COM S 211/ENGRD 211 **March 6, 2003**
Prelim 1 **7:30 PM – 9:00 PM**

Information:

Name (clearly *print* last, first, middle): ______________________________

Net ID: ____________________

CU ID: ____________________

I have followed the rules of academic integrity on this exam (sign): ______________________________

Instructions:

Failure to follow any instruction may result in a point deduction on your exam:
- Turn off all cell phones, beepers, pagers, and any other devices that will interrupt the exam.
- Remove all calculators, reference sheets, or any other material. This test is closed book.
- Fill out the information at the top of this exam.
- Skim the entire exam before starting any of the problems.
- Read each problem completely before starting it.
- Solve each problem using Java, except where indicated.
- Use only the given code in each problem and follow specifications on whether or not to use the API.
- Write your solutions directly on the test using blue/black pen or pencil. Clearly indicate which problem you are solving. You may write on the back of each sheet. If you need scrap paper, ask a proctor.
- Provide only one statement, expression, value, or comment per blank!
- Do not alter, add, or remove any code that surrounds the blanks and boxes.
- Do not supply multiple answers. If you do so, we will choose which one to grade.
- Follow good style! When possible, keep solutions general, avoid redundant code, use descriptive variables, use named constants, indent substructures, avoid breaking out of loops, and maintain other tenets of programming philosophy.
- Comment each control structure, major variable, method, and class (if used), briefly.
- Do not spend too much time on any single question and budget your time based on the amount of points.
- Do not work on bonus problems until you have thoroughly proofread all required (core-point) problems!
- Figure out any problem yourself before raising your hand so that we can avoid disturbing people in cramped rooms.

Core Points:

```
    1. ________   (15 points) __________

    2. ________   ( 5 points) __________

    3. ________   (10 points) __________

    4. ________   (15 points) __________

    5. ________   (30 points) __________

    6. ________   (25 points) __________

Total: ________ /(100 points) __________
```

Bonus Points:

```
 misc: ________ /( 16 points) __________

 eval: ________ /(  6 points) __________

Total: ________ /( 22 points) __________
```

Problem 1 [15 points] *General Concepts*

Answer the following questions. Questions with blanks require only one word. Be concise and clear.

1a [1 point] The correct spelling of your COM S/ENGRD 211 instructor's last name is __________________________.

1b [2 points] What is a data structure?

1c [2 points] What is the difference between a list and a set?

1d [1 point] Why is an array not a dynamic data structure?

1e [1 point] Give one example of a dynamic data structure: ______________________ .

1f [1 point] Because of ______________ binding, Java uses the actual type of an object to access a method.

1g [1 point] Because of ______________ binding, Java uses the reference type to access a field.

1h [2 points] Write the output for the following code for when the user runs the program with this command line:

```
> java args args "args" \"args\"
```

```
public class args {
   public static void main(String[] args) {
      for ( int arg = 0 ; arg < args.length ; arg++ )
         System.out.print( args[arg] +" ");
      }
}
```

1i [2 points] Distinguish between a shallow and deep clone. No answers about sheep or cults…just data structures.

1j [2 points] Distinguish between *height* and *depth* in a tree data structure.

Problem 2 [5 points] *Induction*

Use induction to prove that $1^2 + 2^2 + 3^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$ is *true*. For full credit, you must clearly state the base case, induction hypothesis, inductive step, and conclusion in your proof.

Problem 3 [10 points] *Recursion*

Using *recursion*, complete method **`mod(int n,int d)`**, which returns the remainder, or ***modulus***, of $n \div d$. Note that we do *not* imply integer division of *n* by *d*! For example, **`mod(11,4)`** should return 3, not zero. You may *not* use the **`Math`** class, division (**`/`**), multiplication (**`*`**), exponentiation (namely, **`Math.pow`**, which we already ruled out), the modulus operator (**`%`**), or any helper methods and classes. Assume that $n \geq 0$ and $d \geq 1$.

```
public class Problem3 {

   public static void main(String[] args) {

      System.out.println(mod(11,4)); // output: 3
      System.out.println(mod(10,5)); // output: 0
      System.out.println(mod(1,7));  // output: 1

   }

   // Return remainder of n divided by d. See problem specifications above:
      public static int mod(int n, int d) {
```

```
      } // Method mod

} // Class Problem3
```

Problem 4 [15 points] *Inheritance*, *Subtyping, Fun With Java!*

4a [12 points] In the box on the next page, write the output that the following program will generate. Hint: There are 7 output values.

