
Today's Objectives

Define the *Problem Domain*

Define and give examples of *objects*

Define and give examples of *classes*

Define a glossary

Define *stereotypes*

Define *Domain Model Diagram*

Problem Domain

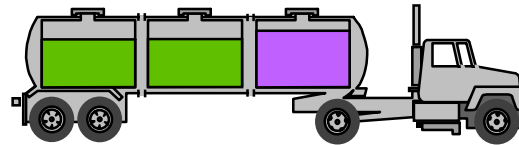
Problem domain refers to the real world concepts and things that are related to the problem that we are investigating.

Domain modeling is the task of discovering objects/classes that represent those things and concepts.

What is an Object?

Informally, an object represents some entity, either physical, conceptual or software.

Physical entity



Truck

Conceptual entity



Chemical Process

Software entity

A More Formal Definition

- . *An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application*
- . *An object is something that has:*

State

Behavior

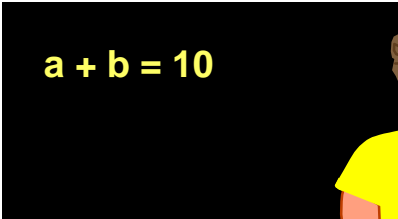
Identity

An Object Has State

The state of an object is one of the possible conditions in which an object may exist

The state of an object normally changes over time

The state of an object is usually implemented by a set of properties (called attributes), with the values of the properties, plus the links the object may have with other objects


$$a + b = 10$$



Professor Clark

Name:	Joyce Clark
Employee ID:	567138
Date hired:	March 21, 1987
Status:	Tenured

An Object Has Behavior

Behavior determines how an object acts and reacts

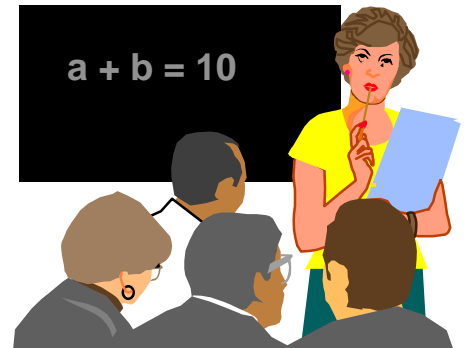
Behavior defines how an object reacts to requests from other objects

The visible behavior of an object is modeled by the set of messages it can respond to (the operations the object can perform)



Registration
System

Assign Professor Clark
(Returns:confirmation)



Algebra 101 Course

An Object Has Identity

Each object has a unique identity, even if its state is identical to that of another object



**Professor “J Clark”
teaches Algebra**



**Professor “J Clark”
teaches Algebra**



**Professor “J Clark”
teaches Algebra**

What are Classes?

There are many objects identified for any domain

A class is a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects (associations and aggregations), and common semantics

An object is an instance of a class.

A class is an abstraction in that it:

Emphasizes relevant characteristics

Suppresses other characteristics

Abstraction helps us deal with complexity

The Relationship Between Classes and Objects

A class is an abstract definition of an object

Objects may be grouped into classes



Naming Classes

- . *A class name should be a singular noun that best characterizes the abstraction*
- . *Difficulty in naming a class may be an indication of a poorly defined abstraction*
- . *Names should come directly from the vocabulary of the domain*

Style Guide for Naming Classes

. *A style guide should dictate naming conventions for classes*

. *Sample style guide*

Classes are named using singular nouns

Class names start with an upper case letter

Underscores are not used

Example: Reservation, VehicleModel, RentalGroup

Glossary

- . *After naming a class, a brief concise description of the class should be made*

Focus on the purpose of the class not on the implementation

- . *The class name and description form the basis for a model glossary*

Look for the “WHATS” and ignore the “HOWS”

Sample Model Glossary

Name: reservation

Working Definition: A request by a person to rent a EU_Rent vehicle of a particular model on a specified date and time for a specific location from a EU-Rent location.

Name: rental location

Working Definition: A EU-Rent business location at which its vehicles are offered for rent.

As more and more about the problem is discovered, refine the class definitions, and add any new classes to the model dictionary.

