CS61B Lecture #17

Announcements

amming Contest coming up Saturday October 14. See
ecs.berkeley.edu/~ctest/contest for more details.

Topics

standard Java Collections classes.

, ListIterators

s and maps in the abstract

nalysis of implementing lists with arrays:

Data Types in the Abstract

time, should *not* worry about implementation of data
earch, etc.

o for us—their specification—is important.

eral standard types (in `java.util`) to represent collec-
cts

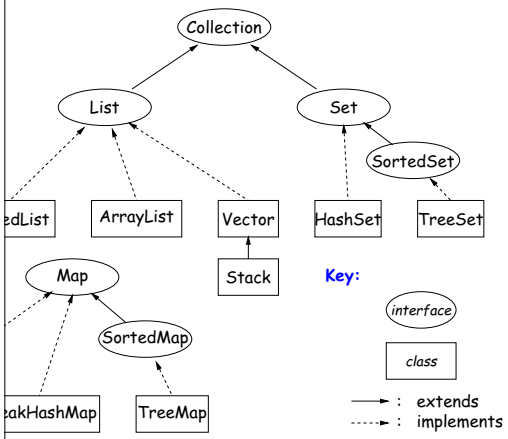
aces:

tion: General collections of items.
ndexed sequences with duplication
rtedSet: Collections without duplication
rtedMap: Dictionaries (key $\mapsto$ value)

classes that provide actual instances: `LinkedList`, `ArrayList`,
`TreeSet`.

hange easier, purists would use the concrete types only
nterfaces for parameter types, local variables.

Collection Structures in java.util



The Collection Interface

terface. Main functions promised:

hip tests: `contains` ($\in$), `containsAll` ($\subseteq$)

eries: `size`, `isEmpty`

`iterator`, `toArray`

modifiers: `add`, `addAll`, `clear`, `remove`, `removeAll` (set
e), `retainAll` (intersect)

The List Interface

lection

represent *indexed sequences* (generalized arrays)

ethods to those of `Collection`:

hip tests: `indexOf`, `lastIndexOf`.

`get(`i`)`, `listIterator()`, `sublist(`B, E`)`.

: `add` and `addAll` with additional index to say *where* to wise for removal operations. `set` operation to go with

erator<Item> extends Iterator<Item>:

vious and hasPrevious.

ve, and set allow one to iterate through a list, inserting, or changing as you go.

t Question: What advantage is there to saying `List L` an `LinkedList L` or `ArrayList L`?

Amortization: Expanding Vectors

array for expanding sequence, best to *double* the size row it. Here's why.

ze s, doubling its size and moving s elements to the new time proportional to $2s$.

there is an additional $\Theta(1)$ cost for each addtion to actually assigning the new value into the array.

ld up these costs for inserting a sequence of N items, st turns out to proportional to N, as if each addition t time, even though some of the additions actually take ional to N all by themselves!

ating Amortized Time: Potential Method

the argument, associate a *potential*, $\Phi_i \geq 0$, to the i^{th} at keeps track of "saved up" time from cheap operations "spend" on later expensive ones. Start with $\Phi_0 = 0$.

he *amortized cost* of the i^{th} operation as

$$a_i = c_i + \Phi_{i+1} - \Phi_i,$$

he real cost of the operation.

erations, we artificially set $a_i > c_i$ and increase Φ ($\Phi_{i+1} >$

e ones, we typically have $a_i \ll c_i$ and greatly decrease Φ t it go negative—may not be "overdrawn").

all this so that a_i remains as we desired (e.g., $O(1)$ for ray), without allowing $\Phi_i < 0$.

t we choose a_i so that Φ_i always stays ahead of c_i.

about Library Design: Optional Operations

ections need to be modifiable; often makes sense just from them.

rations are optional (`add`, `addAll`, `clear`, `remove`, `removeAll`,

developers decided to have *all* Collections implement llowed implementations to throw an exception:

`UnsupportedOperationException`

ve design would have created separate interfaces:

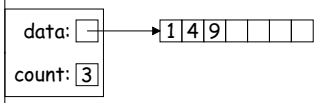
```
lection { contains, containsAll, size, iterator, ... }
andable extends Collection { add, addAll }
inkable extends Collection { remove, removeAll, ... }
ifiableCollection
llection, Expandable, Shrinkable { }
```

ave lots of interfaces. Perhaps that's why they didn't y.

mplementing Lists (I): ArrayLists

ncrete types in Java library for interface `List` are nd `LinkedList`:

t expect, an `ArrayList`, `A`, uses an array to hold data. , a list containing the three items 1, 4, and 9 might be like this:



four more items to `A`, its `data` array will be full, and `data` will have to be replaced with a pointer to a new, that starts with a copy of its previous values.

r best performance, how big should this new array be?

se the size by 1 each time it gets full (or by any con- the cost of N additions will scale as $\Theta(N^2)$, which List look much worse than `LinkedList` (which uses an implementation.)

ortization: Expanding Vectors (II)

	Resizing Cost	Cumulative Cost	Resizing Cost per Item	Array Size After Insertions
	0	0	0	1
	2	2	1	2
	4	6	2	4
	0	6	1.5	4
	8	14	2.8	8
	0	14	2.33	8
	⋮	⋮	⋮	⋮
	0	14	1.75	8
	16	30	3.33	16
	⋮	⋮	⋮	⋮
	0	30	1.88	16
	⋮	⋮	⋮	⋮
-1	0	$2^{m+2} - 2$	≈ 2	2^{m+1}
	2^{m+2}	$2^{m+3} - 2$	≈ 4	2^{m+2}

d out (*amortize*) the cost of resizing, we average at time units on each item: "amortized insertion time is 4 to add N elements is $\Theta(N)$, *not* $\Theta(N^2)$.

Application to Expanding Arrays

g to our array, the cost, c_i, of adding element #i when eady has space for it is 1 unit.

es not initially have space when adding items 1, 2, 4, 8, ner words at item 2^n for all $n \geq 0$. So,

≥ 0 and is not a power of 2; and

1 when i is a power of 2 (copy i items, clear another i l then add item #i).

peration #2^n we're going to need to have saved up at $^{n+1}$ units of potential to cover the expense of expanding nd we have this operation and the preceding $2^{n-1} - 1$ which to save up this much potential (everything since g doubling operation).

t choose $a_i = 5$ (actually, could let $a_0 = 1$).

happens:

4	5	6	7	8	9	10	11	12	13	14	15	16	17
9	1	1	1	17	1	1	1	1	1	1	1	33	1
5	5	5	5	5	5	5	5	5	5	5	5	5	5
10	6	10	14	18	6	10	14	18	22	26	30	34	6