CS 112 Introduction to Programming
(Spring 2012)

Lecture #16: Modular Development & Decomposition

Zhong Shao

Department of Computer Science
Yale University
Office: 314 Watson

http://flint.cs.yale.edu/cs112

Acknowledgements: some slides used in this class are taken directly or adapted from those accompanying the textbook: Introduction to Programming in Java: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne (Copyright 2002-2010)
Stepwise Refinement

**main idea**: use functions to divide a large programming problem into smaller pieces that are individually easy to understand ——— decomposition

**stepwise refinement** (or top-down design)
- start with the main program
- think about the problem as a whole and identify the major pieces of the entire task
- work each of these big pieces *one by one*
- for each piece, think what is its major sub-pieces, and repeat this process

run the calendar example
Example: Stepwise Refinement

We want to implement a calendar program that given any year later than 1900, prints out its calendar, each month is displayed as

February 1992
Su Mo Tu We Th Fr Sa
   1
2  3  4  5  6  7  8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
/*  
* File: calendar.java  
* -------------------  
* This program is used to generate a calendar for a year  
* entered by the user.  
*/

// public static void GiveInstructions();  
// public static int GetYearFromUser();  
// public static void PrintCalendar(int year);

/* Main program */  
public static void main(String[] args) {  
    int year;
        
    GiveInstructions();  
    year = GetYearFromUser();  
    PrintCalendar(year);
}

} // method main
Implementing GiveInstructions

/*
 * Function: GiveInstructions
 * Usage: GiveInstructions();
 * _______________________
 * This procedure prints out instructions to the user.
 */

public static void GiveInstructions() {

    System.out.println("This program displays a calendar for a full");
    System.out.println("year. The year must not be before 1900.");

} // method GiveInstructions
Implementing GetYearFromUser

/*
 * Function: GetYearFromUser
 * Usage: year = GetYearFromUser();
 * --------------------------------
 * This function reads in a year from the user and returns
 * that value. If the user enters a year before 1900, the
 * function gives the user another chance.
 */

public static int GetYearFromUser() {

    int year;

    while (true) {
        System.out.print("Which year? ");
        year = StdIn.readInt();
        if (year >= 1900) return (year);
        System.out.println("The year must be at least 1900.");
    }
    } // method GetYearFromUser
public static void PrintCalendarMonth(int month, int year);
...............................

/*
 * Function: PrintCalendar
 * Usage: PrintCalendar(year);
 * ---------------------------
 * This procedure prints a calendar for an entire year.
 */

public static void PrintCalendar(int year) {

    int month;

    for (month = 1; month <= 12; month++) {
        PrintCalendarMonth(month, year);
        System.out.println();
    }
}

} // method PrintCalendar
Implementing PrintCalendarMonth

February 1992
Su Mo Tu We Th Fr Sa
   1
  2  3  4  5  6  7  8
  9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29

public static void PrintCalendarMonth(int month, int year);

algorithm:
- print out the first two lines
- figure out how many days this month has (depending on leap year)
- decide on what day of the week the beginning of the month falls
- indenting the first line of the calendar so that the first day appears in the correct position
- loop around --- printing each day of the month, wrap around properly ......
Implementing PrintCalendarMonth

Define some symbolic constants:

```java
static final int Sunday = 0;
static final int Monday = 1;
static final int Tuesday = 2;
static final int Wednesday = 3;
static final int Thursday = 4;
static final int Friday = 5;
static final int Saturday = 6;
```

Algorithm for printing days

```java
for (day = 1; day <= nDays; day++) {
    if (day < 10) System.out.print("  "+day);
    else System.out.print(" "+day);
    if (weekday == Saturday) System.out.println();
    weekday = (weekday + 1) % 7;
}
```
/* 
* Function: PrintCalendarMonth 
*/

public static void PrintCalendarMonth(int month, int year) {
    int weekday, nDays, day;

