# Design

# What is “design”?

*activity that gives **structure** to the solution to a given problem*

- takes us from a requirements statement to an implementation


  - analysis has some design activities
  - design touches implementation issues

# What is design generally?

- **a Synthesis**: the combination of things to build a larger whole
- **an Analysis**: the examination of things to understand their important aspects
- Common types of *synthesis* in design:
  - Selection of elements from a library to add to your system
  - Invention of new elements
  - Composition of elements in your system
  - Choice of parameters of elements
- Common types of *analysis* in design:
  - Syntactic and semantic checks
  - Heuristic review by experts
  - Comparison to past experience
  - Design checklists
  - Simulation
  - Empirical analysis (actual measurements)

# Design challenges

- Cognitive challenges:
  - Limited knowledge
  - Biases (expertise, experience, etc.)
- Usability challenges of current tools
- Process involving many stakeholders with intersecting interests

# Design outputs

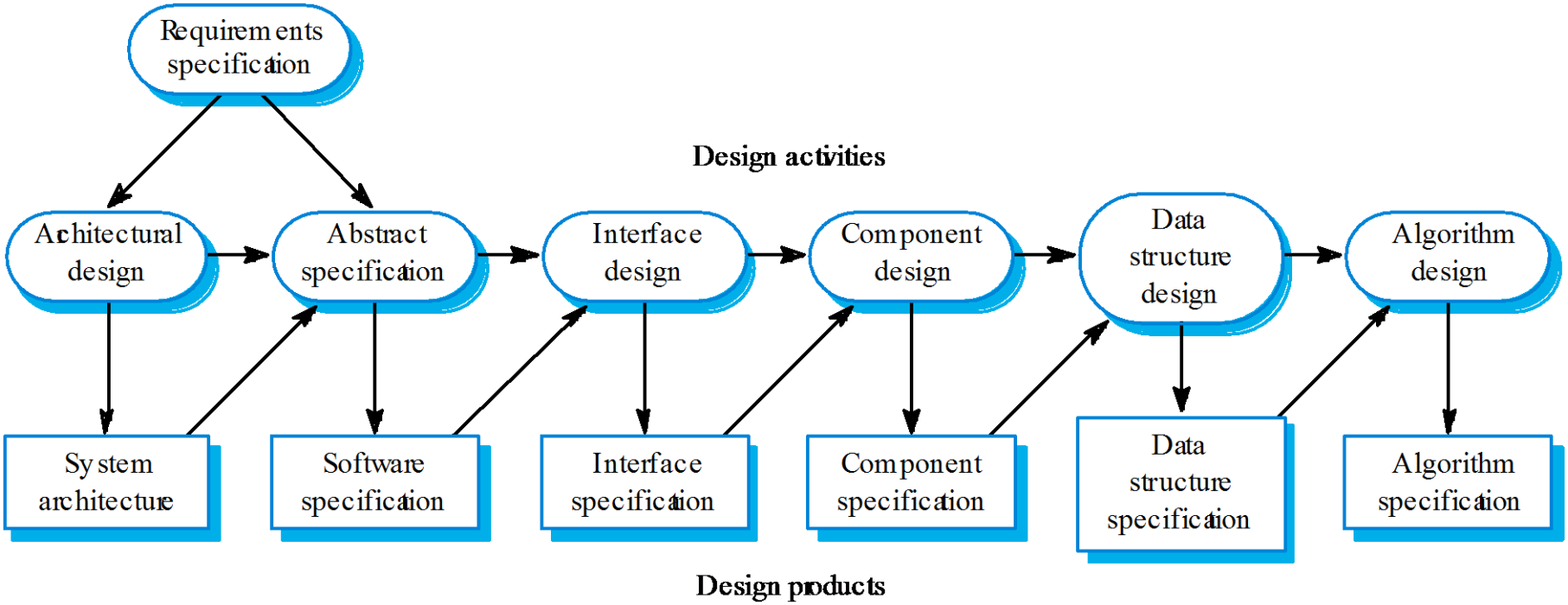
- Primary output: **architecture**

  *definition of **abstractions** of interest in the system and of the relations between them*

  – **abstractions**: objects, processes, tuples, components

    - [abstractions are actually concrete]
    - may not be continuous (physically): a process (e.g. a use case) comprises procedures scattered among various objects

# The software design process



# Object-oriented approach

- OO analysis
  - develop an object model of the application domain
- OO design
  - develop an object-oriented system model to implement requirements
- OO programming
  - realize an OO design using an OO programming language such as Java, C++, C#