4b [3 points] On the next page, answer this question: Would removing the `//` in front of the statement `r4.print(1,2)` cause at an error when recompiling the program? Why or why not?

```
public class Problem4 {
   public static void main(String[] args) {
   A r1 = new B(1);
   A r2 = new C();
   B r3 = (B) r2;
   System.out.println(r3.y);
   I1 r4 = new C();
   // r4.print(1,2)
   }
}

abstract class A {
   public int x = 3;
   public int y = x;
   public A() { print(x+2); }
   public void print() { System.out.println(x); }
   public void print(int x) { print(); }
}

class B extends A {
   public int y = x;
   public B(int x) { x = this.x; print(x+3); }
   public B() { }
   public void print() { super.print(); }
   public void print(int x) { System.out.println(x+1); }
   private void print(int x, int y) { print(x*y); }
}

class C extends B implements I1 {
   C() { super(3); }
   public void print(int x, int y) { print(x-y); }
   public void m1() { }
}

interface I1 { }
```

Problem 4a output:

Problem 4b answer:

Problem 5 [30 points] *Singly-Linked Lists, Building Lists, Traversing Lists*

Background: A ***circular linked list*** is a linked list for which the tail points to the head. To create such a list, you can create a regular singly linked list, starting with the head. By making that head the next element of the tail, you have formed a circular list. By maintaining a reference to the original head, you have an ***entry node*** that links to the entire circular list.

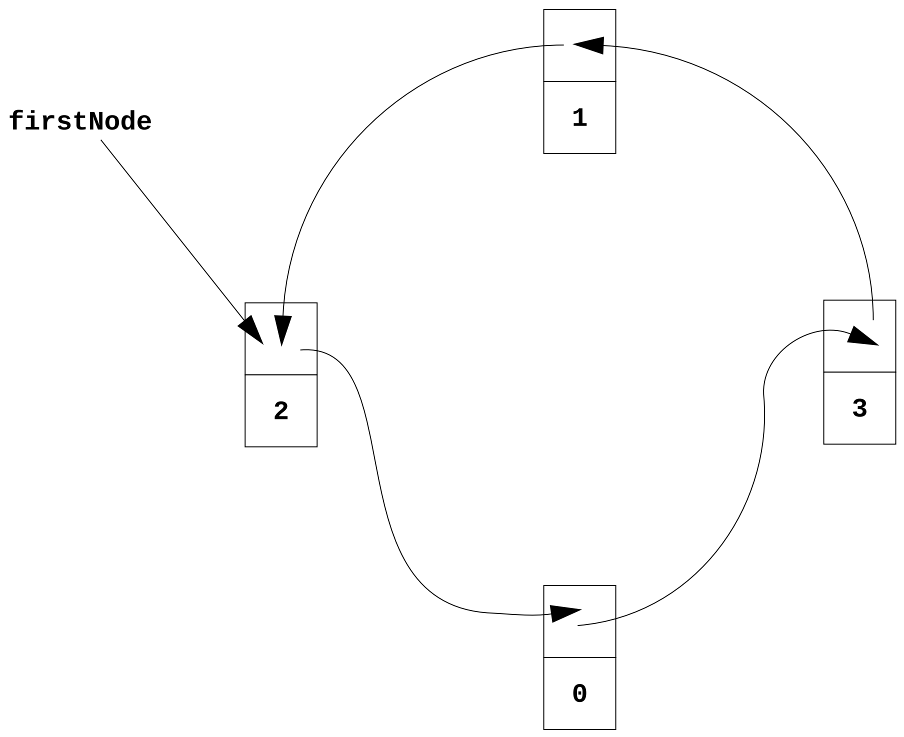
Problem: You need to complete class **`Problem5`** by completing two methods:

- **`createCircle(int nodes)`**. Given a user-input number of **`nodes`**, this method creates a circular linked list composed of **`nodes`** number of nodes with an entry node called **`firstNode`**, which is a class variable in **`Problem5`**.
- **`searchCircle()`**. This method searches for a node with the stored value of **`0`** in the circular linked list that is referenced by **`firstNode`**. If a node with **`0`** is found, the method returns **`true`**. Otherwise, it returns **`false`**. The method assumes the list contains at least one element.

Class **`Problem5`** uses class **`Node`**, which uses class **`MyMath`**. These classes are written at the end of the problem. No credit will be given for using arrays or any API data structure (e.g., **`Vector`**s and **`ArrayList`**s).

Example:

If the user runs the program for 4 nodes as `>` **`java Problem5 4`**, the program might generate the values that are shown in the circular list below. Since at least one node contains **`0`**, the program would return **`true`**.



The problem continues on the next page.

```
public class Problem5 {

   public static Node firstNode;                // entry node into circular list

   public static void main(String[] args) {
      createCircle(Integer.parseInt(args[0]));  // create list of input # of nodes
      System.out.println(searchCircle());       // report search of circle for 0
   }

   // Create a circular linked list with user-input number of nodes.
   // List uses firstNode as the first node to start the list:
   public static void createCircle(int nodes) {



   } // Method createCircle
```

The problem continues on the next page.

```
   // Search the circular linked list for the integer 0.
   // Return true if found and false if not found.
   // Assume the list is non-empty with at least one node:
   public static boolean searchCircle() {



   } // Method searchCircle

} // Class Problem5

class Node {
   public final int value=MyMath.randInt(0,4);
   public Node next;
} // Class Node

class MyMath {
   // Return a random int between low and high, inclusive:
   public static int randInt(int low, int high) {
      return (int) (Math.random()*(high-low+1)) + (int) low;
   }
} // Class MyMath
```

Problem 6 [25 points] *Trees*, *Expression Parsing*

Background: A ***pretty simple expression*** (PSE) has the following grammar:

- $E \rightarrow int$
- $E \rightarrow (E + E)$

Problem: You will complete a program that parses a user-input PSE into a tree and then prints the contents of that tree, where each operator (+) is written *between* its operands. To do so, finish these methods:

- **`makeTree`**. This method stores the **`root`** of the expression tree and controls the parsing of the PSE.
- **`toString`**. You need to complete this method in each class that implements **`INode`**. Depending on the class, **`toString`** returns a description of the node in the form of a **`String`**. By printing the tree in **`main`**, your program will *print the entire tree*, which gives the desired output, as we have demonstrated in the example session.