    System.out.println("   " + MonthName(month) + " " + year);
    System.out.println(" Su Mo Tu We Th Fr Sa");
    nDays = MonthDays(month, year);
    weekday = FirstDayOfMonth(month, year);
    IndentFirstLine(weekday);
    for (day = 1; day <= nDays; day++) {
        if (day < 10) System.out.print("   " + day);
        else System.out.print(" " + day);
        if (weekday == Saturday) System.out.println();
        weekday = (weekday + 1) % 7;
    }
    if (weekday != Sunday) System.out.println();
} // method PrintCalendarMonth
Functions To Be Implemented

public static String MonthName(int month);
   *The English word for a specific month*

public static int MonthDays(int month, int year);
   *Calculate the number of days for a specific month of a specific year*

public static int FirstDayOfMonth(int month, int year);
   *Calculate what day of the week the beginning of the month (in a specific year) falls*

public static void IndentFirstLine(int weekday);
   *Indenting the first line so that the first day appears in the correct position*
Implementing MonthName

public static String MonthName(int month);

The English word for a specific month

public static String MonthName(int month) {
    switch (month) {
        case 1: return ("January");
        case 2: return ("February");
        case 3: return ("March");
        case 4: return ("April");
        case 5: return ("May");
        case 6: return ("June");
        case 7: return ("July");
        case 8: return ("August");
        case 9: return ("September");
        case 10: return ("October");
        case 11: return ("November");
        case 12: return ("December");
        default: return ("Illegal month");
    }
} // method MonthName
Implementing MonthDays

int MonthDays(int month, int year);

calculate the number of days for a specific month of a specific year

/*
 * Function: MonthDays
 * Usage: ndays = MonthDays(month, year);
 */

int MonthDays(int month, int year) {

    switch (month) {
        case 2:
            if (IsLeapYear(year)) return (29);
            return (28);
        case 4: case 6: case 9: case 11:
            return (30);
        default:
            return (31);
    }
} // method MonthDays
Functions To Be Implemented

public static int FirstDayOfMonth(int month, int year);
   *Calculate what day of the week the beginning of the month (in a specific year) falls*

public static void IndentFirstLine(int weekday);
   *Indenting the first line so that the first day appears in the correct position*

public static boolean IsLeapYear(int year);
   *Given a year, testing if it is a leap year!*
Implementing FirstDayOfMonth

/*
 * Function: FirstDayOfMonth
 * ----------------------------------------------
 * This function returns the day of the week on which the indicated month begins.
 * This program simply counts forward from January 1, 1900, which was a Monday.
 */

public static int FirstDayOfMonth(int month, int year){
    int weekday, i;

    weekday = Monday;
    for (i = 1900; i < year; i++) {
        weekday = (weekday + 365) % 7;
        if (IsLeapYear(i)) weekday = (weekday + 1) % 7;
    }
    for (i = 1; i < month; i++) {
        weekday = (weekday + MonthDays(i, year)) % 7;
    }
    return (weekday);
} // method FirstDayOfMonth
Implementing IndentFirstLine

public static void IndentFirstLine(int weekday);

    Parsing indenting so that the first day appears in the correct position

    */
    *
    * Function: IndentFirstLine
    * Usage: IndentFirstLine(weekday);
    * --------------------------------
    * This procedure indents the first line of the calendar
    * by printing enough blank spaces to get to the position
    * on the line corresponding to weekday.
    */

    public static void IndentFirstLine(int weekday) {
        int i;

        for (i = 0; i < weekday; i++) {
            System.out.print("   ");
        }
    } // method IndentFirstLine
Implementing IsLeapYear

public static boolean IsLeapYear(int year);

Given a year, testing if it is a leap year!

/***************************************************************************/
* Function: IsLeapYear
* Usage: if (IsLeapYear(year)) . . .
* ----------------------------------
* This function returns TRUE if year is a leap year.
* */

public static boolean IsLeapYear(int year) {

    return ( ((year % 4 == 0) && (year % 100 != 0))
            || (year % 400 == 0) );

} // method IsLeapYear
Another Case Study: Percolation

Percolation. Pour liquid on top of some porous material. Will liquid reach the bottom?

