CS180 Spring 2011 Final Exam, 3 May, 2011
Prof. Chris Clifton

Turn Off Your Cell Phone. Use of any electronic device during the test is prohibited.

Time will be tight. If you spend more than the recommended time on any question, **go on to the next one**. If you can't answer it in the recommended time, you are either going in to too much detail or the question is material you don't know well. You can skip one or two parts and still demonstrate what I believe to be an A-level understanding of the material.

It is okay to abbreviate in your answers, as long as the abbreviations are unambiguous and reasonably obvious. You can also use the back of sheets for your answers, although the provide space should be adequate.

Total: 60 points. *Scoring suggestions are tentative and subject to change.*

For multiple choice questions, please circle the letter next to your choice.

1 Looping Constructs (4 minutes, 2 points)

Given the following for loop:

```
int sum = 0;
for ( int i=1; i<4; i++ ) {
  if ( i != 3 )
    sum += i;
}
```

A programmer has converted the above for loop into the following while loop:

```
int i = 1;
int sum = 0;

while ( i < 4 ) {
  if ( i != 3 ) {
    sum += i;
    i++;
  }
}
```

Does the converted while loop produce the same result as the for loop? Explain your answer.

No. The result of sum after for loop is 3. However, the while loop enters an infinite loop when i increases to 3.

Scoring: 1 for No, 1 for reasonable explanation matching answer.

2 Inheritance (6 minutes, 7 points)

```
public class Parent {
  public void fun() { /* Make this Parent object happy */  }
}

public class Child extends Parent {
  public void fun() { /* Make this Child object happy */  }
  public void disneyland() { /* Make both parent and child
      parts of this object happy */  }
}
```

2.1 Overriding (3 minutes, 3 points)

Given the following two objects:

```
Parent p = new Parent();
Child c = new Child();
```

For each of the following code fragments, which of the `fun()` methods are executed, the one in the class Child, the one in the class Parent, or both (and in which order)?

- *Part 1:* `p.fun();`

  A **fun() in Parent is executed.**

  B fun() in Child is executed.

  C fun() in Child is called, then fun() in Parent.

  D fun() in Parent is called, then fun() in Child.

- *Part 2:* `c.fun();`

  A fun() in Parent is executed.

  B fun() **in Child is executed.**

  C fun() in Child is called, then fun() in Parent.

  D fun() in Parent is called, then fun() in Child.

- *Part 3:* `p = c;  p.fun();`

  A fun() in Parent is executed.

  B **fun() in Child is executed.**

  C fun() in Child is called, then fun() in Parent.

  D fun() in Parent is called, then fun() in Child.

2.2 Calling Parent class (1 minute, 2 points)

In writing the method `disneyland` in class `Child`, you want to call the `fun()` methods in class `Child` and class `Parent`. Show both calls.

fun();
super.fun();

Scoring: 1 point for each as long as reasonably close.

2.3 Abstract Classes (2 minutes, 2 points)

Assume `Parent` is an abstract class (and at least one of its methods is abstract). What changes would need to be made to `Child` and why?

The Child class must implement any methods declared as abstract in the Parent class.

Scoring: 1 point for "none", 2 for "implement any abstract methods" or "none because all abstract methods implemented".

3 Exceptions (5 minutes, 3 points)

Given the following two classes:

```
public class CS180Exception extends Exception {
}

public class CS180 {
  public void methodA() throws CS180Exception {
  }

  public void methodB() {
    methodA();
  }
}
```

3.1 Errors (2 minutes, 1 points)

The above code will lead to a compilation error. Why?

methodA throws an exception that methodB doesn't throw or catch.

Scoring: 1 for missed exception.

3.2 Correct Use (3 minutes, 2 points)

Rewrite methodB() in two different ways so that the code compiles.

A
```
public void methodB() throws CS180Exception {
```

B
```
try {
    methodA();
  } catch (CS180Exception e) { }
```

Scoring: 1 point for getting a try/catch, 1 for throwing.

