SOFTWARE DESIGN & ARCHITECTURE

(Lecture-5)

USAMA MUSHARAF

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

CONTENT

- Design Principles & Concepts
- Object Oriented Concepts
- Object Oriented Design with a Case Study

DESIGN PRINCIPLES

DESIGN PRINCIPLES

1- The design process should not suffer from “tunnel vision.”

A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job, and the design concepts.

2- The design should be traceable to the analysis model.

DESIGN PRINCIPLES

3- The design should not reinvent the wheel.

- Systems are constructed using a set of design patterns.
- These patterns should always be chosen as an alternative to reinvention.
- Time is short and resources are limited!

DESIGN PRINCIPLES

4- The design should “minimize the intellectual distance” between the software and the problem as it exists in the real world.

DESIGN PRINCIPLES

5- The design should be structured to accommodate change

6- The design should be assessed for quality as it is being created

DESIGN CONCEPTS

FUNDAMENTAL CONCEPTS OF DESIGN

- **abstraction**—data, procedure, control
- **refinement**—elaboration of detail for all abstractions
- **modularity**—compartmentalization of data and function
- **architecture**—overall structure of the software
  - Styles and patterns
- **procedure**—the algorithms that achieve function
- **hiding**—controlled interfaces

ABSTRACTION

"Capture only those details about an object that are relevant to current perspective"

Suppose we want to implement abstraction for the following statement,

"Ali is a PhD student and teaches BS students"

Here object Ali has two ***perspectives*** *one is his* ***student perspective*** *and second is his* ***teacher perspective.***

ABSTRACTION

A cat can be viewed with different perspectives.

Ordinary Perspective	Surgeon's Perspective
A pet animal with	A being with
Four Legs	A Skeleton
A Tail	Heart
Two Ears	Kidney
Sharp Teeth	Stomach

ABSTRACTION

Driver's View

Engineer's View

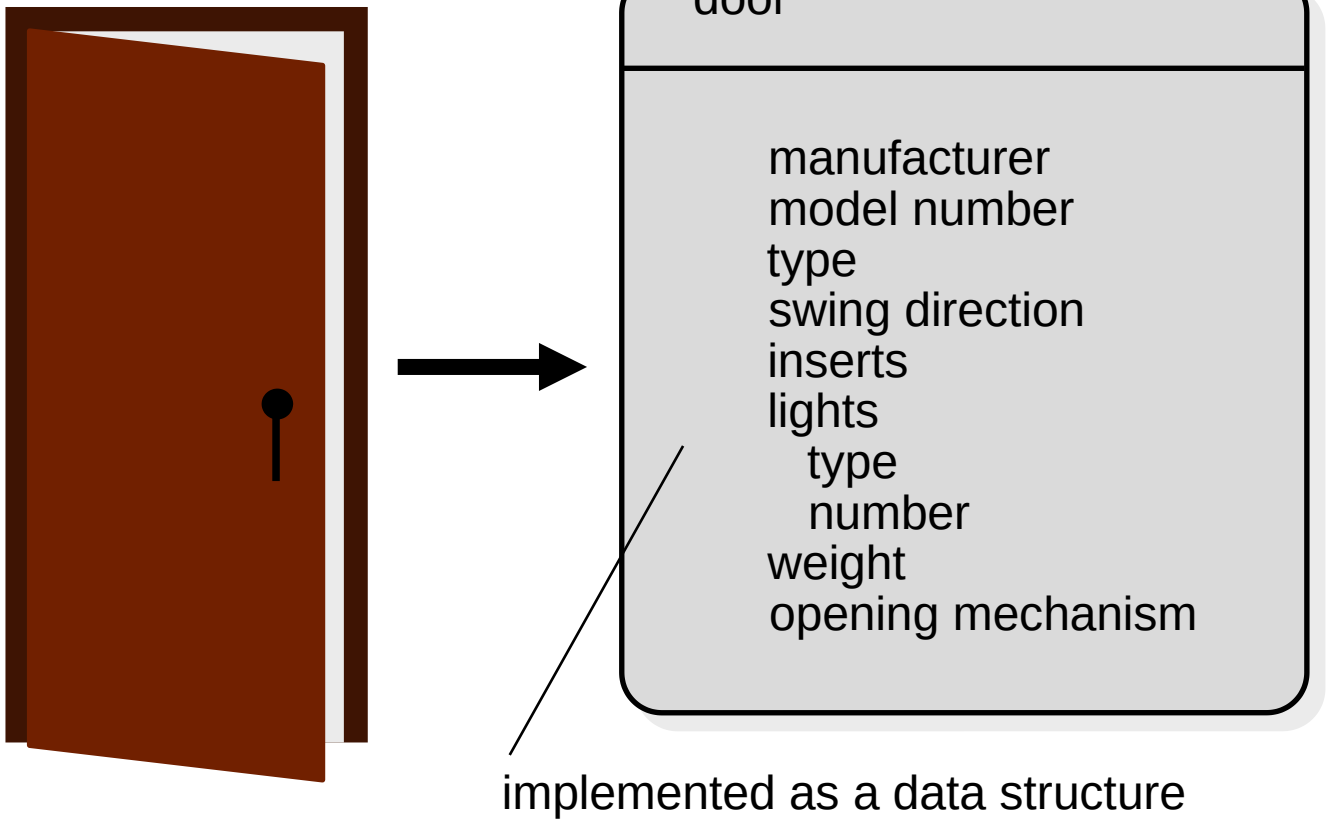
ABSTRACTION ADVANTAGES

- It helps us understanding and solving a problem using object oriented approach as it hides extra irrelevant details of objects.

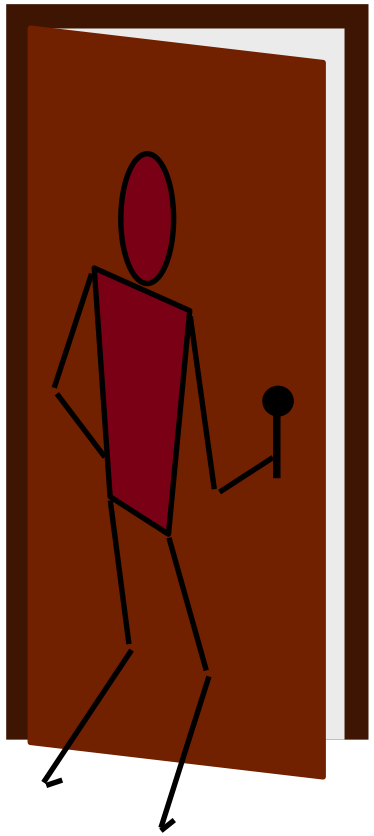
- Focusing on single perspective of an object provides us freedom to change implementation for other aspects of for an object later.

- *Abstraction is used for achieving information hiding as we show only relevant details to related objects, and hide other details.*

