

## CIS 120

### Midterm I — February 20, 2008

Name: \_\_\_\_\_

Student ID: \_\_\_\_\_  
(from your PennCard)

Email \_\_\_\_\_ @seas.upenn.edu

Lab:(circle one)

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

- 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     |       | 20  |
| 2     |       | 10  |
| 3     |       | 6   |
| 4     |       | 4   |
| 5     |       | 10  |
| Total |       | 50  |

1. The following questions pertain to some new methods for the `IntSeq` class discussed in lectures and included in the code handout.

(a) (5 points)

```
public IntSeq m1() {
    IntSeq newSeq = new IntSeq(size());
    for (int i=0; i<data.length/2; i++) {
        if (data[i] > data[data.length-i-1]) {
            newSeq.data[i] = data[data.length-i-1];
            newSeq.data[data.length-i-1] = data[i];
        }
    }
    return newSeq;
}
```

What is the result of the following interaction:

```
> IntSeq u = new IntSeq(new int[]{ 1, -3, 4, 3, -7 })
> u.m1()
```

*Answer: 1 0 0 0 -7*

(b) (5 points) List all lines of the following method that could trigger an `IndexOutOfBoundsException`.

```
1 public int m2(IntSeq other) {
2     int sum = data[size()-1];
3     for (int i=0; i < other.size(); i++) {
4         int j = other.get(i);
5         int k = other.get(size()-i);
6         if (k >= 0)
7             sum += data[k];
8         if (0 <= j && j < size())
9             sum += data[j];
10    }
11    return sum;
12 }
```

*Answer: Lines 2, 5 and 7. Not lines 4 or 9. Line 2 throws for 0 length array. Line 5 because it uses size instead of other.size. Line 7 because k could be larger than data.length. Line 4 is protected by for loop. Line 9 is protected by if.*

- (c) (10 points) Write the method `int findSecondLargest()` which returns the value of the second largest number in the `IntSeq`. For example, if the sequence is 1 2 3 4, the method should return 3. If all of the values in the sequence are the same, the method should return that value. If the sequence is empty, the method should return 0.

```
/**
 * Find and return the second largest int in this IntSeq.
 * If the sequence is empty, return 0.
 * If the sequence contains no second largest, return the largest.
 * (This covers the all-the-same case, and deals with size-1 cases.)
 */
public int findSecondLargest() {
    if (size() <= 0) {
        return 0;
    }
    int largest = get(0);
    int second = get(0);
    int s = size();
    for(int i = 1; i < s; i++) {
        int x = get(i);
        if (x > largest) {
            second = largest;
            largest = x;
        }
        if (x > second && x < largest) {
            second = x;
        }
    }
    return second;
}
```

2. (10 points) Consider the following related classes:

```
public class A {
    protected int x;
    protected static int z;

    public A() { this.x = 6; this.z++; }
    public int m(int y) { return x + y; }
}

public class B extends A {
    public B(int y) {
        super();
        this.x = y;
        this.z++;
    }
}

public class C extends B {
    public C(int x) { super(x); }
    public int m(int y) { return (2 * x) + y; }
}
```

What would the DrJava interpreter return after each of the following interactions? Assume that the interactions occur in the order below and the interactions pane is *not* reset between interactions.

(a) > A a = new A()  
> a.m(4)

*Answer:10*

(b) > a = new B(6);  
> a.m(4)

*Answer:10*

(c) > a = new C(6);  
> a.m(4)

*Answer:16*

(d) > A.z

*Answer:5*

(e) > a instanceof B

*Answer:true*

3. (6 points) This problem refers to the shape classes that were discussed in lectures and are provided in the code handout. For each box, circle either the word **ok** if the interaction succeeds or the word **error** if the interaction causes an error. Assume that the interactions occur in the order below and the interactions pane is *not* reset between interactions.

