ArrayLists CarRepair FRQ

SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

This question involves the scheduling of car repairs. The classes used in the question are used to record information about car repairs. The methods shown and the methods to be written involve the mechanic performing a repair and the bay in which the repair takes place. A bay is an area of a repair shop in which a car is parked while a mechanic performs a repair. Mechanics and bays are identified by sequential integer identification numbers that start with 0.

An individual car repair is represented by the following `CarRepair` class.

```
public class CarRepair
{
   private int mechanicNum;
   private int bayNum;

   public CarRepair(int m, int b)
   {
        mechanicNum = m;
        bayNum = b;
   }

   public int getMechanicNum()
   { return mechanicNum; }

   public int getBayNum()
   { return bayNum; }

   /* There may be other instance variables, constructors, and methods
   not shown. */
}
```

The following *RepairSchedule* class represents the use of bays by mechanics repairing cars. You will write two methods of the *RepairSchedule* class.

```
public class RepairSchedule
{
   /** Each element represents a repair by an individual mechanic in a
   bay. */
   private ArrayList<CarRepair> schedule;

   // Number of mechanics available in this schedule.
   private int numberOfMechanics;

   /** Constructs a RepairSchedule object.
    *  Precondition: n >= 0
    */
   public RepairSchedule(int n)
   {
        schedule = new ArrayList<CarRepair>();
        numberOfMechanics = n;
   }

   /** Attempts to schedule a repair by a given mechanic in a given bay
    *  as described in part (a).
    *  Precondition: 0 <= m < numberOfMechanics and b >= 0
    */
   public boolean addRepair(int m, int b)
   {
        /* to be implemented in part (a) */
   }

   /** Returns an ArrayList containing the mechanic identifiers of all
    *  available mechanics,
    *  as described in part (b).
    */
   public ArrayList<Integer> availableMechanics()
   {
        /* to be implemented in part (b) */
   }

   // Removes an element from schedule when a repair is complete.
   public void carOut(int b)
   {
        /* implementation not shown */
   }
}
```

(a) Write the `addRepair` method. The method attempts to schedule a repair by the mechanic with identifier `m` in the bay with identifier `b`. The repair can be scheduled if mechanic `m` and bay `b` are both available. A mechanic is available if the given mechanic number does not appear in an element of schedule and a bay is available if the given bay number does not appear in an element of schedule.

If the mechanic and bay are both available, the `addRepair` method adds the repair to schedule and returns `true`. If either the mechanic or the bay are not available, the `addRepair` method returns `false`.

The following sequence of statements provides examples of the behavior of the `addRepair` method.

The statement `RepairSchedule r = new RepairSchedule(6);` constructs a new `RepairSchedule` object `r`. No repairs have been scheduled. Mechanics are numbered `0` through `5`. The call `r.addRepair(3, 4)` returns `true` because neither mechanic `3` nor bay `4` are present in schedule. The contents of schedule after the call are as follows.

mechanicNum	3
bayNum	4

The call `r.addRepair(0, 1)` returns `true` because neither mechanic `0` nor bay `1` are present in schedule. The contents of schedule after the call are as follows.

mechanicNum	3	0
bayNum	4	1

The call `r.addRepair(2, 4)` returns `false` because bay `4` is present in schedule. The contents of schedule after the call are as follows.

mechanicNum	3	0
bayNum	4	1

The call `r.carOut(4)` removes the repair in bay `4` from schedule. The `carOut()` method is shown here to illustrate that bays and mechanics become available when car repairs are complete. You do not need to write or call this method. The contents of schedule after the call are as follows.

mechanicNum	0
bayNum	1

The call `r.addRepair(1, 4)` returns true because neither mechanic `1` nor bay `4` are present in schedule. The contents of schedule after the call are as follows.

mechanicNum	0	1
bayNum	1	4

Complete the *addRepair* method.

```
/** Attempts to schedule a repair by a given mechanic in a given bay as
described in part (a).
 * Precondition: 0 <= m < numberOfMechanics and b >= 0
 */
public boolean addRepair(int m, int b)
```

(b) Write the *availableMechanics()* method, which returns an *ArrayList* containing the mechanic numbers of all available mechanics. If there is no available mechanic, an empty list is returned. A mechanic is available if the mechanic's identifier does not appear in schedule. Suppose schedule has the following contents.

mechanicNum	*0*	*1*
bayNum	*1*	*4*

For these contents of schedule, *availableMechanics()* should return an *ArrayList* containing the values *2*, *3*, *4*, and *5 (in any order)*.

Complete the *availableMechanics()* method. Assume that *addRepair()* works as specified, regardless of what you wrote in part *(a)*.

```
/** Returns an ArrayList containing the mechanic identifiers of all
 *  available mechanics,
 *  as described in part (b).
 */
public ArrayList<Integer> availableMechanics()
```