Recursion

• Recursion is a fundamental programming technique that can provide an elegant solution to certain kinds of problems.

• Chapter 12 focuses on:
  – thinking in a recursive manner
  – programming in a recursive manner
  – the correct use of recursion
  – recursion examples
Outline

Recursive Thinking
Recursive Programming
Using Recursion
Recursion in Graphics
Recursive Thinking

• A recursive definition is one which uses the word or concept being defined in the definition itself.

• When defining an English word, a recursive definition is often not helpful.

• But in other situations, a recursive definition can be an appropriate way to express a concept.

• Before applying recursion to programming, it is best to practice thinking recursively.
Recursive Definitions

• Consider a list of numbers:

    24, 88, 40, 37

• A list can be defined as follows:

    A List is a: number
    or a: number comma List

• That is, a List is defined to be a single number, or a number followed by a comma followed by a List

• The concept of a List is used to define itself
Recursive Definitions

- The recursive part of the LIST definition is used several times, terminating with the non-recursive part:

```
LIST: number comma LIST
24 , 88, 40, 37
   number comma LIST
   88 , 40, 37
      number comma LIST
      40 , 37
         number
         37
```
Peano's def. of Natural Numbers

• The following two **axioms** define the natural numbers
  – 0 is a natural number
  – For every natural number \( n \), \( S(n) \) – the successor of \( n \) - is a natural number

• The number 1 can be defined as \( S(0) \), 2 as \( S(S(0)) \) (which is also \( S(1) \)), and, in general, any natural number \( n \) as \( S^n(0) \)

• The next two axioms define their properties:
  – For every natural number \( n \), \( S(n) = 0 \) is false. That is, **there is no natural number whose successor is 0**
  – For all natural numbers \( m \) and \( n \), if \( S(m) = S(n) \), then \( m = n \). That is, **\( S \) is an injection**.
Infinite Recursion

• All recursive definitions have to have a non-recursive part called the base case

• If they didn't, there would be no way to terminate the recursive path

• For instance:

  A List is a: number comma List

• Such a definition would cause infinite recursion

• This problem is similar to an infinite loop
Quiz

What is printing the program described in this flowchart?
public int itFactorial (int n) {
    int m = 1, f = 1;
    while (m < n) {
        ++m;
        f = f * m;
        }
    return f;
}
Recursive Definition: Factorial

• N!, for any positive integer N, is defined to be the product of all integers between 1 and N inclusive

\[ N! = N \times (N-1) \times (N-2) \times \ldots \times 2 \times 1 \]

• This definition can be expressed recursively as:

\[ f(1) = 1 \]
\[ f(n) = n \times f(n-1) \]

• A factorial is defined in terms of another factorial

• Eventually, the base case of 1! is reached
Recursive Factorial

5!
5 * 4!
4 * 3!
3 * 2!
2 * 1!

120
24
6
2
1

Copyright © 2012 Pearson Education, Inc.
Quick Check

Write a recursive definition of $f(n) = e^n$, where $n \geq 0$. 
Quick Check

Write a recursive definition of \( f(n) = e^n \), where \( n \geq 0 \).

\[
\begin{align*}
  e^0 &= 1 & f(0) &= 1 \\
  e^n &= e \times e^{n-1} & f(n) &= e \times f(n-1)
\end{align*}
\]

In this way you can compute \( e^n \) by just using multiplications.
Quick Check

Write a recursive definition of

\[ f(n) = 5 \times n \]

where \( n > 0 \).
Quick Check

Write a recursive definition of
\[ f(n) = 5 \times n \]
where \( n > 0 \).

\[ 5 \times 1 = 5 \]
\[ 5 \times n = 5 + (5 \times (n-1)) \]

\[ f(1) = 5 \]
\[ f(n) = 5 + f(n-1) \]
Quick Check

Write a recursive definition of $f(n) = \frac{(n+1)n}{2}$, where $n > 0$. 
Quick Check

Write a recursive definition of \( f(n) = (n + 1)n/2 \), where \( n > 0 \).

\[
f(1) = 1
\]

\[
f(n+1) = (n + 2)(n + 1)/2 = (n + 1)n/2 + 2(n+1)/2
= f(n) + (n + 1)
\]
private class SliderListener implements ChangeListener {
    private int red, green, blue;

    //---------------------------------------------
    // Gets the value of each slider, then updates
    // the color panel.
    //---------------------------------------------
    public void stateChanged (ChangeEvent event) {
        red = rSlider.getValue();
        green = gSlider.getValue();
        blue = bSlider.getValue();

        rLabel.setText ("Red: " + red);
        gLabel.setText ("Green: " + green);
        bLabel.setText ("Blue: " + blue);

        colorPanel.setBackground (new Color (red, green, blue));
    }
}
Recursive Programming

- A recursive method is a method that invokes itself.