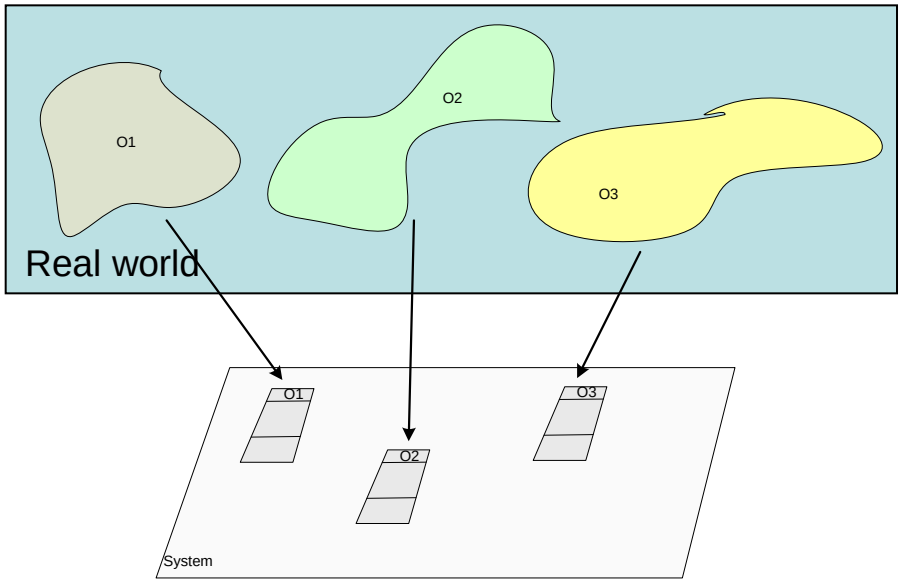
# Object oriented design

# Characteristics

- Objects: abstractions of real-world or system entities
  - manage themselves
  - independent
  - encapsulate state and representation information
- System functionality – expressed in terms of object services
- Features:
  - shared data areas are eliminated
  - objects communicate by message passing
  - objects may be distributed and may execute sequentially or in parallel

# Advantages of OO design

- easier maintenance
- objects may be seen as stand-alone entities
- objects are potentially reusable components
- usually *there is an obvious mapping from real world entities to system objects*



# Objects and classes

- Objects
  - entities in a software system
  - represent instances of real-world and system entities
- Object classes
  - templates for objects
  - may be used to create objects
  - may inherit attributes and services from other object classes

# Objects and classes (Sommerville)

- *An **object** is an entity that has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.*

- *Objects are created according to some **object class** definition. An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.*

# Object communication

- Conceptually, objects communicate by **message passing**
- Message – request for a service
- Message content
  - The name of the service requested by the calling object
  - Copies of the information required to execute the service
  - The name of a holder for the result of the service.
- Usually messages are implemented as procedure (function) calls
  - Name = procedure name;
  - Information = parameter list.

# Objects relationships

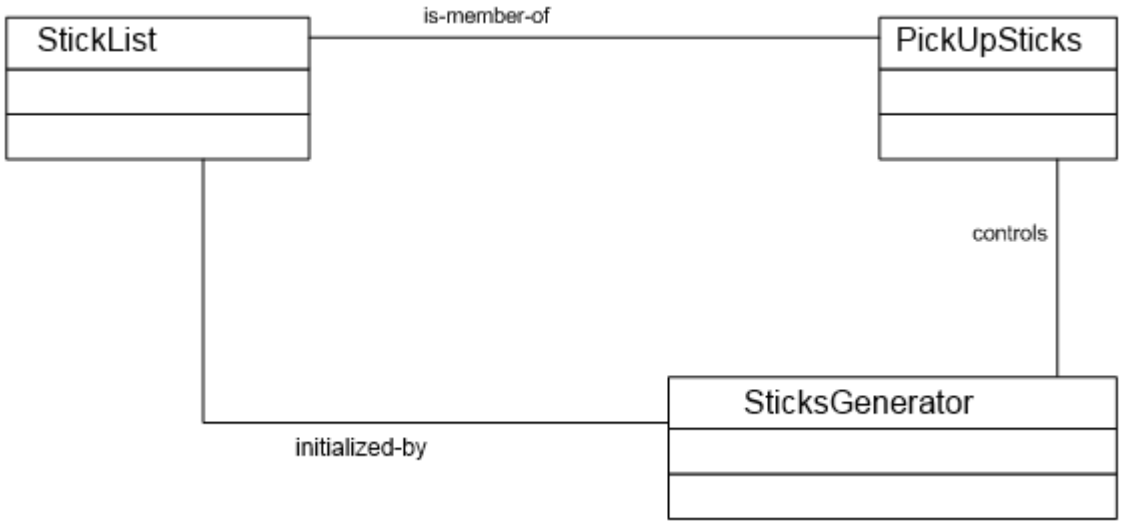
- Inheritance
- Aggregation
- ...
- *Associations: may indicate that an attribute of an object is an associated object or that a method relies on an associated object*

