Object Oriented Programming Concept

Focus of this subject

- This subject is not only about Java
- If you know a programming language and you know how to use computers, it does not mean that you can develop software systems

If you know alphabet and you can type quickly, this may not be enough to write a novel

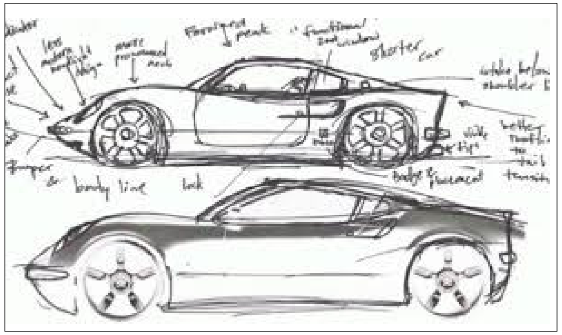
To become a programmer

- you need to be familiar with advanced software development concepts
- you need to know how to efficiently utilize these concepts using appropriate programming languages

Software development challenge

Software development has unique challenges which are not common for other industries

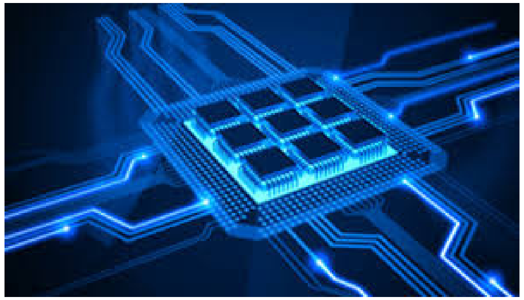
- When you design a car you can foresee how it will look like when it is made
- Reverse engineering can reproduce car design documents (identical to original ones)



However…

- When a program is executed, it is a sequence of invisible electrical signals which represent binary microprocessor instructions
- It is practically impossible to reproduce the original program code from binary instructions

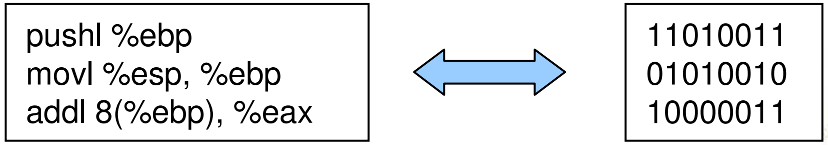
How to design programs when you can't see the final product?



Imperative Programming Concept

The oldest concept where a program consists of a sequence of commands for the microprocessor to execute

Example: Assembly language is a symbolic representation of microprocessor instructions with one-to-one mapping



```
pushl %ebp
movl %esp, %ebp
addl 8(%ebp), %eax
```

```
11010011
01010010
10000011
```

How to increase productivity of software development?

- When you virtually have to 'live in the microprocessor' and think like a machine to write even a simple program, you cannot be productive
- It is close to impossible to reuse assembly code. You need to start every new project practically from scratch

How to find an efficient design methodology ?

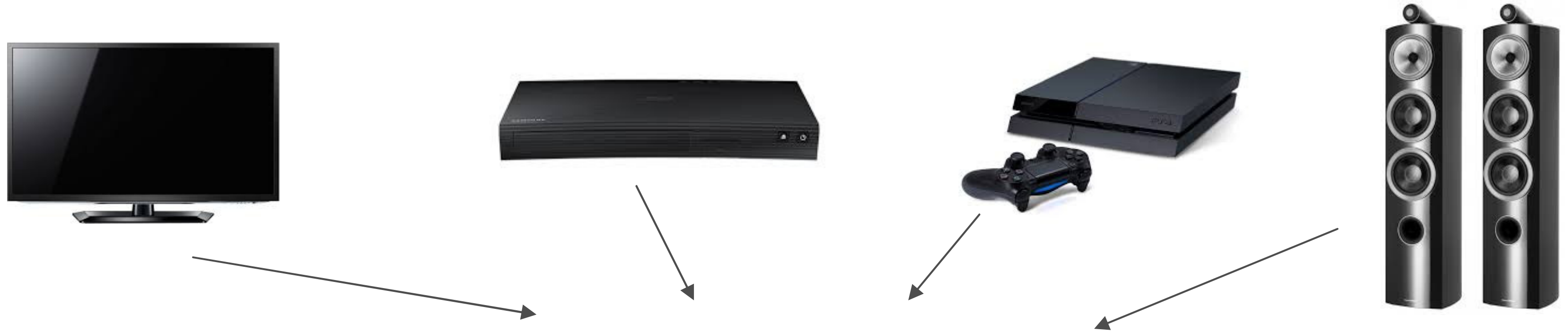
We could try to borrow ideas from other industries

Can software be built from pre-designed pre-tested parts too?

- Cars are made of preassembled parts
- To build a new car you don't need to design every part from scratch – some parts can be reused from previous models, or be purchased from new suppliers without making any changes in their internal structure

How to find an efficient design methodology ?

We could try to borrow ideas from other industries



Can software modules be designed with clearly defined interface to simplify their integration?

- To assemble a home theatre you can buy modules from different manufacturers according to your needs
- They can be easily plugged together as their interfaces are standardized

How to find an efficient design methodology ?

Other examples:

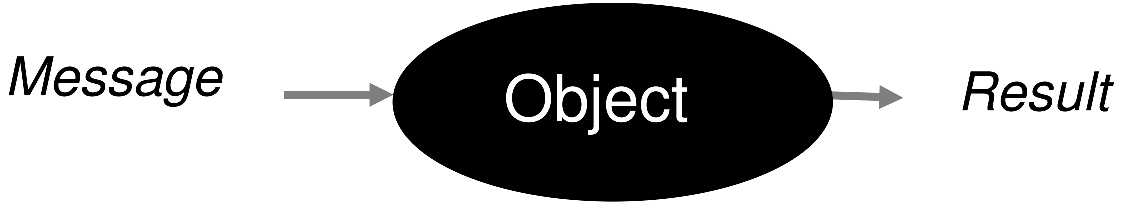
From transistors to integrated circuits

From bricks to pre-fabricated panels

- Transition to pre-fabricated generic reusable modules with well defined interface components has been a common trend for all industries
- To increase productivity, computer programming had to follow the same trend
- Research on efficient programming concepts led to development of the **Object Oriented Design** methodology in the late 60th

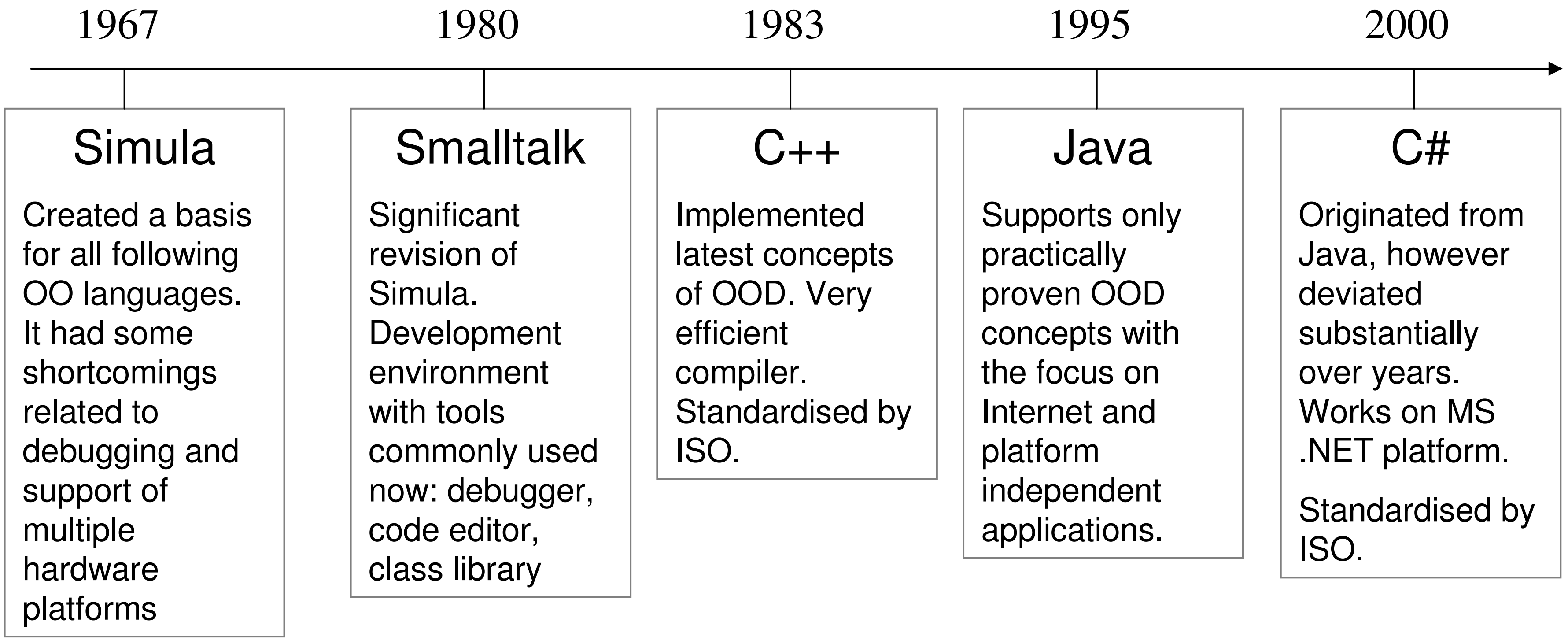
Objects

- Object Oriented Design methodology introduces the concept of objects
- Like mechanical and electrical parts in other industries, objects become major software building blocks
- Rather than think about commands and microprocessor instructions, you need to think about objects, their properties, their behaviors and how their interaction can implement your program functionality
- In general, objects can receive messages from other objects and provide responses. To use them, you don't need to worry about their internal structure. It is hidden.



Evolution of OOD languages

- The OOD methodology has substantially evolved since it was introduced in the 60th
- Evolution of OOD programming languages reflects the evolution of OOD programming concepts



1967

Simula

Created a basis for all following OO languages. It had some shortcomings related to debugging and support of multiple hardware platforms

1980

Smalltalk

Significant revision of Simula. Development environment with tools commonly used now: debugger, code editor, class library

1983

C++

Implemented latest concepts of OOD. Very efficient compiler. Standardised by ISO.

1995

Java

Supports only practically proven OOD concepts with the focus on Internet and platform independent applications.