4 Inheritance (5 minutes, 1 points)

Consider the following class definitions:

```
public class A {
  public A() { System.out.println("A"); }
  public A( String str ) { System.out.println(str); }
}

public class B extends A {
  public B() { System.out.println("B"); }
  public B( String str ) { System.out.println(str); }
}
```

Which code segment would produce output *different* from the rest:

A
```
new B(); new A();
```

B
```
new B("B"); new A();
```

C
```
new A(); new B("B");
```

D
```
new B("B"); new A("A");
```

Answer: C (A,B,D produces ABA while C produces AAB)

5 Scope Rules (10 minutes, 6 points)

For each of the following programs, show the output when the program is run. Partial credit available for explaining your answer even if the final output you give isn't right (e.g., showing values of parameters or variables, or output of method calls), if you show correct understanding.

5.1 Part 1 (4 minutes, 2 points)

```
public class Point {
  public int x, y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}

public class Q1 {

  public static void increasePoint(Point p, String str) {
    str = "new point is :";
    p.x = p.x+1;
    p.y = p.y+1;
  }

  public static void main(String[] args) {
    String str = "old point is";
    Point p = new Point(1,1);
    System.out.println(str+"("+p.x+","+p.y+")");
    increasePoint(p,str);
    System.out.println(str+"("+p.x+","+p.y+")");
  }
}
```

old point is(1,1)
old point is(2,2)

Scoring: 1 for "old point is" for both, 1 for 1,1 followed by 2,2.

5.2 Part 2 (4 minutes, 2 points)

```
public class Q2 {
  public static void increment(int x) {
    x=x+1;
  }

  public static void main(String[] args) {
    int x = 0;
    while(x<5) {
      increment(x);
      System.out.println(x);
    }
  }
}
```

This enteres an infinite loop printing 0.

Scoring: 1 for 0, 1 for infinite loop.

5.3 Part 3 (2 minutes, 2 points)

```
public class Q3 {
  static int x = 0;
  static int y = 0;

  public static void increment(int x)
  {
    x = x+1;
    y = y+1;
  }

  public static void main(String[] args)
  {
    increment(x);
    System.out.println(x);
    System.out.println(y);
  }
}
```

0
1

Scoring: 1 for 0, 1 for 1.

6 Linked Data Structures (6 minutes, 2 points)

```
import java.util.LinkedList;

public class Element {
  public int value;

  public void addToList( LinkedList<Element> linkList ) {
    Element e = new Element();
    for ( int i=0; i<10; i++ ) {
      e.setValue(i);
      linkList.add(e);
    }
  }

  public void setValue( int value ) {
    this.value = value;
  }
}
```

If addToList is invoked, I claim that all 10 'Elements' in `linkList` will have the same 'value' 9. True or False? Explain. (Answer valid only with correct explanation) *Hint: You don't have to worry about how java.util.LinkedList works - the answer is the same with any reasonable implementation of a LinkeList, such as those shown in class. And yes, this does compile.*

Only a single Element object is created, and it is inserted into the list multiple times. Since the setValue is operating on that one object, each time it sets the value, it essentially changes the value of everything in the list.

Scoring: 2 for catching idea that only one element, 1 for showing reasonable understanding of linked list in another way.

7 Use of Inherited classes (5 minutes, 1 points)

What will be the result of attempting to compile and run the following program?

```
public class Polymorphism {
  public static void main(String[] args) {
    A ref1 = new C();
    B ref2 = (B) ref1;
    System.out.println(ref2.f());
  }
}

class A { int f() { return 0; } }
class B extends A { int f() { return 1; } }
class C extends B { int f() { return 2; } }
```

Select the one correct answer:

A The program will fail to compile.

B The program will compile without error, but will throw a ClassCastException when run.

C The program will compile without error and print 1 when run.

