

**CIS 110 — Introduction to Computer Programming  
Fall 2018 — Final Midterm**

Name: \_\_\_\_\_

Recitation # (e.g., 201): \_\_\_\_\_

Pennkey (e.g., paulmcb): \_\_\_\_\_

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

\_\_\_\_\_  
Signature

\_\_\_\_\_  
Date

**Instructions:**

- When you turn in your exam, you may be required to show ID. **If you forgot to bring your ID, talk to an exam proctor immediately.**
- **Do not open this exam until told by the proctor.** You will have exactly 120 minutes to finish it.
- **Make sure your phone is turned OFF (not to vibrate!) before the exam starts.**
- Food, gum, and drink are strictly forbidden.
- **You may not use your phone or open your bag for any reason**, including to retrieve or put away pens or pencils, until you have left the exam room.
- This exam is *closed-book, closed-notes, and closed-computational devices*.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- All code must be written out in proper Java format, including all curly braces and semicolons.
- **Do not separate the pages.** You may tear off the one scratch page at the end of the exam. This scratch paper must be turned in or you lose 3 points.
- Turn in all scratch paper to your exam. Do not take any sheets of paper with you.
- If you require extra scratch paper, please use the backs of the exam pages. Also, extra pages are provided at the end of the exam. **Only answers on the FRONT of pages will be grading. The back is for scratch work only.**
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a proctor will come to answer them.
- We wish you the best of luck.

**Scores:** [For instructor use only]

Question 0		1 pt
Question 1		12 pts
Question 2		12 pts
Question 3		10 pts
Question 4		16 pts
Question 5		9 pts
Question 6		10 pts
Question 7		17 pts
Question 8		23 pts
<b>Total:</b>		<b>110 pts</b>

**0) (1 point) The Easy One:**

- Check that your exam has all **17** pages (excluding the cover sheet).
- Write your name, recitation number, and Pennkey (username) on the front of the exam.
- Sign the certification that you comply with the Penn Academic Integrity Code.

**1.) MISC (12 pts total)**

**1.1) Addition**

**(4 points)** Fill in the blank below for an unnecessarily complicated function that finds the value of  $x + y$  using addition (i.e., repeatedly adding 1  $y$  times). You can assume  $x$  and  $y$  are non-negative integers.

```
public static int addition(int x, int y) {
    if (y == _____) { //base case //1 point
        return _____; //1 point
    } else { // recursive step
        return _____;
        //2 points
    }
}
```

**1.2) List (8 points – 1 point each)**

For each, say if this applies to Java's built-in ArrayList, LinkedList, Neither, or Both. Write one of the underlined choices in each box.

Elements of the List are stored sequentially in memory	
Adding elements at the end of the List is fast	
Removing elements from the front of the List is fast	
Can contain multiple data types within the list	
Can be sorted using Comparator objects and the Collections.sort() method.	
Can access and manipulate node objects in the underlying data of the List data structure	
Has fast access time for elements in the middle of the List (that is, the time it takes to access the elements is not related to the size of the list)	
Must be imported using java.util. Example: "import java.util.ArrayList;" and "import java.util.LinkedList;"	

**2) RECURSION TRACING (12 points total)**

Below is a mystery recursion function. Do not try to work out WHAT it's doing, as the functionality is completely made up. It does nothing useful.

```
public int mystery(int x, int y) {  
    System.out.println(x + ", " + y);  
    if (x % 2 == 1 && y % 2 == 1) {  
        return Math.min(x, y);  
    } else if (x % 2 == 0 || y + x < 20) {  
        return 1 + mystery(x + 1, y + 2);  
    } else {  
        return 2 * mystery(x / 2, y / 2);  
    }  
}
```

In the boxes below, write whatever prints when the function is called with the given arguments in order. At the bottom of each box, say what the function call ultimately returns.

a) `mystery(21,18)`

Returns:

b) `mystery(15,38)`

Returns:

c) `mystery(13,24)`

Returns:

**3) USING OBJECTS (10 points total)**

A recent outbreak of E. coli infections linked to romaine lettuce has forced many restaurants and stores to stop offering romaine lettuce. The CDC has decided to enlist your help in designing a Java Class called **RomaineLettuce** that would aid in the process of discovering contaminated lettuce producers. This class will track every instance of **RomaineLettuce** created by adding that instance to a list. This is so that the CDC can test all the lettuces in the market and announce which producers and lettuce types are contaminated. Each source of lettuce is identified by a producer and id. The CDC provided you with a function called **testSample** that can be used to determine if a sample is contaminated.

```
public class RomaineLettuce {

    private String producer; //The farm that produced this head of lettuce
    private int id; //A unique ID for this head of lettuce
    private int ecoli_cpg; //The number of ecoli cells per gram of this head
    /**
     * The outbreak variable is True for all lettuce objects if
     * ANY one created lettuce is over 110 E. Coli cells per gram.
     */
    private boolean outbreak = false;

    private List<RomaineLettuce> lettuces = new ArrayList<RomaineLettuce>();
    /**
     * Constructor
     */
    public RomaineLettuce(String producer, int id) {
        // TODO: part A

        _____ = producer;

        _____ = id;
        this.ecoli_cpg = testSample();
        if(this.ecoli_cpg > 110) {
            outbreak = true;
        }

        lettuces._____;
    }
    /**
     * Standard toString() method
     */
    public String toString() {
        return producer + ": " + id;
    }
    /**
     * Getter for Producer name
     */
    public String getProducer() {
        return producer;
    }
}
```

```
/**
 * Prints out each contaminated lettuce, separated by a newline. Does not
 * re-test the samples.
 */
public void announceResults() {
    if (!outbreak) {
        System.out.println("No outbreak, everything is fine!");
        return;
    }
    // TODO: part D
}

/**
 * Returns a number to represent the presence of E. coli cells per gram.
 * if this number is over 110, it is likely the sample is unsafe to eat.
 */
private int testSample() {
    //You can assume the CDC has written this method correctly
}
}
```

- a) On the previous page, fill in the blank in the constructor that correctly assigns the `producer` and `id` fields of the object and adds the newly created instance to `lettuces`. **(3 points)**
- b) In the space below, list the VARIABLES (not methods) that should be static for the code to work. **(2 points)**

- c) In the space below, list the METHODS (not variables) that should be static for the code to work. **(2 points)**

- d) In the space below, write the rest of the function `announceResults()`. This function should print out the contaminated lettuces, separated by a newline. The lines you add should be no longer than 5 lines of code. If it is longer than 5 lines (**not counting closing brackets and not counting the lines we gave you**), it will NOT be graded, and you will get zero points. **(3 points)**

**4) Object Tracing (16 points)**

Dr. McBurney wrote some code by making random noises with his mouth and adding Java syntax. Write the values of the variables at each point in the table below. **(1 point per blank)**

```
public class Blarg {
    public String sproing;
    public int pang;
    public Blarg(String s, int i) {
        sproing = s;
        pang = i;
    }
    public void sprang(int pang) {
        pang = this.pang;
    }
    public void poing(String sproing) {
        this.sproing = sproing;
    }
}

public class BlargDemo {
    public static Blarg temp;
    public static Blarg woop(Blarg neep, int spam) {
        Blarg steve = new Blarg(neep.sproing, neep.pang);
        neep.sprang(spam);
        neep.poing("Steve");
        //Point 2
        temp = steve;
        temp.pang++;
        temp.sproing.toUpperCase();
        //Point 3
        return steve;
    }

    public static void main(String[] args) {
        Blarg neep = new Blarg("Ping", 5);
        Blarg steve = neep;
        neep = new Blarg("Bang", -3);
        //Point 1
        neep = woop(steve, 14);
        steve.sproing = temp.sproing;
        //Point 4
    }
}
```