2000

C#

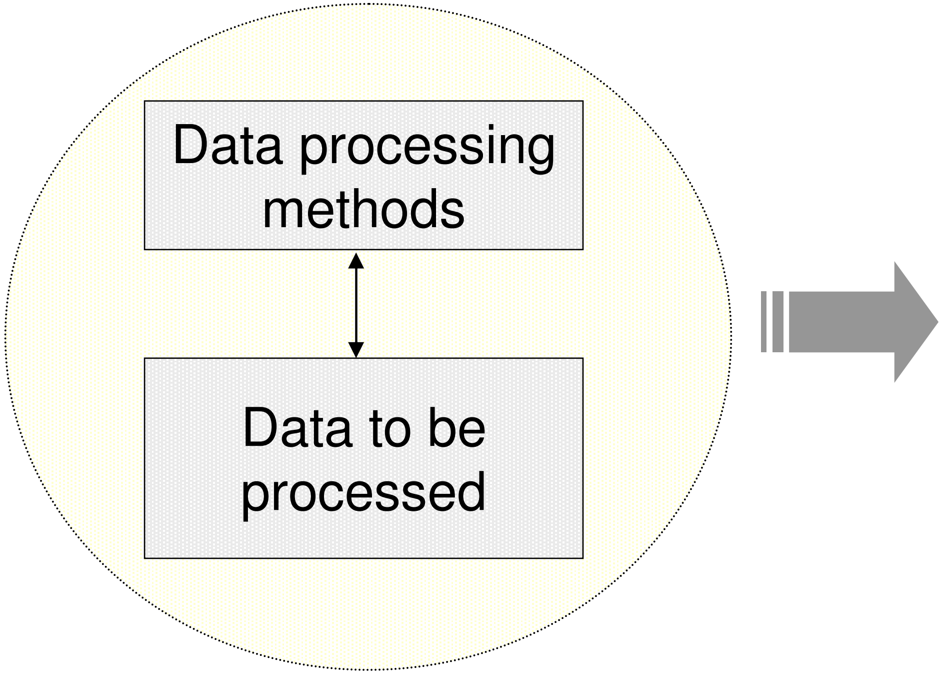
Originated from Java, however deviated substantially over years. Works on MS .NET platform.

Standardised by ISO.

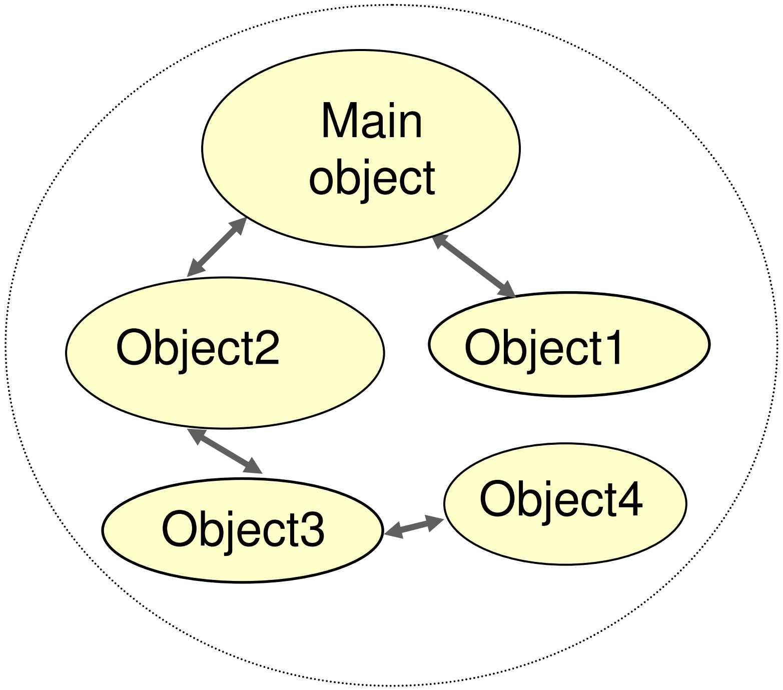
Object Based Design

- The main purpose of any program is to process data
- The major task of OOD is how to split system's data and behaviour into a set of interacting objects

A functional view of the program

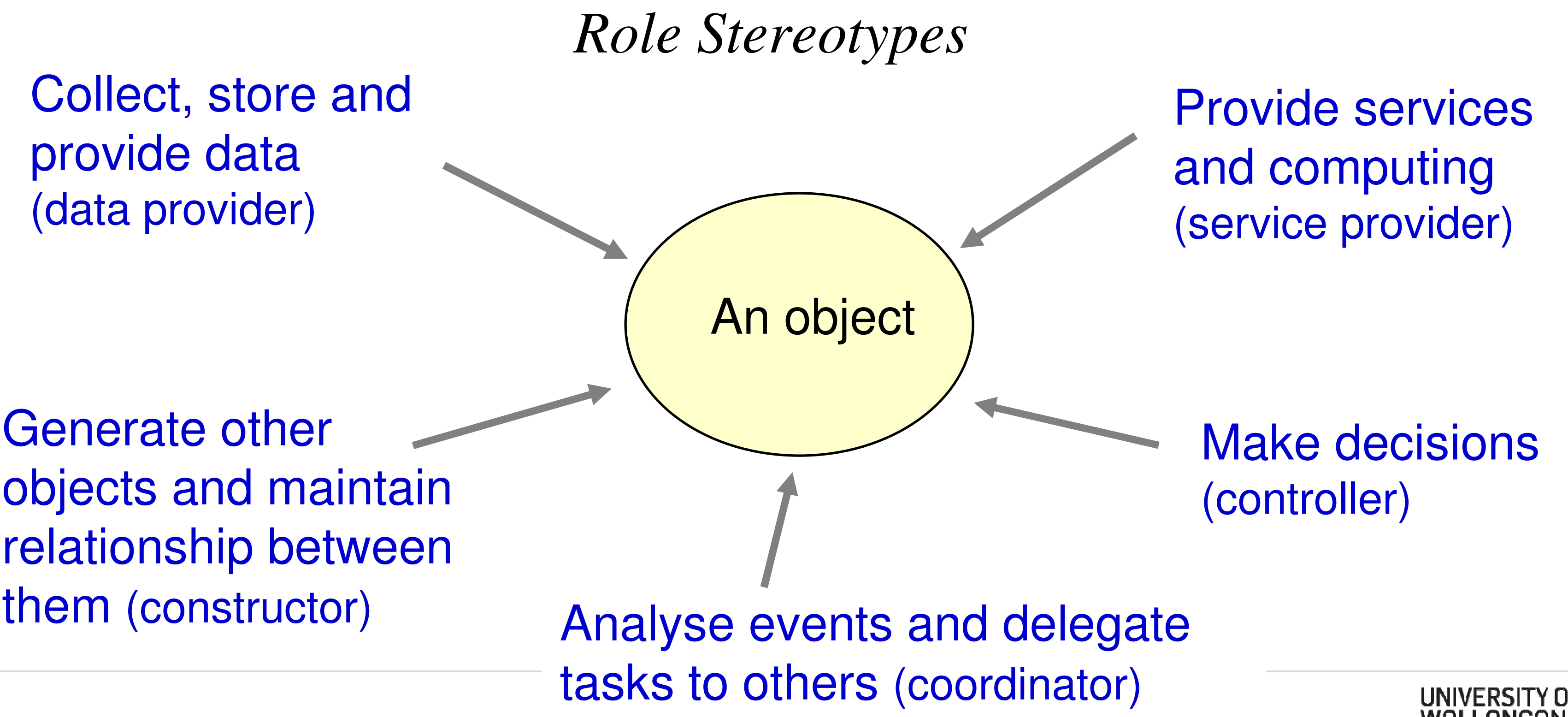


An OOD view of the program



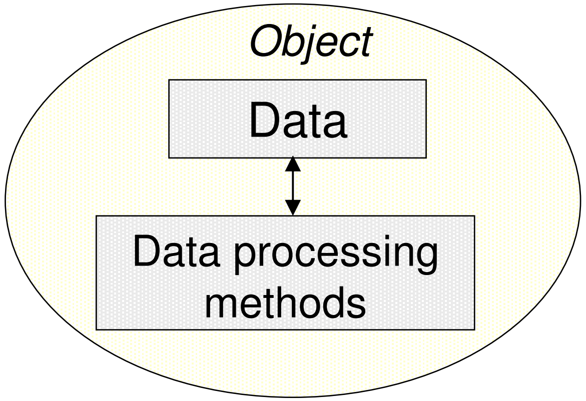
What can an object do ?

- When you describe your program as a collection of interacting objects, you need to have a clear idea what role the objects will play in your application
- An object may play just one role, or several typical roles


Role Stereotypes
Collect, store and provide data (data provider)
Provide services and computing (service provider)
An object
Generate other objects and maintain relationship between them (constructor)
Make decisions (controller)
Analyse events and delegate tasks to others (coordinator)

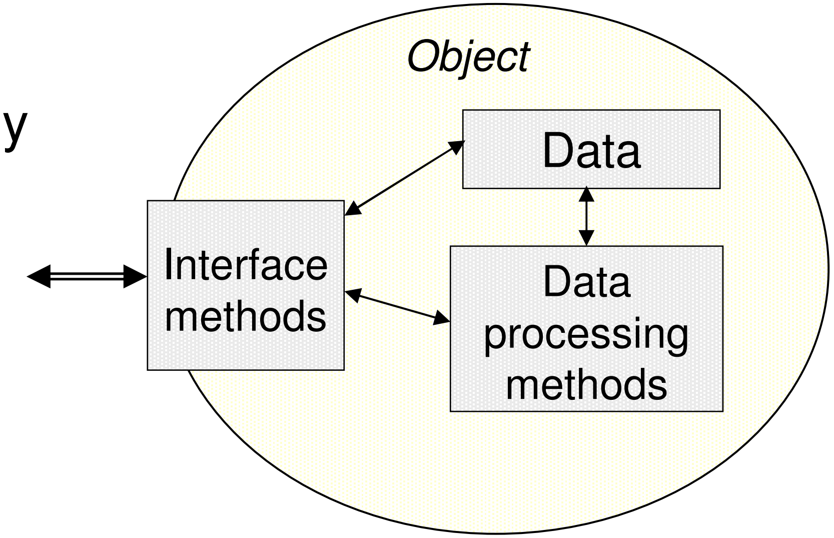
Internal structure of objects

- Each object implements a certain part of the system functionality. Thus, it should contain **data** and **methods** to process this set of data



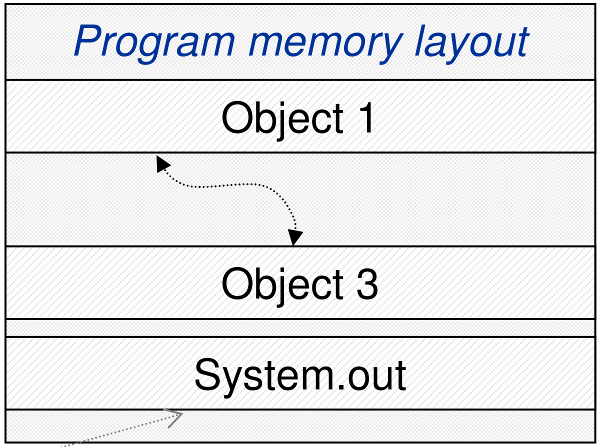
How can objects interact?

