Object Oriented Analysis & Design

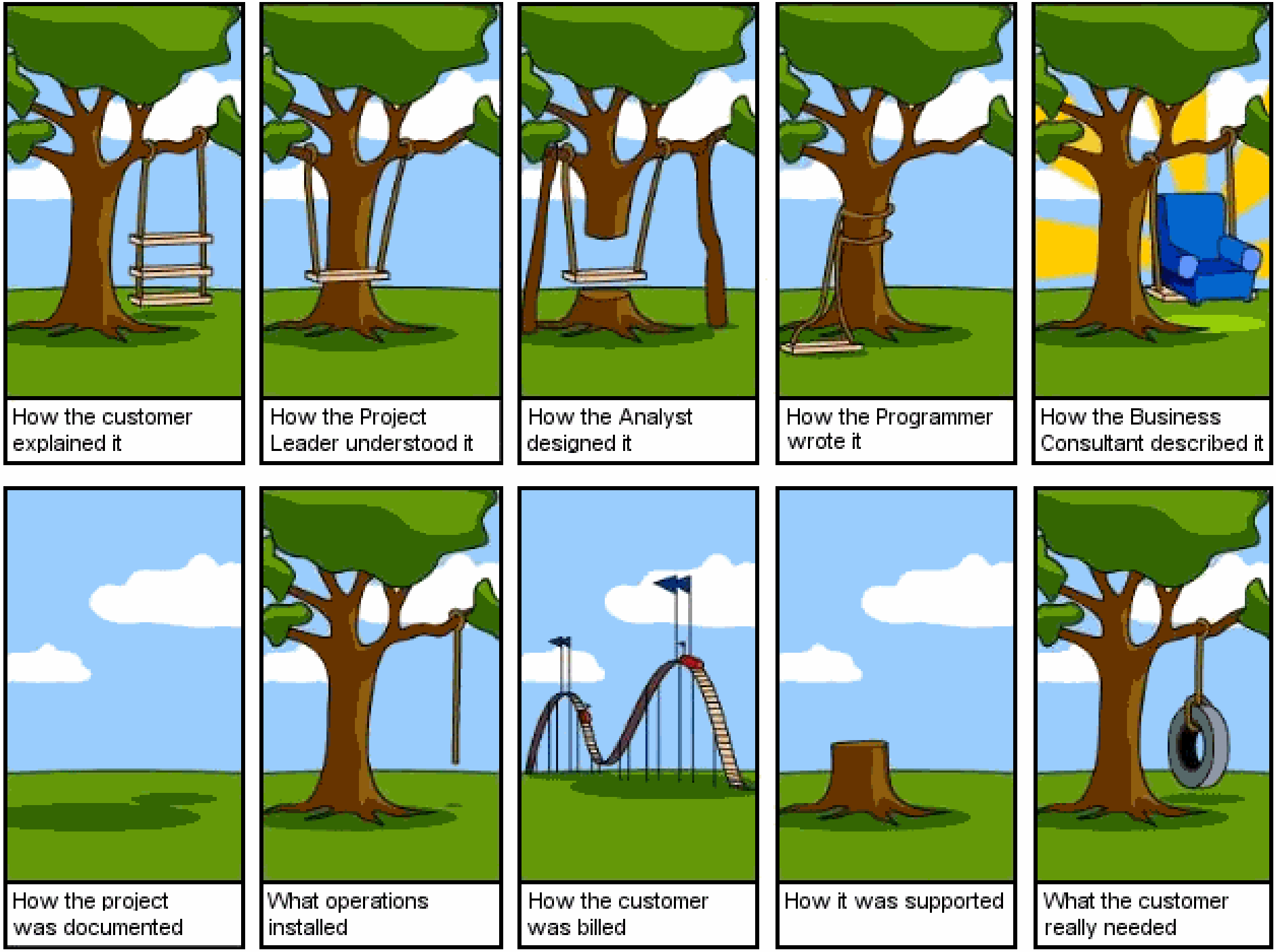
Introduction to OOAD

- **It focuses on objects** where system is broken down in terms of the objects that exist within it.

- Functions (**behaviour**) and data (**state**) relating to a single object are self-contained or encapsulated in one place.

OO Concepts

- Objects
- Classes - Associations
- Inheritance -Aggregations and Compositions
- Polymorphism
- Encapsulation and Data hiding
- Interfaces


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Software Engineering vs OOD:

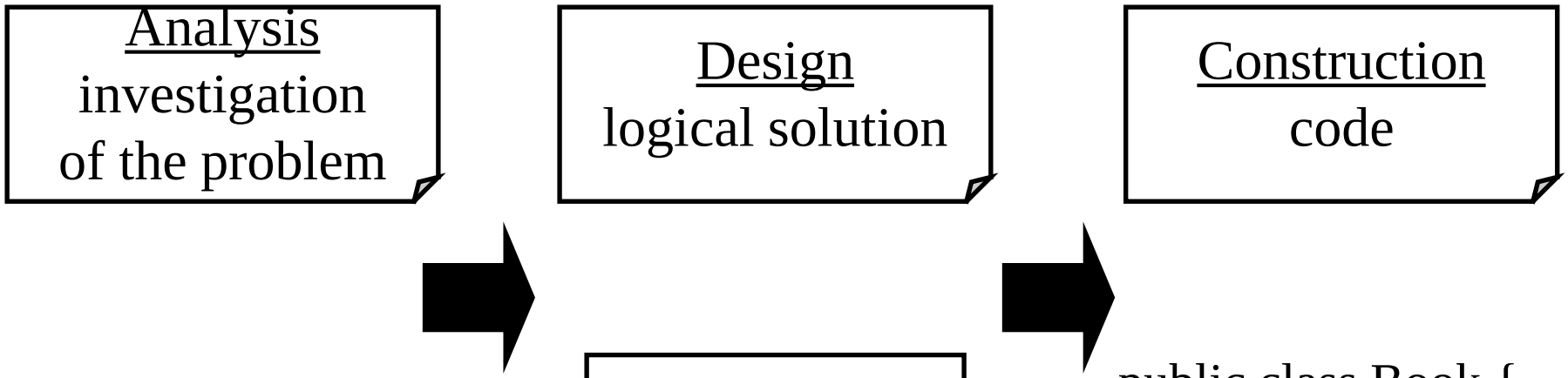
- Software Engineering
  - What needs to be done
  - Who does it
  - When is it done

