MIDTERM EXAM INSTRUCTIONS

1. Midterm Exam: **25 points w/ 0 E.C. points**
2. Due Date & Time: **05-02-2021 at 11:55 PM**

WHAT TO SUBMIT

1. Take-home Exam Report, 1 PDF

HOW TO SUBMIT AND THE RULES TO FOLLOW

- Submit via iLearn, the Assignment and Exam Submission section
- Please follow the exam instructions
- Please follow the Course Policy on Student Conduct and Academic Honesty

PERFORMANCE TRACKER

ASMT	GRADE	YOUR GRADE
ZOOM	05	
01	20	
02-PREPARATION	25	
02	75	
03	75	
MIDTERM EXAM 01	25	
04	75	
MIDTERM EXAM 02	25	
TOTAL	325	

A: 90-100% **B**: 80-89% **C**: 70-79% **D**: 60-69% **F**: 0-60%
The course grader provides feedback to your assignments on iLearn.

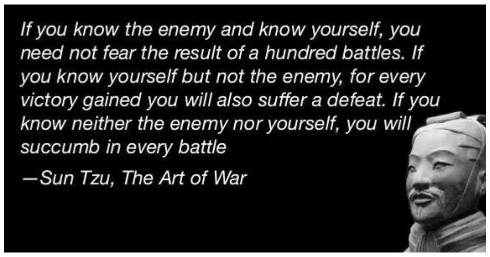
ABOUT

The goal of this take-home exam is for us to **know what we do not know**.

We are taking this exam as seriously as how we take an actual exam in class. Please,

1. Follow all the rules and the guidelines listed at the top of page 1 and page 2
2. Read each question carefully before answering

We will go through the answers to all the exam questions together in class.


If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle
—Sun Tzu, The Art of War

STEP A – Take the Exam, **10 points**

1. Allocate 50 quiet minutes to take the exam on page 2 to the last page.
2. Record the date and time when you start.
3. Stop right at minute 50. Record the date and time when you stop the exam.

STEP B – Correct Your Answers, **10 points**

1. Review the related course materials and write code when necessary to find a correct answer for each question. We should be able to find all the answers using the packages, the in-class discussions, our assignments, and the other course materials.
2. At the end of each of your oringal answers, type in *italic* text and:
   - Give your orginal answer a score.
   - List all the mistakes then explain why, you think, you made the mistakes. Add the correct answer you found. Document how you found the correct answer. Document where you found the materials which support the answer.
   - If you did not make any mistakes, please document how you verified that your answer was accurate. Document how you found the correct answer. Document where you found the materials which support the answer. Outline how you could have done better.
   - Record your total score out of 100 points for all the orginal answers.

STEP C – Reflect and Retake the Exam, **5 points**

1. **Problem Solving**: Reflect if you managed the exam time efficiently and if you strategized your test-taking successfully.
2. Repeat steps A to C again if necessary. Please keep appending new contents as directed in Step B.2.
3. Think if the same topics will be tested again in our final exam, what questions we may get.

It is a good idea to do every step of this assignment thoroughly. We are creating a set of materials which we will use to review for the final exam. And this is also the best way to prepare ourselves to succeed in the second half of the semester. Thank you.

Kullathon “Mos” Sitthisarnwattanachai
921425216

Step A

- Started: 5/2 22:10 PT
- Finished: 5/2 23:00 PT

Step B

- See below each question for the self-evaluated scores, reasoning, and reflection.
- Total score: 70/100. See end of document for Points Breakdown.

Step C

- Upon reflection, I felt that my time management remains to fall short of what is expected.
- Like last exam, I still missed many of the questions and fail on crucial test taking technique such as reading the questions properly.
- Failing to analyze the questions properly have costed me valuable time in answering questions in the test. I have had to redo several questions because I did not read them correctly the first time.

1. Section, Date and Time:
   CSC 220.02+03, Due ##-##-#### at ##:## PM
2. Midterm Exam (2 exams, 0 dropped): 100 points

Full Name in **CAPITAL LETTERS**	**SFSU ID**
KULLATHON SITTHISARNWATTNACHAI	921425216

3. To prepare for this exam, please review all the related materials including WEEK 01-16 packages, slides, mock-up exam(s), reading assignments, in-class practices, sample programs posted in the File Manager, and assignments.
4. You do not need to print this exam. No paper. No handwriting. No scanning. Please type up all your answers in the answer space available in the exam. The provided exam will be in Microsoft Word format. Please submit a single PDF via iLearn.
5. All the rules of an actual exam apply to this exam such as: closed books, closed notes, and no communication with anyone except the course instructor. The course instructor will be available on Zoom or email during the exam time: zoom.ducta.net
6. Please ask all your questions, if any, during the review sessions. Thank you.

