NAME: ____________________________

CMPSCI 187
Programming With Data Structures
First Midterm Exam Fall 2012

D. A. M. Barrington 9 October 2012

DIRECTIONS:

- Answer the problems on the exam pages.
- There are seven questions on pages 3-10, most with multiple parts, for 100 total points. Probable scale is somewhere around A=93, C=69, but will be determined after I grade the exam.
- If you need extra space use the back of a page.
- No books, notes, calculators, or collaboration.

1	/20
2	/10
3	/15
4	/15
5	/10
6	/10
7	/20
Total	/100

Questions 3-6 all deal with the following class (which is exactly the same as the class on the first midterms from last year):

```
import java.util.*; // we will use Stack
public class Dog {
     private String name;
     private int age;

     public Dog (String newName, int newAge) {
          name = newName;
          age = newAge;}

     public String getName {return name;}
     public void setName (String newName) {name = newName;}
     public int getAge (return age;}
     public void setAge (int newAge) {age = newAge;}}
```

Question 7 uses this generic class from DJW:

```
public class LLNode<T> {
    private LLNode link;
    private T info;
    public LLNode (T info) {this.info = info; link = null;}
    public void setInfo (T info) {this.info = info;}
    public T getInfo( ) {return info;}
    public void setLink (LLNode link) {this.link = link;}
    public LLNode getLink ( ) {return link;}}
```

Question 1 – Java Concepts (20): Briefly explain the difference between the two concepts in each pair (2 points each): indicated:

- (a) `String [ ]` and `String [ ] [ ]`
- (b) the `ArrayStringLog( )` and `ArrayStringLog (int k)` constructors in DJW
- (c) software and code
- (d) $O(1)$ running time and $O(n)$ running time
- (e) the `ArrayStringLog` (DJW) and `StringBag` (Project #1) classes
- (f) postfix expressions and infix expressions
- (g) guarding against an exception and catching an exception
- (h) `pop` in `java.util.Stack` and `pop` in DJW's stack classes
- (i) objects and primitives
- (j) observers and transformers

Question 2 – Software Engineering (10): Briefly discuss (in English) how you would make the following modifications to the code for the specified program, with specific reference to the code (5 points each):

- (a) In Project 1, to add a new method to `StringBag` called `replace`, which takes a `String` as a parameter and replaces a randomly-chosen element of the bag with the parameter, keeping all the other elements in the same position.

- (b) In Project 2, to add a method `isUnique` to `SudokuSolver` that takes a `Board` as parameter and returns a boolean that is `true` if the `Board` denotes a puzzle with *one and only one* solution. (The return value should be `false` if there is no solution or if there are more than one.)

Question 3 – Tracing Code (15): Determine the output value of the following blocks of code. In each case, assume that the Dog class from page 2 is present. Include a brief justification of your answer.

- (a)

```
// uses java.util.Stack names for methods
Stack<Dog> left = new Stack<Dog>;
Stack<Dog> right = new Stack<Dog>;
right.push(new Dog("Duncan", 3)):
right.push(new Dog("Biscuit", 3));
right.push(new Dog("Cardie", 5));
right.push(new Dog("Ace", 6));
Dog newDog = new Dog("Sydney", 3));
while (!right.empty( ) && (right.peek( ).getAge( ) > newDog.getAge ( ))
    left.push(right.pop( ));
right.push(newDog);
System.out.println(left.peek( ).getName( ));
```

- (b)

```
int [ ] row = new int[9];
for (int i = 0; i < 9; i++)
    row[i] = i + 1;
boolean duplicate = false;
for (int j = 0; j < 9; j++)
    for (int k = 0; k < 9; k++)
        if (row[j] == row[k]) duplicate = true;
if (duplicate)
   System.out.println ("Duplicate element exists");
else System.out.println ("Elements are unique");
```

- (c)

```
Dog ace = new Dog ("Ace", 6);
Dog biscuit = new Dog ("Biscuit", 3);
Dog cardie = new Dog ("Cardie", 5);
Dog golden = cardie;
Dog pointer = ace;
pointer.setName("Cardie");
golden.setName(ace.getName( ));
biscuit.setName(cardie.getName( ));
pointer.setName("Biscuit");
System.out.println(ace.getName( ));
System.out.println(cardie.getName( ));
```

