CS 247: Software Engineering Principles

Value vs. Entity Objects

Entity vs. Value Objects

Entity Object

- computer embodiment of real-world entity
- each object has a distinct identity
- object with the same attribute values are NOT equal

Entity ADT Examples

- physical objects: airplane, runway, taxiway, ...
- people: passenger, booking agent, ...
- records: customer information, boarding pass, flight schedule, ...
- transactions: reservations, cancellations, receipts, ...

Value Object

- simply represents a value of an ADT
- objects with the same attribute values are considered to be identical

Value ADT Examples

- mathematical types: rational numbers, polynomials, matrices, ...
- measurements: size, distance, weight, mass, energy, duration, ...
- other quantities: money
- other properties: colour, location, date, time, ...
- restricted value sets: names, addresses, postal codes, number ranges, ...

Design of Entity ADTs

An operation on an entity object should reflect a real-world event:

- construction of an entity object should reflect a new object in the real world
- entities referred by pointers (a consequence of the no-copy rule)
- copying an entity does is not meaningful; program no longer reflects reality; operations on copies are uncoordinated and can be lost (when copies disappear)
  - prohibit copy constructor
  - prohibit assignment
  - prohibit type conversions
  - avoid equality
  - clone operation may be useful
- computations on entities are not meaningful
  - think twice before overloading operators except `new` and `delete`
  - `operator<` is useful if overloaded to apply to entity's name or unique id

Design of Value-based ADTs

Equality is important in value types

- equality and other comparison operators
- copy constructor
- assignment operator

Computations involving values may make sense

- consider overloading arithmetic operators

Virtual functions and inheritance are uncommon

Singleton Design Pattern

Singleton design pattern - ensures that exactly one object of our ADT exists

```
class Egg {
  static Egg e;              // singleton instance
  int i;                     // data member
  Egg(int ii) : i(ii) {}     // private constructor
  Egg(const Egg&);           // unimplemented copy-constructor
public:
  static Egg* instance() { return &e; }
  int val() const { return i; }
};

Egg Egg::e(42);              // initialization of singleton
```