D **The program will compile without error and print 2 when run.**

8 Creating a Data Abstraction (12 minutes, 10 points)

In this question, you will create parts of a GradeBook class and methods. A GradeBook contains assignments, for each assignment there is a score for each student. You will have to decide what data types / data structures to use. *Note: There is a lot of partial credit available for even getting small pieces right, so try to write even a partial answer every part even if you know your entire answer isn't correct.*

8.1 Class Definition (4 minutes, 3 points)

Write the basic class definition, with fields to hold the assignments. For full credit, your definition should not limit how many assignments a GradeBook can have. If you want to define multiple classes, that is okay - but you should at least have a GradeBook class.

```
public class GradeBook {
  float grades[][];

}
```

Scoring: 1 for basic definition, 1 for field(s) for assignments, 1 for holding students.

8.2 Constructor (4 minutes, 4 points)

Create a constructor that takes the number of assignments and number of students as arguments, and creates the objects needed for your GradeBook to store the assignment grades. (Note that it would be nice if the GradeBook wasn't limited to the provided number of assignments, but it is okay if it is.)

```
  public GradeBook(int asn, int stud) {
    grades = new float[asn][stud];
  }
```

Scoring: 1 for proper constructor, 1 for arguments, 2 for instantiating fields as appropriate (1 if partially done.)

8.3 Calculation (4 minutes, 3 points)

Create a method `mean` in your GradeBook class that takes an assignment number as an argument, and returns the average (mean) score on that assignment.

```
public float mean(int asn) {
  float sum = 0.0f;
  for ( int i=0; i<grades[asn].length; i++ )
    sum += grades[asn][i];
  return sum / grades[asn].length;
}
```

Scoring: 1 for method with proper argument, 1 for proper return type, 1 for use of objects.

9 Recursion (10 minutes, 5 points)

Given the following recursive methods:

```
public int f( int x )
{
  if ( x == 1 )
    return 1;
  else
    return x * f(x-1);
}
```

```
public int g( int x )
{
  if ( x == 0 )
    return 0;
  else
    return f(x) + g(x-1);
}
```

9.1 Understanding the code (4 minutes, 3 points)

What is returned by `g(4);` ? *If you aren't certain, showing your work may get you partial credit even if you get the wrong answer.*

This is a sum of factorials, 4! + 3! + 2! + 1! = 24 + 6 + 2 + 1 = 33.

Scoring: 3 for correct (or clearly understands but arithmetic error), 2 for showing correct understanding of one method, 1 for showing some understanding of recursion.

9.2 Identifying code problems (3 minutes, 1 point)

The above code will fail with some inputs. Give an example input `x` where `g(x)` will fail.

g applied to a negative number does not terminate.

Scoring: 1 for any negative example.

9.3 Fixing problems (3 minutes, 1 point)

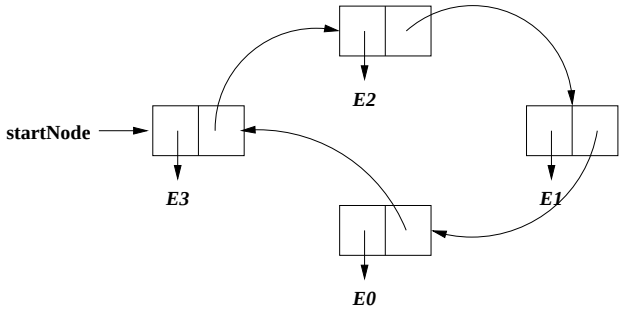
Revise the code of method f or g to prevent the failure you identified in 9.2.

