

CSE 120

Midterm II — March 4, 2009

Name: _____

Student ID: _____
(from your PennCard)

Email _____ @seas.upenn.edu

Lab:(circle one)

201	202	203	204
W 1:00 PM	W 2:30 PM	R 12:00 PM	R 1:30 PM

- This exam text and attached code, the *Java Backpack Reference Guide*, and your brain are the only information sources you can use during this exam.
- You have 50 minutes to answer all of the questions. The entire exam is worth 50 points. The point value of each question is given to help you plan your time.
- Write your answers on the exam pages. The back side of each page may be used as a scratch pad.
- The answers to the questions require only basic Java language facilities and the code given as appendix. No other classes or packages are needed.
- Questions during the exam should be about the wording of the exam only. If you have a question, raise your hand and we'll come to you. (This is less disruptive for others than if you come to us.)
- **Partial credit will be given where appropriate.**
- DON'T PANIC!. If you find a question that you cannot solve right away, consider moving on and returning to it after you finish the other questions.

	Score	Max
1		16
2		9
3		13
4		12
Total		50

1. In this problem, you will add two new methods to the `CharSeqArrImp` class which implements the `CharSeq` interface presented. These were discussed in lecture; code is included in the code handout, modified to include additions to the `CharSeq` interface for the methods you will add. Neither of the methods you will provide depends on the other. Here are some interactions with the methods you will provide:

```
> char[] a = {'a','b','a','b','c'}
> CharSeq s = CharSeqArrImp.makeSeq(a);
> s
a b a b c
> CharSeq w = CharSeqArrImp.makeSeq(new char[] {'a','b'})
> w
a b
> s.findNth(w,0)
0
> s.findNth(w,1)
2
> s.findNth(w,2)
-1
> s
a b a b c
> s.rotate(1)
> s
b a b c a
> s.rotate(0)
> s
b a b c a
> s.rotate(3)
> s
c a b a b
>
```

(Questions on next page)

- (a) (8 points) Write below a method `findNth`

```
public int findNth(IntSeq word, int n).
```

This method is intended to return the index of the n th occurrence of a given word, counting from 0, or -1 if the word does not occur at least $n + 1$ times in this sequence. (Hint: reuse the code for `find`, modified appropriately.)

- (b) (8 points) Write below a method `rotate`

```
public void rotate(int n)
```

This method rotates the sequence it is invoked on n positions “counterclockwise”. That is, if n is 1, the 0th element becomes the last element, the 1st element becomes the 0th element, and in general the i th element becomes the $i - 1$ element. The method should have no effect on a sequence with fewer than two elements.

2. There are three *compilation* errors in the following code fragment. Circle each error, mark it with a letter (a)-(c), and in the space below, *briefly* describe what caused the error. There is only one error per line. (Don't waste time trying to figure out what this program does!)

```
abstract class FunCountImp implements FunnyCount {
    private int count = 512;
    public abstract int getCount();
    public void countExceed() { throw new CountExceededException(); }
    public void maybeIncrement() {
        FunnyCount x = this;
        if (x.getCount() >= 10) {
            return;
        }
        incr();
    }

    public static void incr() { this.count++;}
}

public class CountExceededException extends Exception{
    public CountExceededException(String s){super(s);}
    public CountExceededException(){super();}
}

interface FunnyCount {
    public void countExceeded();
}
```

(a) (3 points)

(b) (3 points)

(c) (3 points)

3. This problem concerns the two Java source files, **TicketFactory.java** and **Ticket.java**, found in the code handout. Please take a minute to look over these source files.

Your job is to implement the class **OnePerFactory** that adapts the **TicketFactory** class so that a single person can buy at most *one* ticket from a *particular* ticket factory.

