MIDTERM EXAMINATION
CSC 112 ♦ FALL 2008

You will have 110 minutes to complete this exam. All work should be written in the exam booklet. Start with the questions that you know how to do, and try not to spend too long on any one question. Partial credit will be granted where appropriate. Good luck!

1. **Program Context** (16 points)

Which line(s) of the following program incorrectly call(s) a method from an improper context?

```
1   public class Context {
2       public static void staticMethod() {
3           nonstaticMethod();
4           Context c = new Context();
5           c.nonstaticMethod();
6       }
7
8       public void nonstaticMethod() {
9           staticMethod();
10          Context c = new Context();
11          c.staticMethod();
12      }
13
14
15      public static void main(String[] args) {
16          staticMethod();
17          nonstaticMethod();
18      }
19  }
```

2. **Programming Style** (12 points)

Give at least three advantages or disadvantages of using classes from the Java Collections framework instead of your own custom-designed classes that perform the same function. Is there ever a situation where your own class would be preferred?

3. **Programming Efficiency** (16 points)

The following code defines several listener classes that print out various messages when the associated button is pressed. Rewrite these three classes as a single class that can fulfill all three roles (i.e., printing the appropriate message). Then rewrite the middle part of the `createButtons` method so that it works with your new class. (The full program is not shown; assume that the surrounding code to create the GUI and display it is working properly.)

Nested class definition for the listeners:

```
private class ButtonOneListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Clicked button one.");
    }
}

private class ButtonTwoListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Clicked button two.");
    }
}

private class ButtonThreeListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Clicked button three.");
    }
}
```

Code in `createButtons()` to create and add the buttons and their listeners:

```
JButton button1 = new JButton("Button One");
JButton button2 = new JButton("Button Two");
JButton button3 = new JButton("Button Three");

// middle part:
ButtonOneListener listener1 = new ButtonOneListener();
ButtonTwoListener listener2 = new ButtonTwoListener();
ButtonThreeListener listener3 = new ButtonThreeListener();

frame.getContentPane().add(button1);
frame.getContentPane().add(button2);
frame.getContentPane().add(button3);
```

4. **Generic Programming** (12 points).

Write a single generic method that would replace the two shown below.

```
static void shift(Integer[] x) {
    Integer tmp = x[0];
    for (int i = 1; i < x.length; i++) {
        tmp[i-1] = tmp[i];
    }
    x[x.length-1] = tmp;
}
static void shift(String[] x) {
    String tmp = x[0];
    for (int i = 1; i < x.length; i++) {
        tmp[i-1] = tmp[i];
    }
    x[x.length-1] = tmp;
}
```

5. **Program Simulation** (12 points)

Simulate execution of the following program, and show its output.

```
public class Sim {
    private static int x = 3;

    public int method(int x) {
        this.x = -this.x;
        x = -x;
        return x;
    }

    public static void main(String[] args) {
        int x = 4;
        Sim s = new Sim();

        System.out.println(x);
        System.out.println(Sim.x);
        System.out.println(s.x);

        x = s.method(x);

        System.out.println(x);
        System.out.println(Sim.x);
        System.out.println(s.x);

        s.x = s.method(Sim.x);

        System.out.println(x);
        System.out.println(Sim.x);
        System.out.println(s.x);
    }
}
```

6. **Stacks and Queues** (16 points)

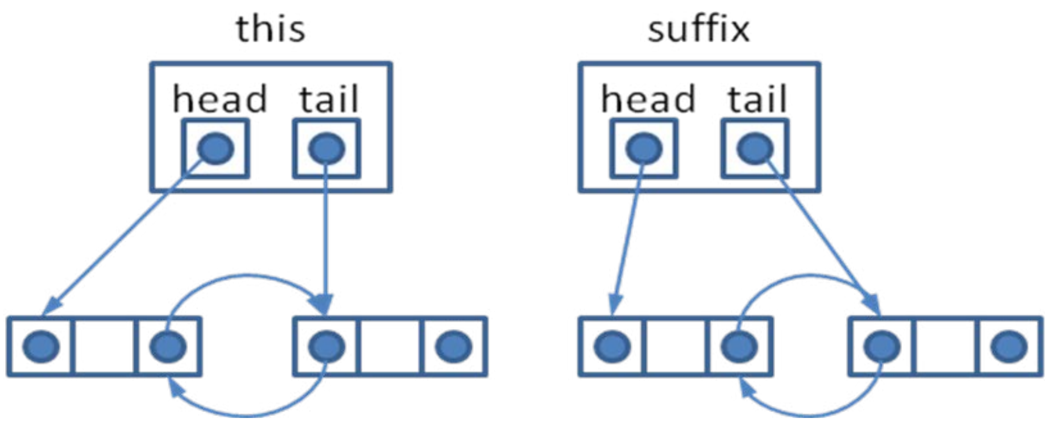
A **deque** is a list data structure that allows items to be added and removed from either end. Suppose that class **`Deque<E>`** is already defined and has methods **`isEmpty`**, **`addLeft`**, **`addRight`**, **`removeLeft`**, and **`removeRight`**. Write implementations of **`push`** and **`pop`** for **`Stack<E>`** implemented using the deque methods. Similarly, write implementations of **`offer`** and **`poll`** for **`Queue<E>`**. Your methods should throw an exception if the user attempts a forbidden action.

```
public class Stack<E> {
   Deque<E> d = new Deque<E>();

   // FILL IN METHODS
}

public class Queue<E> {
   Deque<E> d = new Deque<E>();

   // FILL IN METHODS
}
```

7. **Linked Lists** (16 points)

You are trying to implement an **`append`** method for the **`DL_IntList`** class we developed. The code below will not accomplish this. Assuming that the picture below shows the state of the data structures involved before the method below has executed, draw a picture of the result after it has executed. Then, *in words*, describe the error(s) that have been made, and what should be done instead. Do not attempt to fix the code.

```
public void append(DL_IntList suffix) {
   tail.next = suffix.head;
   tail = suffix.tail;
   suffix.head = tail;
   suffix = null;
}
```


this
head
tail
suffix
head
tail