- To interact with other objects, they also include interface methods
- Interaction between objects is possible only through interface methods



What does an object look like?

- **Physically**, it consists of binary data and microprocessor instructions which occupy a reserved block of memory
  (its actual layout is system dependent)
- Links between objects can be established at the compilation time, or dynamically at run time



System.out *is the standard output object that allows Java applications to display data*

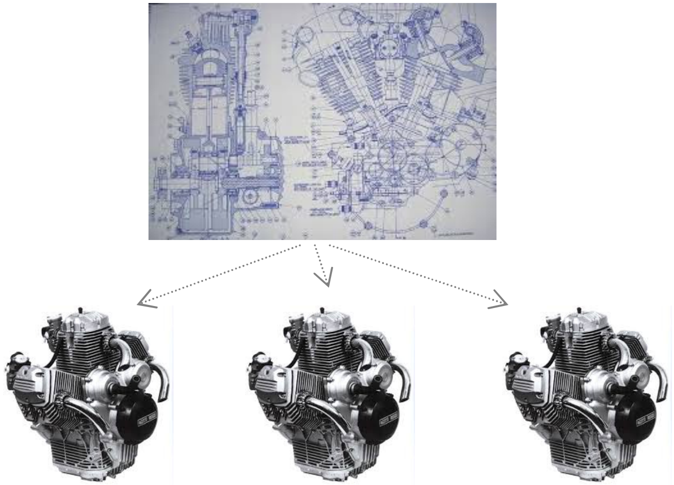
```
class HelloWorldApp {
   public static void main(String[] args) {
        // Display “Hello!"
        System.out.println(“Hello!");
   }
}
```

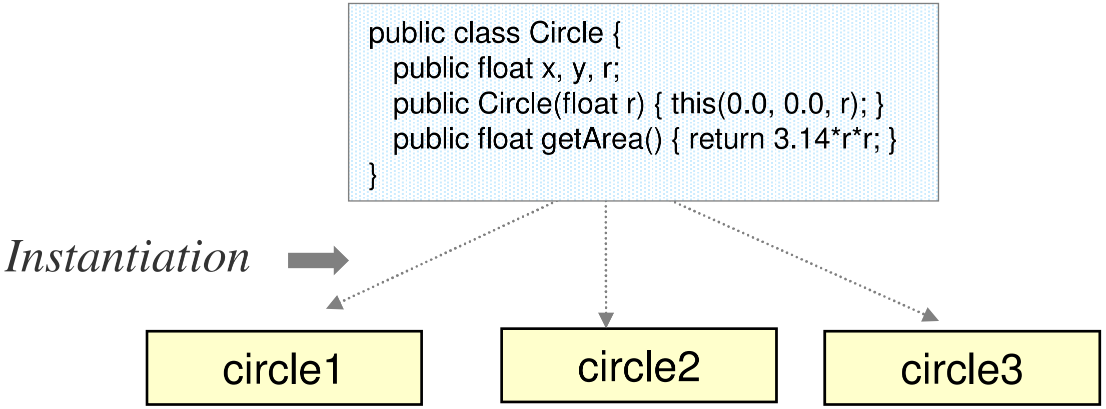
- **Logically**, it must be declared in your program
- When an objects is declared it is given a unique name
- Looking through a program code, you can see all objects declared there and follow the logical relationship between them

How can objects be created ?

Parts of mechanical systems are made based upon their description – blueprints

Objects of a software system are generated based upon their description – classes





```
public class Circle {
   public float x, y, r;
   public Circle(float r) { this(0.0, 0.0, r); }
   public float getArea() { return 3.14*r*r; }
}
```

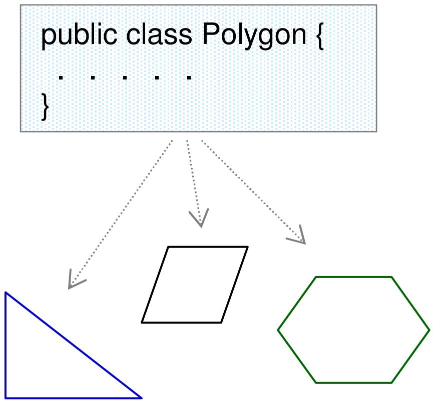
- A class is an abstraction defined by a programmer
- An object is an instance generated and placed in computer memory (many similar objects can be generated from one class)

Classes

- Classes can be interpret as blueprints, or prototypes for objects
- Objects are not necessarily clones even when they are created from the same class

For example: a class `Polygon` can be used to create triangles and rectangles as instances (objects)

- To generate different objects, we may specify different properties such as number of sides, the lengths of those sides, their colors
- The object behaviour are things which require actions - how the area, or the circumference are calculated, how the object is displayed, etc



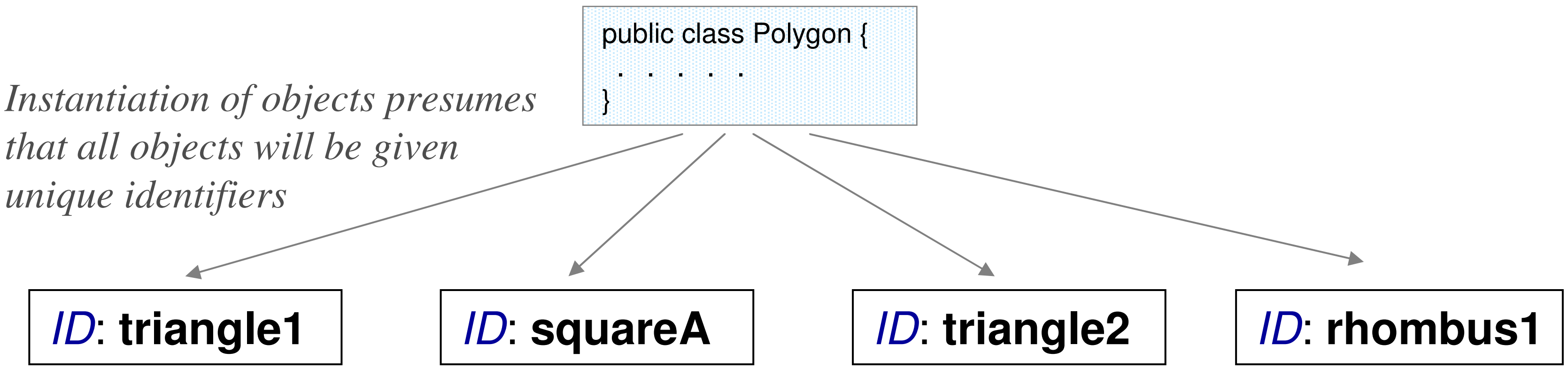
```
public class Polygon {
  . . . . .
}
```

Object intrinsic features

1. **Identity**

- Correct interaction between objects may not possible without reliable identification of objects
- Each object has a unique identifier that is used to recognize it

Instantiation of objects presumes that all objects will be given unique identifiers



```
public class Polygon {
  .  .  .  .  .
}
```

ID: **triangle1**

ID: **squareA**

ID: **triangle2**

ID: **rhombus1**

Identity (a unique name) of an object can be:

- specified by a programmer
- assigned by another object that takes a role of structurer

Object intrinsic features

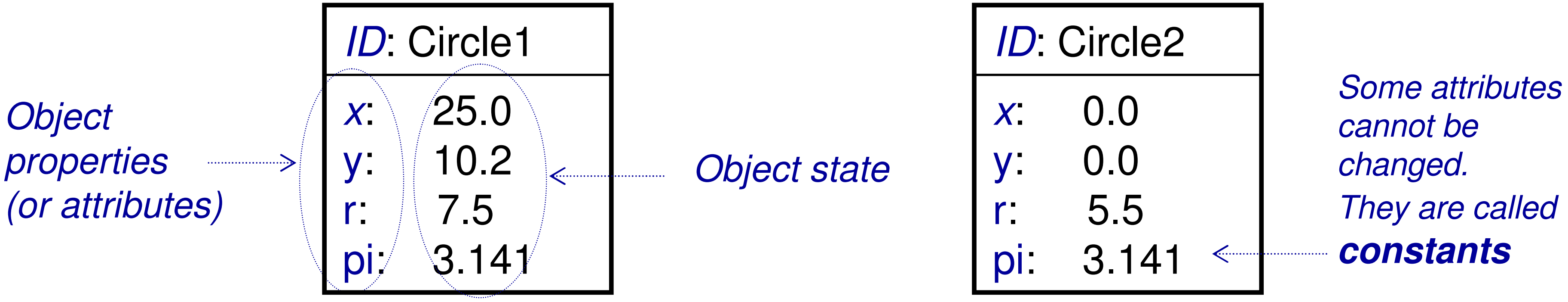
2. **State**

- Object-Oriented concept presumes that objects store data
- Unless an object is a garbage bin, all data are stored with the purpose to reflect properties (or attributes) of objects

Example properties of a circle: (x,y) coordinates of the center, r radius

properties of a bill: due date, amount to pay

- A set of actual values stored in an object is referred as an object state



Object properties (or attributes)

ID: Circle1	
x:	25.0
y:	10.2
r:	7.5
pi:	3.141

Object state

ID: Circle2	
x:	0.0
y:	0.0
r:	5.5
pi:	3.141

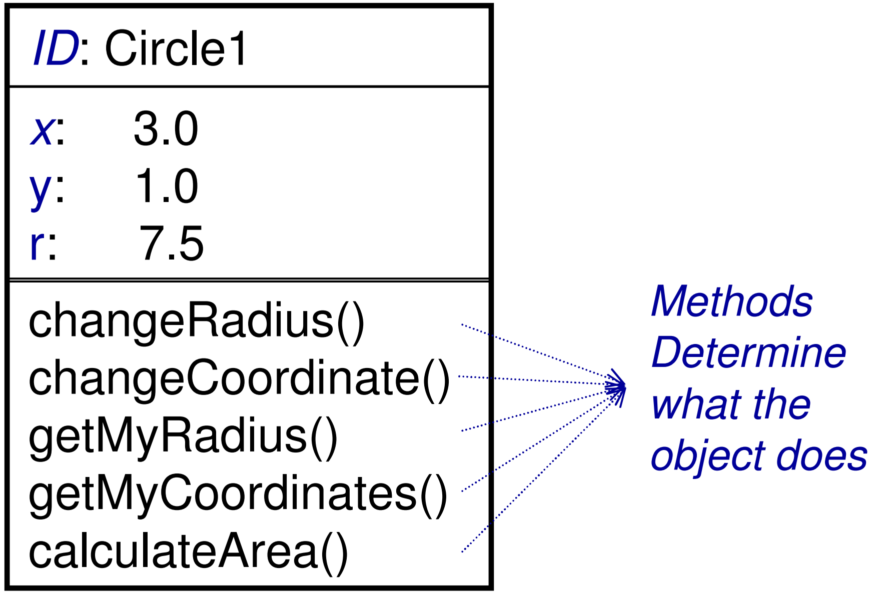
Some attributes cannot be changed.
They are called ***constants***

