Solutions to Laboratory Exercise 3

Recursion

1. One of the following methods:

   (a)
```
public static int factorial(int n) {
// ------------------------------------------------
// Computes the factorial of a nonnegative integer.
// Precondition: n>=0.
// Postcondition: Returns the factorial of n.
// ------------------------------------------------
  if (n==0) {
     return 1;
  }
  else {
     return n*factorial(n-1);
  }  // end if
} // end factorial
```

   (b)
```
public static void writeBackward(String s,int size) {
// ------------------------------------------------
// Writes a character string backward.
// Precondition: s contains size characters and
// size>=0.
// Postcondition: s is written backward, but remains
// unchanged.
// ------------------------------------------------
  if (size>0) {
    System.out.println(s.substring(size-1, size));
    writeBackward(s, size-1);
  } // end if
  // base case: size==0, do nothing
} // end writeBackward
```

   (c)
```
public static int fibonacciTerm(int n) {
// ----------------------------------------
// Computes the nth term in the Fibonacci sequence.
// Precondition: n is a positive integer.
// Postcondition: Returns value of the nth
//                Fibonacci term.
// ----------------------------------------
  if (n<=2) {
     return 1;
  }
  else {
    return fibonacciTerm(n-1)+fibonacciTerm(n-2);
  } // end if
} // end fibonacciTerm
```

   Note: In order to run these methods, you need to define a class in which the methods are called.

   For example, you may construct a main method which allows you to input a positive integer as the argument of the method factorial from the keyboard, and allows you to view the result:

```
import java.io.*;
```

```
public class myfactorial {

   public static void main(String[] args) throws IOException {
     BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
     System.out.println("Please enter a number (>=0):");
     String x=in.readLine();
     int n=Integer.parseInt(x);
     System.out.println(n+"!="+factorial(n));
   }

   public static int factorial(int n) {
   // ------------------------------------------------
   // Computes the factorial of a nonnegative integer.
   // Precondition: n>=0.
   // Postcondition: Returns the factorial of n.
   // ------------------------------------------------
     if (n==0) {
        return 1;
     }
     else {
        return n*factorial(n-1);
     }  // end if
   } // end factorial
} // end myfactorial

import java.io.*;

public class classWriteBackward{

   public static void main(String[] args) throws IOException {
     BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
     System.out.println("Please enter a string:");
     String s=in.readLine();
     System.out.println("The string backward is:");
     writeBackward(s,s.length());
     System.out.println();
   }

   public static void writeBackward(String s,int size) {
   // ------------------------------------------------
   // Writes a character string backward.
   // Precondition: s contains size characters and
   // size>=0.
   // Postcondition: s is written backward, but remains
   // unchanged.
   // ------------------------------------------------
      if (size>0) {
        System.out.print(s.substring(size-1, size));
        writeBackward(s, size-1);
      } // end if
      // base case: size==0, do nothing
   } // end writeBackward

} // end classWriteBackward
```

2. One of the following methods:

   (a)
```
public static int factorial(int n) {
// ------------------------------------------------
// Computes the factorial of a nonnegative integer.
// Precondition: n>=0.
// Postcondition: Returns the factorial of n.
// ------------------------------------------------
  if (n==0) {
     return 1;
  }
  else {
     /*Recursive Call!*/
     return factorial(n-1);
  }  // end if
} end factorial
```

   (b)
```
public static void writeBackward(String s,int size) {
// ------------------------------------------------
// Writes a character string backward.
// Precondition: s contains size characters and
// size>=0.
// Postcondition: s is written backward, but remains
// unchanged.
// ------------------------------------------------
  if (size>0) {
    System.out.println(s.substring(size-1, size));
    /*Recursive Call!*/
    writeBackward(s, size-1);
  } // end if
  // base case: size==0, do nothing
} // end writeBackward
```

   (c)
```
public static int fibonacciTerm(int n) {
// ----------------------------------------
// Computes the nth term in the Fibonacci sequence.
// Precondition: n is a positive integer.
// Postcondition: Returns value of the nth
//                Fibonacci term.
// ----------------------------------------
  if (n<2) {
     return 1;
  }
  else {
    /*Recursive Call(s)!*/
    return fibonacciTerm(n-1)+fibonacciTerm(n-2);
  } // end if
} // end fibonacciTerm
```

3. Replace each /*Recrusive Call(s)!*/ comment above by a System.out.println() statement.

4. One of the followings (iterative methods):

   (a)
```
import java.io.*;

public class myfactorial {

   public static void main(String[] args) throws IOException {
     BufferedReader in =
```

```
            new BufferedReader(new InputStreamReader(System.in));
          System.out.println("Please enter a number (>=0):");
          String x=in.readLine();
          int n=Integer.parseInt(x);
          System.out.println(n+"!="+factorial(n));
        }

        public static int factorial(int n) {
        // -----------------------------------------------
        // Computes the factorial of a nonnegative integer.
        // Precondition: n>=0.
        // Postcondition: Returns the factorial of n.
        // -----------------------------------------------
           int result=1;
           for (int i=1; i<=n; i++) {
              result = result*i;
           } // end for
           return result;
        } // end factorial
      } // myfactorial
```

(b)

```
      import java.io.*;

      public class classWriteBackward{

        public static void main(String[] args) throws IOException {
          BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
          System.out.println("Please enter a string (length>=1):");
          String str=in.readLine();
          System.out.println("The string backward is:");
          writeBackward(str,str.length());
          System.out.println();
        }

        public static void writeBackward(String s,int size) {
        // -----------------------------------------------
        // Writes a character string backward.
        // Precondition: s contains size characters and
        // size>=0.
        // Postcondition: s is written backward, but remains
        // unchanged.
        // -----------------------------------------------
           for (int i=size-1; i>=0; i--) {
             System.out.print(s.charAt(i));
           } // end for
        } // end writeBackward

      } // end classWriteBackward
```

(c)

```
      import java.io.*;

      public class myfibonacciTerm {

        public static void main(String[] args) throws IOException {
          BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
          System.out.println("Please enter a number (>=1):");
```

```
    String x=in.readLine();
    int n=Integer.parseInt(x);
    System.out.println("The "+n+"th Fibonacci term is: "
                       +iterativeFibonacciTerm(n));
  }

  public static int iterativeFibonacciTerm(int n) {
  // ----------------------------------------
  // Computes the nth term in the Fibonacci sequence.
  // Precondition: n is a positive integer.
  // Postcondition: Returns value of the nth
  //                Fibonacci term.
  // ----------------------------------------
// Iterative solution to the Fibonacci problem.
  // initialise base cases:
  int previous=1;
  int current=1;
  int next=1;

// compute next Fibonacci term when n>=3
  for (int i=3; i<=n; i++) {
    next=current+previous;
    previous=current;
    current=next;
  } // end for
  return next;
  } // end iterativeFibonacciTerm
} // myfibonacciTerm
```

5. (Optional) [8]

[8] *No solution available, open for discussion.*