CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

September 11, 2017

Outline

1. Chapter 3: Container Classes
   - Python Lists
   - A Sequential Collection: A Deck of Cards
   - A Sorted Collection: Hand

Outline

1. Chapter 3: Container Classes
   - Python Lists
   - A Sequential Collection: A Deck of Cards
   - A Sorted Collection: Hand

Interface for the list Class

Lists are Containers

A container class provides objects which contain collections of other objects. Usually, containers are homogeneous—the data is all of one type. But a Python list can contain string, int, and float values at the same time. We will design special-purpose container classes that are not built-in to Python (or C++), whose methods will be carefully implemented based on efficiency issues.

Interface for the list Class

Python list Method Specifications

- Concatenation `list1 + list2`
- Repetition `list1 * int1` or `int1 * list1`
- Length `len(list1)`
- Index `list1[i]`
- Slice `list1[start:stop:step]`
- Membership `item in list1`
- Append `list1.append(obj1)`
- Insert `list1.insert(int1, obj1)`
- Delete index `list1.pop(i)`
- Remove object `list1.remove(obj1)`

Interface for the list Class

Parameter Values

- The step parameter in the slice operation can be negative (it means step backwards).
- If the step parameter is missing, its default value is assumed to be 1. (The book only shows start and stop. This will work to step through each value, since the default step is 1. But to skip the odd indices, say, you would use step = 2.)

Specifying the Deck Class

Sequential Collections

- A sequential collection is a container which allows one to traverse its objects sequentially.
- If the collection is not empty, it will have a first item.
- Each item (except the last) will have a next item after it.
- Starting from the first item, the entire collection is traversed by going to the next item until the last item is reached.
- A deck of cards is a sequential collection: the top card is the first, when the current card is removed, the next card is now at the top of the deck. The last card is at the bottom of the deck.

Specifying the Deck Class

Sorted Lists

- A sorted list is a homogeneous list where the items are increasing (each item is less than the next item) or decreasing (each item is greater than the next item).
- The items must be comparable: there is a binary operator (<) returning a boolean value.
- A deck of cards can be sorted, but for games, they are unsorted by shuffling.

Specifying the Deck Class

Problem: To Simulate a Deck of Cards

Provide a class whose objects will behave like a deck of cards: they can be shuffled and dealt to help simulate card games like poker or bridge.

Specifying the Deck Class

Objects

A `Deck` object will be a container class for `Card` objects, which have rank and suit attributes.

Specifying the Deck Class

Methods

- `shuffle` will ensure that dealing cards will produce a random sequence.
- `deal` will return a card from the deck, removing it from the deck in the process.

Implementing the Deck Class

Concrete Representation

Attributes:

- A Python list, `cards`, of `Card` objects, as defined in Chapter 2.

  Remark: An Abstract Data Type, when implemented, should only have attributes which are private, that is, with an underscore (_) as first character. The book does not do that here, which is unsafe. If a function outside the class has access to the concrete representation, then it will become broken if that representation changes, which is exactly what we want to avoid.

Implementing the Deck Class

Concrete Representation

Methods:

- __init__(self) creates a 52 Card deck.
- shuffle(self) prepares for random dealing by putting the list of Cards in random order.
- deal(self), returns a `Card` object, while removing it from the list `cards`.
- size(self) returns the number of cards remaining in the list. (See Deck.py in Chapter 3)

Implementing the Deck Class

Concrete Representation

```
def __init__(self):
    cards = []
    for suit in Card.SUITS:
        for rank in Card.RANKS:
            cards.append(Card(rank,suit))
    self.cards = cards
def shuffle(self):
    n = self.size()
    cards = self.cards
    for i,card in enumerate(cards):
        pos = randrange(i,n)
        cards[i] = cards[pos]
        cards[pos] = card
```

Specifying the Hand Class

Problem: To Simulate a Bridge Hand

We want to write a program to play the card game bridge. We can use the `Card` and `Deck` abstractions, but we need a new class to represent a legal hand for bridge. We need to:

- deal: Deal a shuffled deck into 4 13-card bridge hands.
- sort: Sort the suits of each hand (Ace is highest), and
- dump: print out the contents of each hand. Other methods will be defined in implementing these basic ones.

Specifying the Hand Class

Creating a Bridge Hand

```
class Hand(object):
 def __init__(self, label=""):
    self.label = label
    self.cards = []
 def add(self, card):
    self.cards.append(card)
 def sort(self):
    self.cards.sort()
    self.cards.reverse()
 def dump(self):
    print(self.label + "'s Cards:")
    for c in self.cards:
       print(" ", c)
```

Specifying the Hand Class

Comparing Cards

```
def __eq__(self, other):
    return (self.suit_char == other.suit_char and
        self.rank_num == other.rank_num)
def __lt__(self, other):
    if self.suit_char == other.suit_char:
        return self.rank_num < other.rank_num
    else:
        return self.suit_char < other.suit_char
def __ne__(self, other):
    return not(self == other)
def __le__(self, other):
    return self < other or self == other
```

Specifying the Hand Class

Sorting A Hand Manually With Selection Sort

```
def sort(self):
    cards0 = self.cards
    cards1 = []
    while cards0 != []:
        next_card = max(cards0)
        cards0.remove(next_card)
        cards1.append(next_card)
    self.cards = cards1
```