- A recursive method must be structured to handle both the base case and the recursive case.

- Each call to the method sets up a new execution environment, with new parameters and local variables.

- As with any method call, when the method completes, control returns to the method that invoked it (which may be an earlier invocation of itself).
Sum of 1 to N

• Consider the problem of computing the sum of all the numbers between 1 and any positive integer N.

• This problem can be recursively defined as:

\[
\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i = N + (N-1) + \sum_{i=1}^{N-2} i
\]

\[
= N + N - 1 + N - 2 + \cdots + 2 + 1
\]
Sum of 1 to N

- The summation could be implemented recursively as follows:

```java
// This method returns the sum of 1 to num
public int sum (int num)
{
    int result;

    if (num == 1)
        result = 1;
    else
        result = num + sum (num-1);

    return result;
}
```
public int sum (int num) {
    int result;
    if (num == 1)
        result = 1;
    else
        result = num + sum (num-1);
    return result;
}
Recursive Programming

• Note that just because we can use recursion to solve a problem, doesn't mean we should

• We usually would not use recursion to solve the summation problem, because the iterative version is easier to understand

• However, for some problems, recursion provides an elegant solution, often cleaner than an iterative version (e.g. Fibonacci)

• You must carefully decide whether recursion is the correct technique for any problem
Quiz

• Write a recursive method that computes the factorial of a non-negative int number n:
  factorial(0)=1, factorial(n) = n * factorial(n-1)
Factorial

public int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial(n-1);
}

factorial(4)
factorial(3)
factorial(2)
factorial(1)
factorial(0)
    return 1
    return 1*1 = 1
    return 2*1 = 2
    return 3*2 = 6
    return 4*6 = 24
Indirect Recursion

• A method invoking itself is considered to be direct recursion

• A method could invoke another method, which invokes another, etc., until eventually the original method is invoked again

• For example, method \( m_1 \) could invoke \( m_2 \), which invokes \( m_3 \), which in turn invokes \( m_1 \) again

• This is called indirect recursion, and requires all the same care as direct recursion

• It is often more difficult to trace and debug
Indirect Recursion
Quiz

L is the left propagation and R is the right propagation:

• \( L(n) = L(R(n-1)) \)
• \( R(n) = R(L(n-1)) \)
• \( L(1) = R(1) = k \)

• For which values of \( k > 0 \), \( L(n) \) and \( R(n) \) terminate for all \( n > 0 \)?
Quiz

L is the left propagation and R is the right propagation:

- \( L(n) = L(R(n-1)) \)
- \( R(n) = R(L(n-1)) \)
- \( L(1) = R(1) = k \)

For which values of \( k > 0 \), \( L(n) \) and \( R(n) \) terminate for all \( n > 0 \)?

- **Only \( k = 1 \)** (and \( L \) and \( R \) are constant functions = 1)
- In fact, if \( k = 2 \) then:
  - \( L(2) = L(R(1)) = L(2) \) infinite loop
- If \( k = 3 \) then:
  - \( L(3) = L(R(2)) = L(R(L(1))) = L(R(3)) = L(R(2)) \) infinite loop
- If \( k = 4 \) …
Exercise

• Implement a class Chap12 that contains the right and left (static) methods described before and test that there is an infinite loop if k>1
public class Chap12 {

    private final static int K = 1;

    public static int right(int n) {
        System.out.println("right");
        if (n == 1)
            return K;
        return right(left(n-1));
    }

    public static int left(int n) {
        System.out.println("left");
        if (n == 1)
            return K;
        return left(right(n-1));
    }

    public static void main(String[] args) {
        System.out.println(right(2));
    }
}
Quiz

- What does the following recursive function return? Try it when the parameter s is your name.

```java
public String mystery(String s) {
    int N = s.length();
    if (N <= 1)
        return s;
    String a = s.substring(0, N/2);
    String b = s.substring(N/2, N);
    return mystery(b) + mystery(a);
}
```
Quiz

• What does the following recursive function return?

```java
public String mystery(String s) {
    int N = s.length();
    if (N <= 1)
        return s;
    String a = s.substring(0, N/2);
    String b = s.substring(N/2, N);
    return mystery(b) + mystery(a);
}
```

The reverse of the input string.
Mathematical Induction

• Recursive programming is directly related to mathematical induction, a technique for proving facts about discrete functions.