- Objects instantiated from the same class have the same attributes, but may have different states (object IDs have to be different)

Object intrinsic features

3. **Behaviour**

- Instantiated objects are expected to act and react according to their roles (collect and store data, make decisions, provide services, etc)
- A set of supported actions defines object behaviour (what it does)
- Actions in OOD are called methods



ID: Circle1

x: 3.0
y: 1.0
r: 7.5

changeRadius()
changeCoordinate()
getMyRadius()
getMyCoordinates()
calculateArea()

Methods Determine what the object does

- Some methods can change the object state (changeRadius, …). These methods are called **mutators**
- Some other methods can only provide a value of an attribute without changing the object state (getMyRadius, …) These methods are called **accessors**.

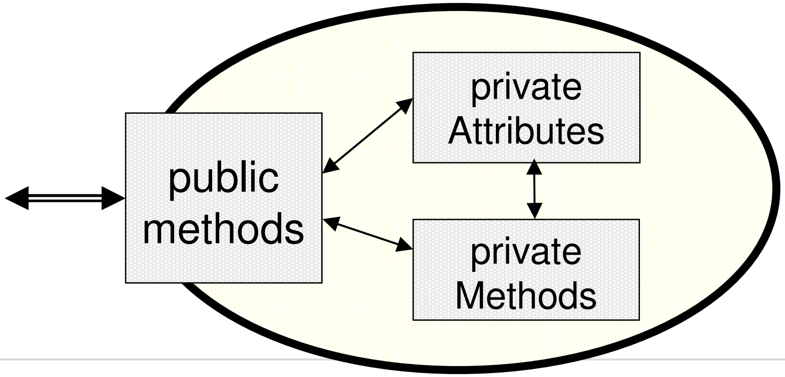
Fundamental OOD concepts

Encapsulation

- Some industrial products are sealed to prevent access to their internal components. They can be used through external terminals
- OOD stipulates that to prevent unauthorized access to the most critical object attributes and methods, they need to be placed into a protective wrapper inside the object to make sure that:
  - well designed and tested objects cannot be alternated or corrupted by other poorly designed objects
  - hidden components can be modified without affecting object interaction (providing that interface methods are not affected by modifications)

- *Attributes and Methods which are specified as **private** are hidden inside objects and cannot be accessed from outside*

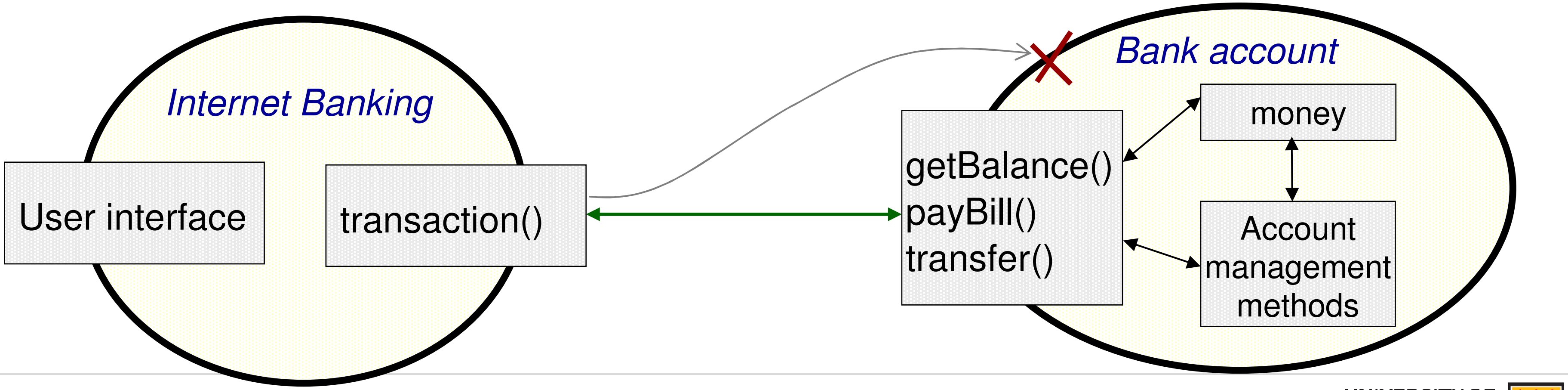
- *Methods specified as **public** can be used for interaction with other objects*



Fundamental OOD concepts

Encapsulation

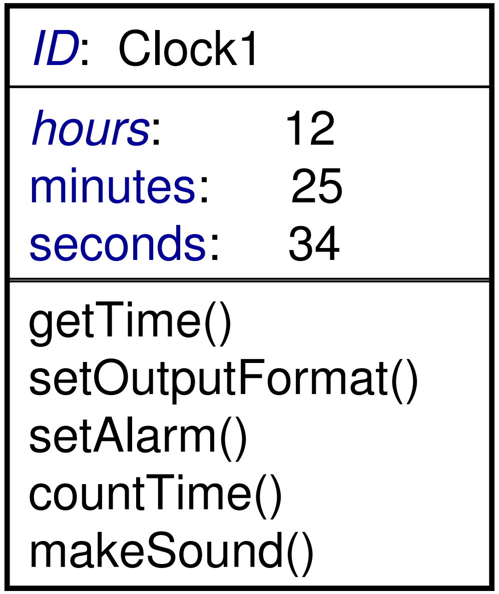
- According to OOD, interaction between objects can take place only through public methods
- In general, all attributes should be specified as private. If they need to be accessed from outside, this should be possible only through a limited set of public methods (accessors, or mutators)
- Methods which are irrelevant to object interaction also should be private
- If a private attribute, or a private method can be accessed directly from outside, this indicates a serious design oversight (a safety bug)



Quiz

You need to implement an object Clock that

- counts time
- provides the current time on request
- can change on request the output format from 24hrs to AM/PM
- can set alarm
- can make beeping sound when alarm is on



ID: Clock1

hours: 12
minutes: 25
seconds: 34

getTime()
setOutputFormat()
setAlarm()
countTime()
makeSound()

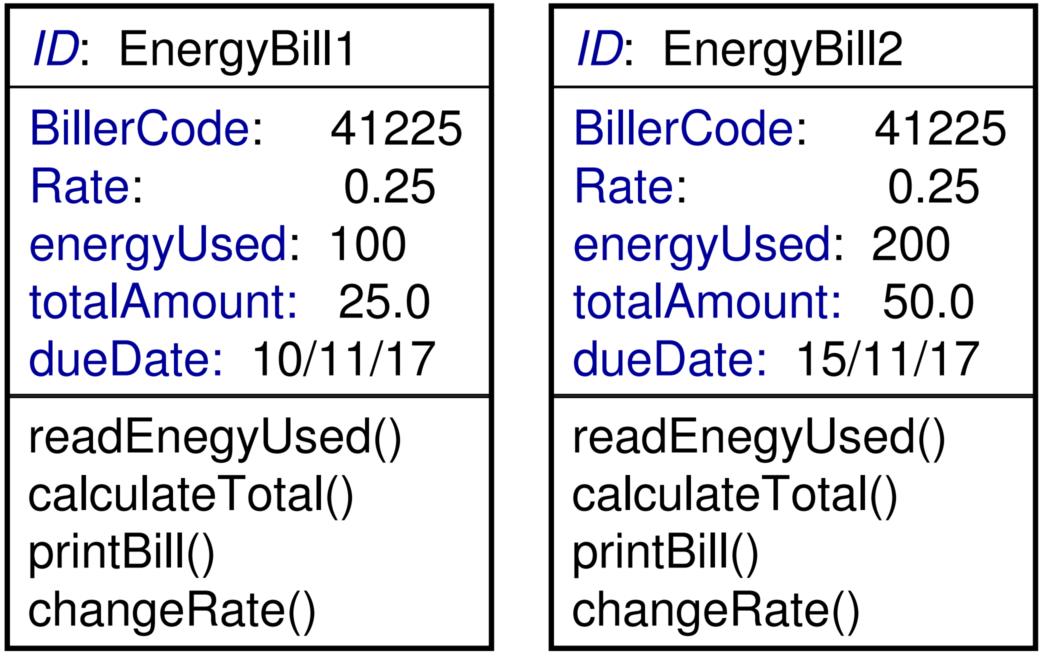
1. Which methods should be public and which ones should be private ?

2. Which methods are accessors and which ones are mutators?

Fundamental OOD concepts

Static attributes and static methods

Consider EnergyBill objects: EnergyBill1, EnergyBill2, EnergyBill3, . . .



