CSC 207 2014S: Extra Session, Week 10

Overview

- Admin
- You ask questions.
- I try to give answers, or at least direct you in the right direction.

Admin

- CS table tomorrow.
- New class project. Yay!

Factories

Why is Sam's an interface and most of the ones out there are classes?

Sam's were anonymous classes, so they were really classes.

- Idea: Sometimes methods need to create objects on the fly, and we want them general: They should be able to work with any object that meets certain criteria
  - E.g., a program might need multiple dictionaries
  - Issue: The client should be able to specify the kind of dictionaries to use.
- Similar to the idea that we can write code that expects a Dictionary and someone can give that code an AssociationList or a HashTable or ...
- Difference: Need to create multiple objects.
- Solution: Factories: Pass in something (method, object, etc.) that knows how to create objects that match the goal

For Dictionaries

```
public interface DictionaryFactory<K,V>
{
  public Dictionary<K,V> dictionary(...);
} // interface DictionaryFactory
```

One factory for association lists

```
public class AssociationListFactory<K,V>
    implements DictionaryFactory<K,V>
{
  public Dictionary<K,V> dictionary(...)
  {
    return new AssociationList<K,V>();
  } //  dictionary(...)
} // class AssociationListyFactory<K,V>
```

Another factory for association lists

```
public DictionaryFactory<K,V> df = new DictionaryFactory<K,V>()
  {
    public Dictionary<K,V> dictionary(...)
    {
      return new AssociationList<K,V>();
    } //  dictionary(...)
  };
```

Lots of uses ... if we have lots of strings to parse, we might pass in the things that turn strings into objects

- E.g., on the homework, we might have something that builds the "right" kind of numbers, and the client could decide what the right kinds of numbers are.

Experienced Java programmers tend to use introspection in making factories.

Good Invariants for Problem 1

- We have an iterative loop to do DNF and recursion to do the quicksorting.
  - Recursive quicksorting is hard. Probably a good problem for the next exam, except that you can probably StackOverflow it.
- The invariant is for DNF, not for Quicksort

Input

```
qsort(T[] values, int lb, int ub)
```

Goal:

```
Break into three parts (< pivot, = pivot, > pivot)
```

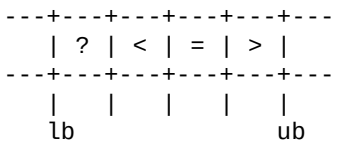
Post-picture

```
---+---+---+---+---
   | < | = | > |
---+---+---+---+---
   |   |   |   |
   lb  s   l   ub
```

Invariant

```
---+---+---+---+---+---+---
   | < | = | ? | > |
---+---+---+---+---+---+---
   |   |   |   |   |
   lb  s   i   l   ub
```

There are lots of other possibilities



Cost of operations

How much does "Hello" + "Goodbye" take?

- Builds a whole new string, so O(sum of lengths)
- Note that print("Hello"); print("Goodbye") might therefore be much more efficeint than print("Hello" + "Goodbye")
- From C: How much time do you expect strcat(t,s) to take?
  - s puts s at the end of t.