CSCI 136 Written Exam #1 **Name: ___________________________________**
Fundamentals of Computer Science II
Spring 2015

This exam consists of 5 problems on the following 7 pages.

You may use your double-sided hand-written 8 ½ x 11 note sheet during the exam. No computers, mobile devices, cell phones, or other 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, **please write legibly and show your work**.

Problem	**Points**	**Score**
1	**10**	
2	**8**	
3	**18**	
4	**10**	
5	**10**	
Total	**56**	

1. **Arrays** (10 points). Consider the following program:

```java
public class Prob1
{
    public static void main(String[] args)
    {
        for (int i = 0; i < args.length; i = i + 2)
        {
            if (args[i].equals(args[i + 1]))
                System.out.print("yes ");
            else
                System.out.print("no ");
        }
        System.out.println("maybe");
    }
}
```

Below are five example executions of the program. Give the output produced by the program. If the given input would cause a runtime error, write “error”.

Command line	**Output**
`% java Prob1`	maybe
`% java Prob1 a a b b`	yes yes maybe
`% java Prob1 a b a b a`	error
`% java Prob1 a AB ab a`	no no maybe
`% java Prob1 abcdefg`	error

2. Threads (8 points). Consider the following two classes that differ only in name and the highlighted line:

```
public class WorkerA implements Runnable
{
  private int [] d = null;

  public WorkerA(int [] d)
  {
    this.d = d;
  }

  public void reverse()
  {
    for (int i = 0; i < d.length / 2; i++)
    {
      int swap = d[i];
      d[i] = d[d.length - i - 1];
      d[d.length - i - 1] = swap;
    }
  }

  public void run()
  {
    for (int i = 0; i < 999; i++)
      reverse();
  }
}
```
	```
	public class WorkerB implements Runnable
	{
	  private int [] d = null;

	  public WorkerB(int [] d)
	  {
	    this.d = d;
	  }

	  public synchronized void reverse()
	  {
	    for (int i = 0; i < d.length / 2; i++)
	    {
	      int swap = d[i];
	      d[i] = d[d.length - i - 1];
	      d[d.length - i - 1] = swap;
	    }
	  }

	  public void run()
	  {
	    for (int i = 0; i < 999; i++)
	      reverse();
	  }
	}
	```

Each of the following code snippets uses either `WorkerA` or `WorkerB`. Give the program's output. If the program could ***potentially*** have different outputs depending on the thread scheduler, just write ***varies***.

```
int [] d = {0, 1, 2, 3, 4, 5};
WorkerA w = new WorkerA(d);

Thread t1 = new Thread(w);
t1.start();

try
{
  t1.join();
}
catch (InterruptedException e) { }

for (int i : d)
  System.out.print(i + " ");
```
	```
	Output:

	5 4 3 2 1 0
	```

2. Threads (continued)

```
int [] d = {0, 1, 2, 3, 4, 5};
WorkerA w = new WorkerA(d);

Thread t1 = new Thread(w);
Thread t2 = new Thread(w);
t1.start();
t2.start();

try
{
  t1.join();
  t2.join();
}
catch (InterruptedException e) { }

for (int i : d)
  System.out.print(i + " ");
```

```
Output:

varies
```

```
int [] d = {0, 1, 2, 3, 4, 5};
WorkerB w = new WorkerB(d);

Thread t1 = new Thread(w);
Thread t2 = new Thread(w);
t1.start();
t2.start();

try
{
  t1.join();
  t2.join();
}
catch (InterruptedException e) { }

for (int i : d)
  System.out.print(i + " ");
```

```
Output:

0 1 2 3 4 5
```

```
int [] d = {0, 1, 2, 3, 4, 5};
WorkerB w = new WorkerB(d);

Thread t1 = new Thread(w);
t1.start();

for (int i : d)
  System.out.print(i + " ");
```

```
Output:

varies
```

3. Multiple choice (18 points, 2 points each). For each question, circle the single ***BEST*** answer.

a) In Java socket programming, which of the following lines causes a ***server*** program to block (i.e. suspend its execution) until a client requests a new connection:

```
  I.  InputStreamReader stream = new InputStreamReader(sock.getInputStream());
 II.  BufferedReader reader = new BufferedReader(stream);
III.  PrintWriter writer = new PrintWriter(sock.getOutputStream());
 IV.  Socket sock = new Socket("localhost", 5000);
```
[✓ V. `Socket sock2 = sock.accept();`]

b) You are developing a client-server program using Java sockets. The client program establishes a connection to the server and has a multi-step conversation over the course of many seconds. What problem would result if you implement a ***single-threaded server*** utilizing a single `Socket` object?