ID: EnergyBill1

BillerCode: 41225
Rate: 0.25
energyUsed: 100
totalAmount: 25.0
dueDate: 10/11/17

readEnegyUsed()
calculateTotal()
printBill()
changeRate()

ID: EnergyBill2

BillerCode: 41225
Rate: 0.25
energyUsed: 200
totalAmount: 50.0
dueDate: 15/11/17

readEnegyUsed()
calculateTotal()
printBill()
changeRate()

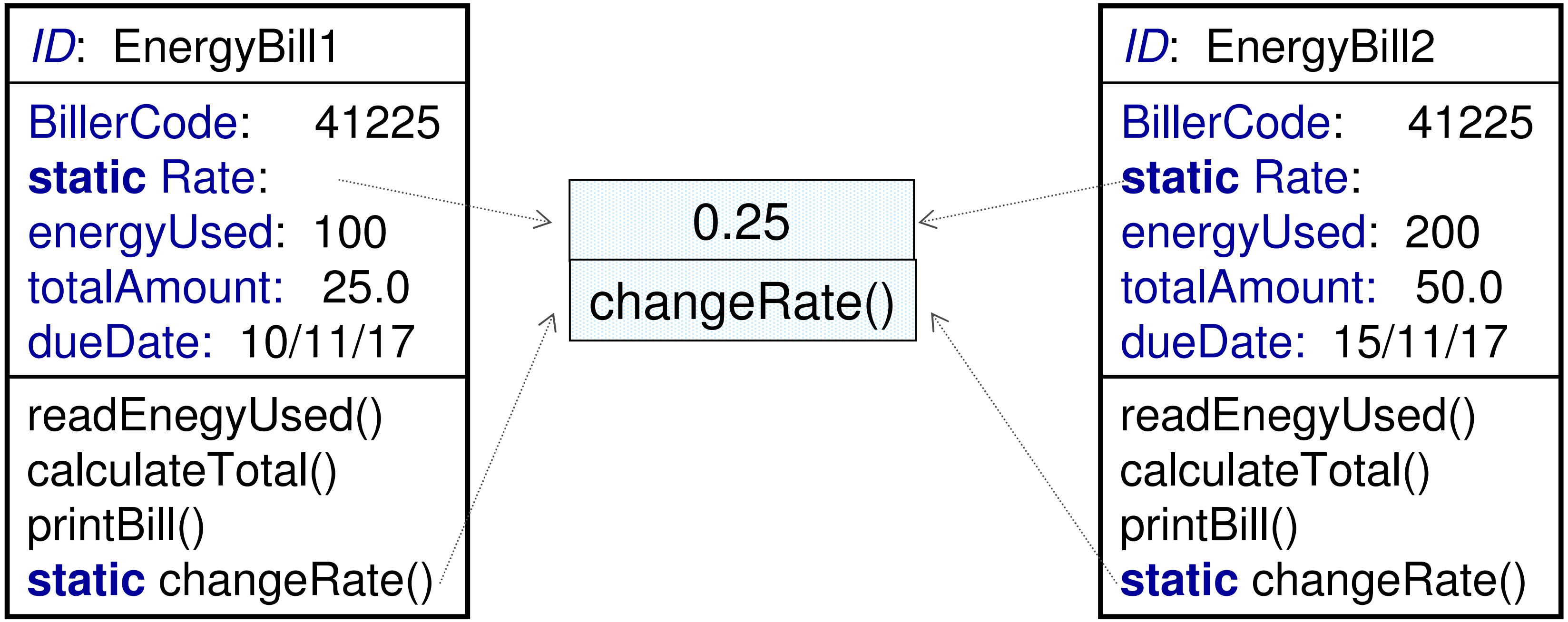
- BillerCode will always remain the same therefore this attribute should be a constant
- Rate may change, using changeRate() method. This change has to be done simultaneously for all EnergyBill objects

How to share a private attribute Rate among all objects, so that if it is changed in one EneryBill, it will simultaneously affect all other objects ?

Fundamental OOD concepts

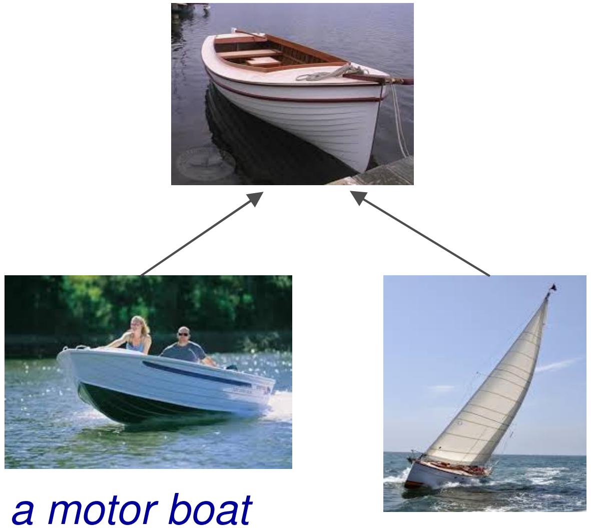
Static attributes and static methods

Attributes and methods specified as static are shared among all objects instances of the same class



As static methods are shared among all objects, they cannot access non-static attributes, or call non-static methods

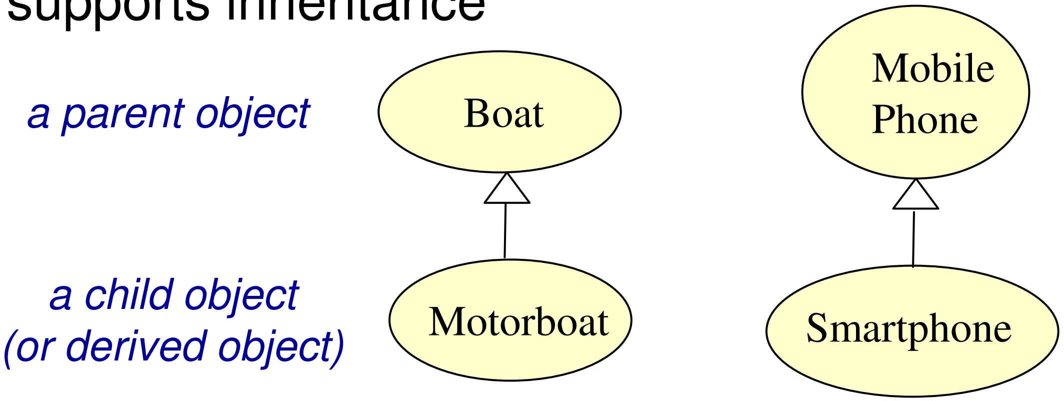
Fundamental OOD concepts



Inheritance

- It is common for many real world objects to inherit properties from other related objects

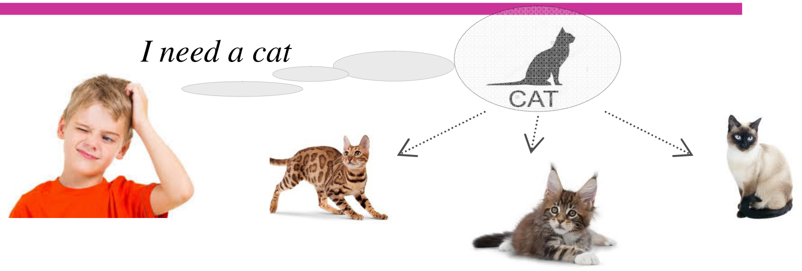
- To facilitate reusability of objects, OOD supports inheritance



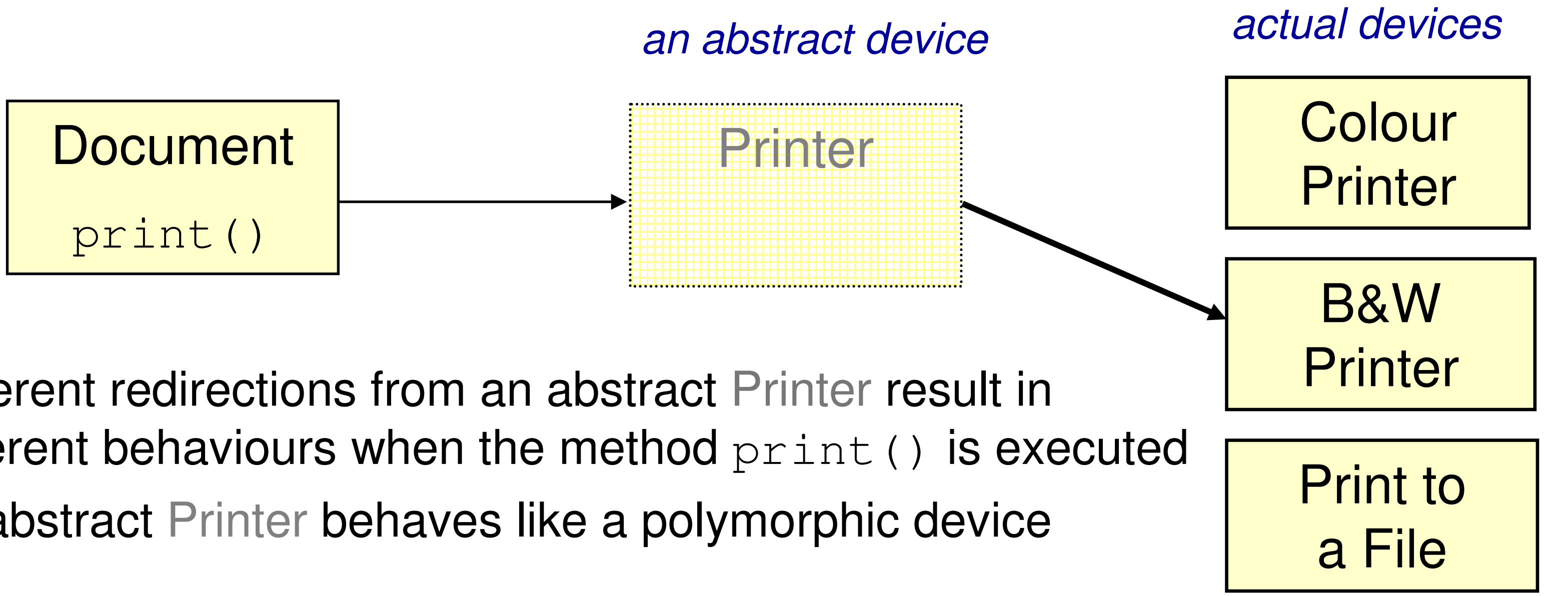
- Inheritance indicates that objects are related
- The derived object inherits attributes and methods of the parent object and therefore does not need to implement them again

Fundamental OOD concepts

Polymorphism



- Polymorphism means “many forms”
- Interaction between objects is not always 'hard-wired'
- Polymorphism allows interaction with a virtual object be redirected to different actual objects depending on the context

