Abstract Data Types

Data type: set of values and operations on those values.
Ex: `int`, `String`, `Complex`, `Card`, `Deck`, `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.
- INTERFACE: contract between client and class.

Intuition

Client

Interface
- volume
- change channel
- adjust picture
- decode NTSC, PAL signals

Implementation
- cathode ray tube
- electron gun
- Sony Wega 36XBR250
- 241 pounds, $2,699

client needs to know how to use interface

implementation needs to know what interface to implement

Implementation and client need to agree on interface ahead of time.

Intuition

Client

Interface
- volume
- change channel
- adjust picture
- decode NTSC, PAL signals

Implementation
- gas plasma monitor
- Pioneer PDP-502MX
- wall mountable
- 4 inches deep
- $19,995

client needs to know how to use interface

implementation needs to know what interface to implement

Can substitute better implementation without changing the client.

ADT Implementation in Java

Java ADTs.

- Keep data representation hidden with `private` access modifier.
- Define interface as operations having `public` access modifier.

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

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

Advantage: can switch to polar representation without changing client.
Note: all of the data types we have created are actually ADTs!

Y2K And Other Time Bombs

Time bombs.

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

```
public class Date {
   int seconds;

   public int getSeconds()
   public int getMinutes()
   public int getHours()
   public int getDays()
   public boolean after(Date d)
}
```

```
public class Date {
   private int seconds;
      ⬆
   public int  getSeconds()
   public void setSeconds()
   public int  getMinutes()
   public void setMinutes()
   public boolean after(Date d)
}
```

```
Date d = new Date();
d.seconds = 31334534;
```

legal (but bad) Java client

```
Date d = new Date();
d.seconds = 31334534;
```

illegal Java client

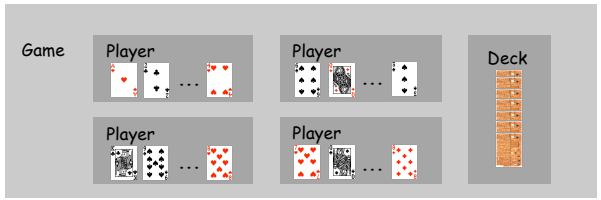
Modular Programming and Encapsulation

ADTs enable modular programming.

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

ADTs enable encapsulation.

- Keep modules independent (include `main` in each class for testing).
- Can substitute different classes that implement same interface.
- No need to change client.



4.7: 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.

Example: 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

Other Symbol Table Applications

Other applications.

- Online phone book: look up a name to find telephone number.
- Spell checker: look up a word to find if it's there.
- ➡ DNS: look up name of web site to find IP address.
- Java compiler: look up variable name to find its type and value.
- File sharer: look up song to find host machines.
- File system: look up file name to find location on hard drive.
- University registrar: look up student to find grades.
- Google: look up phrase and return most relevant web pages.
- Web cache: cache frequently accessed pages.
- Routing table: look up routing info for IP.
- Browser: highlight visited links in purple.
- Bayesian spam filter: use frequencies of spam and ham words to filter email.
- Book index: determine pages on which each word appears.
- "Associative memory."
- Index of any kind.
- . . .

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.cs.princeton.edu", "128.112.136.11");
   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.cs.princeton.edu"));
   System.out.println(st.get("www.harvardsucks.com"));
   System.out.println(st.get("www.simpsons.com"));
}                                 ↑
                         st["www.simpsons.com"]
```

```
128.112.136.11
null
209.052.165.60
```


Symbol Table Client: Remove Duplicates

Remove duplicates. (from a mailing list or voting eligibility list)

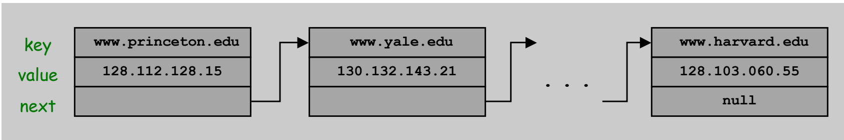
- Read in `key`.
- If `key` is not in symbol table, print out `key` and insert it.

```
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.
- Key = `String`, value = `Object`.
- Use exhaustive search to search for a key.



Symbol Table: Linked List Implementation

```
public class SymbolTable {
   private Node first;      <- a linked list of key-value pairs

   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);
   }                                          insert at front of list

   public Object get(String k) {
      for (Node x = first; x != null; x = x.next)
         if (k.equals(x.key)) return x.value;
      return null;
   }              not found          exhaustively search for key
}
```


Object

Class `Object`.

- All objects "inherit" from the special class `Object`.
- All objects have certain pre-defined methods.

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

Consequences.

- Can have a symbol table of any type of object, e.g, `String` or `Wave`.
- Cast the return value of `get` to desired type.

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
```

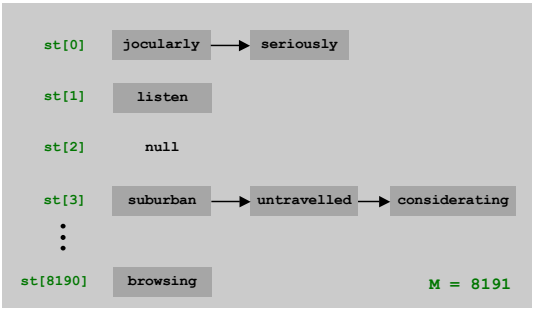
Disadvantage: search is hopelessly slow for large inputs.

hours to dedup Moby Dick

Hashing

Hashing.

- Goal: speed up search by a factor of M by making lists shorter.
- Array `st` of M chains (linked lists).
- Map from string key to integer i between 0 and M-1.
  - put key-value pair in i^{th} linked list



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

Choosing a Good Hash Function

Goal: scramble the keys.

- 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: Strings.

- Bad: first few letters (converted to `int`).
- Good: do calculation involving all characters.

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

String.java

```
s    = "call";
h    = s.hashCode();
hash = h % M;
   7121     8191     3045982
```

insert `"call"` into chain 7121

Symbol Table: Hash Table Implementation

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

   private class Node { AS BEFORE }

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

   public void put(String k, Object val) {
      int i = hash(k);
      st[i] = new Node(k, 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. (can be corrected)

Hash tables improves ALL symbol table clients.

- Makes difference between practical solution and no solution.
- Ex: Moby Dick now takes a few seconds instead of hours.

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


Question

Current code searches for an IP address given a URL.

- "DNS lookup."

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.

Different performance 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

Modular programming and encapsulation.

- Essential for many real applications.
- Crucial software engineering principle.

Issues of ADT design.

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

Ex: building large software project.

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

Ex: build libraries.

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