

Java Examples

```
import jll;
import util;

class driver{

    /* Purpose: Print fibonacci value and factorial of input
    * Precond: None
    * Postcond: Outputs to standard out
    */
    public static void main( String argv[] ) {

        char itemInput;          // Menu choice
        Character charInput;
        int iValue;              // Initial Value
        Integer intInput;
        int finalValue = 0;      // Final Value returned from functions

        for ( ; ; ) {
            // Get input from the user
            System.out.println("Enter the number of a function shown below:");
            System.out.println("(1) Fibonnaci Recursive");
            System.out.println("(2) Fibonnaci Iterative - For Loop");
            System.out.println("(3) Fibonnaci Iterative - While Loop");
            System.out.println("(4) Fibonnaci Iterative - Do While Loop");
            System.out.println("(5) Factorial Recursive");
            System.out.println("(6) Factorial Iterative");
            System.out.println("(Q) Quit");
            charInput = util.readCharacter();

            // Convert to char value
            itemInput = charInput.charValue();

            if( itemInput == 'q' || itemInput == 'Q' ) {
                System.exit(0);
            }

            System.out.println("Enter a number:");

            // Figure out fibonacci and factorial of input
            intInput = util.readLineInteger();

            // convert to integer base
            iValue = intInput.intValue();
        }
    }
}
```

```

// Call appropriate method

switch( itemInput ) {
    case '1': finalValue = j11.fibonacciRecursive(iValue);
              System.out.print("Fibonacci (recursive) of "
                              + iValue + " is: ");
              break;

    case '2': finalValue = j11.fibonacciFor(iValue);
              /*** insert print statement here ***/
              break;

    case '3': finalValue = j11.fibonacciWhile(iValue);
              /*** insert print statement here ***/
              break;

    case '4': finalValue = j11.fibonacciDoWhile(iValue);
              /*** insert print statement here ***/
              break;

    case '5': finalValue = j11.factorialRecursive(iValue);
              /*** insert print statement here ***/
              break;

    case '6': finalValue = j11.factorialIterative(iValue);
              System.out.print("Factorial (Iterative) of "
                              + iValue + " is: ");
              break;

    default: System.out.println("Invalid choice");

} // end switch statement

System.out.println(finalValue);
System.out.println();

} // end for loop

} // end of main
}

```

Object Pattern 2 Collections

Problem: Need to maintain a set of objects, often state objects, and provide a common interface to other objects that use the collection.

Examples: an employee roster, an appointment organizer, a library of books or CDs.

Prototype class structure and interface:

```
class object_name
{
    // Private or protected variables to hold and/or access
    // the
    // contained objects using an array, linked list, tree,
    // etc.
    ...

    // Constructors
    ...

    // Accessor methods -- including some way to access
    // each
    // member of the collection
    ...

    // Modifier methods -- including ways to add or delete
    // members of the collection
    ...
}
```

Debugging strategies:

Similar problems and requirements as with state objects. Each possible internal structure has its own specialized pitfalls. Again, a way to visualize the internal state is crucial.

Object Pattern 3 Iterations

Problem: Need to access all of the objects in collection (typically as part of a *Process All Items* or *Linear Search* procedural pattern).

Examples: draw all of the items in a collection of graphical objects; find which graphical object in a collection was the target of a mouse click.

Prototype interface:

This pattern is not defined by a class, but by additions to the interface of some existing class. Here are two alternatives:

(1) Use an accessor method to provide indexed access to the objects in a collection:

```
public object_type at (int i) {  
    // return a pointer to the ith object in the collection;  
    // the ordering may or may not be externally meaningful  
    . . .  
}
```

(2) Provide a group of accessor methods that give up the objects in a collection in sequence:

```
public object_type first ()  
    // Initializes an iteration and returns a pointer to  
    // the "first" of the objects in the collection;  
  
public object_type next ()  
    // return the next of the objects in the collection;  
    // current iteration state must be remembered internally  
  
public boolean done ()  
    // returns true after all objects in the collection  
    // have been returned by first and next
```

Iterations in Java -- Enumerations

```
public interface Enumeration {
    public boolean hasMoreElements();
    public Object nextElement() throws NoSuchElementException;
}
```

```
public static void printVec(String msg, Vector vec) {
    if (msg != null)
        System.out.println(msg);
    if (vec.isEmpty())
        System.out.println("Empty vector");
    else {
        Enumeration e = vec.elements();
        while (e.hasMoreElements())
            System.out.println("\t" + e.nextElement());
    }
}
```

```
public class Vector implements Cloneable {
    protected Object[] elementData;
    protected int elementCount;
    protected int capacityIncrement;
    public Vector(int initialCapacity, int capacityIncrement);
    public Vector(int initialCapacity);
    public Vector();
    public final String toString();
    public Object clone();
    public final Object elementAt(int index)
        throws IndexOutOfBoundsException;
    public final void setElementAt(Object obj, int index)
        throws IndexOutOfBoundsException;
    public final Object firstElement()
        throws NoSuchElementException;
    public final Object lastElement()
        throws NoSuchElementException;
    public final void addElement(Object obj);
    public final void insertElementAt(Object obj, int index)
        throws IndexOutOfBoundsException;
    public final boolean removeElement(Object obj);
    public final void removeElementAt(int index)
        throws IndexOutOfBoundsException;
    public final void removeAllElements();
    public final boolean isEmpty();
    public final int size();
    public final void setSize(int newSize);
    public final int capacity();
    public final void ensureCapacity(int minCapacity);
    public final void trimToSize();
    public final void copyInto(Object anArray[])
        throws IndexOutOfBoundsException;
    public final Enumeration elements();
    public final boolean contains(Object elem);
    public final int indexOf(Object elem);
    public final int indexOf(Object elem, int index)
        throws IndexOutOfBoundsException;
    public final int lastIndexOf(Object elem);
    public final int lastIndexOf(Object elem, int index)
        throws IndexOutOfBoundsException;
}
```

```
final
class VectorEnumerator implements Enumeration {
    Vector vector;
    int count;

    VectorEnumerator(Vector v) {
        vector = v;
        count = 0;
    }

    public boolean hasMoreElements() {
        return count < vector.elementCount;
    }

    public Object nextElement() {
        synchronized (vector) {
            if (count < vector.elementCount) {
                return vector.elementData[count++];
            }
        }
        throw new NoSuchElementException("VectorEnumerator");
    }
}
```