Emma Vu
CSC 151
5/1/2020

MIDTERM

1.Circle rounder = new Circle(myCircle.getX(), myCircle.getY());
rounder.resize(5);
2. Yes, we can use the default constructor. The default constructor will have radius 10 but we have to use the resize method to change the radius to 5 and set coordinates to the same center location with setLocation()
3.

```
private final int MAX_RADIUS = 20;
private boolean inflateTires(int moreRadius){
        boolean isInflate = false;
        int newFront = frontTire.getRadius() + moreRadius;
        int newBack = backTire.getRadius() + moreRadius;
        if(moreRadius > 0){
                if(newFront <= MAX_RADIUS && newBack <= MAX_RADIUS){
                        frontTire.resize(newFront);
                        backTire.resize(newBack);
                        isInflate = true;
                }
        return isInflate;
}
```

4. No, there shouldn't be a public getter method. Because frontTire and backTire refer to specific objects, the values returned from getter will allow the user to directly access those objects. Hence, code outside the class can directly change the instance variables
5.

a) This constructor violates the goal of robustness since it doesn't check the validity of the radius of those Circle objects. If the user puts the parameters Circle front and Circle back which have the radius > MAX_RADIUS (20). In that case, we have a motorcycle with tires that have radius > MAX_RADIUS, that means messing with the given boundary and robustness since we want tires' radius <= MAX_RADIUS.
b) To fix this, we need to add an if statement to check whether the radius of the parameters is valid. If it is larger than MAX_RADIUS then resize the radius to MAX_RADIUS. This can be a helper method:

```
private void validifyRadius(Circle front, Circle back){
        if(front.getRadius() > MAX_RADIUS){
                front.resize(MAX_RADIUS);
        }
        if(back.getRadius() > MAX_RADIUS){
                back.resize(MAX_RADIUS);
```

```
        }


    public Motorcycle(Circle front, Circle back){
        validifyRadius(front, back);
        frontTire = front;
        backTire = back;
```

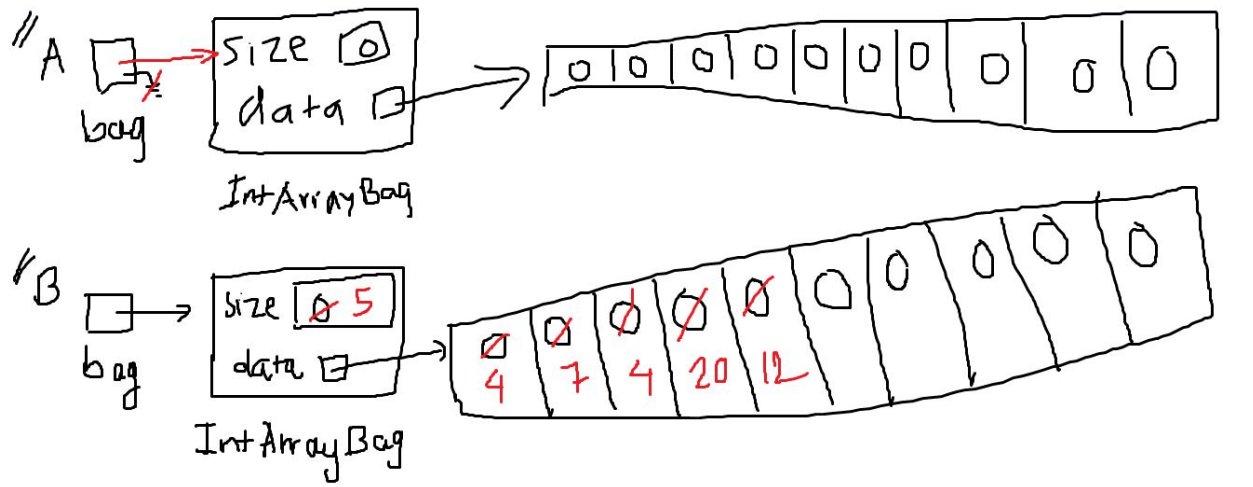
6.

```
public Motorcycle(Circle front, Circle back){
    validifyRadius(front, max);
    frontX = front.getX();
`   frontY = front.getY();
    frontRadius = front.getRadius();
    backX = back.getX();
    backY = back.getY();
    backRadius = back.getRadius();
}
```

