CSCI 135 Final **Name: _____**
Fall 2011

This exams consists of 11 problems on the following 14 pages.

You may use your two-sided hand-written 8 ½ x 11 note sheet during the exam. No calculators, computers, or communication devices of any kind are permitted.

If you have a question, raise your hand and I will stop by. Since partial credit is possible, **write legibly and show your work**.

Problem	Points	Score
1	10	
2	9	
3	8	
4	12	
5	9	
6	8	
7	7	
8	10	
9	6	
10	6	
11	15	
Total	100	

1. **Static methods and arrays** (10 points).

a. Declare and implement a static method `majority` that takes three `boolean` values and returns true if and only if a majority of the values (2 or more) are true:

b. Complete the following method that is supposed to return an array of N random values in [0.0, 1.0).

```
public static double[] getRandNums(int N)
{

}
```

2. **Loops and debugging** (9 points). Consider the following program which is supposed to print the powers of 2 from 2^0 up to an including 2^N, where N is a non-negative integer that is read from the command line. So for example, if N was 3 the program is supposed to print:

```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
```

```
 1 public class PowersOfTwo
 2 {
 3    public static void main(String [] args)
 4    {
 5       int N = Double.parseDouble(args[0]);
 6       long result = 0;
 7       int i = 0;
 8       while (i < N)
 9       {
10          System.out.println("2^" + i + " = " + result);
11          result *= 2;
12          i++;
13       }
14    }
15 }
```

This program has three bugs.

A. Which bug prevents the program from *compiling* successfully? Identify the line number where the bug appears and give a correct version of this line of code.

Line number _____

Correct version:

B. Identify the line numbers of the two other bugs that prevent the program from producing the desired output. Give a correct version of each line of code.

Line number _____

Correct version:

Line number _____

Correct version:

3. **Java basics and Strings** (8 points). The 4 program outputs at the top were generated by one or more of the lettered program fragments (we omitted the class declaration and main method from the fragments). Write the letter of each program fragment into the box of the output it generates. Some of the programs make use of the `charAt()` method from the `String` class. Here is a description of this method:

`public char charAt(int index)`

Returns the `char` value at the specified index. An index ranges from `0` to `length() - 1`. The first `char` value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

% java Prog? abba abba **% java Prog? jklmn** nmlkj	**% java Prog? abba** true **% java Prog? jklmn** false	**% java Prog? abba** aabb **% java Prog? jklmn** jklmn	**% java Prog? abba** false **% java Prog? jklmn** true

A

```
String s = args[0];
boolean result = true;
for (int i = 0; i < s.length() / 2; i++)
{
   if (s.charAt(i) != s.charAt(s.length() - i - 1))
      result = false;
}
System.out.print(result);
```

B

```
String s = args[0];
System.out.print((s.length() % 2) == 0);
```

C

```
String s = args[0];
System.out.print((s.length() % 2) == 1);
```

D

```
int i = args[0].length() - 1;
while (i >= 0)
{
   System.out.print(args[0].charAt(i));
   i--;
}
```

E

```
String s = args[0];
int [] count = new int[256];
for (int i = 0; i < s.length(); i++)
   count[s.charAt(i)]++;
for (int i = 0; i < 256; i++)
{
   for (int j = 0; j < count[i]; j++)
      System.out.print((char) i);
}
```

F

```
String s = args[0];
boolean result = true;
for (int i = 0; i < s.length(); i++)
{
   if (s.charAt(i) == 'a')
   {
      System.out.print("false");
      return;
   }
}
System.out.print(result);
```

G

```
String s = args[0];
String r = "";
for (int i = 0; i < s.length(); i++)
   r = s.charAt(i) + r;
System.out.print(r);
```

H

```
switch (args[0].charAt(0))
{
   case 'a': System.out.print("aabb");  break;
   case 'j': System.out.print("jklmn"); break;
   case 'k': System.out.print("nmlkj"); break;
   case 'm': System.out.print("true");  break;
   case 'n': System.out.print("abba");  break;
   default:  System.out.print("false"); break;
}
```

4. **Classes and methods** (12 points). Recall that Java has a method that rounds a number to the nearest integer. This method has the signature: `static long round(double a)`. Consider the class:

```
public class Mystery
{
   private double a = 22.2;
   public static int b = 11;

   public static String go(int a)
   {
      return "" + Math.round(a * 0.1);
   }

   public static int go(double a, int b)
   {
      return (int) Math.round(a * b);
   }

   public static int go(double a)
   {
       return (int) Math.round(a);
   }
}
```

a. Circle the letter corresponding to the best description of the `go()` methods in the `Mystery` class.

i) Constructors
ii) Overloaded methods
iii) Abstract instance methods
iv) Inherited methods

b. The following statements appear in the `main()` method of the `Mystery` class. Give the output of each statement. If a statement would cause a compile error, write "compile error". If a statement would cause a runtime error, write "runtime error".