DATA ABSTRACTION



PROCEDURAL ABSTRACTION

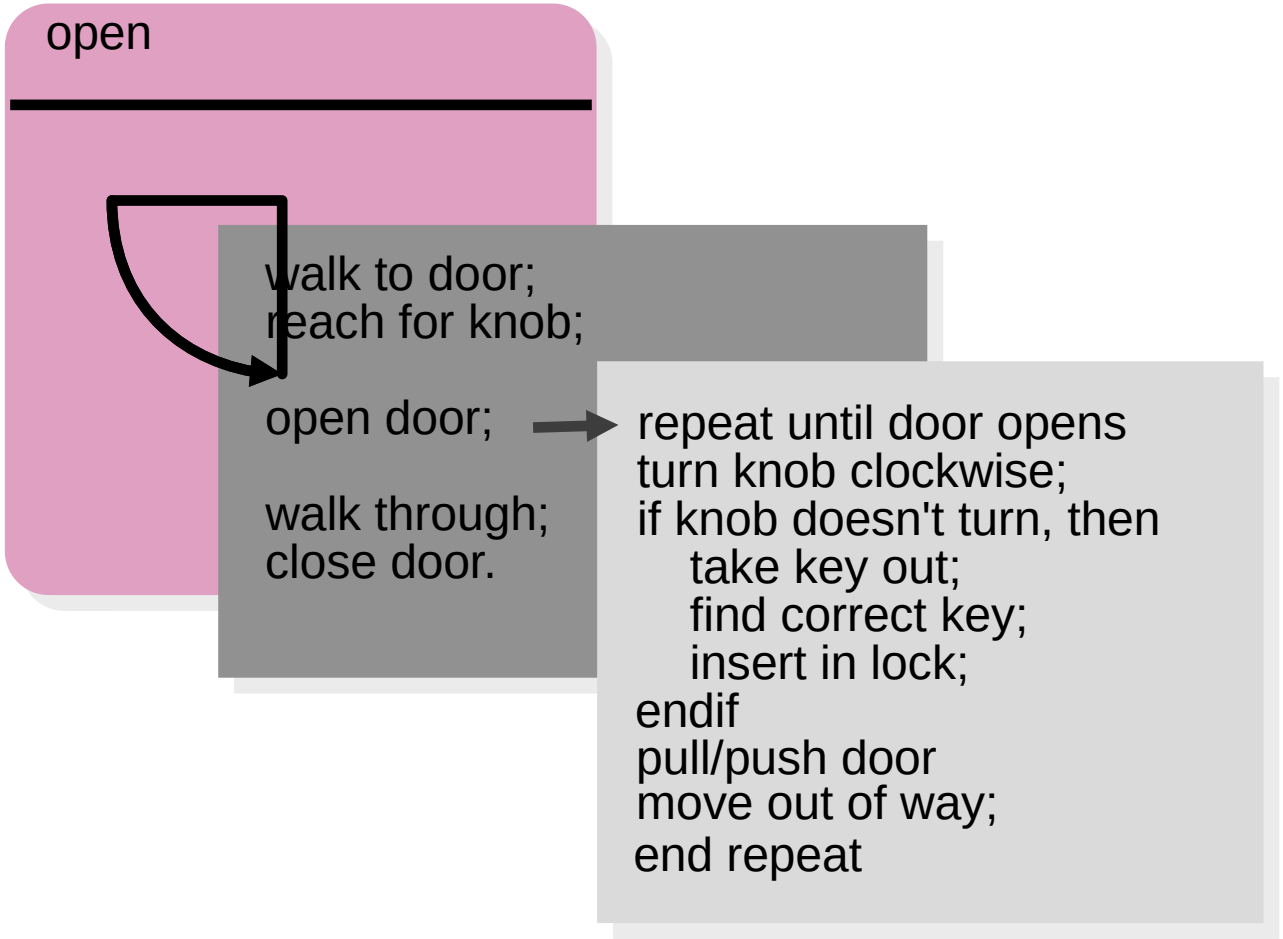


open

details of enter algorithm

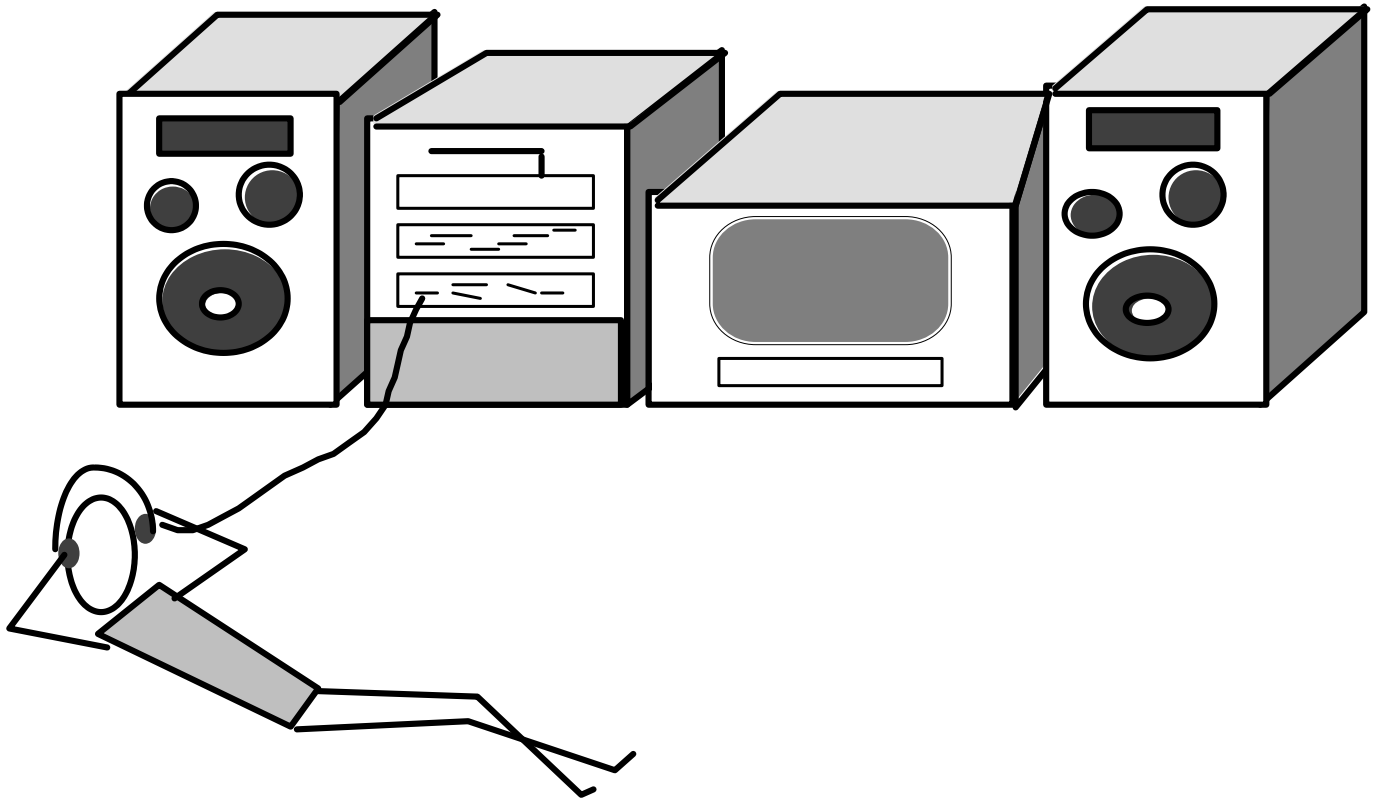
implemented with a "knowledge" of the object that is associated with enter

STEPWISE REFINEMENT



MODULAR DESIGN

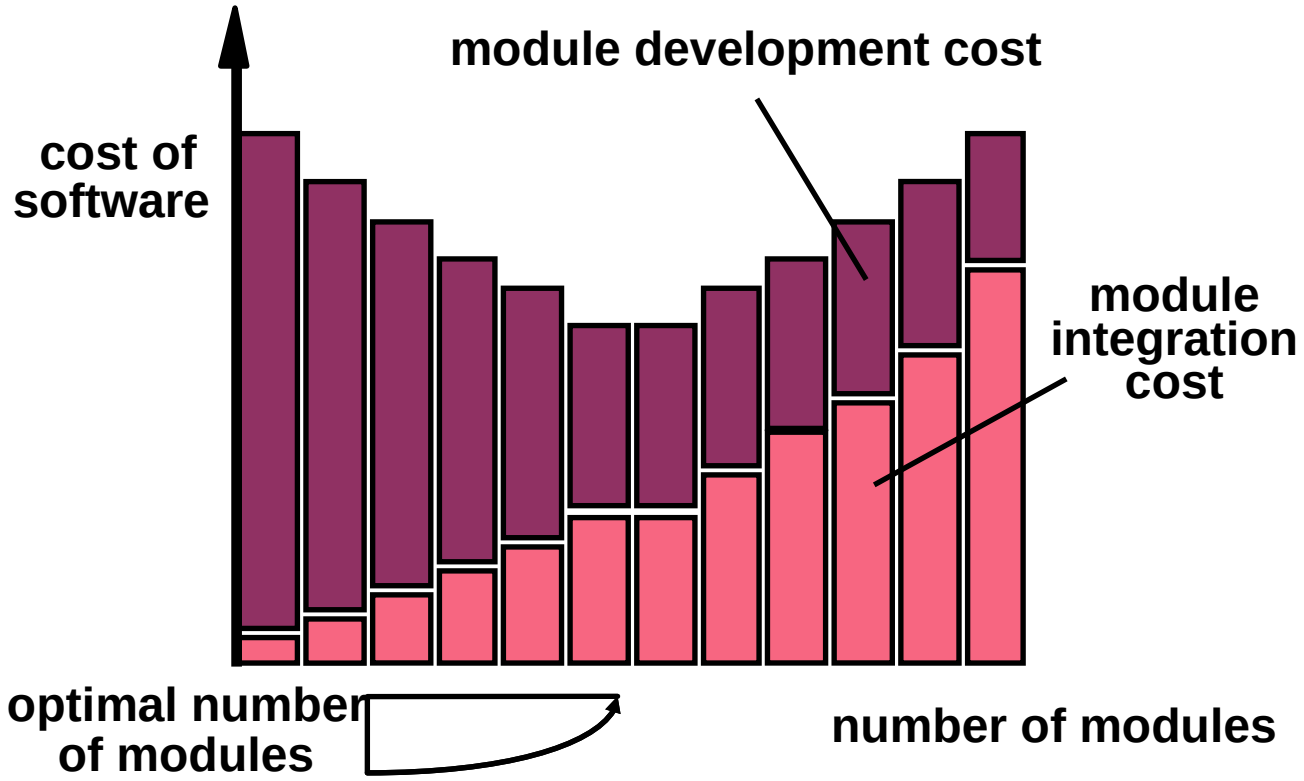
easier to build, easier to change, easier to fix …



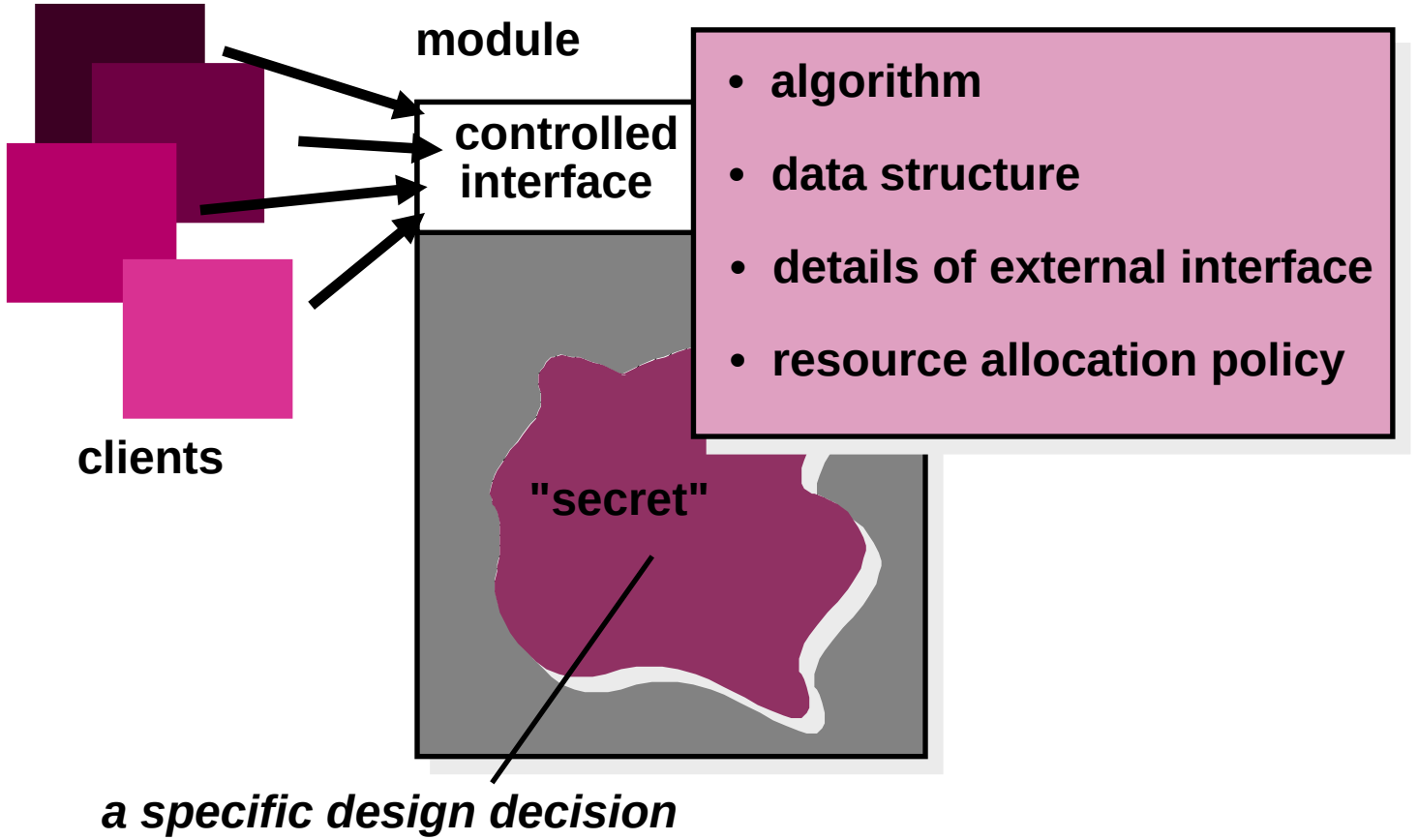
- **Easier to manage**
- **Easier to understand**
- **Reduces complexity**
- **Delegation / division of work**
- **Fault isolation**
- **Independent development**
- **Separation of concerns**
- **Reuse**

MODULARITY: TRADE-OFFS

What is the "right" number of modules for a specific software design?



INFORMATION HIDING



INFORMATION HIDING

- Design the modules in such a way that information (data & procedures) contained in one module is inaccessible to other modules that have no need for such information.
- Independent modules.

Benefits:

when modifications are required, it reduces the chances of propagating to other modules.

EFFECTIVE MODULAR DESIGN

FUNCTIONAL INDEPENDENCE

COHESION - the degree to which a module performs one and only one function.

COUPLING - the degree to which a module is "connected" to other modules in the system.

COUPLING

Coupling is a measure of independence of a module or component.

Loose coupling means that different system components have loose or less reliance upon each other.

Hence, changes in one component would have a limited affect on other components.

COUPLING

High coupling causes problems

- Change propagation- ripple effect
- Difficulty in understanding
- Difficult reuse

COHESION

Cohesion is a measure of the degree to which the elements of the module are functionally related.

It is the degree to which all elements directed towards performing a single task are contained in the component.

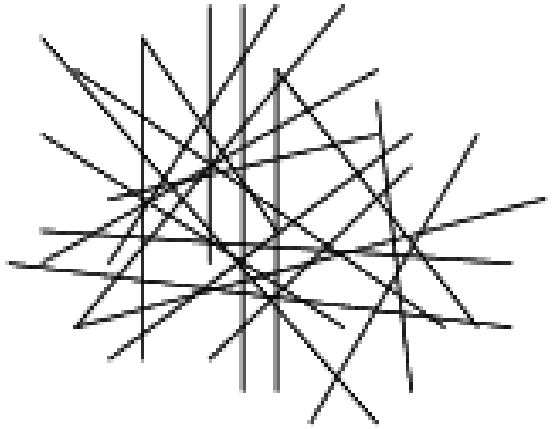
Basically, cohesion is the internal glue that keeps the module together.

A good software design will have high cohesion

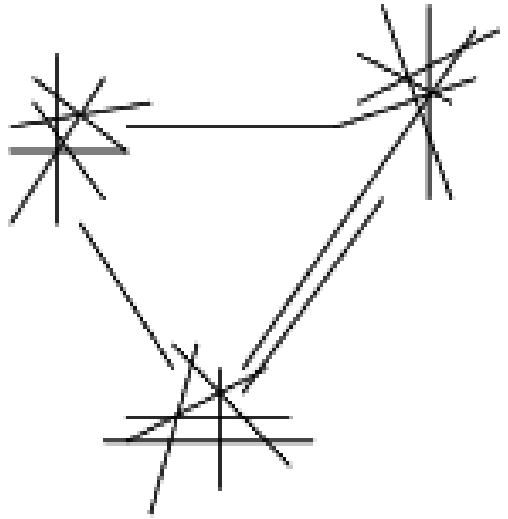
COUPLING & COHESION

A Software should be Lesly coupled and highly cohesive.