# Objects relationships

- may be annotated with information that describes the association

**PickUpSticks**

**Object relationships (excerpt)**



# Object execution

- “execution” – an object gets hold of a processor and puts it to run some *code*
- the code *belongs to a function* of the object (a method)
- usually this executi on takes places inside a *process*
- serial execution
  - the process starts, executes some code, ends
  - also called *single-threaded*

# Threads and objects

- Thread
  - describes a path of execution within a process
- Multithreaded programs
  - Programs that run as processes which start more than one thread
  - The threads compete to grab system resources (processor time)
  - Each thread may consist of functions that span several objects

# Objects and concurrency

- Concurrency: objects may execute in parallel
- A system may consist of more than one process
- Interaction is possible: via messaging
- *Function call* is just an instance of messaging

# Communication

- Synchronous:
  - Call / send message
  - Wait for the callee to complete
- Asynchronous:
  - Call / send message
  - Execute further code

# Types of concurrent objects

- Servers
  - The object runs as a parallel process
  - Methods start after an external message is received
  - After finishing the operation, the server suspends itself
- Active objects
  - The object runs as a parallel process
  - Its state may change by internal operations
  - The process never suspends itself

# Object oriented design processes

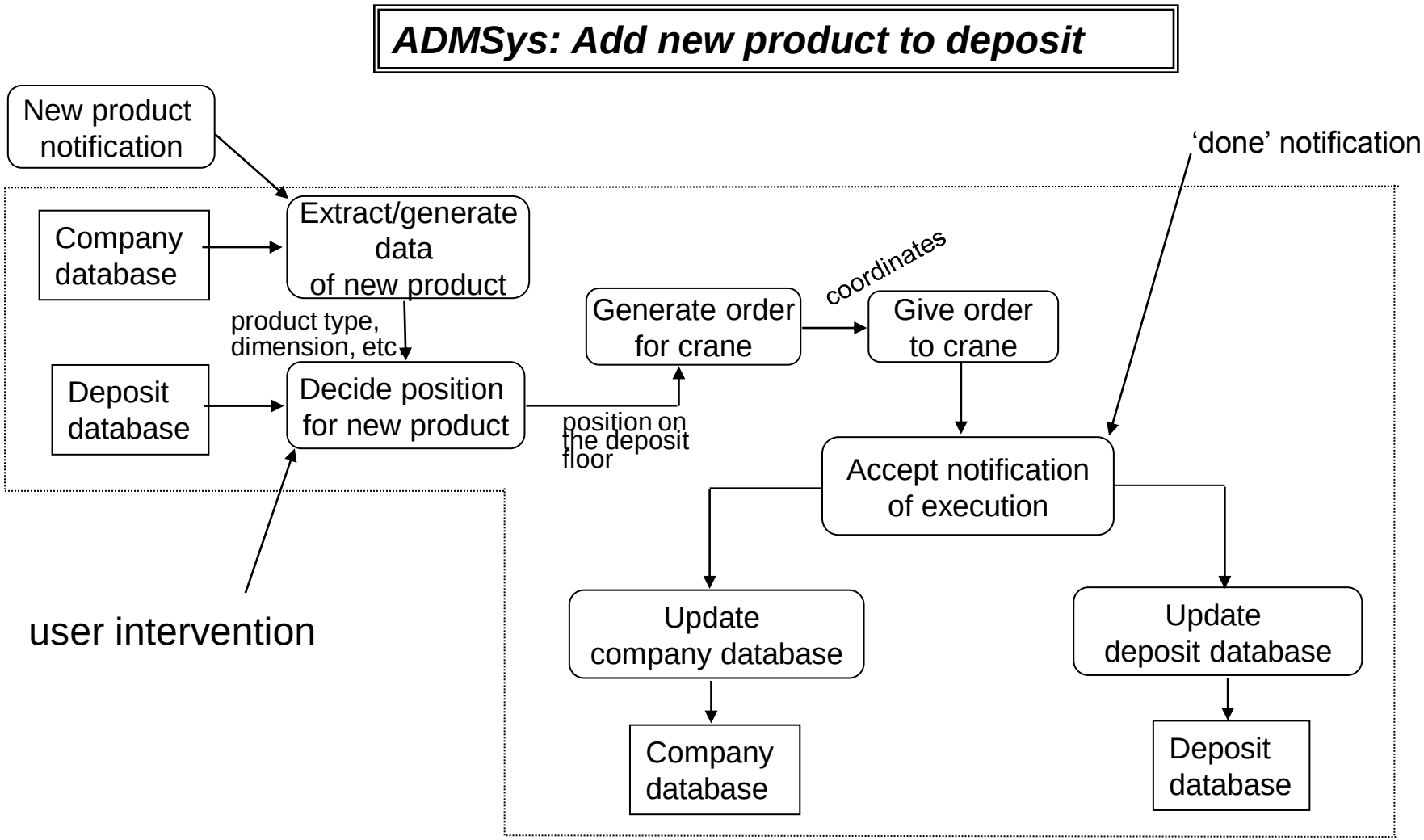
# Process stages

1. Define the context and models of use of the system;
2. Design the system architecture;
3. Identify the principal system objects;
4. Develop design models;
5. Specify object interfaces.

# System context and models of use

- detect the relationships between the software being designed and its external environment
- *system context*
  - static model
  - mention other systems in the environment
  - subsystem model: show parts that make up the system
- models of *system use*
  - dynamic model
  - describes how the system interacts with its environment
  - use-cases:
    - show interactions
    - described in natural language
  - process models
    - diagrams with the most important processing steps