(a) > Square s = new Square(new Point(2,2),3);  
> Rectangle r = s  
> r.getArea()

**ok**    **error**

*Answer:9.0*

(b) > Displaceable d = s  
> d.getArea()

**ok**    **error**

*Answer:Error: No 'getArea' method in 'Displaceable' with arguments: ()*

(c) > Area a = d

**ok**    **error**

*Answer:Error: Bad types in assignment*

(d) > if (d instanceof Area) d.getArea();

**ok**    **error**

*Answer:Error: No 'getArea' method in 'Displaceable' with arguments: ()*

(e) > ((Area)s).getArea()

**ok**    **error**

*Answer:9.0*

(f) > (s instanceof Area)

**ok**    **error**

*Answer:true*

The next questions concern the following (partial) implementation of an extensible 2D array of doubles. Note that the rows of this array do not have to be all of the same length. Your job will be to complete this class.

```
public class Extensible2DArray {
    private double[][] data;

    /* create an empty 2D array */
    public Extensible2DArray() {
        data = new double[][]{};
    }

    /* return the number of rows in the array */
    public int numRows() { return data.length; }

    /* return the number of columns in row i */
    public int numColumns(int i) { return data[i].length; }

    /* access the value at position i, j */
    public double get(int i, int j) { return data[i][j]; }

    /* given the current size and the desired newSize, return
       a number that is at least 50% larger than the current size. */
    private int growAmt(int current, int newSize) {
        return Math.max(newSize, (3*current)/2 + 1);
    }

    /* increase the number of columns in row i to be at least newSize */
    public void growCols(int i, int newSize) {
        double[] newData = new double[newSize];
        for (int k=0; k < data[i].length; k++)
            newData[k] = data[i][k];
        data[i] = newData;
    }
}
```

An example interaction of this class (when completed) is as follows:

```
> Extensible2DArray a = new Extensible2DArray()
> a.numRows()
0
> a.set(1,1,3)
> a.numRows()
2
> a.numColumns(0)
0
> a.numColumns(1)
2
> a.get(1,1)
3.0
```

4. (4 points) Complete the following implementation of the method `set`, which changes the value at location `(i, j)`, growing the array if necessary.

```
public void set(int i, int j, double value) {
    if (i >= data.length) {

        growRows(growAmt(_____, _____));
    }
    if (j >= data[i].length) {

        growCols(i, growAmt(_____, _____));
    }
    data[i][j] = value;
}
```

In the four blanks above, put the letters of the expressions from the list below that complete the method.

- (a) 0
- (b) `i-1`
- (c) `i`
- (d) `i+1`
- (e) `j-1`
- (f) `j`
- (g) `j+1`
- (h) `data.length`
- (i) `data[i].length`
- (j) `data[j].length`
- (k) `data[i][j].length`

*Answer:* `data.length, i+1, data[i].length, j+1`

5. (10 points) Implement the method `public void growRows(int newSize)`, which increases the number of rows to be `newSize`. You may assume that `newSize` is larger than the number of rows in the array. Make sure to double check that the sample interaction behaves appropriately for your implementation.

Hint: the syntax `new double[size][]` creates a new array of references to double arrays, of length `size`. All of the references in the array are initialized to `null`.

```
/* increase the number of rows to be newSize */
public void growRows(int newSize) {
    double[][] newData = new double[newSize][];
    for (int i = 0; i < newSize; i++)
        if (i < data.length)
            newData[i] = data[i];
        else
            newData[i] = new double[0];
    data = newData;
}
}
```

## IntSeq.java

```
1  /**
2   * Sequences of integers represented as arrays.
3   *
4   */
5  public class IntSeq {
6      private int[] data;
7
8      public IntSeq(int size) {
9          data = new int[size];
10     }
11     /**
12      * Create a sequence object from a given array.
13      *
14      * @param toCopy the array.
15      */
16     public IntSeq(int[] toCopy) {
17         data = new int[toCopy.length];
18         for (int i = 0; i < toCopy.length; i++)
19             data[i] = toCopy[i];
20     }
21     /**
22      * Get the length of the sequence.
23      *
24      * @return the length
25      */
26     public int size() { return data.length; }
27     /**
28      * Get an element of the sequence.
29      *
30      * @param i the index of the element
31      * @return the element
32      */
33     public int get(int i) { return data[i]; }
34     /**
35      * Does a word occur at a given position in this sequence?
36      *
37      * @param word the word.
38      * @param pos the position.
39      * @return whether the word occurs in this sequence at <code>pos</code>.
40      */
41     public boolean occursAt(IntSeq word, int pos) {
42         for (int i = 0; i < word.size(); i++) {
43             if (i + pos >= size() ||
44                 data[i+pos] != word.get(i))
45                 return false;
46         }
47         return true;
48     }
49     /**
50      * Find the position of the first occurrence of a word in this sequence.
51      *
52      * @param word the word.
53      * @return the position of <code>word</code>'s first occurrence, or
54      * -1 if <code>word</code> does not occur.
55      */
56     public int find(IntSeq word) {
```

```

57     int where = -1;
58     for (int pos = 0; pos <= size(); pos++)
59         if (occursAt(word, pos)) {
60             where = pos;
61             break;
62         }
63     return where;
64 }
65
66 /**
67  * Create a new sequence by concatenating another on the end.
68  *
69  * @param other the other sequence
70  * @return this + other
71  */
72 public IntSeq concat(IntSeq other) {
73     int newSize = this.size() + other.size();
74     IntSeq newData = new IntSeq(newSize);
75     int i;
76     for (i=0; i< size(); i++) {
77         newData.data[i] = this.get(i);
78     }
79     for (; i < newSize; i++) {
80         newData.data[i] = other.get(i-size());
81     }
82     return newData;
83 }
84
85 public String toString() {
86     StringBuffer b = new StringBuffer();
87     for (int i = 0; i < data.length; i++) {
88         if (i > 0)
89             b.append(' ');
90         b.append(data[i]);
91     }
92     return b.toString();
93 }
94
95 }