Statement	**Output**
`System.out.print(Mystery.go(0.6, 2));`	
`System.out.print(Mystery.go(3));`	
`System.out.print(Mystery.go(2.0, 3.0));`	
`System.out.print(Mystery.go());`	
`System.out.print((Mystery.go(3.2) < 4.0));`	
`System.out.print(Mystery.go(2, 40));`	
`System.out.print(Mystery.a);`	
`System.out.print(Mystery.b);`	
`System.out.print(Mystery.go(12).equals("42"));`	
`System.out.print((Mystery.go(3) < 3.0));`	

5. **Multidimensional arrays** (9 points). Consider the following program:

```
public class Battleship1
{
   public static void main(String [] args)
   {
      final int N = Integer.parseInt(args[0]);
      final int L = Integer.parseInt(args[1]);
      boolean[][] hit = new boolean[N][N];

      for (int i = 0; i < L; i++)
      {
         int x = (int) (Math.random() * N);
         int y = (int) (Math.random() * N);
         hit[x][y] = true;
      }

      for (int x = 0; x < N; x++)
      {
         for (int y = 0; y < N; y++)
         {
            if (hit[x][y])
               System.out.print('*');
            else
               System.out.print('.');
         }
         System.out.println();
      }
   }
}
```

Circle the ***letter(s)*** of all outputs that could ***possibly*** be produced by running the following:

```
% java Battleship1 6 5
```

A	B	C	D	E	F	G
*.....	....*. .*..*. .*.... ...*..	....	...*. .*... .*..* ...*. ..*..	....*	No output: runtime error	No output: stuck in an infinite loop

5. **(continued)** The following program is identical to the previous one except for the starred lines.

```
public class Battleship2
{
   public static void main(String [] args)
   {
      final int N = Integer.parseInt(args[0]);
      final int L = Integer.parseInt(args[1]);
      boolean[][] hit = new boolean[N][N];

      for (int i = 0; i < L; i++)
      {
**       int x, y;
**       do
**       {
**          x = (int) (Math.random() * N);
**          y = (int) (Math.random() * N);
**       } while (hit[x][y]);
         hit[x][y] = true;
      }

      for (int x = 0; x < N; x++)
      {
         for (int y = 0; y < N; y++)
         {
            if (hit[x][y])
               System.out.print('*');
            else
               System.out.print('.');
         }
         System.out.println();
      }
   }
}
```

Circle the ***letter(s)*** of all outputs that could ***possibly*** be produced by running the following:

```
% java Battleship2 6 5
```

A	B	C	D	E	F	G
*.....	....*. .*..*. .*.... ...*..	.... *****.	..... ******	****** ****** ****** ****** ****** ******	No output: runtime error	No output: stuck in an infinite loop

Circle the ***letter(s)*** of all outputs that could ***possibly*** be produced by running the following:

```
% java Battleship2 6 40
```

A	B	C	D	E	F	G
*.....	..... .*.... .*...* ...*.. .*..*.	....*. ..*... ...*.. *....*	.*.**. *.**** ****.* .*..** ****.* .**.*.	****** ****** ****** ****** ****** ******	No output: runtime error	No output: stuck in an infinite loop

6. **Standard input, using objects** (8 points). You are writing the main program for your company's latest and greatest product: a bubble simulator. The simulator loads a set of bubbles that are specified in a text file. The text file has a line for each bubble. Each line has the starting x-location, starting y-location, and a y-velocity (in that order). Here is an example file containing 3 bubbles:

```
0.2 0.0 0.01
0.3 0.1 0.02
0.6 0.2 0.005
```

Luckily you already have two bubble-related classes to help in building the simulator:

```
public class Bubble
-------------------------------------------------------------------------------------------
public         Bubble(double x, double y, double vy) // New bubble @ (x,y) with y-velocity vy
public void    move()                                // Update position based on the velocity
public boolean isAbove(double y)                     // See if bubble's y-position is above y
public void    draw()                                // Draw the bubble

public class Bubbles
-------------------------------------------------------------------------------------------
public      Bubbles()                        // Create a collection that stores bubble objects
public void add(Bubble b)                    // Add the bubble b to the collection
public void draw()                           // Draw all the bubbles in the collection
public void move()                           // Update position of all bubbles in the collection
public void prune(double y)                  // Get rid of any bubbles with a y-position above y
public int  size()                           // Find out how many bubbles are in the collection
```