[✓ I. If a single client is connected to the server, no other clients will be able to receive service until the first client finishes.]
II. The client would not be able to obtain the IP address of the server via DNS.
III. Multiple clients could connect at the same time, but the server would be slow to respond since it would not be able to use any multiple processor cores present on the server.
IV. Deadlock would occur due to concurrency issues handling the simultaneous client requests.
V. The server would quickly exhaust its available pool of socket port numbers.

c) You are developing a client-server program using Java sockets. When establishing the connection, the client provides both a domain name and a port number. What is the purpose of the ***port number***?

I. The client uses the port number to translate the domain name of the server to an IP address.
II. Allows the client's connection to work through all firewalls.
[✓ III. Allows the client to target a specific process running on the target host machine.]
IV. Allows the client to target a specific host machine somewhere on the network.
V. Specifies the version of the IP protocol that the server should use for communication.

d) Which of the following is ***TRUE*** about threading in Java:

I. Creating more Java `Thread` objects than the number of CPU cores causes a runtime exception.
II. When a thread calls `Thread.sleep(1000)`, it yields the CPU for a maximum of 1000 milliseconds.
III. Catching an `InterruptedException` is optional when calling `join()` or `sleep()`.
[✓ IV. Calling `join()` on a worker thread causes the program to wait for that worker thread to complete.]
V. The `Runnable` interface requires you implement a single method, namely `int compareTo()`.

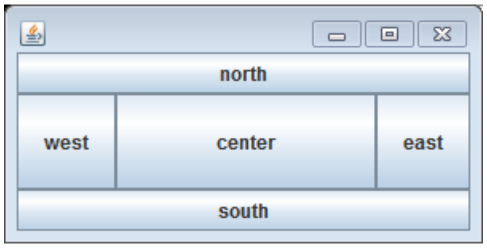
e) You are programming a GUI in Java. Which of the following Java interfaces is used to respond when a user clicks on a `JButton` widget?

I. `Iterable`
II. `Runnable`
III. `Comparable<JButton>`
IV. `Comparable`
[✓ V. `ActionListener`]

f) You are programming a GUI in Java. You need an area to display images, lines, and circles. What widget would be most appropriate?

[✓ I. `JPanel`]
II. `JLabel`
III. `JButton`
IV. `JDrawingArea`
VI. `JDrawable`

g) Consider the following screenshot of a Java GUI that uses the default layout manager for a `JFrame`:



What layout manager is this?
I. `FlowLayout`
II. `BoxLayout`
[✓ III. `BorderLayout`]
IV. `CompassLayout`
V. `CenterLayout`

The next two questions are about the following class:

```
public class Cow extends Animal implements Comparable<Cow>
{
    public Cow()
    {
        super("Bessy");
        // ...more code...
    }

    // ...more code...
}
```

h) What does the first line of the constructor do?
I. Calls a constructor in the class `Cow` that takes a single String as a parameter.
[✓ II. Calls a constructor in the class `Animal` that takes a single String as a parameter.]
III. Calls a constructor in the class Comparable that takes a single String as a parameter.
IV. Calls a method `Bessy` in the class `Cow` that takes a single String as a parameter.
V. Calls a method `Bessy` in the class `Animal` that takes a single String as a parameter.

i) What other method must be implemented in `Cow` in order for the class to compile?
I. `public Cow(String name)`
II. `public Animal(String name)`
[✓ III. `public int compareTo(Cow other)`]
IV. `public void run()`
V. `public int run(Cow other)`

4. Classes and collections (10 points). The following class represents a dog character in a game. Each Dog can be at a different (x, y) location in the game. All dogs are drawn using the same image and make the same barking sound. Add or change the code to achieve the following:

a) Reduce the memory requirements by adding a keyword to one or more of the instance variables.