- OOD
  - What needs to be done
  - What classes are responsible for doing it
  - What classes have the job of seeing it gets done

- **Object-Oriented Analysis (Overview)**
  - An **investigation** of the **problem** (rather than how a solution is defined)
  - During OO analysis, there is an emphasis on finding and describing the objects (or concepts) in the **problem domain**.
  - For example, concepts in a Library Information System include *Book*, and *Library*.
  - High level views found in the application domain.
  - Oftentimes called **domain objects; entities.**

- **Object-Oriented Design**
  - Emphasizes a conceptual **solution** that fulfills the requirements.
  - Need to define sof**tware objects** and how they **collaborate** to meet the requirements.
  - For example, in the Library Information System, a *Book* software object may have a *title* attribute and a *getChapter* method.
    - What are the methods needed to process the attributes?
- Designs are **implemented** in a programming language.
  - In the example, we will have a *Book* class in Java.

From Design to Implementation



Book
(concept)

Domain concept

Book
title print()

Representation in
analysis of concepts

```
public class Book {
      public void print();
      private String title;
}
```

Representation in an
object-oriented
programming language.

Can you see the services / responsibilities in the Book class?

notes in UML

domain concept

Plane
tailNumber

visualization of domain concept

representation in an object-oriented programming language

```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

A Short Example

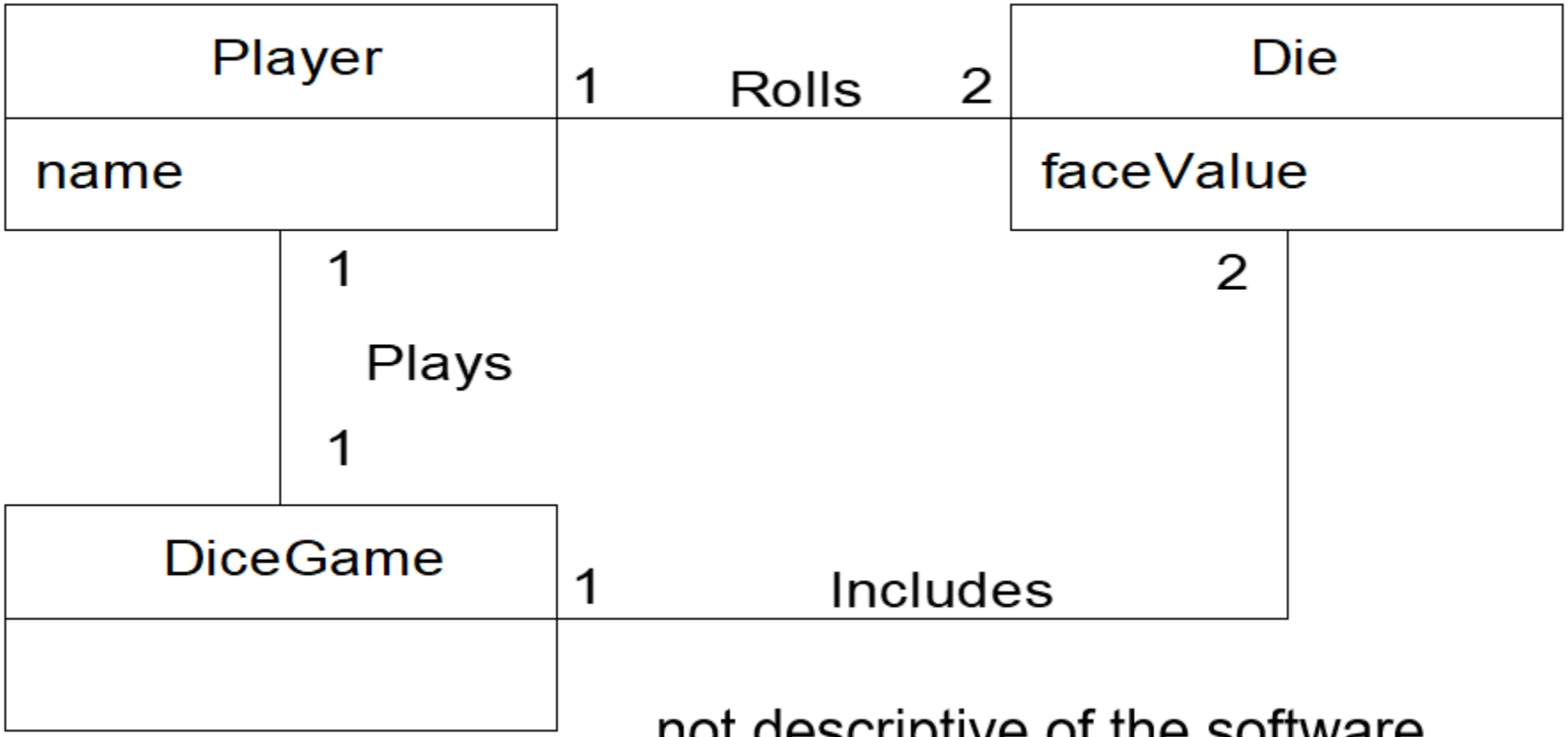
- **Dice Game**: In which a program simulates a player tossing two dice.
- Define Use Cases
  - these are user scenarios, stories, goals

    Play a Dice Game Use Case:

    Player requests a roll of the dice. System presents results:
    If dice show 7 player wins; otherwise player loses

- Define a Domain Model
  - this is a description of the domain from the point of view of the objects involved
  - identify the concepts, attributes and associations
  - result is called the *domain model*

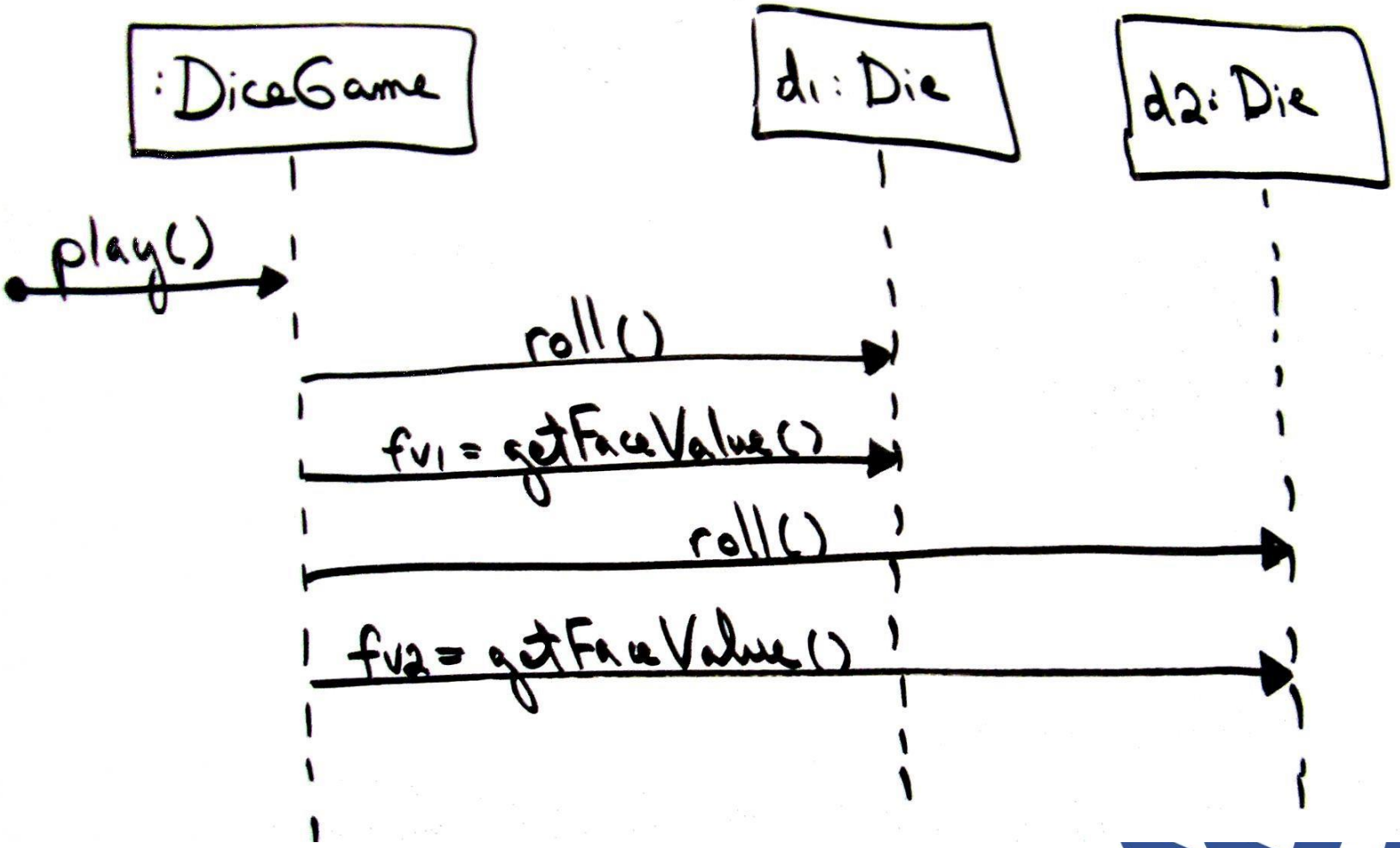
Partial (Conceptual) Domain Model

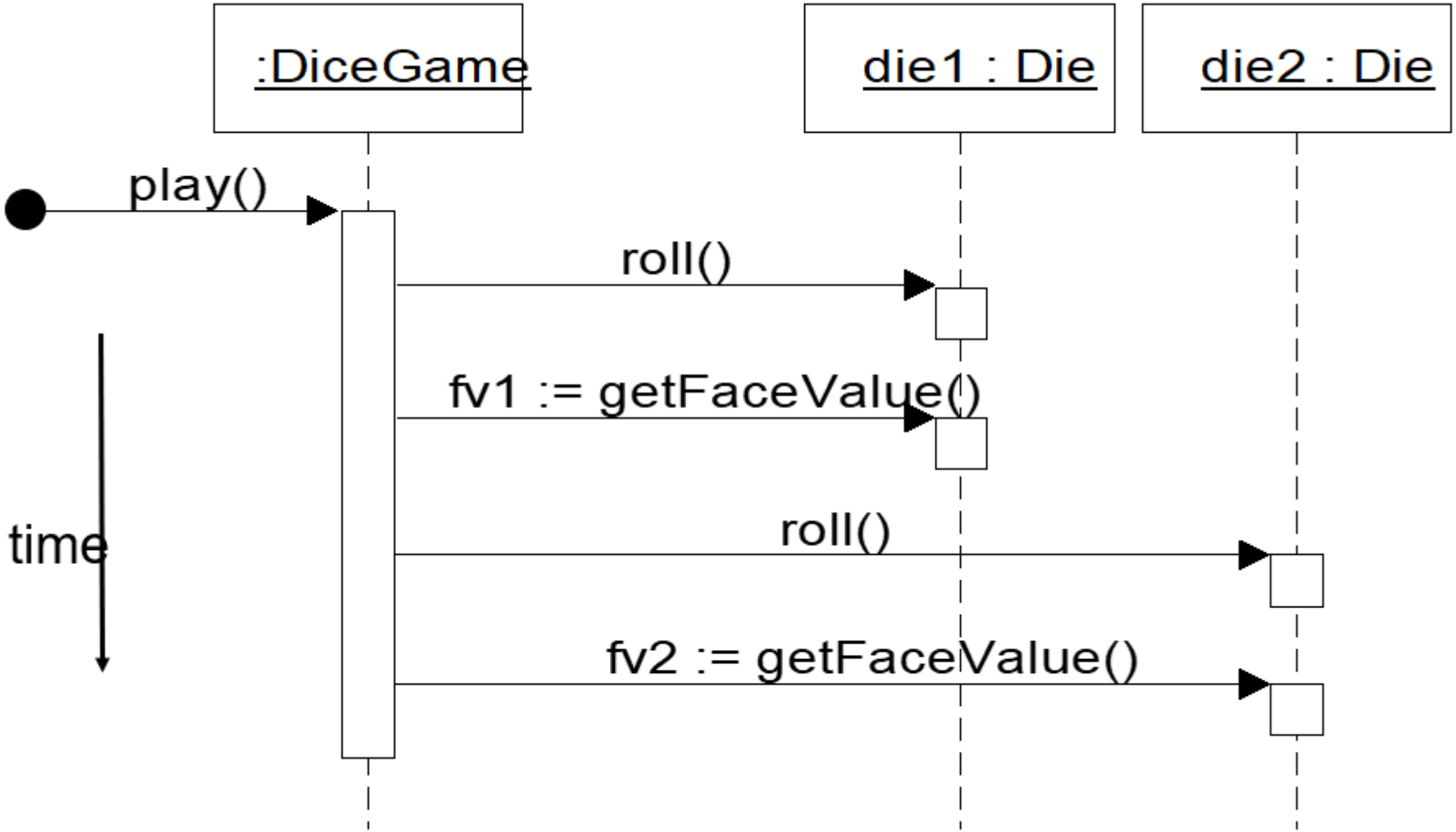


not descriptive of the software objects but rather of the domain

Assigning Responsibilities

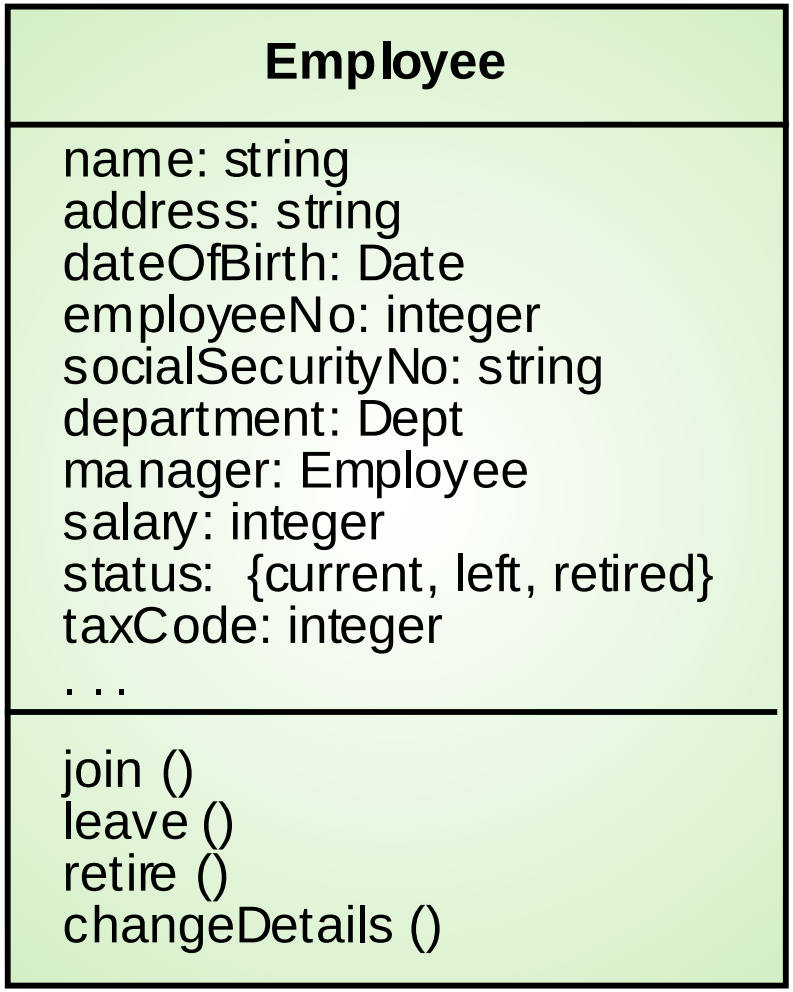
- In a program, objects collaborate – pass each other messages, make data available to one another
- Common notation is called a *sequence diagram*.
- A sequence diagram shows the *flow of messages* between objects
- If we know where the message goes we know who is "responsible" for "responding to" the message.
- Answering the message involves knowing "how" to answer the message
- How close is "flow of message" to the concept of "function call"?


:DiceGame
d1: Die
d2: Die
play()
roll()
fv1 = getFaceValue()
roll()
fv2 = getFaceValue()


:DiceGame
die1 : Die
die2 : Die
play()
roll()
fv1 := getFaceValue()
roll()
fv2 := getFaceValue()
time

Employee object & class

Class



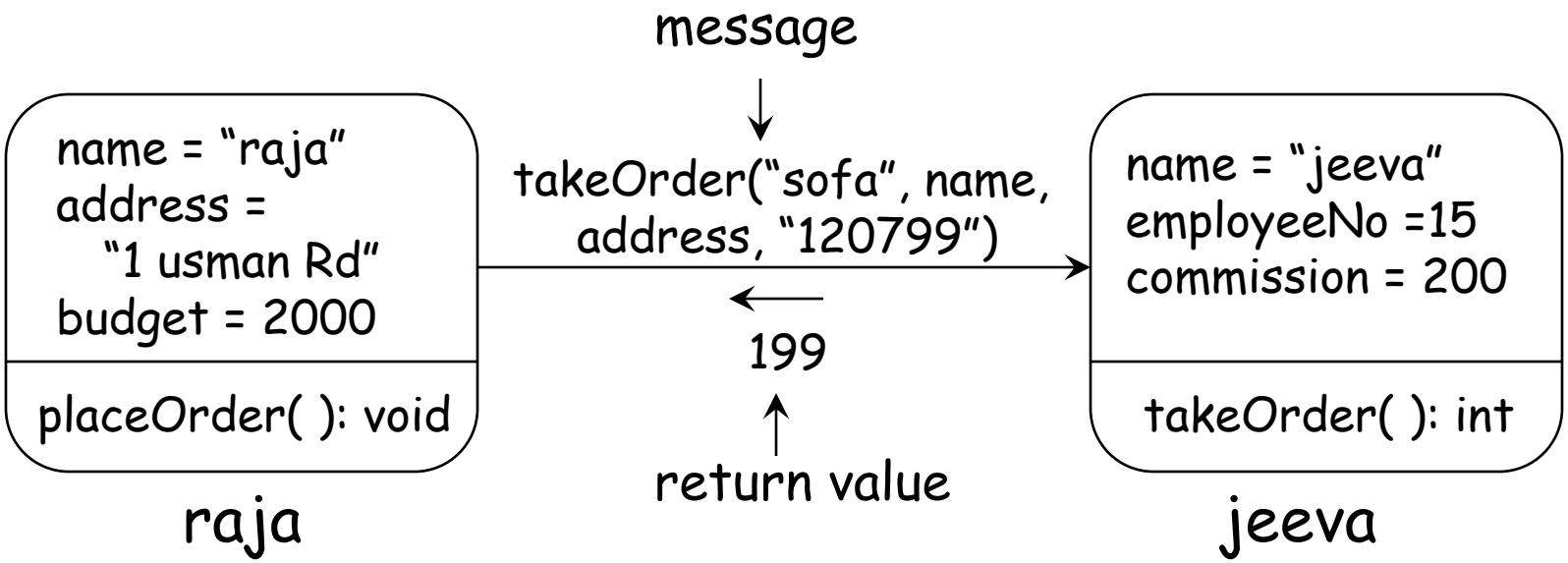
