CSUN | CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

COMP110

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Rev 5-6-22

Intro to Algorithms & Programming

LAB

Part 2

Dr Jeff Drobman

Dr Jeff Software

website → *drjeffsoftware.com*

email → *jeffrey.drobman@csun.edu*

Index

- Labs 4-8
- Project 2

Lab

- Labs

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Lab 4: Palindromes/Anagrams

DR JEFF SOFTWARE INDIE APP DEVELOPER

- *Methods*
- *Strings*

➢ GUI

Rqts– INPUT: OUTPUT:
1) words 1) Palindrome?
2) confirms 2) Anagrams?

➢ GUI

❖ Use check "Methods"
- ❑ IsPal
- ❑ IsAnagram

as intended?

INPUT:
1. Word or phrase
2. Continue?

PROCESS (source code)
1) Input Words
2) Check IF Palindrome
3) Check IF Anagrams –
→**only if 2 words**
4) Confirm continue

OUTPUT:
1) Palindrome? Y/N
2) Anagrams? Y/N/X

correct program?

DEBUGGING/TESTING

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Lab 4: Palindromes

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 4

5
---
civic
kayak
level
madam
put-up
radar
rotor
sagas
Saras
sexes
stats
stets
tenet

6

Hannah

7
---
racecar
repaper
reviver
rotator
---
double word
---
race car
a Toyota

paired words (f/b)
---
bad dab
bat tab
but tub
car arc
dog god
eel lee
gab bag
gal lag
gar rag
gas sag
gat tag
lap pal
lop pol
mad dam
mag gam
mat tam
mid dim
nab ban
net ten

phrases
---
madam I'm Adam
a man, a plan, a canal: panama
straw no; poop on warts
he saw I was eh?
doom a mood
no saw was on

Lab 4: Anagrams


```
carved, craved
carves, craves
casual, causal
caters, caster, recast
cleans, lances
course, source
cosmic, comics
dearer, re-dare, reader
demote, emoted
depart, parted
deport, ported
depots, despot
derate, teared
detent, tented
devote, vetoed
```

```
*7*
-----
alerted, altered
animals, laminas
bearded, breaded
boarder, broader
booster, booters
carving, craving
casuals, causals
cavalry, calvary
chancer, chancre
closets, closest
courses, sources
dairies, diaries
dealing, leading
defocus, focused
dieting, editing
drawing, warding
dresser, redress
```

```
*8*
-----
alerting, altering
altitude, latitude
antimony, antinomy
carvings, cravings
casually, causally
chancers, chancres
compiled, complied
compiles, complies
conserve, converse
customer, costumer
discreet, discrete*
dealings, leadings
```

*homonyms also

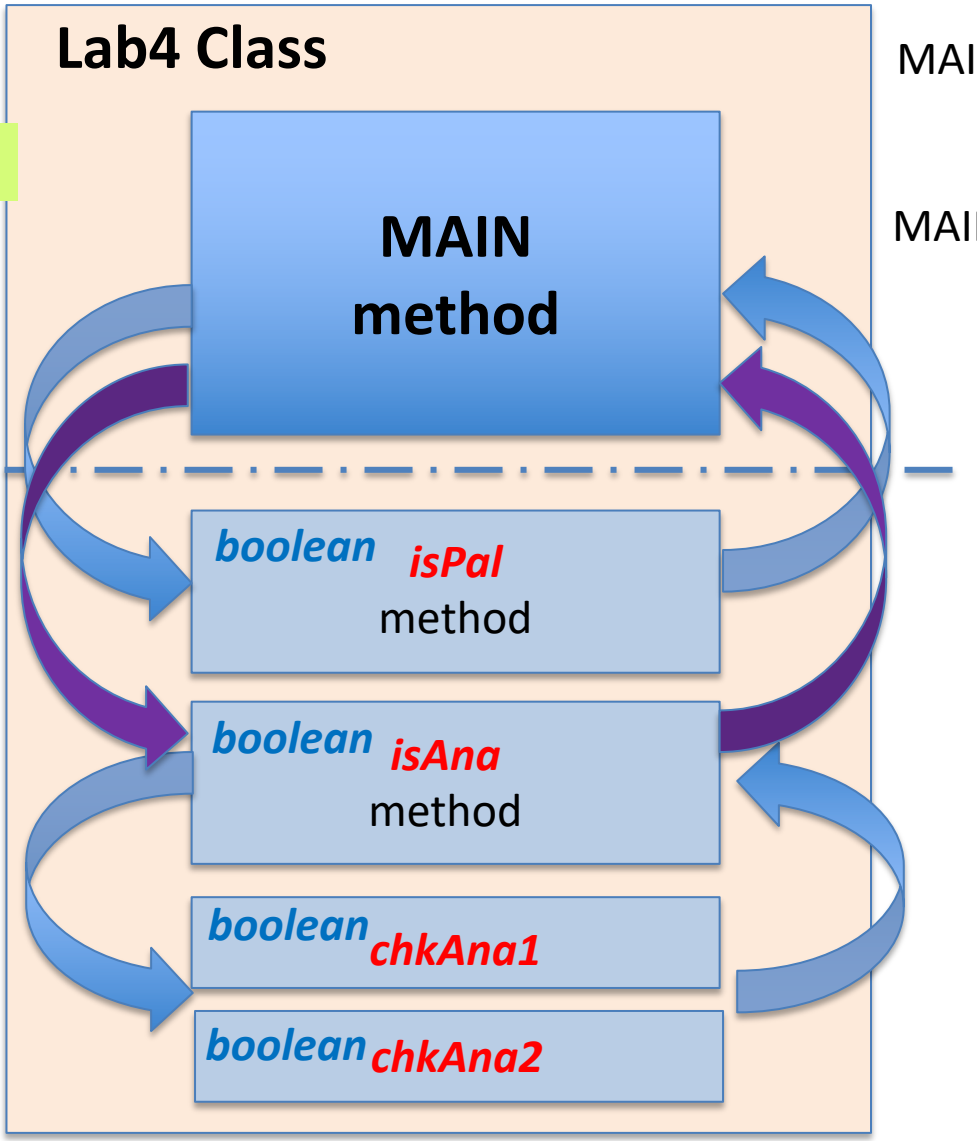
Structure (Macro)

Java ⇔ OOP

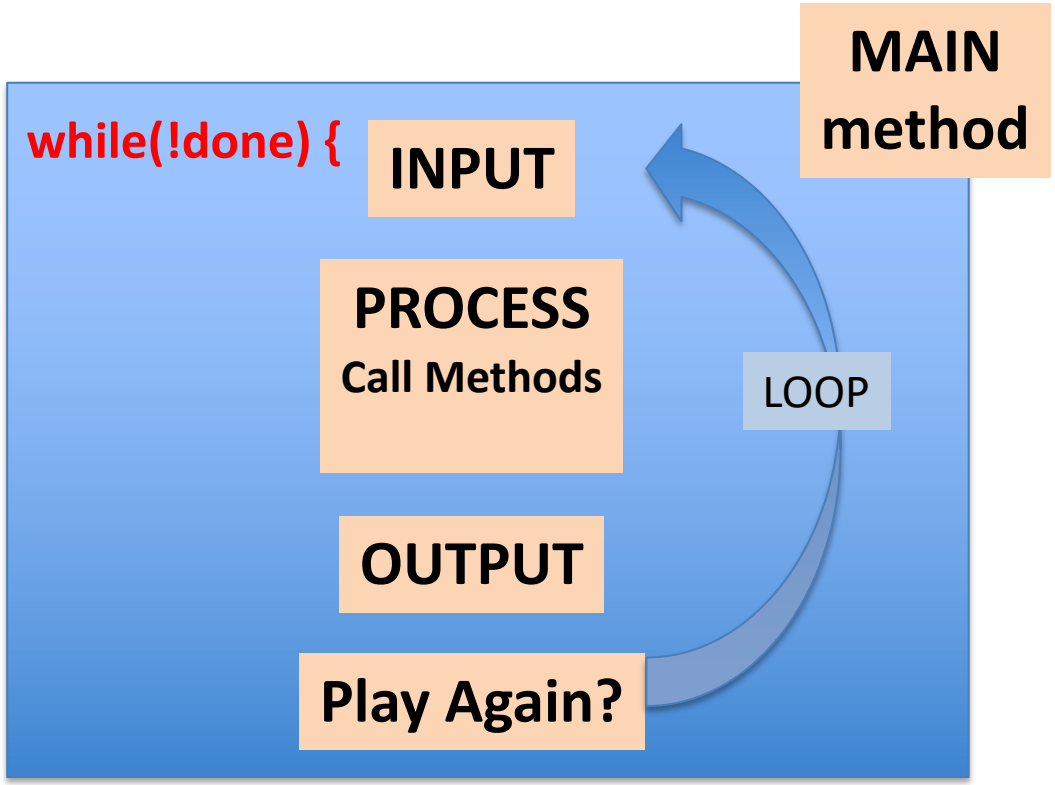
OOP
Structures

Classes/methods

Execution is by *call* sequence



Main Structure: IPO



Methods

Lab 4

```
public static void main(String[ ], args)  {
      <statements– Input + filter>
      boolean palFlag = IsPal(inStr);
      boolean anaFlag = IsAna(inStr);
      <statements– Output>
}
```

❖ *Pseudo-code*

```
public static boolean IsPal(String strParm)  {
      boolean result = true;
      <statements– pre-process:  filter>
      <statements-from book>
      return result;
}
```

```
public static boolean IsAna(String strParm)  {
      boolean result = true;
      <statements– pre-process:  filter + extract>
      <statements– check if 2 words, then equal length>
      result1 = chkAna1(xStr);
      result2 = chkAna2(xStr);
      return result1;  }
```

Main – Input


```
// imports
import javax.swing.*;
import java.util.Arrays;
// **main class**
public class Lab4 {
   static final boolean $DEBUG = true;
   static final String spc = " ";
   static String reason = "";
//main method
   public static void main(String[] args) {
    if ($DEBUG) System.out.println("debug: starting main");
    int playNum = 0;
    String inStr = "";
    boolean done = false;
    while(!done) {//main outer loop
      playNum++;
      //user enters words or phrase after 2 hard coded
      switch(playNum) {
         case 1: inStr = "madam I'm Adam"; break;
         case 2: inStr = "course   source"; break;
         default: inStr = JOptionPane.showInputDialog("Enter words or phrase");
      }//end switch
```

Input

Main – Process


```
28        //call check for both Pals and Anas
29        //output msg:  is/not Pal + is/not Ana or not 2 words
30        //ask user:  "play again?" --> Loop or Quit
31        //check Pals method:
32        //use textbook algorithm after filter non-alpha
33        boolean palFlag = isPal(inStr);
34        String palMsg = "Is a palindrome: ";
35        if ($DEBUG) System.out.println(palMsg + palFlag);
36        //check Anagrams method:
37        boolean anaFlag = isAna(inStr);
38        String anaMsg = "Words are anagrams: ";
39        if ($DEBUG) System.out.println(anaMsg + anaFlag);
```

Main – Output & Ask

Output

```
38  //code to display results: is Pal, is Ana (and why not)
39  JOptionPane.showMessageDialog(null, inStr + //4 lines
40        "\n" +palMsg + palFlag +
41        "\nWords are anagrams: " + anaFlag +
42        "\nreason: " + reason);
43     //ask user if wants to continue?
44     int rep = JOptionPane.showConfirmDialog(null, "Try again?");
45     if (rep >0) {
46        done = true;
47        String msg = "You quit, good-bye!";
48        JOptionPane.showMessageDialog(null, msg);
49        System.out.println(msg);
50     }//end if
51   }//end loop
52  }//end main
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Textbook Palindromes

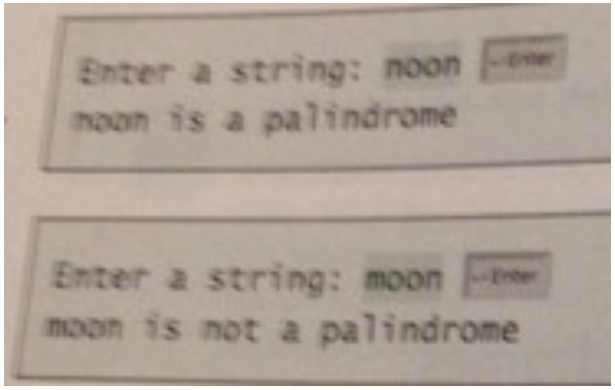
DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 4

Listing 5.14
pp. 187-188

LISTING 5.14 Palindrome.java

```
import java.util.Scanner;

public class Palindrome {
  /** Main method */
  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter a string
    System.out.print("Enter a string: ");
    String s = input.nextLine();

    // The index of the first character in the string
    int low = 0;

    // The index of the last character in the string
    int high = s.length() - 1;

    boolean isPalindrome = true;
    while (low < high) {
      if (s.charAt(low) != s.charAt(high)) {
        isPalindrome = false;
        break;
      }
      low++;
      high--;
    }

    if (isPalindrome)
      System.out.println(s + " is a palindrome");
    else
      System.out.println(s + " is not a palindrome");
  }
}
```




```
boolean $flag = true;
while (low < high) {
if lowchar != hichar {
   $flag = false;
   break; }
low++; high--;
}
return $flag;
```

Filtering

Lab 4

❑ Palindrome

- ❖ Remove all non-letters
- ❖ Ignore case

➢ No of words: **does NOT matter**

```
input = input.replaceAll("[^a-zA-Z]+","").toLowerCase();
```

➢ using “regex”

❑ Anagrams

- ❖ Check if letters only
- ❖ Ignore case

➢ No of words: **must be 2**

Pals: Sample Code


```
if (charLow == charHigh)
{
    low++;
    high--;
}

else if (charLow == ',' || charLow == '.' ||
         charLow == '-' || charLow == ':' ||
         charLow == ';' || charLow == ' ')
{
    low++;
}
else if (charHigh == ',' || charHigh == '.' ||
         charHigh == '-' || charHigh == ':' ||
         charHigh == ';' || charHigh == ' ')
{
    high--;
```

❖ ***DON'T DO THIS!***

❖ ***DO THIS!***

```
String alpha = "abcdefghijklmnopqrstuvwxyz";
ix1 = alpha.indexOf(inchar1);
ix2 = alpha.indexOf(inchar2);
//check if both chars alpha
if (ix1 <0 || ix2 <0) return false;
```

```
input = input.replaceAll("[^a-zA-Z]+","").toLowerCase();
```

Check Anagrams

Lab 4

```
public static boolean isAna(String strParm)  {
      boolean result = true;
      <statements– check method:
            1. array counter, or
            2. string-to-array sort>
      return result;
}
```

- Use BOTH of these 2 algorithms:
  - chkAna1
  - chkAna2

Anagrams: isAna

isAna — Lab 4

```
45  static boolean isAna(String input) {
46     if ($DEBUG) System.out.println("debug: starting chkAna");
47  //check for 2 words: if not, then return
48  //if 2 words -> split into 2 words:  word1, word2
49  //check lengths: if not =, then return
50  //use ONE of these 2 algorithms with--> Arrays.equals
51  //Array:  counters
52  //String:  Sort char array
53     boolean fail = true;
54     String[] words = input.split("\\s+");
55     if ($DEBUG) {
56        for (String ss: words) System.out.print(ss +spc);
57        System.out.println("");
58     }
59     switch(words.length) {
60     case 0:
61        reason = "no words entered! try again.";
62        break;
63     case 1:
64        reason = "only 1 word entered! try again.";
65        break;
66     case 2:
67        reason = "2 words entered: checking";
68        fail = false;
69        break;
70     default:
71        reason = ">2 words entered! try again.";
72     }
```

- ❖ check for 2 words
- split into N words
- case N: no. words

Anagrams: isAna


```
73      if ($DEBUG) System.out.println(reason);
74      if (fail) return false;
75    //else continue: must be 2 words
76      String word1 = words[0];
77      String word2 = words[1];
78    //check lengths:  if not = then set result=false & return
79      if (word1.length() != word2.length()) {
80         reason = "words not same length";
81         return false;
82      }
83    //call 1 of these 2 algorithms, or both
84      boolean result1 = chkAna1(word1, word2);
85      boolean result2 = chkAna2(word1, word2);
86      if (result1 != result2) {
87      System.out.println("algorithm's results not same!");
88      return false;
89      }
90      System.out.println("input words " + word1 + " and " + word2 + " are
91      return result1;
92    }//end of isAna
```

- checked for 2 words

check = lengths

Call alg 1

Call alg 2

```
 ----jGRASP exec: java Lab4
debug: starting main
debug: starting chkAna
debug: starting chkAna1
abc counts= 0 0 1
debug: starting chkAna2
input words course and source are anagrams: true

 ----jGRASP: operation complete.
```

Anagrams: chkAna1

Alg 1 Lab 4

```
//new method 1-array counter
static boolean chkAna1(String w1, String w2) {
   if ($DEBUG) System.out.println("debug: starting chkAna1");
   char inchar1, inchar2;
   int ix1, ix2;
   boolean result;
   String alpha = "abcdefghijklmnopqrstuvwxyz";
   int wlen = w1.length();
   int[] count1 = new int[26];
   int[] count2 = new int[26];
   Arrays.fill(count1,0);//init arrays to 0
   Arrays.fill(count2,0);
   for (int i=0; i< wlen; i++) {
      inchar1= w1.charAt(i);
      inchar2= w2.charAt(i);
      ix1 = alpha.indexOf(inchar1);
      ix2 = alpha.indexOf(inchar2);
      //check if both chars alpha
      if (ix1 <0 || ix2 <0) return false;
      count1[ix1]++;
      count2[ix2]++;
   }
   if ($DEBUG) System.out.printf("abc counts= %d %d %d \n",count1[0],
//compare counts & set result
   return Arrays.equals(count1, count2);
}//end chkAna1
```

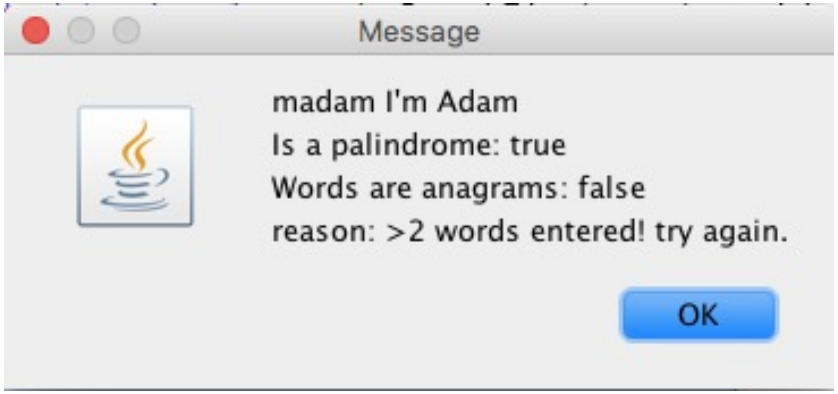
```
----jGRASP exec: java Lab4
debug: starting code
abc counts= 2 2 1
input words aabcb and bcaba are anagrams: true

----jGRASP: operation complete.
```

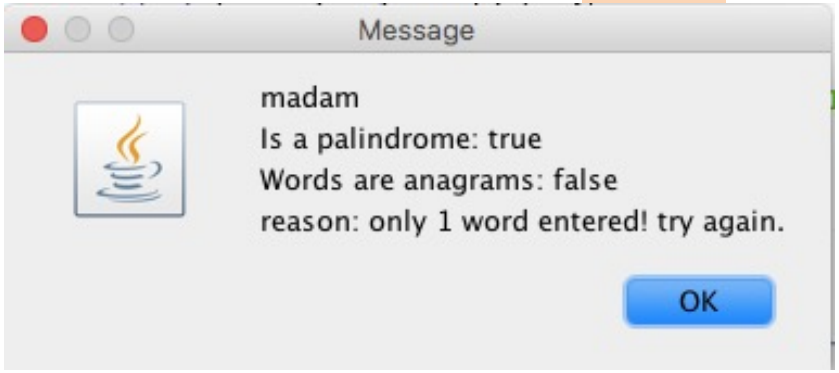
Anagrams: chkAna2

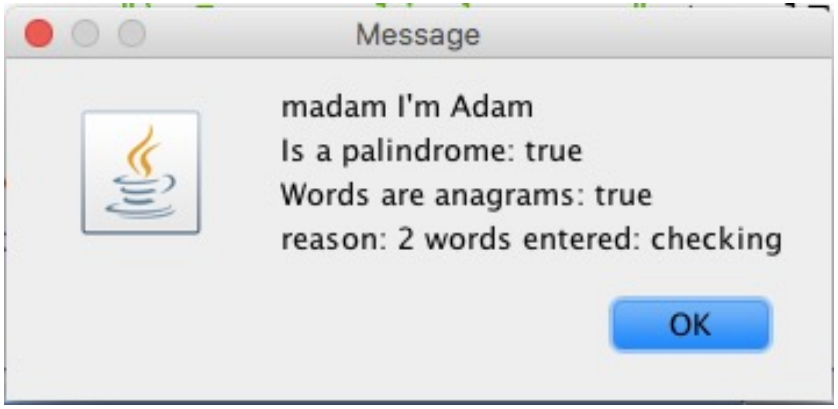

```
//new method 2-sort string
static boolean chkAna2(String w1, String w2) {
    if ($DEBUG) System.out.println("debug: starting chkAna2");
    char[] w1ch = w1.toCharArray();
    Arrays.sort(w1ch);
    char[] w2ch = w2.toCharArray();
    Arrays.sort(w2ch);
    return Arrays.equals(w1ch, w2ch);
}//end chkAna2
```

Output

Lab 4








```
 ----jGRASP exec: java Lab4
debug: starting main
lower input = madam i'm adam
filtered input = madamimadam
Is a palindrome: true
debug: starting chkAna
madam I'm Adam
>2 words entered! try again.

 ----jGRASP: operation complete.
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 5: Homonyms

Lab 5

- *Methods*
- *Strings*
- *Files*

- GUI

Rqts– INPUT:
1) *text file*
2) words
3) confirms

OUTPUT:
1) In dictionary?
2) Homonyms?

- GUI
- ***Submit file***

Create Dictionary:
>= 25 homonym pairs

Text file

INPUT:
1- Text file
2- Words
3- Continue?

PROCESS (source code)
1) Input Words
2) Check IF in dictionary
3) Check IF ***Homonyms***
4) Confirm continue

OUTPUT:
1- In dictionary?
2- Homonyms? Y/N

DEBUGGING/TESTING

Homonyms

Lab 5

homonym | ˈhämə,nim ˈhōmə,nim |

noun

each of two or more words having the same spelling but different meanings and origins (e.g., POLE[1] and POLE[2]); a homograph.

- each of two words having the same pronunciation but different meanings, origins, or spelling (e.g., TO , TOO, and TWO); a homophone.
- *Biology* a Latin name that is identical to that of a different organism, the newer of the two names being invalid.

Examples from the Web:

http://www.singularis.ltd.uk/bifroest/misc/homophones-list.html

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Word World

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Dr Jeff App

Microsoft Visual Studio (.NET Framework)

- Palindromes
- Anagrams
- Homonyms
- Homographs
- Malaprops

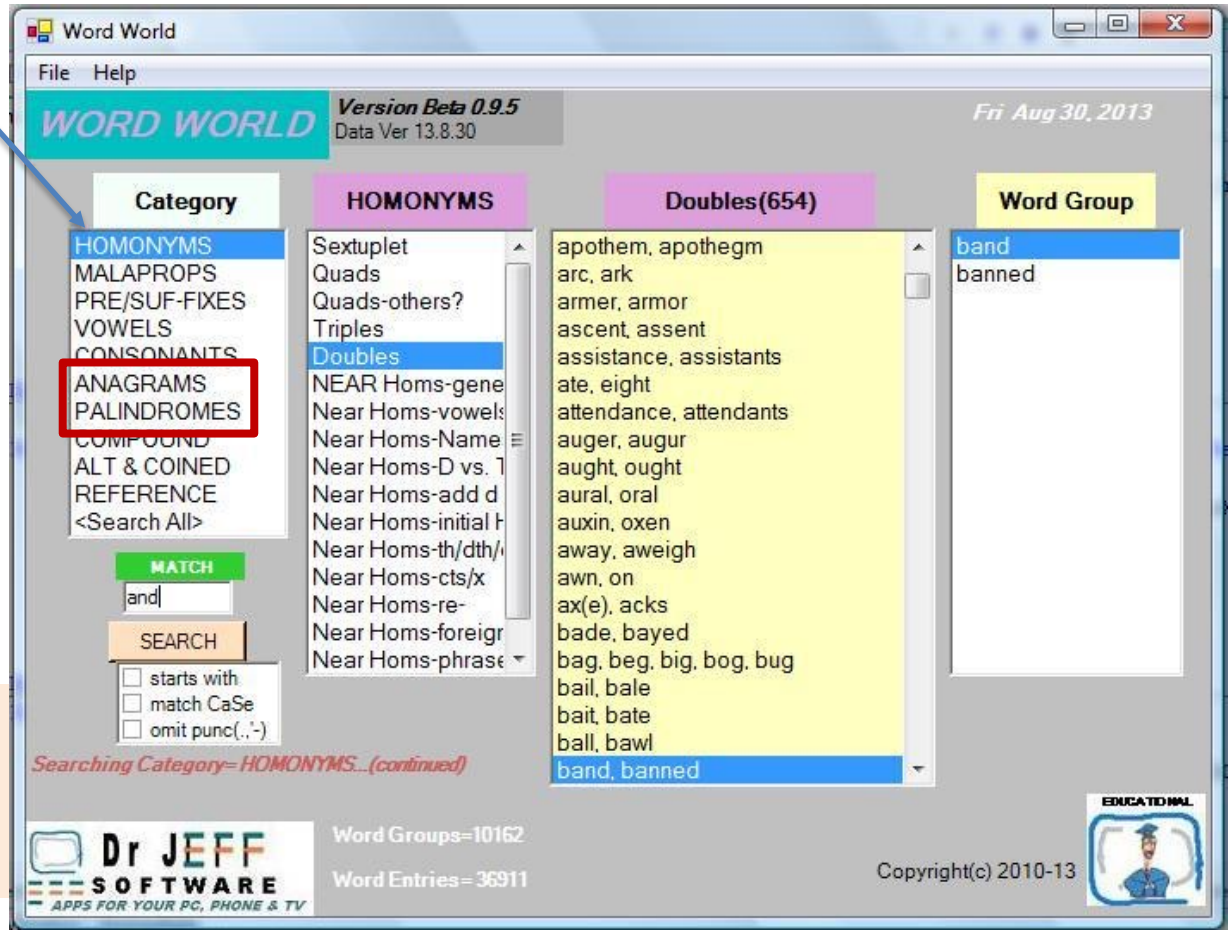
Classic 4-way HOMONYM:

- rite
- Write
- right
- Wright

Classic 2-way HOMOGRAPHS:

- read*
- record

*has HOMONYMS!