# Models of system use

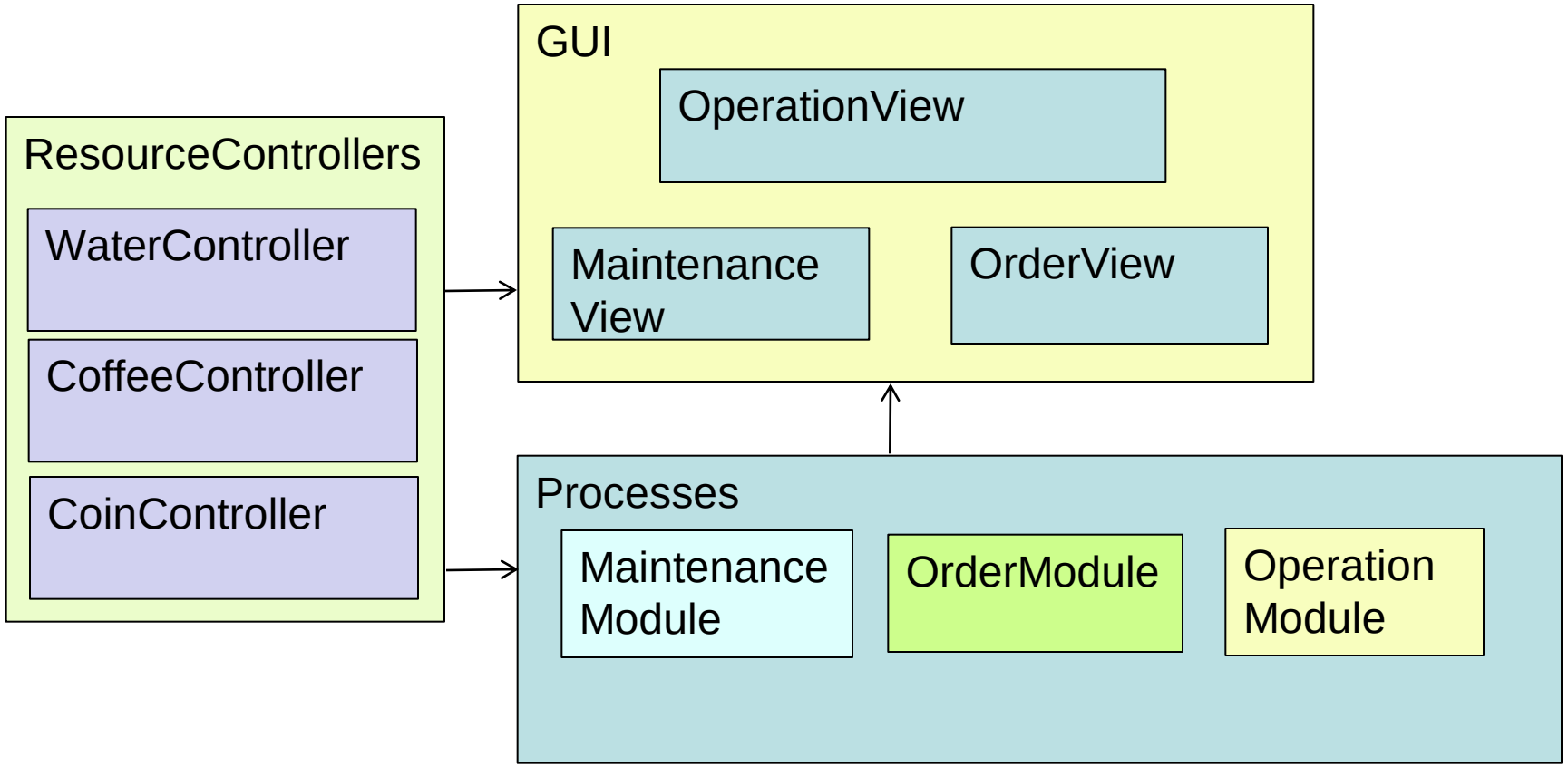


# Architectural design

- use the knowledge about interactions between the system and its environment to design the system architecture
- architecture should be simple: keep the number of entities in an architectural model low

# Example

Simulation of a coffee machine
System architecture



# Example

Simulation of a coffee machine
System architecture
(continued)

GUI
-Provides visualization services
1. Displays the shape of the coffee machine
2. Displays the dynamic of processed
    1. Ordering
    2. Maintenance
    3. Operation (preparation of beverages)

    …

ResourceControllers
-Provides control for resources
1. Water
2. Coffee
3. Coins

- Water Controller
    1. Maintains current water volume
    2. Allows filling (through Maintenance)
    3. Allows removal of specific quantities (through Operation)

    …

# Object identification

- the most difficult part of OO design
- no 'magic formula' for object identification
- use your skills, experience and domain knowledge
- iterative process – you may not get it right first time