Object

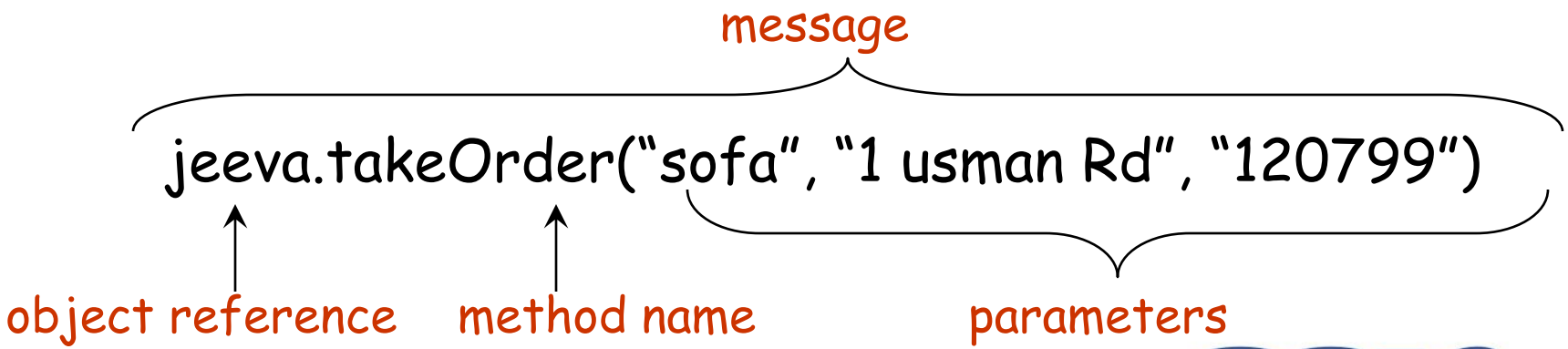
Employee16

name: John
address: M Street No.23
dateOfBirth: 02/10/65
employeeNo: 324
socialecurityNo:E342545
department: Sale
manager: Employee1
salary: 2340
stauts:current
taxCode: 3432
….

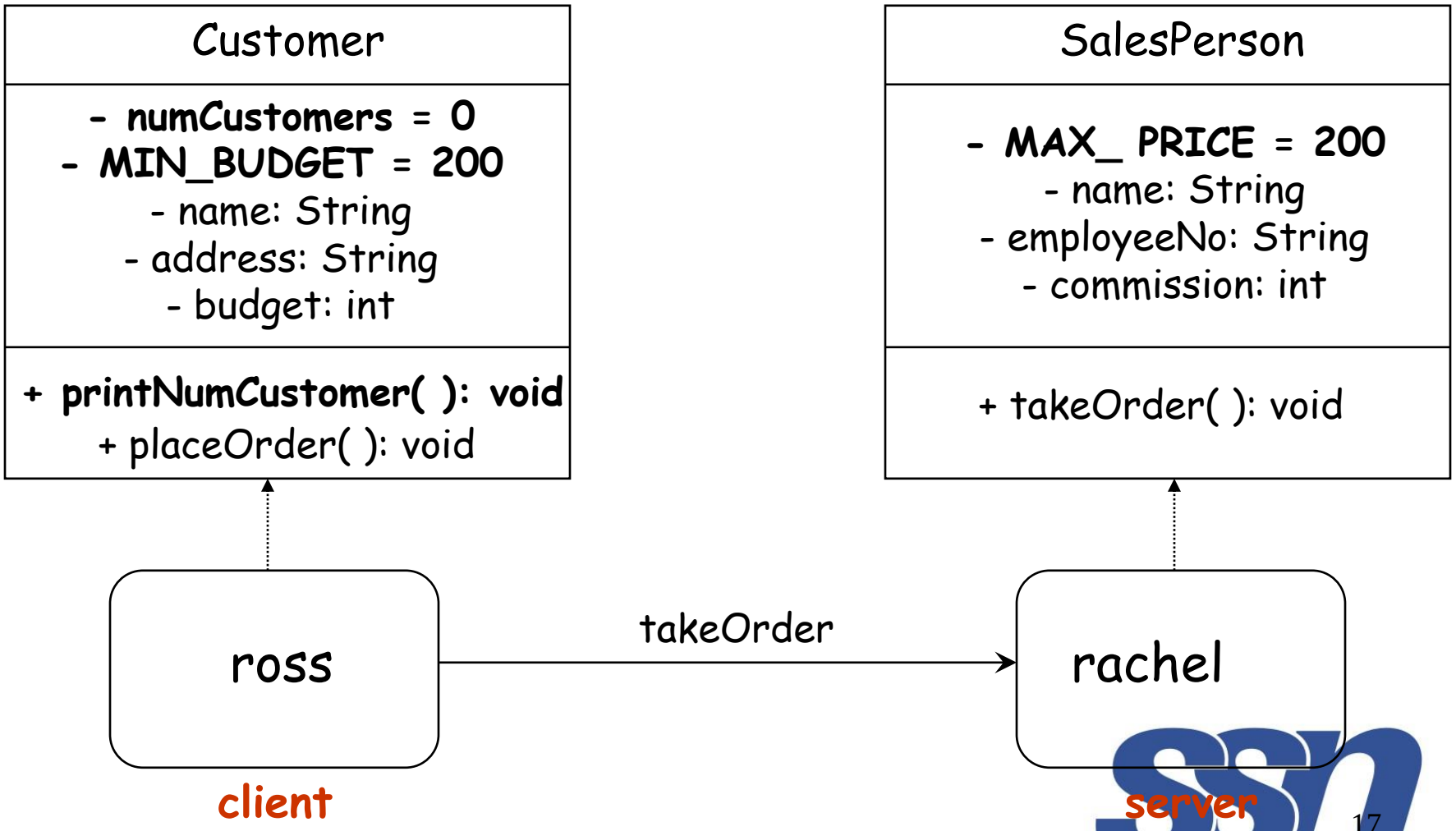
Eployee16.join(02/05/1997)
Eployee16.retire(03/08/2005)
Eployee16.changeDetail(“X Street No. 12”)

Message Passing





Message Passing



OO Analysis and Design

OO Analysis - examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. In other words, the world (of the system) is modelled in terms of objects and classes.

OO Design - OO decomposition and a notation for depicting models of the system under development. Structures are developed whereby sets of objects collaborate to provide the behaviours that satisfy the requirements of the problem.

Object Oriented Analysis

- Analyze the domain problem
- Describe the process systems
- Identify the objects
- Specify attributes
- Defining operations
- Inter-object Communication

Identifying Object

- Objects can be:
  - External Entity (e.g., other systems, devices, people) that produce or consume information to be used by system
  - Things (e.g., reports, displays, letters, signals) that are part of information domain for the problem
  - Places (e.g., book's room) that establish the context of the problem and the overall function of the system.
  - Organizational units (e.g., division, group, team, department) that are relevant to an application,
  - Transaction (e.g., loan, take course, buy, order).

