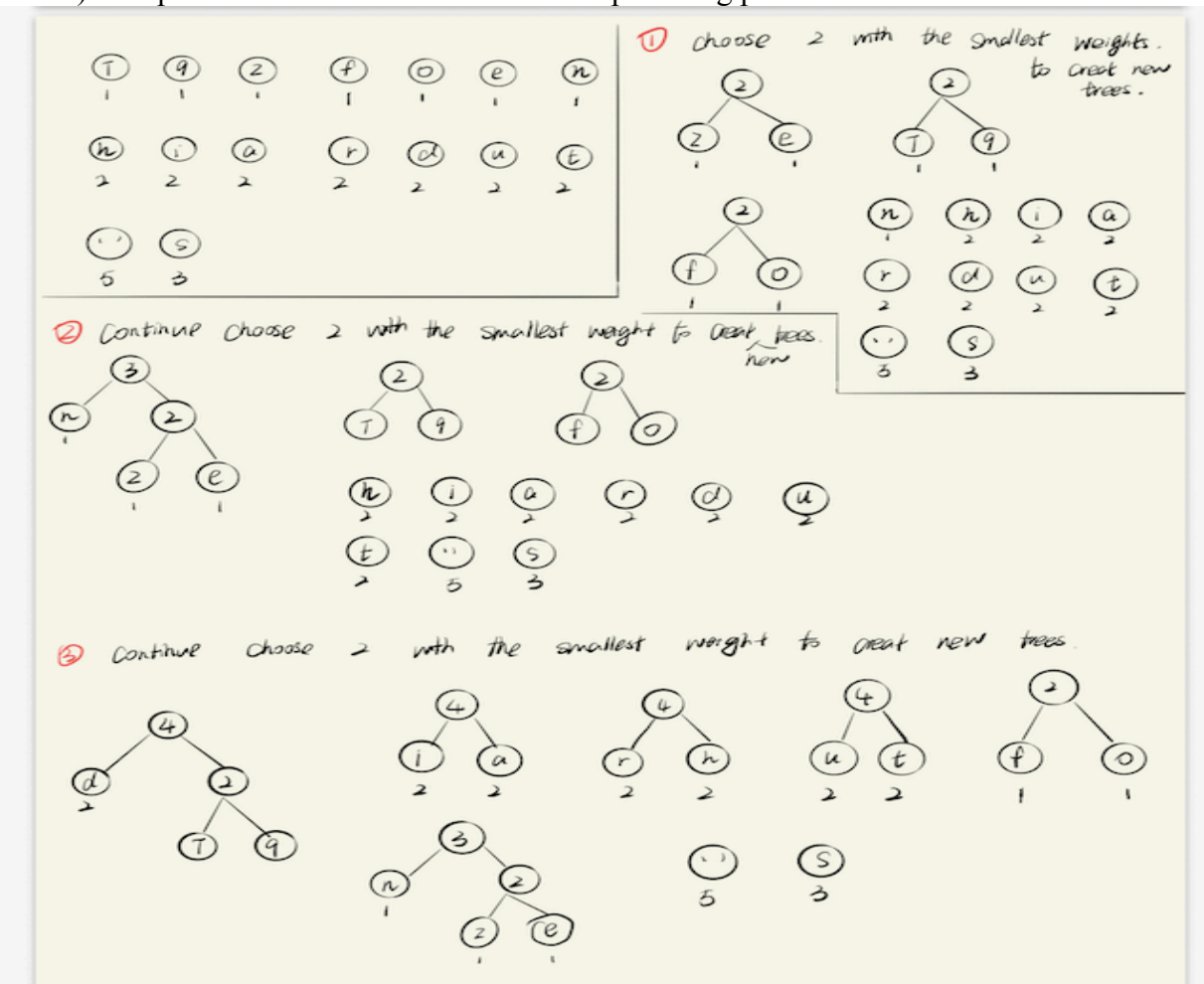
Data Structures and Algorithms
INFO 6205, Wednesday
Homework 4
Due: Friday October 4, 2019