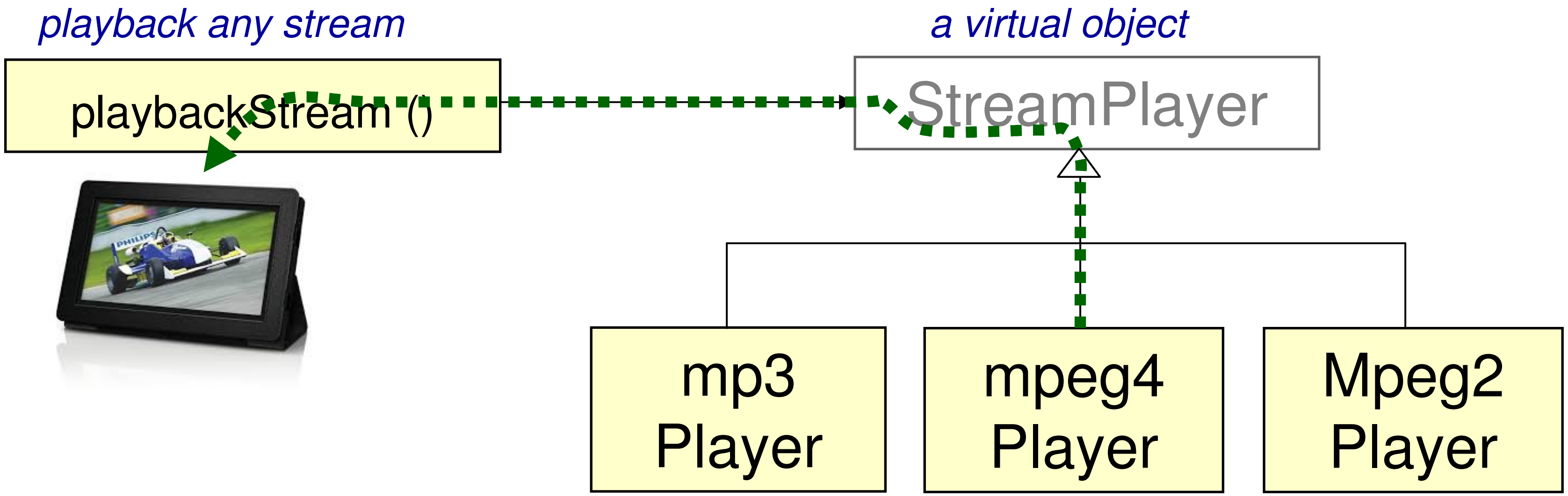


Different redirections from an abstract Printer result in different behaviours when the method `print()` is executed

An abstract Printer behaves like a polymorphic device

Fundamental OOD concepts

Polymorphism



- Polymorphism leads to a generic implementations
- Binding to the right objects is carried out dynamically at run-time
- Polymorphism is one of the most powerful OOD concepts supported by Java programming language

Object Oriented Methodology

- OOD is not just a basis for OO programming languages, but a way of looking at the whole of the software development process – OO Methodology
- Making a program comprised of interacting objects is a big design challenge that cannot be done in a hurry
- OOM presumes that a lot of work needs to be done before you get down to typing your code. OOM prescribes a formal multistage process
- A program should be formally described using the UML (Unified Modeling Language) at the system design stage
- A UML description can be implemented using different OO programming languages

Verbal description, ambiguity, confusion,…

- You may try to describe your program using a natural language
- Natural languages do not have exact rules for technical descriptions
- Even a simple natural language description can be interpreted differently by different listeners (the more details you provide, the higher chance for misinterpretation)
- A formal modelling language is needed to describe software designs unambiguously

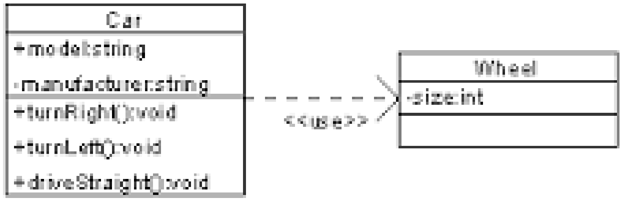
UML is a formal tool for software systems modelling

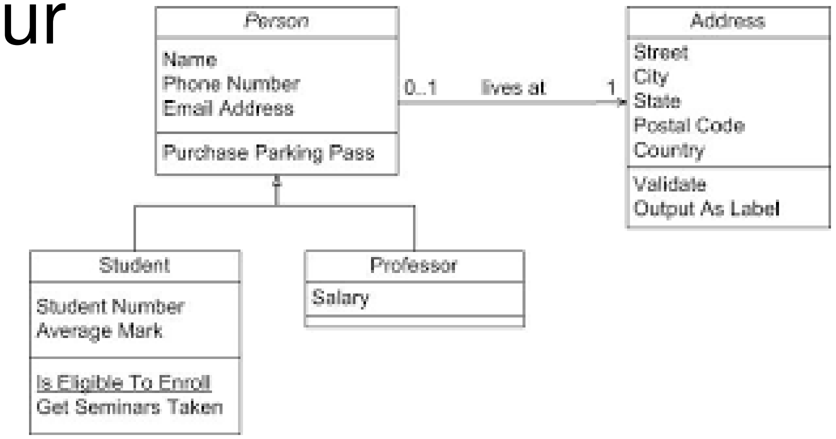
What is UML and how can it help?

- UML (Unified Modelling Language) is a consolidation of research on OO modelling
- UML was accepted by the ISO as an international standard in 2000
- UML can be used
  1. To make sketches when you want to communicate key points of your program architecture
  2. To provide a formal and detailed specification of a software system
  3. To generate code directly from UML diagrams if a special tool is used
- UML models cannot be misinterpreted
- UML is scalable and can be used for small and big projects
- UML facilitates the software development process and simplifies debugging

UML Class Diagrams

- Class diagrams:
  - Contain classes and relations between them
  - Specify class attributes sufficient to describe *any* state of the object instantiated from a class
  - Specify methods sufficient to implement a required object behaviour





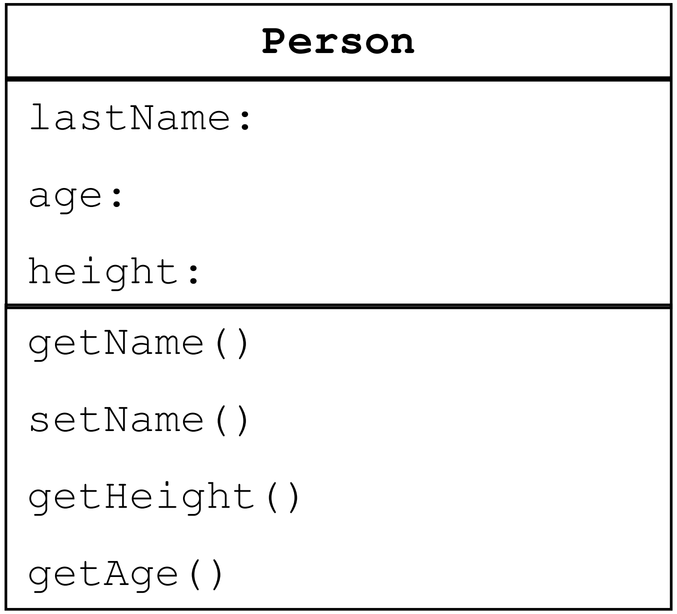
UML class symbol

- The most basic class symbol is a box with the class name. It is used when class properties are not important (usually at initial stages of system design)
- Adding in more details may be needed at later stages
  - Attributes, or data members
  - Methods

class name

attributes

methods



Person

lastName:

age:

height:

getName()

setName()

getHeight()

getAge()

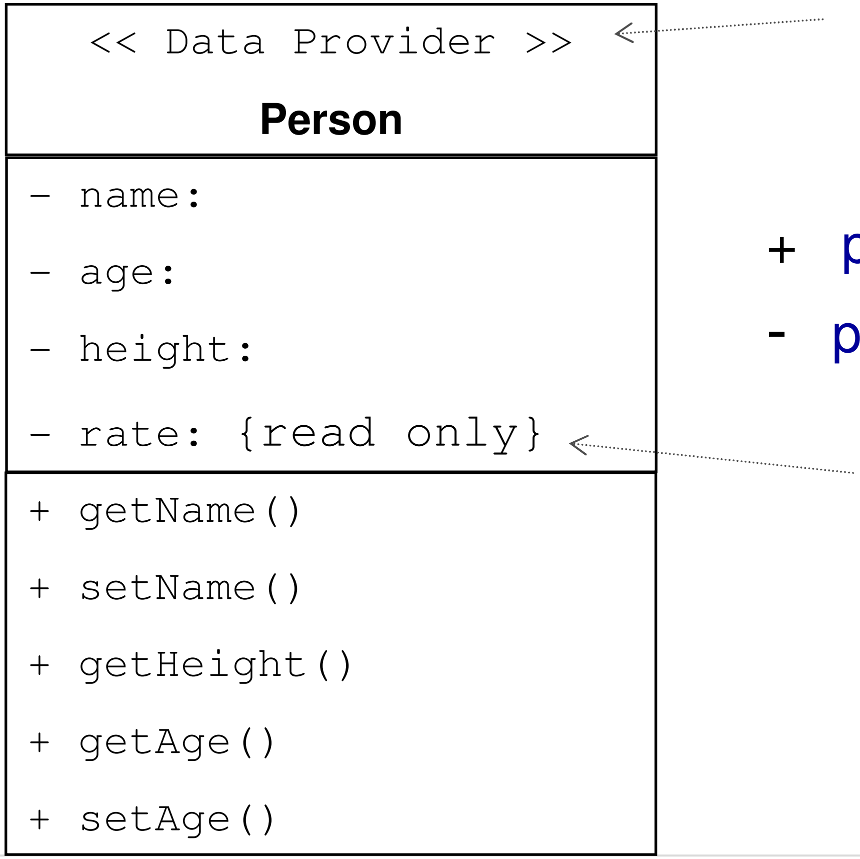
Java

```
class Person {
              . . .
}
```

- The level of details depends on who the diagram is intended for
- Class symbols are the building blocks of class diagrams

UML class symbol

- Role stereotypes
- Visibility prefixes



```
<< Data Provider >>
Person
- name:
- age:
- height:
- rate: {read only}
+ getName()
+ setName()
+ getHeight()
+ getAge()
+ setAge()
```

