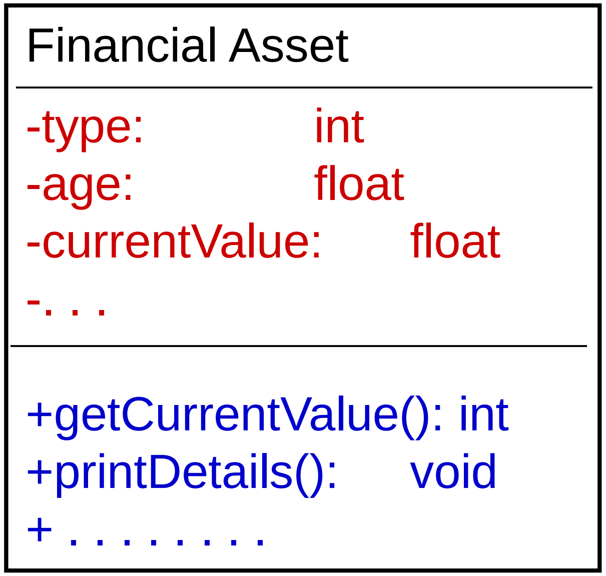
OPERATIONS AND METHODS

An ***operation*** is a function or procedure that may be applied *to* or *by* objects of a class.

A ***method*** is the implementation of an operation for a class.

An operation is ***polymorphic*** if it takes different forms in different classes.

All objects of the same class have the same operations.



Financial Asset

-type: int
-age: float
-currentValue: float
-. . .

+getCurrentValue(): int
+printDetails(): void
+ . . . .

SUMMARY OF CLASS NOTATION

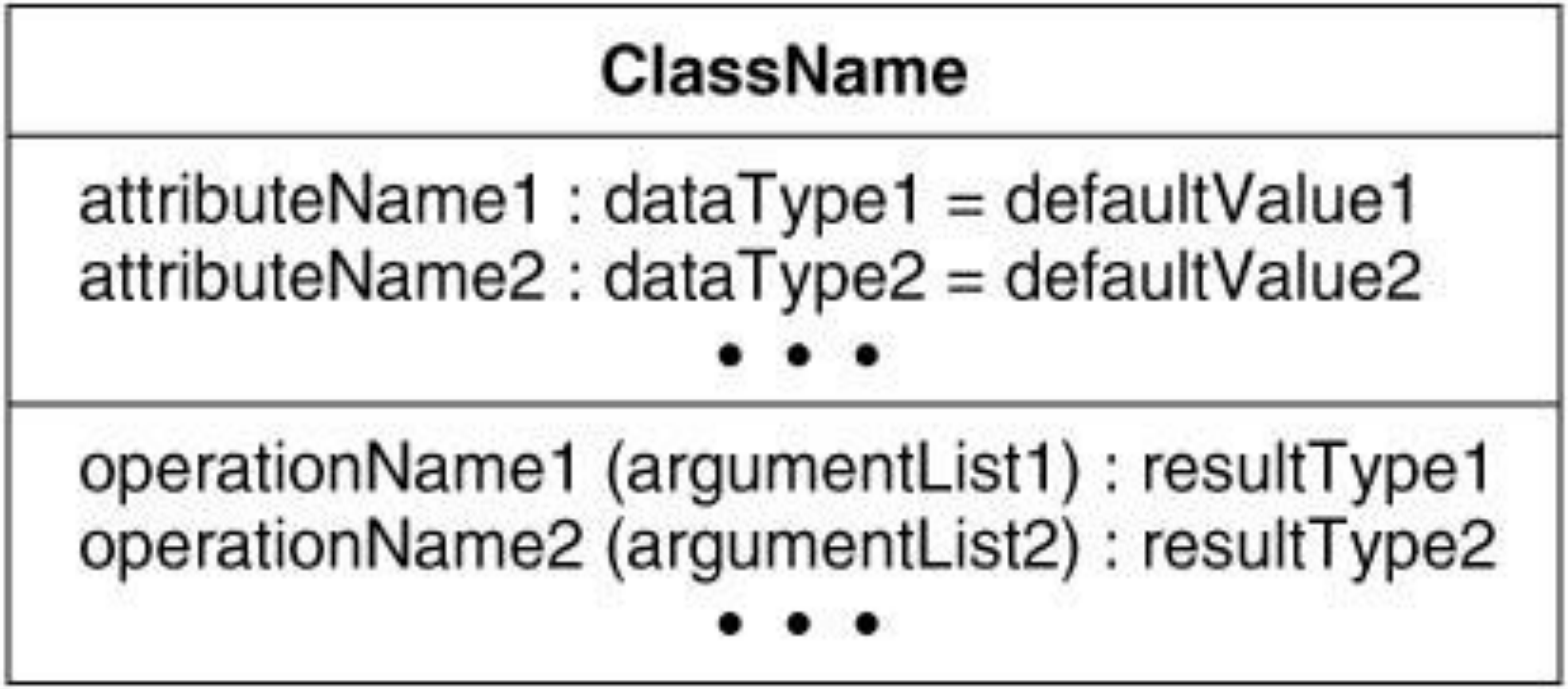
The attribute and operation compartments are optional.

