

## 1999 APS105S Lecture 10: Programming

### News

- getting help
  - please, use the newsgroup, or ask a TA
  - I'm getting lots of email, hard to type...
- quiz delayed by 1 week, Feb 5
  - will cover everything till end of this week
  - test mostly your understanding of Java
  - give you early feedback on how you're following the course
    - it will be a difficult test
    - should get perfect if you know what I've taught
  - most questions like
    - "what's wrong with this?"
    - "what does this program do?"
  - a few "write a short program to do this..." questions
- labs... can be marked 1 week late
- gives you a week to ASK QUESTIONS and get help in lab
- TAs will not answer questions during the late week... get it done BEFORE the lab, and start working on the next one

### Last Day

- Algorithms, pseudocode

### Today

- how to write good programs..... CORRECTLY!

### Golden Rules

- **KEEP IT SIMPLE** (stupid) (KISS)
- Start small, grow big.
- Test everything as you grow.

- indenting

```
if(...) {
  do this;
} else if(...) {
  do this;
} else {
  and this;
}

if(...)
{
  or this;
}

if(...)
{
  don't do this;
}

if(...) {
  ok so far
} else {
  if(...) {
    are you sure you want to do this;
  } else {
    don't do this;
  }
}
```

### Style

- align matching { .. }
- comments
  - say in English what you are trying to do
- methods calling methods...must do a return eventually!
- DON'T DO THIS:

```
deposit() {
  ...do deposit...
  if( Stdin.getChar() == 'w' )
    withdraw();
  ...
}
withdraw() {
  ...do withdraw...
  if( Stdin.getChar() == 'd' )
    deposit();
  ...
}
```
- variables
  - use intelligent names
  - spell it out, use acronyms only if OBVIOUS or necessary
  - use for *common subexpressions* eg:  $b^2-4ac$
- no "global variables"
  - hard to do in Java...learn what these are later
- no *magic numbers*
  - don't hardcode constants as numbers
  - use Math.PI instead of 3.14
  - define & use "final double SAVINGS\_INTEREST = 0.01;"

## 5 Steps to Good Quality Program Development

1. Obtain a clear statement of what the program is to do:

### "Requirements Specification"

- in assignment
- from boss
- from your own head
- from customer
- a major task, especially when people aren't quite sure of what they want

2. Analyze Specification / Problem

- identify input eg:  $a, b, c$
- identify outputs eg:  $root1, root2$
- identify constraints eg:  $b^2-4ac < 0$

3. Design Program

- *break the problem up into smaller problems*
- use English or pseudocode
- the English statements become comments in your program
  - read coefficients from user
  - check if termination
  - check if coefficients are valid
  - calculate root
  - display root
  - loop back to beginning
- start looking for "corner cases"

#### 4. Create Code

- translate English statements into Java
- // comment at top to identify author, date, copyright, etc.
- // comment at top to describe purpose of program
- // comment before each method to describe its task
  - // describe its parameters, return value, side effects

#### 5. Test Program

- compile → syntax errors → fix → repeat
- how do you know if it works?
  - “walk through program in head, see if it makes sense”
  - try one set of inputs “*test case*” — not enough!!!!
  - try all possible inputs? — too many!!!!
  - try a set of different, reasonable inputs
    - look at corner cases, test each one
    - cases must test each if clause, each else clause, each loop condition, each method
    - DIFFICULT to do! intuition, practise, cleverness
- debugging
  - finding and fixing errors

- is n prime?
    - 1, 2 are prime
    - a number is prime *iff* the only numbers that evenly divide it is 1 and the number itself
    - test if 2,3,4,... evenly divides number
      - last value to try is  $\leq \sqrt{\text{number}}$ ... why?
- ```
isPrime(number)
  if number <= 2
    return true
  else
    for i = 2..sqrt(number)
      if evenlyDivides(i,number)
        return false
      end for
    return true
```
- method to print prime factors of a number n
- ```
printPrimeFactors(n)
  for i = 1..n
    if isPrime(i) && evenlyDivides(i,n)
      print i
    end if
  end for
```
- build main() around this...
- ```
main()
  loop
    input n
    if n <= 0 quit
    primeFactors(n)
  end loop
```