HONOR CODE:

- Please follow the CS Department's policies: https://cs.sfsu.edu/student-policies

- Please follow the course's policies: http://csc220.ducta.net/00-README-StudentConduct_AcademicHonesty.pdf

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122

PART A – 50 Points

A.1 - 5 Points

What does **`compareTo()`** possibly return when the method compares **`x`** vs. **`y`**?

Assuming x and y refer to strings, then: "x".compareTo("y") will return -1 and "y".compareTo("x") will return 1.

Step B	
Original score:	*5/5*
Reasoning and corrections:	*No deductions.* • *Tested the case in an IDE and works as expected.*
Room for improvement:	*n/a*

A.2 - 5 Points
Among **stack**, **queue**, **deque**, and **priority queue**, which structure(s) does not accept `null`? Explain why?

Priority queue (PQ) cannot accept a null value as it does not make sense. PQs orders its element by "priority value" which is derived from the value of each elements. For example, string elements may be ordered by their lexicographical order or integers may be ordered by their values. Having a null value prevents from these comparisons from being made and prevents the ADT from placing the element in an appropriate order.

Step B	
Original score:	*5/5*
Reasoning and corrections:	*No deductions.* *Reference: Lecture slides on ADTs* • *On PQs: "ordered": ordered by* ***priority*** *value. Thus an item cannot have* null *value."*
Room for improvement:	• *Perhaps an example would make a better explanation.*

A.3 - 10 Points
What is the output? And **why**?

```
String x, y;
char w;

x = "ComputingQuantum";
y = "ComputingClassical";

System.out.println(y.compareTo(x));
```

Output: -14
And why: 67-81 = -14. Compares the ASCII value of the first difference in the strings, which is Q and C where their values are 81 and 67, respectively.

```
x = "QuantumComputing";
y = "QuantumKomputing";

System.out.println(x.compareTo(y));
```

Output: -8
And why: 67-75 = -14. Compares the ASCII value of the first difference in the strings, which is K and C where their values are 75 and 67, respectively.

```
x = "ClassicalComputing";
y = "ClassicalComputingX";

System.out.println(y.compareTo(x));
```

Output: 1
And why: x and y are exactly identical except y has a one more character, as such the compareTo returns the length difference, which is one.

```
x = "ConventionalComputing";
y = "ConventionalComputinG";

System.out.println(x.compareTo(y));
```

Output: 32
And why: x and y are exactly identical except the last character where y has an uppercase and x has a lowercase. Compares the ASCII value of the last character, where all ASCII character value difference by case is always 32.

```
x = "x" + "y";
w = 'x';
System.out.println(x.compareTo(w));
```

Output: (Exception message)
And why: Incompatible types. Comparing char to String.

Step B	
Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *Tested the case in an IDE and works as expected.*
Room for improvement:	*n/a*

A.4 - 10 Points
How does **Selection Sort** work? What are the steps?

1. Start at the zeroth index.
2. Then, find the smallest value in the list and switch it with the zeroth index.
3. Steps 1 and 2 guarantees that the zeroth index is now the lowest in the list. So, find the next smallest value in the list starting from the first index.
4. Then, switch it with the first index.
5. Repeat the steps until the last index has been examined.

9 7 5 8 1 7 4 3 2
Show the contents of the array above **each time** a **Selection Sort** changes it while sorting the array into **ascending order**.

9	7	5	8	1	7	4	3	2
1	7	5	8	9	7	4	3	2
1	2	5	8	9	7	4	3	7
1	2	3	8	9	7	4	5	7
1	2	3	4	9	7	8	5	7
1	2	3	4	5	7	8	9	7
1	2	3	4	5	7	7	9	8
1	2	3	4	5	7	7	8	9

Step B	
Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *Tested the case in an IDE and works as expected.* *Reference: Packages 10 and 11.* • *Additional information provided on lecture slides on Sorting.*
Room for improvement:	*n/a*

A.5 - 10 Points
How does **Insertion Sort** work? What are the steps?

1. Consider the zeroth element to be a sublist of the list.
2. Then, sort the sublist. In this iteration, a singleton can be considered already sorted.
3. Next, add the adjacent element into the sublist.
4. Then, sort the sublist, if needed.
5. Repeat steps 1 and 2 until all of the elements have been added and sorted in the sublist.