Example of candidate objects

Just a Line management wishes to increase security, both in their building and on site, without antagonizing their employees. They would also like to prevent people who are not part of the company from using the *Just a Line* car park.

It has been decide to issue identity cards to all employees, which they are expected to wear while on the *Just a Line* site. The cards records the name, department and number of the member of staff, and permit access to the *Just a Line* car park.

A barrier and a card reader are placed at the entrance to the car park. The driver of an approaching car insert his or her numbered card in the card reader, which then checks that the card number is known to the *Just a Line* system. If the card is recognized, the reader sends a signal to raise the barrier and the car is able to enter the car park.

At the exit, there is also a barrier, which is raised when a car wishes to leave the car park.

When there are no spaces in the car park a sign at the entrance display "Full" and is only switched off when a car leaves.

Special visitor's cards, which record a number and the current date, also permit access to the car park. Visitor's cards may be sent out in advance, or collected from reception. All visitor's cards must be returned to reception when the visitor leaves *Just a Line.*

Candidate objects:

Just a Line	management	security	building
site	employee	people	company
car park	card	name	department
number	member of staff	access	barrier
card reader	entrance	driver	car
system	signal	exit	space
sign	visitor	reception	

Candidate objects' rejection

- **duplicates**: if two or more objects are simply different names for the same thing.
- **irrelevant**: objects which exists in the problem domain, but which are not intended.
- **Vague(not clear)**: when considering words carefully it sometimes becomes clear that they do not have a price meaning and cannot be the basis of a useful in the system.
- **general**: the meaning is too broad.
- **attributes**: as the attribute of objects.
- **associations**: actually represents the relationships between objects.
- **roles**: sometimes objects referred to by the role they play in a particular part of the system.

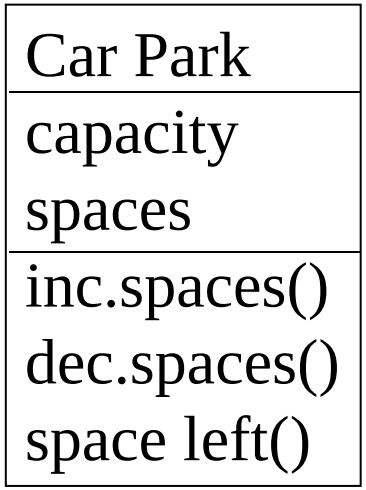
Rejected Candidate objects

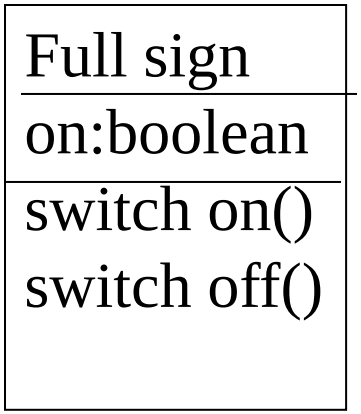
Candidate objects	Rejection criteria
Just a Line, member of staff	duplicates with company, employee respectively
management,company, building, site, visitor and reception	irrelevant to the system
security, people	vague
system	too general
name, department,	attribute
access	association
driver	role

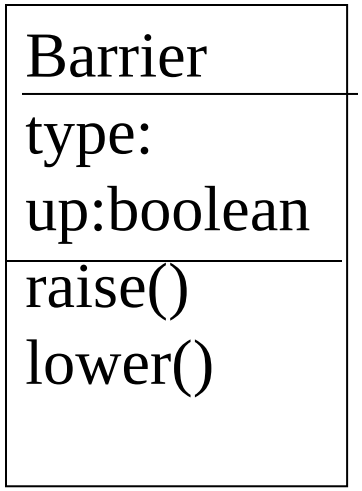
Rest Objects

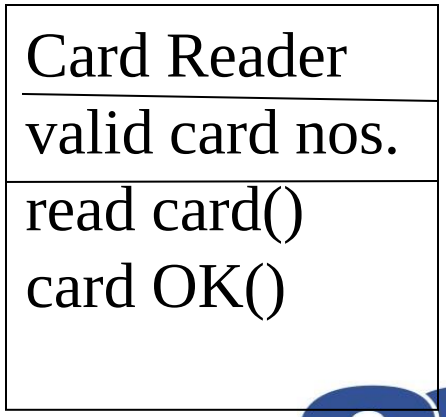
Car park	Staff Card	Visitor’s card
Employee	Entrance	exit
card reader	barrier	Full sign
space	sensor	car

Define class attributes

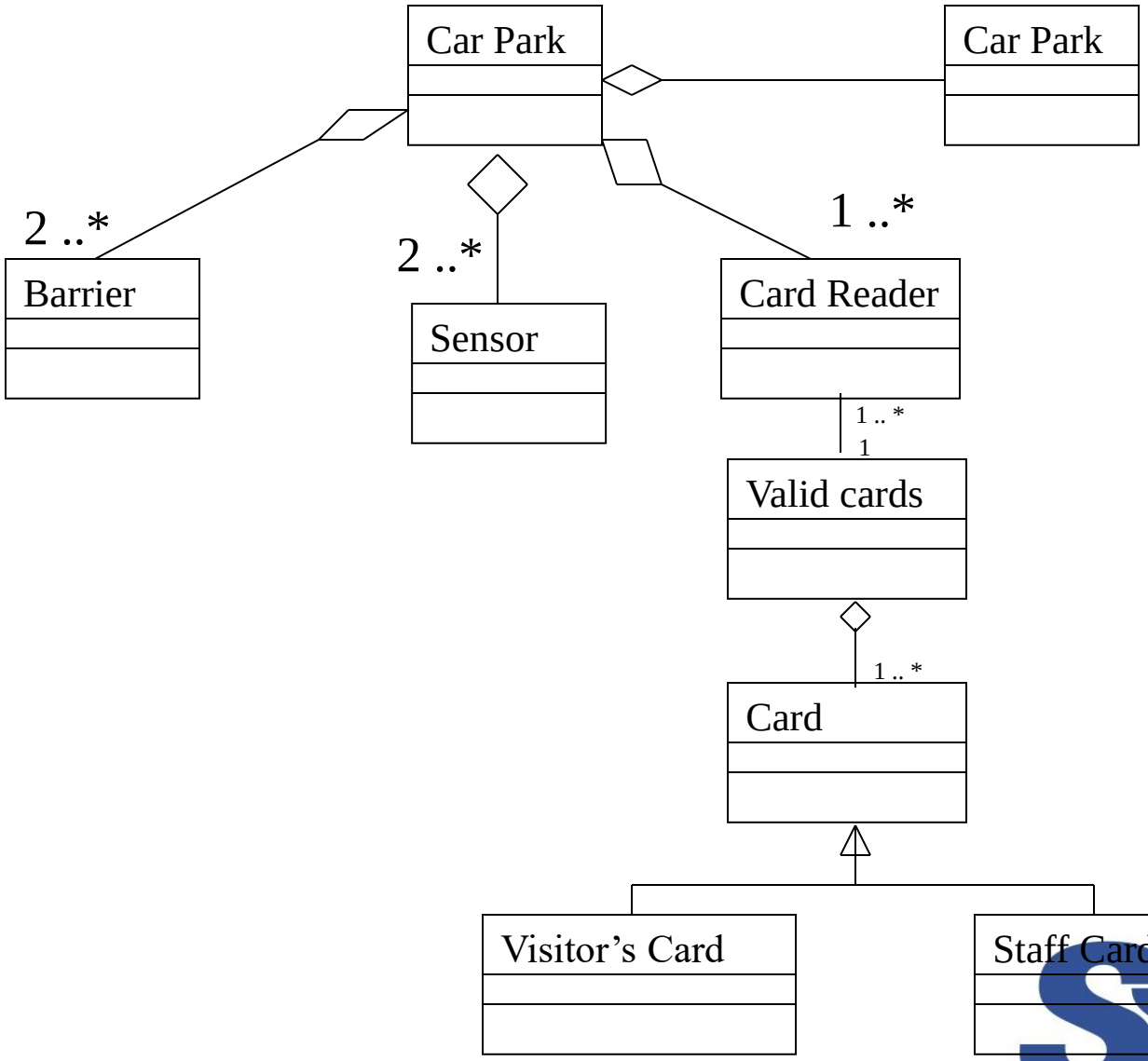








Objects Relationship



Object behaviour modelling

- A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a use-case
- Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects

OO Analysis and Design