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Homonym List

Lab 5

DR JEFF SOFTWARE
INDIE APP DEVELOPER

```
DRJ File/Word World
id=Word World data file
ver=16.10.11
========
***WORD WORLD***
(C) 2007-2016 Dr Jeff Software (all rights reserved)
========
@100
HOMONYMS
========
@101
Sextuplet
---------
air, are, ere, 'ere, err, heir
---------
@102
Quads
-------
ayed, eyed, I'd, ide
lase, lays, laze, leis
maize, mase, mays, maze
mal, mall, maul, moll
not, knot, naught/nought, naut
or, oar, ore, o'er
road, rode, roed, rowed
use, yews, yous, ewes
wheal, wheel, weal, we'll
whys, why's, wise, wyes
wind, wined, whined, wynd
wreaks, wrecks, recks, rex
wright, right, write, rite
---------
@103
Quads-others?
-------
ades, aids, aides, AIDS (acronym)
ah, aw, awe, au (French)
by, buy, bye, bi- (hyphenate)
new, knew, gnu, nu (Greek)
raise, rays, raze, reys (Sp deriv.)
record, re-chord, re-cord, re-cored (hyphenate)
rose, roes, rows, rhos (Greek)
seas, sees, seize, cees (Cs)
```

```
Triples
-------
ade, aid, aide
ah, aw, awe
aisle, isle, I'll
aweful, awful, offal
axel, axil, axle
a, aye, eh
ay(e), eye, I
bailee, bailey, bailie
bailer, bailor, baler
baize, bays, beys
bald, balled, bawled
balk, bock, bok
balm, baum, bomb
bel, bell, belle
bite, bight, byte
bole, boll, bowl
bold, bolled, bowled
born, borne, bourn(e)
braise, brays, braze
borough, burrow, burro
by, buy, bye
call, caul, col
c/karat, caret, carrot
cart, carte, kart
caught, cot, khat
cedar, seeder, ceder
cel, cell, sell
cord, chord, cored
close, clothes, cloze
coal, cole, kohl
die, dye, di-
dire, dyer, dier
do, dew, due
doe, dough, do
does, doughs, doze
error, errer, airer
fane, fain, feign
fays, faze, phase
feat, feet, fete
flew, flu, flue
```

```
Doub
--------
a, uh
acre, acher
ad, add
aerie, airy or eerie/eery
aero, arrow
affect, effect
ail, ale
airship, heirship
all, awl
aloud, allowed
altar, alter
ant, aunt
ante, auntie (anti-)
apothem, apothegm
arc, ark
armer, armor
ascent, assent
assistance, assistants
ate, eight
attendance, attendants
auger, augur
aught, ought
aural, oral
auxin, oxen
away, aweigh
awn, on
ax(e), acks
bade, bayed
bail, bale
bait, bate
ball, bawl
band, banned
bank, banc
bar, barre
bard, barred
baron, barren
basal, basil
base, bass
bask, basque
```

Homographs

Lab 5

@130
HOMOGRAPHS

- Exclude duplicate pronunciations

attribute
aye
bass
bow
buffet
capo
choral
close
coax
compound
compress
concert
console
content, contest
contract
coup
desert
do
does
dove
excuse
flower
glower
house
implicate
incense
inter
intimate
invalid
lead
live(d)
minute
number
object
plead
P/polish
prayer
perfect
present
produce
project
puss, pussy
putting
read
rebel
record
recreate
refuse
reject
reprise
resent
resign
resume
revile
route
row
sake
secreted
separate
sewer
shower
slough
sow
stein
subject
tarry
tear
tier
wind/s/y
wound

Homonym Dictionary

Create Dictionary:
>= 25 homonym pairs

Lab 5

homs.txt

homonym, pronunciation pairs

phonemes

phonemes

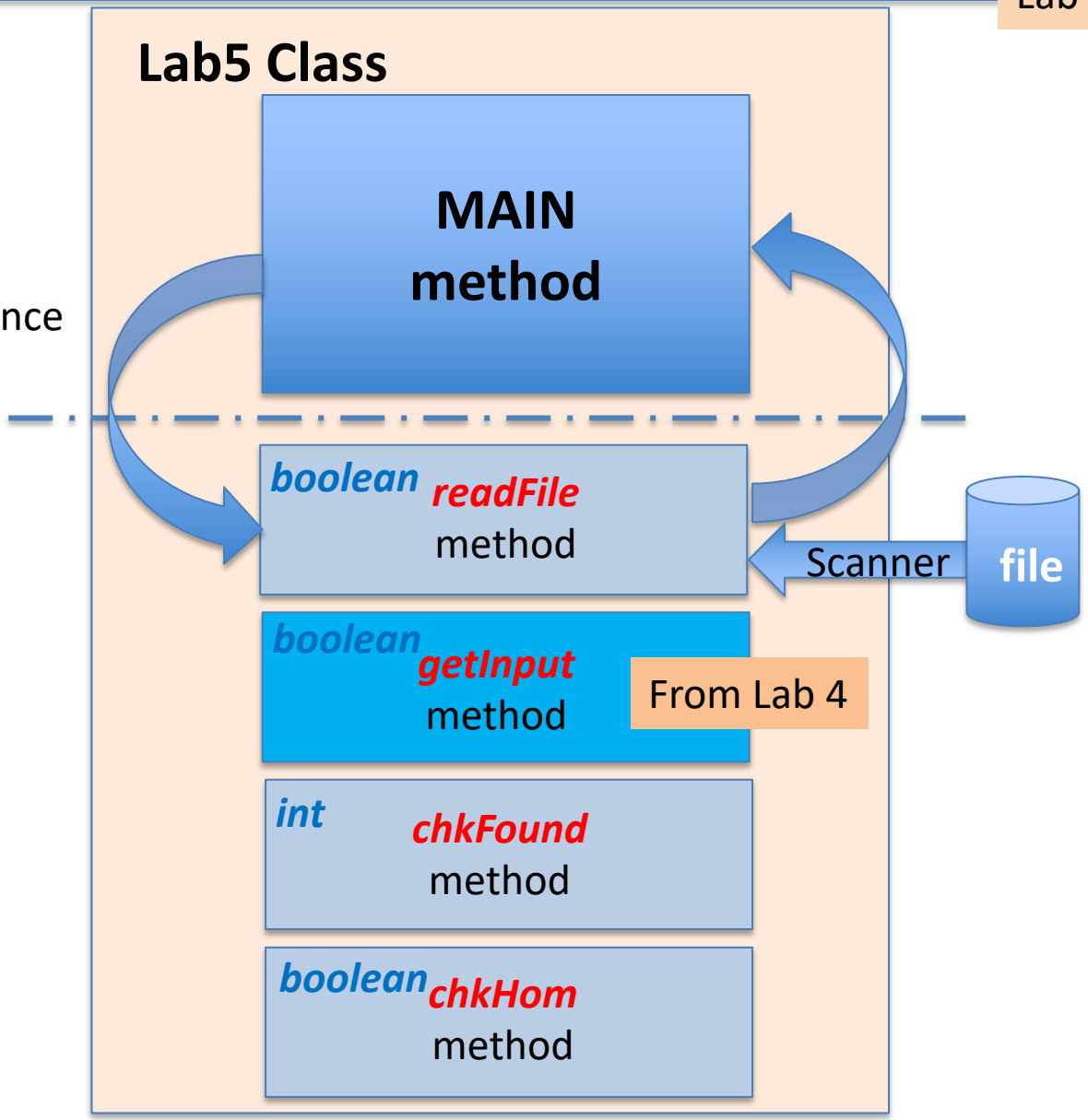
| ad |	ad, ad
	add, ad
| āl |	ail, Al
	ale, Al
| ə'wā |	away, awA
	aweigh, awA
| bāl |	bail, bAl
	bale, bAl
| bôl |	ball, bal
	bawl, bal
	⋮
rite | rīt |	right, rIt
write | rīt |	rite, rIt
right | rīt |	wright, rIt
wright | rīt |	write, rIt

Structure (Macro)

Lab 5

OOP
Structures

Methods
Execution is
by *call* sequence



Text File Input

Lab 5

Listing 12.15
pp. 478-9

```
import java.util.*;
import javax.swing.*;
import java.io.*;

public class CS110 {
public static void main(String[], args) {

    //**text file input
    //create a file instance
    java.io.File fName = new java.io.File("myFile.txt");
    //create Scanner for file
    Scanner input = new Scanner(fName);
    //read data from file
    while (input.hasNext()) {
        String word = input.next();
        String pron = input.next();
        //alt:  String line = input.nextLine();
    }
    //when done (now) close file
    input.close();
```

throws Exception

input

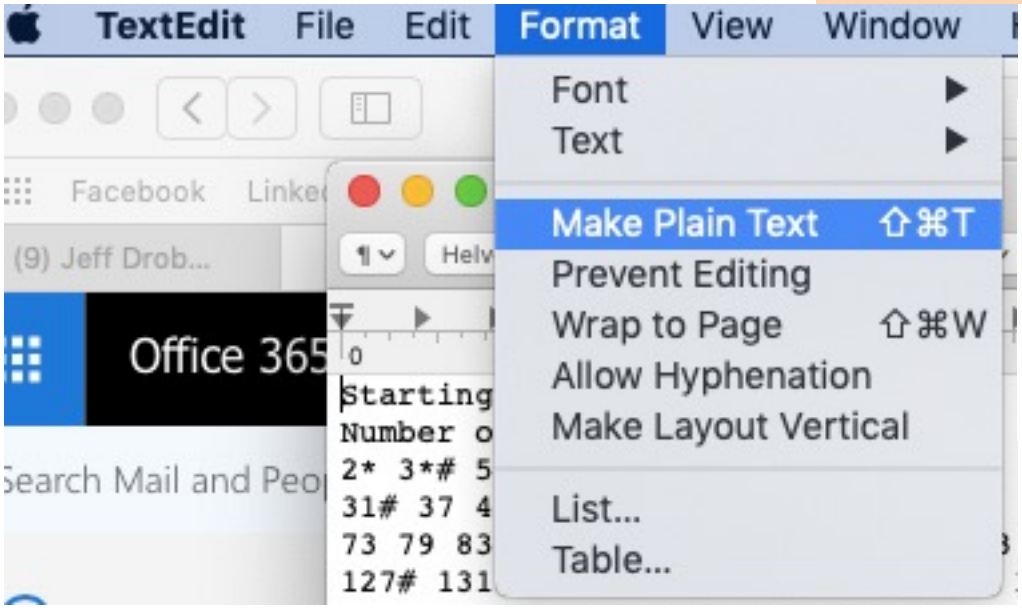
```
public class cs110TextFile {
public static void main(String[] args) throws FileNotFoundException {
    //test I/O
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

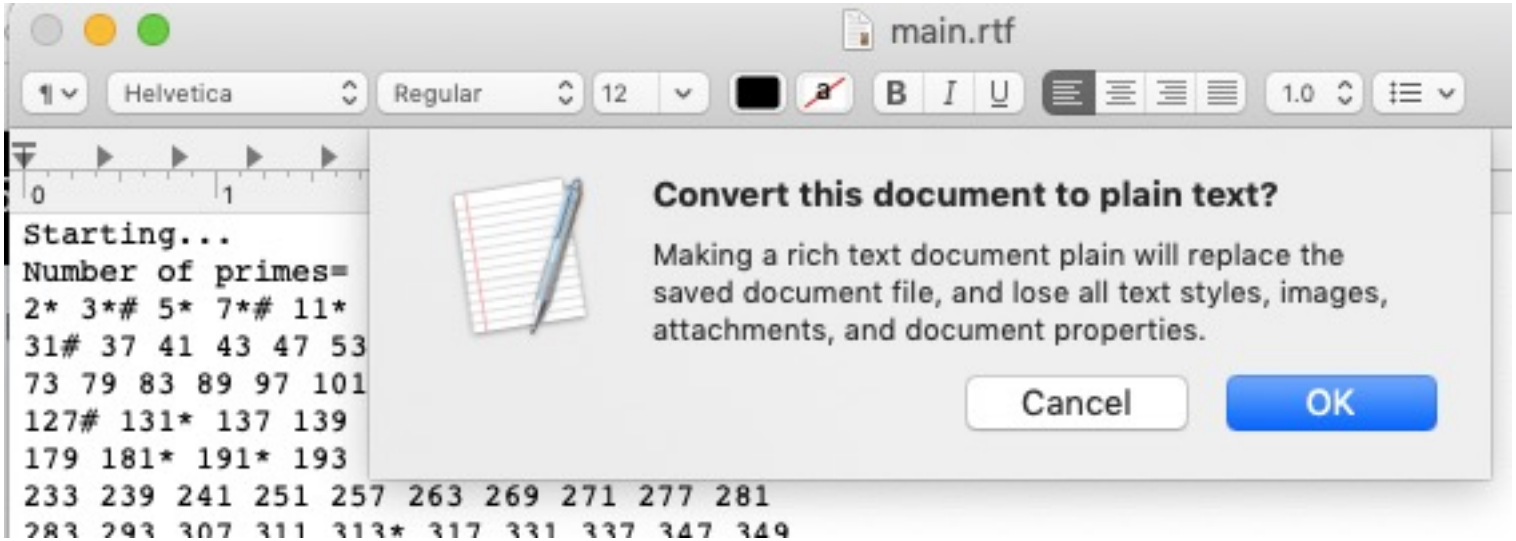
DR JEFF SOFTWARE
INDIE APP DEVELOPER

Mac Files

.rtf → .txt




```
Starting...Number of primes= 168 2* 3*# 5* 7*# 11* 13 17 19 23 29
31# 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101* 103 107 109
113 127# 131* 137 139 149 151* 157 163 167 173 179 181* 191* 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283
293 307 311 313* 317 331 337 347 349 353* 359 367 373* 379 383* 389
397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487
491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701
709 719 727* 733 739 743 751 757* 761 769 773 787* 797* 809 811
821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919*
929* 937 941 947 953 967 971 977 983 991 997 Count per 100 numbers
checked:[25, 21, 16, 16, 17, 14, 16, 14, 15, 14]
```



Main Method

Lab 5

```
7  import java.io.*;
8
9  public class Lab5Homs {
10    static final boolean $DEBUG = true; //flag
11    static final String spc = " ";
12    public static void main(String[] args) throws Exception {
13     //test I/O & debug mode
14       if ($DEBUG) System.out.println("starting main...\n");
15     //read data from text file into arrays w, p
16       String[] wArr = new String[50];
17       String[] pArr = new String[50];
18       String macPathName="Documents/Classroom+ITT+CSUN/CSUN/Java code/";
19       String pcPathName="Documents\\folder\\";
20       String fileName = "homs.txt"; //text file
21       readFile(fileName, wArr, pArr); //method loads arrays
22     //main control loop
23       while(true) {
24       //use input dialog to get 2 words from user: w1, w2
25       String w1 = "ad", w2 = "add"; //example
26       //check if found in dictionary: if both found, then continue
27       //check if homonyms
28       boolean isHom = chkHomonym(w1, w2);
29       //output result: "w1 and w2 are homonyms: " + isHnym
30       //ask user to continue Y/N ?
31          int cont = JOptionPane.showConfirmDialog(null, "Continue?");
32          if (cont > 0)
33             break; //exit loop or continue
34       }
35     //end main
36   }
```

- Declare arrays
- Call “readFile”

- “chkFound” (next slide)

Main Method – Found

Lab 5

```
19      String[] pArr = new String[50];
20      String fileName = "homs.txt"; //text file
21      readFile(fileName, wArr, pArr); //method loads arrays
22    //main control loop
23      while(true) {
24      //use input dialog to get 2 words from user
25      String w1="bale", w2 = "bail";
26      //check each word if in dictionary
27      int w1ix = chkFound(w1, wArr);
28      boolean isFound = (w1ix >= 0);
29      System.out.println(w1 + " is found: " + isFound);
30      int w2ix = chkFound(w2, wArr);
31      if (w1ix >=0 && w2ix >=0) msg = "both words " + w1 + " and " + w2 + "\n\tare
32      else {msg = "one or more words not in dictionary: ";
33         if (w1ix <0) msg += w1 +" "; //identify which words
34         if (w2ix <0) msg += w2 +" ";}
35      System.out.println(msg);
36      //GUI output msg
37      //check if homonyms5
38      boolean isHom = chkHom(w1ix, w2ix, pArr);
39      //output result
40      //ask user to continue Y/N ?
41         int cont = JOptionPane.showConfirmDialog(null, "Continue?");
42         if (cont > 0)
43            break; //exit loop or continue
44      }
45    //end main
```

- Call “chkFound”

Main Method Output


```
starting

1 ad| ad
2 add| ad
3 ail| Al
4 ale| Al
5 all| al
6 awl| al
7 ant| ant
8 aunt| ant
9 ate| At
10 eight| At
11 aural| oral
12 oral| oral
13 away| awA
14 aweigh| awA
15 bail| bAl
16 bale| bAl
17 ball| bal
18 bawl| bal
19 bol| bal
20 base| bAs
21 bass| bAs
22 bay| bA
23 bey| bA
24 bare| bAr
25 bear| bAr
30 brake| brAk
31 break| brAk
32 rite| rIt
33 write| rIt
34 right| rIt
35 wright| rIt
bale is found: false
one or more words not in dictionary: bale bail

----jGRASP: operation complete.
```

Input: Split/Check

Lab 5

```
//main control loop
  while(true) {
  //use input dialog to get 2 words from user
  String instr = "to too two";
  //check if 2 words & trim
  instr = instr.trim();
  String[] words = instr.split("\\s");
  int numWords = words.length;
  if ($DEBUG) {
     System.out.println("no. of words= " + numWords);
     for(String w: words)
     System.out.println("word= " + w);
     }
```

split into **3** words

```
file homs.txt closed
no. of words= 3
word= to
word= too
word= two

 ----jGRASP: operation complete.
```

Homonyms: readFile

Lab 5

```
public static void readFile(String xName, String[] wArr, String[] pArr)
 //create a file instance
//full absolute path = "Jeffreys-MacBook-Air:/Macintosh-HD/Users/jhdphd/
 File fName = new File(xName);
//create Scanner for file
 Scanner input = new Scanner(fName);
//read data from file
 int i = 0, siz = wArr.length;
 while (input.hasNext()) {
  String word = input.next();
  String pron = input.nextLine(); //clear cr/lf
  wArr[i] = word;
  pArr[i] = pron;
  i++;
  if ($DEBUG) System.out.println(word + pron);
  if (i >= siz) { //don't overflow array
   if ($DEBUG) System.out.println("array overflow");
   break;}
 }
//when done (now) close file
 input.close();
 return;
}
```

- "readFile"
  - Input next word
  - Store in arrays

```
----jGRASP exec: java cs110Homs
starting

ad ad
add ad
ail Al
ale Al
```

readFile with Commas

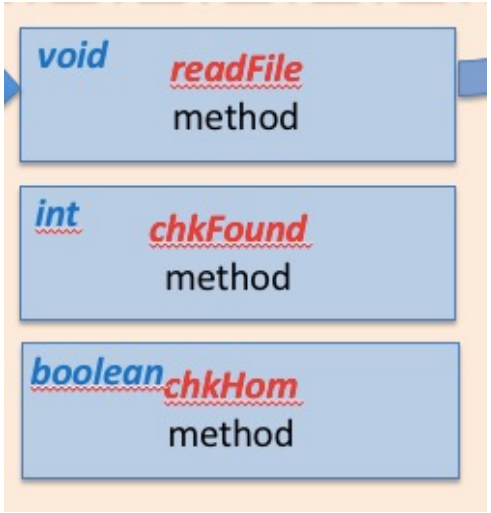
Lab 5

```
 //INput method -- text file input
public static void readFile(String xName, String[] wArr, String[] pArr)
  //create a file instance
   File fName = new File(xName);
 //create Scanner for file
   Scanner input = new Scanner(fName);
 //read data from file
   int i = 0, siz = wArr.length;
   boolean comma;
   while (input.hasNext()) {
      String word = input.next();
      String pron = input.nextLine(); //clear cr/lf
      wArr[i] = word;
      pArr[i] = pron;
      i++;
      if ($DEBUG) System.out.println(word + pron);
      int wend = word.length() - 1;
      comma = word.charAt(wend) == ',';
      if (comma) {
         word = word.substring(0, wend);
         if ($DEBUG) System.out.println(word);}
      if (i >= siz) { //don't overflow array
         if ($DEBUG) System.out.println("array overflow");
         break;}
   }
```

- delete commas

Check Methods


```
68   public static int chkFound(String w1, String[] wArr) {
69   return -1;
70   }//end chkFound
71
72    public static boolean chkHom(int x1, int x2, String[] pArr) {
73    return true;
74   }//end chkHom
```



ChkFound

Search dictionary

Lab 5

- methods to find if a “word” is contained in an array (representing a “dictionary”)

```
public static boolean contains(String[] Arr , String word){

for(String a : Arr){
    if(a.equals(word)){
        return true;
        }
}
    return false;
}
```

- *for each*

- need to add a counter (i++) for index
- So use ”**for(int i=0; i<Arr.length; i++)**”

ChkFound with ArrayList

Search dictionary

Lab 5

- or use "**.asList**"

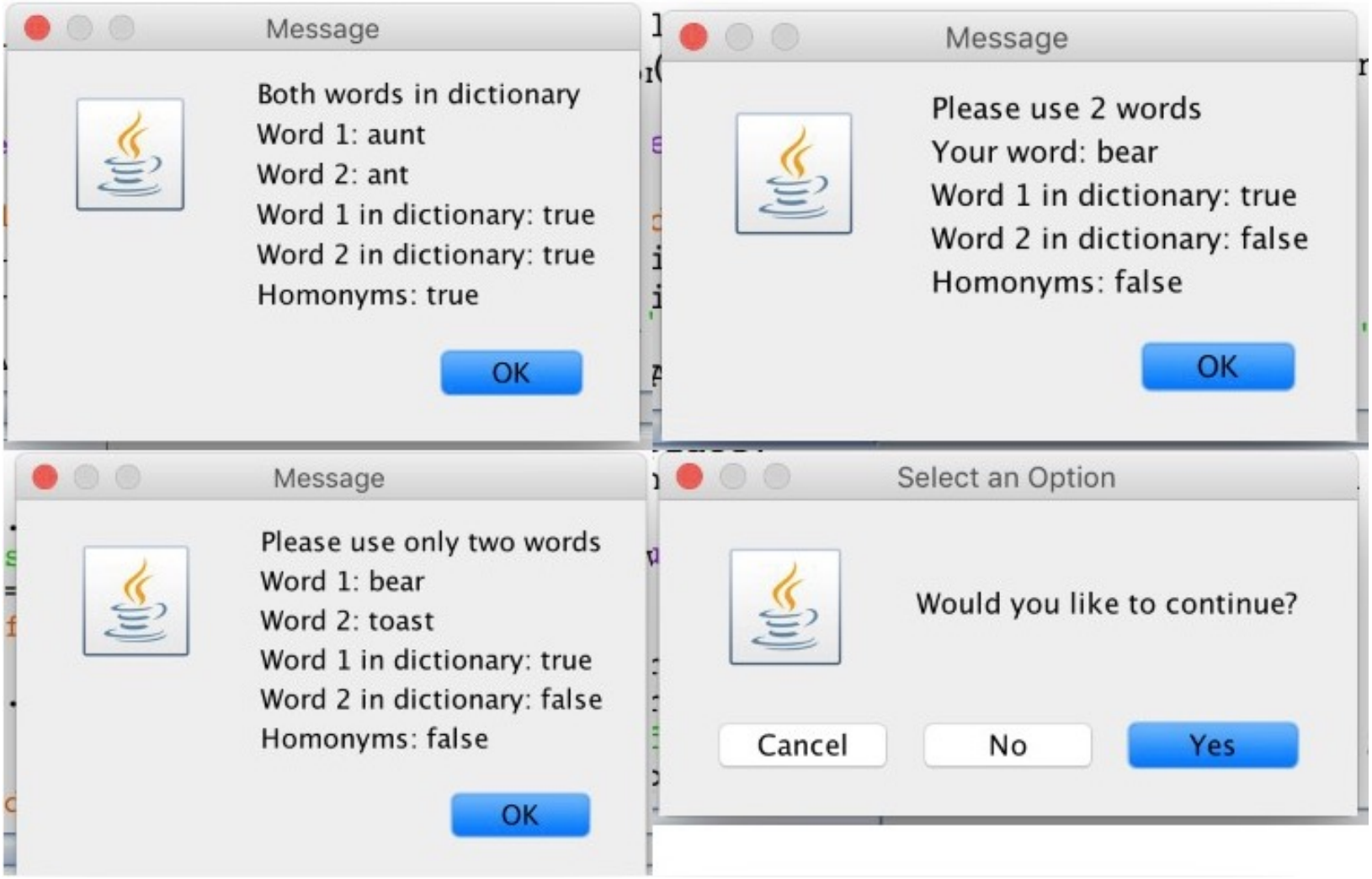
```
public static boolean contains(String[] wArr, String word){
    if (Arrays.asList(wArr).contains(word)) {
        return true;}
```

- With "**.indexOf**"

```
if ((containresult == true) && (containresult1 == true)) {
    int b = Arrays.asList(wArr).indexOf(word1);
    int c = Arrays.asList(wArr).indexOf(word2);
```

Sample Output

Lab 5



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

File Methods

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 5

```
debug: starting code
Does file exist: true
File size in byte: 306
Can file be read: true
Can file be written: true
Is name a dir: false
Is name a file: true
Is path absolute: false
Is file hidden: false
Absolute path is: /Users/jhdphd/Documents/Classr
Date last modified: Mon Oct 30 20:59:07 PDT 2017
```