```

## Displaceable.java

```
1  /**
2   * Type of shapes that have a location and can be moved.
3   */
4  public interface Displaceable {
5      /**
6       * Get the X coordinate of this shape's location.
7       *
8       * @return the X coordinate
9       */
10     public double getX();
11     /**
12      * Get the X coordinate of this shape's location.
13      *
14      * @return the X coordinate
15      */
16     public double getY();
17     /**
18      * Move this shape by a given displacement vector.
19      *
20      * @param dx the X length of the displacement
21      * @param dy the Y length of the displacement
22      */
23     public void move(double dx, double dy);
24 }
```

## Area.java

```
1  /**
2   * Type of shapes that have area.
3   */
4  public interface Area {
5      /**
6       * Get the area of this shape.
7       *
8       * @return the area
9       */
10     public double getArea();
11 }
```

## Rectangle.java

```
1  /**
2   * A rectangle defined by its lower-left corner, width, and height.
3   */
4  public class Rectangle implements Displaceable, Area {
5      private Point lowerLeft;
6      private double width, height;
7      /**
8       * A new rectangle with given lower-left corner point, width, and height.
9       *
10      * @param lowerLeft the lower left corner
11      * @param width the width
12      * @param height the height
13      */
14     public Rectangle(Point lowerLeft, double width, double height) {
```

```

15     this.lowerLeft = lowerLeft;
16     this.width = width;
17     this.height = height;
18 }
19 /**
20  * Get the X coordinate of the rectangle's lower-left corner.
21  *
22  * @return the X coordinate
23  */
24 public double getX() { return lowerLeft.getX(); }
25 /**
26  * Get the Y coordinate of the rectangle's lower-left corner.
27  *
28  * @return the Y coordinate
29  */
30 public double getY() { return lowerLeft.getY(); }
31 /**
32  * Get the width of the rectangle.
33  *
34  * @return the width
35  */
36 public double getWidth() { return width; }
37 /**
38  * Get the height of the rectangle.
39  *
40  * @return the height
41  */
42 public double getHeight() { return height; }
43 /**
44  * Move the rectangle by a displacement vector.
45  *
46  * @param dx the X displacement
47  * @param dy the Y displacement
48  */
49 public void move(double dx, double dy) {
50     lowerLeft.move(dx, dy);
51 }
52 /**
53  * Get the area of the rectangle.
54  *
55  * @return the area
56  */
57 public double getArea() {
58     return width * height;
59 }
60 public String toString() {
61     return lowerLeft + "[" + width + " * " + height + "];"
62 }
63 }

```

## Square.java

```
1  /**
2   * A Rectangle with identical width and height.
3   */
4  public class Square extends Rectangle {
5      /**
6       * A new square with given lower-left corner and side.
7       *
8       * @param lowerLeft X lower-left corner
9       * @param side the side length
10     */
11     public Square(Point lowerLeft, double side) {
12         super(lowerLeft, side, side);
13     }
14     public String toString() {
15         return "(" + getX() + "," + getY() + ")" + "[" + getWidth() + "^2]";
16     }
17 }
```