Object-Oriented (OO) development is **very different** from structured development:

- Structured approach focuses on major **functions** within a system and on the **data** used by the functions.
- OO approach focuses on the **objects** in a system and on the relationships between those objects.

- Unlike functional decomposition, OO views a complex problem as a meaningful collection of objects that collaborate to achieve some higher level behaviour => closely **mirrors how people view complex problems** => using OO should make the job of developing large, complex systems more manageable.

Object Oriented Model

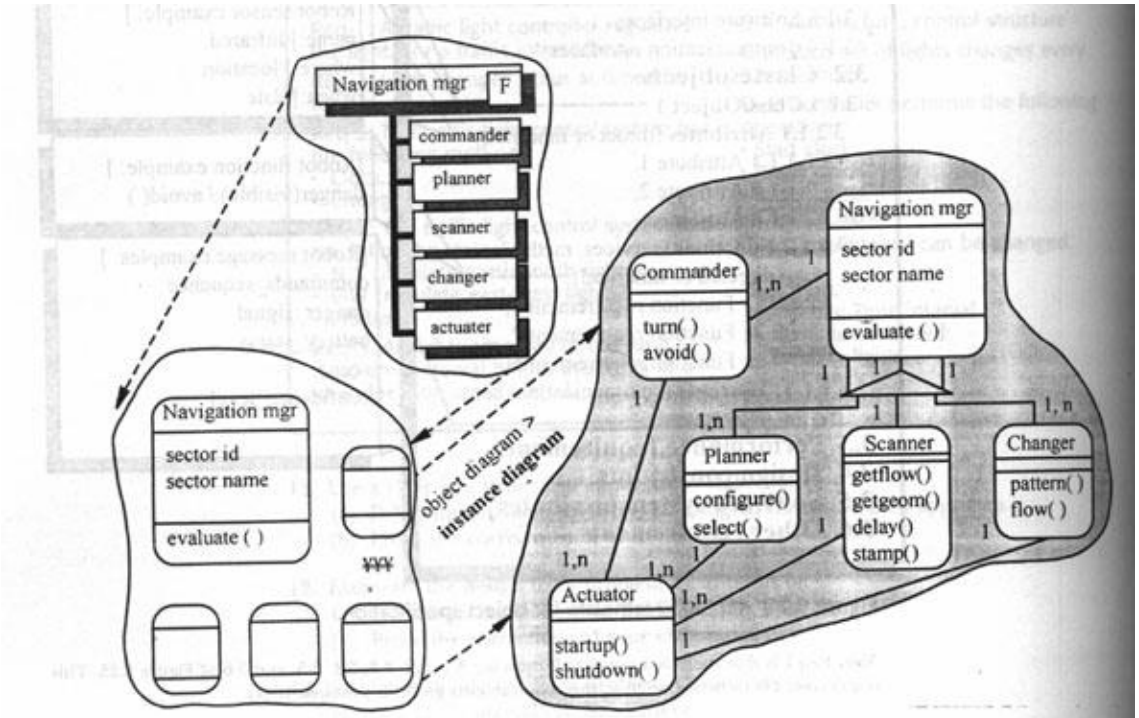
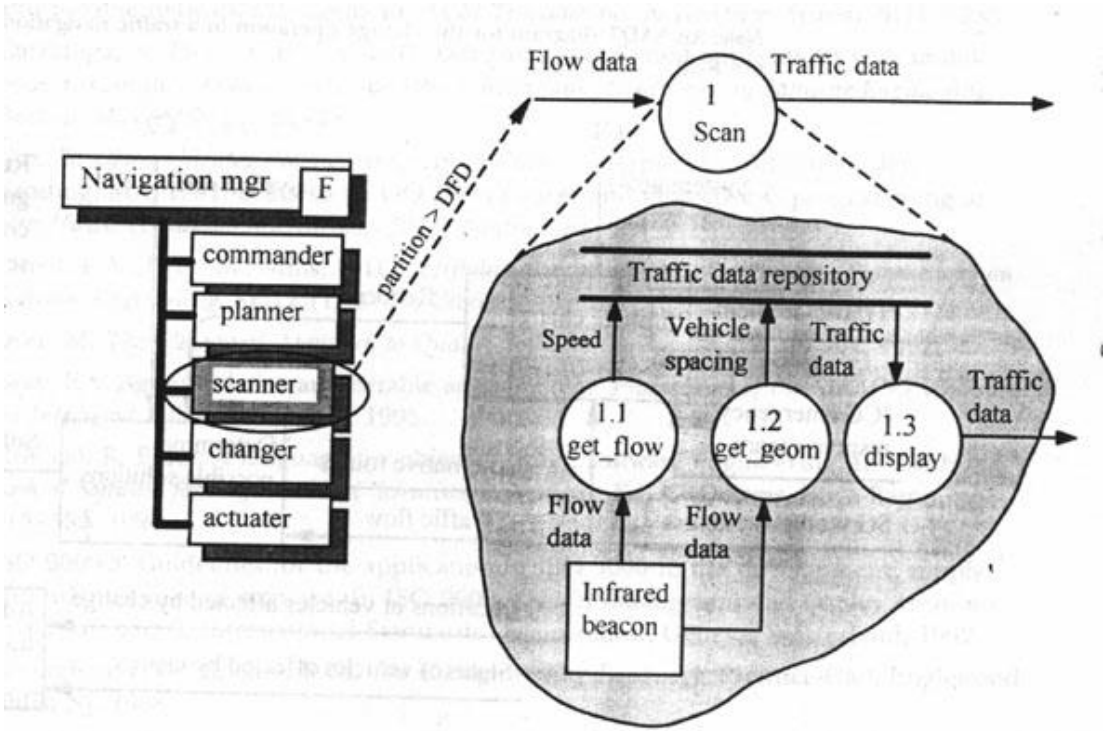


Figure 8.26 Object diagram for a traffic navigation manager.

Structured Model



Comparison

OO:

Systems **decomposed into collections of data objects**; function + data in one place =>

- System components more independent => more resilient to requirements and maintenance changes.
- Inheritance and polymorphism are possible => reuse, extension, and tailoring of software/designs is possible.
- Closely mirrors how humans decompose and solve complex.

Structured:

Systems **decomposed into functions**; functions and data modelled separately =>

- System components are more dependent on each other => requirements and maintenance changes more difficult
- Inheritance and polymorphism not possible => limited reuse possible.
- System components do not map closely to real-world entities => difficult to manage complexity.

Comparison

OO:

Process allows for **iterative** and **incremental** development =>