COUPLING

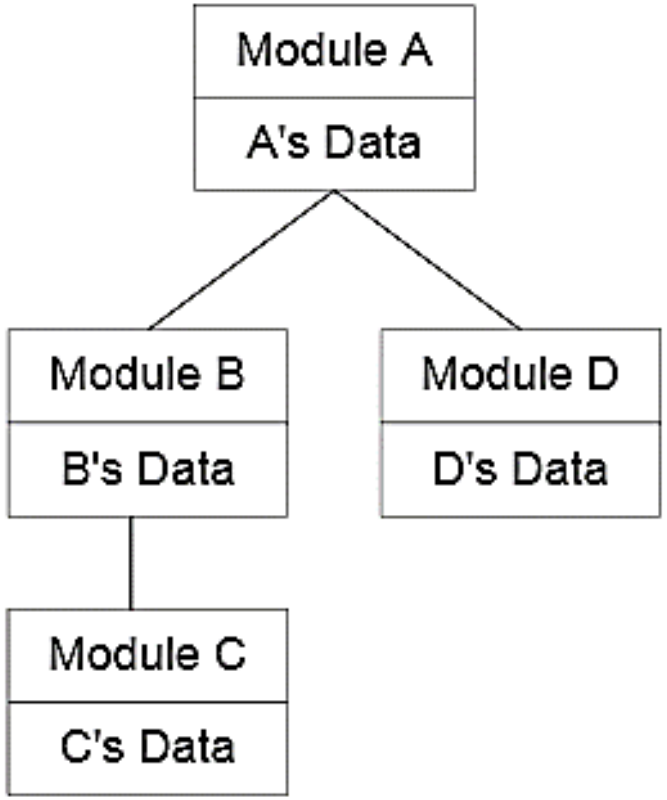


High Coupling

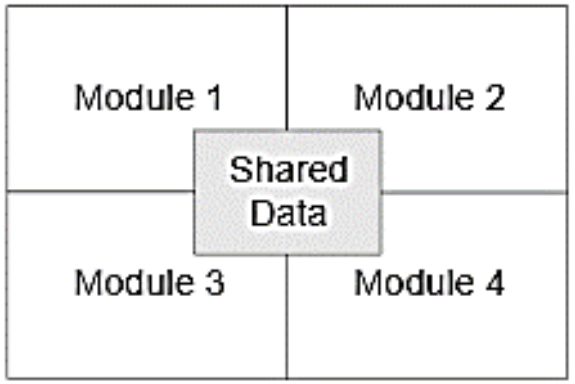


Low Coupling

COUPLING

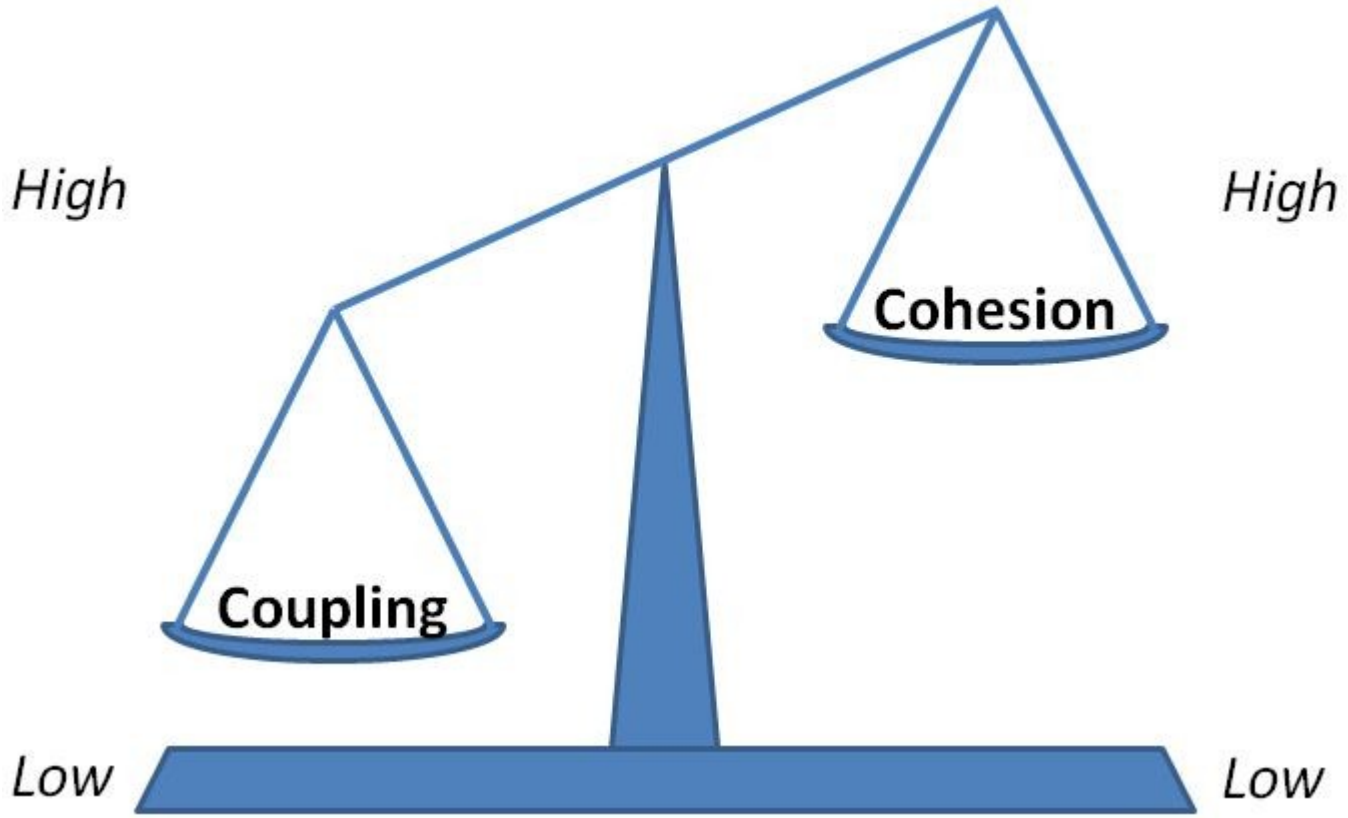


Low Coupling



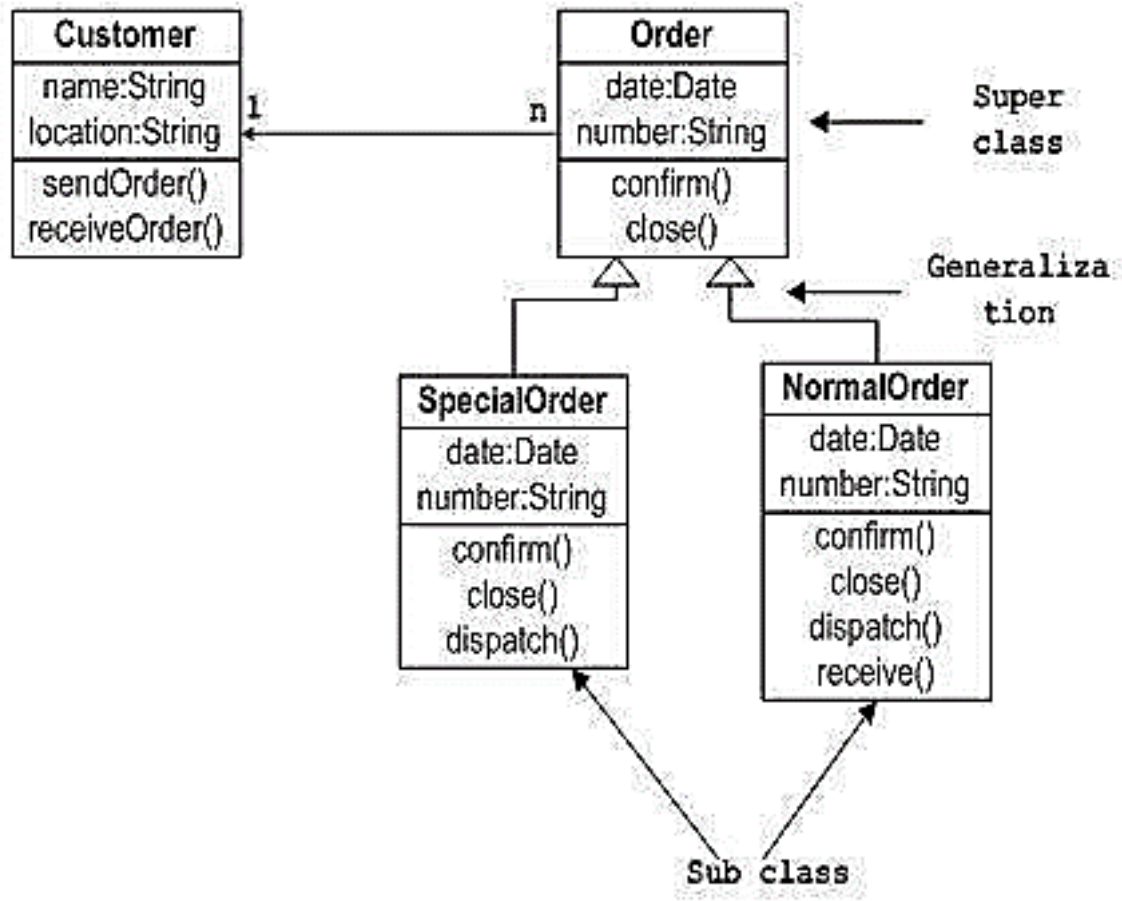
High Coupling

RELATIONSHIP BETWEEN COUPLING AND COHESION

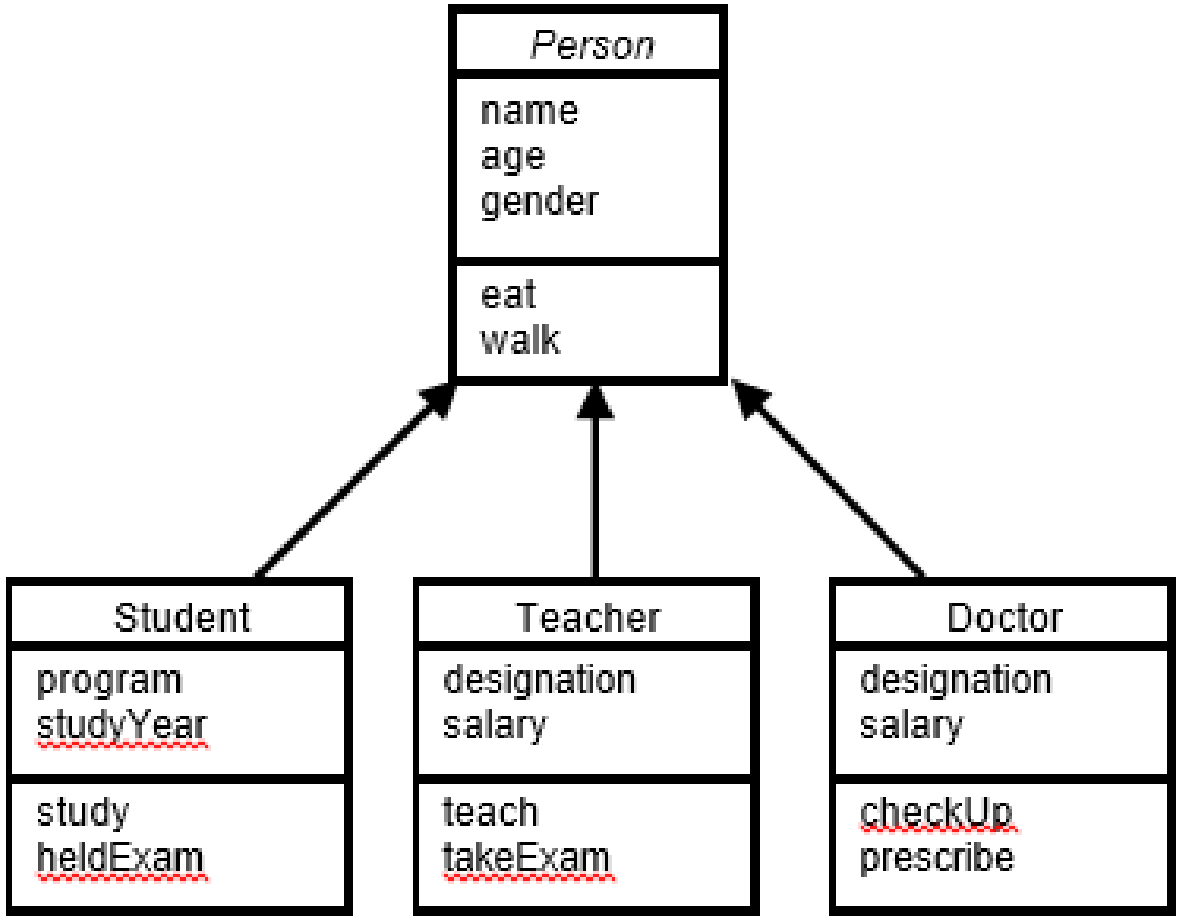


OBJECT ORIENTED DESIGN CONCEPTS

CLASS DIAGRAM



INHERITANCE



Advantages of Inheritance

1. *Reuse*
2. *Less redundancy*
3. *Increased maintainability*

KINDS OF ASSOCIATION

There are two main types of association which are then further subdivided i.e.