Qian Cai
001389278

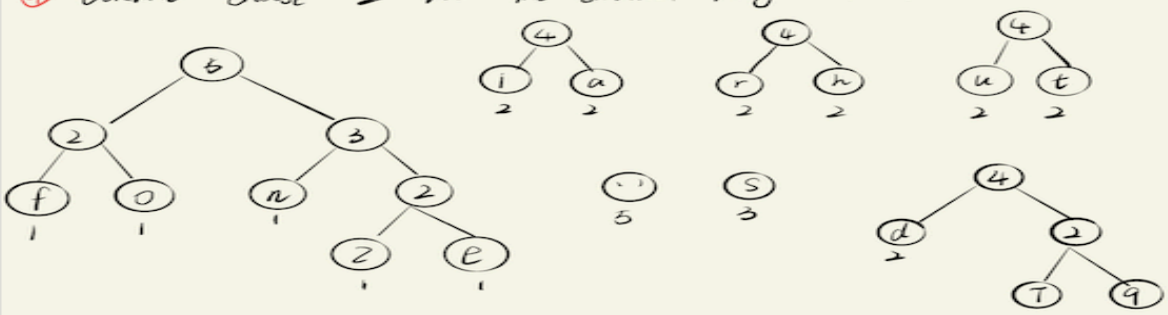
Put all your java, compiled class files and documentation files into a zip file named Homework4.zip and submit it via dropbox to blackboard before the END of due date. Put your name on all .java files. There will be a short quiz on this homework.

1. Consider String "This a hard quiz for students"
    a) Generate a binary Huffman Tree
    b) Show binary data both before and after compression. Analyze difference.
    c) Consider Java code:
        https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/
        Write Pseudo-Code for the Huffman algorithm, SHOW as to why/how the algorithm uses PriorityQueue
    d) Compile Java code and run it with the input string provided above.

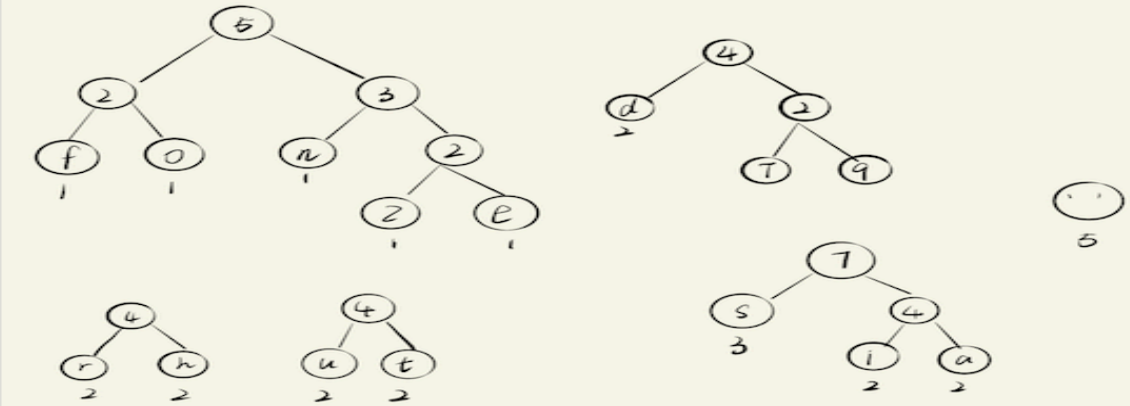
a)



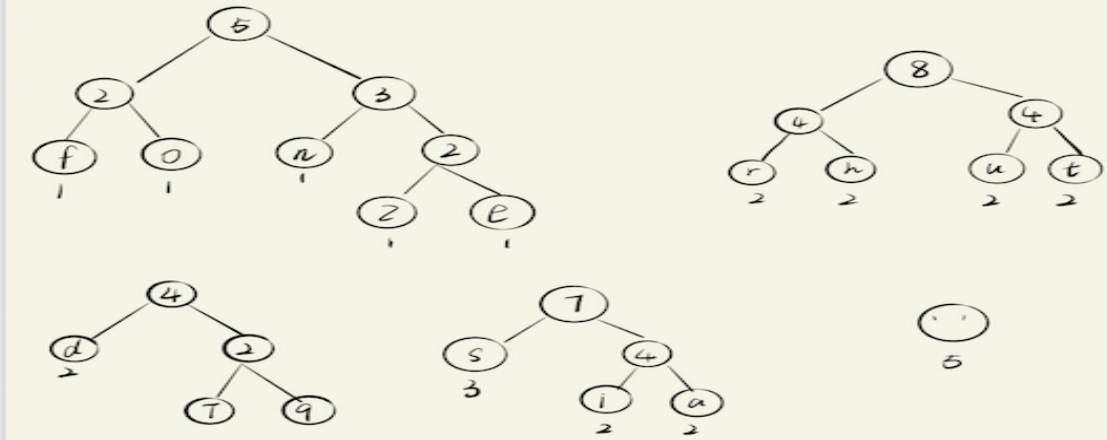
④ Contine choose 2 with the smallest weight to creat new trees