A sample interaction with this new class might be as follows:

```
> import ticket.*;
> OnePerFactory tf = new OnePerFactory("Springsteen", 200);
> Ticket t0 = tf.issue("Victoria");
> t0
Ticket # 0 issued to Victoria
> Ticket t1 = tf.issue("Annie");
> t1
Ticket # 1 issued to Annie
> Ticket t2 = tf.issue("Victoria");
> t2
null
```

The beginning of the definition of class **OnePerFactory** is below. This definition includes the declaration of two instance variables: **tickets** should store an array of tickets issued by the factory, and **ticketFactory** is an instance of the **TicketFactory** class. Your task will be to implement the constructor and two methods for this class. You may not add any new instance variables, nor change the declarations below.

```
package ticket;
```

```
public class OnePerFactory {
    // Array of tickets that have been issued by this factory.
    private Ticket[] tickets;

    // A ticket factory
    private TicketFactory ticketFactory;
```

- (a) (3 points) Complete the constructor for this class.

```
    public OnePerFactory(String name, int maxTickets) {
```

(b) (4 points) Implement a helper method of the class `OnePerFactory` that determines whether a person has already bought a ticket.

(c) (6 points) Implement the `issue` method to make sure that each purchaser can only buy one ticket. If a purchaser has previously bought a ticket, this method should return `null`.

4. (12 points) The next problem concerns the implementation of the method `divide`.

The static method, `divide` in the class `IntegerDivision`, prompts the user for two integers, divides the first integer (the dividend) by the second (the divisor) using integer division and returns the result. If the divisor is zero, and hence the computer attempts to divide by zero, the method should prompt the user for another two numbers. For example, a DrJava interaction might look like:

```
> IntegerDivision.divide(System.in)
Enter dividend:
 5
Enter divisor:
 0
Divide by zero - try again
Enter dividend:
 5
Enter dividend:
 2
2
>
```

In the implementation of `divide`, all input should be from the parameter `input` of type `InputStream`, and all output should be to `System.out`, using the method `println`. Input should be processed using an instance of the class `java.util.Scanner` created in the following manner:

```
Scanner scanner = new Scanner(input);
```

To read inputs, `divide` should use the method `nextInt` from the `java.util.Scanner` class. More information about this method is available in the Appendix.

The `divide` method must be very careful about exceptions during input. If the user inputs a zero as the divisor, `divide` will attempt to divide by zero, and throw an `ArithmeticException`. In this case, `divide` should catch the exception and ask the user for other inputs, as shown above. If the user inputs are not integers, `nextInt` will throw a `InputMismatchException`. If other input errors occur, `nextInt` will throw two other runtime exceptions (see the documentation in the code appendix). If any of these three exceptions are thrown, `divide` should catch them and throw a `ScanningException` (which should extend `Exception`) to indicate that there was some trouble. You may ignore exceptions that could arise from `System.out.println`.

In the class `IntegerDivision` on the next page, define the method `divide`, which takes a single argument, the `InputStream` `input`, and returns an `int`. Do not forget to include the appropriate `throws` annotation.

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

public class IntegerDivision {
```

CharSeq.java

```
1  /** Interface for an immutable data structure that stores sequences of characters.
2   */
3
4  interface CharSeq {
5     /**
6      * Get the length of the sequence.
7      *
8      * @return the length
9      */
10    public int size();
11
12    /**
13     * Get an element of the sequence.
14     *
15     * @param i the index of the element
16     * @return the element
17     */
18    public char get(int i);
19
20    /**
21     * Does a word occur at a given position in this sequence?
22     *
23     * @param word the word.
24     * @param pos the position.
25     * @return whether the word occurs in this sequence at <code>pos</code>.
26     */
27    public boolean occursAt(CharSeq word, int pos);
28
29    /**
30     * Find the position of the first occurrence of a word in this sequence.
31     *
32     * @param word the word.
33     * @return the position of <code>word</code>'s first occurrence, or
34     * -1 if <code>word</code> does not occur.
35     */
36    public int find(CharSeq word);
37
38    /**
39     * See problems
40     */
41
42    public int findNth(CharSeq word, int n);
43
44    /**
45     * See problems
46     */
47    public void rotate(int n);
48
49    /**
50     * Create a new sequence by concatenating another on the end.
51     *
52     * @param other the other sequence
53     * @return this + other
54     */
55    public CharSeq concat(CharSeq other);
56 }
```

CharSeqArrImp.java