```
4  file:  File.java
5  */
6  // imports
7  import javax.swing.*;
8  import java.util.*;
9  import java.io.*;
10 // **main class**
11 public class Filex {
12    static final boolean $DEBUG = true;
13 //main method
14    public static void main(String[] args) {
15       //debug
16       if ($DEBUG) System.out.println("debug: starting code");
17       //"is" methods
18       File fname = new File("homs.txt");
19       System.out.println("Does file exist: " + fname.exists());
20       System.out.println("File size in byte: " + fname.length());
21       System.out.println("Can file be read: " + fname.canRead());
22       System.out.println("Can file be written: " + fname.canWrite());
23       System.out.println("Is name a dir: " + fname.isDirectory());
24       System.out.println("Is name a file: " + fname.isFile());
25       System.out.println("Is path absolute: " + fname.isAbsolute());
26       System.out.println("Is file hidden: " + fname.isHidden());
27       //"get" methods
28       System.out.println("Absolute path is: " + fname.getAbsolutePath());
29       //date methods
30       System.out.println("Date last modified: " + new Date(fname.lastModified()));
31    } //end main method
32 } //end class
```

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Lab 6: Prime Numbers

DR JEFF SOFTWARE INDIE APP DEVELOPER

Lab 6

Rqts– INPUT: none

OUTPUT: all primes from 1 to 1000

- GUI or Console

- *Methods*
- *Strings*
- *Arrays*

- ❑ *into Array[ ]*
- ❑ *use Method*

- *grouped 10 per line*
- *add count per 100* → *[26], [21], …*
- *Flag/tag Pals* + Mersennes#*

INPUT:
none

PROCESS (source code)

1) set N=1000
2) **Test** IF prime
   - *using Textbook algorithm*
   - *Dr Jeff algorithm*
3) **Compare** results
4) **Output** primes

OUTPUT:

- Primes grouped in 10s
- with count per 100
- total count
- flags/tags (*#)
- *store into data file*

- ***Submit file***

Listing 5.15
pp. 189-190

DEBUGGING/TESTIN

```
The first 50 prime numbers are

2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Lab 6

DR JEFF SOFTWARE
INDIE APP DEVELOPER

6.23 Lab 6- Prime Numbers Lab

6.23 Lab 6- Prime Numbers

Complete the given Template code to do this:

1. print all prime numbers from 2 to 50 (all on one line)
2. flag all palindromes (*)
3. print the counts of:
    - primes
    - palindromes

```
10    public static void main(String[] args) {
11     //inits
12        int N = 50; //max number
13        int[] primes = new int[N/2];
14        int count = 0; //primes counter
15        int countPals = 0;
```

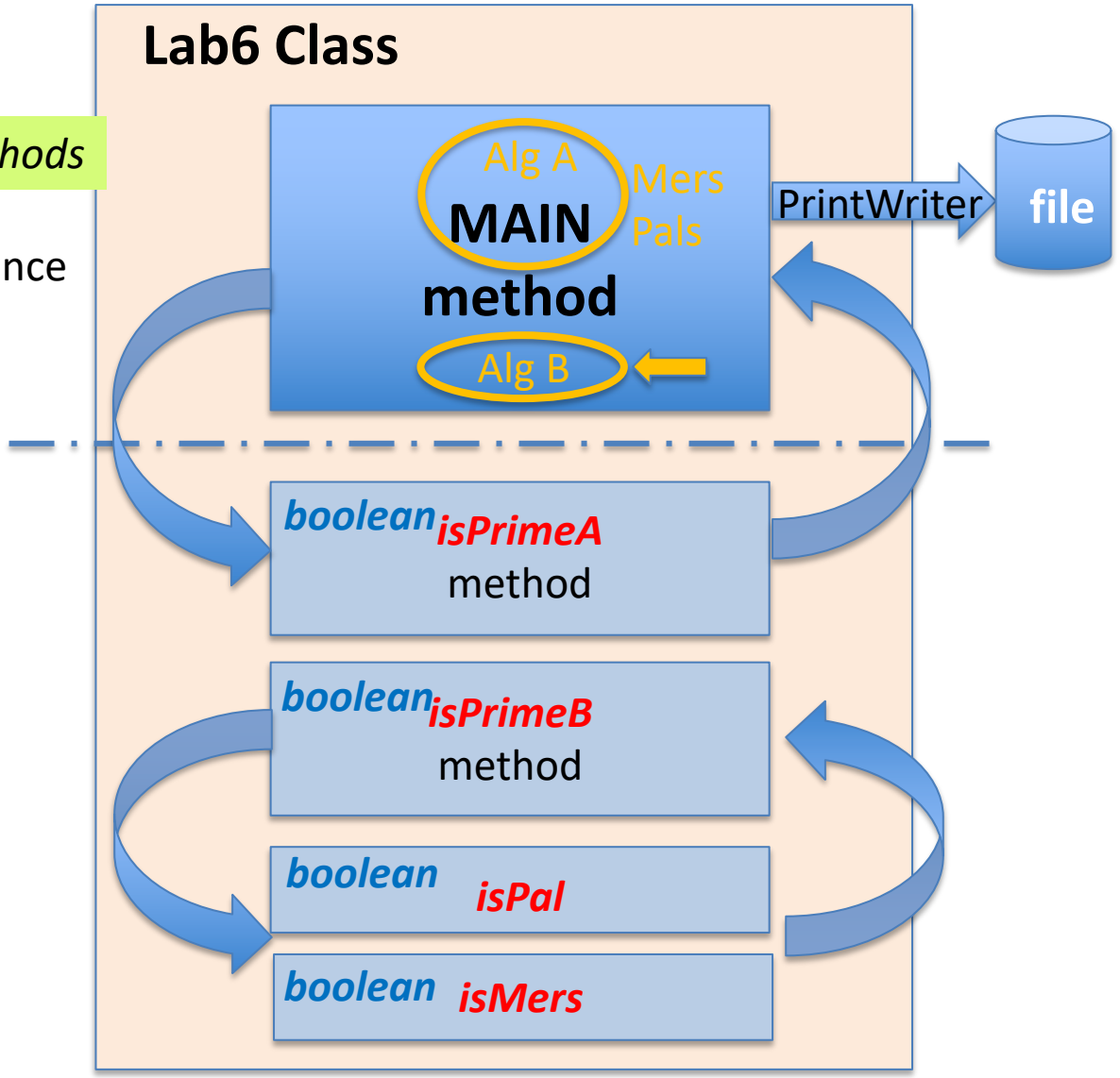
Structure (Macro)

Lab 6

OOP
Structures

Classes/methods

Execution is
by *call* sequence



Main: Inits


```
 9 public class Lab6Primes {
10    static final boolean $DEBUG = true; //flag
11    static final String spc = "  ";
12    public static void main(String[] args) throws FileNotFoundException {
13     //inits
14       int[] primesA = new int[1000];
15       int[] primesB = new int[500];
16       primesB[0] = 2; //init array with 1st 3 primes
17       primesB[1] = 3;
18       primesB[2] = 5;
19       int N = 1000; //max number
20       int countA = 0, countB = 3; //primes counters
21       int countMers = 0, countPals = 0;
22       int[] count100s = new int[10];
23       Arrays.fill(count100s,0);//init to 0
24       File fnam = new File("primes.txt");
25       PrintWriter outFile = new PrintWriter(fnam);
26     //start code
```

Main Loop

Lab 6

```
26      //start code
27        if ($DEBUG) System.out.println("starting...");
28        //textbook alg
29        for (int i=2; i<=N; i++) {//all ints 2..N
30           if (isPrimeA(i)) {//add new prime
31              primesA[countA] = i;
32              countA++;
33           //print i to console & file
34              System.out.print(i);
35              if(isPal(i)) {
36                 System.out.print("*");
37                 countPals++;}
38              if(isMers(i)) {
39                 System.out.print("#");
40                 countMers++;}
41              System.out.print(spc);
42              outFile.printf("%4d", i);//file
43              if(countA %10 == 0) {//10 per line
44                 System.out.println();
45                 outFile.println();}
46           }//end found prime
47        }//end loop: textbook alg
48
49        outFile.close();//close file
50        //print counts
51        System.out.println("\nNumber of primes found= " +countA);
52        System.out.println("Number of Mersennes= " +countMers);
53        System.out.println("Number of palindromes= " +countPals);
54        System.out.println("Counts of 100s:");
55        //print count100s array here
56
```

Add prime

Flag pal

Flag Mers

\n every 10

Print all counts

Algorithms– Prime Numbers

Old **FORTRAN**

nested
"DO loops"

FIGURE 9-5 LIST OF PRIME NUMBERS

```
      DO 5 I = 5,10000,2
         K = SQRT(REAL(I))
         DO 4 J = 3,K,2
            IF(MOD(I,J).EQ.0)GO TO 5
    4    CONTINUE
         WRITE(6,31)I
    5 CONTINUE
      STOP
   31 FORMAT(1X,I6)
      END
```

Skip all even numbers.
Determine $\sqrt{I}$. Argument of SQRT must be real.

Check if I is divisible by any integer up to and possibly including $\sqrt{I}$.
Number is prime; print it.

Test routine → *subroutine*

Close to "Dr Jeff" algorithm:
→ Test only **ODD** numbers
→ Limit = **sqrt**(num)

Basic Algorithm

Lab 6

Textbook algorithm

```
for(int divisor = 2; divisor <= number/ 2; divisor++){
   if(number % divisor == 0){
      isPrime = false;
      break;
```

Parameters

- Numbers = 1 .. 50th prime
  - ***replace with N = 1000***
- divisors = **all** integers = 2 .. limit
- limit = Number**/2**

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Textbook Prime Numbers

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 6

Listing 5.15
pp. 189-190

```
 8     System.out.println("The first 50 prime numbers are \n");
 9
10     // Repeatedly find prime numbers
11     while (count < NUMBER_OF_PRIMES) {
12       // Assume the number is prime
13       boolean isPrime = true; // Is the current number prime?
14
15       // Test if number is prime
16       for (int divisor = 2; divisor <= number / 2; divisor++) {
17         if (number % divisor == 0) { // If true, number is not pri
18           isPrime = false; // Set isPrime to false
19           break; // Exit the for loop
20         }
21       }
22
23       // Print the prime number and increase the count
24       if (isPrime) {
25         count++; // Increase the count
26
27         if (count % NUMBER_OF_PRIMES_PER_LINE == 0) {
28           // Print the number and advance to the new line
29           System.out.println(number);
30         }
31         else
32           System.out.print(number + " ");
33       }
34
35       // Check if the next number is prime
36       number++;
37     }
38   }
39 }
```

- flag
- test

Parameters

Change this to 1..1000

```
The first 50 prime numbers are

2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
```

Algorithms: Textbook+Dr Jeff

Lab 6

```
69    //textbook alg
70    static boolean isPrimeA(int num) {
71    //parms
72       int divStart = 2;
73       int divStep = 1;
74       int divLimit = num/2;
75       boolean prFlag = true;
76    //check algorithm-loop
77       for(int i=divStart; i<=divLimit; i+=divStep) {
78          if(num %i == 0) {
79             prFlag = false;
80             break;}
81       }//end loop
82       return prFlag;
83    }//end isPrimeA
84
85    //Dr Jeff alg
86    static boolean isPrimeB(int num) {
87    //parms->correct
88       int divStart = 2;
89       int divStep = 1;
90       int divLimit = num/2;
91       boolean prFlag = true;
92    //check algorithm-loop
93    /*enter code here*/
94    return prFlag;
95    }//end isPrimeB
```

Parameters

- Textbook alg

- Dr Jeff alg

Parameters ← Correct these values

Sample Output

Lab 6

```
Prime numbers between 1 and 1000 are

2  3*  5  7*  11  13  17  19  23  29
31*  37  41  43  47  53  59  61  67  71
73  79  83  89  97  101  103  107  109  113
127*  131  137  139  149  151  157  163  167  173
179  181  191  193  197  199  211  223  227  229
233  239  241  251  257  263  269  271  277  281
283  293  307  311  313  317  331  337  347  349
353  359  367  373  379  383  389  397  401  409
419  421  431  433  439  443  449  457  461  463
467  479  487  491  499  503  509  521  523  541
547  557  563  569  571  577  587  593  599  601
607  613  617  619  631  641  643  647  653  659
661  673  677  683  691  701  709  719  727  733
739  743  751  757  761  769  773  787  797  809
811  821  823  827  829  839  853  857  859  863
877  881  883  887  907  911  919  929  937  941
947  953  967  971  977  983  991  997
```

**flagged*

- *Store primes in File*

- *Required extra outputs*

```
The number of primes between 1 and 1000 are 168
The number of Mersennes is 4
The counts of Primes per 100 numbers are:
[25], [21], [16], [16], [17], [14], [16], [14], [15], [14]
```

Actual Output

*pals(?)

#Mers(4)

```
----jGRASP exec: java Lab6Primes
starting...
2* 3# 5 7# 11* 13* 17 19* 23* 29
31*# 37 41* 43 47 53 59 61* 67* 71*
73* 79* 83* 89* 97 101* 103 107* 109 113*
127*# 131* 137* 139* 149 151 157* 163* 167* 173
179* 181* 191 193 197* 199* 211 223* 227* 229
233 239* 241* 251 257* 263* 269* 271* 277 281*
283* 293* 307 311* 313 317* 331* 337 347* 349*
353* 359 367* 373 379* 383 389 397 401 409*
419* 421 431* 433* 439* 443 449 457* 461 463*
467 479* 487* 491 499* 503 509* 521* 523* 541*
547* 557* 563 569* 571* 577* 587* 593 599 601
607 613 617* 619* 631* 641 643* 647* 653* 659*
661* 673 677* 683* 691* 701 709* 719 727* 733*
739* 743 751* 757* 761* 769 773 787* 797* 809*
811 821* 823* 827* 829* 839 853* 857 859* 863
877* 881* 883* 887 907 911 919 929 937 941*
947* 953* 967 971 977* 983 991* 997*
Number of primes found= 168
Number of Mersennes= 4
Number of palindromes= 103
Counts of 100s:
Algorithms are equivalent: false

----jGRASP: operation complete.
```

Sample Output with Flags

main.txt

```
Starting...
Number of primes= 168
2* 3*# 5* 7*# 11* 13 17 19 23 29
31# 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101* 103 107 109 113
127# 131* 137 139 149 151* 157 163 167 173
179 181* 191* 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313* 317 331 337 347 349
353* 359 367 373* 379 383* 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727* 733
739 743 751 757* 761 769 773 787* 797* 809
811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919* 929* 937 941
947 953 967 971 977 983 991 997

Count per 100 numbers checked:
[25, 21, 16, 16, 17, 14, 16, 14, 15, 14]
```

- Text file does NOT need:
  - Flags
  - Counts

Formatting 10 Per Line

Lab 6

primes[0]

primes[10]

```
 2  3*  5  7*  11  13  17  19  23  29  31
 31*  37  41  43  47  53  59  61  67  71
 73  79  83  89  97  101  103  107  109  113
 127*  131  137  139  149  151  157  163  167  173
 179  181  191  193  197  199  211  223  227  229
```

key: start counter at **1** (not 0)
then can use **% 10**

```
if(countA %10 == 0) {//10 per line
    System.out.println();
```

Methods: Pals & Mers

Lab 6

- Mers alg
- Pals alg

```
97    //Mers
98    static boolean isMers(int num) {
99    boolean merFlag = false;
100   int[] mers = {3, 7, 31, 127};
101   for(int i=0; i<4; i++) {
102      if(num == mers[i]) {
103         merFlag = true;
104         break;}
105   }//end loop
106   return merFlag;
107   }//end isMers
108
109   //Pal: ->use your Lab 3 method
110   static boolean isPal(int num) {
111   boolean palFlag;
112   //random algorithm -> replace
113   if((int)(Math.random()*2) ==0)
114      palFlag = true;
115      else palFlag = false;
116   return palFlag;
117   }//end isPal
```

Text File *Output*

Lab 6

Listing 12.16 pp. 480-1

```
import java.io.*;

public class cs110TextFile {
public static void main(String[] args) throws FileNotFoundException {
    //test I/O
```

```
  //create OUTput (method later)
 File fName2 = new File("Documents/OutFile.txt");
 PrintWriter output = new PrintWriter(fName2);
 //create array for output data
 String[] outArr = new String[100];
 //write data TO file
 for (int i=0; i< outArr.length; i++){
 output.println(outArr[i]);
 }
```

- *PrintWriter*

```
java.io.File file = new java.io.File("scoresPrime.txt");
if(file.exists()){
   System.out.println("File already exists");
   System.exit(1);
}
```

check file optional

output.close( );

Text File Output (Formatted)

Lab 6

```
File fnam = new File("primes.txt");
PrintWriter outFile = new PrintWriter(fnam);
```

primes.txt

```
  2   3   5   7  11  13  17  19  23  29
 31  37  41  43  47  53  59  61  67  71
 73  79  83  89  97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733
739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997
```

Printf(“%4d”, num)

```
outFile.printf("%4d", i);//file
```

Computing 100s Counts

Lab 6

```
int Arr1[] = new int [10];
for (int i = 0; i<count;i++) {
    Arr1[Arr[i]/100]++;
}//end for
```

- simple formula

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Mersenne Primes

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 6

Mersenne prime

$2^n - 1$

From Wikipedia, the free encyclopedia

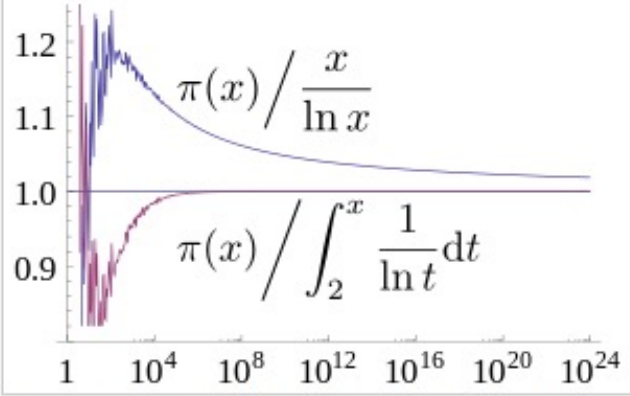
In mathematics, a **Mersenne prime** is a prime number that is one less than a power of two. That is, it is a prime number that can be written in the form $M_n = 2^n - 1$ for some integer n. They are named after Marin Mersenne, a French Minim friar, who studied them in the early 17th century. The first four Mersenne primes (sequence A000668 in the OEIS) are 3, 7, 31, and 127.

Mersenne prime

Named after	Marin Mersenne
Publication year	1536[1]
Author of publication	Regius, H.
Number of known terms	49
Conjectured number of terms	Infinite
Subsequence of	Mersenne numbers
First terms	3, 7, 31, 127
Largest known term	$2^{74,207,281} - 1$ (January 2016)

4	3
8	7
16	15
32	31
64	63
128	127
256	255
512	511
1024	1023

511 = 7x73



Graph showing ratio of the prime-counting function $\pi(x)$ to two of its approximations, x/log x and Li(x). As x increases (note x axis is logarithmic), both ratios tend towards 1. The ratio for x/log x converges from above very slowly, while the ratio for Li(x) converges more quickly from below.

Prime number theorem

From Wikipedia, the free encyclopedia

In number theory, the **prime number theorem** (**PNT**) describes the asymptotic distribution of the prime numbers among the positive integers. It formalizes the intuitive idea that primes become less common as they become larger by precisely quantifying the rate at which this occurs. The theorem was proved independently by Jacques Hadamard and Charles Jean de la Vallée-Poussin in 1896 using ideas introduced by Bernhard Riemann (in particular, the Riemann zeta function).

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Mersenne Primes

COMP110L

$2^n - 1$

Lab 6

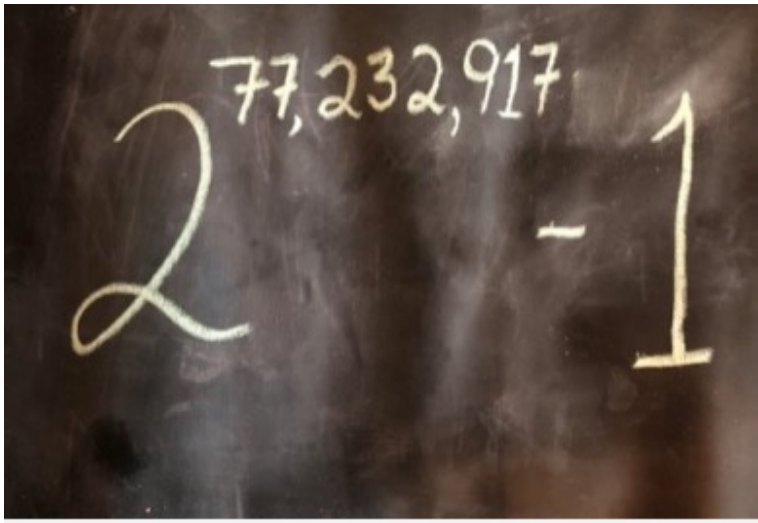
New Prime Number Discovered

$2^{77,232,917} - 1$

The new prime number is nearly one million digits larger than the previous record prime number. The primality proof took six days of non-stop computing on a PC with an Intel i5-6600 CPU.

Read more ...
Source: Science Daily (2018-01-05)

The Great Internet Mersenne Prime Search (GIMPS) has discovered the largest known prime number, $2^{77,232,917}-1$, having 23,249,425 digits. A computer volunteered by Jonathan Pace made the find on December 26, 2017. Jonathan is one of thousands of volunteers using free GIMPS software.



The new prime number, also known as M77232917, is calculated by multiplying together 77,232,917 twos, and then subtracting one.

Credit: Copyright Dan Hogan

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Mersenne Primes

$2^n - 1$

Lab 6

The Great Internet Mersenne Prime Search (GIMPS) was formed in January 1996 by George Woltman to discover new world record size Mersenne primes. In 1997 Scott Kurowski enabled GIMPS to automatically harness the power of thousands of ordinary computers to search for these "needles in a haystack." Most GIMPS members join the search for the thrill of possibly discovering a record-setting, rare, and historic new Mersenne prime. The search for more Mersenne primes is already under way. There may be smaller, as yet undiscovered Mersenne primes, and there almost certainly are larger Mersenne primes waiting to be found. Anyone with a reasonably powerful PC can join GIMPS and become a big prime hunter, and possibly earn a cash research discovery award. All the necessary software can be downloaded for free at www.mersenne.org/download/.

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Mersenne Primes

$2^n - 1$

Lab 6

DR JEFF SOFTWARE
INDIE APP DEVELOPER

The primality proof took six days of non-stop computing on a PC with an Intel i5-6600 CPU. To prove there were no errors in the prime discovery process, the new prime was independently verified using four different programs on four different hardware configurations.

- Aaron Blosser verified it using Prime95 on an Intel Xeon server in 37 hours.
- David Stanfill verified it using gpuOwL on an AMD RX Vega 64 GPU in 34 hours.
- Andreas Höglund verified the prime using CUDALucas running on NVidia Titan Black GPU in 73 hours.
- Ernst Mayer also verified it using his own program Mlucas on 32-core Xeon server in 82 hours. Andreas Höglund also confirmed using Mlucas running on an Amazon AWS instance in 65 hours.

GIMPS Prime95 client software was developed by founder George Woltman. Scott Kurowski wrote the PrimeNet system software that coordinates GIMPS' computers. Aaron Blosser is now the system administrator, upgrading and maintaining PrimeNet as needed. Volunteers have a chance to earn research discovery awards of $3,000 or $50,000 if their computer discovers a new Mersenne prime. GIMPS' next major goal is to win the $150,000 award administered by the Electronic Frontier Foundation offered for finding a 100 million digit prime number.

he has been hunting for big primes with GIMPS for over 14 years. The discovery is eligible for a $3,000 GIMPS research discovery award.

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Top Mersenne Primes

>rank	prime	digits	who	when	comment
1	$2^{82589933}-1$	24862048	G16	Dec 2018	Mersenne 51?? (**)
2	$2^{77232917}-1$	23249425	G15	Jan 2018	Mersenne 50?? (**)
3	$2^{74207281}-1$	22338618	G14	Jan 2016	Mersenne 49?? (**)
4	$2^{57885161}-1$	17425170	G13	Feb 2013	Mersenne 48? (**)
5	$2^{43112609}-1$	12978189	G10	Aug 2008	Mersenne 47 (**)
6	$2^{42643801}-1$	12837064	G12	Jun 2009	Mersenne 46 (**)
7	$2^{37156667}-1$	11185272	G11	Sep 2008	Mersenne 45 (**)
8	$2^{32582657}-1$	9808358	G9	Sep 2006	Mersenne 44 (**)
9	$10223\cdot 2^{31172165}+1$	9383761	SB12	Nov 2016	(**)
10	$2^{30402457}-1$	9152052	G9	Dec 2005	Mersenne 43 (**)
11	$2^{25964951}-1$	7816230	G8	Feb 2005	Mersenne 42 (**)
12	$2^{24036583}-1$	7235733	G7	May 2004	Mersenne 41 (**)
13	$2^{20996011}-1$	6320430	G6	Nov 2003	Mersenne 40 (**)
14	$1059094^{1048576}+1$	6317602	L4720	Nov 2018	Generalized Fermat (**)
15	$919444^{1048576}+1$	6253210	L4286	Sep 2017	Generalized Fermat (**)
16	$168451\cdot 2^{19375200}+1$	5832522	L4676	Sep 2017	(**)
17	$7\cdot 2^{18233956}+1$	5488969	L4965	Oct 2020	Divides Fermat F(18233954) (**)
18	$\mathrm{Phi}(3,-123447^{524288})$	5338805	L4561	Feb 2017	Generalized unique (**)
19	$7\cdot 6^{6772401}+1$	5269954	L4965	Sep 2019	
20	$8508301\cdot 2^{17016603}-1$	5122515	L4784	Mar 2018	Woodall (**)

Computing 2^N

$2^N = 2^{N/2} * 2^{N/2} = (2^{N/2})^2$ → $2^N = (2^{N/k})^k$

```
int x = 1;
for (i=1; I <= N; i++) {
   x *= 2;
}
…println(x);
```

```
int x = 1;
for (i=1; I <= N/2; i++) {
   x *= 2;
}
…println(x*x);
```

2^N = 10000..0

2^N -1 = 11111..1

binary

Sample Output 2

Lab 6

```
----jGRASP exec: java Primes
2* 3* 5* 7* 11* 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101* 103 107 109 113
127 131* 137 139 149 151* 157 163 167 173
179 181* 191* 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313* 317 331 337 347 349
353* 359 367 373* 379 383* 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727* 733
739 743 751 757* 761 769 773 787* 797* 809
811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919* 929* 937 941
947 953 967 971 977 983 991 997

Are 6A and 6B primes equal? true
Both 6A and 6B have 168 entries.
There are 17 palindromes.
```

new conjecture proof

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Sieve of Eratosthenes

DR JEFF SOFTWARE INDIE APP DEVELOPER

Lab 6

Not assigned

The Sieve of Eratosthenes is a simple algorithm for finding all prime numbers up to a specified integer. It was created in the 3rd century BC by Eratosthenes, an ancient Greek mathematician.