• Proving by mathematical induction that a statement involving an integer $N$ is true for all $N$ involves two steps:
  - **The base case:** to prove the statement true for some specific value or values of $N$ (usually 0 or 1).
  - **The induction step:** assume that a statement is true for all positive integers less than $N$, then use that fact to prove it true for $N$. 
Proof by Induction Example

• Prove that:
  – \( 1 + 2 + 3 + 4 + \ldots + N = \frac{(N+1)N}{2} \)

• Base case:
  – \( 1 = (1+1)1/2 \) TRUE

• Induction step:
  – Assume that it is true for \( N-1 \)
    • \( 1+ \ldots+(N-1) = \frac{N(N-1)}{2} \)
  – Then:
    • \( 1+ \ldots+(N-1) + N = \frac{N(N-1)}{2} + N \)
    • = \( \frac{N^2 - N + 2N}{2} \)
    • = \( \frac{N^2 + N}{2} = \frac{(N + 1)N}{2} \) Q.E.D.
Without using Induction

• Prove that:
  – $1 + 2 + 3 + 4 + \ldots + N = S_n = (N+1)N/2$

• $(1 + 2 + 3 + 4 + \ldots + N) + (1 + 2 + 3 + 4 + \ldots + N) = 2S_n$
• $(1 + 2 + 3 + 4 + \ldots + N) + (N + (N-1) + \ldots + 1) = 2S_n$
• $(1 + N) + (2 + N-1) + (3 + N-2) + \ldots + (N + 1) = 2S_n$
• $N(N+1) = 2S_n$
• $S_n = (N+1)N/2$
Quiz

• Consider the fibonacci sequence:
  – $f(0) = 0$, $f(1) = 1$, $f(n) = f(n-1) + f(n-2)$
  – 0, 1, 1, 2, 3, 5, 8, 13, ..

• Prove by induction that:
  – For all $n > 0$, $f(3n)$ is even
Quiz

• Consider the fibonacci sequence:
  – \( f(0) = 0, \ f(1) = 1; \ f(n) = f(n-1) + f(n-2) \)
  – 0, 1, 1, 2, 3, 5, 8, 13, ..

• Prove by induction that:
  – For all \( n > 0 \), \( f(3n) \) is even

• Base case \( n=1 \)
  – \( f(3) = 2 \) TRUE

• Induction step
  – if \( f(3n) \) is even we must prove that \( f(3(n+1)) \) is even
  – \( f(3(n+1)) = f(3n+2) + f(3n+1) = f(3n+1) + f(3n) + f(3n+1) = 2f(3n+1) + f(3n) \) THIS is EVEN
Recursion can be inefficient

```c
int fibonacci(int n) {
    if (n < 2)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

How many calls to `fibonacci` for computing `fibonacci(5)`?
Recursion can be inefficient

```c
int fibonacci(int n) {
    if (n < 2)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

How many calls to `fibonacci` for computing `fibonacci(5)`?

\[
\begin{array}{ccccccc}
  & f(5) & & & & & \\
  f(4) & & & & & & \\
  f(3) & f(3) & f(2) & f(2) & f(1) & & \\
  f(2) & f(2) & f(1) & f(0) & f(1) & f(0) & \\
  f(1) & f(1) & f(0) & f(0) & & & \\
  f(1) & f(0) & & & & &
\end{array}
\]

15 calls
return 5

f(5)

return 3

f(4)

return 2

f(3)

return 1

f(2)

return 1

f(1)

return 1

f(0)

return 2

f(1)

return 1

f(1)

return 1

f(0)

return 1

f(1)

return 1

f(1)

return 0

f(0)

return 1

f(1)

return 0

f(0)

return 1

f(1)

return 0

f(0)

return 1

f(1)

return 1

f(1)

return 0

f(0)
Iterative version of Fibonacci

```c
int itFibonacci(int n) {
    int result = 0, prec = 1;
    for (int i = 1; i <= n; i++) {
        result += prec; // f(i) = f(i-1) + f(i-2)
        prec = result - prec; // f(i-1) = f(i) - f(i-2)
    }
    return result;
}
```

\[ f(0) = 0, f(1) = 1, f(2) = 1, \ldots, f(n) = f(n-1) + f(n-2) \]
\[ 0, 1, 1, 2, 3, 5, 8, 13, \ldots \]
Maze Traversal

• We can use recursion to find a path through a maze
• From each location, we can search in each direction
• The recursive calls keep track of the path through the maze
• The base case is an invalid move or reaching the final destination