- Class Association
- Object Association

CLASS ASSOCIATION

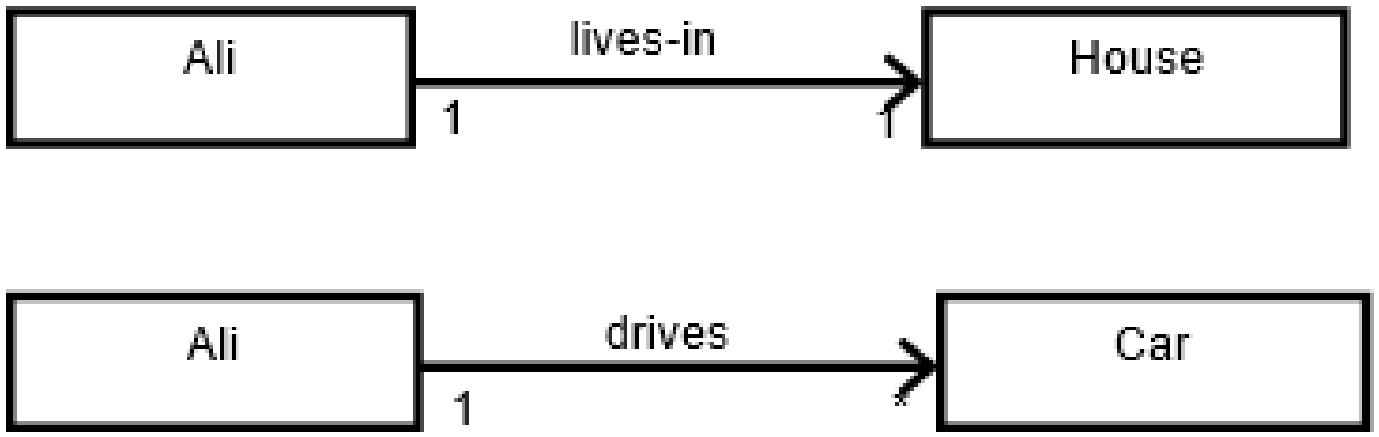
- *Class association is implemented in terms of Inheritance.*
- *Inheritance implements generalization/specialization relationship between objects.*
- *Inheritance is considered class association.*

OBJECT ASSOCIATION

It can be of one of the following types,

- *Simple Association*
- *Composition*
- *Aggregation*

SIMPLE ASSOCIATION



It is generally called as "**association**" instead of "**simple association**"

KINDS OF SIMPLE ASSOCIATION

Simple association can be categorized in two ways,

- With respect to direction (navigation)
- With respect to number of objects (cardinality)

KINDS OF SIMPLE ASSOCIATION W.R.T NAVIGATION

With respect to navigation association has the following types,

- One-way Association
- Two-way Association

ONE-WAY ASSOCIATION

In one-way association we can navigate along a single direction only, it is denoted by an arrow towards the server object.

- **Examples:**

- Ali lives in a House

- Ali drives his Car

TWO-WAY ASSOCIATION

In two-way association we can navigate in both directions, it is denoted by a line between the associated objects.

Exa

Employee works for company
Company employs employees

KINDS OF SIMPLE ASSOCIATION W.R.T CARDINALITY

With respect to cardinality association has the following types,

- Binary Association
- Ternary Association
- N-ary Association

BINARY ASSOCIATION

It associates objects of exactly two classes; it is denoted by a line, or an arrow between the associated objects.

Example

Association "works-for" associates objects of exactly two classes

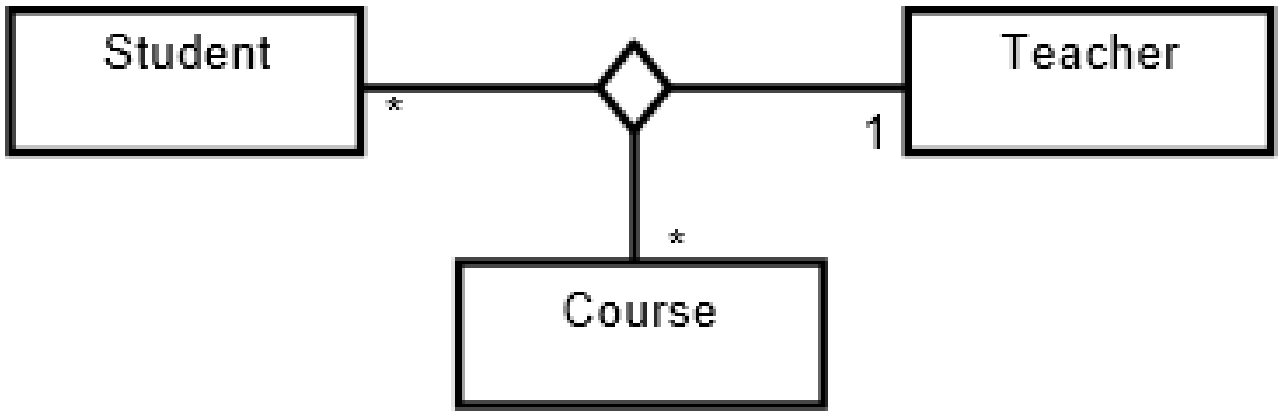
Association "drives" associates objects of exactly two classes

TERNARY ASSOCIATION

It associates objects of exactly three classes; it is denoted by a diamond with lines connected to associated objects.

Example:

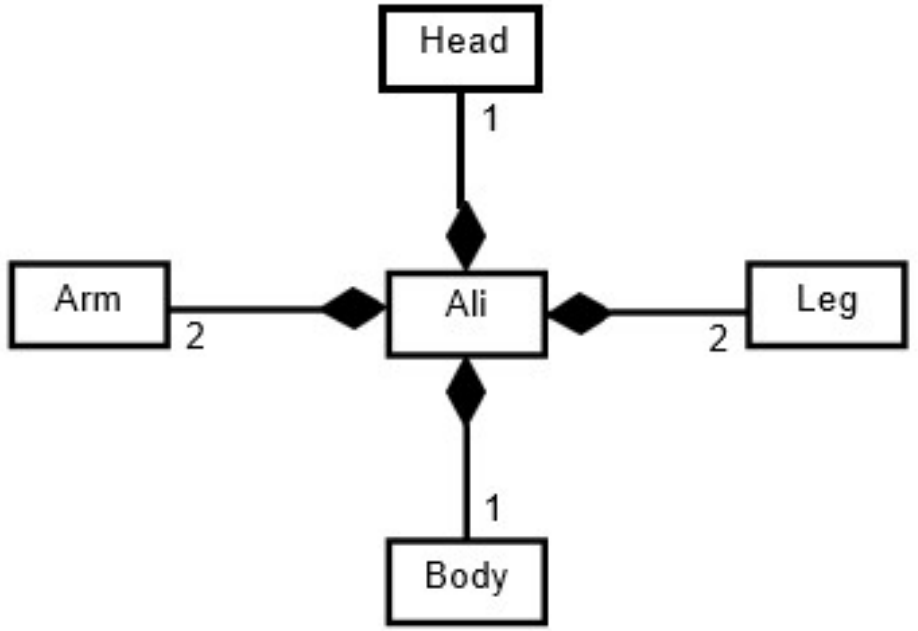
Objects of exactly three classes are associated



COMPOSITION

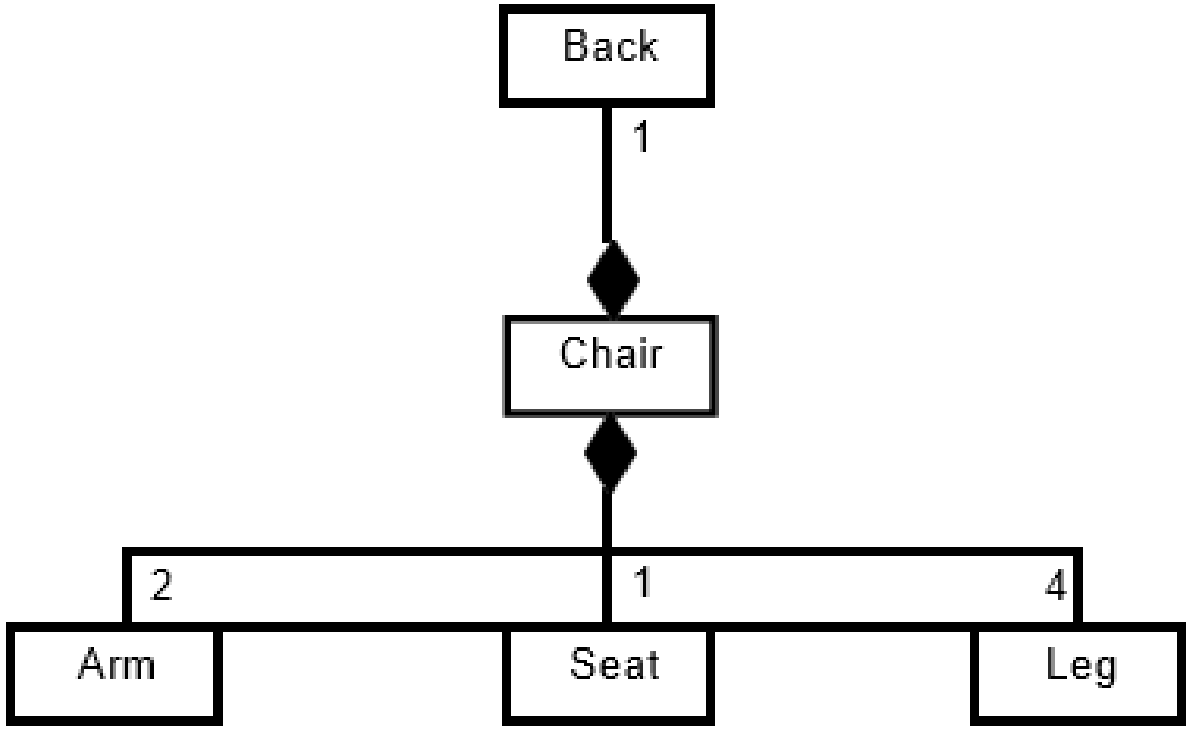
An object may be composed of other smaller objects, the relationship between the “part” objects and the “whole” object is known as Composition.

Composition is represented by a line with a filled-diamond head towards the composer object.