Question 4 – Finding Errors (15): Each of the following code fragments has a specific error that either prevents it from compiling, will cause an exception if it is run, or will cause it to produce a *clearly* unintended output. Find the error and explain what will happen (5 points each):

- (a) (A new generic class)

  ```
  public class Group<T> {
      private T [ ] elements;
      public Group ( ) {
          elements = new T[100];}}
  ```

- (b) (Uses DJW syntax for stacks)

  ```
  ArrayStack<Dog> as = new ArrayStack<Dog> (10);
  as.push (new Dog ("Ace", 6));
  as.push (new Dog ("Biscuit", 3));
  as.push (new Dog ("Cardie", 5));
  while (as.top( ) != null)
      as.pop( );
  as.push (new Dog ("Cardie", 5));
  ```

- (c)

  ```
  Integer [ ] ia = new Integer [10];
  for (int i = 0; i < 5; i++)
      ia[i] = i;
  for (int j = 0; j < ia.length; j++)
      ia[j]++;
  ```

Question 5 – Timing Analysis (10): Find the worst-case asymptotic running time of each block of code, as a function of the input size N (5 points each):

- (a) (a method to be added to the `LinkedStringLog` class)

```
public int sumLengths ( ) {
     int sum = 0;
     LLStringNode cur = log; // "log" is the head of the list of N elements
     while (cur != null) {
          sum += cur.getInfo( ).length( );
          cur = cur.getLink( );}
     return sum;}
```

- (b) (This is pseudocode and uses the classes from Project #2.)

```
// create a Board object b with all cells unfixed
// read an array of N Move objects and call b.move( ) for each
// set all nonzero elements of b to be fixed
// solve the resulting sudoku puzzle using the solution to Project \#2
```

Question 6 – Short Code Writing (10): Write a class `Kennel` so that each `Kennel` object will have an array `dogs` of `Dog` objects, and an `int` variable `size` that will keep track of how many dogs are in the array. Don't worry about error handling – if bad input to one of your methods causes an exception, that is fine. The class should have the following methods:

- a constructor `Kennel (int k)` that creates an object that can hold up to `k` `Dog` objects,
- a `boolean` method `free (int pos)` that tells whether position `pos` now has no `Dog` in it,
- a void method `insert(Dog d, int pos)` that puts `d` into position `pos` if it is free, and does nothing if it is not,
- a method `remove (int pos)` that removes and returns the `Dog` in position `pos`, or returns `null` if there is no `Dog` there, and
- a void method `consolidate( )` that moves all the `Dog` objects to an initial segment of the locations – to positions `0` through `size - 1`, where `size` is the number of `Dog` objects currently stored.

Question 7 – Long Code Writing (20): In this question you are to give complete definitions of two classes, whose objects will each be linked lists of `Dog` objects. An `AscendingAgeDogList` must have the property that every `Dog` in it has an age less than or equal to that of any `Dog` following it. A `DescendingAgeDogList` has the opposite property, that every `Dog` in it has an age greater than or equal to that of any `Dog` following it. (You may abbreviate these classes `AADL` and `DADL`.)

(Note: These two classes are obviously going to be similar to each other. You may save yourself writing by designing one of them, and then *clearly* indicating what has to be changed to get the other.)

Using the `LLNode<Dog>` class from page 2, define each class to have the following methods:

- a constructor with no parameters that makes an empty list,
- a `void` method `add (Dog d)` that puts `d` at the head of the list if that meets the list's conditions, and does nothing otherwise,
- a boolean method `canAdd (Dog d)` that tells whether `d` may legally be added at the head of the list,
- a method `remove( )` that removes and returns the `Dog` from the head of the list, returning `null` if the list is empty,
- a boolean method `isEmpty( )` telling whether the list is empty, and
- a `void` method `insert (Dog d)` that inserts `d` into the list at the first legal place.

(**Hint:** The most difficult method to implement is `insert`. But you can do it by removing the objects that are in the way of the place to insert, and storing them in an object of the *other* class.)