Com S 228
Fall 2014
Exam 1

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Name: _____

ISU NetID (username): _____

Recitation section (please circle one):

1. M 11:00 am (Caleb V and Jesse)
2. M 10:00 am (Bryan and Jacob)
3. M 4:10 pm (Anthony and Ben)
4. T 11:00 am (Nick and Arko)
5. T 10:00 am (Monica and Susan)
6. T 4:10 pm (Alex and Caleb B)

Closed book/notes, no electronic devices, no headphones. Time limit 60 minutes. Partial credit may be given for partially correct solutions.

- Use correct Java syntax for writing code.
- You are not required to write comments for your code; however, brief comments may help make your intention clear in case your code is incorrect.

If you have questions, please ask!

Question	Points	Your Score
1	28	
2	24	
3	24	
4	24	
Total	100	

1. (28 pts) Refer to the class hierarchy on pages 10–13 to answer the questions below. (It helps to peel off pages 10–18 from your exam sheets for code lookup convenience and scratch purpose.) For each section of code, fill in the box stating one of the following:

   - the output, if any, *or*
   - that there is a compile error (briefly explain the error), *or*
   - the type of exception that occurs at runtime.

`Skate is = new InlineSkate(3.5, .95);` `System.out.println(is.go(10));`	
`Skate s = new Skate(4.5);` `System.out.println(s.go(10));`	

`Mechanical m = new SkateBoard(.95);`
`Skate s = (Skate) m;`
`System.out.println(s.go(15));`	

`LocomotiveDevice ld = new InlineSkate(3.5, .95);`
`System.out.println(ld.go(25));`
`Skate s = (Skate) ld;`
`System.out.println(s.getMA());`	

`Skate s = new SkateBoard(.95);` `System.out.println(s.go(5));`	
`Mechanical m = new Bicycle(4.5, .92);` `System.out.println(m.getEfficiency());`	

2. (24 pts) You are given two classes `Complex` and `ComplexTuple` on the next page. They implement complex numbers and tuples of complex numbers, respectively.

   a) (10 pts) Add a `clone()` method to the `Complex` class; the method must override `java.lang.Object`'s `clone()` method. The signature is shown below; you just need to fill in the `try` and `catch` blocks.

```
@Override
public Object clone()
{
   try
   {
      Complex c = (Complex) super.clone();









   }
   catch (CloneNotSupportedException e)
   {

   }
}
```

   b) (14 Pts) Add a method to the `ComplexTuple` class that overrides the `equals()` method from `java.lang.Object` to perform a deep comparison between `this` object and an existing `ComplexTuple` object. Two `Complexes` are considered to be equal if they have the same real and imaginary parts. You can assume that `Complex` has a correctly implemented `equals()` method.

```
@Override
public boolean equals(Object another)
{
   // TODO









}
```

Sample code for Problem 2

```
public class Complex implements Cloneable
{
   private int re;  // real part

   private int im;  // imaginary part

   public Complex(int re, int im)
   {
      this.re = re;
      this.im = im;
   }

   @Override
   public boolean equals(Object o)
   {
      // Assume this is already implemented; the
      // implementation details irrelevant.
   }
}

public class ComplexTuple
{
   private Complex c1;
   private Complex c2;

   public ComplexTuple(Complex c1, Complex c2)
   {
      this.c1 = c1;
      this.c2 = c2;
   }
}
```

3. (24 pts) Determine the worst-case execution time of each of the following methods as a function of the length of the input array(s). Express each of your answers as big-O of a simple function (which should be the simplest and slowest-growing function you can identify). For convenience, the analysis of each part has been broken down into multiple steps. For each step, you just need to fill in the blank a big-O function as the answer (in the ***worst case*** always).

   a) (6 pts)

```
void methodA(int[] arr)
{
   int p = 0;
   for (int i = 0; i < arr.length; ++i)
   {
      for (int j = i; j < arr.length; ++j)
      {
         p = (p + arr[j]) % 2;
      }
   }
}
```

   Suppose that `arr` is of length n.

   i) Number of iterations of the outer `for` loop: _____

   ii) Number of iterations of the inner `for` loop: _____

   iii) Worst-case execution time: _____

   b) (6 pts) Assume that the method `foo()` takes $O(n)$ time and method `bar()` takes $O(n^2)$ time.

```
public static void methodB(int[] arr)
{
   int n = arr.length;
   while (n > 0)
   {
      foo(arr);
      n = n/3;
   }

   bar(arr)
}
```

   i) Number of iterations of the `while` loop: _____

   ii) Time per iteration: _____

iii) Total time for the `while` loop: _____

iv) Total worst-case execution time for `methodB`: _____

c) (6 pts)

```
public static int max(int[] arr, int i)
{
   if (i == arr.length-1)
      return arr[i];
   int curMax = max(arr,i+1);
   if arr[i] > curMax
      return arr[i];
   else
      return curMax;
}
```

Suppose `arr` has length n, where n is at least 1. Assume that we call `max(arr,0)`.

i) Number of recursive calls to `max`: _____

ii) Worst-case execution time: _____

d) (6 pts) Consider the following algorithm, which takes three `int` arrays `A`, `B`, and `C`, each of length n, and returns a new array that contains the combined elements of `A`, `B`, and `C` in sorted order. `Merge` is the algorithm for merging two sorted arrays that we saw in class.

```
Sort3(A,B,C):
   sort A using mergesort
   sort B using mergesort
   sort C using mergesort
   p = A.length, q = B.length, r = C.length
   create an empty array D of length p+q
   D = Merge(A,B)
   create an empty array E of length p+q+r
   E = Merge(D,C)
   return E
```

What is the big-O time complexity of this algorithm? (For partial credit, to the right of each step of the algorithm write down the big-O time that it takes.)