COMPOSITION

Composition of chair:



COMPOSITION

Composition is stronger relationship because

- *Composed object becomes a part of the composer.*
- *Composed object can’t exist independently.*

Example I

- Ali is made up of different body parts They can’t exist independent of Ali

Example II

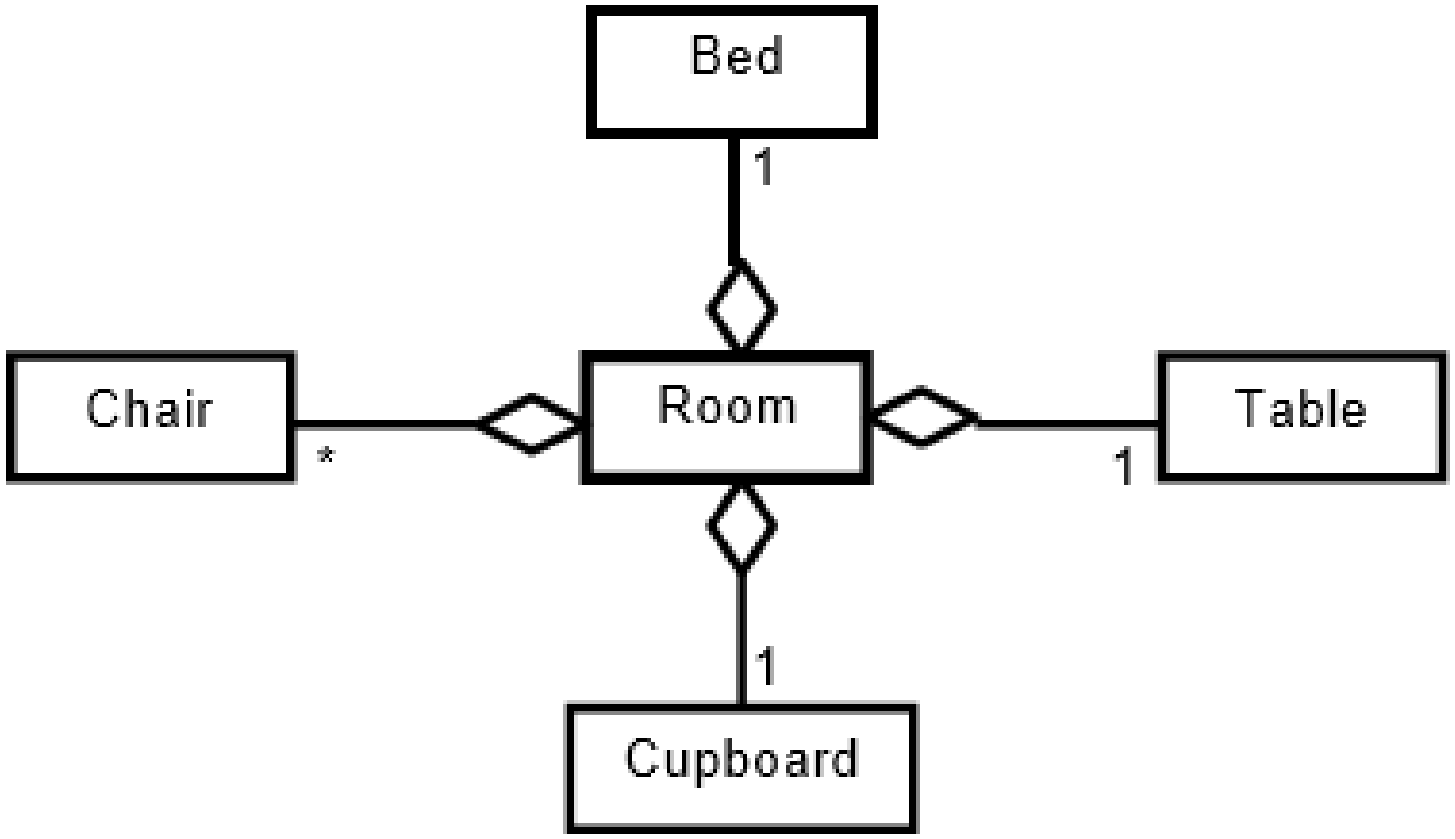
- Chair’s body is made up of different parts They can’t exist independently

AGGREGATION

- An object may contain a collection (aggregate) of other objects, the relationship between the container and the contained object is called aggregation,
- Aggregation is represented by a line with unfilled-diamond head towards the container.

Example – Aggregation

AGGREGATION



AGGREGATION

Aggregation is weaker relationship because

- *Aggregate object is not a part of the container*
- *Aggregate object can exist independently*

Example I

- Furniture is not an intrinsic part of room
- Furniture can be shifted to another room, and so can exist independent of a particular room

Example II

- A plant is not an intrinsic part of a garden
- It can be planted in some other garden, and so can exist independent of a particular garden

POLYMORPHISM

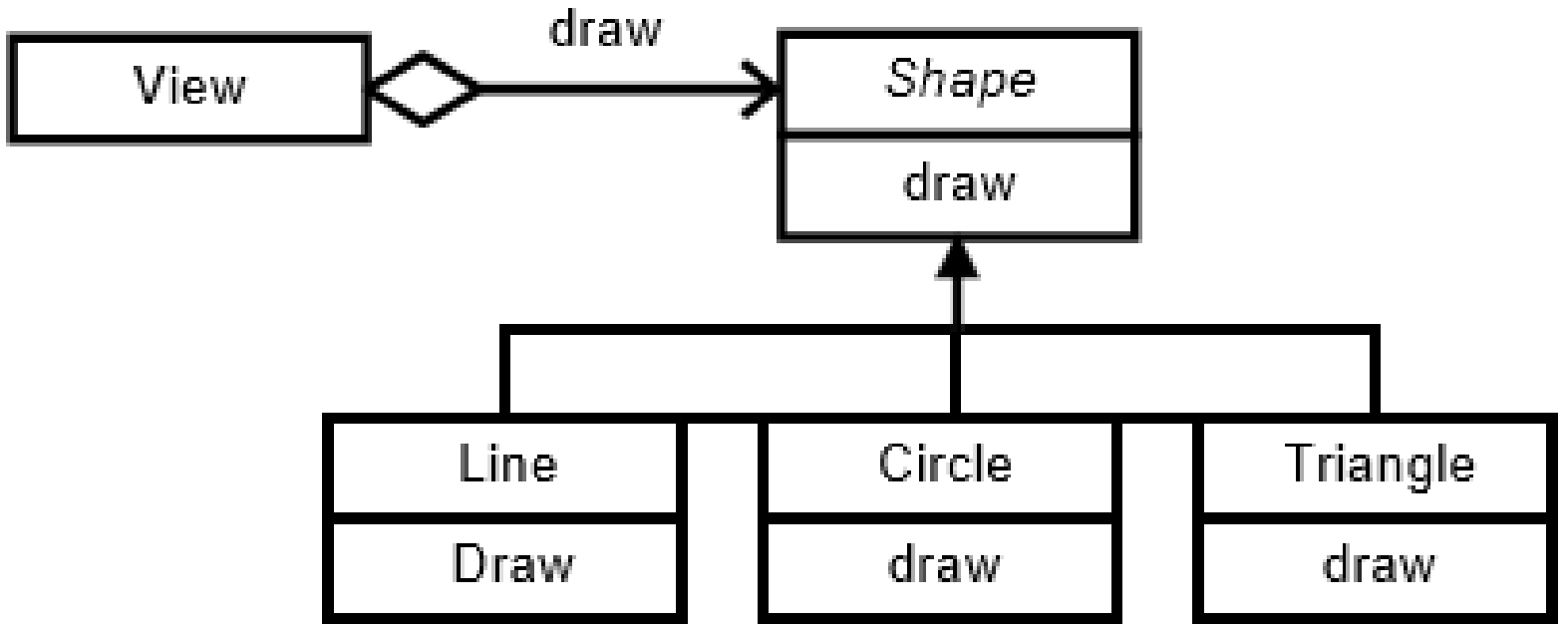
It is also essential component of object oriented modeling (paradigm).

In general, polymorphism refers to existence of ***different forms*** of a single entity.

For example, both Diamond and Coal are different forms of Carbon.

- *In OO model, polymorphism means that different objects can behave in different ways for the same message (stimulus).*

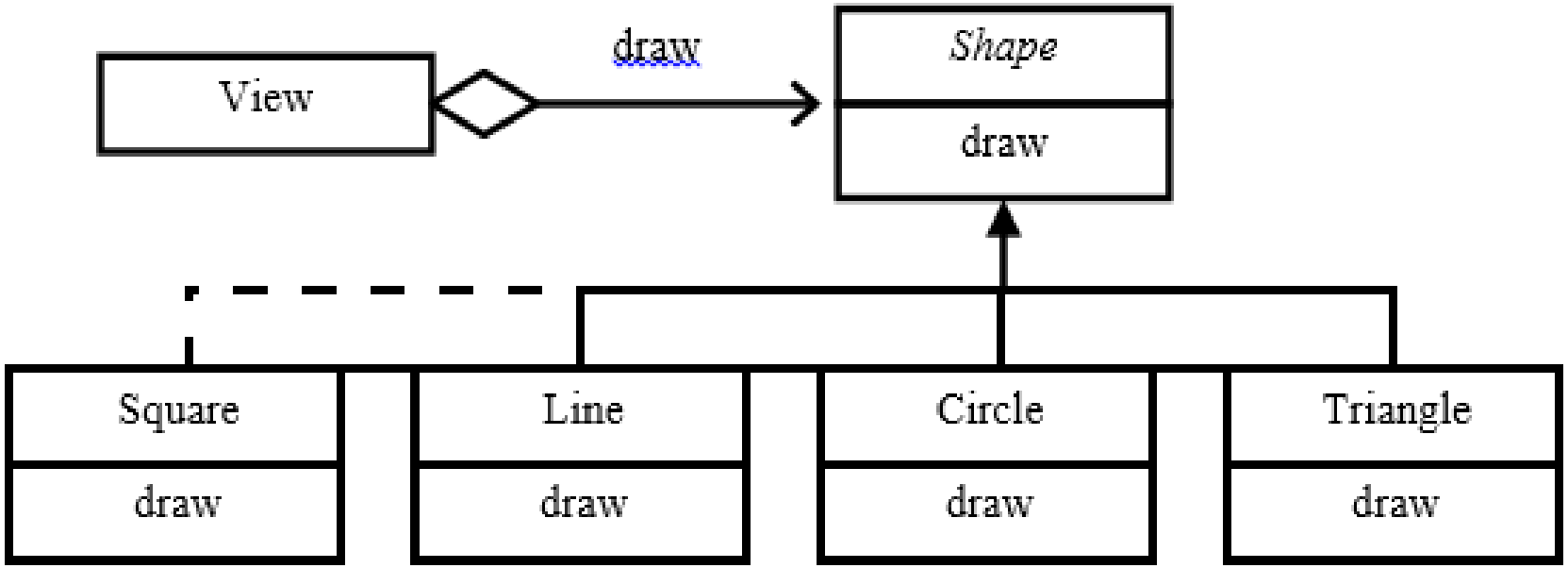
POLYMORPHISM



POLYMORPHISM ADVANTAGES

Messages can be interpreted in different ways depending upon the receiver class.