Sieve of Eratosthenes

- Pick a value n.
- Write out a table of the integers from 2 to n.
- Cross out all entries that are multiples of 2.
- Find the smallest remaining number > 2, which is 3.
- Cross out all entries that are multiples of 3.
- Continue until you reach the floor of the square root of n.
- The numbers that remain are prime.

- test only odds
- use my test for 3

{P} = all INTEGERS <= sqrt(n)

- better to test only ODD numbers
- can easily cast out 5s (...5, ...0)
- so only need to start at 7
- but really only need to check PRIMES

{P} = all **found primes** <= sqrt(NUM)

- test sequence
  - 2, 3, 5, 7, 11, 13, 17, …
  - ***all primes***

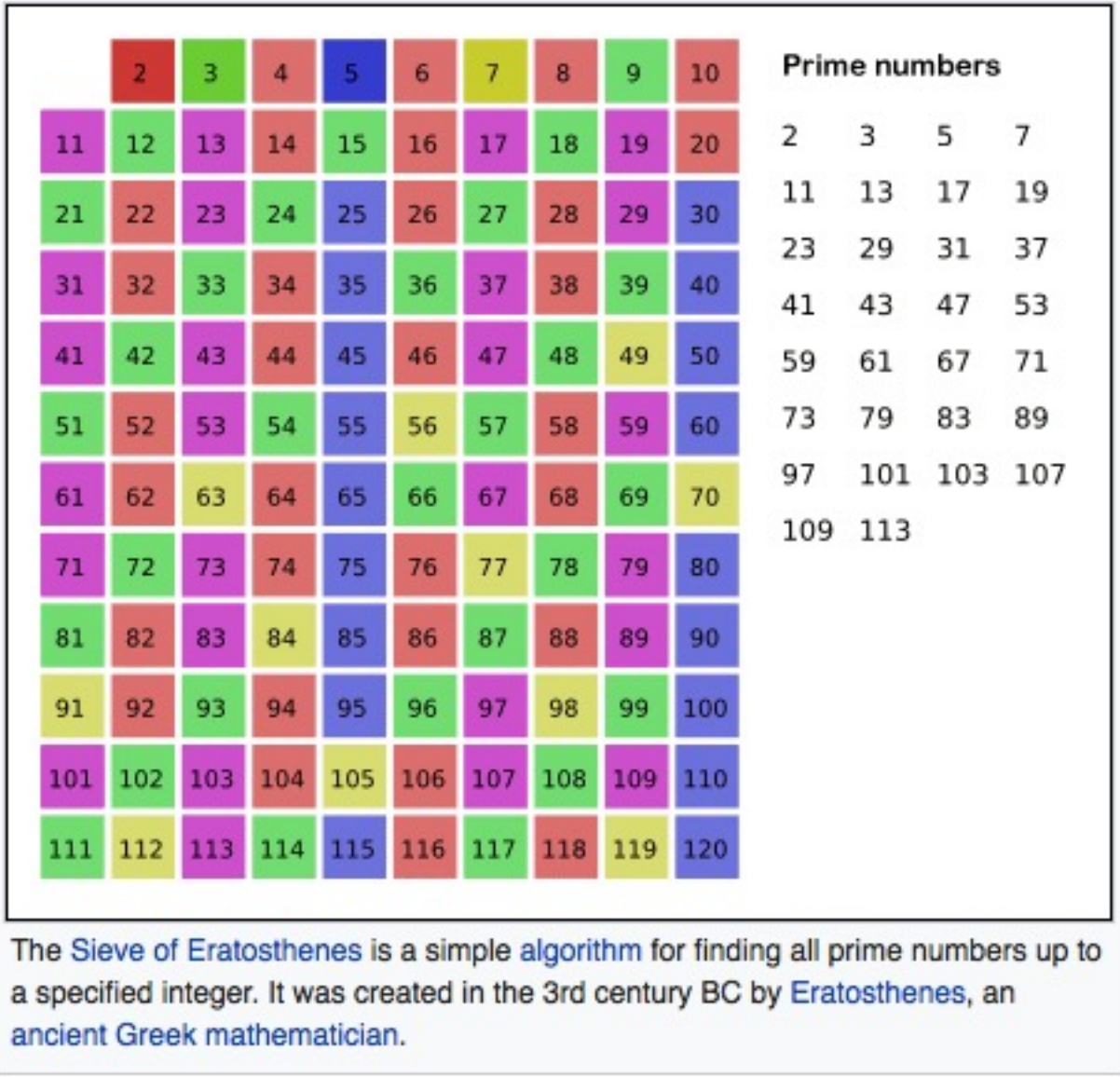
- my test for 3:
  - sum all digits
  - test sum for %3=0

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Sieve of Eratosthenes

Lab 6

DR JEFF SOFTWARE
INDIE APP DEVELOPER



The Sieve of Eratosthenes is a simple algorithm for finding all prime numbers up to a specified integer. It was created in the 3rd century BC by Eratosthenes, an ancient Greek mathematician.

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Dr Jeff Algorithm

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 6

Dr Jeff ***Optimized*** algorithm

- Numbers = 1 .. 1000 → ***odd*** only
- divisors = {P} = all **found primes** <= limit
- limit = **sqrt** (Number)

conjecture

to be proven

- can easily cast out 5s (…5, ...0)
  - ➢ so only need to start at 7

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

DR JEFF SOFTWARE INDIE APP DEVELOPER

Prime Number Algorithms

Lab 6

Find all prime numbers from 1 to N

Output: 2, 3, 5

Init: NUM = 7

A

CALL TEST

TEST P PRIME? Y → Output: NUM

N

Incr: NUM += 2

odd # only

NUM>= N? DONE? N → GOTO A

Y

STOP

Dr Jeff algorithm

TEST

Clear P flag

5? TEST div by 5 Y → RETURN

easy

N

skip 3s

3? TEST div by 3 Y → RETURN

N

{P}=all found primes <= sqrt(NUM)

TEST div {P} Y → RETURN

N

SET P flag

RETURN

Preset Primes


```
int count = 0; //co
int number = 7; //
```

- start at 7

```
// first 3 prime numbers
primes[count++] = 2;
primes[count++] = 3;
primes[count++] = 5;
```

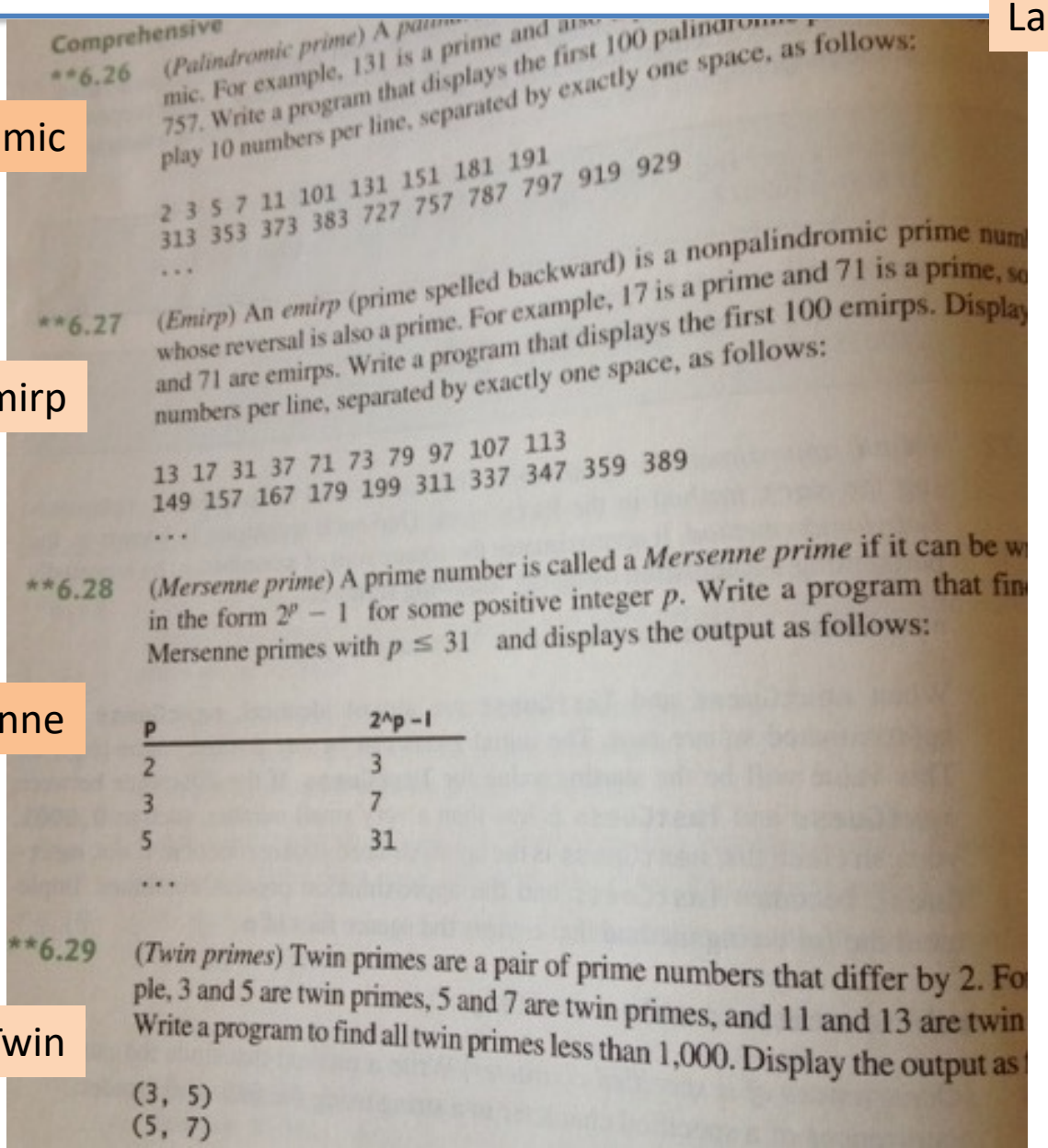
- prepopulate

```
primesB[0] = 2;
primesB[1] = 3;
primesB[2] = 5;
```

Special Primes

Lab 6

- Palindromic
- Emirp
- Mersenne
- Twin



Comprehensive

**6.26 (*Palindromic prime*) A palindromic. For example, 131 is a prime and also 757. Write a program that displays the first 100 palindromic primes. Display 10 numbers per line, separated by exactly one space, as follows:

```
2 3 5 7 11 101 131 151 181 191
313 353 373 383 727 757 787 797 919 929
...
```

**6.27 (*Emirp*) An *emirp* (prime spelled backward) is a nonpalindromic prime number whose reversal is also a prime. For example, 17 is a prime and 71 is a prime, so 17 and 71 are emirps. Write a program that displays the first 100 emirps. Display 10 numbers per line, separated by exactly one space, as follows:

```
13 17 31 37 71 73 79 97 107 113
149 157 167 179 199 311 337 347 359 389
...
```

**6.28 (*Mersenne prime*) A prime number is called a *Mersenne prime* if it can be written in the form $2^p - 1$ for some positive integer p. Write a program that finds all Mersenne primes with $p \leq 31$ and displays the output as follows:

p	2^p - 1
2	3
3	7
5	31
...	

**6.29 (*Twin primes*) Twin primes are a pair of prime numbers that differ by 2. For example, 3 and 5 are twin primes, 5 and 7 are twin primes, and 11 and 13 are twin primes. Write a program to find all twin primes less than 1,000. Display the output as follows:

```
(3, 5)
(5, 7)
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Other Special Primes

Lab 6

Type	Prime	Number of decimal digits	Date	Found by
Mersenne prime	$2^{74,207,281} - 1$	22,338,618	January 7, 2016[24]	Curtis Cooper, Great Internet Mersenne Prime Search
not a Mersenne prime (Proth number)	$10,223 \times 2^{31,172,165} + 1$	9,383,761	October 31, 2016[25]	Péter Szabolcs, PrimeGrid[26]
factorial prime	208,003! – 1	1,015,843	July 2016	Sou Fukui[27]
primorial prime	1,098,133# - 1	476,311	March 2012	James P. Burt, PrimeGrid[28]
twin primes	$2,996,863,034,895 \times 2^{1,290,000} \pm 1$	388,342	September 2016	Tom Greer, PrimeGrid[29]

factorials

3! = 6 → -1 = 5
4! = 24 → -1 = 23
5! = 120 → -1 = 119
6! = 720 → -1 = 719
7! = 5040 → -1 = 5039
8! = 40320 → -1 = 40319
9! = 362880 → -1 = 362879
10! = 3628800 → -1 = 3628799

Mersenne twins

3 & 5
7 & 9
31 & 33
127 & 129
1023 & 1025
2047 & 2049

others?

$2^N + 1$
$10^N - 1$ X
$10^N + 1$
$10^N - 3$

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Singularity of 1

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 6

Notation [edit]

Mathematicians use **N** or $\mathbb{N}$ (an N in blackboard bold) to refer to the set of all natural numbers. Older texts have also occasionally employed *J* as the symbol for this set.[29] This set is countably infinite: it is infinite but countable by definition. This is also expressed by saying that the cardinal number of the set is aleph-naught ($\aleph_0$).[30]

To be unambiguous about whether 0 is included or not, sometimes an index (or superscript) "0" is added in the former case, and a superscript "*" or subscript ">0" is added in the latter case:[1]

$$\mathbb{N}^0 = \mathbb{N}_0 = \{0, 1, 2, \ldots\}$$
$$\mathbb{N}^* = \mathbb{N}^+ = \mathbb{N}_1 = \mathbb{N}_{>0} = \{1, 2, \ldots\}.$$

Alternatively, natural numbers may be distinguished from positive integers with the index notation, but it must be understood by context that since both symbols are used, the natural numbers contain zero.[31]

$$\mathbb{N} = \{0, 1, 2, \ldots\}.$$
$$\mathbb{Z}^+ = \{1, 2, \ldots\}.$$

The double-struck capital N symbol, often used to denote the set of all natural numbers (see List of mathematical symbols).

Primality of one

Most early Greeks did not even consider 1 to be a number,[4] so they could not consider it to be a prime. By the Middle Ages and Renaissance many mathematicians included 1 as the first prime number.[5] In the mid-18th century Christian Goldbach listed 1 as the first prime in his famous correspondence with Leonhard Euler; however, Euler himself did not consider 1 to be a prime number. [6] In the 19th century many mathematicians still considered the number 1 to be a prime. For example, Derrick Norman Lehmer's list of primes up to 10,006,721, reprinted as late as 1956,[7] started with 1 as its first prime.[8] Henri Lebesgue is said to be the last professional mathematician to call 1 prime.[9] By the early 20th century, mathematicians began to arrive at the consensus that 1 is not a prime number, but rather forms its own special category as a "unit".[5]

A large body of mathematical work would still be valid when calling 1 a prime, but Euclid's fundamental theorem of arithmetic (mentioned above) would not hold as stated. For example, the number 15 can be factored as $3 \cdot 5$ and $1 \cdot 3 \cdot 5$; if 1 were admitted as a prime, these two presentations would be considered different factorizations of 15 into prime numbers, so the statement of that theorem would have to be modified. Similarly, the sieve of Eratosthenes would not work correctly if 1 were considered a prime: a modified version of the sieve that considers 1 as prime would eliminate all multiples of 1 (that is, all other numbers) and produce as output only the single number 1. Furthermore, the prime numbers have several properties that the number 1 lacks, such as the relationship of the number to its corresponding value of Euler's totient function or the sum of divisors function.[10]

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Algorithm Efficiency: O( )

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Big O notation

Lab 6

Test	Developed in	Type	Running time	Notes
AKS primality test	2002	deterministic	$O(\log^{6+\varepsilon}(n))$	
Elliptic curve primality proving	1977	deterministic	$O(\log^{5+\varepsilon}(n))$ *heuristically*	
Baillie-PSW primality test	1980	probabilistic	$O(\log^3 n)$	no known counterexamples
Miller–Rabin primality test	1980	probabilistic	$O(k \cdot \log^{2+\varepsilon}(n))$	error probability 4^{-k}
Solovay–Strassen primality test	1977	probabilistic	$O(k \cdot \log^3 n)$	error probability 2^{-k}
Fermat primality test		probabilistic	$O(k \cdot \log^{2+\varepsilon}(n))$	fails for Carmichael numbers

PARTICIPATION ACTIVITY 9.3.1: Determining Big O notation of a function.

Start 2x speed

O → "Order of"

Algorithm steps: $5 + 13 \cdot N + 7 \cdot N^2$

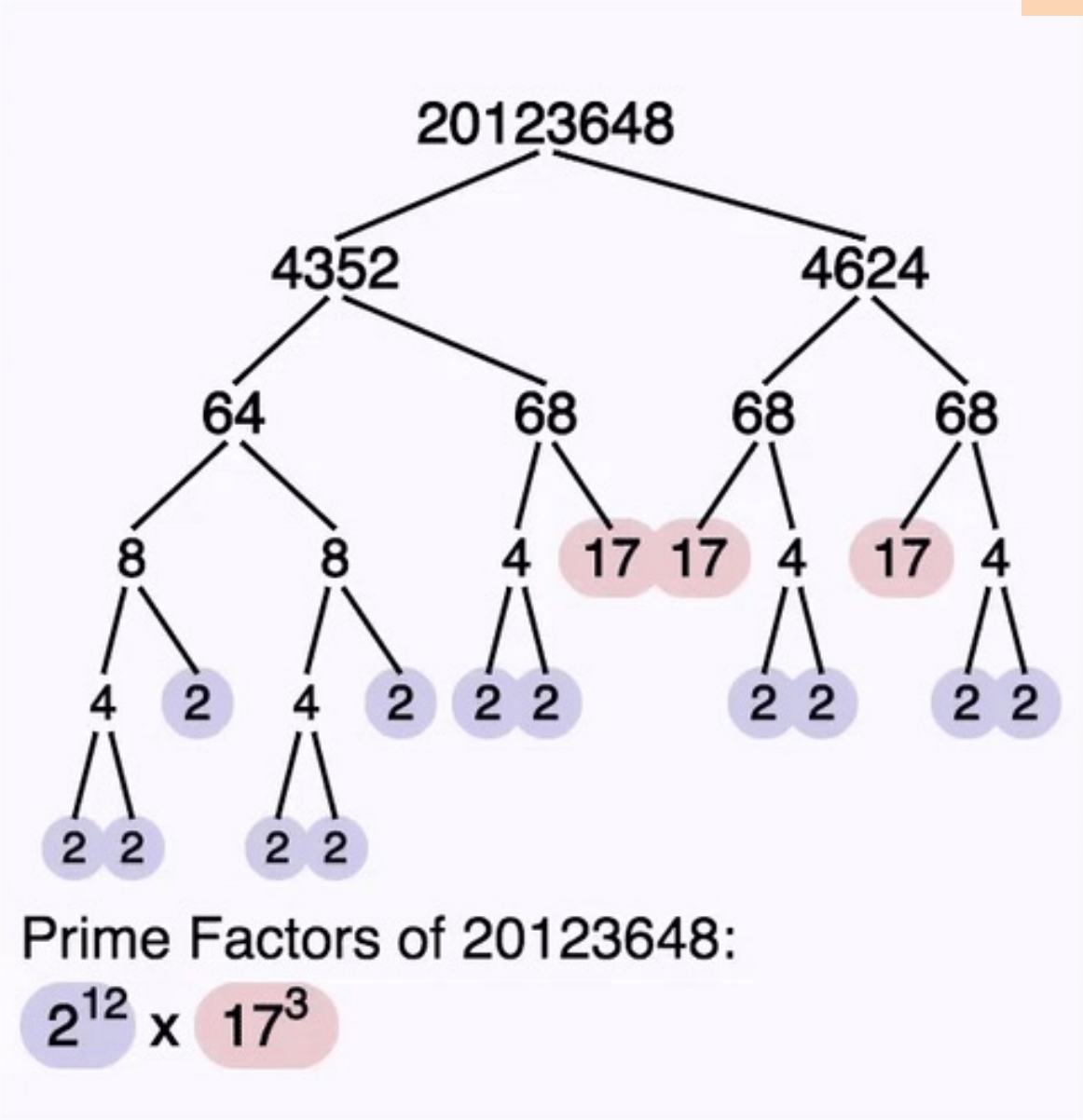
Big O notation: $O(5 + 13 \cdot N + 7 \cdot N^2) = O(7 \cdot N^2)$

$= O(N^2)$ Pronounced: "Oh N squared"

Rule 1: If f(x) is a sum of several terms, the highest order term is kept and others are discarded.

Rule 2: If f(x) has a term that is a product of several factors, all constants are omitted.

Prime Factors

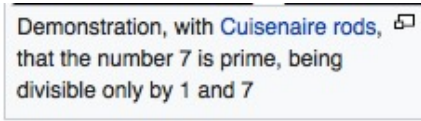


CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Primality Theory

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 6



Demonstration, with Cuisenaire rods, that the number 7 is prime, being divisible only by 1 and 7

The property of being prime is called primality. A simple but slow method of verifying the primality of a given number n is known as trial division. It consists of testing whether n is a multiple of any integer between 2 and $\sqrt{n}$. Algorithms much more efficient than trial division have been devised to test the primality of large numbers. These include the Miller–Rabin primality test, which is fast but has a small probability of error, and the AKS primality test, which always produces the correct answer in polynomial time but is too slow to be practical. Particularly fast methods are available for numbers of special forms, such as Mersenne numbers. As of January 2016, the largest known prime number has 22,338,618 decimal digits.

There are infinitely many primes, as demonstrated by Euclid around 300 BC. There is no known simple formula that separates prime numbers from composite numbers. However, the distribution of primes, that is to say, the statistical behaviour of primes in the large, can be modelled. The first result in that direction is the prime number theorem, proven at the end of the 19th century, which says that the probability that a given, randomly chosen number n is prime is inversely proportiona to its number of digits, or to the logarithm of n.

Many questions regarding prime numbers remain open, such as Goldbach's conjecture (that every even integer greater than 2 can be expressed as the sum of two primes), and the twin prime conjecture (that there are infinitely many pairs of primes whose difference is 2). Such questions spurred the development of various branches of number theory, focusing on analytic or algebraic aspects of numbers. Primes are used in several routines in information technology, such as public-key cryptography, which makes use of properties such as the difficulty of factoring large numbers into their prime factors. Prime numbers give rise to various generalizations in other mathematical domains, mainly algebra, such as prime elements and prime ideals.

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Primes Conjectures

Lab 6

- **Twins** conjecture
  - There exists an infinite number of ***Twin*** primes

It's well known that consecutive prime numbers become more widely separated from each other as they become larger. The twin prime conjecture stipulates that, despite this, there are infinitely many pairs of prime numbers that are separated from each other by only two (for example, 11 and 13).

2018

Zhang proved a weaker variant of this conjecture: that there are infinitely many pairs of prime numbers that are separated from each other by some fixed number that is greater than two, but less than 70 million.

Zhang's success followed from the construction of a new kind of prime-number filter. Previous work had used a strong filter, or sieve, which discarded prime numbers that were too far apart from each other. It allowed mathematicians to prove that there were always neighboring pairs of prime numbers closer together than some moving average. But they couldn't prove

Why is the twin prime problem famous?

First, the conjecture has been known for a long time—at least 100 or maybe 200 years, maybe longer. Second, the statement is very simple. It is easy to understand for many people. Third, the problem is believed to be very important.

Primes Conjectures

- **Goldbach** conjecture
  - Every even number >2 can be expressed as the sum of 2 primes

- Check all even numbers
  - ✓ Find 1st prime < N/2 = P1
  - ✓ Compute P2 = N – P1
  - ✓ Check if P2 is prime: search Primes array > P1

Primes Conjectures

Lab 6

Zhang's work also set off a flurry of follow-up activity in the mathematics community, including a collaboration of a dozen or so mathematicians called the Polymath8 project, which is attempting to reduce the maximum separation between primes in Zhang's proof to something less than 70 million. As of late August, it had provisionally lowered it to 4,680. Terence Tao, a math professor at UCLA and a member of the collaboration, described Zhang's work as a technical breakthrough that should lead to more progress on other questions in analytic number theory, including the Goldbach conjecture, which states that every even number is the sum of two primes.

Erica Klarreich, a Berkeley-based science writer who has a Ph.D. in mathematics and has written about Zhang, says his proof demonstrates the remarkable balance between order and randomness within the prime numbers. "Prime numbers are anything but random—they are completely determined," Klarreich says. "Nevertheless, they seem to behave in many respects like randomly-sprinkled numbers that eventually display all possible clumps and clusters. Zhang's work helps to put this conjectured picture of the primes on a solid footing."

Quantum Primes Algorithms

Lab 6

Algorithm: Primality Proving
Speedup: Polynomial
Description: Given an n-bit number, return a proof of its primality. The fastest classical algorithms are AKS, the best versions of which [393, 394] have essentially-quartic complexity, and ECPP, where the heuristic complexity of the fastest version [395] is also essentially quartic. The fastest known quantum algorithm for this problem is the method of Donis-Vela and Garcia-Escartin [396], with complexity $O(n^2(\log\ n)^3\ \log\ \log\ n)$. This improves upon a prior factoring-based quantum algorithm for primality proving [397] that has complexity $O(n^3\ \log\ n\ \log\ \log\ n)$. A recent result of Harvey and Van Der Hoeven [398] can be used to improve the complexity of the factoring-based quantum algorithm for primality proving to $O(n^3 \log n)$ and it may be possible to similarly reduce the complexity of the Donis-Vela-Garcia-Escartin algorithm to $O(n^2(\log\ n)^3)$ [399].

Lab

- **Lab 7**

Cryptography

(ref. "Crypto" slides)

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 7: Cryptography

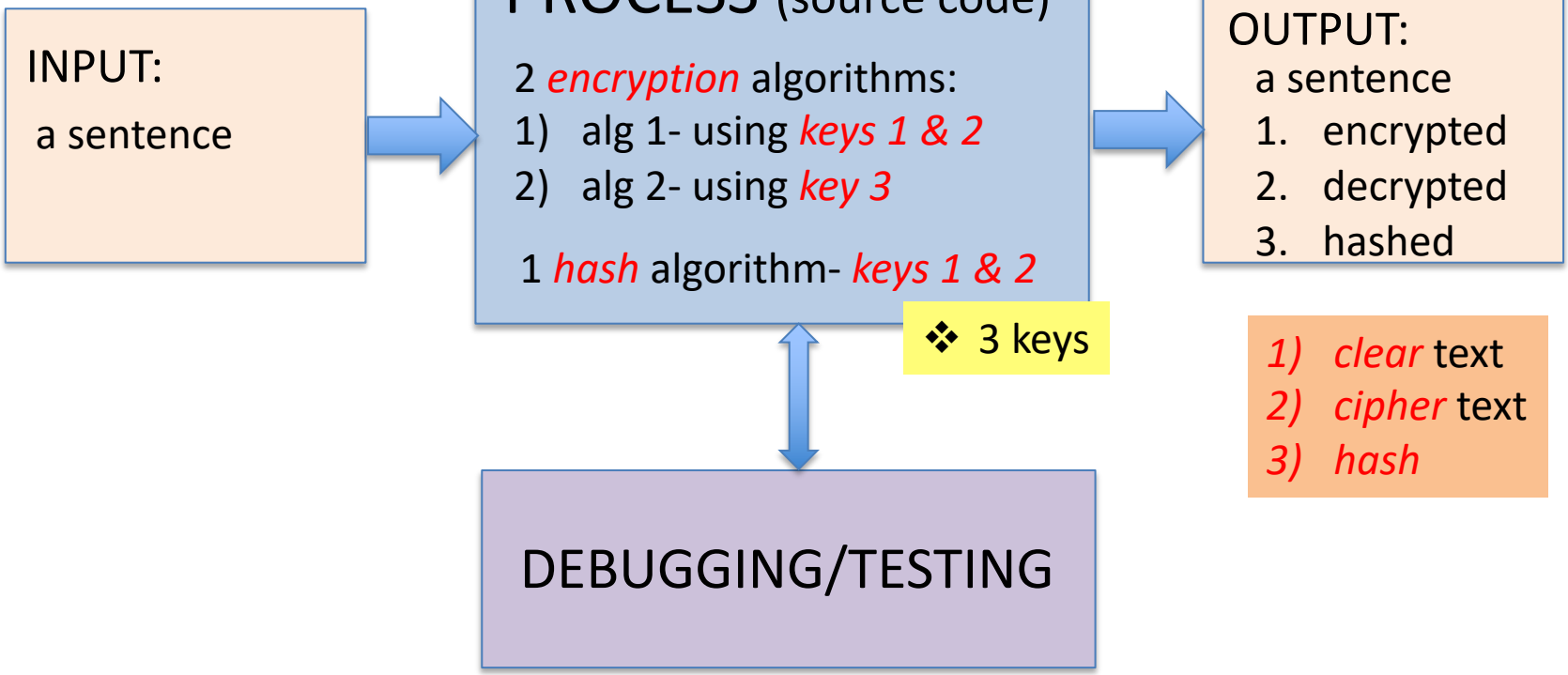
Lab 7

- *Methods*
- *Strings*
- *Arrays*

Rqts– INPUT: OUTPUT:

(see below)

- GUI
- GUI or Console

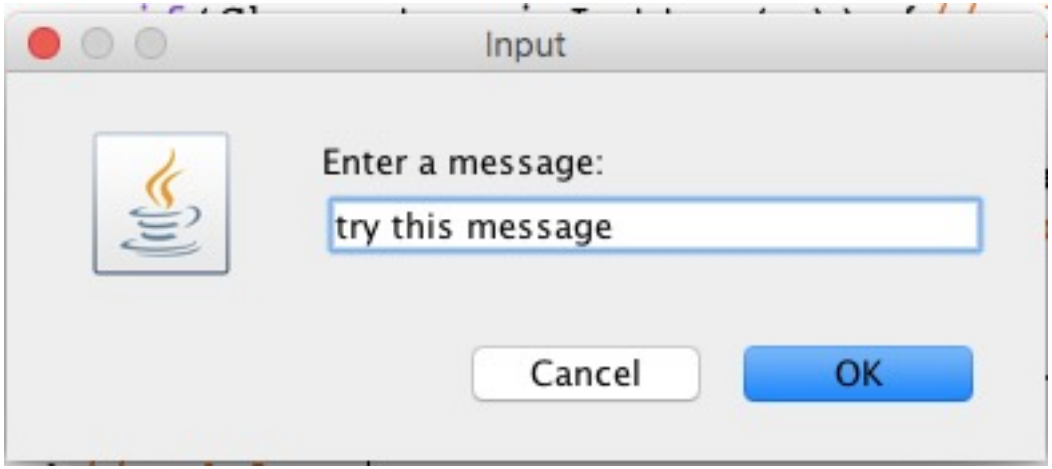


- 3 keys

1) *clear* text
2) *cipher* text
3) *hash*

Input GUI Box

Lab 7



Main Class

keys

Random numbers