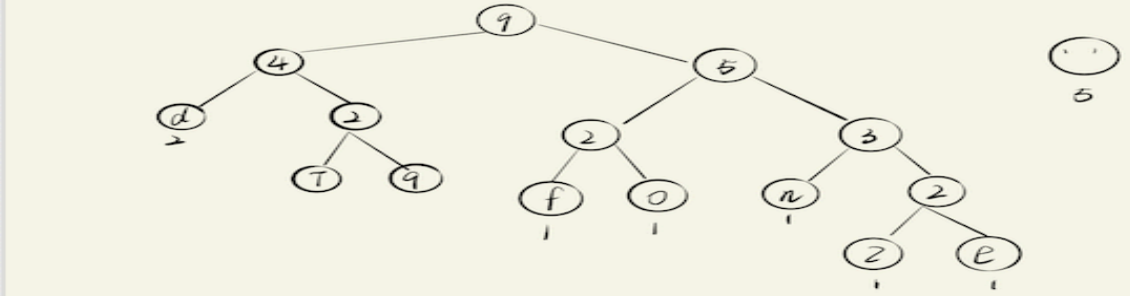
⑤ Contine choose 2 with the smallest weight to creat new trees

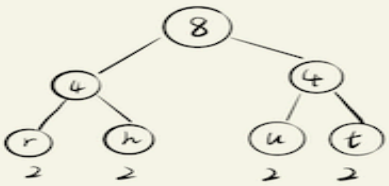


⑥ Contine choose 2 with the smallest weight to creat new trees

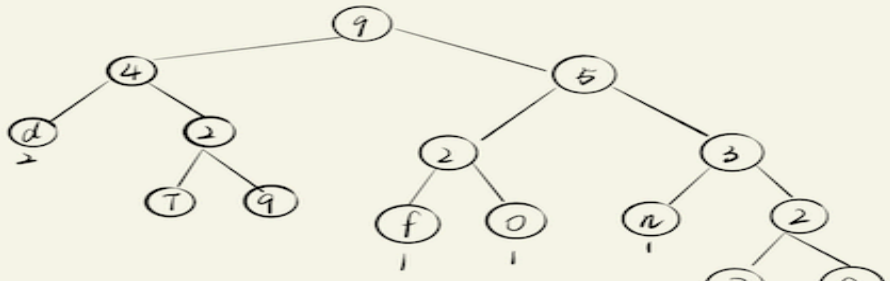


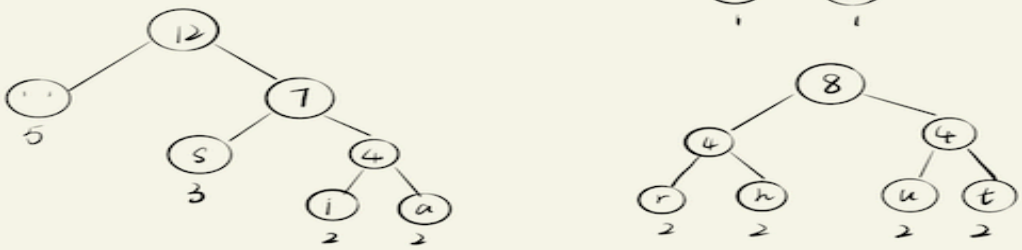
⑦ Contine choose 2 with the smallest weight to creat new trees