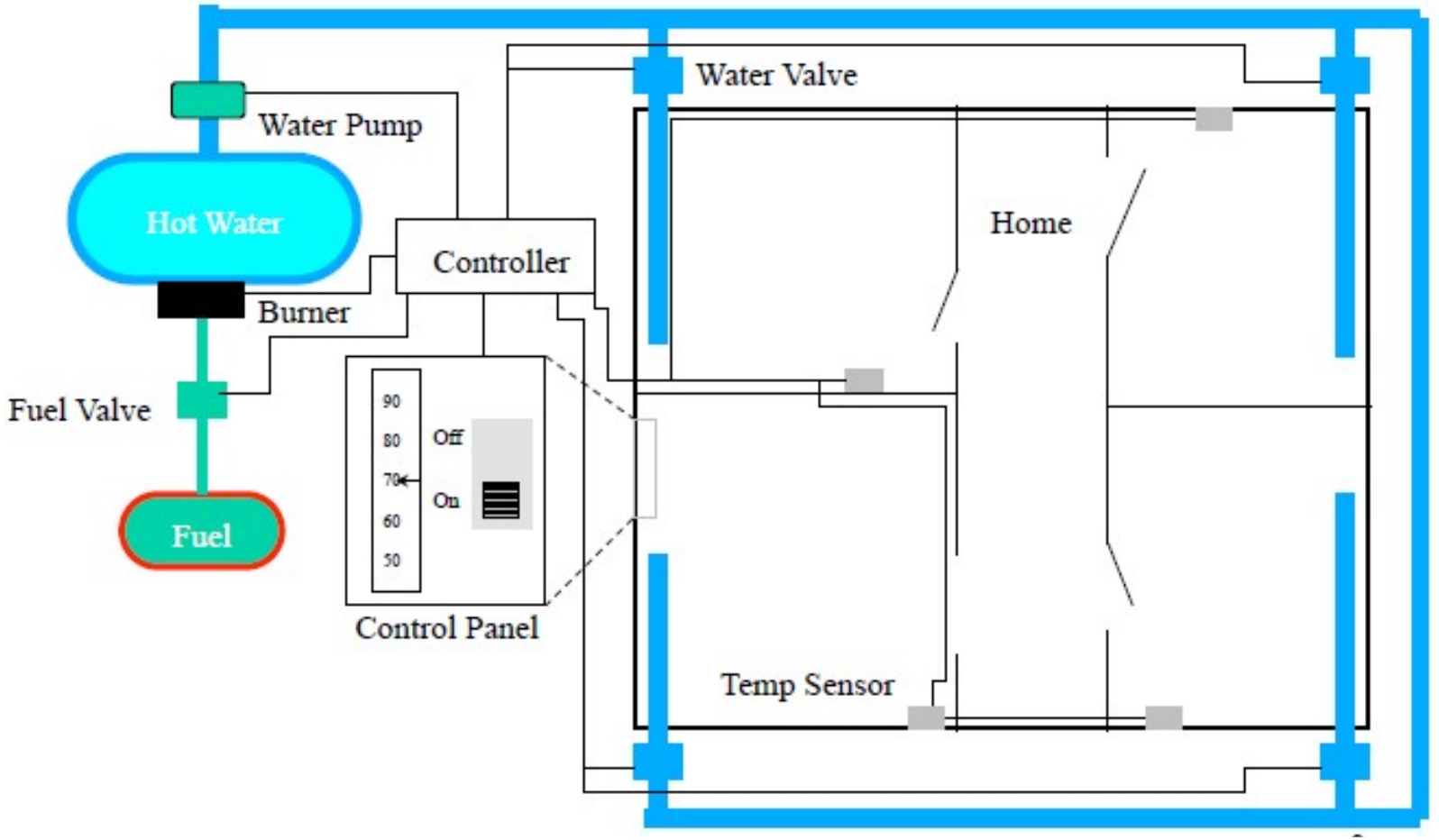
New classes can be added without changing the existing model.



In general, polymorphism is a powerful tool to develop flexible and reusable systems

OBJECT ORIENTED DESIGN WITH A CASE STUDY

THE HOME HEATING SYSTEM



HOME HEATING REQUIREMENTS

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software shall maintain the temperature of each room within a specified range by controlling the heat flow to individual rooms.

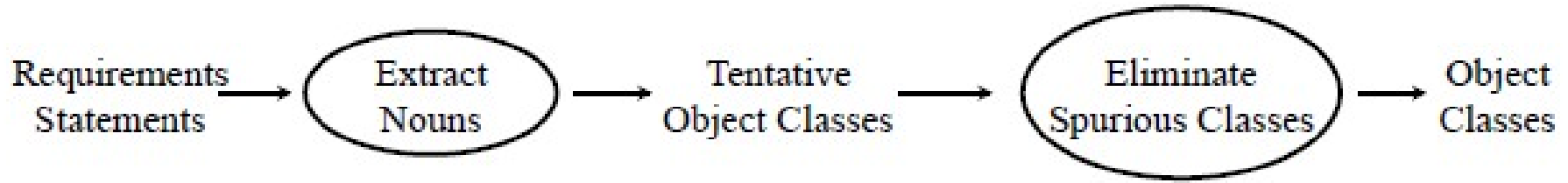
HOME HEATING REQUIREMENTS

- The software shall control the heat in each room
- The room shall be heated when the temperature is 2F below desired temp
- The room shall no longer be heated when the temperature is 2F above desired temp
- The flow of heat to each room shall be individually controlled by opening and closing its water valve
- The valve shall be open when the room needs heat and closed otherwise
- The user shall set the desired temperature on the thermostat
- The operator shall be able to turn the heating system on and of[f

HOME HEATING REQUIREMENTS

- The furnace must not run when the system is off
- When the furnace is not running and a room needs heat, the software shall turn the furnace on
- To turn the furnace on the software shall follow these steps
  - open the fuel valve
  - turn the burner on
- The software shall turn the furnace off when heat is no longer needed in any room
- To turn the furnace off the software shall follow these steps
  - close fuel valve

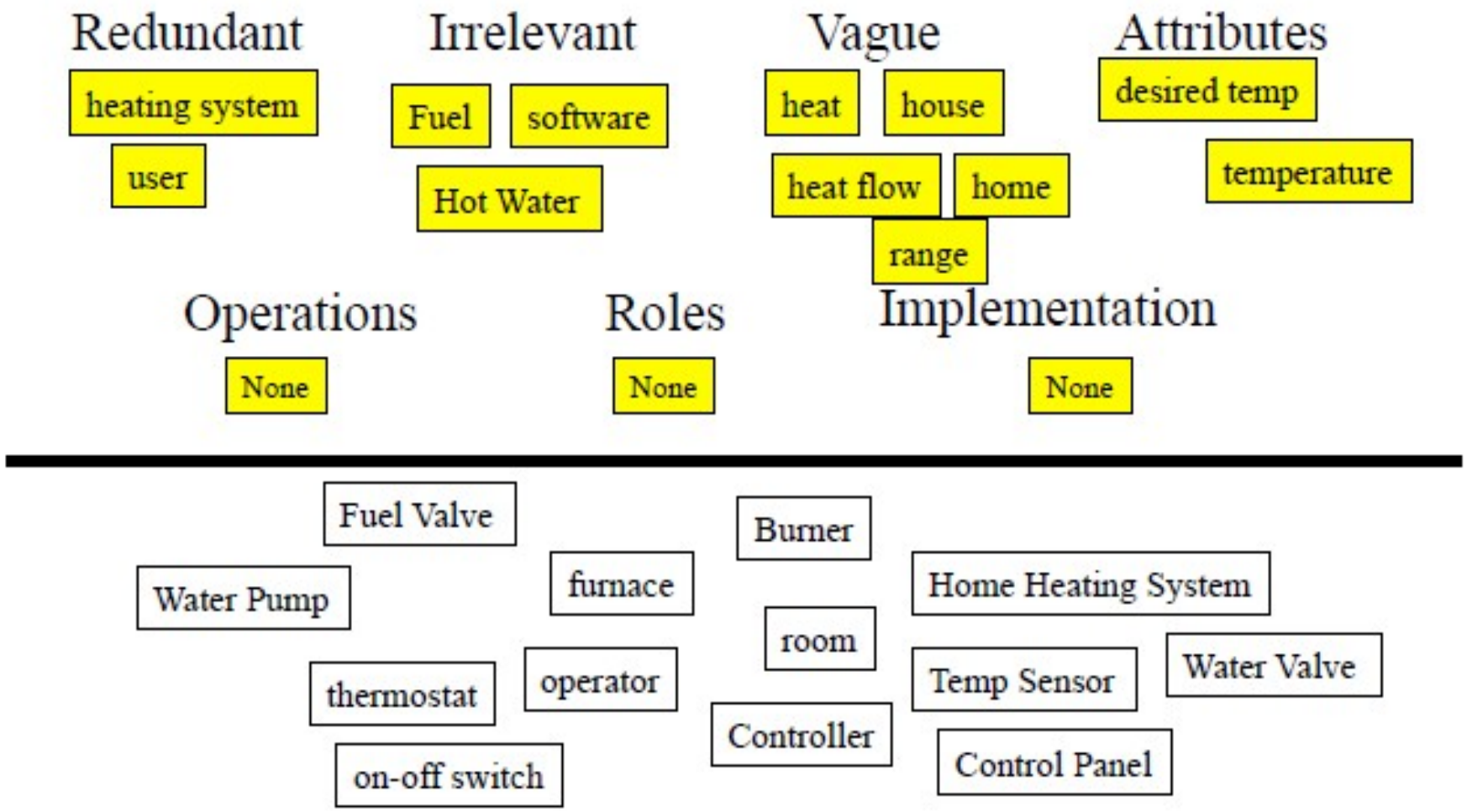
IDENTIFY OBJECT CLASSES



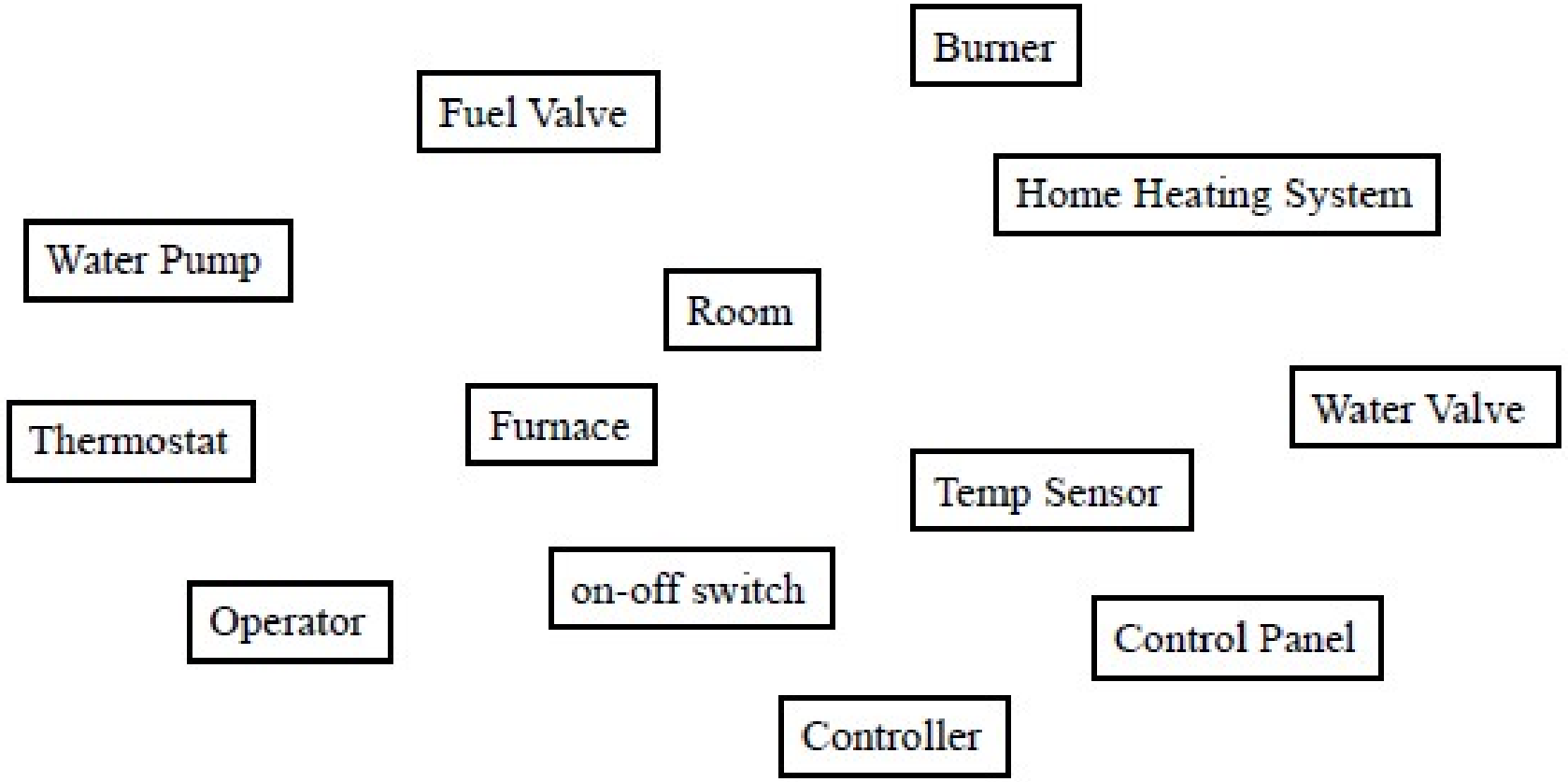
Candidate Classes

Water Pump
Hot Water
Burner
furnace
Fuel Valve
Fuel
desired temp
on-off switch
operator
thermostat
heating system
heat
Temp Sensor
house
room
temperature
Home
Controller
Home Heating System
Water Valve
range
software
Control Panel