```
4  file:  Lab7Cypher.java
5  */
6  // imports
7  import javax.swing.*;
8  import java.util.*;
9  //import java.io.*;
10 // **main class**
11 public class Lab7Cypher {
12    static final boolean $DEBUG = true;
13    static final int[] key1 = {5,2,1,7,0,9,8,3};
14    static final int[] key2 = {14,27,33,2,9,25,10,51};
15    static final String keyBinStr = "11010100";
16    static final int key3 =  Integer.parseInt(keyBinStr);
17    static final int keyLen = key1.length;//=8
18 //main method
19    public static void main(String[] args) {
20       //debug checks
21       if ($DEBUG) System.out.println("debug: starting main...");
22       if ($DEBUG) System.out.println("\u0061" + (char) 97);//test if 'a'
23       if ($DEBUG) System.out.println("keyBinStr= " + keyBinStr);//check
24       if ($DEBUG) System.out.println("key3= " + key3);
25       String clear = "encrypt this message";//test msg
26    //main loop
```

Main Loop


```
27    while(true) {
28       System.out.println("test string= " +clear);
29       //ENcrypt:  add key to clear msg
30       char[] cypherText11 = encrypt1(clear, key1);//key1
31       char[] cypherText12 = encrypt1(clear, key2);//key2
32       char[] cypherText23 = encrypt2(clear, key3);//alg2,key3
33       String cypherStr = Arrays.toString(cypherText11);
34       System.out.println("cypher msg= " + cypherStr);
35       for(char x: cypherText11) System.out.print(x);//show string
36       System.out.println();
37       for(char x: cypherText12) System.out.print(x);//show string
38       System.out.println();
39       //DEcrypt:  sub key from cypher msg
40       char[] clear11 = decrypt1(cypherText11,key1);
41       char[] clear12 = decrypt1(cypherText12,key2);
42       char[] clear23 = decrypt2(cypherText23,key3);
43       System.out.println("decrypted messages again...");
44       for(char x: clear11) System.out.print(x);//show string
45       System.out.println();
46       for(char x: clear12) System.out.print(x);//show string
47       System.out.println();
48       for(char x: clear23) System.out.print(x);//show string
49       System.out.println();
50       //perform hash
51       long hashx = hash(clear,key1);
52       System.out.println("hash value(key1)= " + hashx);
53       hashx = hash(clear,key2);
54       System.out.println("hash value(key2)= " + hashx);
55       //get next input msg
56       clear = JOptionPane.showInputDialog("Enter a message:");
57       if(clear == null) break;//STOP!
58    }//end loop
```

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Encryption Alg 1

DR JEFF SOFTWARE INDIE APP DEVELOPER

Lab 7

```
46    //start ENcrypt1:  additive
47    static char[] encrypt1(String msg, int[] key) {
48       char[] cyph = msg.toCharArray();
49       int k = 0;//key index
50       for(int i=0; i<msg.length(); i++) {
51          char x = cyph[i];
52          if(Character.isLetter(x)) {//only do letters
53             boolean Lcase = Character.isLowerCase(x);
54             boolean Ucase = Character.isUpperCase(x);
55             x += (key[k]);//add
56             if(Lcase && x > 'z') x = (char) ('a'+(x-'z')-1);//mod 26
57             else if(Ucase && (x >'Z')) x = (char) ('A'+(x-'Z')-1);
58             k = k++ % keyLen;//rotate around key
59          }//end if
60          //x= (char)('a' + i++);
61          if ($DEBUG) System.out.print(x + ".");
62          cyph[i] = x;//replace char
63       }//end loop
64       if ($DEBUG) System.out.println();
65       return cyph;
66    }//end encrypt1
```

```
x = (char) ('A'+(x-'Z')-1);
```

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz

97 98 99 120 121 122

ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ

65 66 67 90 91 92

ASCII Codes- 7-bit

USASCII code chart

65

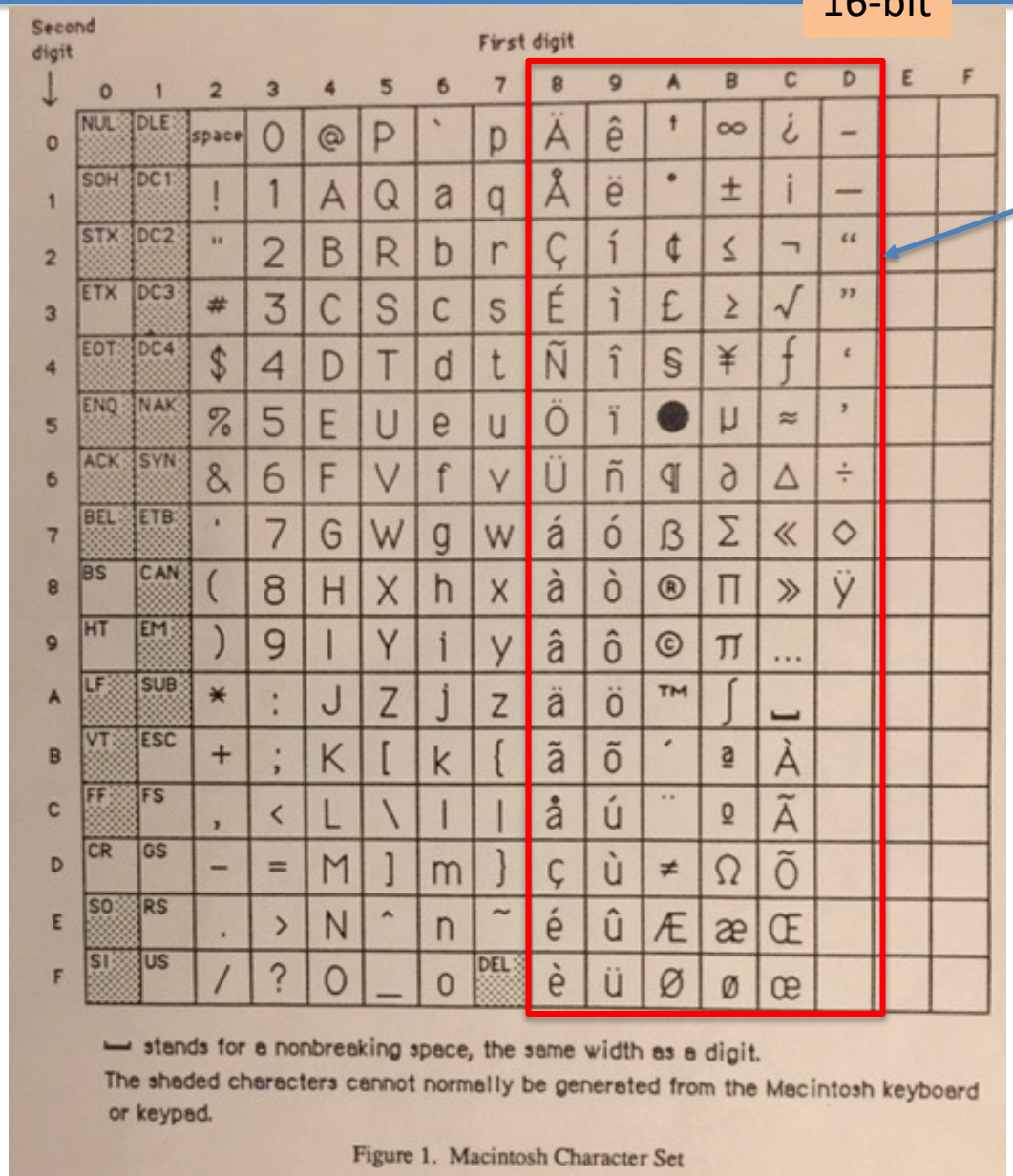
97

Row	b_4	b_3	b_2	b_1	000	001	010	011	100	101	110	111
					0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
1	0	0	0	1	SOH	DC1	!	1	A	Q	a	q
2	0	0	1	0	STX	DC2	"	2	B	R	b	r
3	0	0	1	1	ETX	DC3	#	3	C	S	c	s
4	0	1	0	0	EOT	DC4	$	4	D	T	d	t
5	0	1	0	1	ENQ	NAK	%	5	E	U	e	u
6	0	1	1	0	ACK	SYN	&	6	F	V	f	v
7	0	1	1	1	BEL	ETB	'	7	G	W	g	w
8	1	0	0	0	BS	CAN	(	8	H	X	h	x
9	1	0	0	1	HT	EM	)	9	I	Y	i	y
10	1	0	1	0	LF	SUB	*	:	J	Z	j	z
11	1	0	1	1	VT	ESC	+	;	K	[	k	{
12	1	1	0	0	FF	FS	,	<	L	\	l	|
13	1	1	0	1	CR	GS	-	=	M	]	m	}
14	1	1	1	0	SO	RS	.	>	N	^	n	~
15	1	1	1	1	SI	US	/	?	O	_	o	DEL

IANA encourages use of the name "US-ASCII" for Internet uses of ASCII

Old Mac Char Codes

16-bit



Second digit ↓	First digit 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	space	0	@	P	`	p	Ä	ê	†	∞	¿	–		
1	SOH	DC1	!	1	A	Q	a	q	Å	ë	°	±	¡	—		
2	STX	DC2	"	2	B	R	b	r	Ç	í	¢	≤	¬	“		
3	ETX	DC3	#	3	C	S	c	s	É	ì	£	≥	√	”		
4	EOT	DC4	$	4	D	T	d	t	Ñ	î	§	¥	ƒ	‘		
5	ENQ	NAK	%	5	E	U	e	u	Ö	ï	•	µ	≈	’		
6	ACK	SYN	&	6	F	V	f	v	Ü	ñ	¶	∂	Δ	÷		
7	BEL	ETB	'	7	G	W	g	w	á	ó	ß	Σ	«	◊		
8	BS	CAN	(	8	H	X	h	x	à	ò	®	Π	»	ÿ		
9	HT	EM	)	9	I	Y	i	y	â	ô	©	π	…			
A	LF	SUB	*	:	J	Z	j	z	ä	ö	™	∫	␣			
B	VT	ESC	+	;	K	[	k	{	ã	õ	´	ª	À			
C	FF	FS	,	<	L	\	l	|	å	ú	¨	º	Ã		
D	CR	GS	–	=	M	]	m	}	ç	ù	≠	Ω	õ			
E	SO	RS	.	>	N	^	n	~	é	û	Æ	æ	Œ			
F	SI	US	/	?	O	_	o	DEL	è	ü	Ø	ø	œ			

␣ stands for a nonbreaking space, the same width as a digit.
The shaded characters cannot normally be generated from the Macintosh keyboard or keypad.

Figure 1. Macintosh Character Set

unique special chars

Encryption Results

Lab 7

```
test string= encrypt this message
j.s.h.w.d.u.y. .y.m.n.x. .r.j.x.x.f.l.j.
s.b.q.f.m.d.h. .h.v.w.g. .a.s.g.g.o.u.s.
QZWFMD@ @\]G YQGGUSQ
81.90.87.70.77.68.64.20.64.92.93.71.20.89.81.71.71.85.83.81.
cypher msg= [j, s, h, w, d, u, y,  , y, m, n, x,  , r, j, x,
jshwduy ymnx rjxxflj
sbqfmdh hvwg asggous
```

key 1
key 2
key 3
ASCII
key 1
key 2

```
test string= 123 FOR
1.2.3. .K.T.W.
1.2.3. .T.C.F.
  r{f
5.6.7.20.114.123.102.
cypher msg= [1, 2, 3,  , K, T, W]
123 KTW
123 TCF
```

Decryption Alg 1

Lab 7

```
67    //start DEcrypt
68    static char[] decrypt1(char[] xcyf, int[] key) {//String msg
69       int k = 0;//key index
70       for(int i=0; i<xcyf.length; i++) {
71          char x = xcyf[i];
72          if(Character.isLetter(x)) {
73             boolean Lcase = Character.isLowerCase(x);
74             boolean Ucase = Character.isUpperCase(x);
75             x -= (key[k]);//subtract
76             if(Lcase && x < 'a') x = (char) ('z'-('a'-x)+1);//mod 26
77             else if(Ucase && (x <'A')) x = (char)('Z'-('A'-x)+1);
78             k = k++ % keyLen;//rotate around key
79          }//end if
80          if ($DEBUG) System.out.print(x + '.' -'.');
81          xcyf[i] = x;//replace char
82       }//end loop
83       if ($DEBUG) System.out.println();
84       return xcyf;
85    }//end decrypt1
```

```
x = (char) ('Z'-('A'-x)+1);
```

edit

```
e.n.c.r.y.p.t. .t.h.i.s. .m.e.s.s.a.g.e.
e.n.c.r.y.p.t. .t.h.i.s. .m.e.s.s.a.g.e.
encrypt this message
decrypted messages again...
encrypt this message
encrypt this message
encrypt this message
```

```
1.2.3. .F.O.R.
1.2.3. .F.O.R.
123 FOR
decrypted messages again...
123 FOR
123 FOR
123 FOR
```

Encryption Alg 2

Lab 7

- Simple logic expression with "XOR" (^)
- Output:
  - cypher text
  - decrypted text (clear)

Operator	Name	Example	Result
&	Bitwise AND	11101 & 00111	00101
|	Bitwise OR	00010 | 11000	11010
^	Bitwise XOR	00111 ^ 11111	11000

bit flip

```
100    //start ENcrypt2:  XOR
101    static char[] encrypt2(String msg, int key) {
102       char[] cyph = msg.toCharArray();
103       int k = 0;//key index
104       for(int i=0; i<msg.length(); i++) {
105          char x = cyph[i];
106          x ^= key;
107          if ($DEBUG) System.out.print(x);
108          cyph[i] = x;//replace char
109       }//end loop
110       if ($DEBUG) System.out.println();
111       for(char ch: cyph){//print ASCII codes
112          if ($DEBUG) System.out.print(ch + 0);//ASCII
113          if ($DEBUG) System.out.print(".");
114       }//end for
115       if ($DEBUG) System.out.println();
116       return cyph;
117    }//end encrypt1
```

Decryption Alg 2


```
118    //start DEcrypt2:  XOR
119    static char[] decrypt2(char[] xcyf, int key) {
120       int k = 0;//key index
121       for(int i=0; i<xcyf.length; i++) {
122          char x = xcyf[i];
123          x ^= key;
124          if ($DEBUG) System.out.print(x);
125          xcyf[i] = x;//replace char
126       }//end loop
127       if ($DEBUG) System.out.println();
128       return xcyf;
129    }//end decrypt2
```

Hash Algorithm

Lab 7

- Simple arithmetic expression with "*"
- Use "long" type
- Output
  - Integer value (long)
  - Bonus: Format in hex digits

```
130    //start hash
131    static long hash(String msg, int[] key) {
132       long hashVal = 0;
133       char[] hashArr = msg.toCharArray();
134       int k = 0;//key index
135       for(int i=0; i<msg.length(); i++) {
136          hashVal += hashArr[i] * key[k];
137          k = k++ % keyLen;//rotate around key
138       }//end for
139       return hashVal;
140    }//end hash
```

Hash Algorithm

Lab 7

```
test string= encrypt this message
hash value(key1)= 10090
hash value(key2)= 28252
```

```
123 FOR
hash value(key1)= 2065
hash value(key2)= 5782
```

```
FOR 123
hash value(key1)= 2065
hash value(key2)= 5782
```

Collision!

Console Output

Lab 7

```
----jGRASP exec: java Lab7Cypher
debug: starting main...
aa
keyBinStr= 11010100
key3= 11010100
test string= encrypt this message
j.s.h.w.d.u.y. .y.m.n.x. .r.j.x.x.f.l.j.
s.b.q.f.m.d.h. .h.v.w.g. .a.s.g.g.o.u.s.
QZWFMD@ @\]G YQGGUSQ
cypher msg= [j, s, h, w, d, u, y,  , y, m, n, x,
jshwduy ymnx rjxxflj
sbqfmdh hvwg asggous
e.n.c.r.y.p.t. .t.h.i.s. .m.e.s.s.a.g.e.
e.n.c.r.y.p.t. .t.h.i.s. .m.e.s.s.a.g.e.
encrypt this message
decrypted messages again...
encrypt this message
encrypt this message
encrypt this message
hash value(key1)= 10090
hash value(key2)= 28252

----jGRASP: operation complete.
```

encrypted

decrypted

hashes

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Lab 8: Tic-Tac-Toe

Lab 8

- *Methods*
- *Strings*
- *Arrays*
- *Games*

- GUI

Rqts– INPUT:
1. No. players
2. Moves from Player 1 (2)

OUTPUT:
1. Board updates
2. Win/lose/draw

- GUI or Console

- Create **Class**
  - *Board/Game*

INPUT:
1-Players: 0, 1, 2 or Quit
2-Player moves

PROCESS (source code)
1) Ask "num players = 0-2 or Quit?"
2) Get next move
3) Update board
4) Check for win/done?
5) Play again?

OUTPUT:
1-Updated board
2-Winner/draw ("cats game")

Player modes:
0: *Random* X & O
1: X plays, O *computer**
2: X, O each play

DEBUGGING/TESTING

*computer:
- **required: random**
- bonus: algorithm

Tic-Tac-Toe Moves

Lab 8

Player modes:
0: *Random* X, O
1: X plays, O *computer*
2: X, O play

- Use Methods for each
  - Random
  - Player
  - Computer (bonus)

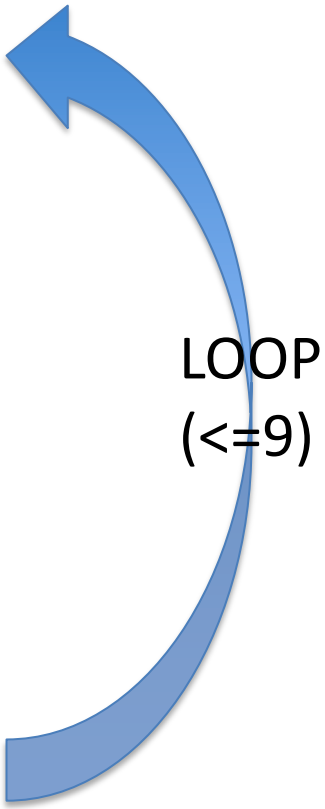
- Random
  - Math.random( ) * 9
  - Random.nextInt(9)

➢ Use either one

- Computer (bonus)
  - Follow my slides per O move
  - Use my strategy algorithm for first 2 moves
  - Extra bonus for using algorithm for moves 3 and 4.

Overall Flow

1) get *player number* choice (0/1/2 in GUI)
2) Print BOARD (label Rows & Cols, console)
3) get player *moves* (X or X/O GUI)
   - Player
     - GUI input
   - Random
     - Math.random( ) * 9
     - Random.nextInt(9)
   - Computer
     - **Random** –or– Follow my slides per O move
     - Use my strategy algorithm for first 2 moves
     - Bonus for using algorithm for moves 3 and 4
4) Check: Win, Lose or Draw
   ("score" method → after 3rd moves



- Note: CONSOLE has *permanence*, while GUI box is temporary

Structure

Lab 8

OOP
Structures

Lab8 Main Class

MAIN method

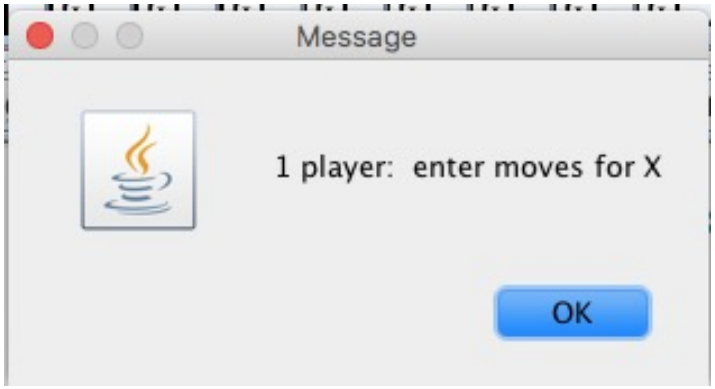
```
TicTT Tgame = new TicTT();
//Ask play again?
```

Lab8 Class: TicTT

- **Data Fields**
- **Constructor**
- ***play*** method — Driver
- ***randMove*** method
- ***compMove*** method
- ***playerMove*** method
- ***prtBoard*** method
- ***score*** method

Sample GUI

Lab 8

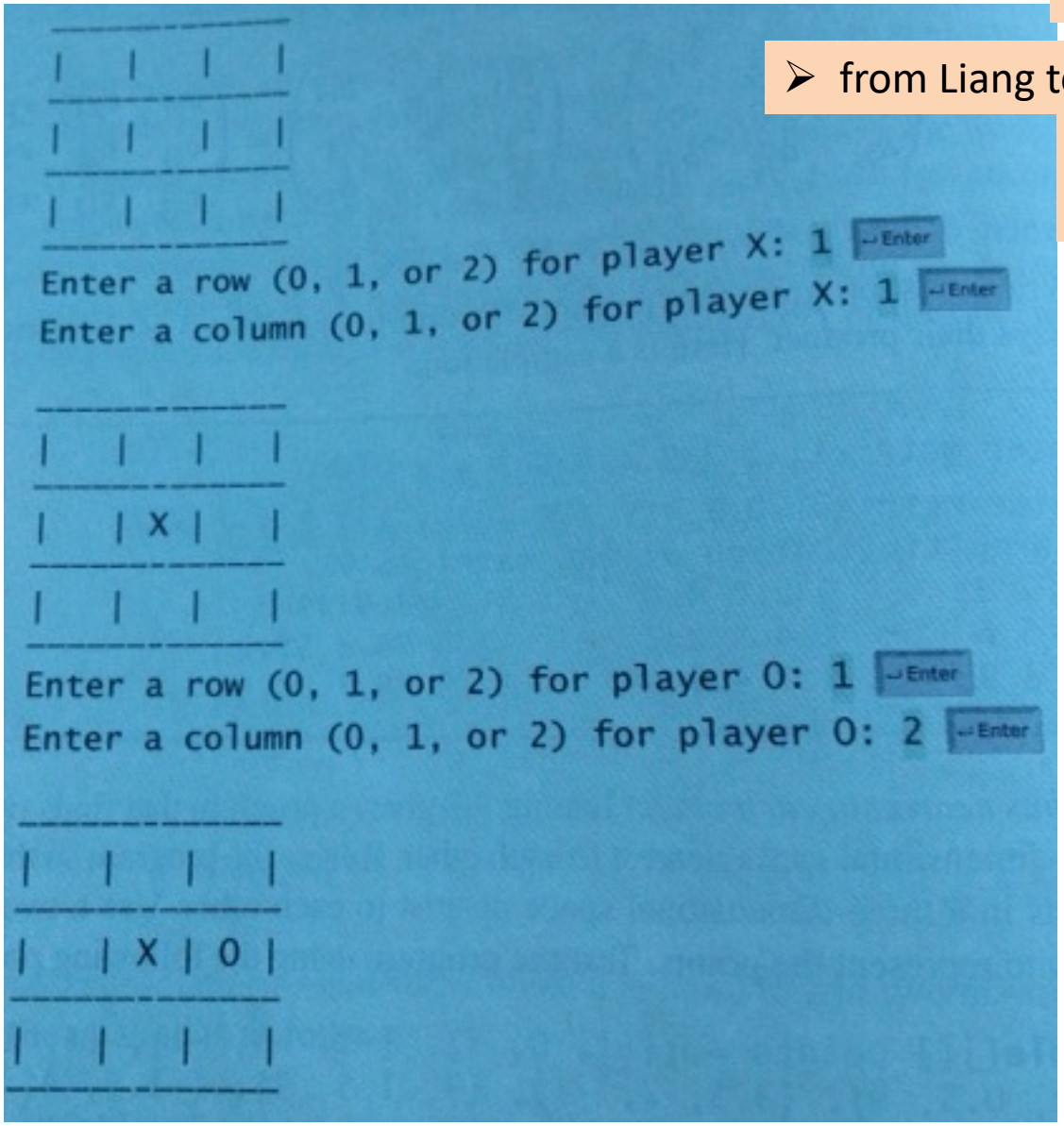


Textbook Input/Output

Lab 8

- from Liang textbook

Ref: Sec 8.9
pp. 308-9



Sample Output

Lab 8

```
Position Map

A1 | A2 | A3 |
-------------
B1 | B2 | B3 |
-------------
C1 | C2 | C3 |
```

added col labels

added row labels

```
Move number: 1
   1     2     3
a     |     |     |
   -------------
b     |     |  X  |
   -------------
c     |     |     |
```

```
 O |   | X |
-----------
 O | X |   |
-----------
   |   |   |

Computer move...