• See MazeSearch.java
• See Maze.java
public class MazeSearch
{
    //---
    //  Creates a new maze, prints its original form, attempts to
    //  solve it, and prints out its final form.
    //---
    public static void main (String[] args)
    {
        Maze labyrinth = new Maze();

        System.out.println (labyrinth);

        if (labyrinth.traverse (0, 0))
            System.out.println ("The maze was successfully traversed!");
        else
            System.out.println ("There is no possible path.");

        System.out.println (labyrinth);
    }
}
public class MazeSearch {

// Creates a new maze, prints its original form, attempts to
// solve it, and prints out its final form.

public static void main (String[] args) {
    Maze labyrinth = new Maze();
    System.out.println(labyrinth);
    if (labyrinth.traverse(0, 0))
        System.out.println("The maze was successfully traversed!");
    else
        System.out.println("There is no possible path.");
    System.out.println(labyrinth);
}

Output

1110110001111
1011101111001
0000101010100
1110111010111
1010000111001
1011111101111
1000000000000
1111111111111

7770110001111
3077707771001
0000707070300
7770777070333
7070000773003
else 7077777703333
7000000000000
7777777777777

The maze was successfully traversed!
public class Maze
{
    private final int TRIED = 3;
    private final int PATH = 7;

    private int[][] grid = {
        {1,1,1,0,1,0,0,0,1,1,1,1},
        {1,0,1,1,0,1,1,1,0,1,0,1},
        {0,0,0,1,0,1,0,1,1,0,1,0},
        {1,1,1,0,1,0,1,1,0,1,1,1},
        {1,0,1,0,0,0,1,1,1,0,1,1},
        {1,0,1,1,1,1,0,1,0,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0},
        {1,1,1,1,1,1,1,1,1,1,1,1}
    };

    // continued
public boolean traverse (int row, int column) {
    boolean done = false;

    if (valid (row, column)) {
        grid[row][column] = TRIED; // this cell has been tried

        if (row == grid.length - 1 && column == grid[0].length - 1)
            done = true; // the maze is solved - base case
        else {
            done = traverse (row+1, column); // down
            if (!done)
                done = traverse (row, column+1); // right
            if (!done)
                done = traverse (row-1, column); // up
            if (!done)
                done = traverse (row, column-1); // left
        }

        if (done) // this location is part of the final path
            grid[row][column] = PATH;
    }

    return done;
}
private boolean valid (int row, int column)
{
    boolean result = false;

    // check if cell is in the bounds of the matrix
    if (row >= 0 && row < grid.length &&
        column >= 0 && column < grid[row].length)
    {
        // check if cell is not blocked and not previously tried
        if (grid[row][column] == 1)
            result = true;

        return result;
    }
}

continued
public String toString ()
{
    String result = "\n";

    for (int row=0; row < grid.length; row++)
    {
        for (int column=0; column < grid[row].length; column++)
            result += grid[row][column] + "";
        result += "\n";
    }

    return result;
}
Quiz

• Trace the calls to traverse() for the maze row0=11, row1=01
Trace

return true

traverse(0,0)

return false

traverse(1,0)

return true

traverse(0,1)

return true

traverse(1,1)
Quiz

- More elaborated trace the calls to `traverse()` and `valid()` for the maze row0=11, row1=01
Towers of Hanoi

• The *Towers of Hanoi* is a puzzle made up of three vertical pegs and several disks that slide onto the pegs.

• The disks are of varying size, initially placed on one peg with the largest disk on the bottom with increasingly smaller ones on top.
Towers of Hanoi

• The goal is to move all of the disks from one peg to another

• Under the following rules:
  – Move only one disk at a time
  – A larger disk cannot be put on top of a smaller one
Towers of Hanoi

Original Configuration

Move 1

Move 2

Move 3
Towers of Hanoi

Move 4

Move 5

Move 6

Move 7 (done)
Recursive Solution

• To solve a N-tower
  1. Solve the (N-1)-tower: move the (N-1)-tower in the middle peg
  2. Move the largest disc to target peg
  3. Solve the (N-1)-tower: move the (N-1)-tower from the middle peg to the target peg
Towers of Hanoi

• An iterative solution to the Towers of Hanoi is quite complex

• A recursive solution is much shorter and more elegant

• See SolveTowers.java

• See TowersOfHanoi.java
public class SolveTowers
{
    //------------------------------------------------
    //  Creates a TowersOfHanoi puzzle and solves it.
    //------------------------------------------------
    public static void main (String[] args)
    {
        TowersOfHanoi towers = new TowersOfHanoi (4);  
        towers.solve();
    }
}
public class SolveTowers {
    
