Index of /~cs61b/lectures/14

Name	Last modified	Size	Description
Parent Directory	24-Sep-98 21:22	-	
list/	25-Sep-98 00:58	-	

```
                     CS61b: Lecture 14
                    Friday, September 25

ANNOUNCEMENT: Midterm Mon. 10-11am, Last name A-M in 155 Dwinnelle, N-Z in Sibley.
Old exams of mine in the course reader.
Review session on Sunday evening at 6pm in 306 Soda.

FINALLY (finishing exceptions)
-------
A finally clause can also be added to a "try."  This will execute in any case.

  try {
    S1;
    return 1;
  } finally {
    return 2;
  }

Will always return 2.  This example is contrived.  Finally clauses
are used to do things like close files, that need to be done in either
normal or exceptional executions.

PACKAGES
--------
For very large programs, there may be two different classes with the same name.
Packages in Java allow this, e.g., java.awt.Panel and ford.parts.Panel.

A "package" is a collection of types (classes & interfaces) and (sub)packages.
Two benefits of using a package:
  1) can have hidden classes that are used by the package but not visible outside.
  2) can have data fields that are visible by all classes in the package, but
    not outside.

Examples:
  1) java.io is a package of the I/O-related classes.
  2) a list package containing List, ListNode, and ListEnum classes.

Package names are hierarchical, e.g., java.awt.image.Model refers to:
   the class Model inside package image inside package awt inside package java

Tedious to use the "fully qualified" names all the time, so we use import:
 import java.io.File;  // can now refer to File things without java.io.
 import java.io.*;     // can now refer to all things in java.io

Implicit import of java.lang.* at the beginning of every program.
```

BUILDING PACKAGES

When someone imports one of your packages, they only see the public classes.

```
  public class { ... }
```

The javac compiler requires that:
 1) all public classes appear in a file with the same name as the class.
 2) the name of the package matches the name of the directory

```
/* List.java */                      /* ListNode.java */

package list;                        package list;

public class List {                  class ListNode {
  ....                                 Object item;
  ListNode head;                       ListNode next;
}                                      }
```

List class can create and use ListNode objects.
List class can access item and next fields in ListNode object.
Other classes outside the package cannot see these.

ENUMERATIONS

In java.util there is a standard interface for iterating over objects.

```
  interface Enumeration {
     public boolean hasMoreElements();
     public Object nextElement();
  }
```

IMPLEMENTING LIST ENUMERATIONS

A "Enumeration" is a control abstraction. Needs to keep track of the data structure and the location in the data structure. For a List, this just means the current ListNode.

```
/* ListEnum.java */
package list;
import java.util.*;

public class ListEnum implements Enumeration {
  public ListEnum( ListNode l ) { trav = l; }
  public boolean hasMoreElements() { return (trav != null); }
  public Object nextElement() {
    Object retItem = trav.item;
    trav = trav.next;
    return retItem;
  }
  private ListNode trav;
}
```

and inside List.java:

```
  public Enumeration elements() {   // note that return type is Enumeration
    return new ListEnum(head);      // not ListEnum
  }
```

ENUMERATIONS ON OTHER TYPES

The values over which you’re enumerating need not be explicitly represented:

```
class Primes implements Enumerate {
   // build enumeration for all primes less than x
   Primes(int x) { ... }
   ....  }
```

This is a nice way to encapsulate complex indexing:

```
class Neighbors implements Enumeration {  // only N,S,E,W unlike pj2
  public Neighbors (int i, int j, Ocean o) {
    count = 4;  x = i;  y = j;
  }
  public boolean hasMoreElements() { return (count > 0);  }
  public Object nextElement() {  // not quite right for edges
    if (count == 4) return (new Point (x, y-1));
    if (count == 3) return (new Point (x+1, y));
    if (count == 2) return (new Point (x, y+1));
    if (count == 1) return (new Point (x-1, y));
  }
  int count; int x; int y;
}
```