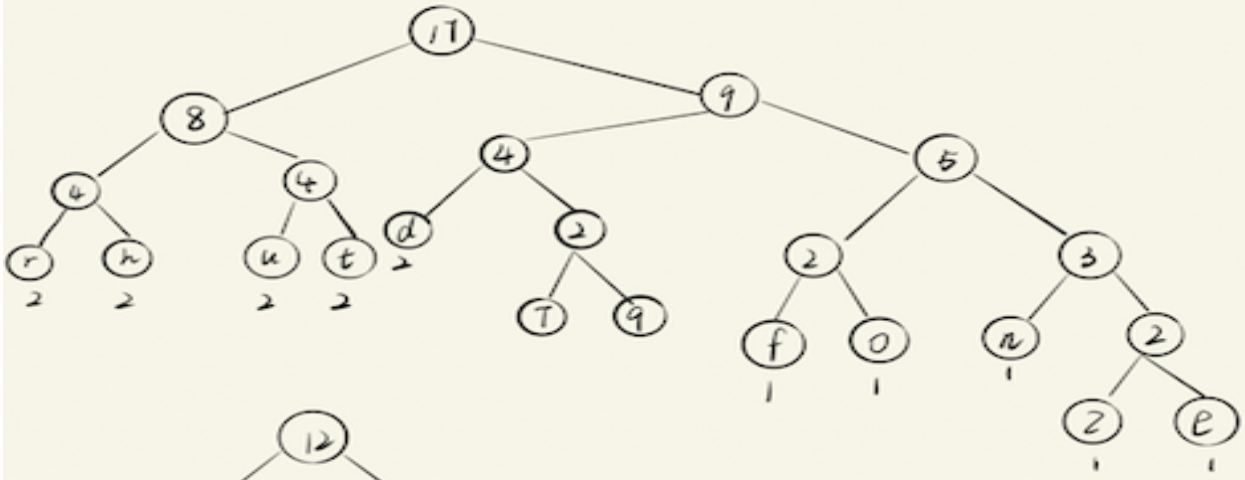


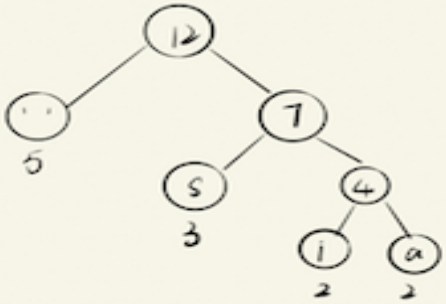
⑧ Continue choose 2 with the smallest weight to creat new trees

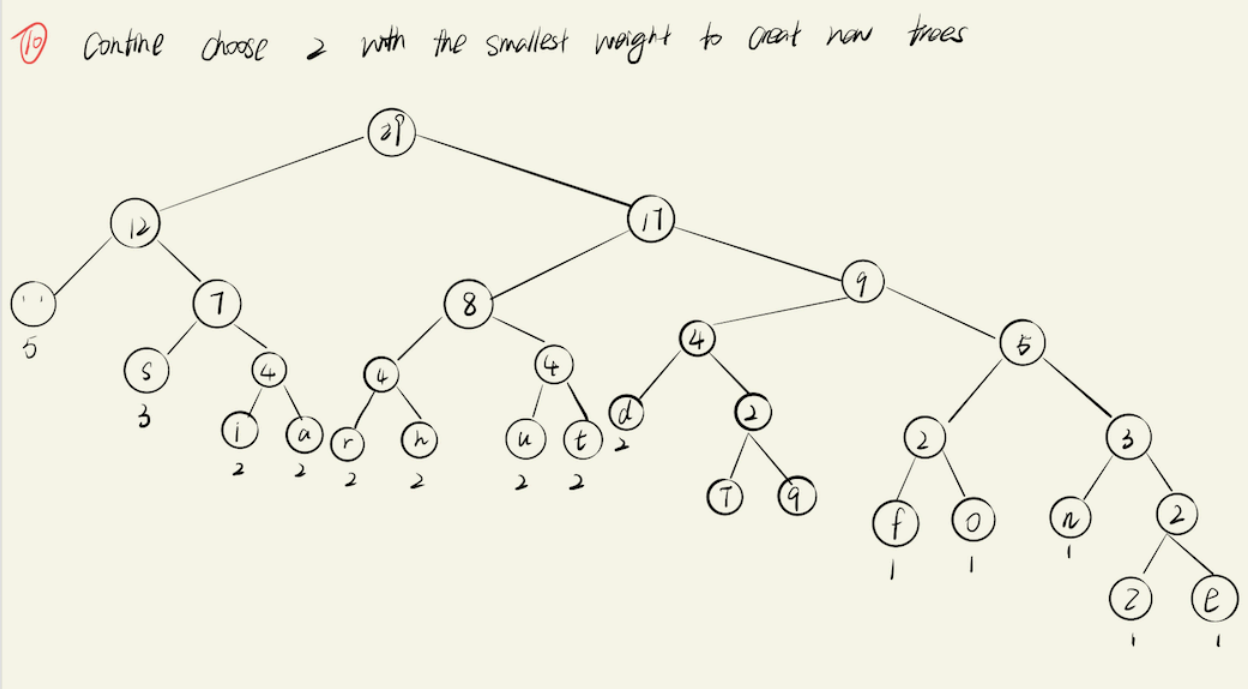




⑨ Continue choose 2 with the smallest weight to creat new trees







b) before compression

T	q	z	f	o	e	n	h
0101 0100	0111 0001	0111 1010	0110 0110	0110 1111	0110 0101	0110 1110	0110 1000
i	a	r	d	u	t	s	space
0110 1001	0110 0001	0111 0010	0110 0100	0111 0101	0111 0100	0111 0011	00100000

after compression:

