Why functions (methods)?

- Reduce redundant code -- If you use it twice, make it a function
- Simplify main programs/procedures
- Provide abstraction/decomposition/interfaces
- Building blocks -- make new statements
- Recursion
- “Object oriented” and top down design

Methods: Functional Abstraction

- Structured Programming
  - The flow of control in a program should be as simple as possible
  - The construction of a program should embody top-down design
- Top-Down Design
  - Repeatedly decompose a problem into smaller subproblems
    - Each problem solved by using subproblem solutions
  - Eventually, smallest subproblems are directly solvable

Example

- Problem: Play tic-tac-toe
  1. Find the best move
  2. Make that move
  3. Wait for the other player to move
  4. Go to step 1
- Find the best move
  1. If there is a winning move, choose it
  2. If there is a blocking move, choose it
  3. If there is a move that leads to a win, choose it
  4. Etc.

Method Invocation

- A simple program contains one or more methods (also called functions), including
  - main(), where program execution begins
- When program control encounters a method name followed by (), it is called or invoked
  - Program control passes to the called method
  - When the called method is finished executing, program control returns to the calling method, where program execution continues

Static Method Definition

```java
// Message.java: Simple method use
class Message {
    public static void main(String [] args){
        System.out.println("HELLO DEBRA");
        printMessage();    //method call
        System.out.println("Goodbye.");
    }

    // definition of method printMessage
    static void printMessage(){
        System.out.println("A message for you:");
        System.out.println("Have a nice day!
");
    }
}
```

- public static – trust me for now
- <returntype> - The type of data returned by the method
  - void means nothing is returned
- <ident> - the method name
- <paramlist> - list of inputs to the method
- <block> - the code that will get executed when the method is invoked
Details

- **Parameters**
  - Values passed from the calling function to the called function
  - Act like variables inside the called function
  - Other variables no longer available

- **Body of the method (<block>)**
  - Variable declarations and statements that are executed when the method is called

The return Statement

- Returns program control to the calling method
- May return a value of the appropriate type
  ```java
  return a;
  return (a + b);
  return "error!";
  ```
- A method can have zero or more return statements
  - Control returns to the calling method as soon as one is reached
  - If no return statement is reached, control returns to the calling method when the end of the method is reached

Scope of Variables

- The **scope** of a variable is the range of statements that can access it
- Any variable declared within a method is a **local** variable
  - Fresh copy each time the method is called
  - Cease to exist after the method finishes executing
  - Scope: any statement after the declaration and before the end of the block in which it is declared

  - The scope of variables declared in the initialization portion of a for loop includes the boolean expression, update expression, and the loop body

Drawing Shapes

- **Problem:** Draw a triangle or diamond on the screen based on input
- **Top Down design:**
  ```java
  // Min2.java -return expression in a method
class Min2 {
    public static void main(String[] args)
    {
        int j = 78, k = 3 * 30, m;
        System.out.println("Minimum of two integers Test:");
        m = min(j, k);
        System.out.println("The minimum of:" + j + ":" + k + "is:" + m);
    }
}
  ```

//Min2Bad.java -doesn't work because of scope
class Min2Bad {
    public static void main(String[] args)
    {
        int j = 78, k = 3 * 30, m;
        System.out.println("Minimum of two integers Test:");
        m = min();
        System.out.println("The minimum of:" + j + ":" + k + "is:" + m);
    }
    static int min() {
        if (j < k)
            return j;
        else
            return b;
    }
}

//Min2Bad.java -doesn't work because of scope
class Min2Bad {
    public static void main(String[] args)
    {
        int j = 78, k = 3 * 30, m;
        System.out.println("Minimum of two integers Test:");
        m = min();
        System.out.println("The minimum of:" + j + ":" + k + "is:" + m);
    }
    static int min() {
        if (j < k)
            return j;
        else
            return k;
    }
}
Example of Top-Down Design

- Problem: Find the relative areas of a unit circle and a unit square
- One way to do this:
  - Dartboard with a square with a circle inside
  - Throw darts blindfolded and count the number that fall inside the circle and divide by the total number thrown
  - Or, by simulating the dartboard, generate random numbers representing dart locations

Algorithm

1. Find out the number of trials to execute
2. Execute the specified number of trials
3. Calculate the relative areas
4. Output the results

1. Find out the number of trials to execute
   1. Ask the user how many trials to execute
   2. Store the number in a local variable