Point	neep.sproing	neep.pang	steve.sproing	steve.pang
1				
2				
3				
4				

**5) Comparator vs. Comparable (9 points)**

a) Between a Comparator and a Comparable, which of the two must exist as its own class, separate from the object being sorted? (1 point)

b) The Dean of Admissions at McBurney University (go Fightin' Baldies) thinks that students who do well with regards to high school GPA also do well on the SAT exam. In order to test their hypothesis, they choose to sort applications received this year in two different ways: SAT scores and high school GPA. Using your answer to question a) write classes that assist in sorting a list of students (`List<Student>`)

```
public class Student {  
    public double highSchoolGPA;  
    public int satScore;  
}
```

```
public class StudentSATComparer implements _____<Student>{ //3 points
```

```
public class StudentGPAComparer implements _____<Student>{ //3 points
```





a) Write out the full lines of code for insertion sort, not just the line numbers **(5 points)**

```
public static void insertionSort(Pasta[] arr) {
```

b) Write out the full lines of code for selection sort, not just the lines numbers **(5 points)**

```
public static void selectionSort(Pasta[] arr) {
```

**7) Linked List (17 points)**

Joe went to the casino and sat down at the table to play a card game. After a few rounds, he realizes that there is something fishy going on with the dealer. Joe discovers two very interesting sneaky moves:

**Sneaky Move 1:** In the beginning of the game, the dealer would deal a card, and would sneak an identical card into the exact middle of the deck (If there are an odd numbers of the cards in the deck after the card is removed, the dealer inserts it after the smaller half)

**Sneaky Move 2:** Towards the end of the game, the dealers lays out all of the cards on the table, quickly remembers the deck order, and writes on a piece of paper the value of the largest card in the odd numbered indices (that is, every other card starting with card 1, then card 3, then card 5, etc.) The deck begins at index 0, and cards are of positive integer values. Example, if the deck contained, in order, values 2, **5**, 7, **3**, 10, **6**, 8, the dealer will write down 6 (as it is the largest odd indexed card, where the odd indexed cards are bolded). If there are no odd indexed cards, the dealer writes down the number 0.

Fill in the blank for the two functions below within the Dealer class.

```
public class Dealer {
    private class Card {
        public int value;
        public Card next;

        public Card(int value, Card next) { //2 points total
            _____; //1 point
            _____; //1 point
        }
    }

    private Card topOfDeck; //the top card on the deck

    public Dealer() {
        //dealer constructs a deck of cards
        //you don't need to do anything here
    }

    public void sneakyMove1() { //8 points total
        //if there are no cards

        if ( _____ ) { //1 point
            return;
        }
        Card c = this.topOfDeck;
        //continued on the next page
    }
}
```

```

//remove the first card

this.topOfDeck = _____; //1 point
if (topOfDeck != null) {
    int length = 0;

    for ( _____ ) {
        //2 points
        length++;
    }
    int midIdx = _____; //1 point
    Card i = this.topOfDeck;
    for (int counter = 0; counter < midIdx - 1; counter++) {

        i = _____; //1 point
    }

    c.next = _____; //1 point

    i.next = _____; //1 point
} //end of sneakyMove1()

/**
 * returns the number the dealer writes down on the piece of paper
 */
public int sneakyMove2() { // 3 points total

    if ( _____ == null || _____ == null) { //2 points
        return 0;
    }

    return sneakyMove2Helper( _____ ); //1 point
}

private int sneakyMove2Helper(Card c) { //4 points total

    if ( _____ ) { //1 point
        return 0;
    } else if ( _____ ) { //1 point
        return c.value;
    }

    return Math.max ( _____, sneakyMove2Helper( _____ ) );
    //1 point each
}
}

```

**8) Object Writing (23 points)**

Imagine that we have the following interface that represents a student at Penn.