# Approaches to identification

- grammatical approach:
  - based on a natural language description of the system
  - nouns – objects and attributes
  - verbs – operations and services
- identification of tangible things in the application domain
  - objects, roles, events, interactions, locations, organisational units
- behavioural approach
  - understand first the overall behaviour of the system
  - assign behaviours to parts
  - identify objects based on what participates in what behaviour
- scenario-based analysis
  - identify objects, attributes and methods in each scenario

# Object identification: practicalities

- Extract some initial objects
- Refine the structure by extending them and adding more objects
- Strategies
  - Top-down
  - Decomposition at equal level of abstraction

# Design models

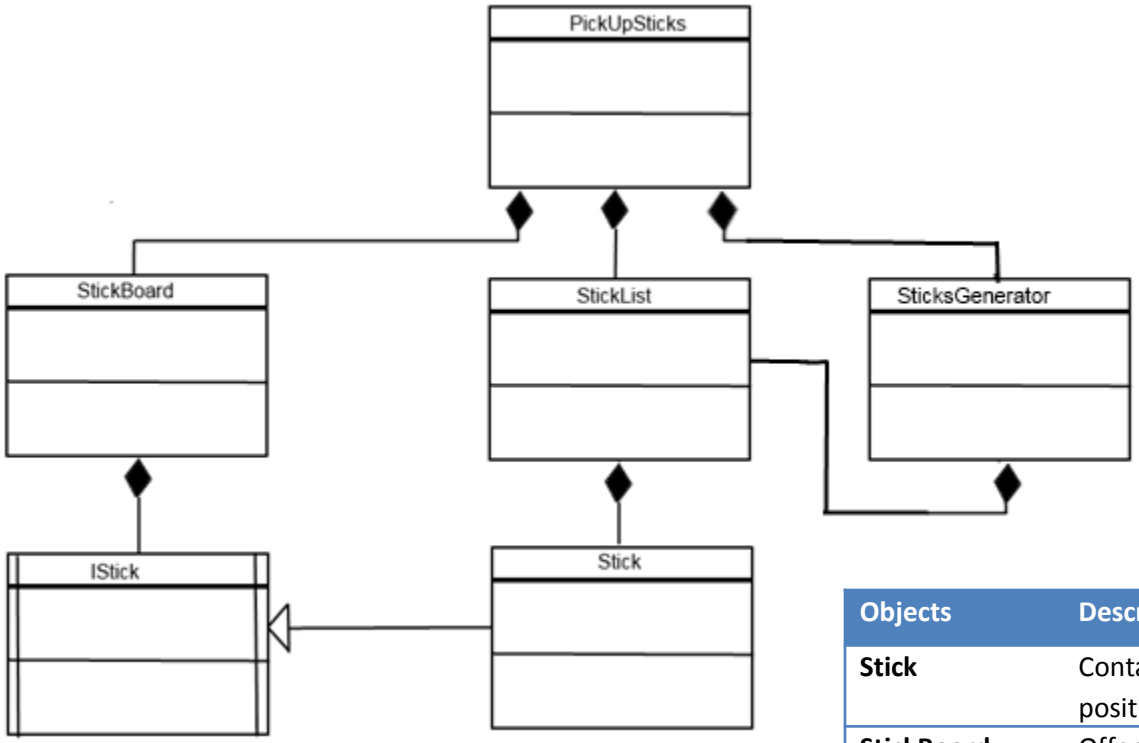
- show the *objects and object classes* and *relationships* between them
- Static models
  - describe the static structure of the system in terms of object classes and relationships
- Dynamic models
  - describe the dynamic interactions between objects.