ELIMINATION



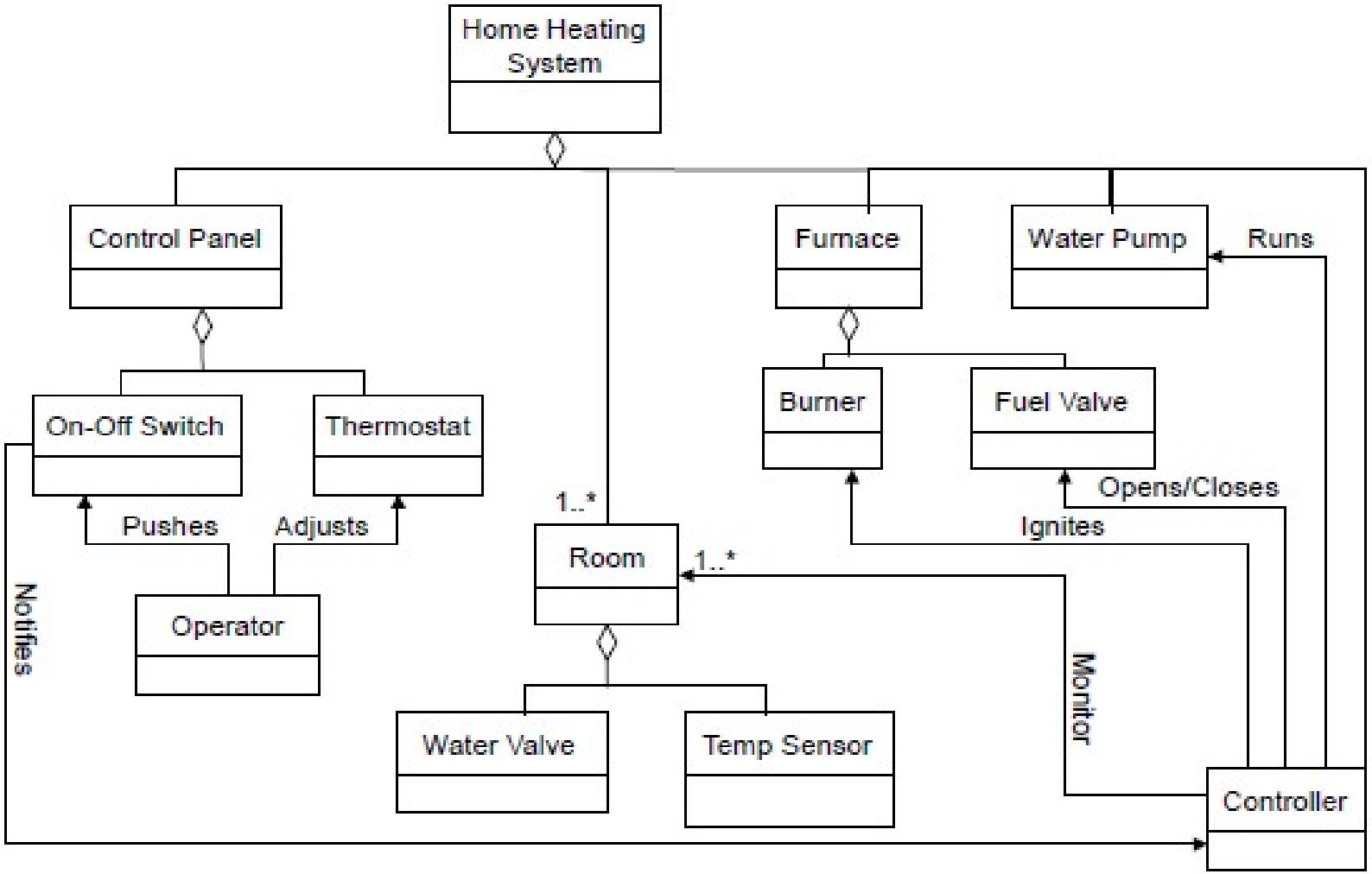
AFTER ELIMINATION



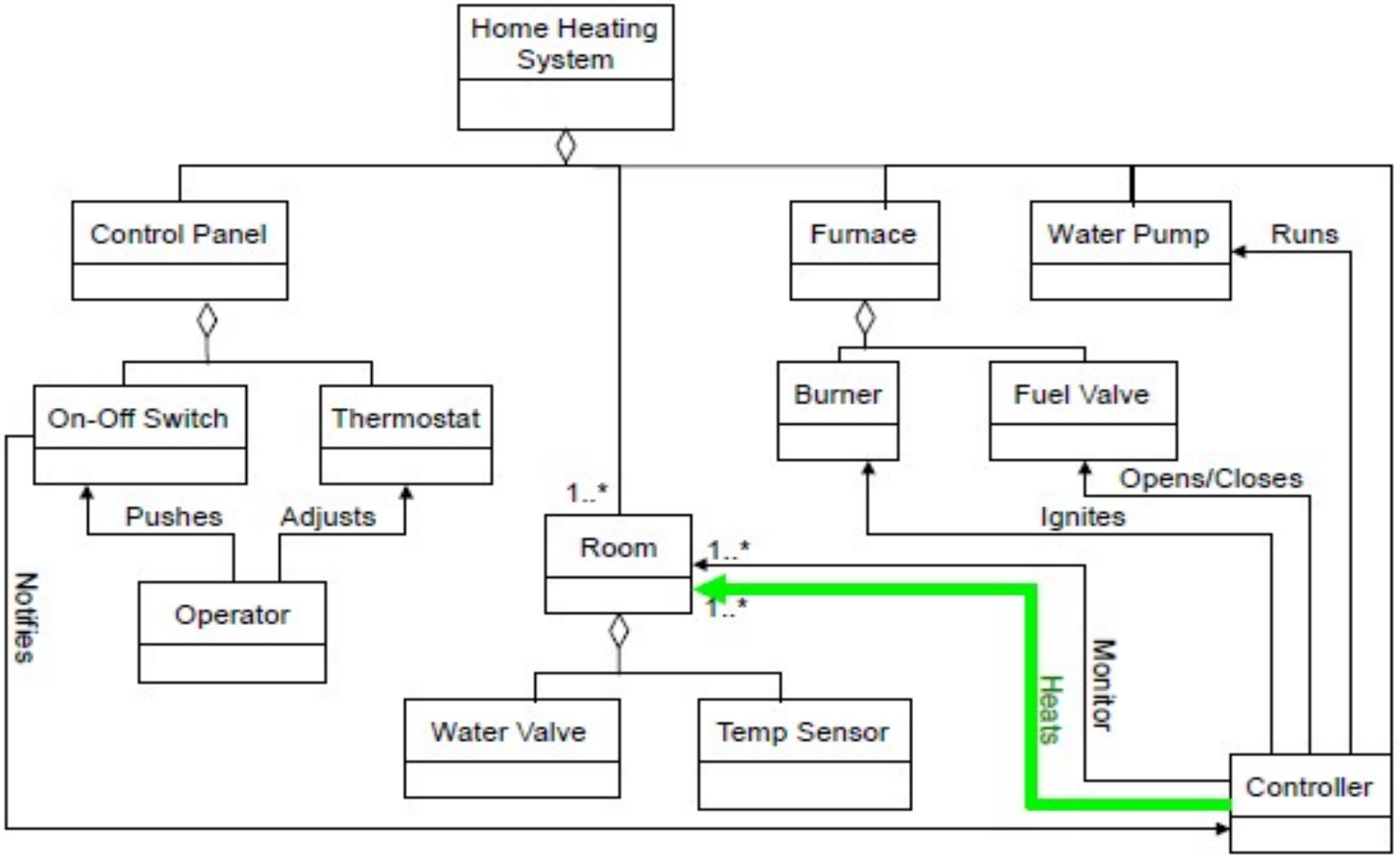
POSSIBLE ASSOCIATIONS

- The home heating system consists of a furnace, rooms, a water pump, a control panel, and a controller
- The furnace consists of a fuel pump and a burner
- The control panel consists of an on-off switch and a thermostat
- The controller controls the fuel pump
- The controller controls the burner
- The controller controls the water pump
- The controller monitors the temperature in each room
- The controller opens and closes the valves in the rooms
- The operator sets the desired temperature
- The operator turns the system on and off