Class Compartments

A class is represented using a compartmented rectangle

A class is comprised of three sections

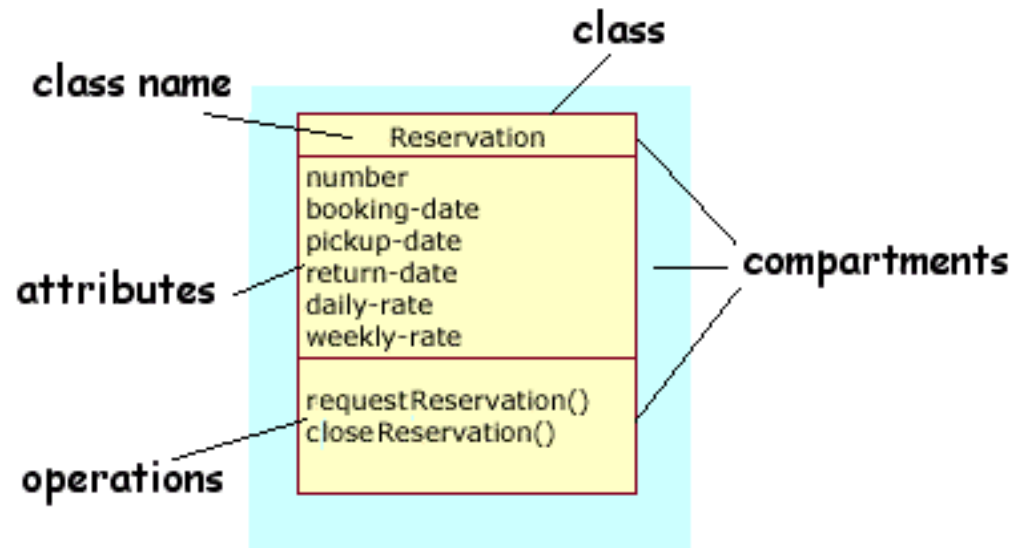
The first section contains the class name.

The second sections shows the structure (attributes).

The third section shows the behavior (operations).

The second and third sections may be suppressed if they need not be visible on the diagram

Class model diagram elements – the basic notation



Stereotypes

Every class may have at most one stereotype

Common stereotypes

entity, boundary, controller, exception

Stereotypes are shown in the class name compartment enclosed in << >>

Common stereotypes

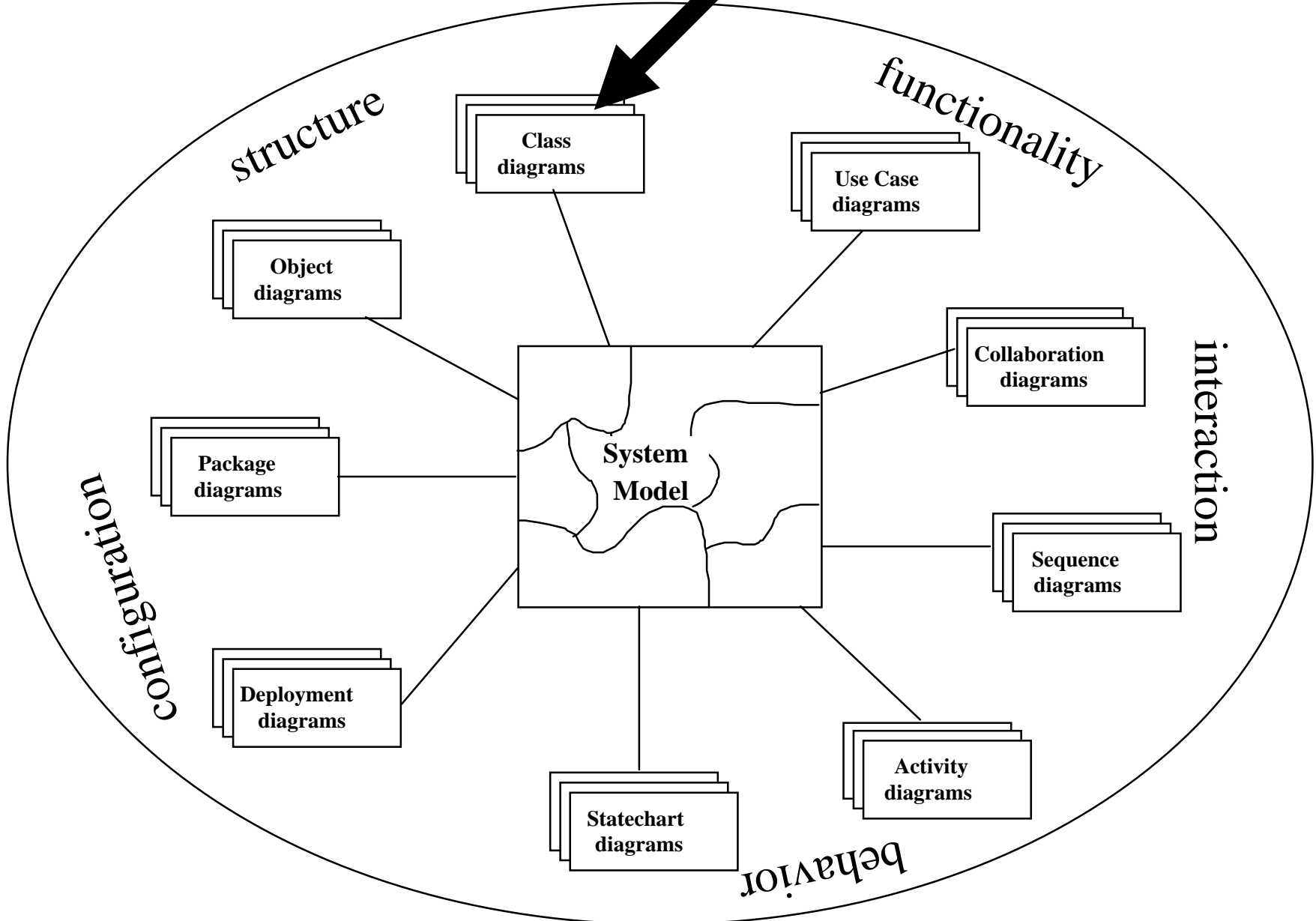
- . *A boundary class models communication between the system's surroundings and its inner workings*