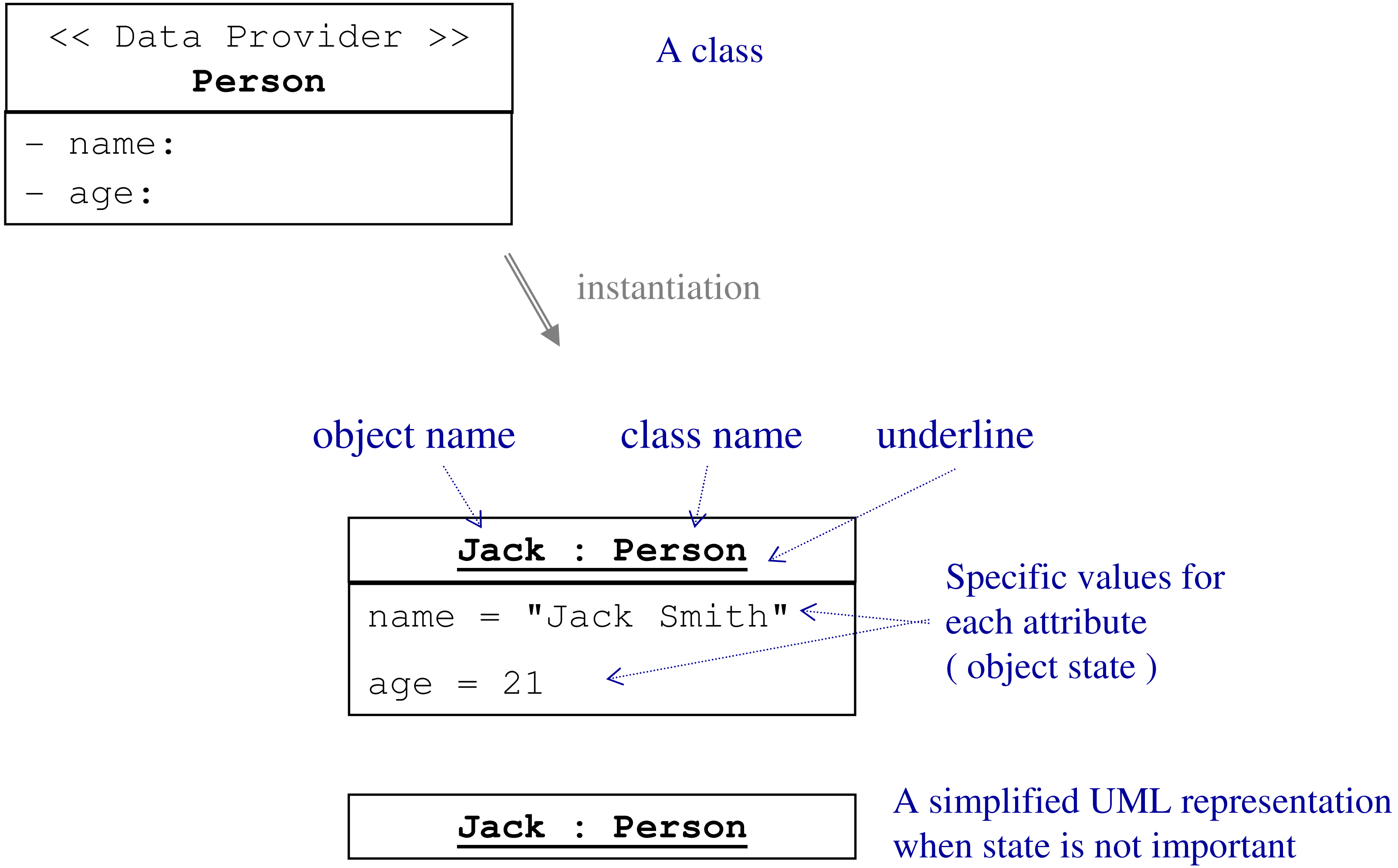
Role stereotype

+ public attributes and methods

- private attributes and methods

constant

UML object symbol



```
<< Data Provider >>
      Person
- name:
- age:
```

A class

instantiation

object name

class name

underline

```
Jack : Person
name = "Jack Smith"
age = 21
```

Specific values for each attribute (object state)

```
Jack : Person
```

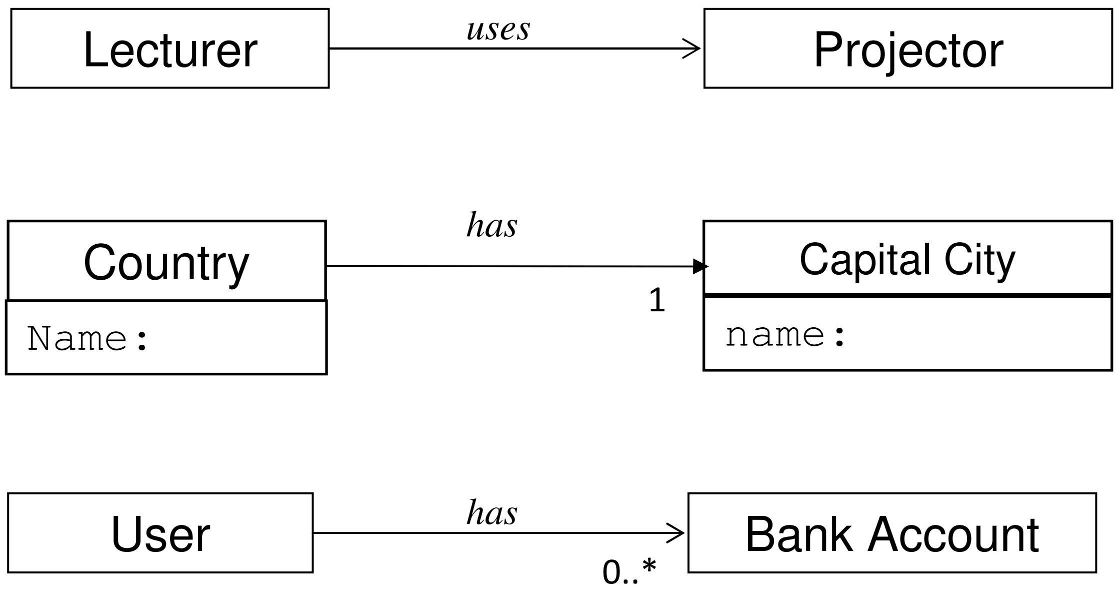
A simplified UML representation when state is not important

Class Relationship

- Class symbols by themselves provide very limited information about your program architecture
- A model of the system should reflect relationship between classes
- OOD defines the following types of relationship between classes:
  - association
  - composition
  - inheritance

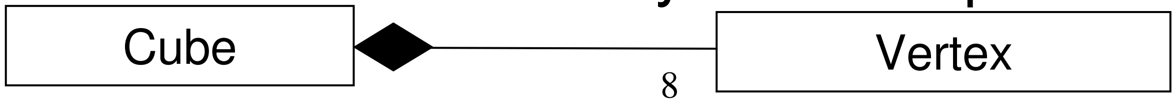
Association

- Indicates that one class uses another class
- Association can be described as “has” or “uses” type of relationship



Composition

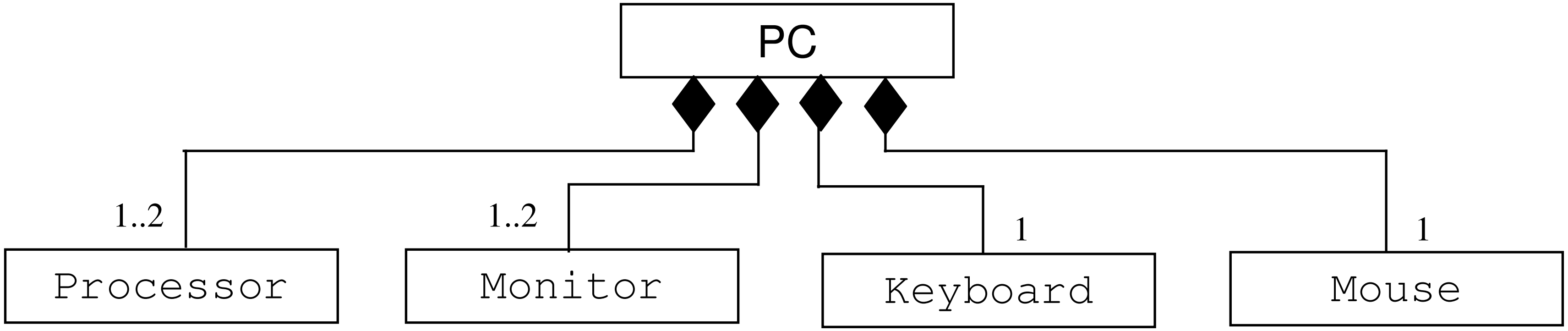
- Composition is a special kind of association
- Composition reflects “contains” or “owns” type of relationship
- Components “live” inside the container with their lifespan synchronized with the container
- Deletion of the container destroys the component objects



Cube

Vertex

8



PC

1..2

Processor

1..2

Monitor

1

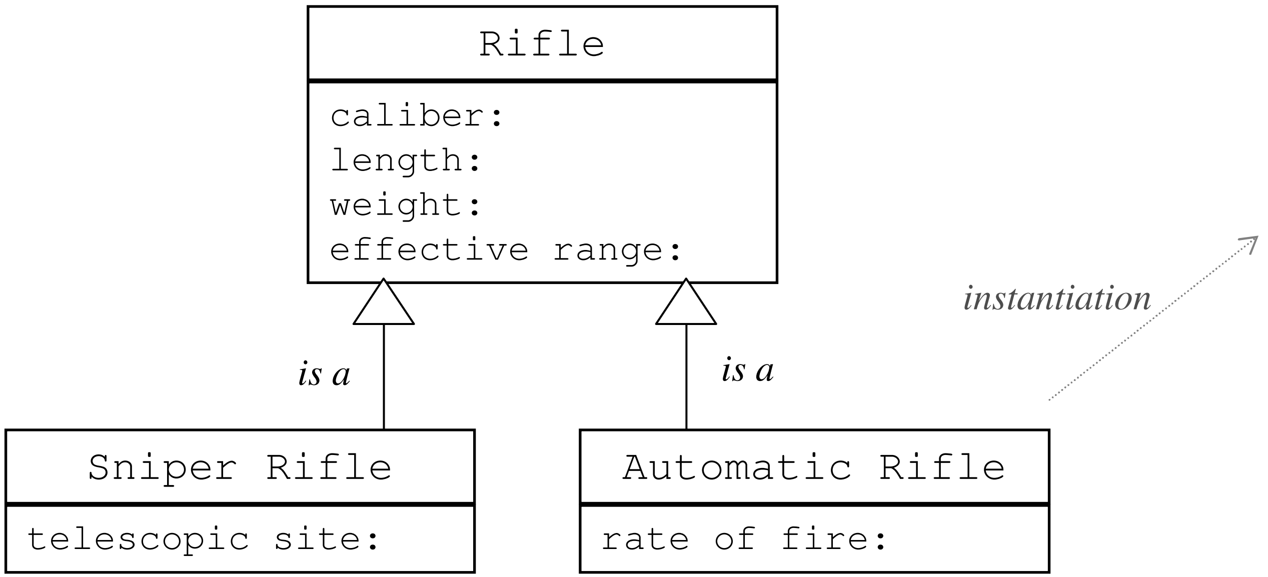
Keyboard

1

Mouse

Inheritance

- Inheritance implements an “is a” type of relationship
  - Motorboat **is a** boat
  - Smartphone **is a** mobile phone
- A child class inherits attributes and methods from its parent class and adds new attributes and methods to implement new properties and behaviours



Rifle

caliber:
length:
weight:
effective range:

is a

Sniper Rifle

telescopic site:

is a

Automatic Rifle

rate of fire:

instantiation

M4 : Automatic Rifle

```
caliber = 5.56
length = 840
weight = 3.4
effective range = 500
rate of fire = 950
```