You may show them or not depending on the level of abstraction you want.

A missing attribute compartments means that the attributes are not specified yet.

But empty compartment means that the attributes are specified but there are none.

SUMMARY OF BASIC CLASS NOTATION



ClassName

attributeName1 : dataType1 = defaultValue1
attributeName2 : dataType2 = defaultValue2
• • •

operationName1 (argumentList1) : resultType1
operationName2 (argumentList2) : resultType2
• • •

Figure 3.5 Summary of modeling notation for classes. A box represents a class and may have as many as three compartments.

A SAMPLE CLASS MODEL

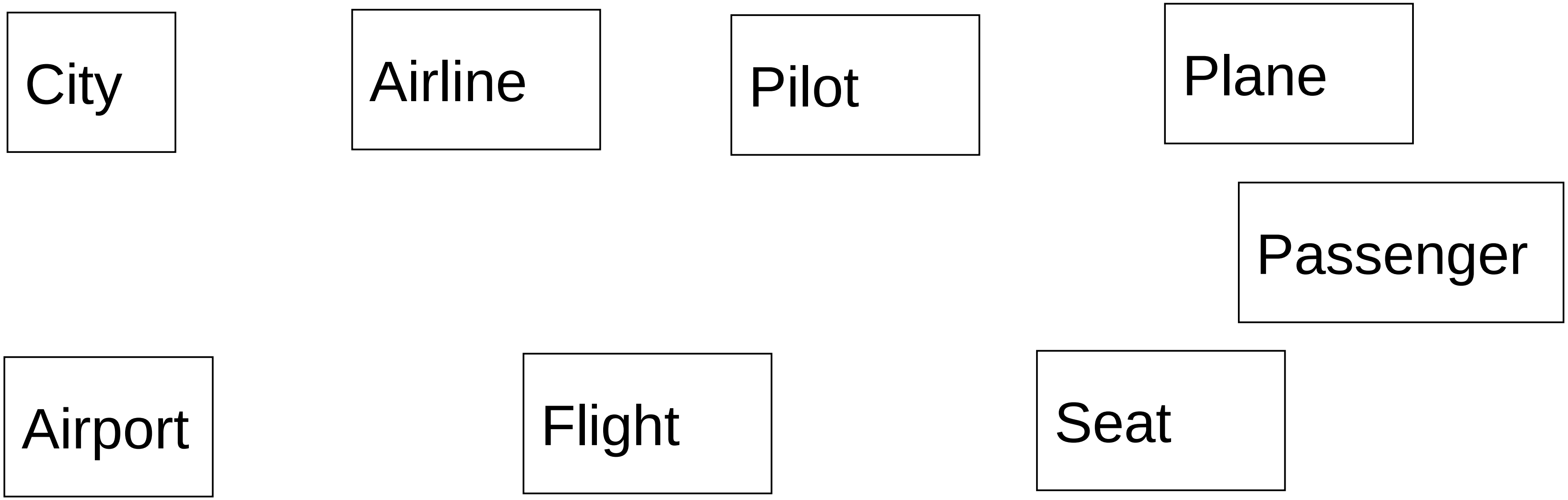
Model the classes in a system that represents flights. Each city has at least an airport. Airlines operate flights from and to various airports. A flight has a list of passengers, each with a designated seat. Also a flight uses one of the planes owned by the operating airline. Finally a flight is run by a pilot and a co-pilot.

A SAMPLE CLASS MODEL

Model the classes in a system that represents ***flights***. Each ***city*** has at least an ***airport***. ***Airlines*** operate flights from and to various airports. A flight has a ***list*** of ***passengers***, each with a designated ***seat***. Also a flight uses one of the ***planes*** owned by the operating airline. Finally a flight is run by a ***pilot*** and a ***co-pilot***.

A SAMPLE CLASS MODEL

- ***Flights***
- ***List*** of ***Passengers***
- ***Pilot*** and a ***Co-Pilot***
- ***City***
- ***Seat***
- ***Airlines***
- ***Planes***


City

Airline

Pilot

Plane

Passenger

Airport

Flight

Seat

THANKS! .. QUESTIONS?