Applications. [ chemistry, materials science, ... ]

- Chromatography.
- Spread of forest fires.
- Natural gas through semi-porous rock.
- Flow of electricity through network of resistors.
- Permeation of gas in coal mine through a gas mask filter.
- ...


A Case Study: Percolation

Percolation. Pour liquid on top of some porous material. Will liquid reach the bottom?

Abstract model.
- $N$-by-$N$ grid of sites.
- Each site is either blocked or open.
A Case Study: Percolation

Percolation. Pour liquid on top of some porous material. Will liquid reach the bottom?

Abstract model.
- $N$-by-$N$ grid of sites.
- Each site is either blocked or open.
- An open site is full if it is connected to the top via open sites.
A Scientific Question

Random percolation. Given an N-by-N system where each site is vacant with probability $p$, what is the probability that system percolates?

Remark. Famous open question in statistical physics.

Recourse. Take a computational approach: Monte Carlo simulation.

<table>
<thead>
<tr>
<th>$p$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>does not percolate</td>
</tr>
<tr>
<td>0.4</td>
<td>does not percolate</td>
</tr>
<tr>
<td>0.5</td>
<td>does not percolate</td>
</tr>
<tr>
<td>0.6</td>
<td>percolates</td>
</tr>
<tr>
<td>0.7</td>
<td>percolates</td>
</tr>
</tbody>
</table>
Data representation. Use one $N$-by-$N$ boolean matrix to store which sites are open; use another to compute which sites are full.

Standard array I/O library. Library to support reading and printing 1- and 2-dimensional arrays.

```
8 8
0 0 1 1 1 0 0 0
1 0 0 1 1 1 1 1
1 1 1 0 0 1 1 0
0 0 1 1 0 1 1 1
0 1 1 1 0 1 1 0
0 1 0 0 0 0 1 1
1 0 1 0 1 1 1 1
1 1 1 1 0 1 0 0
```

shorthand: 0 for blocked, 1 for open

open[][]
Data Representation

Data representation. Use one $N$-by-$N$ boolean matrix to store which sites are open; use another to compute which sites are full.

Standard array I/O library. Library to support reading and printing 1- and 2-dimensional arrays.

**shorthand:** 0 for not full, 1 for full
public class StdArrayIO {
    ...

    // read M-by-N boolean matrix from standard input
    public static boolean[][] readBoolean2D() {
        int M = StdIn.readInt();
        int N = StdIn.readInt();
        boolean[][] a = new boolean[M][N];
        for (int i = 0; i < M; i++)
            for (int j = 0; j < N; j++)
                if (StdIn.readInt() != 0) a[i][j] = true;
        return a;
    }

    // print boolean matrix to standard output
    public static void print(boolean[][] a) {
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                if (a[i][j]) StdOut.print("1 ");
                else StdOut.print("0 ");
            }
            StdOut.println();
        }
    }
}
Scaffolding

Approach. Write the easy code first. Fill in details later.

```java
public class Percolation {

    // return boolean matrix representing full sites
    public static boolean[][] flow(boolean[][] open) {

        // does the system percolate?
        public static boolean percolates(boolean[][] open) {
            int N = open.length;
            boolean[][] full = flow(open);
            for (int j = 0; j < N; j++)
                if (full[N-1][j]) return true;
            return false;  \ system percolates if any full site in bottom row
        }

        // test client
        public static void main(String[] args) {
            boolean[][] open = StdArrayIO.readBoolean2D();
            StdArrayIO.print(flow(open));
            StdOut.println(percolates(open));
        }
    }
```
Vertical Percolation
Vertical Percolation

Next step. Start by solving an easier version of the problem.