- Integration of programs is series of incremental prototypes.
- Users and developers get important feedback throughout development.
- Testing resources distributed more evenly.
- If time is short, coding and testing can begin before the design is finished.

Structured:

Process **less flexible** and largely **linear** =>

- Integration of programs is ‘big bang’ effect.
- Users or developers provided with little or no feedback; see system only when it has been completed.
- Testing resources are concentrated in the implementation stage only.
- Coding and testing cannot begin until all previous stages are complete.

Fig. 1.6

- 3 ways to use UML
  - Sketch
    - Very quick, requires no fancy tools
    - Preferred by Agile modelers
    - Often good enough
  - Blueprint
    - Detailed model drawn by tool
    - Use to generate code
    - Reverse engineer from code

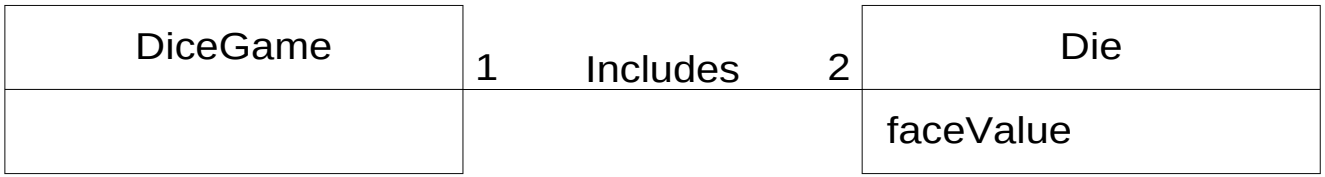
Fig. 1.6

- 3 ways to use UML
  - Executable programming language
    - Prepare complete suite of models for system
    - Compile models directly into executable code
    - Eliminates costly coding activity
    - Not ready for ‘primetime’

Fig. 1.6

3 perspectives of UML modeling

Conceptual class



Conceptual Perspective (domain model)

Raw UML class diagram notation used to visualize real-world concepts.

Specification or Implementation Perspective (design class diagram)

Raw UML class diagram notation used to visualize software elements.

SW or Implementation class

Implementation perspective uses

Language-specific syntax