    // Demonstrates recursion.
    
    public static void main (String[] args) {
        TowersOfHanoi towers = new TowersOfHanoi (4);
        towers.solve();
    }
}

Output

Move one disk from 1 to 2
Move one disk from 1 to 3
Move one disk from 2 to 3
Move one disk from 1 to 2
Move one disk from 3 to 1
Move one disk from 3 to 2
Move one disk from 1 to 2
Move one disk from 1 to 3
Move one disk from 2 to 3
Move one disk from 2 to 1
Move one disk from 3 to 1
Move one disk from 2 to 3
Move one disk from 1 to 2
Move one disk from 1 to 3
Move one disk from 2 to 3
public class TowersOfHanoi
{
    private int totalDisks;

    // Sets up the puzzle with the specified number of disks.
    public TowersOfHanoi (int disks)
    {
        totalDisks = disks;
    }

    // Performs the initial call to moveTower to solve the puzzle.
    public void solve ()
    {
        moveTower (totalDisks, 1, 3, 2);
    }
}

continued
private void moveTower (int numDisks, int start, int end, int temp) {
    if (numDisks == 1)
        moveOneDisk (start, end);
    else
    {
        moveTower (numDisks-1, start, temp, end);
        moveOneDisk (start, end);
        moveTower (numDisks-1, temp, end, start);
    }
}

private void moveOneDisk (int start, int end) {
    System.out.println ("Move one disk from " + start + " to " + end);
}

public int mystery(int x, int y) {
    if (x == y) return 0;
    else return mystery(x-1, y) + 1;
}

If the method is called as mystery(8, 3), what is returned?
Trace the recursive calls.
A) 11
B) 8
C) 5
D) 3
E) 24
Quiz

public int mystery(int x, int y) {
    if (x == y) return 0;
    else return mystery(x-1, y) + 1;
}

If the method is called as mystery(8, 3), what is returned?

C) 5

The method computes x - y if x > y. The method works as follows: each time the method is called recursively, it subtracts 1 from x until (x == y) is becomes true, and adds 1 to the return value. So, 1 is added each time the method is called, and the method is called once for each int value between x and y.
Quiz

Calling the previous method will result in infinite recursion if which condition below is initially true?

A) \( x == y \)
B) \( x != y \)
C) \( x > y \)
D) \( x < y \)
E) \( x == 0 && y != 0 \)
Quiz

Calling the previous method will result in infinite recursion if which condition below is initially true?

A) \((x == y)\)
B) \((x != y)\)
C) \((x > y)\)
D) \((x < y)\)
E) \((x == 0 && y != 0)\)

If \((x < y)\) is true initially, then the else clause is executed resulting in the method being recursively invoked with a value of \(x - 1\), or a smaller value of \(x\), so that \((x < y)\) will be true again, and so for each successive recursive call, \((x < y)\) will be true and the base case, \(x == y\), will never be true.
Quiz

What does the following method compute? Assume the method is called initially with $i = 0$

```java
public int mystery(String a, char b, int i) {
    if (i == a.length()) return 0;
    else if (b == a.charAt(i))
        return mystery(a, b, i+1) + 1;
    else return mystery(a, b, i+1);
}
```
Quiz

What does the following method compute? Assume the method is called initially with \( i = 0 \)

```java
public int mystery(String a, char b, int i) {
    if (i == a.length()) return 0;
    else if (b == a.charAt(i))
        return mystery(a, b, i+1) + 1;
    else return mystery(a, b, i+1);
}
```

The number of times char \( b \) appears in String \( a \). The method compares each character in String \( a \) with char \( b \) until \( i \) reaches the length of String \( a \). 1 is added to the return value for each match.
Outline

Recursive Thinking
Recursive Programming
Using Recursion
Recursion in Graphics
Tiled Pictures

• Consider the task of repeatedly displaying a set of images in a mosaic
  – Three quadrants contain individual images
  – Upper-left quadrant repeats pattern

• The base case is reached when the area for the images shrinks to a certain size