Vertical percolation. Is there a path of open sites from the top to the bottom that goes **straight down**?
Vertical Percolation

Q. How to determine if site \((i,j)\) is full?

A. It's full if \((i,j)\) is open and \((i-1,j)\) is full.

**Algorithm.** Scan rows from top to bottom.
Vertical Percolation

Q. How to determine if site \((i,j)\) is full?
A. It’s full if \((i,j)\) is open and \((i-1,j)\) is full.

Algorithm. Scan rows from top to bottom.

```java
public static boolean[][] flow(boolean[][] open) {
    int N = open.length;
    boolean[][] full = new boolean[N][N];
    for (int j = 0; j < N; j++)
        full[0][j] = open[0][j];

    for (int i = 1; i < N; i++)
        for (int j = 0; j < N; j++)
            full[i][j] = open[i][j] && full[i-1][j];

    return full;
}
```
Vertical Percolation: Testing

Testing. Use standard input and output to test small inputs.

% more testT.txt
5
0 1 1 0 1
0 0 1 1 1
1 1 0 1 1
1 0 0 0 1
0 1 1 1 1

% more testF.txt
5
1 0 1 0 0
1 0 1 1 1
1 1 1 0 1
1 0 0 0 1
0 0 0 1 1

% java VerticalPercolation < testT.txt
5
0 1 1 0 1
0 0 1 0 1
0 0 0 0 1
0 0 0 1 1
ture

% java VerticalPercolation < testF.txt
5
1 0 1 0 0
1 0 1 0 0
1 0 1 0 0
1 0 0 0 0
0 0 0 0 0
false
Vertical Percolation: Testing

Testing. Add helper methods to generate random inputs and visualize using standard draw.

```java
public class Percolation {

    // return a random N-by-N matrix; each cell true with prob p
    public static boolean[][] random(int N, double p) {
        boolean[][] a = new boolean[N][N];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                a[i][j] = StdRandom.bernoulli(p);
        return a;
    }

    // plot matrix to standard drawing
    public static void show(boolean[][] a, boolean foreground)
}
```
Data Visualization

**Visualization.** Use standard drawing to visualize larger inputs.

```java
public class Visualize {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double p = Double.parseDouble(args[1]);
        boolean[][] open = Percolation.random(N, p);
        boolean[][] full = Percolation.flow(open);
        StdDraw.setPenColor(StdDraw.BLACK);
        Percolation.show(open, false);
        StdDraw.setPenColor(StdDraw.CYAN);
        Percolation.show(full, true);
    }
}
```

% java Visualize 20 .95 1    % java Visualize 20 .91
Vertical Percolation: Probability Estimate

**Analysis.** Given $N$ and $p$, run simulation $T$ times and report average.

```java
public class Estimate {

    public static double eval(int N, double p, int T) {
        int cnt = 0;
        for (int t = 0; t < T; t++) {
            boolean[][] open = Percolation.random(N, p);
            if (VerticalPercolation.percolates(open)) cnt++;
        }
        return (double) cnt / M;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double p = Double.parseDouble(args[1]);
        int T = Integer.parseInt(args[2]);
        StdOut.println(eval(N, p, T));
    }
}
```
**Vertical Percolation: Probability Estimate**

**Analysis.** Given $N$ and $p$, run simulation $T$ times and report average.

<table>
<thead>
<tr>
<th>Command</th>
<th>Expected Probability</th>
<th>Observed Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>% java Estimate 20 .7 100000</td>
<td>$1 - (1 - p^N)^N$</td>
<td>0.015768</td>
</tr>
<tr>
<td>% java Estimate 20 .8 100000</td>
<td>$1 - (1 - p^N)^N$</td>
<td>0.206757</td>
</tr>
<tr>
<td>% java Estimate 20 .9 100000</td>
<td>$1 - (1 - p^N)^N$</td>
<td>0.925191</td>
</tr>
<tr>
<td>% java Estimate 40 .9 100000</td>
<td>$1 - (1 - p^N)^N$</td>
<td>0.448536</td>
</tr>
</tbody>
</table>

**Running time.** Proportional to $TN^2$.

**Memory consumption.** Proportional to $N^2$. 

a lot of computation!

takes about 1 minute

takes about 4 minutes

agrees with theory
General Percolation
Percolation. Given an $N$-by-$N$ system, is there any path of open sites from the top to the bottom.