Additional Specifications:

- Assume that the user enters only *one* completely legal PSE at the command-window. Do not check for errors.
- You may need to use **`CS211In`**'s **`check()`**, **`getOp()`**, and/or **`getInt()`** methods to process the input.
- All nodes in the tree must implement the **`INode`** interface, which we have also provided.

Example Session:

```
> java PSE
Enter a PSE: ( ( 1 + 2 ) + ( 3 + ( 4 + 5 ) ) )
1+2+3+4+5
```

Reminder of `CS211In`:

```
interface CS211InInterface {
   int INTEGER = -1, WORD = -2, OPERATOR = -3, EOF = -4;
   int peekAtKind();        // returns "type" (int!) of token w/o "eating" it
   int getInt();            // reads an int and returns it ; else complains
   String getWord();        // reads a word and returns it ; else complains
   char getOp();            // reads an op  and returns it ; else complains
   boolean check(char c);   // is the next thing c?
   void pushBack();         // back up by one token
}
```

```
public class PSE {
   private static CS211InInterface fin = new CS211In();
   public static void main(String[] args) {
      System.out.print("Enter a PSE: ");
      System.out.println(new ParseTree(fin));
      fin.close();
   }
} // Class PSE

interface INode {
   public String toString();
}
```

The problem continues on the next page.

```
class ParseTree implements INode {
   private INode root;
   public ParseTree(CS211InInterface fin) {
      root = makeTree(fin);
   }
   // Build a PSE:
   private INode makeTree(CS211InInterface fin) {
      switch( fin.peekAtKind() ) {
```

```
      } // end switch
      return null; // return null if something went wrong (no PSE was parsed)

   } // Method makeTree
   // Return String of entire tree using root.
   // If tree is null, return empty string:
   public String toString() {









   } // Method toString()
} // Class ParseTree

class AddNode implements INode {
   private INode leftExpr;
   private INode rightExpr;
   public AddNode(INode leftExpr, INode rightExpr) {
      this.leftExpr=leftExpr;
      this.rightExpr=rightExpr;
   }
   // Return String description of AddNode:
   public String toString() {







   } // Method toString
} // Class AddNode

class IntNode implements INode {
   private int value;
   public IntNode(int value) {
      this.value = value;
   }
   // Return String description of value:
   public String toString() {






   } // Method toString
} // Class IntNode
```

Bonus: Do not work on these problems until you have thoroughly finished all core-point (required) problems!

B1) [1 bonus point] What is the answer to this question? ______________

B2) [1 bonus point] What does *UoS* stand for? You must correctly spell it for full credit.

B3) [9 bonus points] Correctly spell the last name of each TA for this course.

B4) [1 bonus point] What is the *complete* title of the book that DIS refers to as *DS&SD*?

B5) [1 bonus point] Which consultant(s) has (have) the most number of hours according to the website?

B6) [3 bonus points] What is the correct output for the following program? Yes, it really does work.

```
class X {
   int X;
   X(int X) { this(X,X++); this.X=X; }
   X(int X, int XX) { this.X+=XX+=this.X+X++; }
   int X(int X) { X(this); return X; }
   X X(X X) { X.X+=++x.XX; return new X(X.X); }
} // class X

public class x {
   static int XX;
   public static void main(String[] XXX) {
      X X = new X( (new X(++x.XX)).X );
      X(X.X(X.X));
      X(X.X(X).X);
      X(X(X.X(X)));
   } // method main

   static int X(X X) {return X.X;}
   public static void X(int X) {System.out.println(X);}
} // class x

// Output1: ___________

// Output2: ___________

// Output3: ___________
```

Bonus: [6 bonus points] Mid-Semester Evaluation.

You may do the following evaluation after you have finished writing and checking your prelim. If you run out of time, we will give you extra time *after* the test ends to complete this portion. Remember that bonus points do not count towards your core-point total! To receive bonus points, tear this sheet off from the exam, make sure the proctor records the points on the front page, and shuffle it in a separate pile to maintain anonymity.

1) How many hours per week are you spending on doing homework for this course?

2) What are 1 to 3 things we can do to improve lecture? You may also say what you like, as well. If you do *not* attend lecture, please explain why.

3) What are 1 to 3 things we can do to improve section? You may also say what you like, as well. If you do *not* attend section, please explain why.

4) What are 1 to 3 things we can do to improve consulting? You may also say what you like, as well. If you do *not* attend consulting, please explain why.

5) Is there anything we can do to improve CS211, overall?