```
1  /**
2   * Sequences of integers represented as arrays.
3   */
4  public class CharSeqArrImp implements CharSeq {
5      private final char[] data;
6
7      public CharSeqArrImp() {
8          this.data = new char[0];
9      }
10
11     /**
12     * Create a sequence object from a given array,
13     * without copying the array. Made private so that
14     * only this class can call this constructor.
15     *
16     * @param toCopy the array.
17     */
18     private CharSeqArrImp(char[] newData) {
19         this.data = newData;
20     }
21
22     /**
23     * Create a sequence object from a given array.
24     *
25     * @param toCopy the array.
26     */
27     public static CharSeq makeSeq(char[] toCopy) {
28         char[] data = new char[toCopy.length];
29         for (int i = 0; i < toCopy.length; i++)
30             data[i] = toCopy[i];
31         return new CharSeqArrImp(data);
32     }
33
34     /**
35     * Get the length of the sequence.
36     *
37     * @return the length
38     */
39     public int size() { return data.length; }
40
41     /**
42     * Get an element of the sequence.
43     *
44     * @param i the index of the element
45     * @return the element
46     */
47     public char get(int i) { return data[i]; }
48
49     /**
50     * Does a word occur at a given position in this sequence?
51     *
52     * @param word the word.
53     * @param pos the position.
54     * @return whether the word occurs in this sequence at <code>pos</code>.
55     */
56     public boolean occursAt(CharSeq word, int pos) {
57         for (int i = 0; i < word.size(); i++) {
58             if (i + pos >= size() ||
```

```

57         data[i+pos] != word.get(i))
58         return false;
59     }
60     return true;
61 }
62 /**
63  * Find the position of the first occurrence of a word in this sequence.
64  *
65  * @param word the word.
66  * @return the position of <code>word</code>'s first occurrence, or
67  * -1 if <code>word</code> does not occur.
68  */
69 public int find(CharSeq word) {
70     int where = -1;
71     for (int pos = 0; pos <= size(); pos++) {
72         if (occursAt(word, pos)) {
73             where = pos;
74             break;
75         }
76     }
77     return where;
78 }
79
80 /**
81  * Create a new sequence by concatenating another on the end.
82  *
83  * @param other the other sequence
84  * @return this + other
85  */
86 public CharSeq concat(CharSeq other) {
87     int newSize = this.size() + other.size();
88     char[] newData = new char[newSize];
89     int i;
90     for (i=0; i< size(); i++) {
91         newData[i] = this.get(i);
92     }
93     for (; i < newSize; i++) {
94         newData[i] = other.get(i-size());
95     }
96     return new CharSeqArrImp(newData);
97 }
98
99 public String toString() {
100     StringBuffer b = new StringBuffer();
101     for (int i = 0; i < data.length; i++) {
102         if (i > 0)
103             b.append(' ');
104         b.append(data[i]);
105     }
106     return b.toString();
107 }
108 }

```

TicketFactory.java

```
1 package ticket;
2
3 public class TicketFactory {
4     private String name;        // name of event
5     private int issued;        // number of tickets that have been issued
6     private int maxTickets;    // maximum number of tickets available
7
8     // constructor, initializes instance variables
9     public TicketFactory(String name, int maxTickets) {
10        this.name = name;
11        this.maxTickets = maxTickets;
12        this.issued = 0;
13    }
14
15    // accessor methods for instance variables
16    public int getIssued()      { return issued; }
17    public int getMaxTickets() { return maxTickets; }
18    public String getName()    { return name; }
19
20    // Factory method: issue a ticket to a particular
21    // purchaser as long as the maximum number of tickets
22    // has not been reached. Return null otherwise.
23    public Ticket issue(String purchaser) {
24        if (issued < maxTickets) {
25            Ticket t = new Ticket(purchaser, issued);
26            issued++;
27            return t;
28        }
29        else {
30            return null;
31        }
32    }
33 }
```

Ticket.java

```
1 package ticket;
2
3 public class Ticket {
4     private String purchaser; // the purchaser of this ticket
5     private int number;       // the ticket number
6
7     // Note: Constructor has package scope
8     Ticket(String purchaser, int number) {
9         this.purchaser = purchaser;
10        this.number = number;
11    }
12
13    // accessor methods for instance variables.
14    public String getPurchaser() { return purchaser; }
15    public int    getNumber()    { return number; }
16
17    public String toString() {
18        return "Ticket # " + number + " issued to " + purchaser;
19    }
20 }
```