• See TiledPictures.java
import java.awt.*;
import javax.swing.JApplet;

public class TiledPictures extends JApplet {
    private final int APPLET_WIDTH = 320;
    private final int APPLET_HEIGHT = 320;
    private final int MIN = 20; // smallest picture size

    private Image world, everest, goat;

    continue
public void init()
{
    world = getImage (getDocumentBase(), "world.gif");
    everest = getImage (getDocumentBase(), "everest.gif");
    goat = getImage (getDocumentBase(), "goat.gif");

    setSize (APPLET_WIDTH, APPLET_HEIGHT);
}

public void drawPictures (int size, Graphics page)
{
    page.drawImage (everest, 0, size/2, size/2, size/2, this);
    page.drawImage (goat, size/2, 0, size/2, size/2, this);
    page.drawImage (world, size/2, size/2, size/2, size/2, this);

    if (size > MIN)
        drawPictures (size/2, page);
}
continue

//--- Perform the initial call to the drawPictures method.  
public void paint (Graphics page)
{
    drawPictures (APPLET_WIDTH, page);
}
}
continue

// Perform the initial call to the drawPictures method.

public void paint(Graphics page)
{
    drawPictures(APPLET_WIDTH, page);
}

Applet started.
import javax.swing.JFrame;

public class TiledPicturesApp {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Tiled Pictures");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new TiledPicturesPanel());
        frame.pack();
        frame.setVisible(true);
    }
}

Application version of the previous applet
public class TiledPicturesPanel extends JPanel {

    private final int PANEL_WIDTH = 320;
    private final int PANEL_HEIGHT = 320;
    private final int MIN = 20;  // smallest picture size

    private BufferedImage world, everest, goat;

    public void paint(Graphics g) {

        int x = 0, y = 0;
        while (x < PANEL_WIDTH) {
            while (y < PANEL_HEIGHT) {
                world = ImageIO.read(new File("world.png"));
                g.drawImage(world, x, y, this);
                g.drawImage(everest, x, y, this);
                g.drawImage(goat, x, y, this);
                y += MIN;
            }
            y = 0;
            x += MIN;
        }
    }
}

import java.awt .*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.*;
public TiledPicturesPanel() {
    try {
        world = ImageIO.read(new File("world.gif"));
        everest = ImageIO.read(new File("everest.gif"));
        goat = ImageIO.read(new File("goat.gif"));
    } catch (IOException e) {
    }
    setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
}

continue
public void drawPictures(int size, Graphics page) {
    page.drawImage(everest, 0, size / 2, size / 2, size / 2, this);
    page.drawImage(goat, size / 2, 0, size / 2, size / 2, this);
    page.drawImage(world, size / 2, size / 2, size / 2, size / 2, this);

    if (size > MIN) {
        drawPictures(size / 2, page);
    }
}

public void paintComponent(Graphics page) {
    super.paintComponent(page);
    drawPictures(PANEL_WIDTH, page);
}

Fractals

• A fractal is a geometric shape made up of the same pattern repeated in different sizes and orientations

• The Koch Snowflake is a particular fractal that begins with an equilateral triangle

• To get a higher order of the fractal, the sides of the triangle are replaced with angled line segments

• See KochSnowflake.java
• See KochPanel.java
import javax.swing.JFrame;

public class KochSnowflakeApp {

    public static void main (String[] args) {
        JFrame frame = new JFrame("Kock Snowflake");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        frame.getContentPane().add(new KochMainPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
// Demonstrates the use of recursion in graphics.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KochMainPanel extends JPanel implements ActionListener {

private final int PANEL_WIDTH = 400;
private final int PANEL_HEIGHT = 440;

private final int MIN = 1, MAX = 9;

private JButton increase, decrease;
private JLabel titleLabel, orderLabel;
private KochPanel drawing;
private JPanel tools;

continue
public KochMainPanel()
{
    tools = new JPanel();
    tools.setLayout (new BoxLayout(tools, BoxLayout.X_AXIS));
    tools.setPreferredSize (new Dimension (PANEL_WIDTH, 40));
    tools.setBackground (Color.yellow);
    tools.setOpaque (true);

    titleLabel = new JLabel ("The Koch Snowflake");
    titleLabel.setForeground (Color.black);

    increase = new JButton (new ImageIcon ("increase.gif"));
    increase.setPressedIcon (new ImageIcon ("increasePressed.gif"));
    increase.addActionListener (this);

    decrease = new JButton (new ImageIcon ("decrease.gif"));
    decrease.setPressedIcon (new ImageIcon ("decreasePressed.gif"));
    decrease.addActionListener (this);
continue

orderLabel = new JLabel("Order: 1");
orderLabel.setForeground(Color.black);

tools.add (titleLabel);
tools.add (Box.createHorizontalStrut (40));
tools.add (decrease);
tools.add (increase);
tools.add (Box.createHorizontalStrut (20));
tools.add (orderLabel);

drawing = new KochPanel (1);

add (tools);
add (drawing);

setPreferredSize (PANEL_WIDTH, PANEL_HEIGHT);
}
// Determines which button was pushed, and sets the new order
// if it is in range.
//-----------------------------------------------------------------
public void actionPerformed(ActionEvent event) {
    int order = drawing.getOrder();
    
    if (event.getSource() == increase)
        order++;
    else
        order--;

    if (order >= MIN && order <= MAX)
    {
        orderLabel.setText("Order: "+ order);
        drawing.setOrder(order);
        repaint();
    }
}