 O |   | X |
-----------
 O | X |   |
-----------
 X |   |   |

Computer wins with a solution on the Right Diagonal!
```

```
Move number: 5
   1     2     3
a  O |     |     |
   -------------
b  O |     |     |
   -------------
c  X |  X  |  X  |

   ======

state= 000000111
X Wins! by BR
```

State Strings

Lab 8

```
Move number: 1
a     |     |     |
  -------------
b     |  X  |     |
  -------------
c     |     |     |

   ======

Move number: 2
state= 000010000
a     |     |     |
  -------------
b     |  X  |     |
  -------------
c     |     |  O  |

   ======

Move number: 3
state= 000000001
```

- state string X
- state string O

Sample Output


```
----jGRASP exec: java Lab8Tictactoe
1 player X, computer plays O
starting new game...

a     |     |     |
  -------------
b     |     |     |
  -------------
c     |     |     |

   ======

Move number: 1
a  X  |  X  |  X  |
  -------------
b     |     |     |
  -------------
c     |     |     |

   ======

Move number: 4
a  X  |  X  |  X  |
  -------------
b     |     |     |
  -------------
c  O  |  O  |     |
```

❖ Test of Score method

➢ X picks top row (3 moves)

➢ X wins!

```
Move number: 6
state= 111000000
win state= 111000000
XWins! by Top Row
-------------
```

➢ O picks bottom row (2 moves)

Sample Output


```
----jGRASP exec: java Lab8Tictactoe
1 player X, computer plays O
starting new game #1

Move number: 4
   1   2   3
a  X |   |   |
  -------------
b    | X |   |
  -------------
c  O |   | O |

   ======

state= 110010000
Move number: 5
   1   2   3
a  X | X |   |
  -------------
b    | X |   |
  -------------
c  O |   | O |

   ======
```

➢ O wins!

```
state= 000000111
O Wins! by BR
Move number: 5
   1   2   3
a  X | X |   |
  -------------
b    | X |   |
  -------------
c  O | O | O |

   ======
```

➢ X wins!

```
Move number: 5
   1   2   3
a  X | O |   |
  -------------
b  O | X |   |
  -------------
c    |   | X |

   ======

X Wins! by LD
```

➢ Cat's game

```
state= 101101001
Move number: 9
   1   2   3
a  X | O | X |
  -------------
b  X | O | X |
  -------------
c  O | O | X |

   ======

------
cat's game!
------------
```

Sample Output


```
0 players: random moves both players
starting new game #1

Move number: 2
   1   2   3
a      |     |     |
  ------------
b      |  X  |     |
  ------------
c      |  O  |     |

   ======

Move number: 3
   1   2   3
a      |     |  X  |
  ------------
b      |  X  |     |
  ------------
c      |  O  |     |

   ======
```

```
Move number: 4
   1   2   3
a      |     |  X  |
  ------------
b      |  X  |  O  |
  ------------
c      |  O  |     |

   ======

state= 001010100
X Wins! by RD
Move number: 4
   1   2   3
a      |     |  X  |
  ------------
b      |  X  |  O  |
  ------------
c   X  |  O  |     |

   ======
```

- X wins!

Code: Main


```
12    public static void main(String[] args) {
13       //inits
14       boolean cont= true;
15       final byte end= 5; //stop after move 5
16       String[] options = {"0", "1", "2", "Quit"};
17       byte r,c;//indices
18       while(cont){//main loop
19          //ask user
20          int players = JOptionPane.showOptionDialog(null, "select number of players",
21       "options", 0, JOptionPane.INFORMATION_MESSAGE, null, options, options[1]);
22          switch(players){
23             case 0: //0-player
24              System.out.println("0 players- random moves both players");
25              JOptionPane.showMessageDialog(null, "0-player:  random game, no players");
26             break;
27             case 1: //1-player
28              System.out.println("1 player X, computer plays O");
29              JOptionPane.showMessageDialog(null, "1 player:  enter moves for X");
30             break;
31             case 2: //2-player
32              System.out.println("2 players- enter moves for both X and O");
33              JOptionPane.showMessageDialog(null, "2-player:  enter moves for X & O");
34             break;
35             default: //Quit
36              JOptionPane.showMessageDialog(null, "You have quit-- goodbye!");
37              System.exit(0);
38          }//end switch
39          //start new game**
40          Game Gm = new Game();//instantiate
41          //Gm.resetBoard();->now in constructor
42          System.out.println("starting new game...\n");
43          //play: get moves fm X and O, score
44          Gm.play(players);
45          cont = JOptionPane.showConfirmDialog(null,"Play again?")==0;
46       }//end main loop
```

```
gameNum++;
System.out.println("starting new game #" +gameNum +"\n");
```

End of Main: Play Again?


```
45            cont = JOptionPane.showConfirmDialog(null,"Play again?")==0;
46         }//end main loop
47         //DONE!
48         JOptionPane.showMessageDialog(null, "You quit, good-bye!");
49      } //end main
50   }//end main Class
```

Code: Game Class

Class+Methods | Lab 8

```
51 //class to create a new game
53 class Game {
54    //data fields (globals)
55    static final boolean $DEBUG = false; //Lab8Tict=X,2=O
56    static final char[] let = {' ','X','O'};
57    static final int stop = 9;//stop after N moves
58    static final String empty = "";
59    //non-static = instance
60    int moves = 0;
61    boolean quit = false;
62    //3x3 array: 0=space,1=X,2=O
63    int[][] board = {{0,0,0},{0,0,0},{0,0,0}};
64
65    //constructor
66    Game() {
67    }//end constr
      //print board
      void prtBoard(){
         System.out.println("Move number: " + moves);
72       System.out.println("\t1    2    3");
73       char[] row = {'a','b','c'};//display
74       for (int r=0;r<3;r++) { //each row
75       System.out.print(row[r] + " ");
76          for (int c=0;c<3;c++) { //each col per row
77             char xox = let[board[r][c]];//map 0..2 into X,O
78             System.out.print(" " + xox + " | ");
79          }//end cols c
80          if (r<2) System.out.println("\n  ------------");
81       }//end rows r
82       System.out.println("\n\n   ======\n");//space out
83    }//end prtBoard
```

- ❖ reset board
- ❖ empty
- ❖ not static
- ❖ print board

Play Method (Driver)

Lab 8

```
85    void play(int nPlay) {//driver for game
86       boolean xo = true;//true=X; false=O
87       int stop = 9;//stop after 9 moves
88       for(moves=0; moves<=9; moves++) {//loop
89          prtBoard();
90          switch(nPlay){
91             case 0://random moves X and O (2 loops)
92                randPlay(xo);//pick random move X/O
93             break;
94             case 1://X plays, O computer
95                if(xo) playerMove(xo);
96                else compMove(xo);
97             break;
98             case 2://X plays, O plays
99                playerMove(xo);
100            break;
101            default://invalid
102               JOptionPane.showMessageDialog(null, "invalid game selection, try again");
103            return;
104         }//end switch
105         if (quit) return;
106         if (moves >= 4) {//check win after 5 moves
107            if (score(xo)) {//score X/O
108               prtBoard();//final board on win
109               return;
110            }
111         }//end if
112         if (moves >= stop) break;//exit loop early
113         xo = !xo;//switch X/O
114      }//end for loop
115      prtBoard();//final board
116         JOptionPane.showMessageDialog(null, "cat's game!");
117         System.out.println("------\ncat's game!");
118         System.out.println("------------");
119   }//end play
```

- Improve *prtBoard* calls

Random Play

Lab 8

```
125     //random play
126     void randPlay(boolean xo) {
127        int r = 0, c = 0;
128        int mxo = (byte)(Math.random() *9);//0..8
129        for(int i=0; i<25; i++) { //find empty
130           //if($DEBUG) System.out.print(mxo);
131           r = mxo/3; c = mxo%3;
132           if(board[r][c]==0) break;//found empty
133           mxo = ++mxo %9;//try next square
134        }//end loop: found empty
135        board[r][c] = (xo? 1:2);//set X/O
136     }//end randPlay
```

- **xo** (boolean) → *whose* move: X,O
- **xox** (char) → map: " ",X,O (from "let")
- mxo (int) → *next* move: X,O (random++ %9)

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110L

Moves: Player, Computer

Lab 8

DR JEFF SOFTWARE INDIE APP DEVELOPER

```
133    void playerMove(boolean xo) {
134       char xox = (xo? 'X':'O');
135       boolean again = true;
136       while(again) {//loop for valid move
137          String mov = JOptionPane.showInputDialog("Enter next move for " + xox);
138          //check for valid input
139          if (mov == null) {
140             JOptionPane.showMessageDialog(null, "you QUIT game!");
141             System.out.println("player Quit: " + xox);
142             quit = true;
143             return; }//done
144          if (mov.length() != 2) {
145             JOptionPane.showMessageDialog(null, "Invalid or empty input:  try again!");
146             continue; }
147          char Rch = mov.charAt(0);
148          char Cch = mov.charAt(1);
149          if (Rch >='a' && Rch <='c' && Cch >='1' && Cch <='3') {//valid
150             int r = Rch - 'a';//0..2
151             int c = Cch - '1';//0..2
152             if (board[r][c] == 0) {//empty
153                board[r][c] = (xo? 1:2);
154                again = false;}//done
155             else
156                JOptionPane.showMessageDialog(null, "Square already taken, try again!");
157           }//end if
158          else //not valid
159             JOptionPane.showMessageDialog(null, "Invalid input:  try again!");
160       }//end loop
161    }//end player move
162    //computer move
163    void compMove(boolean xo) {
164       //use random play
165       randPlay(xo);
166       //else use algorithm
167    }//end computer move
```

Score Method

Lab 8

```
168    //score game method
169    boolean score(boolean xo){
170       char xox = (xo? 'X':'O');
171       int xo12 = (xo? 1:2);
172       String msg = "";
173       //solutions array of 8
174       String[] solutions = {"111000000","000111000","000000111",
175    "100100100","010010010","001001001","100010001","001010100"};
176       String[] solTxt = {"Top Row", "Middle Row", "BR", "LC", "MC", "RC", "LD", "RD"};
177       //map current board state to bit strings for X or O
178       String state = "";
179       for (int r=0; r<3; r++){//build state string
180          for (int c=0; c<3; c++){
181             if (board[r][c]==xo12) state += "1"; else state += "0";
182       }}//end fors
183       if ($DEBUG) System.out.println("state= " + state);
184       //check for WIN: create match string
185       int stI = Integer.parseInt(state);
186       for (int ix=0; ix<8; ix++) {
187          int sol = Integer.parseInt(solutions[ix]);
188          boolean win = (stI == sol);//mask=sol & stI
189          if (win) {
190             msg = xox +" Wins! " +"by " +solTxt[ix];
191             System.out.println(msg);
192             JOptionPane.showMessageDialog(null, msg);
193             return true; }
194       }//end for
195       return false; //no win
196    }//end score
```

- generate “state” string
- bitwise AND (mask)

Score Method Revised

Lab 8

```
180    boolean score(boolean xo){
181       char xox = (xo? 'X':'O');
182       int xo12 = (xo? 1:2);
183       //solutions array of 8
184       String[] solutions = {"111000000","000111000","000000111",
185          "100100100","010010010","001001001","100010001","001010100"};
186       String[] solTxt = {"Top Row", "Middle Row", "BR", "LC", "MC", "RC", "LD",
187       //map current board state to bit strings for X or O
188       String state = "";
189       for (int r=0; r<3; r++){//build state string
190          for (int c=0; c<3; c++){
191             state += (board[r][c]==xo12? "1":"0");//map bits to X or O
192          }}//end fors
193       if ($DEBUG) System.out.println("state= " + state);
194       //check for WIN: create match string
195       int stDec = Integer.parseInt(state);//decimal int
196       int stBin = Integer.parseInt(state,2);//binary int
197       if ($DEBUG) System.out.printf("soln as bitstring in hex= %h, dec= %d \n",
198       for (int i=0; i<8; i++) {
199          int solBin = Integer.parseInt(solutions[i],2);//bin int
200          int stMasked = stBin & solBin;//bitwise AND: mask
201          boolean win = (stMasked == solBin);//bit strings
202          if (win) {
203             printMsg(xox +" Wins! " +"by " +solTxt[i]);
204             return true; }
205       }//end for
206       return false;//no win
```

- generate “state” string
- bitwise AND (mask)

Sample Code: Solutions

Lab 8

- solution sets (8)

```
173     //solutions array of 8
174     String[] solutions = {"111000000","000111000","000000111",
175  "100100100","010010010","001001001","100010001","001010100"};
176     String[] solTxt = {"Top Row", "Middle Row", "BR", "LC", "MC", "RC", "LD", "RD"}
```

```
                                                    //->complete it

public final String SOLUTION_TEXT[]={"Top Row",//list of solution text
                    "Middle Row",
                    "Bottom Row",
                    "Left Column",
                    "Middle Column",
                     "Right Column",
                     "Left Diagonal",
                     "Right Diagonal",
                     "None"};
```

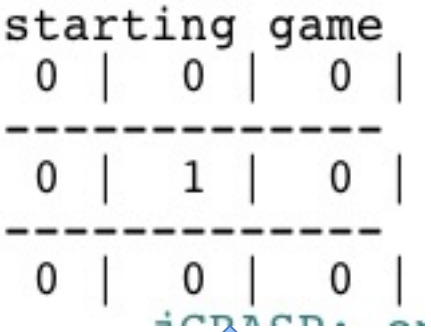
Computer Moves for O

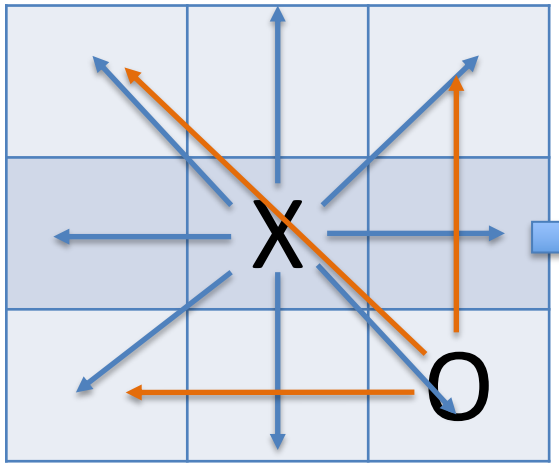
❖ optional bonus

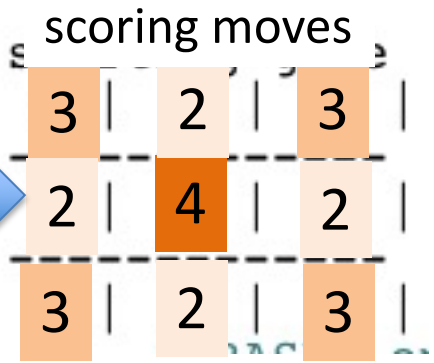
Lab 8

```
void computer(int moveNumber) {
   switch(moveNumber) {
      case 1:
         //use my strategy for 1st move
      case 2:
         //use my strategy for 2nd move
      case 3:
         //use my strategy for 3rd move
      case 4:
         //use my strategy for 4th move
   } //end switch
} //end computer
```

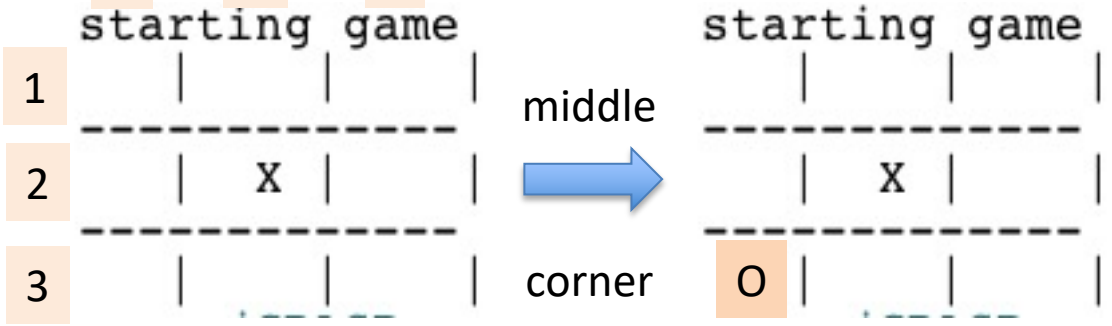
Strategy: 1st Move



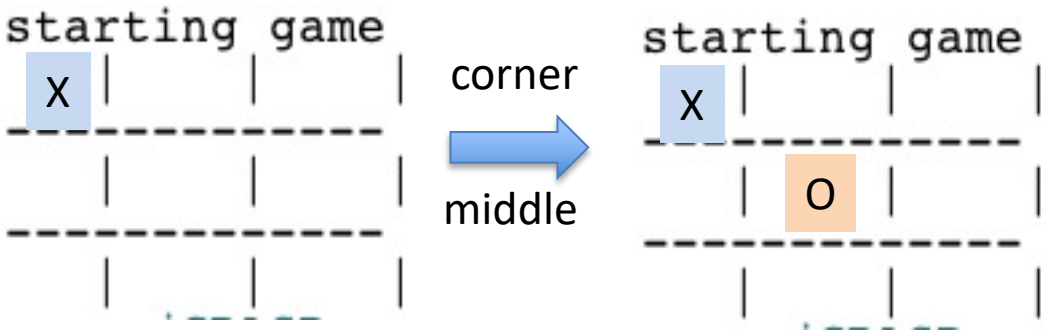




a b c



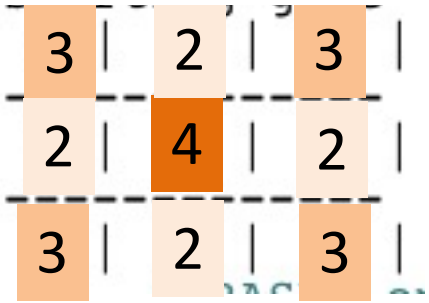
O strategy:
1. if X in *middle*, take a *corner*
2. else take *middle*



Strategy: 1st Move

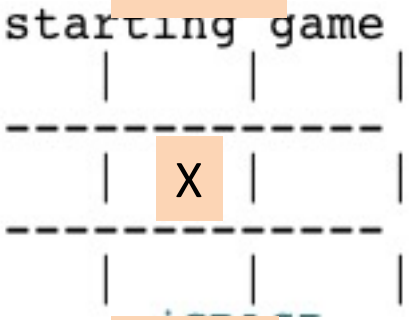
Lab 8

initial state scoring

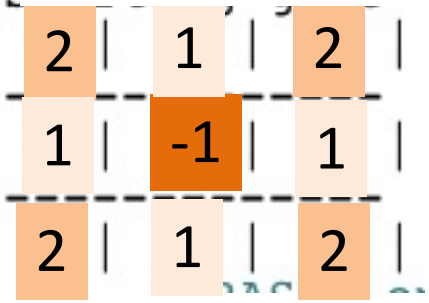



```
switch(N) {
case 1: //corner
case 2: //end
case 3: //corner
case 4: //end
case 5: //middle
case 6: //end
case 7: //corner
case 8: //end
case 9: //corner
```

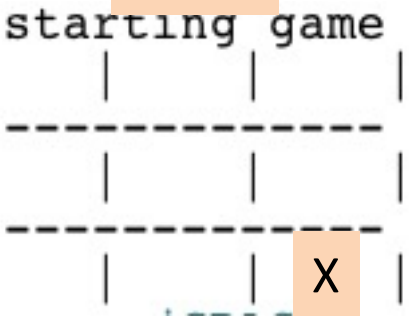
middle



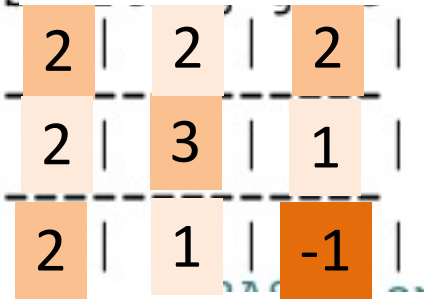
scoring change



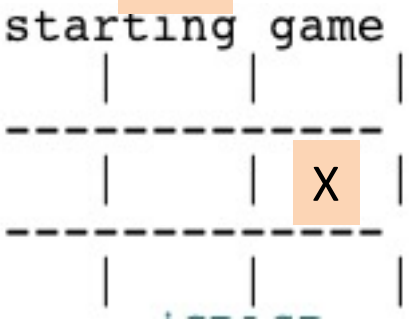
corner



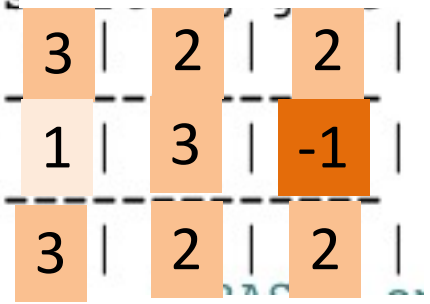
scoring change



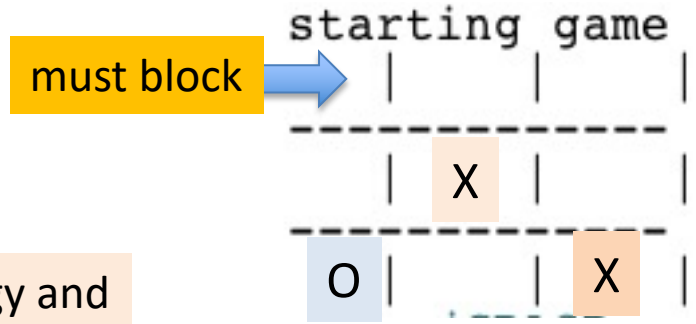
end



scoring change

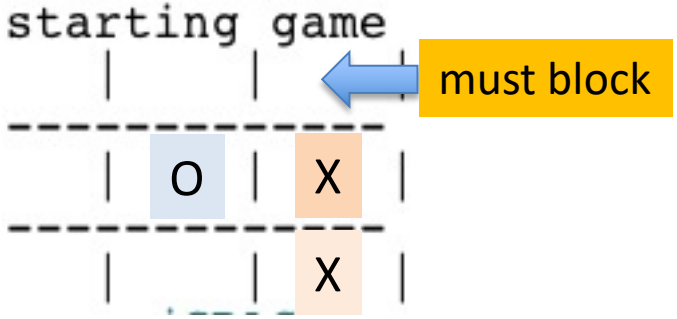


Strategy: 2nd Move

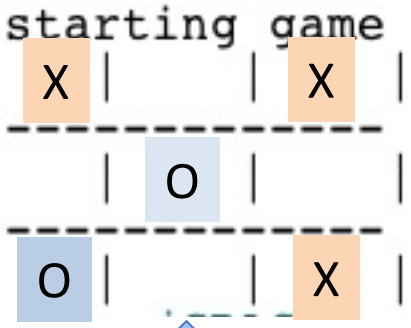


O strategy:
1. check "must block"
2. check X at opposite *corners*
   a. take any *end*
3. else take best move

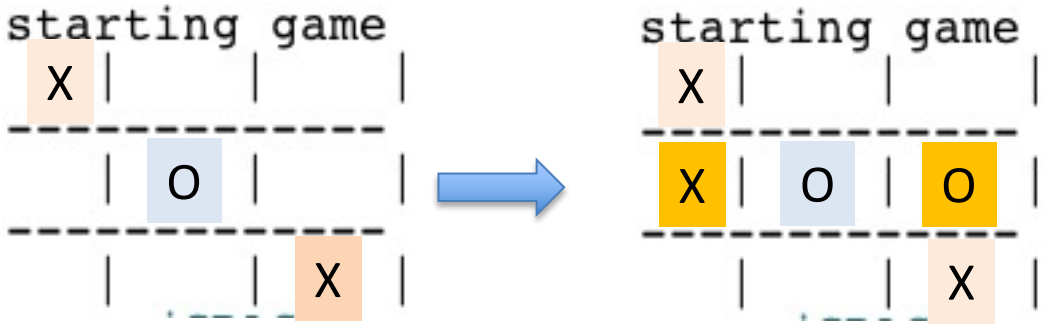
X best strategy and only winning one:
3 corners



X wins!



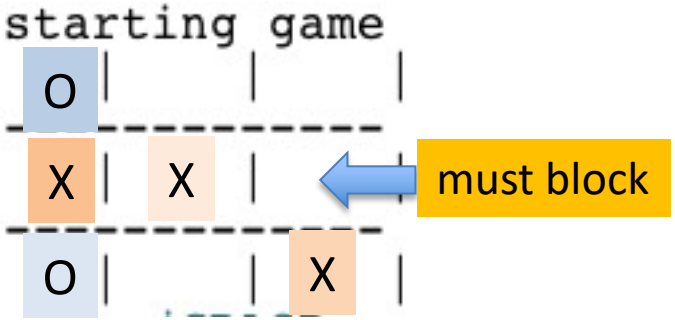
cats game



O must NOT take a corner!

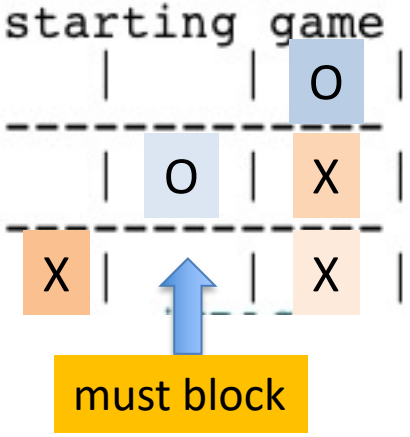
Strategy: 3rd Move

- O can NOT win
  - unless X doesn't block
- O goal: cats game



O Strategy:

1. check for **win** opp
2. else check for **block**
3. else pick best square
4. 4th move: only 2 choices