- . *An entity class models information and associated behavior that must be stored*

- . *A control class models control behavior specific to one or more use cases*

Class Diagrams in Context

"We are here!" ~ static structure diagrams



Class model

This is **the** central model in an "object oriented" approach.

Class Model concepts come in two flavors:

basic concepts:

broad usage

general consensus

advanced concepts:

usage-dependent

style-dependent

Our emphasis will be on the "basic concepts."

(refer to Fowler Chapter 6 for the "advanced concepts.")

Class model

A Class Model describes:

the kinds of objects in the system, in terms of:

structure ~ the various kinds of static relationships

things an object can be expected to "know"

its attributes

its relationships

behavior ~ the dynamics

things an object can be expected to "do"

its operations

rules

constraints and guidance an object is expected to follow

Class model

A Class Model defines the static structure of concepts, types, and classes.

concept – how users think about the world

~ the semantics

type – software component interface

class – software component implementation

note: in many OO languages, the notion of ‘class’ combines both ‘interface’ & ‘implementation.’

but the distinction is important!

. *"Program to a class's interface rather than to its implementation."*

Class models & the development process

3 perspectives we can take when defining a class model:

Conceptual

the terms & facts of the problem space
i.e., the system glossary or domain model
typically, business-focused

Specification

the software, rather than the business
only the software "type" (the *interface*)

Implementation

language-specific realization
the implementation – *"the classes that implement the type"*
(or, *"implement the interface"*)
– *one type (interface) specification can have multiple implementations.*

This course deals with the "conceptual" and "specification" perspective class models; the Java course deals with the "implementation" perspective.

Class model diagram elements

Class – a description of a set of objects that share the same responsibilities (attributes, relationships, operations, rules) and semantics.

Attributes – the “value facts” the system records
the “variables”

Relationships between classes – 3 types:

association

generalization (supertype/subtype)

aggregation (*“advanced”*)

Operations – the behavior

the “methods”

Rules – the constraints that govern both structure (relationships & attributes) and behavior (operations).

Many of these elements can be shown visually as a Class Model Diagram.

Class behavior – Operations

informal definition:

An **operation** specifies what an object can “do.”

– *i.e., the processes a class knows how to carry out, when requested.*

example:

An ATM machine *knows how to* “accept a deposit.”

A reservation *knows how to* “close a reservation.”

UML definition:

An **operation** is the specification of a transformation or query that an object may be called on to execute.

Operations in perspective

Conceptual

operation = a responsibility (in a way that a business person might describe)

For example, “I (an ATM machine) am responsible to know how to accept a deposit.”

Specification

operation = signature specification (the “interface”)

Implementation (e.g., Java)

operation = realization = method (method body)

Operation properties

These are some of the things that can be specified about an operation:

visibility

public: +

protected: #

private: -

name (verb + noun phrase)

think: "I, an instance of class-ABC, know how to <verb + noun-phrase>"

e.g., "I, an instance of Order, know how to CloseAnOrder."

parameter-list

return-type

We will come back to the topic of “operations” in a later lesson.

Class structure – Attributes

informal definition:

An **attribute** is a feature of a class that describes what an object of the class can "know."

example:

A customer "knows" its address.

An order "knows" its date-placed.

An order line "knows" its quantity.

UML definition:

An **attribute** is the description of a named slot of a specified type in a class; each object of the class separately holds a value of the type.

Attributes in perspective

Conceptual

attribute = a kind of "fact" in the users' problem space

facts about the kinds of values that need to be recorded.

example:

A customer has a name (that can be any 'String' value).

An order has an order-price (that is numeric in USD to 2 decimal places)

Specification

attribute = responsibility

an object is "responsible to know (and tell)" the value of its attributes.

Implementation (e.g., Java)

attribute = value and reference types

Attribute properties

The following things can be specified about an attribute:

name – noun or noun phrase

multiplicity

typically only **1** (“mandatory”) or **0..1** (“optional”)

type – datatype (built-in or user-defined)

initial value – system-supplied value at birth?

changeability – changeable, frozen, or addOnly?

visibility – *(same as for operation)*

encapsulation

Implementation perspective: Attribute values are considered hidden; an attribute’s value must be requested.

Specification perspective: We will treat all attributes as visible (“public” with an assumed accessor operation in the design model).