Depth first search. To visit all sites reachable from i-j:
- If i-j already marked as reachable, return.
- If i-j not open, return.
- Mark i-j as reachable.
- Visit the 4 neighbors of i-j recursively.

Percolation solution.
- Run DFS from each site on top row.
- Check if any site in bottom row is marked as reachable.
public static boolean[][] flow(boolean[][] open) {
    int N = open.length;
    boolean[][] full = new boolean[N][N];
    for (int j = 0; j < N; j++)
        if (open[0][j]) flow(open, full, 0, j);
    return full;
}

public static void flow(boolean[][] open,
                        boolean[][] full, int i, int j) {
    int N = full.length;
    if (i < 0 || i >= N || j < 0 || j >= N) return;
    if (!open[i][j]) return;
    if (full[i][j]) return;

    full[i][j] = true;  // mark
    flow(open, full, i+1, j);  // down
    flow(open, full, i, j+1);  // right
    flow(open, full, i, j-1);  // left
    flow(open, full, i-1, j);  // up
}
General Percolation: Probability Estimate

Analysis. Given $N$ and $p$, run simulation $T$ times and report average.

```
% java Estimate 20 .5 100000
0.050953

% java Estimate 20 .6 100000
0.568869

% java Estimate 20 .7 100000
0.980804

% java Estimate 40 .6 100000
0.595995
```

Running time. Still proportional to $T N^2$.

Memory consumption. Still proportional to $N^2$. 
Adaptive Plot
In Silico Experiment

Plot results. Plot the probability that an $N$-by-$N$ system percolates as a function of the site vacancy probability $p$.

Design decisions.
- How many values of $p$?
- For which values of $p$?
- How many experiments for each value of $p$?

![too few points](image1)
![too many points](image2)
![judicious choice of points](image3)
Adaptive Plot

Adaptive plot. To plot $f(x)$ in the interval $[x_0, x_1]$:

- Stop if interval is sufficiently small.
- Divide interval in half and compute $f(x_m)$.
- Stop if $f(x_m)$ is close to $\frac{1}{2}(f(x_0) + f(x_1))$.
- Recursively plot $f(x)$ in the interval $[x_0, x_m]$.
- Plot the point $(x_m, f(x_m))$.
- Recursively plot $f(x)$ in the interval $[x_m, x_1]$.

Net effect. Short program that judiciously chooses values of $p$ to produce a "good" looking curve without excessive computation.
public class PercolationPlot {
    public static void curve(int N, double x0, double y0,
    public static void curve(int N, double x1, double y1) {
        double gap = 0.05;
        double error = 0.005;
        int T = 10000;
        double xm = (x0 + x1) / 2;
        double ym = (y0 + y1) / 2;
        double fxm = Estimate.eval(N, xm, T);
        if (x1 - x0 < gap && Math.abs(ym - fxm) < error) {
            StdDraw.line(x0, y0, x1, y1);
            return;
        }
        curve(N, x0, y0, xm, fxm);
        StdDraw.filledCircle(xm, fxm, .005);
        curve(N, xm, fxm, x1, y1);
    }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        curve(N, 0.0, 0.0, 1.0, 1.0);
    }
}
Adaptive Plot

**Plot results.** Plot the probability that an $N$-by-$N$ system percolates as a function of the site vacancy probability $p$.

Phase transition. If $p < 0.593$, system almost never percolates; if $p > 0.593$, system almost always percolates.
Dependency Call Graph

Case study dependencies (not including system calls)
Lessons

Expect bugs. Run code on small test cases.

Keep modules small. Enables testing and debugging.

Incremental development. Run and debug each module as you write it.

Solve an easier problem. Provides a first step.

Consider a recursive solution. An indispensable tool.

Build reusable libraries. StdArrayIO, StdRandom, StdIn, StdDraw, ...