4. (24 pts) The following tables list the input array (first line), output array (last line), and internal array state in sequential order for each of the sorts that we have studied in class (SELECTIONSORT, INSERTIONSORT, MERGESORT, and QUICKSORT). The two $O(n^2)$ sorts print internal results as the last operation of their outer loops. MERGESORT prints the output array after each call to MERGE. QUICKSORT prints after each call to PARTITION. Array contents occupy rows in the following tables, with the top rows containing the input and proceeding down through time to the output on the bottom.

   There is exactly one right answer to each problem. For partial credit, please explain your reasoning in the space below (4 pts each).

   a)

4	6	0	3	1	7	2	5
0	6	4	3	1	7	2	5
0	1	4	3	6	7	2	5
0	1	2	3	6	7	4	5
0	1	2	3	6	7	4	5
0	1	2	3	4	7	6	5
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

   A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

   Reasoning: _____

   b)

4	6	0	3	1	7	2	5
4	6	0	3	1	7	2	5
0	4	6	3	1	7	2	5
0	3	4	6	1	7	2	5
0	1	3	4	6	7	2	5
0	1	3	4	6	7	2	5
0	1	2	3	4	6	7	5
0	1	2	3	4	5	6	7

   A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

   Reasoning: _____

c)

4	6	0	3	1	7	2	5
1	2	0	3	4	7	6	5
0	1	2	3	4	7	6	5
0	1	2	3	4	7	6	5
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

Reasoning: _____

d)

4	6	0	3	1	7	2	5
4	6	0	3	1	7	2	5
4	6	0	3	1	7	2	5
0	3	4	6	1	7	2	5
0	3	4	6	1	7	2	5
0	3	4	6	1	7	2	5
0	3	4	6	1	2	5	7
0	1	2	3	4	5	6	7

A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

Reasoning: _____

e)

1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

Reasoning: _____

_____

_____

f)

1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1

A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

Reasoning: _____

_____

_____

Sample Code for Problem 1

```
interface LocomotiveDevice
{
    String go(int effort);
}

interface MechanicalAdvantage
{
    String getMA();
}

interface Mechanical
{
    String getEfficiency();
}

abstract class HumanPoweredLocomotiveDevice implements LocomotiveDevice
{
    public static final double windResistance = 0.9;

    public String aggregateAdvantage()
    {
        return Double.toString(windResistance);
    }
}

abstract class Skate extends    HumanPoweredLocomotiveDevice
                    implements MechanicalAdvantage
{
    protected double mechanicalAdvantage;

    protected Skate(double mechanicalAdvantage)
    {
        this.mechanicalAdvantage = mechanicalAdvantage;
    }

    @Override
    public String getMA()
    {
        return Double.toString(mechanicalAdvantage);
    }
```

```
    @Override
    public String aggregateAdvantage()
    {
        return super.aggregateAdvantage() + " * " + getMA();
    }
}

class IceSkate extends Skate
{
    public IceSkate(double mechanicalAdvantage)
    {
        super(mechanicalAdvantage);
    }

    @Override
    public String go(int effort)
    {
        return aggregateAdvantage() + " * " + effort;
    }
}

class InlineSkate extends Skate implements Mechanical
{
    double efficiency;

    public InlineSkate(double mechanicalAdvantage,
                       double efficiency)
    {
        super(mechanicalAdvantage);
        this.efficiency = efficiency;
    }

    @Override
    public String getEfficiency()
    {
        return Double.toString(efficiency);
    }

    @Override
    public String aggregateAdvantage()
    {
        return super.aggregateAdvantage() + " * " + getEfficiency();
    }
```

```
    @Override
    public String go(int effort)
    {
        return aggregateAdvantage() + " * " + effort;
    }
}

class SkateBoard extends HumanPoweredLocomotiveDevice implements Mechanical
{
    double efficiency;

    public SkateBoard(double efficiency)
    {
        this.efficiency = efficiency;
    }

    @Override
    public String getEfficiency()
    {
        return Double.toString(efficiency);
    }

    @Override
    public String aggregateAdvantage()
    {
        return super.aggregateAdvantage() + " * " + getEfficiency();
    }

    @Override
    public String go(int effort)
    {
        return aggregateAdvantage() + " * " + effort;
    }
}

class Bicycle extends    HumanPoweredLocomotiveDevice
              implements Mechanical, MechanicalAdvantage
{
    double efficiency;
    double mechanicalAdvantage;
```

```
    public Bicycle(double mechanicalAdvantage,
                   double efficiency)
    {
        this.mechanicalAdvantage = mechanicalAdvantage;
        this.efficiency = efficiency;
    }

    @Override
    public String getEfficiency()
    {
        return Double.toString(efficiency);
    }

    @Override
    public String getMA()
    {
        return Double.toString(mechanicalAdvantage);
    }

    @Override
    public String aggregateAdvantage()
    {
        return super.aggregateAdvantage() + "␣*␣" +
               getMA() + "␣*␣" + getEfficiency();
    }

    @Override
    public String go(int effort)
    {
        return aggregateAdvantage() + "␣*␣" + effort;
    }
}
```

Scratch paper

Scratch paper

Scratch paper

Scratch paper

Scratch paper