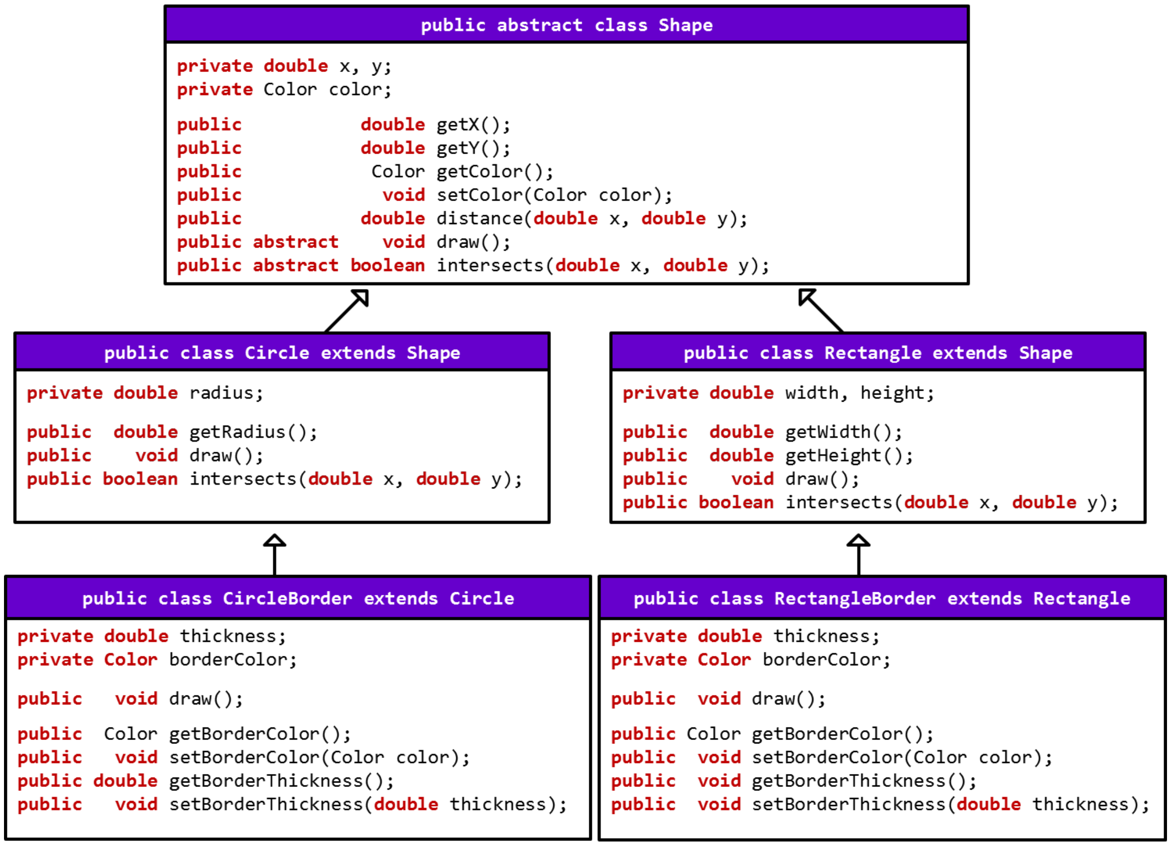
b) Add code in `main()` to create and populate an `ArrayList` containing the number of dogs specified by the 1st command line argument. Give each dog a random x- and y-location, both in [0.0, 1.0).

c) Add an instance variable and code to constructor to keep track of the total number of Dog objects constructed. Your code should work in any client that creates Dog objects via the **new** operator.

d) Add a `getCount()` method to return the total number of created Dog objects.

```
public class Dog
{
   private double x = 0.0;
   private double y = 0.0;

   private static final String IMAGE = "dog.jpg";
   private static final String AUDIO = "dog.wav";
   private static int count = 0;

   public Dog(double x, double y)
   {
      count++;
      this.x = x;
      this.y = y;

   }

   public static int getCount()
   {
      return count;
   }

   public static void main(String [] args)
   {
        final int N = Integer.parseInt(args[0]);

        ArrayList<Dog> dogs = new ArrayList<Dog>();

        for (int i = 0; i < N; i++)
        {

            dogs.add(new Dog(Math.random(), Math.random()));

        }

   }
}
```

5. Methods and inheritance (10 points). Consider the following hierarchy of classes:



```
public abstract class Shape

private double x, y;
private Color color;

public          double getX();
public          double getY();
public           Color getColor();
public            void setColor(Color color);
public          double distance(double x, double y);
public abstract   void draw();
public abstract boolean intersects(double x, double y);
```

```
public class Circle extends Shape

private double radius;

public  double getRadius();
public    void draw();
public boolean intersects(double x, double y);
```

```
public class Rectangle extends Shape

private double width, height;

public  double getWidth();
public  double getHeight();
public    void draw();
public boolean intersects(double x, double y);
```

```
public class CircleBorder extends Circle

private double thickness;
private Color borderColor;

public   void draw();

public  Color getBorderColor();
public   void setBorderColor(Color color);
public double getBorderThickness();
public   void setBorderThickness(double thickness);
```

```
public class RectangleBorder extends Rectangle

private double thickness;
private Color borderColor;

public  void draw();

public Color getBorderColor();
public  void setBorderColor(Color color);
public  void getBorderThickness();
public  void setBorderThickness(double thickness);
```

For each part, list the class or classes meeting the stated condition. ***If no classes do, write "none".***

a) Classes that can be ***instantiated*** using the **new** operator

Circle, Rectangle, CircleBorder, RectangleBorder

b) Classes that ***override*** an implemented method in a parent (i.e. they override a non-abstract method)

CircleBorder, RectangleBorder

c) Classes that have one or more ***overloaded*** methods

none

d) Subclasses of `Rectangle`

`RectangleBorder`

e) Subclasses of `CircleBorder`

none