## Example

### Step 1: Requirements Specification

- Continuously ask user for a number
- if  $\leq 0$  quit
  - else print its prime factors

### Step 2: Analyze Problem

*inputs:* integer n

*outputs:* all of its prime factors

*constraints:*

- each factor must *evenly divide* n
- each factor must *be prime*
- smallest factor will be 1
- largest possible factor is n

### Step 3: Design Pseudocode

*break the problem up!!!!!!*

- need to test if a number d evenly divides n
- need to test if a number is prime
- need to test only prime divisors d into n
  - print them if they are prime & evenly divide
- does d evenly divide n?

```
evenlyDivides(d,n) {
  return ((n/d)*d) == n
}
```

### Step 4: Create Code

```
// Written by Guy Lemieux Jan 25, 1999
// Input a number n > 0.
// Compute and print the prime
// factors of n. Enter <= 0 to quit.
class PrimeFactors
{
  // Returns true if d evenly divides n
  static boolean
  evenlyDivides( int d, int n )
  {
    return ((n/d)*d) == n;
  }

  // Return true only if n is prime.
  // Only look for factors up to sqrt(n).
  static boolean isPrime( int n )
  {
    int absN = (int)Math.abs( n ); //safe
    if( absN <= 2 )
      return true;
    else {
      double sqrtN = Math.sqrt( absN );
      for( int i=2; i<=sqrtN; i++ ) {
        if( evenlyDivides(i,absN) ) {
          return false;
        }
      }
    }
    return true;
  }
}
```

```

// Prints all the prime factors of n,
// n >= 0
static void printPrimeFactors( int n )
{
    for( int i=1; i<=n; i++ ) {
        if( evenlyDivides(i,n)
            && isPrime(i) )
            // why this order inside if() ???
            System.out.print( i + " " );
    }
    System.out.println();
}

// Ask user for an integer > 0, print
// its prime factors. Any value <= 0
// quits.
public static void main(String[] args)
{
    while(true) {
        int n = Stdin.getInt();
        if( n <= 0 )
            break;
        printPrimeFactors(n);
    }
}

```

### Step 5: Test Program

Some sample test input, prepare expected output.

```
0 1 2 3 4 5 6 8 100 256 1000 9999993
```

**Reading: 7.1, 7.2, 7.3, 7.4, 7.5, 7.7 (PAGE 314)**

## Debugging

- logic error in your program — called a *bug*
- must find and fix these — called *debugging*
- find errors by using good test cases
- easy to discover an error exists,
  - but difficult to find cause of error “*the bug*”

### Debugging techniques:

- test *each method* separately and completely
- find test case that cause incorrect program behaviour
- trace program with test case
  - in your head, on paper
- add print statements to print intermediate results
- work out intermediate results by hand
  - if results agree... error has not been seen yet
  - if disagree.... error has already occurred

### Test your assumptions!!!!

- use `if()` statements to help you!
- assertions — a statement that should always be true at that point in your program

```

static int deposit( int balance )
{
    if( balance < 0 ) {
        System.out.println(
            "Assertion error, balance < 0" );
    }
    ...
}

```

- loop invariants — a statement inside a loop that is true during every loop iteration (an assertion inside a loop)

```

for( i=3; i<=n; i++ ) {
    fibN = fibN1 + fibN2;
    if( fibN < fibN1 || fibN1 < fibN2 )
        System.out.println(
            "Loop invariant error" );
    fibN2 = fibN1;
    fibN1 = fibN;
}

```

- use `//` or `/* */` comments to easily add/remove code

```

// power(3.0,2);
/* System.out.println("The answer");
System.out.println(" is wrong.");
*/

```

- use `if( false )` to easily add/remove code

```

boolean DEBUGFLAG = false;
...
if( DEBUGFLAG ) {
    System.out.println( "x = " + x );
}

if( false ) {
    //code to be removed is between {...}
}

```