**The interface is copied onto the page 16 of the exam which can be ripped off.  
DO NOT RIP OFF THIS PAGE**

```
public interface PennStudent {
    /**
     * returns the name of the student
     */
    public String getName(); //1 point

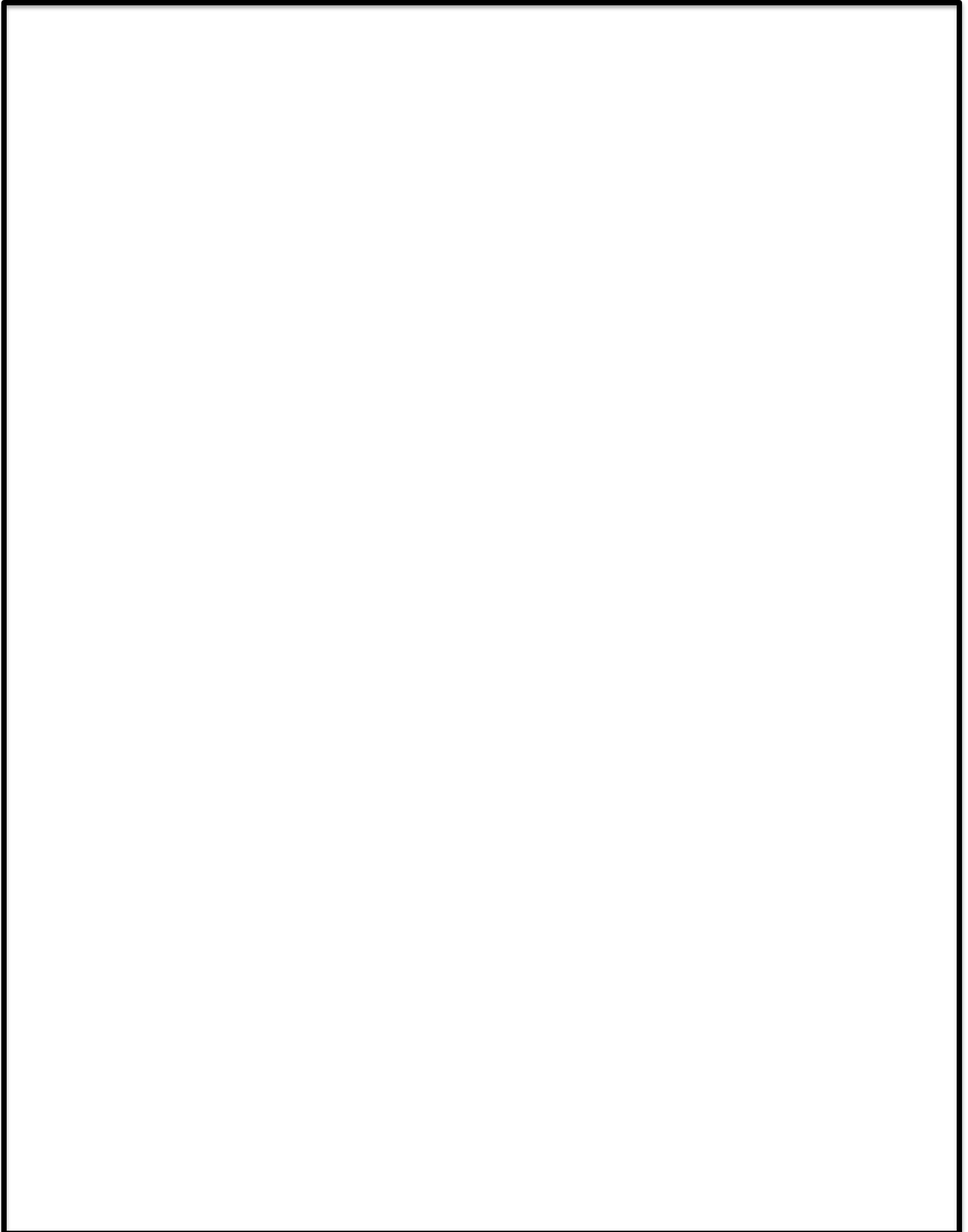
    /**
     * returns the class year of the student. For example, if the
     * student was a senior, then this should return 2019
     */
    public int getClassYear(); //1 point

    /**
     * adds the homework grade to the student's record. These grades
     * are a number from 0-100 (you do not need to error check this)
     */
    public void addAssignmentGrade(int grade); //2 points

    /**
     * adds several homework grades to the student's record. These
     * grades are a number from 0-100 (again, don't error check this)
     */
    public void addAssignmentGrade(int[] grades); //3 points

    /**
     * returns the average of all the grades the student has
     * received in the class. All assignments are out of 100
     * There is not an exam or participation component in this object
     */
    public double getClassGrade(); //3 points
}
```