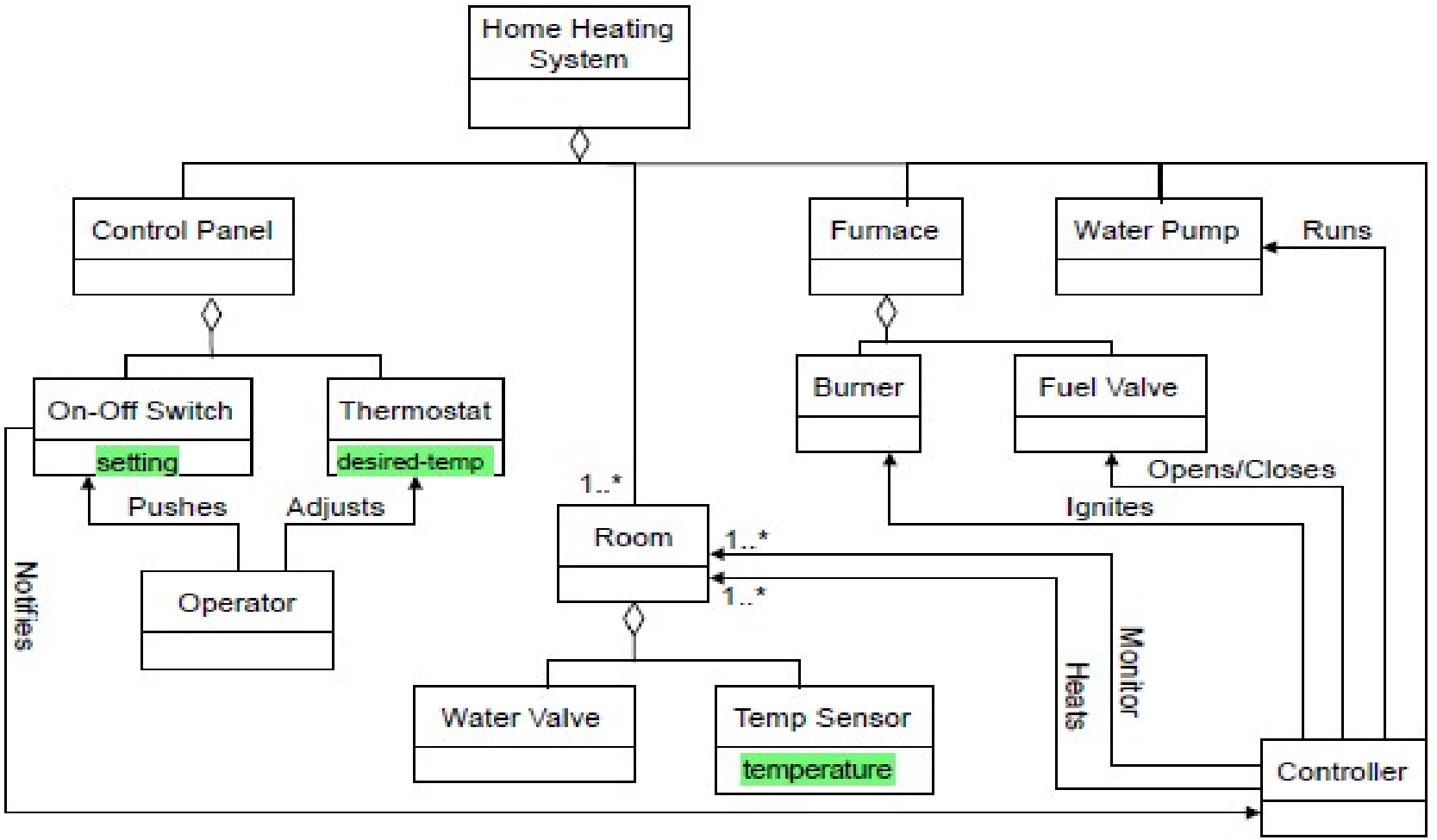
OBJECT MODEL



OBJECT MODEL - MODIFIED

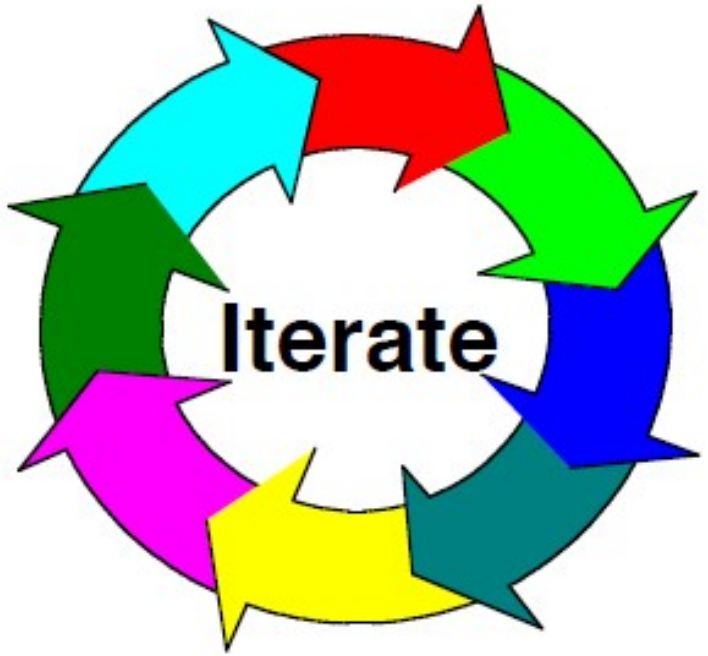


OBJECT MODEL: ATTRIBUTES

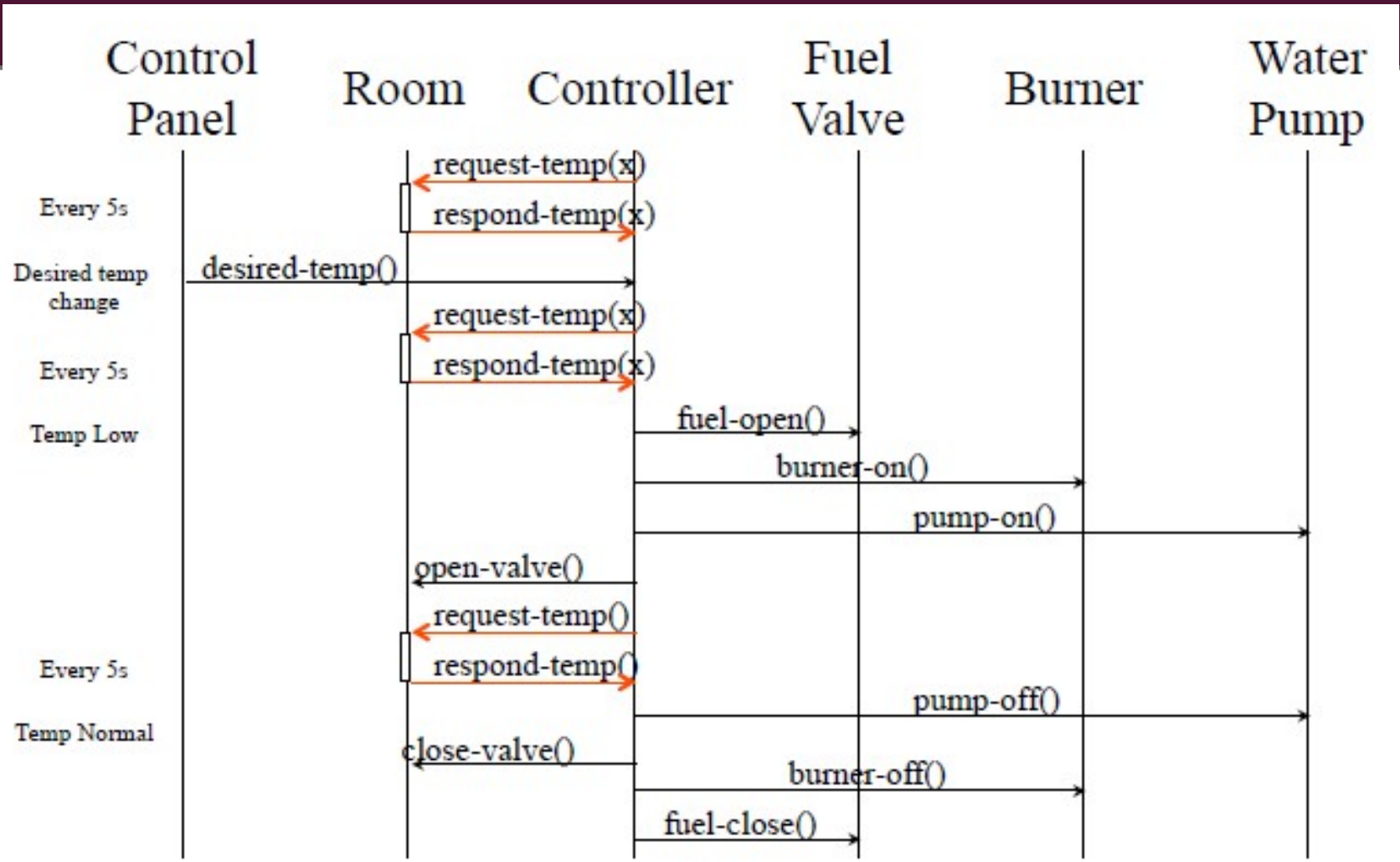


ITERATE THE MODEL

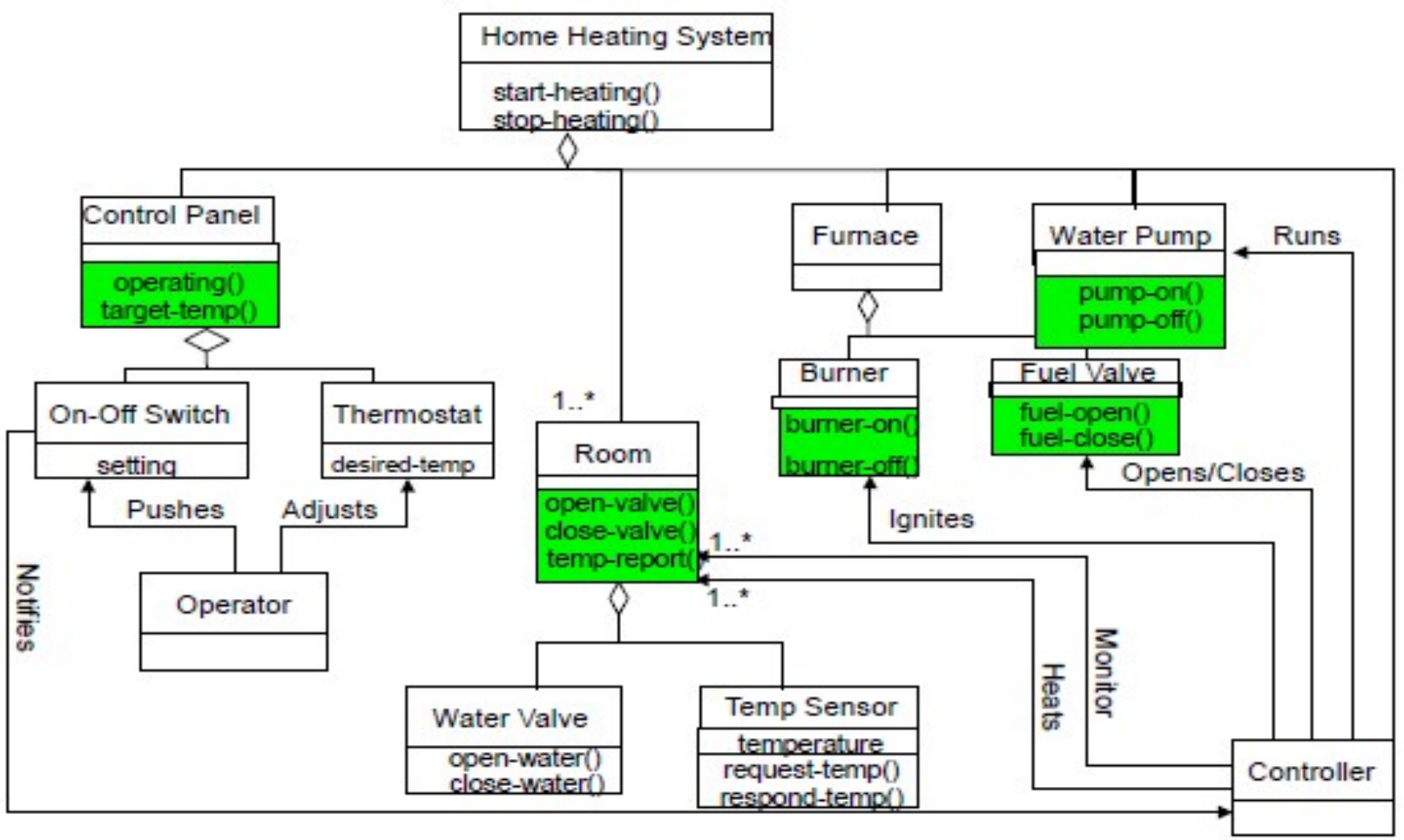
- Keep on doing this until you, your customer, and your engineers are happy with the model



SEQUENCE DIAGRAM

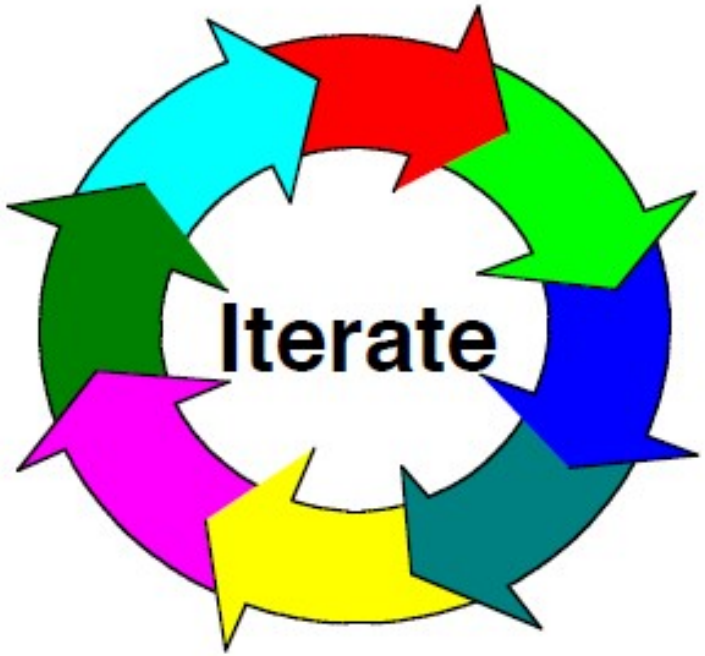


OO MODEL – MODIFIED AGAIN!



ITERATE THE MODEL

- Keep on doing this until you, your customer, and your engineers are happy with the model



HAVE A GOOD DAY!