3.4 Encapsulation and ADTs

Bond. What's your escape route?
Saunders. Sorry old man. Section 26 paragraph 5, that information is on a need-to-know basis only. I'm sure you'll understand.

Abstract Data Types

Data type. Set of values and operations on those values.
Ex. `int`, `String`, `Complex`, `Vector`, `Document`, `Wave`, `Tour`, ...

Abstract data type. Data type whose internal representation is hidden.

Separate implementation from design specification.
- Class provides data representation and code for operations.
- Client uses data type as black box.
- API specifies contract between client and class.

Counter Data Type

Counter. Data type to count electronic votes.

```
public class Counter {
   int count;

   public Counter()  { count = 0;     }
   public void hit() { count++;       }
   public int get()  { return count; }

}
```

Legal Java client.

```
Counter c = new Counter();
c.count = -16022;
```

Pitfall. Al Gore receives -16,022 votes in Volusia County, Florida.

Counter Abstract Data Type

Counter. Abstract data type to count electronic votes.

```
public class Counter {
   private int count;

   public Counter()  { count = 0;     }
   public void hit() { count++;       }
   public int get()  { return count; }

}
```

Does not compile.

```
Counter c = new Counter();
c.count = -16022;
```

Benefit. Can guarantee invariant that each data type value remains in a consistent state.

Changing Internal Representation

Java ADTs.
- Keep data representation hidden with `private` access modifier.
- Expose API to clients using `public` access modifier.

```
public class Complex {
   private double re;
   private double im;

   public Complex(double re, double im)   { … }
   public double abs()                    { … }
   public Complex plus(Complex b)         { … }
   public Complex times(Complex b)        { … }
   public String toString()               { … }
}
```

Advantage. Can switch to polar representation without changing client.
Note. All our data types are already ADTs!

Time Bombs

Y2K. Two digit years: January 1, 2000.
Y2038. 32-bit seconds since 1970: January 19, 2038.

ZIP codes. USPS changed from ZIP to ZIP + 4 code in 1983.

VIN numbers. Will run out by 2010 ⇒ representation change ahead!

Lesson. By exposing data representation to client, need to sift through millions of lines of code in client to update.

Ask, Don't Touch

Encapsulation.
- Can't "touch" data and do whatever you want.
- Instead, "ask" object to manipulate its data.

"Ask, don't touch."

Adele Goldberg
Former president of ACM
Co-developed Smalltalk

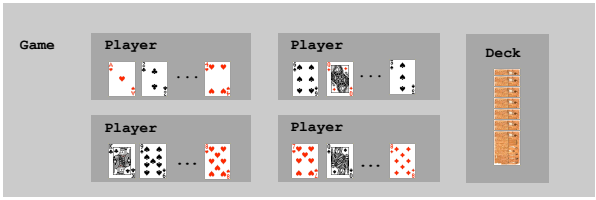
Lesson. Limiting scope makes programs easier to maintain and understand.

"principle of least privilege"

Modular Programming and Encapsulation

ADTs enable modular programming.
- Separate compilation.
- Split program into smaller modules.
- Different clients can share the same ADT.

ADTs enable encapsulation.
- Keep modules independent from each other.
- Can substitute different classes that implement same API.



4.6 Symbol Tables

Symbol Table ADT

Symbol table, Key-value pair abstraction.

- Insert value with specified key.
- Search for value given key.
- Delete value with given key.

Ex. key = URL, value = IP address.

- Insert URL with specified IP address.
- Given URL, find corresponding IP address.

Web Site	IP Address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.simpsons.com	209.052.165.60
key	value

Symbol Table Applications

Application	Purpose	Key	Value
Phone book	Look up phone number	Name	Phone number
Dictionary	Look up word	Word	Definition
Book index	Find relevant pages	Keyword	List of pages
Web search	Find relevant documents	Keyword	List of documents
Bank	Process transaction	Account number	Transaction details
File system	Find file on disk	File name	Location on disk
File share	Find song to download	Name of song	Computer ID
Web cache	Download	Filename	File contents
Genomics	Find markers	DNA string	Known positions
DNS	Find IP address given URL	URL	IP address
Reverse DNS	Find URL given IP address	IP address	URL
Compiler	Find properties of variable	Variable name	Value and type
Routing table	Route Internet packets	Destination	Best route

Symbol Table Client: DNS Lookup

DNS lookup client program.