**a)** In the space provided on the next page, please write the code for a `Cis110Student` class that implements the `PennStudent` interface. Remember to import any libraries you use that require importing. **You may not add additional public methods or fields.** Additionally, add a meaningful constructor. *You do not need to comment your code, we realize this is a timed exam.* **(12 points total)**



b) President Amy Gutmann wants a way to keep track of student statistics for the students in CIS 110. Using your newfound programming skills, you decide to help her by creating a `Cis110Course` class that will keep track of the students enrolled in the course. Your `Cis110Course` class will manage multiple instances of your `Cis110Student` class. **The instances must be stored in an `ArrayList`.** Your class must include a constructor that takes in an array of `Cis110Student` objects as a parameter. You must also implement ALL of the following methods in your code: **(11 points total)**

`ArrayList<String> getStudentNames()` - returns a list containing the names of every student in the course (these do not need to be in any particular order). **(2 points)**

`int countSeniors()` - return the number of seniors (that is, every student in the class of 2019) currently enrolled in the class **(2 points)**

`boolean dropStudent(String name)` - removes student from CIS110 with the given name. If the student exists to be removed, return true. If that student doesn't exist, return false and do nothing. **(3 points)**

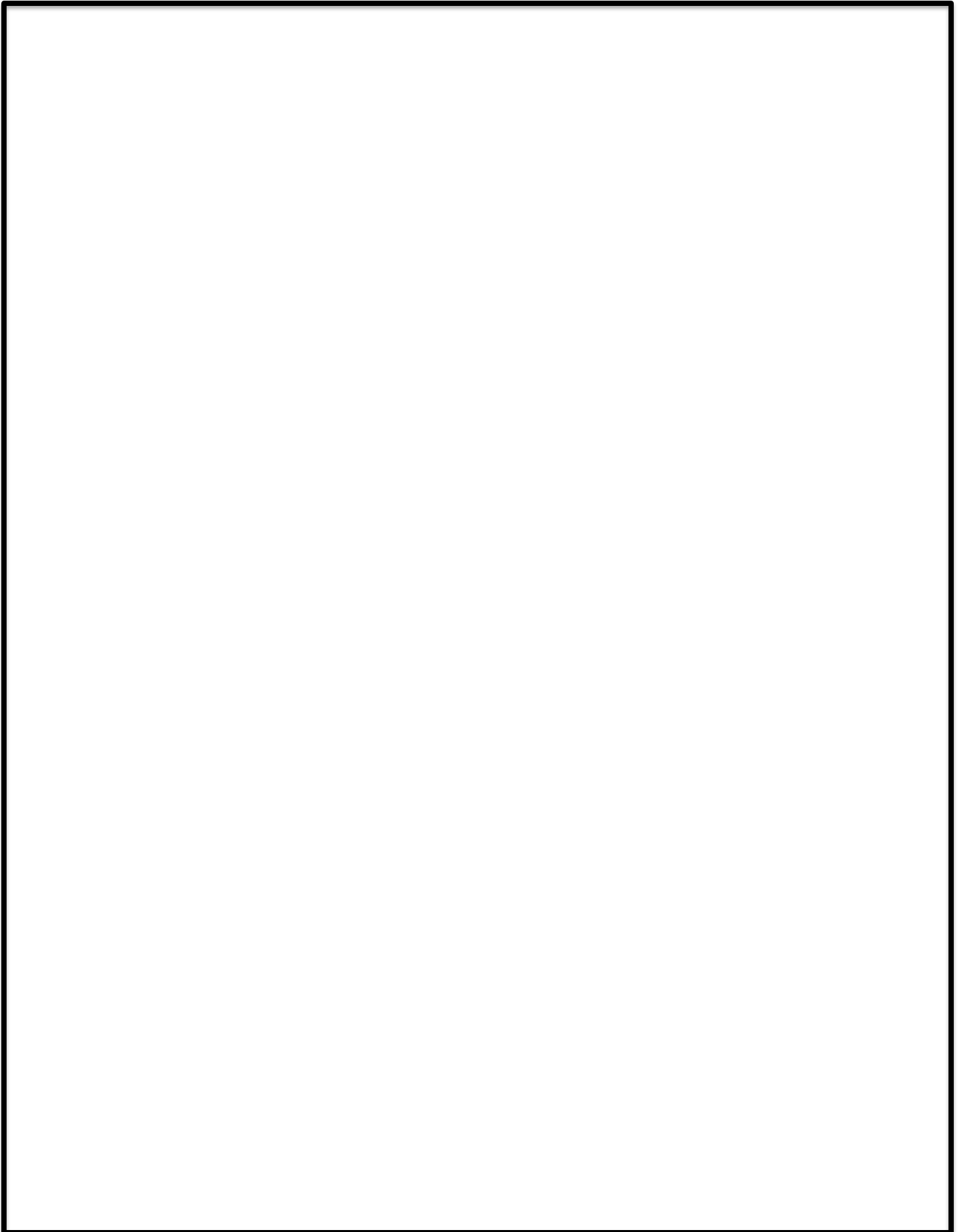
`double getMaxGrade()` - returns the highest class grade out of all of the students in the course **(2 points)**

```
//add import statements here

public class Cis110Course {
    // TODO: add any fields that you find necessary

    public Cis110Course(Cis110Student[] allStudents) { //2 points
        // TODO: implement constructor
        // Note that this takes in an array, but you must have an
        // ArrayList to store the student objects.

        //implement all other methods on the next page
    }
}
```



