• Answer all questions. Please give clear answers. If you give more than one answer, we will choose one to grade and ignore the other, so please clearly mark out any answer you do not want us to grade.

• Scratch paper is attached to this exam. Turn it in along with your exam.

• Make sure you clearly write your name, EID, lecture and discussion times, and TA on this page.

• On any question that asks you to write code, you must write Java code.

• You may not use any methods from the Arrays class on this test. You may not use concepts or classes that we have not covered in CS 305j.

• Before you begin the exam, put away caps, mp3 players, calculators and everything except a pencil or pen. Everything else must be placed out of sight underneath your desk.

• You must show your UT ID to the proctor before you leave the exam room in order to receive credit for this exam.

Good Luck

<table>
<thead>
<tr>
<th>Question</th>
<th>Points Lost</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>24, 4 each</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>16, 4 each</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
1. **Java Expressions.** For each Java expression in the left-hand column, write its value in the right-hand column. The way you write each value should indicate its type, e.g., 7.0 for a double (not 7), Strings in double quotes.

   a) (char)('b' + 3)

   b) "elvisLives".substring(2)

   c) !(a' != 'A') || ('elvisLives'.indexOf('e') > 0)

   d) 13 % 5

   e) 7/3

   f) "exam" + 1 + 2
2. **Code Simulation.** Consider the following method:

```java
public static void arrayMystery(int[] a) {
    for(int i = a.length - 2; i > 0; i--) {
        if(a[i+1] <= a[i-1]) {
            a[i]++;
        }
    }
}
```

For each call to this method, indicate what values would be stored in the array afterwards.

```java
int[] a1 = {42};
arrayMystery(a1);   a1 after method is called: _______________________

int[] a2 = {1, 8, 3, 6};
arrayMystery(a2);   a2 after method is called: _______________________

int[] a3 = {5, 5, 5, 5, 5};
arrayMystery(a3);   a3 after method is called: _______________________

int[] a4 = {1, 0, 1, 0, 0, 1, 0};
arrayMystery(a4);   a4 after method is called: _______________________
```
Implementing Classes. Write a class called `TimeSpan`, that stores a span of time in hours and minutes (for example, the time span between 8:00am and 10:30am is 2 hours, 30 minutes). Each `TimeSpan` object should have the following public methods:

- **`TimeSpan(int hours, int minutes)`**: constructs a `TimeSpan` object with the given time span of hours and minutes. If the specified number of hours or the specified number of minutes is negative, initialize the time span object to represent 0 hours and 0 minutes. Keep in mind that the number of minutes in your time span object should be between 0 and 59, so for the call
  ```java
  ...TimeSpan(15, 100);
  ```
  this would correspond to a time span of 16 hours and 40 minutes.

- **`TimeSpan()`**: constructs a `TimeSpan` object with hours and minutes set to 0.

- **`add(int hours, int minutes)`**: adds the given amount of time to the time span. For example, (2 hours, 15 min) + (1 hour, 45 min) = (4 hours). If the hours and minutes are not both non-negative, do not change this time span.

- **override the `equals(Object x)` from the `Object` class**: this method should return true if the argument is a `TimeSpan` object the represents the same time span as this object.

**Note:** The minutes should always be reported in the range of 0 to 59. That means that you may have to “carry” 60 minutes into a full hour.

**Note:** You will lose points if you do not use appropriate access specifiers (e.g., private, public, etc.) for the methods.

```java
public class TimeSpan {
    // instance variables
    private int hours;
    private int minutes;
}
```
3. Write the two argument constructor for the `TimeSpan` class, as specified above.

4. Write the no argument constructor for the `TimeSpan` class, as specified above. Invoke the other constructor in the body of this constructor.
5. Write the `add()` method, as specified above. Remember that `minutes` must be an integer from 0 to 59, inclusive.
6. Write the `equals()` method, as specified above.
Extra space for TimeSpan
7. **File Processing.** Write a static method `repeat` that takes a `Scanner` object which is already attached to a file. The file contains a sequence of integer/String pairs. Your method should print one line of output for each pair with the given `String` being repeated the specified number of times. For example, if the file contains the following data:

```
3 cake. 10 ! 2 13
```

Your method should produce this output:

```
cake.cake.cake.
!!!!!!!!!!!!
1313
```

Note that the first line has 3 copies of “cake.”, the second line contains 10 copies of “!” , and the third line contains 2 copies of “13”. There are no extra spaces added in the output. You should exactly reproduce this sample output. You may assume that the format of the file is correct, and consists of zero or more integer/String pairs.
Extra space for repeat
8. **Arrays.** Write a static method `isUnique` that takes an array of integers as its arguments and returns a boolean value indicating whether or not the values in the array are unique. The values in the array are unique if no value occurs more than once in the array.

**Note:** You may not call any methods from the `Arrays` class in your solution.

For example, consider this array:

```java
type list = {14, 0, -10, 8, 3, 0, 4, 11};
```

The call `isUnique(list)` should return `false`, since the value 0 appears more than once in this list.

Consider this array:

```java
type list = {19, 44, 2, 8, 3, -7, 7, 14};
```

Then the call `isUnique(list)` should return `true`, since there are no duplicated values in the list.
extra space for isUnique
9. **2D Arrays.** Write a static method `columnMax` that takes a 2-D array of `Strings` and returns a one-dimensional array of integers. In the returned array, the value at index `i` is the length of the maximum length string in column `i`. For example, consider this array:

```java
String[][] a = {{"help", "ok", "city lights"},
    {"elvisLives", "hello", "punch-out"},
    {"hurrah", "humptydumpty", "123"}};
```

Then the call `columnMax(a)` should return 

`[10, 12, 11]`

Note that in column 0, “elvisLives” is the longest string, and it has length 10. In column 1, “humptydumpty” is the longest string, and its length is 12. In column 2, “city lights” is the longest string, and it has length 11.
Extra space for columnMax