T	q	z	f	o	e	n	h
11010	11011	111110	11100	11101	111111	11110	1001
i	a	r	d	u	t	s	space
0110	0111	1000	1100	1010	1011	010	00

Difference：ASCII encoding of each character need 8 bits. The 29 character string "This a hard quiz for students" requires 232 bits. After compression we use fewer bits for more frequently occurring characters. Now this string only require 112 bits and more bits can be saved.

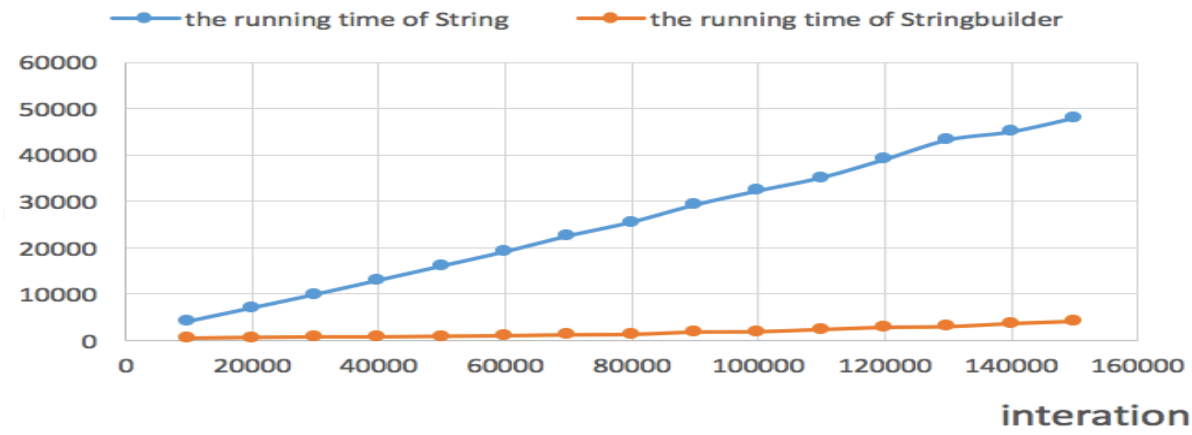
c) Pseudo-Code:

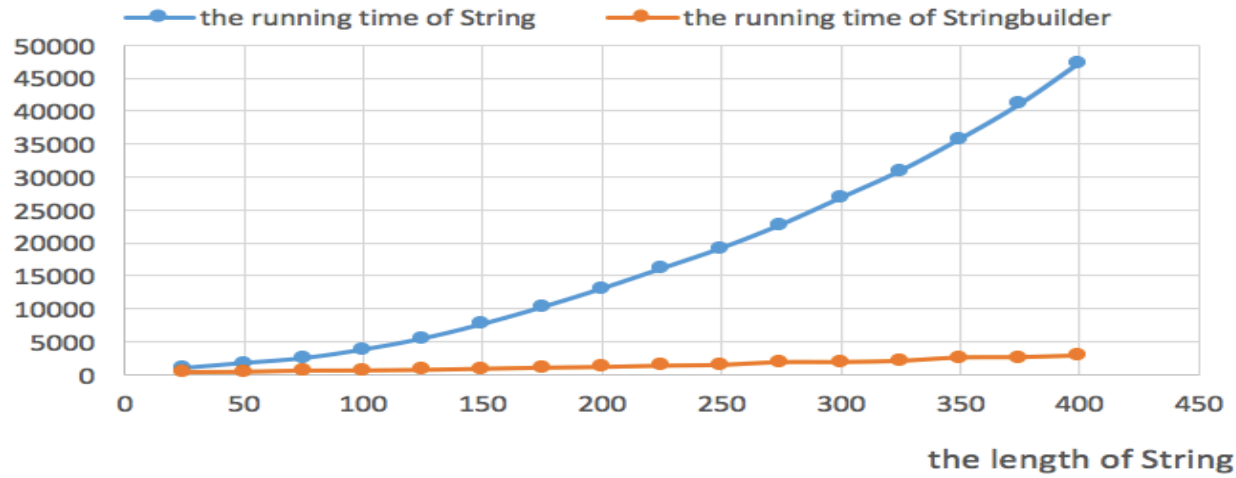
1. Creating a Huffman class with the value of the data, c, left, right of the Huffman node.
2.Creating comparator class to compare the data values of Huffman node.
Return x.data-y.data.
3.Creating a priority queue q and makes a min-priority queue by using comparator class to compare the node.
If return positive number, put y before x.
If return negative number, put x before y.
4.for (int i = 0; i < n; i++)
Creating a Huffman node object and add it to the priority queue.
5.Create a root node
6.While the size of priority queue >1
extract first min and second min of priority queue