- `st.put(key, value)` inserts a key-value pair into symbol table.
- `st.get(key)` searches for the given key and returns the value.

```
public static void main(String[] args) {
   SymbolTable st = new SymbolTable();

   st.put("www.princeton.edu",    "128.112.128.15");
   st.put("www.yale.edu",         "130.132.143.21");
   st.put("www.simpsons.com",     "209.052.165.60");
          st["www.simpsons.com"] = "209.052.165.60"

   System.out.println(st.get("www.harvardsucks.com"));
   System.out.println(st.get("www.simpsons.com"));
}                                  st["www.simpsons.com"]
```

```
% java SymbolTableTest
null
209.052.165.60
```

Symbol Table Client: Remove Duplicates

Remove duplicates. [from a mailing list or voting eligibility list]

- Read in `key`.
- If `key` is not in symbol table, print out `key` and insert it.

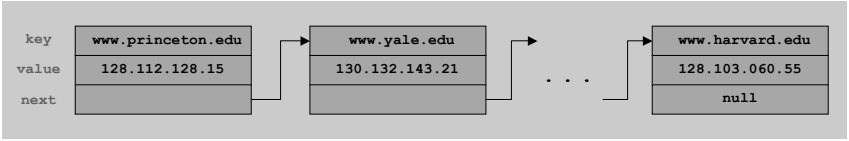
```
public class DeDup {
   public static void main(String[] args) {
      SymbolTable st = new SymbolTable();
      while (!StdIn.isEmpty()) {
         String key = StdIn.readString();
         if (st.get(key) == null) {
            System.out.println(key);
            st.put(key, "");
         }
      }
   }
}
```

insert empty string as value

Symbol Table: Linked List Implementation

Maintain a linked list of key-value pairs.

- Insert new key-value pair at beginning of list.
- Exhaustive search list to find given key.



Symbol Table: Linked List Implementation

```
public class SymbolTable {
   private Node first;
   private class Node {
      String key;
      Object value;
      Node next;
      Node(String key, Object value, Node next) {
         this.key   = key;
         this.value = value;
         this.next  = next;
      }
   }

   public void put(String k, Object val) {
      first = new Node(k, val, first);
   }

   public Object get(String k) {
      for (Node x = first; x != null; x = x.next)
         if (k.equals(x.key)) return x.value;
      return null;
   }
}
```

a linked list of key-value pairs

insert at front of list

not found

exhaustively search for key

Object

Class `Object`. All objects "inherit" certain methods from the special superclass `Object`.

Method	Description	Default	Example
`toString()`	convert to string	memory address	`"hello " + s`
`equals()`	are two object values equal?	are two memory addresses equal?	`if (s.equals(t))`
`hashCode()`	convert to integer	memory address	`s.hashCode()`

`Complex`, `Wave`, `String`

Consequence. Can have a symbol table whose values are any type.

Annoyance. Must cast the return value of `get()` to desired type.

can avoid cast using "generics"

Linked List Implementation: Performance

Advantages. Not much code, fast insertion.

```
% java DeDup < toSpamList.txt
wayne@cs.princeton.edu
chlamtac@cs.princeton.edu
dgabai@cs.princeton.edu
cdecoro@cs.princeton.edu
cbienia@cs.princeton.edu
```

```
% java Dedup < mobydick.txt
moby
dick
herman
melville
call
me
ishmael
some
years
ago
...
```

210,028 words
16,834 distinct

Disadvantage. Search is hopelessly slow for large inputs.

hours to dedup Moby Dick

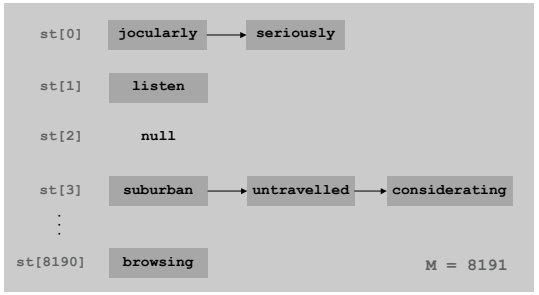
Hashing

Goal. Speed up search by using an array `st[]` of `M` linked lists.

up to factor of M times faster (!)