// Determines which button was pushed, and sets the new order
// if it is in range.

public void actionPerformed(ActionEvent event) {
    int order = drawing.getOrder();
    if (event.getSource() == increase)
        order++;
    else
        order--;
    if (order >= MIN && order <= MAX) {
        orderLabel.setText("Order: " + order);
        drawing.setOrder(order);
        repaint();
    }
}
Koch Snowflakes

\[
\begin{align*}
\langle x_1, y_1 \rangle & \quad \text{Becomes} \quad \langle x_5, y_5 \rangle \\
\langle x_5, y_5 \rangle & \quad \text{Becomes} \quad \langle x_4, y_4 \rangle \\
\langle x_4, y_4 \rangle & \quad \text{Becomes} \quad \langle x_3, y_3 \rangle \\
\langle x_3, y_3 \rangle & \quad \text{Becomes} \quad \langle x_2, y_2 \rangle \\
\langle x_2, y_2 \rangle & \quad \text{Becomes} \quad \langle x_1, y_1 \rangle
\end{align*}
\]

\[= \sqrt{\frac{3}{6}} \cdot |P_5 - P_1|\]
import java.awt.*;
import javax.swing.JPanel;

public class KochPanel extends JPanel {
    private final int PANEL_WIDTH = 400;
    private final int PANEL_HEIGHT = 400;

    private final double SQ = Math.sqrt(3.0) / 6;

    private final int TOPX = 200, TOPY = 20;
    private final int LEFTX = 60, LEFTY = 300;
    private final int RIGHTX = 340, RIGHTY = 300;

    private int current; // current order

    continue
// Draws the fractal recursively. The base case is order 1 for
// which a simple straight line is drawn. Otherwise three
// intermediate points are computed, and each line segment is
// drawn as a fractal.

public void drawFractal (int order, int x1, int y1, int x5, int y5,
            Graphics page)
{
    int deltaX, deltaY, x2, y2, x3, y3, x4, y4;

    if (order == 1)
        page.drawLine (x1, y1, x5, y5);
    else
    {
        deltaX = x5 - x1; // distance between end points
        deltaY = y5 - y1;

        x2 = x1 + deltaX / 3; // one third
        y2 = y1 + deltaY / 3;

        x3 = (int) ((x1+x5)/2 + SQ * (y1-y5)); // tip of projection
        y3 = (int) ((y1+y5)/2 + SQ * (x5-x1));
    }
}
continue
\texttt{continue}

\begin{verbatim}
x4 = x1 + deltaX \times 2/3;  // two thirds
y4 = y1 + deltaY \times 2/3;

drawFractal (order-1, x1, y1, x2, y2, page);
drawFractal (order-1, x2, y2, x3, y3, page);
drawFractal (order-1, x3, y3, x4, y4, page);
drawFractal (order-1, x4, y4, x5, y5, page);
\}
\}

//-----------------------------------------------------------------
// Performs the initial calls to the drawFractal method.
//-----------------------------------------------------------------
public void paintComponent (Graphics page)
{
    super.paintComponent (page);

    page.setColor (Color.green);

    drawFractal (current, TOPX, TOPY, LEFTX, LEFTY, page);
drawFractal (current, LEFTX, LEFTY, RIGHTX, RIGHTY, page);
drawFractal (current, RIGHTX, RIGHTY, TOPX, TOPY, page);
}
\texttt{continue}
\end{verbatim}
```java
// Sets the fractal order to the value specified.
public void setOrder (int order)
{
    current = order;
}

// Returns the current order.
public int getOrder ()
{
    return current;
}
```
Summary

• Chapter 12 has focused on:
  – thinking in a recursive manner
  – programming in a recursive manner
  – the correct use of recursion
  – recursion examples
Exercise

• Write a recursive definition of a valid Java identifier (see Chapter 1). Imagine that a letter is either a letter of the English alphabet or $ or _

Identifier

Java Letter

Java Letter

Java Digit
Exercise

• Write a recursive definition of a valid Java identifier (see Chapter 1). Imagine that a Letter is either a letter of the English alphabet or $ or _

A Java-Identifier is a: Letter

or a: Java-Identifier followed by a Letter

or a: Java-Identifier followed by a digit
Exercise

• Write a recursive definition of $i \times j$ (integer multiplication), where $i > 0$.

