Objectives:
1. To understand the concept of recursive definition
2. To understand the difference between iteration and recursion
3. To understand when recursion is appropriate

Reading Assignment:
1. Nell/Chip/Mark, Chapter 13

Concepts:
1. Recursion

Announcements:
1. Final Exam

Handout:
1. Final Exam Study Guide
• Recursion
  • Recursive Call – A method call in which the method being called is the same as the one making the call
  • Base Case – the case for which the solution can be stated nonrecursively
  • General Case – the case for which the solution is expressed in terms of a smaller version of itself, also known as the recursive case
  • Recursive Algorithm – a solution that is expressed in terms of 1) smaller instances of itself and 2) a base case
  • Infinite Recursion – the situation in which a method calls itself over and over endlessly
• How recursion works
  • Factorial example

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

public class FactorialTest extends JApplet {
    JTextArea outputArea;

    // create GUI and calculate factorials of 0-10
    public void init() {
        outputArea = new JTextArea();
        Container container = getContentPane();
        container.add(outputArea);

        // calculate the factorials of 0 through 10
        for (long counter = 0; counter <= 10; counter++) {
            outputArea.append(counter + "! = " + factorial(counter) + "\n" );
        }
    }

    // recursive declaration of method factorial
    public long factorial( long number ) {
        // base case
        if ( number <= 1 )
            return 1;

        // recursive step
        else
            return number * factorial( number - 1 );
    }

    // end class FactorialTest
```
Objectives:
1. To be familiar and have practice with the use of recursion

Fill in the missing code. The missing part is the recursive implementation of the Fibonacci series.

```java
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class FibonacciTest extends JApplet implements ActionListener {

    JLabel numberLabel, resultLabel;
    JTextField numberField, resultField;

    // set up applet's GUI
    public void init() {
        // obtain content pane and set its layout to FlowLayout
        Container container = getContentPane();
        container.setLayout( new FlowLayout() );

        // create numberLabel and attach it to content pane
        numberLabel = new JLabel( "Enter an integer and press Enter" );
        container.add( numberLabel );

        // create numberField and attach it to content pane
        numberField = new JTextField( 10 );
        container.add( numberField );

        // register this applet as numberField’s ActionListener
        numberField.addActionListener( this );

        // create resultLabel and attach it to content pane
        resultLabel = new JLabel( "Fibonacci value is" );
        container.add( resultLabel );

        // create numberField and attach it to content pane
        numberField = new JTextField( 15 );
        resultField.setEditable( false );
        container.add( numberField );

        // obtain user input and call method fibonacci
        public void actionPerformed( ActionEvent event ) {
            long number, fibonacciValue;

            // obtain user’s input and convert to long
```
number = Long.parseLong( numberField.getText() );

showStatus( "Calculating ..." );

// calculate fibonacci value for number user input
fibonacciValue = fibonacci( number );

// indicate processing complete and display result
showStatus( "Done." );
resultField.setText( Long.toString( fibonacciValue ) );

} // end method actionPerformed

// recursive declaration of method fibonacci
public long fibonacci( long n )
{

// base case
if ( n == 0 || n == 1 )
    return n;

// recursive step
else
    return fibonacci( n - 1 ) + fibonacci( n - 2 );

} // end method fibonacci

} // end class FibonacciTest

Answer:

// recursive declaration of method fibonacci
public long fibonacci( long n )
{
    // base case
    if ( n == 0 || n == 1 )
        return n;

    // recursive step
    else
        return fibonacci( n - 1 ) + fibonacci( n - 2 );

} // end method Fibonacci
• Testing phase of Software Life Cycle
• Debugging
• Atomic Data Types
• Composite Data Types
  • Unstructured
  • Structured
• Arrays
  • Implementation
  • passed as an argument
  • role of arrays of Objects
• Lists
  • basic operations
  • sequential and binary search
• Inheritance
• Polymorphism
• Scope
• Basic File Input/Output
• Exception Handling
  • possible uses
  • try block
• Recursion
  • advantages
  • disadvantages
  • when to use recursion

• Recursion vs. Iteration
• Arrays vs. Lists
• Atomic vs. Composite Data Types