Hashing. Map from key to integer `i` between `0` and `M-1`.

- `put(key, value)`: add key-value pair to i^{th} linked list.
- `get(key)`: only need to search for key in i^{th} linked list.



key	hash
call	7121
me	3480
ishmael	5017
seriously	0
untravelled	3
suburban	3
...	.

Choosing a Good Hash Function

Hashing goal: scramble the keys.

- Efficiently computable.
- Each table position equally likely for each key.

thoroughly researched problem

Ex. [Social Security numbers]

- Bad: first three digits. 573 = California, 574 = Alaska
- Better: last three digits. assigned in chronological order within a given geographic region

Ex. [phone numbers]

- Bad: first three digits.
- Better: last three digits.

Hash Function: String Keys

Hash code. Java method returns a 32 bit integer.

```
public int hashCode() {
   int hash = 0;
   for (int i = 0; i < s.length(); i++)
      hash = (31 * hash) + s[i];
   return hash;
}
```

ith character of s

char	Unicode
...	...
'a'	97
'b'	98
'c'	99
...	...

- Equivalent to: $hashcode = 31^{L-1}s_0 + 31^{L-2}s_1 + \cdots + s_{L-1}$

Hash value. Convert hash code to integer between `0` and `M-1`.

$3045982 = 31^3(99) + 31^2(97) + 31(108) + (108)$

```
key       = "call";
h         = key.hashCode();
hash(key) = h % M;
```

3045982 % 8191 = 7121

Symbol Table: Hash Table Implementation

```
public class SymbolTable {
   private int M = 8191;   ← number of chains (often a prime)
   private Node[] st = new Node[M];

   private class Node {  // AS BEFORE  }

   private int hash(String key) {
      return Math.abs(key.hashCode() % M);
   }                                          between 0 and M-1

   public void put(String key, Object val) {
      int i = hash(key);
      st[i] = new Node(key, val, st[i]);
   }                                          insert at front of ith chain

   public Object get(String k) {
      int i = hash(k);
      for (Node x = st[i]; x != null; x = x.next)
         if (k.equals(x.key)) return x.value;
      return null;
   }                                          exhaustively search ith chain for key
}
```


Hash Table Implementation: Performance

Advantages. Fast insertion, fast search.

Disadvantage. Hash table has fixed size `M`.

can be corrected

Remark. Hash tables improves all symbol table clients.

Moby Dick now takes a few seconds instead of hours

```
% java DeDup < mobydick.txt
moby
dick
herman
melville
call
me
ishmael
some                    210,028 words
...                     16,834 distinct
```

Bottom line. Difference between a practical solution and no solution.

Question

Q. How to efficiently search for an IP address given a URL?

A. Hash table.

"DNS lookup"

Q. What if we want to search for a URL given an IP address?

"reverse DNS lookup"

Symbol Table Summary

Symbol table. Quintessential database lookup data type.

Choices. Different characteristics with different implementations.

- Linked list, hash table, binary search tree, ...
- Java has built-in libraries for symbol tables.
  - `HashMap` = hash table implementation.
  - `TreeMap` = "red-black" tree implementation.

```
import java.util.HashMap;
public class HashMapDemo {
   public static void main(String[] args) {
      HashMap st = new HashMap();
      st.put("www.cs.princeton.edu", "128.112.136.11");
      st.put("www.princeton.edu",    "128.112.128.15");
      st.put("www.simpsons.com",     "209.052.165.60");
      st.remove("www.simpsons.com");
      System.out.println(st.get("www.cs.princeton.edu"));
   }
}
```

ADT Advantages

Encapsulation.

- Useful for small applications.
- Essential for large ones.

Issues of ADT design.

- Feature creep.
- Formal specification problem.
- Implementation obsolescence.

Ex. Build large software project.

- Software architect specifies design specifications.
- Each programmer implements one module.

Ex. Build reusable libraries.

- Language designer extends language with ADTs.
- Programmers share extensive libraries.