• Define the multiplication process in terms of integer addition.
  – For example, $4 \times 7$ is equal to $7$ added to itself $4$ times.
Exercise

• Write a recursive definition of $i \times j$ (integer multiplication), where $i > 0$.

• Define the multiplication process in terms of integer addition. For example, $4 \times 7$ is
  • equal to 7 added to itself 4 times.

• $1 \times j = j$

• $i \times j = j + (i-1) \times j$ for $i > 1$
Exercise

• Modify the method that calculates the sum of the integers between 1 and N shown in this chapter. Have the new version match the following recursive definition:
  – The sum of 1 to N is the sum of 1 to (N/2) plus the sum of (N/2 + 1) to N.

• Trace your solution using an N of 7.
public int sum(int n1, int n2) {
    int result;
    if (n2 - n1 == 0) {
        result = n1;
    } else {
        int mid = (n1 + n2) / 2;
        result = sum(n1, mid) + sum(mid + 1, n2);
    }
    return result;
}
Trace

```
sum(1,7)
  ┌───────┐
  │      │
  │      │
sum(1,4) ┘───┐  sum(5,7)
  │      │
  │      │
sum(1,2) ┘───┘  sum(3,4) ┘───┘
  │      │      │      │
  │      │      │      │
sum(1,1) ┘───┘  sum(2,2) ┘───┘
```

sum(3,3)  sum(4,4)  sum(5,6)  sum(7,7)
  │      │      │      │
  │      │      │      │
sum(5,5)  sum(6,6)
Exercise

• Write a(nother) recursive method to reverse a string.

• Use the following String methods
  – charAt(int n) : char
  – substring(int beginIndex, int endIndex) : String

• Implement a procedure that concatenates the last character of the input string with the reverse of the string composed by the first N-1 characters of the string (N is the length of the string).
public String reverse(String text) {
    String result = text;
    if (text.length() > 1) {
        result = text.charAt(text.length() - 1) + reverse(text.substring(0, text.length() - 1));
    }
    return result;
}
Tower of Hanoi

• Produce a chart showing the number of moves required to solve the Towers of Hanoi puzzle using the following number of disks: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10

• Write a recursive definition of the formula giving \( \text{Moves}(n) \), the number of moves required to solve the Hanoi tower of \( n \) disks
## Solution

<table>
<thead>
<tr>
<th>Disks</th>
<th>Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
</tr>
<tr>
<td>8</td>
<td>255</td>
</tr>
<tr>
<td>9</td>
<td>511</td>
</tr>
<tr>
<td>10</td>
<td>1023</td>
</tr>
</tbody>
</table>

Moves(1) = 1

\[
Move(n) = Move(n-1) + 1 + Move(n-1)
\]
Kock Snowflake

• How many line segments are used to construct a Koch snowflake of order N? Produce a chart showing the number of line segments that make up a Koch snowflake for orders 1 through 7.

• Give the general formula \( \text{Segments}(n) = ? \)
Solution

<table>
<thead>
<tr>
<th>Order</th>
<th>Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4*3 = 12</td>
</tr>
<tr>
<td>3</td>
<td>4<em>4</em>3 = 48</td>
</tr>
<tr>
<td>4</td>
<td>4<em>4</em>4*3 = 192</td>
</tr>
<tr>
<td>5</td>
<td>4<em>4</em>4<em>4</em>3 = 768</td>
</tr>
<tr>
<td>6</td>
<td>4<em>4</em>4<em>4</em>4*3 = 3072</td>
</tr>
<tr>
<td>7</td>
<td>4<em>4</em>4<em>4</em>4<em>4</em>3 = 12288</td>
</tr>
</tbody>
</table>

Segments(1) = 3
Segments(n) = Segments (n-1) * 4

Segments(n) = 3*4^{n-1}
Exercise

• Give the value of mistery2(3):

```java
public static String mistery2(int n) {
    if (n <= 0) {
        return "";
    }
    return mistery2(n - 3) + n + mistery2(n - 2) + n;
}
```
misery2(3)

misery2(0)

misery2(1)

misery2(-2)

misery2(-1)

return ""

return ""

return ""+1+""+1

return ""+3+""11""+3 = ""3113""
Exercise

• Criticize the following recursive function:

```java
public static String mistery3(int n) {
    String s = mistery3(n - 3) + n + mistery3(n - 2) + n;
    if (n <= 0)
        return "";
    return s;
}
```