The University of Iowa

22C:22 (CS:2820)

Object-Oriented Software Development

Fall 2013

Classes and Objects

by

Cesare Tinelli

Objects

An object is an entity that has

- Identity
- State
- Behavior

Object Identity

Essential feature that makes an object distinct from another

Note:

- two distinct objects may as well be identical in all other aspects
- two objects are distinct iff they have different identities

Object State

A set of attributes (properties) together with their values

- Attributes (aka fields in OO languages) are usually static
- Attribute values are usually dynamic
- An object's state can be seen as a mapping from attributes to their values
- An object's behavior depends on its state
- The internal representation of the state is usually hidden

Object Behavior

How an object acts on other objects, reacts to other objects, and changes its state

- behavior is defined by a set of operations, or *messages*, the objects responds to
- an operation is a service provided by the object, possibly using services from other objects

Common Operations

Modifier

- changes the object's state

Selector

- accesses the state without changing it

Iterator

- accesses parts of the object in some well defined order

Common Operations

Constructor

- creates an object and initializes its state

Destructor

- destroys the object and releases its resources to the system

Objects vs Classes

- Attributes and behavior are defined collectively in a class, for all objects that are instances of that class
- An object's attributes and behavior are then obtained from the class(es) it instantiates
- Only identity and attribute values are specific to each object

Classes as Contracts

- We can characterize the behavior of an object, the server, in terms of the services it provides to other objects, the clients
- An object's class defines a contract
  - that other objects depend on and
  - that must be honored by the object
- This contract establishes all assumptions a client may make about the behavior of the server

Contracts and Inheritance

- Subclassing implies contract inheritance
- If $\mathcal{B}$ is a subclass of $\mathcal{A}$, its own contract should be a refinement of $\mathcal{A}$'s contract:
  - each instance of $\mathcal{B}$ should provide at least the services provided by instances of $\mathcal{A}$, and may provide more
  - a client of $\mathcal{A}$ should be able to work with instances of $\mathcal{B}$ as if they were direct instances of $\mathcal{A}$ (no surprises!)

Recall: Design by Contract

Each service provided by an object, the server, has a set of

- preconditions, to be satisfied by the client when invoking the service
- postconditions, guaranteed by the server upon completion of the service
- invariants, properties maintained between operations by the server

Inheriting Contracts

When a subclass modifies a service m inherited from a superclass $\mathcal{A}$, it

- may relax but not strengthen m's preconditions
  - i.e., may require less from the client, but not more
- may strengthen but not relax m's postconditions
  - i.e., may offer more to the client, not less
- it must ensure that m preserve $\mathcal{A}$'s invariants

Liskov's Substitution Principle

Informal version:

If a program $\mathcal{P}$ uses

- objects of class $\mathcal{A}$ and
- $\mathcal{B}$ is a subclass of $\mathcal{A}$

replacing instances of $\mathcal{A}$ in $\mathcal{P}$ by instances of $\mathcal{B}$ should not alter the expected behavior of $\mathcal{P}$

Liskov's Substitution Principle

More formal version
(behavioral subtyping):

For all types $\mathcal{T}$ and subtypes S of $\mathcal{T}$,

every property of interest satisfied by
objects of type $\mathcal{T}$ should be satisfied by
objects of type S as well

Objects as Machines

- Objects can be also understood as little machines
- Technically, they are transition systems:
  - They have an initial state and
  - they move from one state to another in response to external messages or internal events

Objects as Machines

- Objects can be *active* or *passive*
- An active object runs independently
  - it can change its state autonomously from other objects
  - it is sometimes called an *actor*
- A passive object changes its state only when acted upon by another object

The Role of Classes/Objects in Analysis and Design

Primary tasks in analysis and early design

1. Identify relevant classes in the problem domain
2. Figure out how instances of those classes can cooperate to achieve the desired functionality

This is an incremental, iterative process