Assigning the sum of data of the two nodes to the new node f.
First min->f.left
Second min->f.right
F->root
Add f to the priority queue.
7. print the codes by traversing the tree

2. Write a Java program that generates random text string with a length of 400 bytes for 150,000 iterations. For each iteration, reverse the string using: a) String operations, and b) StringBuilder operation(s). Then c) What is the running time complexity of (a) and (b) after all iterations? d) Present your results in (c) as a graph showing the running times.

c)400 byte->n
150000 iterations->m
The running time complexity of a is O($n^2 * m$).
The running time complexity of b is O(n $* m$).
The running time complexity of a is greater than b, because String is immutable and StringBuilder is mutable.
d)





3. Consider the following code for User class.

A) Discuss code in details

B) Write Java code to test User class with multiple test cases to test quals, hashCode and CompareTo methods.

```
//User class implements the Comparable interface
  public class User implements Comparable<User> {
        private String name;
        private int id;
        private Date birth;

// Parameterized constructor
//Set the value for name, id, birth

        public User (String name, int id, Date birth)
         { this.name = name; this.id = id; this.birth = birth; }

        @Override
//return true or false.
        public boolean equals(Object other) {
// check if the "addresses" of other and this object are the same
                if (this == other) return true;
// check whether the value of other is null.
//tests whether the types of other and this object are identical
                if (other == null || (this.getClass() != other.getClass()))
                        { return false; }
                User guest = (User) other;
//check if the value of guest and this object are the same.
                return(this.id == guest.id) &&
              (this.name = null && name.equals(guest.name)) &&
              (this.birth != null && birth.equals(guest.birth));
        }

        @Override

//Get the hashcode of each instance, return the hashcode number.
                public int hashCode() {
                int result = 0;
                result = 31*result + id;
                result = 31*result + (name !=null ? name.hashCode() : 0);
                result = 31*result + (birth !=null ? dob.hashCode() : 0);
                return result;
        }

        @Override
```

//Compare the id of o and this object. Return negative number, positive number or 0.

```
        public int compareTo(User o) {
                return this.id - o.id; }
        }
}
```

B) Code is in User.java

4. Why Java is Pass-by-Value?

Consider the following two programs,

program-1:

```
public static void main(String[] args) {
   Dog aDog = new Dog("Bella");
   Dog oldDog = aDog;

   changeName(aDog);

   aDog.getName().equals("Bella");       True/False, why?
```

True. Because Java is pass-by-value and the method of equal is compare value of objects, the value of aDog is "Bella" and it was not changed in the function changeName.

```
   aDog.getName().equals("Molly");       True/False, why?
```

False. Because the value of aDogs is "Bella" and it differs from "Molly".

```
   aDog == oldDog;                       True/False, why?
```

True. Because aDog and oldDog are pointing to same object.

```
 }

 public static void changeName(Dog d) {
    d.getName().equals("Bella");          True/False, why?
```

True. Because Java is pass-by-value and the method of equal is compare value of objects, the value of aDog is "Bella"

```
    d = new Dog("Molly");
    d.getName().equals("Molly");
 }
```

program-2:

```
public static void main(String[] args) {
   Dog aDog = new Dog("Bella");
   Dog oldDog = aDog;

    changeName(aDog);
```

aDog.getName().equals("Molly"); True/False, why?

True. Because when changeName function run, the value of aDog has been changed to "Molly".

aDog == oldDog; True/False, why?

True. Even thought the value of aDog has been changed, aDog and oldDog are pointing to same object.

}

public static void changeName(Dog d) {

d.getName().equals("Bella"); True/False, why?

True. Because Java is pass-by-value and the method of equal is compare value of objects, the value of aDog is "Bella".

d.setName("Molly"); }

While calling a method, parameters passed to the called method will be clones of original parameters. Any modification done in called method will have no effect on the original parameters in called method. So Java is Pass-by-Value.