CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

September 6, 2017

Outline

Outline

The Process of OOD

Guidelines

- Look for object candidates. Use nouns from the problem statement which can be represented by a specific set of information items.
- Identify instance variables. Choose the specific types of information an object will need to be useful in solving the problem.
- Think about interfaces. Choose enough methods to completely express the behavior of an object.
- Refine the nontrivial methods. Continue to use top-down design: add new methods or classes to implement details of methods as you design them.

The Process of OOD

Guidelines

- Design iteratively. Revise higher levels of abstraction if lower level strategies change.
- Try out alternatives.When an idea can't be made to work, try another.
- Keep it simple. Programming is complicated enough. Try to reuse code and existing classes to control complexity. Use abstraction to hide the details of a method so you don't have to think about it when working on some other part of the system. Always give names to new classes, methods, and attributes which are descriptive.

Identifying an ADT (Dataset)

Problem

A program that takes a set of exam scores as input and prints a report that summarizes student performance.
from the Textbook, Chapter 1
(Details: The report will give the maximum, minimum, and average of the scores, and the standard deviation.)

Identifying an ADT (Dataset)

Objects

- A score is a number (float, in Python).
- A set of scores is called a Dataset by statisticians.

Identifying an ADT (Dataset)

Methods

- min returns the lowest score.
- max returns the highest score.
- average returns the mean of all the scores.
- std_deviation returns the **standard deviation**.
- size returns the size of the Dataset.
- add adds a new score to the Dataset.
- __init__ creates a new, empty Dataset.

Implementing the ADT (Dataset)

Concrete Representation #1

Attributes:

- A list object of float values (using classes built in to the Python language). The implementations of the methods specified for the interface will use this information.

Implementing the ADT (Dataset)

Concrete Representation #1

Methods:

- add appends a float value to the list.
- size returns the length of the list.
- max, min, average, std_deviation perform calculations and return values.

Implementing the ADT (Dataset)

Concrete Representation #2

Attributes:

- _average, _min, _max, _size, keep the values that are returned by the methods of the interface.
- _sum, _sum_squares keep values needed to calculate average and standard deviation values.

Implementing the ADT (Dataset)

Concrete Representation #2

Methods:

- add adds a new float value by calculating and updating all attributes.
- size returns self._size.
- max, min, average return their respective attribute values.
- std_deviation returns the standard deviation of the Dataset based on this formula:

$$s = \sqrt{\frac{\sum x_i^2 - \frac{(\sum x_i)^2}{n}}{n-1}}$$

.

Implementing the ADT (Dataset)

Comparing Representations

While #1 is easier to develop, using top-down design, it loses in comparison with #2 in two critical areas:

- Space efficiency: the **Dataset** objects in #2 take up less space, since it does not hold the entire list of **float** values, only the vital statistics of the data set.
- Time efficiency: in #1, each method must loop over the entire list of values to calculate and return the statistics. This gives performance of $O(n)$, or more precisely, $\Theta(n)$. In #2, statistics are already available as attribute values to be returned immediately, giving performance of $O(1)$ (indeed, $\Theta(1)$).

Identifying the ADT

Problem

Python has no way to represent the values of certain fractions such as $\frac{1}{3}$ exactly because the denominator is not a power of two. Yet there are simple rules to add, subtract, multiply and divide fractions exactly. Design a class to implement operations on exact rational numbers.

Operator Overloading

Operations in Existing Python Classes

Python allows addition of integers using different syntactic forms:

```
>>> a = 3
>>> a+4
7
>>> a.__add__(4)
7
>>> int.__add__(a, 4)
7
```

Operator Overloading

Adapting Existing Operations for New Classes

Similar examples can be given for most other binary operations. To bring these operations to new classes, define appropriately named methods, with appropriate behavior, returning objects having the desired value. For example:

```
class Rational(object):
   ...
   def __add__(self, other):
   ...
      return Rational(num,den)
```

will allow `a+b` to have the correct rational value if `a` and `b` are `Rational` objects.

The Rational Class

Concrete Representation

Attributes:

- num (integer, numerator)
- den (integer greater than zero, denominator)

The Rational Class

Concrete Representation

Methods:

- Constructor `__init__(self, n, d)`
- Overloaded operators, using `__add__`, `__sub__`, et cetera.