The simulator reads in data using `StdIn`. The bubbles are animated, moving according to their velocity. The sim continues as long as any bubbles remain. Using methods in the above APIs, fill in the missing parts of the program.

```
public class BubbleSim
{
   public static void main(String [] args)
   {
      // Create an object to store all the bubbles


      // Read in data using StdIn and store all the bubbles









      // Main animation loop, keeps going as long as there are bubbles remaining
      while (                          )
      {
         StdDraw.clear();









         StdDraw.show(100);
      }
      System.out.println("All done!");
   }
}
```

7. **Creating objects, arrays** (7 points). Disaster! Your bubble simulator was working smashingly, but then a stray neutrino hit your computer and corrupted your `Bubbles.java` file. There are pieces of code everywhere. What's worse is your correct code got mixed up with some of your old buggy code. Reconstruct `Bubbles.java` by putting the letter of the correct code fragment in each box.

```
import java.util.*;

public class Bubbles
{
   [          ]

   public Bubbles()
   {
      [          ]
   }

   public void add(Bubble b)
   {
      [          ]
   }

   public void draw()
   {
      [          ]
   }

   public void move()
   {
      [          ]
   }

   public void prune(double y)
   {
      [          ]
   }

   public int size()
   {
      [          ]
   }
}
```

A.
```
private Bubble [] bubbles;
private int index = 0;
```

B.
```
private ArrayList<Bubble> bubbles;
```

C.
```
private ArrayList<double> x  = new ArrayList<double>();
private ArrayList<double> y  = new ArrayList<double>();
private ArrayList<double> vy = new ArrayList<double>();
```

D.
```
bubbles = new Bubble[];
```

E.
```
bubbles = new Bubble[index++];
```

F.
```
bubbles = new ArrayList<Bubble>;
```

G.
```
bubbles = new ArrayList<Bubble>();
```

H.
```
bubbles[index++] = b;
```

I.
```
x.add(b.x); y.add(b.y); vy.add(b.vy);
```

J.
```
bubbles.add(b);
```

K.
```
return index;
```

L.
```
return bubbles.size();
```

M.
```
for (int i = size() - 1; i >= 0; i--)
{
   if (bubbles.get(i).isAbove(y))
      bubbles.remove(i);
}
```

N.
```
for (Bubble b : bubbles)
   b.move();
```

O.
```
for (Bubble b : bubbles)
   b.draw();
```

P.
```
for (int i = 0; i <= bubbles.size(); i++)
{
   if (bubbles.get(i).isAbove(y))
      bubbles.remove(i);
}
```

Q.
```
bubbles.remove(y);
```

R.
```
bubbles[index].isAbove(y) = null;
```

S.
```
for (int i = 0; i < index; i++)
   b[i].move();
```

T.
```
for (int i = 0; i < index; i++)
   b[i].draw();
```

8. **Debugging, OOP** (10 points). After fixing your `Bubbles.java` class, you run your simulator but all the bubbles appear in the lower-left corner and just sit there. The problem is somewhere in `Bubble.java`:

```
public class Bubble
{
	double		x	= 0.0;
	double		y	= 0.0;
	double		vy	= 0.0;
	final double	SIZE	= 0.01;

	public Bubble(double x, double y, double vy)
	{
		x  = x;
		y  = y;
		vy = vy;
	}

	public void move()
	{
		y += vy;
	}

	public boolean isAbove(double y)
	{
		return (this.y > y);
	}

	public void draw()
	{
		StdDraw.circle(x, y, SIZE);
	}









}
```

a. Correct the above code so that your simulator runs correctly (just write your corrections in the above box).

b. While debugging, you printed out a `Bubble` object `b` by doing `System.out.println(b)`. You got the fairly useless result of "`Bubble@1cb1c5fa`". You would rather it print something more informative like:
`(0.10, 0.31) vel=0.005`
Add a method that causes `Bubble` objects to print like this when they are passed to `println()`.

c. Your show your `Bubble.java` class to your boss who looks at you disapprovingly and says "In this office, we obey the laws of data encapsulation!" Fix the above code to appease your boss.

d. You show your code to your cubemate, a hotshot know-it-all programmer. He smirks and asks "While I like how you used a named constant for `SIZE`, it is costing 8 bytes of memory per `Bubble` object." Change the declaration of `SIZE` so only 8 bytes are required for `SIZE` regardless of the number of `Bubble` objects.

