```
/**
 * This class answers the Part 2 problems for Practice Exam A.
 *
 * @author Mr. Merlis
 * @version January 2007
 */
public class PracA1P2
{
    //====== QUESTION 1 ------------------------

    // part a
    public boolean isFull()
    {
        for(int i = 0; i < levels.length; i++)
        {
            if(levels[i].findEmptySpace() > -1) // there is a space on this level
                return false;
        }
        return true; // no levels had any open spaces, the lot must be full
    }//========================

    // part b
    public boolean parkCar(Car v)
    {
        if(!isFull() && !isCarAlreadyParked(v.getID())) // garage is NOT full and this car is NOT in the garage
        {
            // find the next available spot
            int garageLevel = 0; // no need to declare outside of IF b/c it may not be needed
            while(levels[garageLevel].findEmptySpace() == -1) // find the first level that has an open spot
            {
                garageLevel++;
            }
            // at this point, we know the level where there is an open space
            spotToParkCarIn = levels[garageLevel].findEmptySpace();
            levels[garageLevel].setCar(SpotToParkCarOn, v);
            totalCars++;
            return true;
        }
        return false;   // garage is full or this car is already parked in the garage
    }//===========================

    // part c
    public Car removeCar(String id)
    {
        Car carToReturn = null;
        if(isCarAlreadyParked(id)) // this car is in the lot
        {
            // find the car
            for(int i = 0; i < levels.length; i++)   // go through each level of the parking garage
            {
                for(int j = 0; j < levels[i].numSpaces(); j++) // search each space on the ith level for this car
                {
                    if(levels[i].getCar(j) != null && ((Car)levels[i].getCar(j)).getID().equals(id)) // found the
spot this car is in
                    {
                        carToReturn = levels[i].getCar(j);
                        levels[i].setCar(j, null);
                        totalCars--;
                        return carToReturn; // exits the method
                    }
                }
            }// end the for loop
        }
        return carToReturn; // no car with the specified id, this will return null
    }//============================

    //=============================================================================

    //========== QUESTION 2-------------------

    // part a
    public static String findFirstTag(String text)
    {
        int locOfTagStart =  text.indexOf("<");
        if(locOfTagStart == -1) // "<" does not exist in this string
            return null;
```

```
        int locOfTagEnd = text.indexOf(">", locOfTagStart+1); // search for the ">" after the "<"
        if(locOfTagEnd == -1) // ">" does not exist or comes before the first "<"
            return null;

        // if this point is reached, we know that '<' and '>' exist, with '<' coming before '>'
        return text.substring(locOfTagStart, locOfTagEnd+1); // the +1 is to include the '>'
    }//============================================

    // part b
    public static String remove(String text, String str)
    {
        int startOfStrToRemove = text.indexOf(str);
        if(startOfStrToRemove > -1) // str exists in text
        {

            String toReturn = text.substring(0, startOfStrToRemove); // get everything up to that point
            toReturn += text.substring(startOfStrToRemove + str.length());
            return toReturn; // with str removed
        }
        return text;    // unchanged
    }//============================================

    // part c
    public static String removeAllTags(String text)
    {
        // go through entire text
        String startTag = findFirstTag(text);
        String endTag = "";
        while(startTag != null) // a tag exists
        {
            endTag = "</" + startTag.substring(1); // put the "</" in front of the tag contents
            if(text.indexOf(endTag) < text.indexOf(startTag)) // either no end tag or it comes before the opening
tag
                throw new IllegalArgumentException();
            text = remove(text, startTag);
            text = remove(text, endTag);

            startTag = findFirstTag(text); // note no need to do a shift because the previous tag was removed
from text
        }
        return text;
    }//============================================

    //=========================================================================

    //========== QUESTION 3 -------------------

    // part a
    public CoralReefFish(Environment env, Location loc)
    {
        super(env, loc, env.randomDirection(), Color.yellow);
        placeOfBirth = loc;
    }//============================================

    // part b
    protected double distanceFromHome(Location loc)
    {
        int  x = loc.col() - placeOfBirth.col(); // change in column (horizontal)
        int y = loc.row() - placeOfBirth.row(); // change in row (vertical)
        return Math.sqrt(x*x + y*y);
    }//====================================

    // part c
    protected Location nextLocation()
    {
        ArrayList emptyNbrs = emptyNeighbors(); // same as a regular fish
        // note that we are NOT removing the direction behind the fish

        // remove locations that are too far away
        int index = 0;
        while(index < emptyNbrs.size())
        {
            if(distanceFromHome((Location)emptyNbrs.get(index)) > 4)
                emptyNbrs.remove(index);
            else
```

```
                index++; // if we DO remove a loc, we DON'T want to update index
                    // because we would skip over a possible location
            }

            if(emptyNbrs.size() == 0)
                return location(); // nowhere to go

            // rest is the same as nextLocation() in fish
            Random randNumGen = RandNumGenerator.getInstance();
            int randNum = randNumGen.nextInt(emptyNbrs.size());
            return (Location) emptyNbrs.get(randNum);
        }//==================================

        //============================================================================

        //========== QUESTION 4 -------------------

        // part a
        public class APStudent
        {
            private String studentName;
            private ArrayList exams;

            public APStudent(String name) { ... )
            public String getName() { ... ) // accessor
            public ArrayList getExams() { ... ) // accessor
            public void add(APExam exam) { ... )
            public double getAverageGrade() { ... )
        }

        // part b
        public int getAwardLevel(APStudent student)
        {
            ArrayList exams = student.getExams();
            int numExams = exams.size();
            double years = 0;
            int award = 0;

            for(int i = 0; i < numExams; i++)
            {
                APExam exam = (APExam) exams.get(i);
                if(exam.getGrade() >= 3)
                    years += .5 * exam.getLevel(); // remember that full year are worth 2, half worth 1
            }

            if(years >= 4 && student.getAverageGrade() >= 3.25)
                award = 2;
            else if(years >= 3.0)
                award = 1;

            return award;
        }//=========================================

        // part c
        public double[] getStats(ArrayList list)
        {
            int[] typeOfAward = new int[3];
            int award;

            for(int i = 0; i < list.size(); i++)
            {
                award = getAwardLevel((APStudent)list.get(i));
                typeOfAward[award]++;   // typeOfAward functions as a counter for the 3 types
            }

            double[] percents = new double[3];
            for(award = 0; award < 3; award++)
                percents[award] = 100.0 * typeOfAward[award]/list.size();
                // note no cast above because Order Of Ops will do the 100.0 x tOfAw. first

            return percents;
        }//======================================
    }
```