The first line of g should be changed to: `if ( x <= 0 )`

Scoring: 1 for any legal check for < 0 *in g, or if they came up with a different issue in previous question, a solution to their issue.*

10 Linked Data Structures (15 minutes, 6 points)

A circular linked list is a list in which the link field of the last node is made to point to the start/first node of the list. The diagram at the right shows a CircularLinkedList with nodes added in the order *E0, E1, E2, E3*. To help you to implement this class, we provide you the following class definitions:




```
class Node<E> {
  public E info;
  public Node<E> next;
  public Node (E info, Node<E> next)
  {
    this.info = info;
    this.next = next;
  }
}
```

```
public class CircularLinkedList<E> {
  private Node<E> startNode;
  public CircularLinkedList () {
    startNode = null;
  }
  public void insert(E item) {
    /* item E is inserted at the position of the startNode and becomes
       the new start of the circular linked list. This new start
       node's next attribute should point to the original startNode.
       Be sure that after inserting, the list still a valid circular list. */

// Solution below.

Node<E> temp = new Node<E>(item, startNode);

if ( startNode == null )
  temp.next = temp;
else {
  temp.next = startNode;
  Node<E> current = startNode;
  while ( current.next != startNode ) current = current.next;
    current.next = temp;
}
startNode = temp;

// Solution ends here.

  }
}
```

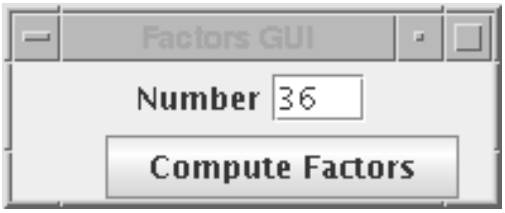
Your task is to implement the insert method of the CircularLinkedList class (fill in the `insert` method.) Hint: You should consider two possible cases: when the list is empty and when it has at least one element.

Scoring: For each case (empty list, non-empty list), you can get 1 point for creating a new node with the correct value, 1 for having startNode point to the new node, and 1 for the tail pointing to this new node. If this doesn't add up to six, you may receive up to two points for a generally correct structure that shows an understanding of linked lists.

11 Event Handling (22 minutes, 7 points)

For this question, you will finish writing a GUI which has a simple text field and button interface. By inputting a (positive) number into the field and pressing the button, a new thread should be spawned which computes the factors of the entered number (e.g., if the number is n=36, the factors are 1, 2, 3, 4, 6, 9, 12, 18, 36). This thread can simply iterate from 1 to the given number n, printing out the factor if the iterated number is a factor of n. Between each iteration, the thread should sleep for 1 second (that is, 1000 milliseconds). Since the computation is concurrent in a separate thread, if the GUI's button is pressed again with another number m, then m's factors should be computed in another thread. Here's an example print out in the console with the factors of 36, 25, and 16 being computed (note that the order of the print statements is entirely based on the timing of the threads being started):

1 is a factor of 36
2 is a factor of 36
1 is a factor of 25
3 is a factor of 36
4 is a factor of 36
1 is a factor of 16
6 is a factor of 36
2 is a factor of 16
5 is a factor of 25
4 is a factor of 16
9 is a factor of 36
12 is a factor of 36
8 is a factor of 16
18 is a factor of 36
16 is a factor of 16
25 is a factor of 25
36 is a factor of 36



A partial implementation of the GUI is given below:

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

public class ConcurrentFactorsGui
{
    // the frame of the gui
    private JFrame frame;
    // the text field used for input
    private JTextField numberField;

    // an ActionListener for the button press
    public class FactorButtonListener implements ActionListener
    {
        /*** PART 1 BEGIN ***/

        public void actionPerformed(ActionEvent e)
        {
            int num = Integer.parseInt(numberField.getText());
            (new ConcurrentFactorsThread(num)).start();
        }

        /*** PART 1 END ***/
    }
```

```
    // constructor sets up the gui
    public ConcurrentFactorsGui()
    {
        // set up the frame, primary panel, and layout
        frame = new JFrame("Factors GUI");
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.PAGE_AXIS));