This instantiated object has inherited all properties of the parent class

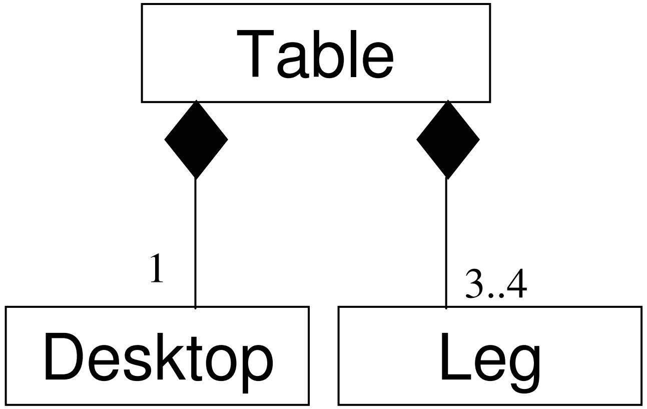
Quiz

- What is the most appropriate relationship between the following classes:

Table

Desktop

Legs



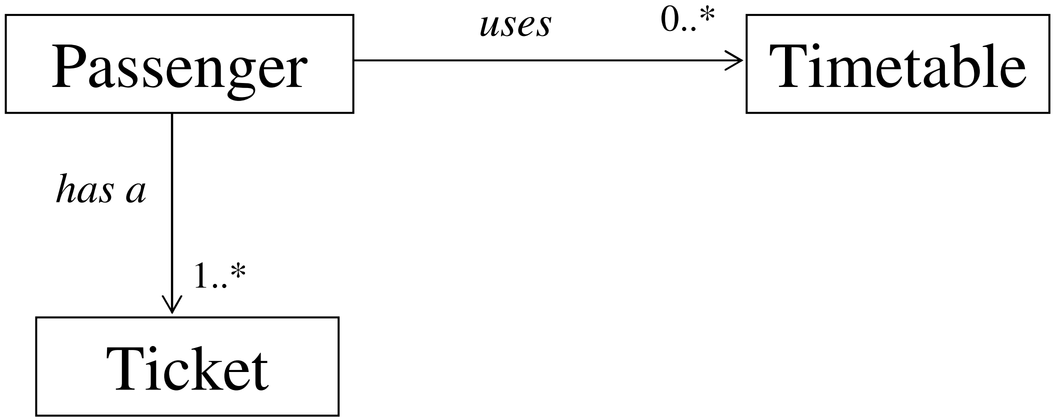
Quiz

- What is the most appropriate relationship between the following classes:

Passenger

Ticket

Timetable



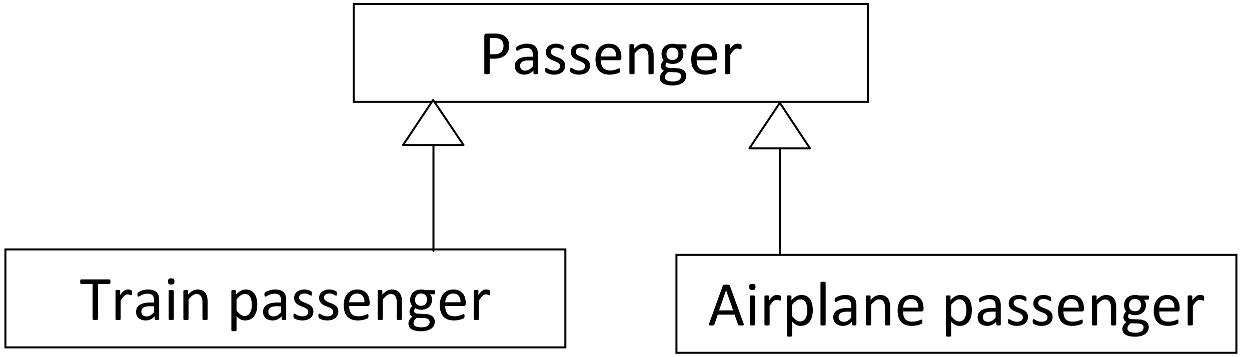
Quiz

- What is the most appropriate relationship between the following classes:

Passenger

Train passenger

Airplane passenger



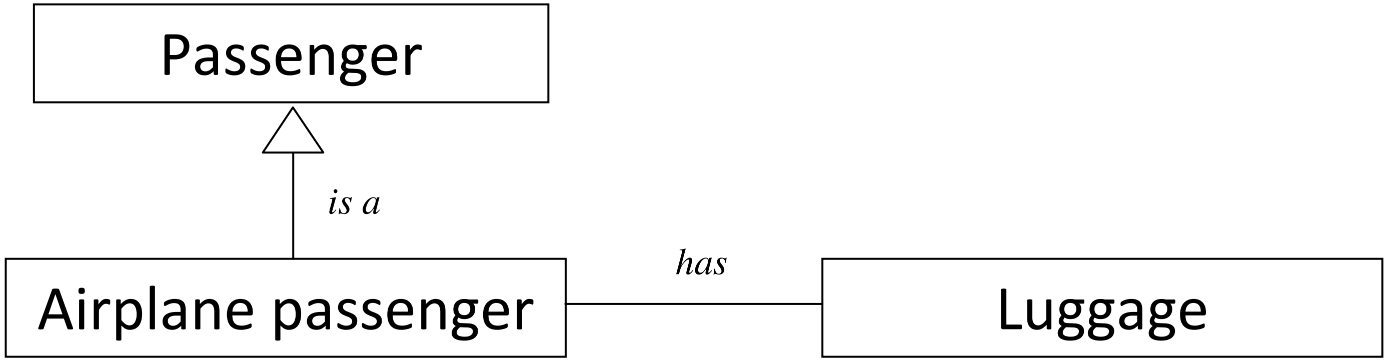
Quiz

- What is the most appropriate relationship between the following classes:

Passenger

Airplane passenger

Luggage



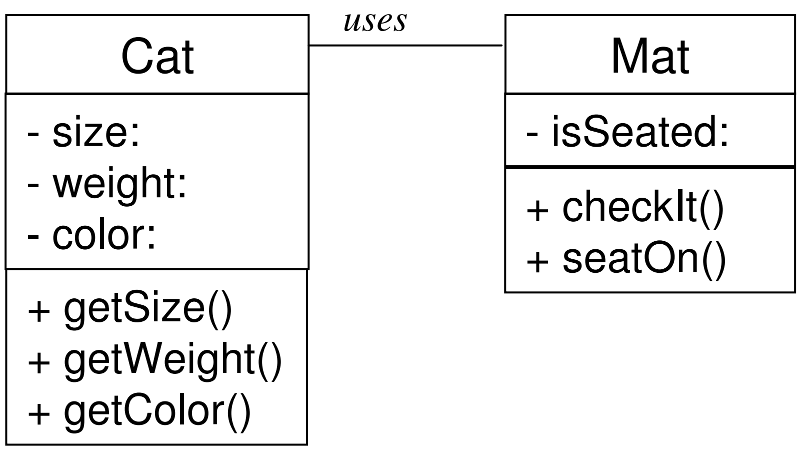
Defining classes

- The problem of defining an appropriate collection of classes and their relationship for your project is generally non-trivial
- Although there are some complicated theories which should help, practically you can mostly rely on your experience and intuition
- We are going to look at an made-up simple application scenario to show how to determine the objects and properties which seem appropriate

Scenario: A big heavy red cat seats on a mat

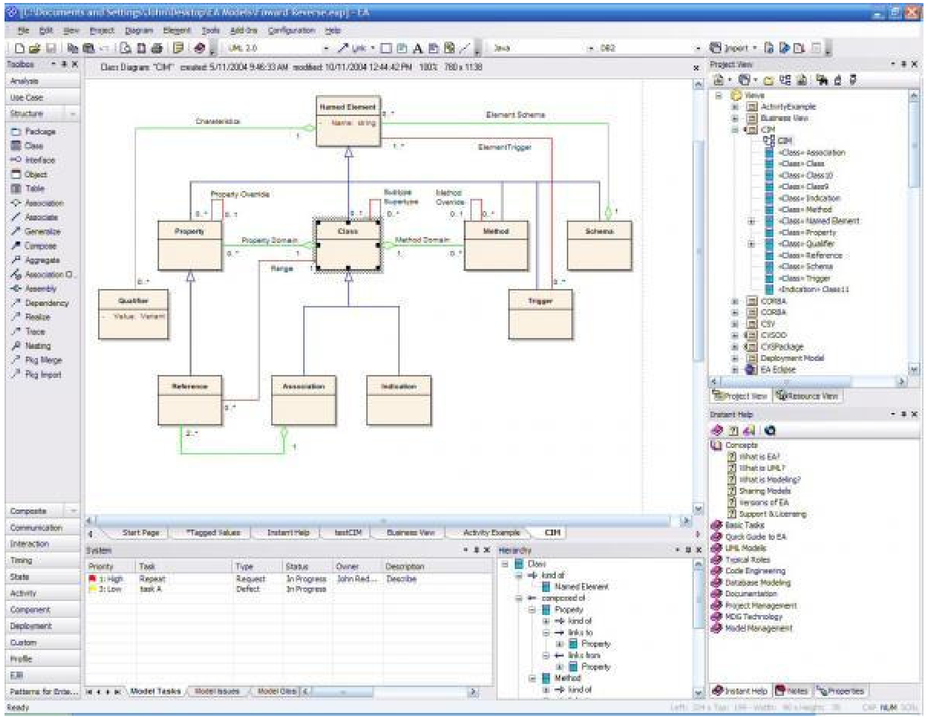
- A question we need to answer:
  - What objects are needed to implement this scenario and what are their properties?
- **Objects?**
  - Cat, Mat
- **Cat:**
  - Attributes?
    - Big → Size? → Width, Height, Depth?
    - Heavy → Weight
    - Red → Colour
  - Methods/Behaviours?
    - Measure Size→ Test for “bigness”
    - Measure Mass→ Test for “heaviness”
    - Show colour
    - Sitting on the mat (interaction)
- **Mat:**
  - Behaviours?
    - Can be sat on
- Classes: Cat, Mat, … ?

Association



UML Design Tools

Enterprise Architect



- One of the most advanced and most widely used tools in the industry
- Supports all UML diagrams
- Checks correctness of models
- Dynamic model simulation
- Generation of documentation and reports in a specified format
- Generation of Java, C++, C# source code from UML models
- Helps to visualize your applications by supporting reverse engineering

Suggested reading

Java: How to Program (Early Objects), 11th Edition

- Chapter 1
  1.5 Introduction to object technology