Chapter 13 Collections (Data Structures)

Page 658

**ArrayList**: An indexed sequence that grows and shrinks dynamically.

**LinkedList**: An ordered sequence that allows efficient insertions and removal at any location.

**HashMap**: Stores key/value associations

**TreeSet**: Uses total ordering to store key/value associations.
**Java ArrayList**

- An `ArrayList` is a *resizable-array* of *object references* in *continuous memory*.

- Each `ArrayList` instance has a capacity. The capacity is the size of the array used to store the elements in the list.

- The capacity is always at least as large as the list size.

- As elements are added to an `ArrayList`, its capacity grows automatically.

- Details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.

**ArrayList Constructors (partial list)**

- `ArrayList()`  
  - Constructs an empty list with an initial capacity of ten.

- `ArrayList(int initialCapacity)`  
  - Constructs an empty list with the specified initial capacity.

- `ArrayList<classname>(int initialCapacity)`  
  - Constructs an empty list with the specified initial capacity.
  - Allows the compiler to check that all adds and gets insert and return an object that subclasses `classname`. 
Java ArrayList Methods (partial list)

- `add(Object o)`
  Appends the specified element to the end of this list.

- `add(int index, Object element)`
  Inserts the specified element at the specified position in this list.

- `Object get(int index)`
  Returns the element at the specified position in this list.

- `clear()`
  Removes all elements from this list.

Java ArrayList Methods (partial list)

- `ensureCapacity(int minCapacity)`
  Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

  An application can increase the capacity of an ArrayList instance before adding a large number of elements. This may reduce the amount of incremental reallocation.
Java ArrayList Methods (partial list)

- boolean contains(Object elem)
  Returns true if this list contains the specified element. *Note: this only checks equality of element references.*

- int indexOf(Object elem)
  Searches for the first occurrence of the given argument, testing for equality using the equals method.

- int size()

- Object[] toArray()

ArrayList: Access, Add and Insert

An ArrayList is a continuous block of memory. The $i^{th}$ element can be accessed in constant time.

ArrayList.add(element): \textit{constant time}

\[
\begin{array}{c|c}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} \\
\end{array}
\quad
\begin{array}{c|c}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{X} \\
\end{array}
\]

ArrayList.add(index, element): \textit{O(n) time}

\[
\begin{array}{c|c}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} \\
\end{array}
\quad
\begin{array}{c|c}
\text{A} & \text{B} & \text{C} & \text{X} & \text{D} & \text{E} & \text{F} \\
\end{array}
\]

 LinkedList Access and Insertion

- A LinkedList is, in general, a disjoint set of memory.
- The $i$th element can be accessed in $O(n)$ time.

```
A ➔ B ➔ C ➔ D ➔ E
```

- LinkedList.add(index, element): constant time

```
A ➔ B ➔ C ➔ D ➔ E
```

 Prime Factors using ArrayList: Part 1

```java
import java.util.ArrayList;

public class HelloWorld
{
    public static int[] primeFactors(int n)

    public static String intArrayToStr(
        int[] intArray)

    public static void main(String[] args)
}
public static int[] primeFactors(int n) {
    int testNumber = 2;
    int endTest = (int)Math.ceil((Math.sqrt(n)));
    ArrayList<Integer> factors = new ArrayList<Integer>();
    while (testNumber <= endTest) {
        if (n % testNumber == 0) {
            factors.add(new Integer(testNumber));
            n = n/testNumber;
            if (n==1) break;
        }
        else testNumber++;
    }
    if (n>1) factors.add(new Integer(n));
    int[] factorArray = new int[factors.size()];
    for (int i = 0; i<factors.size(); i++) {
        factorArray[i] = factors.get(i);
    }
    return factorArray;
}

Quiz: Prime Factors – Output?

public static int[] primeFactors(int n) {
    int testNumber = 2;
    int endTest = (int)Math.ceil((Math.sqrt(n)));
    ArrayList<Integer> factors = new ArrayList<Integer>();
    while (testNumber <= endTest) {
        if (n % testNumber == 0) {
            factors.add(new Integer(testNumber));
            n = n/testNumber;
            System.out.print("test=":testNumber+ ": ");
            if (n==1) break;
        }
        else testNumber++;
    }
    
    a) test=2 : test=3 : test=4 : test=5 :
    b) test=2 : test=3 : test=3 : test=4 : test=5 :
    c) test=3 : test=3 : test=3 : test=5 :
    d) test=3 : test=3 : test=5 :
Graphs and Trees

Data Structure for a Graph

The way a graph is represented and the information stored depends on the needs of the application.

- Are the edges directed?
- Is there a logical root?
- In what ways will the data in the graph need to be accessed?
- Can there be more than one path to a node? (graph or tree).
- How will the graph / tree be build?
  - All at once?
  - Will insertions need to be made between existing edges?
Lab 13: Due Wed, Dec 9 @ midnight

Node root = new Node("Vertebrate", null);
Node mammal = new Node("Mammal", root);
Node bird = new Node("Bird", root);
Node finch = new Node("Finch", bird);
Node owl = new Node("Owl", bird);
Node reptile = new Node("Reptile", root);
Node lizard = new Node("Lizard", reptile);
Node snake = new Node("Snake", reptile);
Node boa = new Node("Boa", snake);
Node python = new Node("Python", snake);
Node pitViper = new Node("Pit-Viper", snake);
Node copperhead = new Node("Copperhead", pitViper);
Node rattlesnake = new Node("Rattlesnake", pitViper);
Node gecko = new Node("Gecko", lizard);
Node chameleon = new Node("Chameleon", lizard);
Node snowy = new Node("Snowy", owl);
Node flammulated = new Node("Flammulated", owl);
snake.printChildren(); // print names of all decedents.