9. **Object inheritance** (6 points). To track items in your online store, you created the class hierarchy shown on the left. Answer the questions on the right. Each question has ***1 or more correct answers***.

```
public abstract class Item
{
   protected String ISBN  = "";
   protected double price = 0.0;

   public double getPrice()
   {
      return price;
   }

   public String toString()
   {
      return ISBN + ", " + price;
   }

   public abstract double getShipCost(int qty);

}
```

```
public class Book extends Item
{
   private String author = "";
   private String title  = "";
   private static final double SHIP = 4.95;

   public String toString()
   {
      return title + " by " + author +
             ", " + price;
   }

   public double getShipCost(int qty)
   {
      return qty * SHIP;
   }
}
```

```
public class CD extends Item
{
   private String artist = "";
   private String album  = "";
   private static final double SHIP = 2.95;

   public String toString()
   {
      return album + " by " + artist +
             ", " + price;
   }

   public double getShipCost(int qty)
   {
      return qty * SHIP;
   }
}
```

a. Inside the toString method of the Item class, which instance variable(s) is/are accessible?

i. ISBN
ii. price
iii. author
iv. title
v. artist
vi. album

b. Inside the toString method of the CD class, which instance variable(s) is/are accessible?

i. ISBN
ii. price
iii. author
iv. title
v. artist
vi. album

c. Assume variable b is of type Book. What toString methods are called when the line System.out.println(b) executes?

i. First Item's method, then Book's method
ii. Only Book's method
iii. First Book's method, then Item's method
iv. Only CD's method

d. You have an array of type Item. What object types can you successfully create (using the new operator) and put into the array?

i. Item
ii. Book
iii. CD

e. You have an array of type Book. What object types can you successfully create (using the new operator) and put into the array?

i. Item
ii. Book
iii. CD

f. If Book.java did not implement a getShipCost method, what would happen?

i. Item's getShipCost method would get called instead
ii. CD's getShipCost method would get called instead
iii. Item.java would fail to compile
iv. Book.java would fail to compile

10. **Defensive programming , exception handling** (6 points). You have a program that prints N random numbers between 0.0 and X. N is an `int` that is specified as the first command line argument. X is a `double` that is specified as the second command line argument. Here is your current program:

```
public class RandomGen
{
   public static void main(String [] args)
   {
      final int N = Integer.parseInt(args[0]);
      final double X = Double.parseDouble(args[1]);
      for (int i = 0; i < N; i++)
         System.out.println(Math.random() * X);
   }
}
```

Your program crashes in 3 different cases. Here is the current output and the desired output for these cases:

Case	Current output	Desired output
Not enough command line arguments	**% java RandomGen** Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0 at RandomGen.main(RandomGen.java:6) **% java RandomGen 10** Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1 at RandomGen.main(RandomGen.java:7)	**% java RandomGen** Need 2 args! **% java RandomGen 10** Need 2 args!
First argument not an integer	**% java RandomGen foo 2.0** Exception in thread "main" java.lang.NumberFormatException: For input string: "foo" at java.lang.NumberFormatException.forInputString (NumberFormatException.java:48) at java.lang.Integer.parseInt(Integer.java:449) at java.lang.Integer.parseInt(Integer.java:499) at RandomGen.main(RandomGen.java:6)	**% java RandomGen foo 2.0** Can't parse args!
Second argument not a floating-point value	**% java RandomGen 10 ick** Exception in thread "main" java.lang.NumberFormatException: For input string: "ick" at sun.misc.FloatingDecimal.readJavaFormatString (FloatingDecimal.java:1222) at java.lang.Double.parseDouble(Double.java:510) at RandomGen.main(RandomGen.java:7)	**% java RandomGen 10 ick** Can't parse args!

(continued on next page)

10. **(continued)** Create a new version of your program that prints the error messages given in the desired output column rather than crashing with a runtime error.

```
public class RandomGen
{
   public static void main(String [] args)
   {









      for (int i = 0; i < N; i++)
         System.out.println(Math.random() * X);









   }
}
```

11. **Creating objects** (15 points). You are tasked with completing a class that represents a balloon. A balloon has a current volume of air, a maximum volume of air, and some text that appears on the balloon (e.g. "Happy Birthday!", "It's a girl!", etc). The volume of air is a floating-point value. The current class is missing its instance variables and the implementation of its methods. Fill in the missing parts.

```
public class Balloon
{
   private static final double VOL_PER_PUMP = 1.3;  // Volume of air per pump

   // Instance variables









   // Create a new empty balloon with the given maximum volume and text
   public Balloon(double maxVol, String text)
   {









   }

   // Getter method for the current volume of the balloon
   public double getCurrentVolume()
   {









   }
```

(continued on the next page)

11. **Creating objects (continued)**

```
        // Pump up the balloon a certain number of times.
        //
        // A single pump adds a volume of air specified by the constant VOL_PER_PUMP.
        // If the pumping results in exceeding the maximum volume, the balloon
        // pops and the current and maximum volume are set to 0.
        //
        // No pumping should occur if numPumps is <= 0.
        //
        // Returns the current volume after the pumping is completed.
        public double pump(int numPumps)
        {









        }
}
```