2. Execute the specified number of trials
   1. Set i equal to zero
   2. If i is less than the number of trials
      1. Execute a trial
      2. Record the result
      3. Increment i
      4. Repeat

Execute a trial

1. Generate two random numbers x and y, between 0 and 1
2. See if (x,y) lies within distance 1/2 of (1/2,1/2), i.e. in radius 1/2 circle
3. If so, return true
4. Otherwise, return false

3. Calculate the relative areas
   1. Divide the number of successful trials by the total number of trials
   2. Return the result
// Calculate the percentage of a unit square taken up by a unit circle
class RelativeAreas {
    public static void main(String[] args) {
        int count, successful;
        double ratio;

        System.out.println("Please enter the number of trials: ");
        System.in.readInt();
        numTrials = Console.in.readInt();
        return numTrials;
    }
}

static boolean oneTrial() {
    double x, y;
    double distance;
    x = Math.random();
    y = Math.random();
    distance = Math.sqrt((0.5 - x)*(0.5 - x) + (0.5 - y)*(0.5 - y));
    return (distance <= 0.5);
}

static void printResults(int successful, int count) {
    double ratio = (double)successful / count;
    System.out.println("Percentage = " + ratio * 100);
}

// FailedSwap - Call-By-Value test
class FailedSwap {
    public static void main(String[] args) {
        int numOne = 1, numTwo = 2;
        swap(numOne, numTwo);
        System.out.println("numOne =+numOne");
        System.out.println("numTwo =+numTwo");
    }

    static void swap(int x, int y) {
        int temp;
        System.out.println("x =+=x");
        System.out.println("y =+=y");
        temp = x;
        x = y;
        y = temp;
        System.out.println("x =+=x");
        System.out.println("y =+=y");
    }
}

Invocation and Call-by-Value

- To call one method from another method in the same class
  - Write the name of the method, and
  - a list of arguments in parentheses
- The arguments have to match in number and type those listed in the method definition
- Each argument is evaluated, and its value is used to initialize the corresponding formal parameter in the method invocation
  - Changing the value of a parameter in a method does not change the value of the thing passed to it!

Boxes R Us

- Custom box manufacturer
- Find good box size for different size items
- Get customers to buy our custom boxes
- Use program as advertising tool
Recall: Software Life Cycle

• Requirements analysis and definition
• Design
• Implementation
• Testing
• Maintenance

Requirements Questions

• What is the role of the program?
• How will it get data? What output?
• Correct behavior (performance)?

Requirements

1. Be user friendly
2. Get a set of item sizes (height, width, depth)
   Inches/mm? Integer/fractions?
3. Print good sizes that pack items well
   1. What does “well” mean?
   2. Can items be rotated?
   3. What is a good size?

Algorithm (high level)

Assume items must be top up, can swap depth and width
1. Print directions and get sizes of items
2. Find box height that is even multiple of item heights
3. Try all possible width/depth swaps to find smallest width/depth that is multiple of rotated item width/depths
4. Any multiple of these sizes works for box dimensions

Step 2 find Box Height from item heights

• Want the least common multiple of item heights (LCM)
• LCM of two integers is product divided by greatest common divisor (GCD)
• Generalize to more than two (an exercise)
• What if item heights not an integer, for example: 3 1/8 in?

Euclid’s GCD algorithm

• To compute GCD of j and k:
  – If k < j swap them // ensures j <= k
  – If j == 0 then GCD is k
  – Else
    • Compute i = k mod j
    • GCD of j and k equals GCD of j and i
  Note that the integers go down in size
GCD example

- Find GCD of 12 and 20
  - 20 mod 12 = 8  find GCD of 12 and 8
  - 12 mod 8 = 4  find GCD of 8 and 4
  - 8 mod 4 = 0  4 is GCD of 8 and 4
  - GCD of 12 and 20 is also 4

- LCM of 12 and 20 is 12*20/4 = 60

Box Height algorithm (design)

- Given two (positive) heights, j and k
  - Compute GCD of j and k using Euclid’s alg
  - Compute (and return) j*k/GCD

- Objects: numbers j, k and their GCD
- Operations: LCM and GCD

Let’s implement LCM!

Testing

- At a minimum you want to
  - Execute every instruction at least once
  - Take every branch at least once
  - Try every possible valid input
  - Try every possible type of invalid input
- This isn’t always possible
  - NORAD
  - Calculator program
  - Do the best you can
- Unit testing and Defensive programming is easy and useful!