```
select(move#) {
case 1:  //best square
case 2:  //block
case 3:  //win or block
case 4:  //win or block
(note:  O only gets 4 squares
```

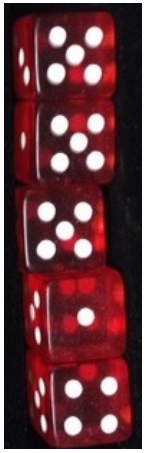
Lab

- Project 2

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

DR JEFF SOFTWARE INDIE APP DEVELOPER

Project 2: Play Cards



- Required Functions
  - (Re-)Shuffle + display deck
  - Ask Player (1) choice
    - Blackjack
    - Poker (pick 1)
      - **5-card stud [**Texas Hold 'em (2+5=7 cards)]
    - War (1 card only, bonus for "war")
    - Bridge (entire deck)
  - Deal hands
    - Player + Dealer
    - Display/print hands
  - [Play hand/game] -- Bonus
    - Blackjack
      - deal cards until "stay" or "bust"
      - *Dealer* must hit <17, stay >=17
      - Determine winner or "push" (draw)
  - Bonus: determine winners

- Deal 1 hand of each
- Bonus: ***Play*** 1 hand of **Blackjack**

- Use GUI Option Dialogs:
  1 Blackjack
  2 Poker
  3 War
  4 Bridge
- Use *arrays*
  - Deck[52]
  - Suits[4]
  - Rank[13]
- Create **Classes**
  - ***Card Fns***
  - ***Games***

- Extras
  - **UML**
  - User Guide

Class Structure

Project 2

Proj2 Main Class

MAIN method

select game

```
P2CardGm NewGame=
new P2CardGm();

//Ask play again?
```

P2TicTT Class

- Data Fields
- Constructor
- ***play*** method — Driver
- ***randMove*** method
- ***compMove*** method
- ***playerMove*** method
- ***prtBoard*** method
- ***score*** method

Optional →

P2CardFns Class

- Data Fields
- Constructor
- ***newDeck/shuffleDeck*** method
- ***displayDeck*** method
- ***displayHand*** method
- ***dealHands*** method
- ***cards*** method

Game Sub-classes

Project 2

Super Class

P2CardGm Class

- Data Fields
- Constructor
- ***playGame*** method
  - Shuffle deck
  - Display deck
  - Deal hands
  - Display hands

P2BJack Class

Extends P2CardGm

- Data Fields
- Constructor
- ***play*** method
- ***score*** method
- ***bet*** method

Optional

P2Poker Class

Extends P2CardGm

- Data Fields
- Constructor
- ***play*** method
- ***score*** method
- ***bet*** method

Other Classes

Project 2

Super Class **P2CardGm Class**

- **Data Fields**
- **Constructor**
- ***playGame*** method
  - Shuffle deck
  - Display deck
  - Deal hands
  - Display hands

Alternative Structure

P2CardGames Class

- **Data Fields**
- **Constructor**
- ***playGame*** method — Driver
- ***blackjack*** method
- ***poker*** method
- ***bridge*** method
- ***war*** method

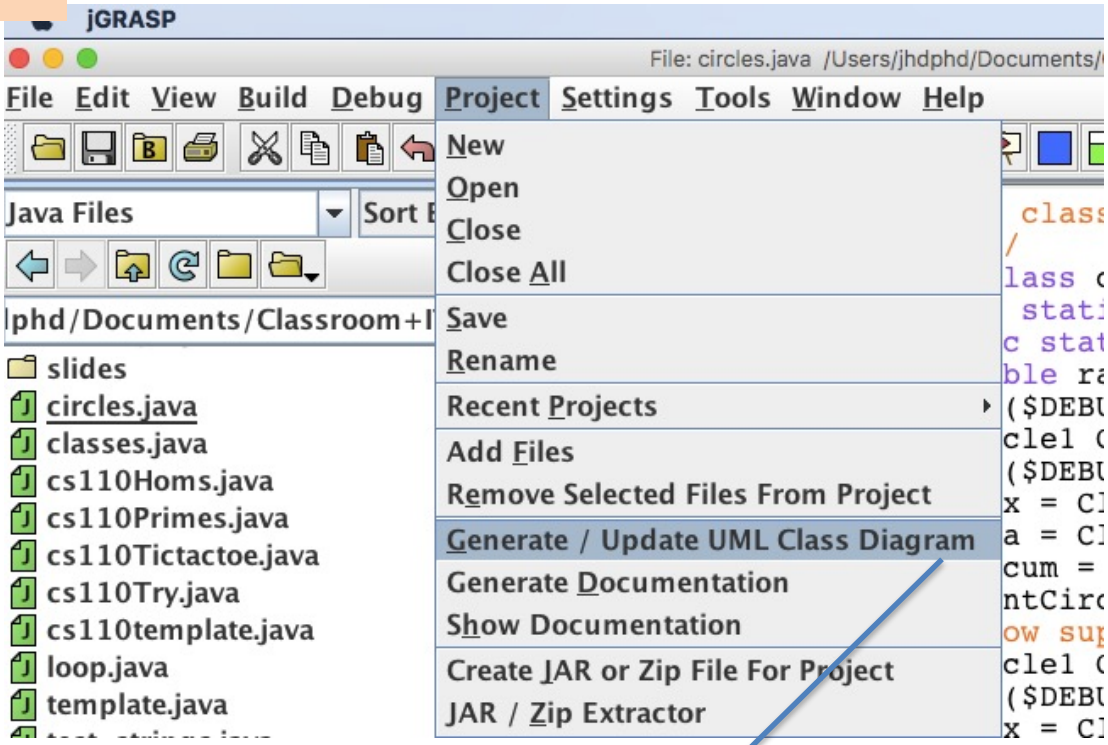
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

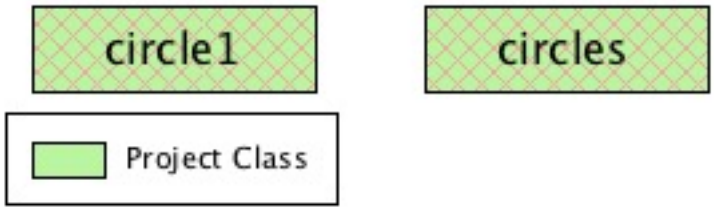
COMP110L

DR JEFF SOFTWARE
INDIE APP DEVELOPER

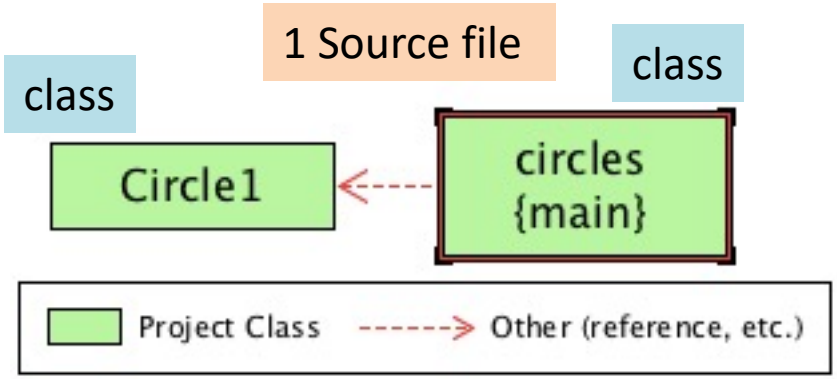
Circle UML in jGRASP

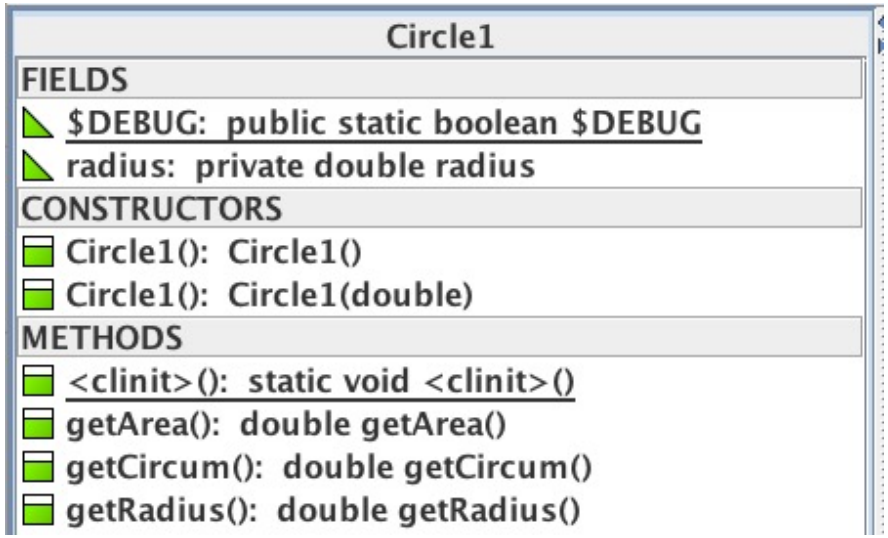
Ch 9

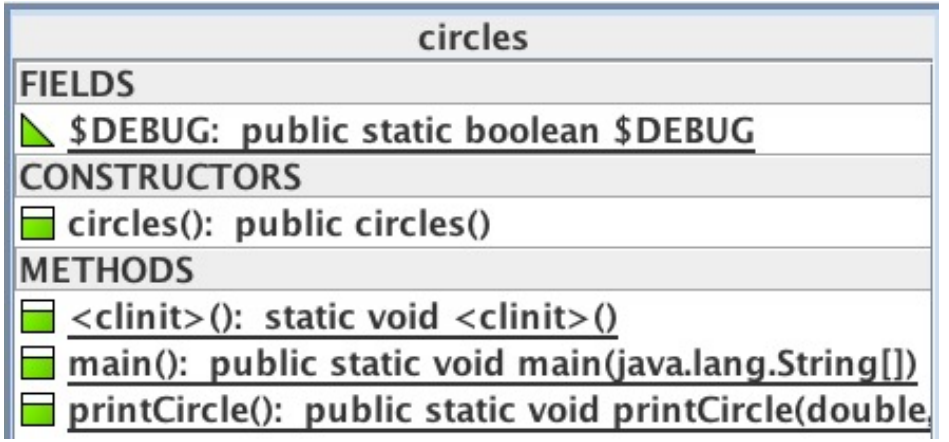




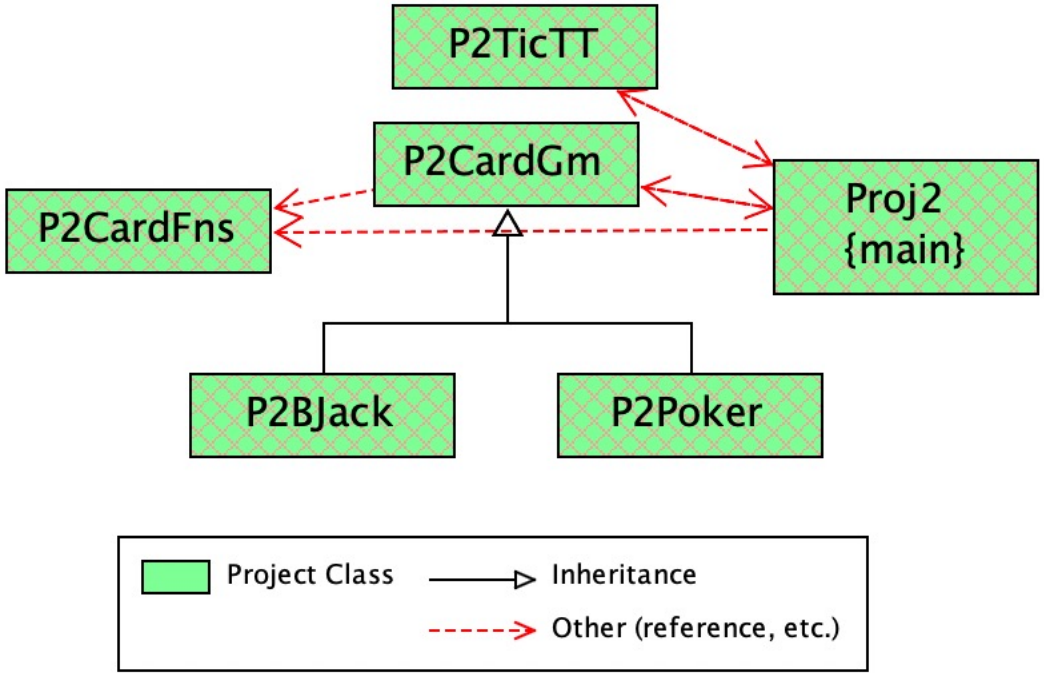
Circles UML in jGRASP

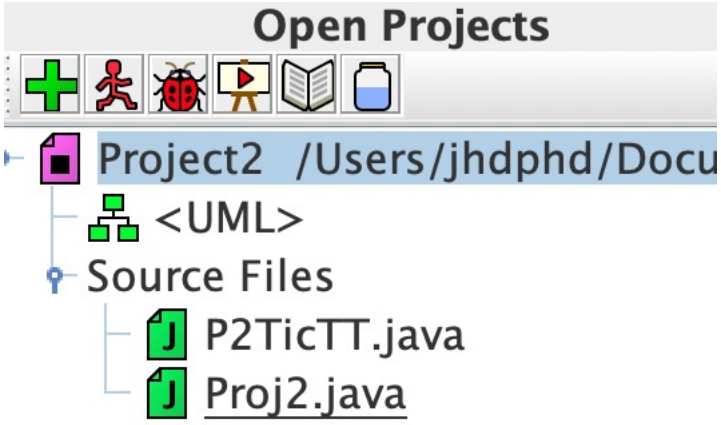






Project 2 UML





Project 2: Deck of Cards

Project 2

❖ Setup

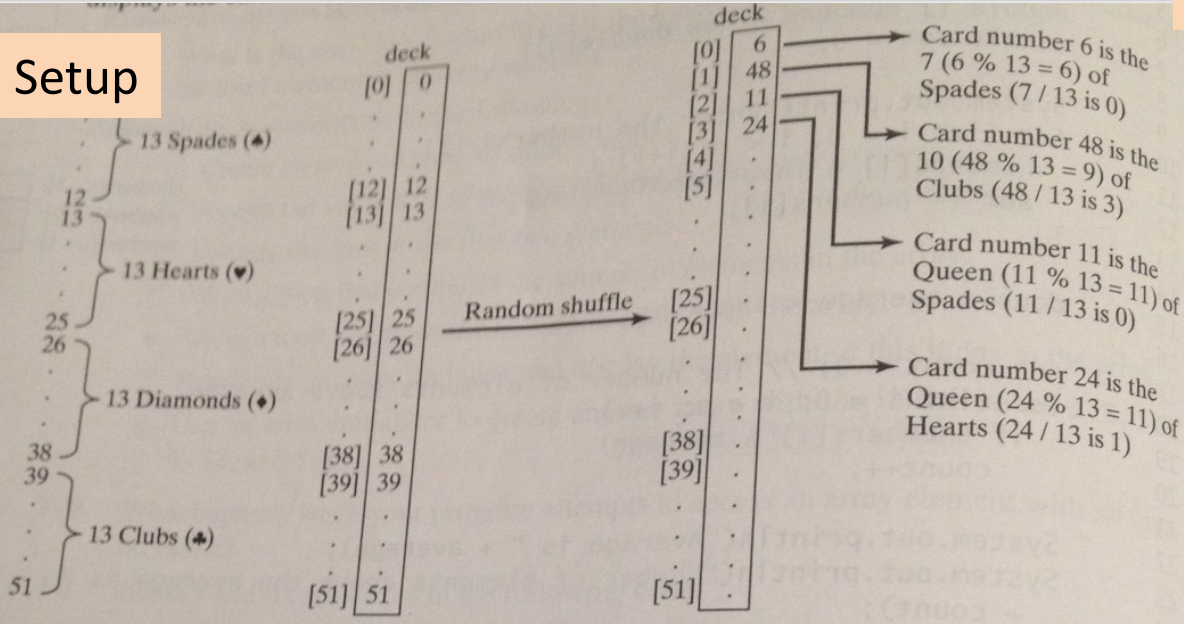


FIGURE 7.2 52 cards are stored in an array named **deck**.

❖ suit: /13

❖ rank: %13

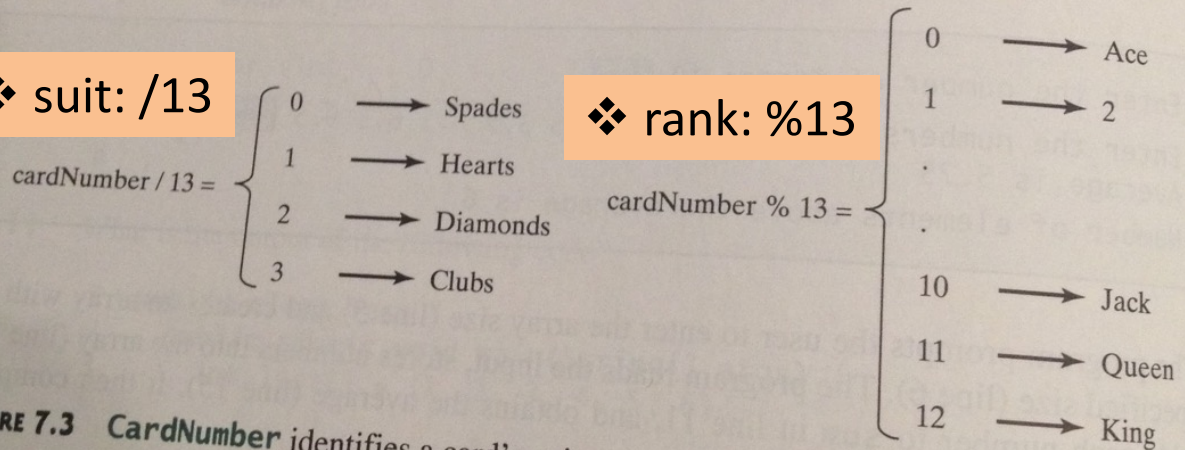


FIGURE 7.3 **CardNumber** identifies a card's suit and rank number.

Project 2: Deck of Cards

Project 2

LISTING 7.2

```
public class DeckOfCards {
  public static void main(String[] args) {
    int[] deck = new int[52];
    String[] suits = {"Spades", "Hearts", "Diamonds", "Clubs"};
    String[] ranks = {"Ace", "2", "3", "4", "5", "6", "7", "8",
      "10", "Jack", "Queen", "King"};

    // Initialize the cards
    for (int i = 0; i < deck.length; i++)
      deck[i] = i;

    // Shuffle the cards
    for (int i = 0; i < deck.length; i++) {
      // Generate an index randomly
      int index = (int)(Math.random() * deck.length);
      int temp = deck[i];
      deck[i] = deck[index];
      deck[index] = temp;
    }

    // Display the first four cards
    for (int i = 0; i < 4; i++) {
      String suit = suits[deck[i] / 13];
      String rank = ranks[deck[i] % 13];
      System.out.println("Card number " + deck[i] + ": "
        + rank + " of " + suit);
  }
}
```

❖ Setup

❖ Shuffle

❖ Display

pp. 254-5

Displaying the Deck

Project 2

```
new deck of cards (unshuffled)
♠A ♠2 ♠3 ♠4 ♠5 ♠6 ♠7 ♠8 ♠9 ♠10 ♠J ♠Q ♠K
♥A ♥2 ♥3 ♥4 ♥5 ♥6 ♥7 ♥8 ♥9 ♥10 ♥J ♥Q ♥K
♦A ♦2 ♦3 ♦4 ♦5 ♦6 ♦7 ♦8 ♦9 ♦10 ♦J ♦Q ♦K
♣A ♣2 ♣3 ♣4 ♣5 ♣6 ♣7 ♣8 ♣9 ♣10 ♣J ♣Q ♣K
instantiating a new game #1
♠2 ♠7 ♥4 ♣3 ♦K ♦2 ♠K ♦J ♠5 ♥A ♥3 ♣2 ♣4
♦3 ♥J ♣6 ♥10 ♠3 ♣A ♥2 ♦9 ♣8 ♥8 ♠4 ♥7 ♣10
♦4 ♣5 ♠6 ♦8 ♥5 ♠8 ♦A ♣Q ♣7 ♦5 ♣9 ♥6 ♠10
♥K ♦10 ♣K ♦7 ♦Q ♠A ♣J ♠Q ♥9 ♠J ♠9 ♦6 ♥Q
```

- Shuffled

Project 2: Deal Hands

Project 2

❖ Blackjack

Spades	A
Hearts	
Diams	5
Clubs	4

Total	19

❖ Poker

Spades	A
Hearts	Q 8
Diams	9
Clubs	4

❖ Hold 'em

❖ Use methods
- ❑ dealHands
- ❑ displayHand

➢ Bonus

or use Unicode symbols

U+2660	U+2665	U+2666	U+2663
♠	♥	♦	♣

Display Hands

Project 2

```
instantiating a new game #1
♠3 ♦5 ♥8 ♣6 ♠2 ♥Q ♠4 ♠J ♠7 ♥9 ♠5 ♣9 ♦2
♦9 ♠A ♦6 ♦3 ♣A ♥10 ♥J ♦Q ♦4 ♦8 ♠6 ♦7 ♥4
♥6 ♥3 ♣5 ♣4 ♣3 ♥7 ♠9 ♦K ♣K ♥K ♦10 ♦A ♥2
♠8 ♠K ♠10 ♣8 ♣10 ♣7 ♦J ♣J ♣Q ♥5 ♣2 ♠Q ♥A
starting a new game #1
Playing Poker...
player:
♠3 ♥8 ♠2 ♠4 ♠7
dealer:
♦5 ♣6 ♥Q ♠J ♥9
```

❖ Deck

❖ Poker Hands

```
instantiating a new game #1
♦3 ♠K ♥5 ♥K ♣5 ♠9 ♣8 ♣K ♠6 ♣Q ♠3 ♦2 ♦Q
♥9 ♠7 ♠A ♠2 ♦K ♣2 ♦10 ♦A ♠4 ♥3 ♥8 ♣J ♥7
♦8 ♥6 ♥10 ♣7 ♣3 ♥2 ♠10 ♥4 ♣6 ♦4 ♦7 ♠8 ♣4
♥Q ♠J ♦9 ♥J ♠Q ♣A ♦5 ♦6 ♥A ♣10 ♣9 ♦J ♠5
starting a new game #1
Playing Bridge...
♦3 ♣5 ♠6 ♦Q ♠2 ♦A ♣J ♥10 ♠10 ♦7 ♠J ♣A ♣10
♠K ♠9 ♣Q ♥9 ♦K ♠4 ♥7 ♣7 ♥4 ♠8 ♦9 ♦5 ♣9
♥5 ♣8 ♠3 ♠7 ♣2 ♥3 ♦8 ♣3 ♣6 ♣4 ♥J ♦6 ♦J
♥K ♣K ♦2 ♠A ♦10 ♥8 ♥6 ♥2 ♦4 ♥Q ♠Q ♥A ♠5
```

❖ Deck

❖ Bridge Hands

Main Class: Proj2


```
4  file:  Proj2.java
5  */
6  import java.util.*;
7  import javax.swing.*;
8  //class
9  public class Proj2 {
10    //data fields (static)
11    static final boolean $DEBUG = true;
12    static int gameNum = 0;
13    static String msg = "", msgNP = "Now playing: ";
14    static final String endLines = "==========";
15 //main
16    public static void main(String[] args) {
17       String[] mainOpts = {"TicTacToe", "Cards", "Quit"};
18       String[] cardOpts = {"Blackjack", "Poker", "War", "Bridge"};
19       String cardGmName = "none";
20       int selGame = 0;
21       P2CardFns.newDeck();//new pack of cards
22       System.out.println("new deck of cards (unshuffled)");
23       P2CardFns.displayDeck();
24       while(true) {//main loop
25          gameNum++;
26          int selMain = JOptionPane.showOptionDialog(null, "Select game",
27             "options", 0, 1, null, mainOpts, mainOpts[0]);
28          msg = mainOpts[selMain];
29          switch(selMain){
```

Main Class: Proj2

Project 2

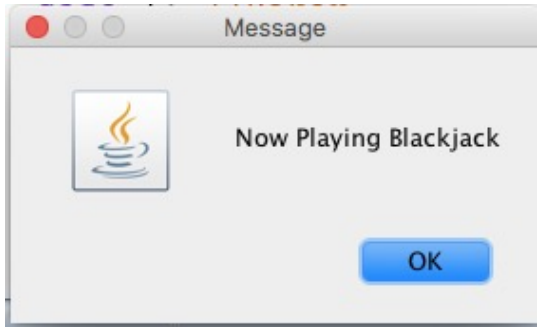
```
24        while(true) {//main loop
25           gameNum++;
26           int selMain = JOptionPane.showOptionDialog(null, "Select game",
27              "options", 0, 1, null, mainOpts, mainOpts[0]);
28           msg = mainOpts[selMain];
29           switch(selMain){
30              case 0: //TicTacToe
31                 printMsg(msg);
32                 //instan TicTacToe
33                 P2TicTT TTTgm = new P2TicTT();
34                 TTTgm.selectGm();
35                 System.out.println(endLines);
36                 continue;//--> top of main loop
37              case 1: //Cards
38                 printMsg(msg);
39                 selGame = JOptionPane.showOptionDialog(null, "Select card game:",
40                    "games", 0, 1, null, cardOpts, cardOpts[0]);
41                 if (selGame<0 || selGame>3) {
42                    cardGmName="invalid";
43                    continue;}
44                 break;
45              default://Quit or Cancel
46                 printMsg("Player Quit: Good-bye!");
47                 System.exit(0);//stop
48           }//end switch
```

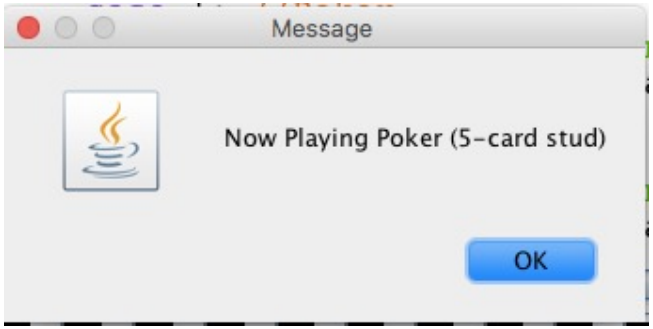
Main Class: Proj2

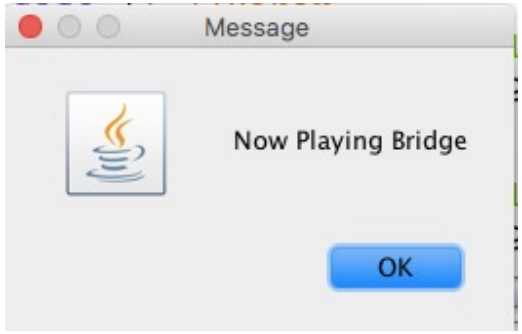