**EXTRA ANSWER SPACE: You may NOT rip this page off, but work written on the FRONT of this page may be graded.**

**List interface methods you may use for a List<E> where E is the object type the list is storing. We've covered these in class, so it is expected you know what they do. You are allowed to use these on this exam when using the List<E> interface in java.**

```
boolean add(E e);  
void add(int index, E e);  
boolean remove(E e);  
boolean contains(E e);  
int size();  
boolean isEmpty();  
E get(int index);
```



**REFERENCE PAGE: You may rip this page off but nothing on this page will be graded. YOU MUST TURN THIS PAGE IN WITH YOUR EXAM OR YOU LOSE ALL POINTS FOR SORTING QUESTION OR THE INTERFACE QUESTION**

**Copy of lines of code from sorting question**

```

1.     arr[z] = redSauce;
2.     arr[q] = arr[q+1];
3.     arr[q+1] = whiteSauce;
4.     arr[q+1] = arr[q];
5.     arr[noodle] = arr[z];
6.     else{
7.     for (int q = z + 1; q < arr.length; q++) {
8.     for (int q = z - 1; q >= 0; q--) {
9.     for (int z = 0; z < arr.length; z++) {
10.    for (int z = 1; z < arr.length; z++) {
11.    if (arr[q].getLength() < arr[noodle].getLength()) {
12.    if (arr[q].getLength() > arr[noodle].getLength()) {
13.    if (arr[q].getLength() > arr[q+1].getLength()) {
14.    int noodle = z;
15.    q = -1;
16.    noodle = q;
17.    noodle = z;
18.    Pasta whiteSauce = arr[q];
19.    Pasta redSauce = arr[noodle];

```

**Copy of interface**

```

public interface PennStudent {
    /**
     * returns the name of the student
     */
    public String getName(); //1 point

    /**
     * returns the class year of the student. For example, if the
     * student was a senior, then this should return 2019
     */
    public int getClassYear(); //1 point

    /**
     * adds the homework grade to the student's record. These grades
     * are a number from 0-100 (you do not need to error check this)
     */
    public void addAssignmentGrade(int grade); //2 points

    /**
     * adds several homework grades to the student's record. These
     * grades are a number from 0-100 (again, don't error check this)
     */
    public void addAssignmentGrade(int[] grades); //3 points

    /**
     * returns the average of all the grades the student has
     * received in the class. All assignments are out of 100
     * There is not an exam or participation component in this object
     */
    public double getClassGrade(); //3 points
}

```

**SCRATCH PAGE: You may use this as scratch page and rip it off, but nothing on this page will be graded.**