Recursion

- When a method calls itself, this is referred to as recursion
- Recursion can be confusing, but is extremely powerful
- Often used when a mathematical operation is defined in terms of other values of itself
  - Examples: factorials, fibonacci numbers, …

Recursive Methods

- Recursive methods have three parts
  - A part that does something
  - A part that calls the method
  - A part that does not call the method
    - Otherwise it would go forever
    - There is a test to decide whether or not to call the method again
Form of a Recursive Function

```java
public static <type> recursiveMethod(<args>) {
    <whatever>
    if(<stopping condition>)
        <whatever you do at the end>
    else
        recursiveMethod(<different args>);
}
```

Example: Factorial

- n! = n * (n-1) * (n-2) * … * 2 * 1
- n! = n * (n-1)!
- Recall: 0! = 1 and 1! = 1

```java
public static int factorial(int n) {
    if(n <= 1)
        return 1;
    else
        return (n * factorial(n-1));
}
```

Example: Factorial (cont.)

- Suppose we execute factorial(4)
  - main calls factorial(4) <a>
  - <a> calls factorial(3) <b>
  - <b> calls factorial(2) <c>
  - <c> calls factorial(1) <d>
    - <d> returns 1
    - <c> returns 2 * 1 (= 2)
    - <b> returns 3 * 2 (= 6)
    - <a> returns 4 * 6 (= 24)
  - and that is the answer: 24

Example: Fibonacci numbers

- Each Fibonacci numbers is defined as the sum of the two previous fibonacci numbers
- fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
- fibonacci(0) = 1, fibonacci(1) = 1

```java
public static int fibonacci(int n) {
    if(n <= 1)
        return 1;
    else
        return (fibonacci(n-1) + fibonacci(n-2));
}
```

Example: Fibonacci (cont.)

- Suppose we execute fibonacci(3)
  - main calls fibonacci(3) <a>
  - <a> calls fibonacci(2) <b> and fibonacci(1) <c>
  - <b> calls fibonacci(1) <d> and fibonacci(0) <g>
    - <f> returns 1
    - <g> returns 1
    - <c> returns 1
    - <d> returns 1
    - <b> returns 2
    - <a> returns 3
  - and that is the answer: 3

Recursion Wrapup

- Recursion is appropriate for any mathematical function that can be defined in terms of previous values of itself:
  - f(x) = g(f(y)), where y < x
- Examples:
  - Exponential: \(x^n = x \cdot x^{n-1}\)
Example: Mathematical Functions

- Often want to know the zero crossings of a function - the values of x for which f(x) = 0
- This example doesn’t illustrate any specific point having to do with methods, but does bring up lots of useful things to discuss
- We will examine two possible solutions
  - Linear search
  - Binary search

```java
class SimpleFindRoot {
    public static void main(String[] args) {
        double a = 0.0, b = 10.0, x = a, step = 0.001;
        while (f(x) != 0.0 && x < b)
            x = x + step;
        if (x < b)
            System.out.println("root is "+x);
        else
            System.out.println("root not found");
    }
    static double f(double x) {
        return (x * x - 2.0);
    }
}
```

```java
class FindRoot {
    public static void main(String[] args) {
        double a = 0.0, b = 10.0, eps = 0.00001, root = 0.0, residual;
        while (b - a > eps) {
            root = (a + b) / 2.0;
            residual = f(root);
            if (residual > 0)
                b = root;
            else
                a = root;
        }
        System.out.println("root is "+root);
    }
    static double f(double x) {
        return (x * x - 2.0);
    }
}
```

Method Overloading

- Simple idea: The method called is determined by:
  - the name of the method, and
  - the number and type of parameters in the call
- So, two methods can have the same name as long as they have different numbers and/or types of parameters

```java
static int min(int s, int t) {
    if (s < t)
        return s;
    else
        return t;
}
static double min(double s, double t) {
    if (s < t)
        return s;
    else
        return t;
}

public static void main(String[] args) {
    double a, b, c;
    int w, x, y, z;
    c = min(a, b);
    z = min(x, y);
    w = min(a, y);
}
```
Other examples of method overloading
• System.out.println()
• ...

Applets
• Graphical java programs
• Have no “main()” method
• Run inside a viewer or browser
  − appletViewer
    appletViewer FirstApplet.java
  − In a web page
    <applet code="FirstApplet.class" width=500 height=200></applet>

FirstApplet.java
• paint() method instead of main()
• Parameter: Graphics object
  − Supports drawing methods (see javadoc)

AppletSum.java

DrawChairs.java