Program 6 up this afternoon
Next OWL assignment due friday

Topics:
Big Programs:
  Manhattan Binary Search
  Word Work and StringTokenizer
  Sieve Prime finder
Topics for today
Some more ambitious programs

Inverse Manhattan
Dice in pictures
Word Frequency

**Eratosthenes**- Sieve for primes
public class CompoundInterest{

    private double rate;
    private double amount;
    private double fee = 0.0; // yearly fee
    private int years;
    final double tolerance = 100.00; // used by findRate

    public double getAmount(){ return amount; }

    public double getFee(){ return fee; }

    public void setFee(double fee){
        this.fee = fee; }

    public double getRate(){
        return rate; }
}
public CompoundInterest(double r, double start, int years) {
    amount = start;
    rate = r;
    this.years = years;
}

public CompoundInterest(double r, double start, int years, double fee) {
    amount = start;
    rate = r;
    this.fee = fee;
    this.years = years;
}

public CompoundInterest() {
    amount = 0.0;
    rate = 0.0;
    years = 0;
}
public double calcAmt()
{
    // calculates amount accrued for given rate, start amount
    // a fee is deducted every year
    double amt = amount;
    for(int y = 0; y < years; y++)
    {
        amt = (1 + rate)*amt - fee;
    }
    return(amt);
}

public double calcAmt(double someAmt,int years,double r){
    // r is any interest rate, amount any start amount
    // calculates amount accrued for this rate, initial amount
    double amt = someAmt;
    for(int y = 0; y < years; y++)
    {
        amt = (1 + r)*amt - fee;
    }
    return(amt);
}

// calcAmt is overloaded
The inverse Manhattan problem:

Given a start amount, a target amount, a number of years

What interest rate gets you from start to target?
First panel: spread of 10% between mid and high
Second panel: spread of 5% between low and mid
Next try: 12.5%
public double findRate(double amt, double target, int years) {
    double low = 0.0;
    double high = findUpBound(amt, target, years);
    double mid = (high + low) / 2; // mid: middle rate
    double curRate = mid;
    double curAmt = calcAmt(amt, years, mid);
    while (Math.abs(curAmt - target) > tolerance) {
        System.out.println("current middle rate: " + mid);
        if (curAmt > target) {
            high = mid;
            mid = (mid + low) / 2; }
        else{
            low = mid;
            mid = (mid + high) / 2; }
        curAmt = calcAmt(amt, years, mid);
    } // end loop
    return mid;
}
public double findUpBound(double amt, double target, int years) {
    final double increment = .10; // guess, going up by 10% each year
    double rate = increment;
    while (calcAmt(amt, years, rate) < target) {
        rate = rate + increment;
    } // loop
    return rate;
}
As text indicates:

$24

380 years

Target of one trillion dollars:

rate required 0.0665 (6.65%)
Another application -

A simple word frequency program

Reads in multiple lines of text (end with empty line: 2<CR>s)

Break the lines into words (tokens)

Store words in an array, keeping track of word multiplicities

Five participating classes
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>fox</td>
<td></td>
<td>good</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

WordData objects in array in WordStore
import java.util.*;
public class WordDriver{
    public static void main(String[] args){
        WordStore store = new WordStore();
        Scanner scan = new Scanner(System.in);
        StringTokenizer str;
        String t = " "; String s;
        System.out.println("Enter lines, two returns to end");
        while(t.length() > 0){
            t = scan.nextLine(); // read next line
            t = t.toLowerCase(); // convert line to all lower case
            str = new StringTokenizer(t, "\t\n\r\f,.?!;: ");
            while(str.hasMoreTokens()){  
                s = str.nextToken();
                store.updateWords(s);
            }  
        }
        store.wordReport(); } }
public class WordData{
    private String word;
    private int count;

    public WordData(String w){
        word = w;
        count = 1; }

    public String getWord(){ return word; }

    public int getCount(){return count;}

    public void incCount() {count++;
    }

    public String toString() {
        return(word + " ---- " + count);
    }
}
}
public class WordStore{
    final int LAST_WORD = 100;
    private WordData[] words = new WordData[LAST_WORD];
    private int lastEmpty = 0;

    public void updateWords(String w){
        int where = -1; // -1 means - word not yet found
        where = findWord(w);
        if (where >= 0) words[where].incCount();
        else addWord(w);
    }

    public int findWord(String w){
        int ans = -1;
        for(int j=0; j < lastEmpty; j++){
            if (w.equals(words[j].getWord())){
                ans = j;
                break;
            }
        }
        return ans; }
}
public void addWord(String w){
    if ((lastEmpty + 1) == LAST_WORD)
        System.out.println("word store full "+w+" not added");
    else{
        words[lastEmpty] = new WordData(w);
        lastEmpty++;
    }
}

public void wordReport(){
    for(int j = 0; j < lastEmpty; j++)
        System.out.println(words[j].toString());
}
}
import java.util.StringTokenizer;

public class TokenizerTest{
    public static void main(String[] args){
        StringTokenizer str;
        Scanner scan = new Scanner(System.in);
        System.out.println("enter a line of text");
        String s = scan.nextLine();
        str = new StringTokenizer(s);
        while (str.hasMoreTokens()){
            System.out.println(str.nextToken());
        }
    }
}

enter a line of text

now, yes now is the time!
now,
yes
now
is
the
time!
import java.util.StringTokenizer;

public class TokenizerTest{
    public static void main(String[] args){
        StringTokenizer str;
        Scanner scan = new Scanner(System.in);
        System.out.println("enter a line of text");
        String s = scan.nextLine();
        str = new StringTokenizer(s,".,");
        while (str.hasMoreTokens()){
            System.out.println(str.nextToken());
        }
    }
}

The crucial line: notice 2nd argument
now, yes now is the time!
  now
  yes
  now
  is
  the
  time!

So StringTokenizer treats ",," as a sort of white space, but not "!"
So what’s a prime number?

2, 11, 13, 17, 19 - NO proper divisors
6, 15, 21, 51 (= 3x17!) are not primes

Old concept - Eratosthenes (275-195 BC) came up with a Sieve method for finding primes up to some number n.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
x2 > 2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
x3>3
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
x5>5
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Basic Idea of implementation:

An array of booleans.

“true” at cell j means: “ j is not a multiple of any earlier number”

Initially array is all true

First pass: all multiples of 2 (e.g. 4, 6, 8, …) all set to false

Then: all multiples of 3 (e.g. 6, 9, 12, …) all set to false

Then: 5 (skip 4: it’s already false-> mult of 2)
// we’ll do primes < 100

public class SieveTester {
    public static void main(String[] args) {
        Sieve s = new Sieve(100);
        s.init();  // initialize row of numbers
        s.process();  // run sieve process
        s.report();  // report primes
    }
}

public class Sieve {
    int top;
    boolean[] nums;

    public Sieve(int cap) {
        top = cap;
    }

    public void init() {
        nums = new boolean[top];
        for (int j = 0; j < top; j++) nums[j] = true;
    }

    // Notice sieve (array) size decided at run-time
public void process()
    {
        for (int j = 2; j < top; j++)
            if (nums[j] == true) killMultiples(j);
    }

public void report()
    {
        for (int j = 2; j < top; j++)
            if (nums[j])
                System.out.println(j);
    }

public void killMultiples(int k){ // kills 2*k, 3*k etc
        for (int j = 2; j*k < top; j++)
            nums[j*k] = false;
    }