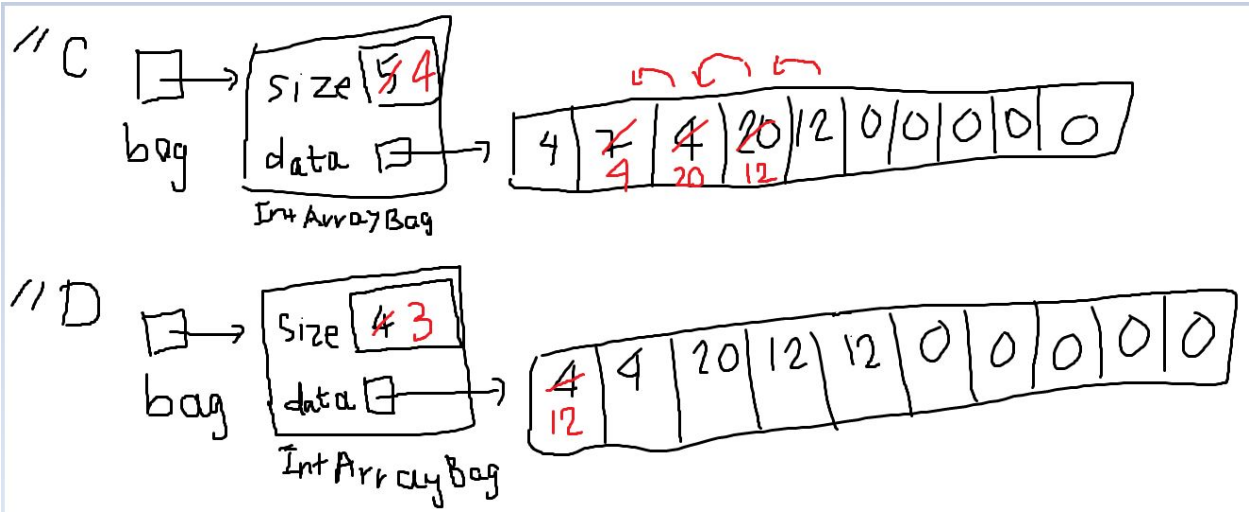
7. No, it would be a bad design because once the class is available to use, the javadocs will say the constructor public Motorcycle(Circle front, Circle back) can be called. Changing the signature will cause the code that uses the constructor crashes

8.

a) The primitive for integer in an array is 0. Value 0, in this case, means we do not care about the value it stores, it's just a way to initialize.



b)



c) The remove methods still preserve the class invariant 1: No order in a bag. In remove2(4), the value 4 appears twice, once in index 0, the other in index 1 but the find() returns the first occurrence of the value 4, which proves that the order doesn't matter here.

The remove methods still preserve the class invariant 2: the number of the elements is still less than data.length. This is true in both remove methods because the size after remove1 is 4 and the size after remove2 is 3, which is still less than data.length, which is 10.

The remove methods still preserve the class invariant 3: For an empty bag, we do not care what is stored in any the data array; for a nonempty bag, the elements in the bag are stored in data[0] through data[size-1], and we don't care what's in the rest of data. After using both the remove methods, the rest is still empty so we don't care what they are. Putting 0 is just a way to initialize the primitive type in the array.

9.

a) Array:
   - Accessing the ith element: O(1)
   - Inserting new element at the start of the list: O(n)
   - Searching for the first occurrence: O(n)

   LinkedList:
   - Accessing the ith element: O(n)
   - Inserting new element at the start of the list: O(1)
   - Searching for the first occurrence: O(n)

In this case, n is the length of the array/LinkedList

b) Both remove methods will have the complexity of O(n), where n is the size of the bag
For remove1, in the helper method find(), there is a for loop to find the index of the first occurrence of a certain value. Hence, the helper find() takes O(n) times. The for loop in remove1 where the previous values will move 1 place to the left takes O(n) times. The worst-case in that for loop is when found_position = 0 and it takes n-1 iterations. Inside

the for loop the assignment operation takes O(1) times. The line where size decrements by 1 takes O(1) times. Therefore, remove1's complexity in terms of big O is O(n) + O(n) + O(1) = O(n) since we only care about the highest power and constants don't matter.

For remove2, the helper method find() is also used which takes O(n) times since in the method find(), there is a for loop to find the index of a certain value. The if statement in remove2 has the code block in the body which both take O(1) times since it is a simple assignment data[found_position] = data[size-1] and size decrements by 1. In total, remove2's complexity is O(n) + O(1) = O(n) since we only care about the highest power.

However, between remove1 and remove2, in actual runtime, remove1 is slower since the for loop that assigns value to new positions takes longer time, which is O(n) times, than the if statement in remove2, which takes O(1) times.

I affirm that I have carried out the attached academic endeavors with full academic honesty, in accordance with the Union College Honor Code and the course syllabus (Signed, Diep Vu)