        // set up a panel to get the number
        JPanel numberPanel = new JPanel();
        numberPanel.setLayout(new FlowLayout());
        numberField = new JTextField();
        numberField.setText("36");
        numberField.setColumns(3);
        JLabel numberLabel = new JLabel("Number");
        numberLabel.setLabelFor(numberField);
        numberPanel.add(numberLabel);
        numberPanel.add(numberField);
        mainPanel.add(numberPanel);

        // set up the button to compute the factors of the input number
        JButton factorsButton = new JButton("Compute Factors");
        /*** PART 2 BEGIN ***/

factorsButton.addActionListener(new FactorButtonListener());

        /*** PART 2 END ***/
        mainPanel.add(factorsButton);

        frame.add(mainPanel);
        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args)
    {
        ConcurrentFactorsGui gui = new ConcurrentFactorsGui();
    }
}
```

You are responsible for completing the program in 3 locations, described below.

11.1 FactorButtonListener (6 minutes, 3 points)

At **Part 1**, finish the implementation of FactorButtonListener, which will be the ActionListener when the factorsButton JButton is pressed. When the button is pressed, the number can be retrieved from the numberField JTextField with the getText method of JTextField (you'll have to convert the text to the integer yourself). For simplicity, you may assume that the text will always contain an integer.

Scoring: 1 point for defining ActionPerformed, 1 for getting text and performing any attempt at converting to integer, 1 for starting thread.

11.2 Register Listener (4 minutes, 2 points)

At **Part 2**, add code which registers an instance of your FactorButtonListener as the ActionListener to factorsButton.

Scoring: 1 for adding listener, 1 for creating instance of FactorButtonListener.

11.3 Concurrency (10 minutes, 2 points)

Write a thread class ConcurrentFactorsThread which computes the factors of its given number and prints them out on the console.

Solution:

```
public class ConcurrentFactorsThread extends Thread
{
    private int num;

    public ConcurrentFactorsThread(int num)
    {
        this.num = num;
    }

    @Override
    public void run()
    {
        for (int i=1; i<=num; i++)
        {
            if (num % i == 0)
                System.out.printf("%d is a factor of %d\n", i, num);
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

Scoring: 1 each for reasonable definition of class/constructor, 1 for reasonable definition of run method.

12 File I/O (20 minutes, 9 points)

Write a program that reads in lines from the file "input.txt", and writes the lines to the file "output.txt" *in the opposite order* (last line in input.txt is the first in output.txt, 2nd to last in input.txt is 2nd in output.txt, etc.)

For example, if the file "input.txt" contains:

```
CS180 is awesome
CS180 is very awesome
CS180 is very very awesome
```

then your program should create a file "output.txt" containing:

```
CS180 is very very awesome
CS180 is very awesome
CS180 is awesome
```

For full credit (worth one point), you should have a class with a method
`void reverse(Scanner in, PrintWriter out)` . *You can ignore exception handling – it is okay to assume that none of the methods you call throw exceptions.*

Hints: Scanner has a method `String nextLine()` and `boolean hasNextLine()`, PrintWriter has a method
`void println(String x)`. Both have constructors that take a `File` or `InputStream` object as an argument; a `File` can be constructed with a `String` argument (the name of the file.) Given this information, there is a very simple recursive solution.

Solution follows:

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

public class Reverse {

  public static void reverse(Scanner in, PrintWriter out) {
    if ( in.hasNextLine() ) {
      String line = in.nextLine();
      reverse(in, out);
      out.println(line);
    }
  }

  public static void main(String args[]) {
    try {
      PrintWriter out = new PrintWriter(new File("output.txt"));
      reverse(new Scanner(new File("input.txt")), out);
      out.close();
    } catch (FileNotFoundException e) { }
  }
}
```

Scoring: 1 for reverse method, 1 for constructing Scanner/Printwriter (or InputStream/OutputStream) objects, 1 for constructing properly from given file names, 1 for appropriate reading and/or writing, 1-3 for quality of "reverse" logic (1 for idea, 2 for workable idea, 3 for correct), 1 for having class, 1 for main()