```
49            //continue to play new selected card game
50            cardGmName = cardOpts[selGame];
51            msg = cardGmName;
52            P2CardGm NewGame = new P2CardGm(cardGmName);
53            NewGame.playGame(cardGmName);
54            System.out.println(endLines);
55         }//end main loop
56      } //end main method
57
58      //print utility
59      static void printMsg(String xmsg) {
60         String msg = msgNP + xmsg;
61         System.out.println(msg);
62         JOptionPane.showMessageDialog(null, msg);
63      }//end printMsg
64   }//end main class
```

Main Dialogs

Project 2








```
starting a new game #1
Playing Blackjack...
starting a new game #2
Playing Poker...
starting a new game #3
Playing War...
starting a new game #4
Player Quit
```

Class: Cards

Project 2

```
58 class P2Cards {
59    public static boolean $DEBUG = true;
60    public static final int size = 52;
61    public static final int handsMax = 4;
62    public static final int lenMax = size/handsMax;
63    public static int[] deck = new int[size];
64    public static int[][] hands = new int[handsMax][lenMax];
65 //public static char[] suits = {'S', 'H', 'D', 'C'};
66    public static final String[] suits = {"\u2660", "\u2665", "\u2666", "\u2663"};
67    public static final String[] faces = {"A", "J", "Q", "K"};
68 //constructor
69    P2Cards() {
70       if ($DEBUG) System.out.println("Instantiating Cards...");
71    }
72 //methods
73    static void newDeck() {
74       for (int i=0; i<size; i++) {
75          deck[i] = i;} //init ranks
76    }//end newDeck
77    static void shuffle(){
78       for(int i=0; i<size; i++) { //all 52 cards
79       //generate a random index to swap with
80          int cardNum = (int)(Math.random() * size); //num (0..51)
81          int temp = deck[i];
82          deck[i] = deck[cardNum];
83          deck[cardNum] = temp; //swap
84       }//end for
```

- methods
  - newDeck
  - shuffle

Class: Cards(cont)

Project 2

```
80 static void displayDeck() {
81    for(int i=0; i<4; i++) {
82    //print a row of 13 cards (4x)
83    String dispStr = "";
84       for(int j=0; j<13; j++) {
85          dispStr += card(deck[i*13 + j]) + " ";
86       }//end for j
87       System.out.println(dispStr);
88    }//end for i
89 }//end displayDeck
90 static String card(int n) {//convert # to card string
91    if(n<0 || n>51) {
92       System.out.println("error: bad card number=" +n);
93       return "?";}
94    int suit = n/13;
95    int rank = n%13 +1;//note +1
96    String rnk = rank +"";
97    if(rank==1) rnk = faces[0];
98       else if(rank>10) rnk = faces[rank-10];
99    return suits[suit] + rnk;
.00 }//end card
.01 }//end class Cards
```

- methods
  - displayDeck
  - card

Class: Cards(cont)

Project 2

```
91 static void displayHand(int[] hand, int size) {
92    String dispStr = "";
93    for(int i=0; i<size; i++) {
94       dispStr += card(hand[i]) + " ";
95       }//end for i
96    System.out.println(dispStr);
97 }//end displayHand
```

```
109    static int[][] dealHands(int numHands,int lenHands) {
110       int cardNum = 0;
111       int[][] hand = new int[numHands][lenHands];
112       for (int i=0; i<lenHands; i++) {//alt hands
113          for (int j=0; j<numHands; j++) {
114             hand[j][i] = deck[cardNum++];//deal card
115          }}//end loops
116          return hand;
117    }//end dealHands
```

- methods
  - dealHands
  - displayHand

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Unicode Symbols

Project 2

Playing cards in Unicode

U+2660	U+2665	U+2666	U+2663
♠	♥	♦	♣
Black Spade Suit	Black Heart Suit	Black Diamond Suit	Black Club Suit
&spades;	&hearts;	&diams;	&clubs;
U+2664	**U+2661**	**U+2662**	**U+2667**
♤	♡	♢	♧
White Spade Suit	White Heart Suit	White Diamond Suit	White Club Suit

Playing Cards[1][2]

Official Unicode Consortium code chart (PDF)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1F0Ax		🂡	🂢	🂣	🂤	🂥	🂦	🂧	🂨	🂩	🂪	🂫	🂬	🂭	🂮	
U+1F0Bx		🂱	🂲	🂳	🂴	🂵	🂶	🂷	🂸	🂹	🂺	🂻	🂼	🂽	🂾	
U+1F0Cx		🃁	🃂	🃃	🃄	🃅	🃆	🃇	🃈	🃉	🃊	🃋	🃌	🃍	🃎	🃏
U+1F0Dx		🃑	🃒	🃓	🃔	🃕	🃖	🃗	🃘	🃙	🃚	🃛	🃜	🃝	🃞	🃟

Class P2CardGm


```
140 class P2CardGm {
141    //data fields (non-static=instance)
142    public static boolean $DEBUG = true;
143    //get data fields from P2Cards
144    int size = P2CardFns.size;
145    int[] deck = P2CardFns.deck;
146    int[][] hands = P2CardFns.hands;
147    int numHands, lenHands, len;//len for var length hands
148    String[] plNames = {"Player", "Dealer"};
149    String[] bridgeNames = {"West", "North", "East", "South"};
150    String msg = "";
151 //constructors
152    P2CardGm() { }//empty
153    P2CardGm(String name) {
154       msg = " game #" +Proj2.gameNum +"=  " +name;
155       Proj2.printMsg(msg);
156    }//end constructor
```

- *Use* methods
  - Shuffle
  - displayDeck
  - dealHands
  - displayHand

P2CardGm.playGame


```
157 //game methods
158     void playGame(String game) {
159         switch(game){
160             case "Blackjack":
161                 numHands = 2; lenHands = 2; len = 2;
162                 break;
163             case "Poker":
164                 numHands = 2; lenHands = 5;
165                 break;
166             case "War":
167                 numHands = 2; lenHands = 1;
168                 break;
```

Blackjack

Project 2

```
153    void blackjack() {
154        int numHands = 2, lenHands = 2, len = 2;
155        String[] plNames = {"Player", "Dealer"};
156        //deal 2 cards each & display
157        playGame(numHands, lenHands, plNames);
158        //**continue to play blackjack
```

- Replace set of methods with a simple “Case” construct

Poker, War, Bridge

Project 2

```
226    void poker() {
227        int numHands = 2, lenHands = 5;
228        String[] plNames = {"Player", "Dealer"};
229        //deal hands & display them
230        playGame(numHands, lenHands, plNames);
231    } //end poker
232
233    void war() {
234        int numHands = 2, lenHands = 1;
235        String[] plNames = {"Player", "Dealer"};
236        //deal hands & display them
237        playGame(numHands, lenHands, plNames);
238    }//end war
239
240    void bridge() {
241        int numHands = 4, lenHands = 13;
242        String [] plNames = {" West", "North", " East", "South"};
243        //deal 4 hands & display them
244        playGame(numHands, lenHands, plNames);
245    }//end bridge
```

- Replace set of methods with a simple “Case” construct

Strategy

Game Playing

Computer Game Playing

- Games
  - Chess
  - Go
  - Bridge
- Algorithms
  - Chess
  - Go
  - Bridge
- Complexity
  - Hard
  - NP Complete
- Strategies
  - Beginning/opening
  - Middle
  - End

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

Bridge

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Game Playing

Computer bridge

From Wikipedia, the free encyclopedia

https://en.wikipedia.org/wiki/Computer_bridge#Computers_versus_humans

Computer bridge is the playing of the game contract bridge using computer software. After years of limited progress, since around the end of the 20th century the field of computer bridge has made major advances. In 1996 the American Contract Bridge League (ACBL) established an official World Computer-Bridge Championship, to be held annually along with a major bridge event. The first championship took place in 1997 at the North American Bridge Championships in Albuquerque. Since 1999 the event has been conducted as a joint activity of the American Contract Bridge League and the World Bridge Federation. Alvin Levy, ACBL Board member, initiated this championship and has coordinated the event annually since its inception. The event history, articles and publications, analysis, and playing records can be found at the official website.

World Computer-Bridge Championship

The World Computer-Bridge Championship is typically played event are:

- 1997 *Bridge Baron*
- 1998 *GIB*
- 1999 *GIB*
- 2000 *Meadowlark Bridge*
- 2001 *Jack*
- 2002 *Jack*
- 2003 *Jack*
- 2004 *Jack*
- 2005 *Wbridge5*
- 2006 *Jack*
- 2007 *Wbridge5*
- 2008 *Wbridge5*
- 2009 *Jack*
- 2010 *Jack*
- 2011 *Shark Bridge*
- 2012 *Jack*
- 2013 *Jack*
- 2014 *Shark Bridge*
- 2015 *Jack*
- 2016 *Wbridge5*[3]
- 2017 *Wbridge5*[4]

Bridge

Game Playing

World Computer-Bridge Championship

The World Computer-Bridge Championship is typically played event are:

- 1997 *Bridge Baron*
- 1998 *GIB*
- 1999 *GIB*
- 2000 *Meadowlark Bridge*
- 2001 *Jack*
- 2002 *Jack*
- 2003 *Jack*
- 2004 *Jack*
- 2005 *Wbridge5*
- 2006 *Jack*
- 2007 *Wbridge5*
- 2008 *Wbridge5*
- 2009 *Jack*
- 2010 *Jack*
- 2011 *Shark Bridge*
- 2012 *Jack*
- 2013 *Jack*
- 2014 *Shark Bridge*
- 2015 *Jack*
- 2016 *Wbridge5*[3]
- 2017 *Wbridge5*[4]

Bridge

Game Playing

Computers versus humans [edit]

In Zia Mahmood's book, *Bridge, My Way* (1992), Zia offered a £1 million bet that no four-person team of his choosing would be beaten by a computer. A few years later the bridge program *GIB*, brainchild of American computer scientist Matthew Ginsberg,[5] proved capable of expert declarer plays like winkle squeezes in play tests. In 1996, Zia withdrew his bet. Two years later, *GIB* became the world champion in computer bridge, and also had a 12th place score (11210) in declarer play compared to 34 of the top humans in the 1998 Par Contest (including Zia Mahmood).[6] However, such a par contest measures technical bridge analysis skills only[*clarification needed*], and in 1999 Zia beat various computer programs, including *GIB*, in an individual round robin match.[7]

Further progress in the field of computer bridge has resulted in stronger bridge playing programs, including *Jack* [8] and *Wbridge5*.[9] These programs have been ranked highly in national bridge rankings. A series of articles published in 2005 and 2006 in the Dutch bridge magazine *IMP* describes matches between five-time computer bridge world champion *Jack* and seven top Dutch pairs including a Bermuda Bowl winner and two reigning European champions. A total of 196 boards were played. *Jack* defeated three out of the seven pairs (including the European champions). Overall, the program lost by a small margin (359 versus 385 IMPs).

Cardplay algorithms [edit]

Bridge poses challenges to its players that are different from board games such as chess and go. Most notably, bridge is a stochastic game of incomplete information. At the start of a deal, the information available to each player is limited to just his/her own cards. During the bidding and the subsequent play, more information becomes available via the bidding of the other three players at the table, the cards of the partner of the declarer (the dummy) being put open on the table, and the cards played at each trick. However, it is only at the end of the play that full information is obtained.

Today's top-level bridge programs deal with this probabilistic nature by generating many samples representing the unknown hands. Each sample is generated at random, but constrained to be compatible with all information available so far from the bidding and the play. Next, the result of different lines of play are tested against optimal defense for each sample. This testing is done using a so-called "double-dummy solver" that uses extensive search algorithms to determine the optimum line of play for both parties. The line of play that generates the best score averaged over all samples is selected as the optimal play.

Efficient double-dummy solvers are key to successful bridge-playing programs. Also, as the amount of computation increases with sample size, techniques such as importance sampling are used to generate sets of samples that are of minimum size but still representative.

Bridge

Game Playing

Comparison to other strategy games [edit]

While bridge is a game of incomplete information, a double-dummy solver analyses a simplified version of the game where there is perfect information; the bidding is ignored, the contract (trump suit and declarer) is given, and all players are assumed to know all cards from the very start. The solver can therefore use many of the game tree search techniques typically used in solving two-player perfect-information win/lose/draw games such as chess, go and reversi. However, there are some significant differences.

- Although double-dummy bridge is in practice a competition between two generalised players, each "player" controls two hands and the cards must be played in a correct order that reflects four players. (It makes a difference which of the four hands wins a trick and must lead the next trick.)
- Double-dummy bridge is not simply win/lose/draw and not exactly zero-sum, but constant-sum since two playing sides compete for 13 tricks. It is trivial to transform a constant-sum game into a zero-sum game. Moreover, the goal (and the risk management strategy) in general contract bridge depends not only on the contract but also on the form of tournament. However, since the double-dummy version is deterministic, the goal is simple: one can without loss of generality aim to maximize the number of tricks taken.
- Bridge is incrementally scoring; each played trick contributes irreversibly to the final "score" in terms of tricks won or lost. This is in contrast to games where the final outcome is more or less open until the game ends. In bridge, the already determined tricks provide natural lower and upper bounds for alpha-beta pruning, and the interval shrinks naturally as the search goes deeper. Other games typically need an artificial evaluation function to enable alpha-beta pruning at limited depth, or must search to a leaf node before pruning is possible.
- It is relatively inexpensive to compute "sure winners" in various positions in a double-dummy solver. This information improves the pruning. It can be regarded as a kind of evaluation function, however while the latter in other games is an approximation of the value of the position, the former is a definitive lower bound on the value of the position.
- During the course of double-dummy game tree search, one can establish equivalence classes consisting of cards with apparently equal value in a particular position. Only one card from each equivalence class needs to be considered in the subtree search, and furthermore, when using a transposition table, equivalence classes can be exploited to improve the hit rate. This has been described as *partition search* by Matthew Ginsberg.
- Numerous strategy games have been proven hard in a complexity class, meaning that any problem in that complexity class can be reduced in polynomial time to that problem. For example, generalized $x \times x$ chess has been proven EXPSPACE-complete (both in EXPSPACE and EXPSPACE-hard), effectively meaning that it is among the hardest problems in EXPSPACE. However, since there is no natural structure to exploit in double-dummy bridge towards a hardness proof or disproof, unlike in a board game, the question of hardness remains.[10]

Unsolved problem in computer science:

? *Is the problem of deciding the winner in double-dummy bridge hard in any complexity class?*

(more unsolved problems in computer science)

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Lab 9: Bowling League

DR JEFF SOFTWARE INDIE APP DEVELOPER

- *Arrays*
- *Files*
- *Stats*
- *Classes*

- GUI

Rqts– INPUT:
1) *text file*
2) Selection

OUTPUT:
1) Selected stats

- GUI or console

- Create **Class**
  - *Team*

INPUT:
1- Text file
2- Select option
- Bowler
- Team
- League
- Quit

PROCESS (source code)
1) Input Data
2) Give menu of options
3) Compute stats
4) Print formatted stats

OUTPUT:
1- Bowler data
2- Team data
3- League data
4- Date

DEBUGGING/TESTING

Create Data File:
- ❑ 4-man teams (first, last names
- ❑ 6 teams (4x6=24 bowlers)
- ❑ all scores (3 scores per bowler)

Submit file

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Bowling League Data

DR JEFF SOFTWARE
INDIE APP DEVELOPER

CSUN Bowling League
Stats for week 1: Nov 9, 2016
======

Team 1 Matador
Bob Smith 145 153 132
Jim Jones 114 136 152
Mary Valdez 110 117 121
Sonia Gomez 98 132 114
******

Team 2 Toro
Bob Smith 145 153 132
Jim Jones 114 136 152
Mary Valdez 110 117 121
Sonia Gomez 98 132 114
******

Team 3 Picador
Bob Smith 145 153 132
Jim Jones 114 136 152
Mary Valdez 110 117 121
Sonia Gomez 98 132 114
******

Team 4 Burrito
Bob Smith 145 153 132
Jim Jones 114 136 152
Mary Valdez 110 117 121
Sonia Gomez 98 132 114
******

Team 5 Duenas
Bob Smith 145 153 132
Jim Jones 114 136 152
Mary Valdez 110 117 121
Sonia Gomez 98 132 114
******

Team 6 Vaqueros
Bob Smith 145 153 132
Jim Jones 114 136 152
Mary Valdez 110 117 121
Sonia Gomez 98 132 114
******

##EOF

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Bowling League Output

DR JEFF SOFTWARE
INDIE APP DEVELOPER

```
CSUN  Bowling League
Stats for week 1:  Nov 9, 2016
======
Team 1 Matador
Bob Smith 145 153 132 = 430
```

❖ Bowler

OUTPUT:
1- Bowler data
2- Team data
3- League data
4- Date

```
CSUN  Bowling League
Stats for week 1:  Nov 9, 2016
======
Team 1 Matador
Bob Smith 145 153 132 = 430
Jim Jones  114 136 152 = 402
Mary Valdez  110 117 121 = 348
Sonia Gomez  98 132 114 = 344
------------
Total pins = 1524
```

❖ Team

```
CSUN  Bowling League
Stats for week 1:  Nov 9, 2016
======
High Score
Bob Smith 212
High Series
Jim Jones 520
High Team
Team 1 Matador 1524
```

❖ League

Bowling League Arrays


```
int teams = 6;
int bowlers = 4;
int games = 3;

String [ ] [ ]  names = new String [teams] [bowlers];
int [ ] [ ] [ ] scores = new int [teams] [bowlers] [games];
```

example:

```
names[0] [i] = {“Bob Smith”, “Jim Jones”, “Mary Valdez”, “Sonia Gomez”};

scores[0] [i] [j] = {145, 132, 153}
```

- Read data from text file

Bowling League Classes

- Bowler
- Team
  - Extends Bowler

Lab

Color
&
GUI Design

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Icons – Colors

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Google Guidelines

Cool	Tint	Shade / Shadow
Base Color (50–900)	White	Indigo 900
Blue	20%	20%
Indigo	20%	20%
Deep Purple	20%	20%
Purple	20%	20%

Warm	Tint	Shade / Shadow
Base Color (50–900)	White	Deep Orange 900
Yellow	20%	20%
Amber	20%	20%
Orange	20%	20%

Fresh	Tint	Shade / Shadow
Base Color (50–900)	White	Blue Grey 900
Blue Grey	20%	20%
Light Blue	20%	20%
Cyan	20%	20%
Teal	20%	20%
Green	20%	20%
Light Green	20%	20%
Lime	20%	20%

Hot	Tint	Shade / Shadow
Base Color (50–900)	White	Brown 900
Deep Orange	20%	20%
Red	20%	20%
Pink	20%	20%
Brown	20%	20%

Neutral	Tint	Shade / Shadow
Base Color	White	Grey 900
Grey 50	40%	10%
Grey 100	40%	10%
Grey 200	40%	10%
Grey 300	40%	10%
Grey 400	20%	20%
Grey 500	20%	20%
Grey 600	20%	20%
Grey 700	20%	20%
Grey 800	20%	20%

Color (step 12)

Color Uses

- Use color to assist in formatting a screen
  - — Relating or tying elements into groupings.
  - — Breaking apart separate groupings of information.
  - — Associating information that is widely separated on the screen.
  - — Highlighting or calling attention to important information by setting it off from the other information.
- Use color as a visual code to identify
  - — Screen components.
  - — The logical structure of ideas, processes, or sequences.
  - — Sources of information.
  - — Status of information.
- Use color to
  - — Realistically portray natural objects.
  - — Increase screen appeal.

Color (step 12)

Table 12.3: Common Color Connotations

COLOR	POSITIVE	NEGATIVE
Red	Active Attractive Dominating Exciting Invigorating Powerful Strong	Aggressive Alarming Energetic
Blue	Abstinent Controlled Deep Dreamy Faithful Harmonious Intellectual Mysterious Pornography Rational Sensible Tenderness	Aggressive Cold Introverted Melancholic
Blue-green or Turquoise	Refreshing	Aloof Cold Self-willed Sterile Unemotional
Green	Calm Close to nature Conciliatory Gentle Harmonious Optimistic Refreshing Strong willed	Envious Inexperienced Jealous
Yellow	Cheerful Colorful Extroverted Full of fun Light Lively Youthful	Cowardly Exaggerated Superficial Vain
Orange	Alive Communicative Direct Exciting Joyful Warm	Cheap Intimate Possessive Vigorous
Purple	Luxurious Royal Serious	Sad

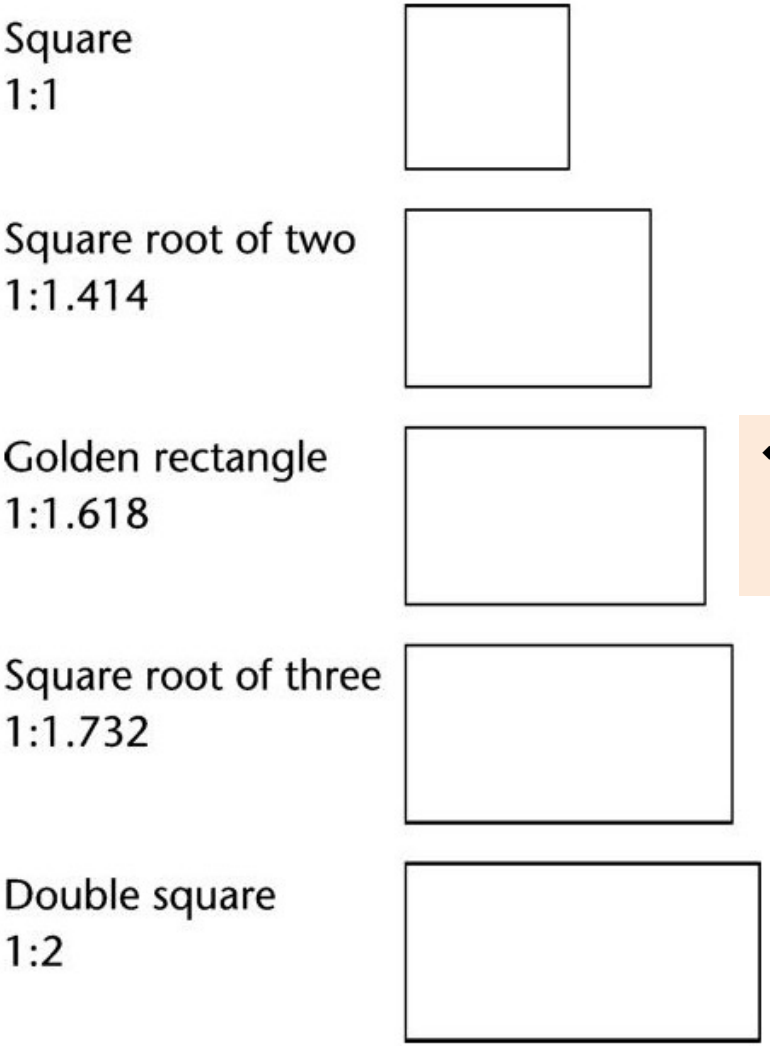
Derived from Gotz (1998) and Stone et al. (2005).

Step 3 (CONT)

STEP 3 Understand the Principles of Good Interface and Screen Design

Pages 133-146

Figure 3.8: Pleasing proportions.



- Golden ratio
  - Fibonacci sequence
    - 1, 1, 2, 3, 5, 8, 13, 21

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110L

Golden Ratio

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Golden ratio

From Wikipedia, the free encyclopedia

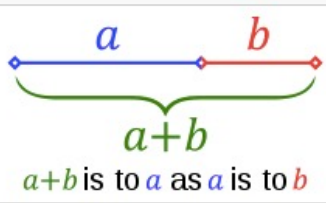
This article is about the number. It is not to be confused with the pop music album or the calendar dates.

In mathematics, two quantities are in the **golden ratio** if their ratio is the same as the ratio of their sum to the larger of the two quantities. The figure on the right illustrates the geometric relationship. Expressed algebraically, for quantities a and b with $a > b > 0$,

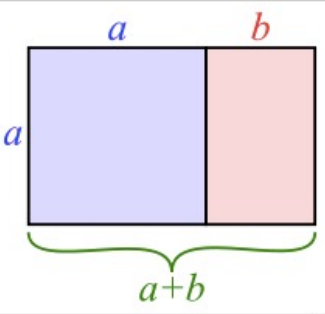
$$\frac{a+b}{a} = \frac{a}{b} \stackrel{\text{def}}{=} \varphi,$$

where the Greek letter phi (φ or ϕ) represents the golden ratio. Its value is:

$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots.$$ OEIS A001622



Line segments in the golden ratio



A golden rectangle (in pink) with longer side a and shorter side b, when placed adjacent to a square with sides of length a, will produce a similar golden rectangle with longer side $a + b$ and shorter side a. This illustrates the relationship $\frac{a+b}{a} = \frac{a}{b} \equiv \varphi$.