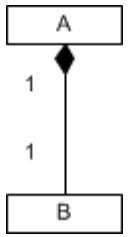
# Examples of design models

- Sub-system models
  - show logical groupings of objects into coherent subsystems
- Sequence models / diagrams
  - show the sequence of object interactions
- State machine models
  - show how individual objects change their state in response to events
- Other models
  - use-case models
  - aggregation models
  - generalization models

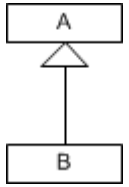
# Example

## PickUpSticks Object Model




An instance of class A contains an instance of class B as member

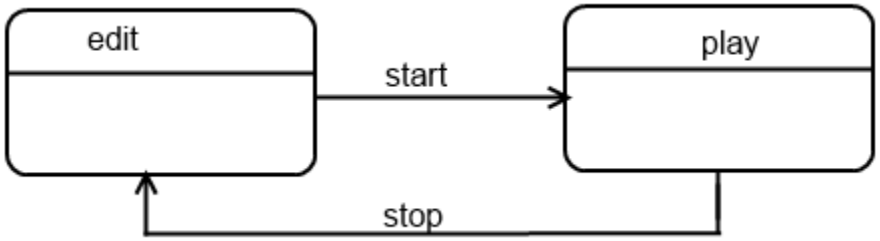



Class B inherits class A, or class B implements interface A.

| Objects | Description |
|---|---|
| **Stick** | Contains information about one stick (color, position, orientation( |
| **StickBoard** | Offers a visualization of the list of sticks. Collects mouse events. |
| **…** | |

# Example

PickUpSticks
State machine diagram




| States | |
|---|---|
| **"edit"** | In 'edit' state, the system allows the user to edit parameters of the game, generate a new set of sticks, and view hall of fame. |
| **"play"** | In 'play' state, the system allows the user to remove sticks by clicking on them. While in 'play' state, it is not possible for the user to modify game settings or to generate a new set of sticks. |
| **Events** | |
| **"start"** | The user clicks on the "Start playing" button. |
| **"stop"** | The user clicks on the "Stop" button. |

# Object interface specification

- needed for designing in parallel objects and components
- objects may have several interfaces
  = viewpoints on the methods provided.
- details of the implementation of an interface should not be specified

- an interface is a *contract*
  – no matter the implementation, the class that represents it is assumed to fulfil the contract.

- Next delivery:
  - A set of models
    - System architecture
    - *Process* and/or *Data flow* and/or *State machine* and/or *Data models* and/or *Sequence diagrams*
    - Object models with textual descriptions of classes.

*(you should describe the most important abstractions in your project within at least 4 models)*

*(do not forget: textual descriptions of all diagram elements!)*

  - Snapshot(s) / sketches of the GUI

You can use e.g. Visual Paradigm for drawing the models

- Deadline: 2017 May 25

# Homework

Consider the module diagram of ADMSys, on slide 20 of lecture 6.

Assuming that there is an object that realizes the crane controller, figure out some of its most important services (public methods).

(Please be aware that there are at least two different schemes for the services offered by a crane controller.)