9 1 8 7 6 5 2 3 4
Show the contents of the array above **each time** an **Insertion Sort** changes it while sorting the array into **ascending order**.

9	**1**	**8**	**7**	**6**	**5**	**2**	**3**	**4**
1	**9**	**8**	**7**	**6**	**5**	**2**	**3**	**4**
1	**8**	**9**	**7**	**6**	**5**	**2**	**3**	**4**
1	**7**	**8**	**9**	**6**	**5**	**2**	**3**	**4**
1	**6**	**7**	**8**	**9**	**5**	**2**	**3**	**4**
1	**5**	**6**	**7**	**8**	**9**	**2**	**3**	**4**
1	**2**	**5**	**6**	**7**	**8**	**9**	**3**	**4**
1	**2**	**3**	**5**	**6**	**7**	**8**	**9**	**4**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Step B

Original score: *10/10*

Reasoning and corrections: *No deductions.*
- *Tested the case in an IDE and works as expected.*

Reference: Packages 10 and 11.
- *Additional information provided on lecture slides on Sorting.*

Room for improvement:
- *Actually read the given array first. Assumed it was the same as A.4 in the first place and wasted time performing the sorting twice.*

A.6 - 10 Points
How does **Shell Sort** work? What are the steps?

1. First, use the number of index and half it to get n/2 where n is the size of the list.
2. Then, consider each of the n/2 sublists separately and sort them using insertion sort.
3. Combine the indexes from step 2 and half the value of n/2 again (now n/4). Then, repeat step 2.
4. Repeat 1 and 2 again if needed until n =/ 2 == 1.
5. Finally, perform insertion sort on the resulting list.

9 6 8 7 5 1 3 2 4 5
Show the contents of the array above **each time** a **Shell Sort** changes it while sorting the array into **ascending order**.

9	**6**	**8**	**7**	**5**	**1**	**3**	**2**	**4**	**5**	10/2=5
1					**9**					#1
	3					**6**				#2
		2					**8**			#3
			4					**7**		#4
				5					**5**	#5
1	**3**	**2**	**4**	**5**	**9**	**6**	**8**	**7**	**5**	5/2=2
1		**2**		**5**		**6**		**7**		#1

	3		4		5		8		9	#2
1	3	2	4	5	5	6	8	7	9	2/2=1
No more halves, do insertion										
1	3	2	4	5	5	6	8	7	9	
1	3	2	4	5	5	6	8	7	9	
1	2	3	4	5	5	6	8	7	9	Already sorted from 2nd insertion
1	2	3	4	5	5	6	8	7	9	
1	2	3	4	5	5	6	8	7	9	
1	2	3	4	5	5	6	8	7	9	
1	2	3	4	5	5	6	8	7	9	
1	2	3	4	5	5	6	8	7	9	
1	2	3	4	5	5	6	8	7	9	

Step B

Original score: *10/10*

Reasoning and corrections: *No deductions.*

- *Tested the case in an IDE and works as expected.*

Reference: Packages 10 and 11.

- *Additional information provided on lecture slides on Sorting.*

Room for improvement: *n/a*

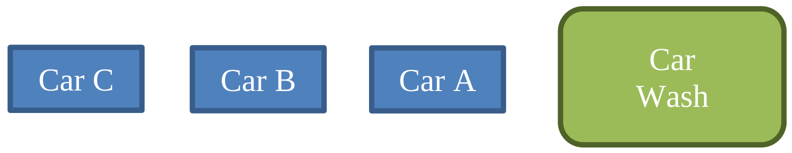
PART B – 50 Points

B.1 - 10 Points

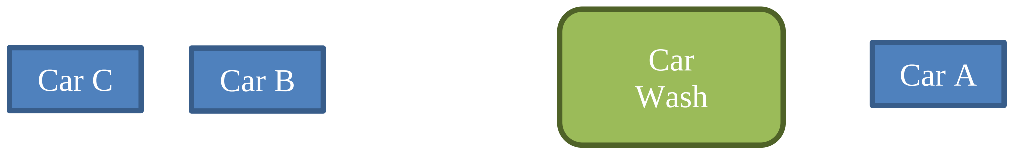
What is a **queue**? How does it work? Use an example and diagrams to explain its operations.

A queue is a data structure that preserves the order of insertion. Much like a real-life queue line, data goes in one way (at the end of the queue) and is removed on the other end (at the front of the queue). In other words, first item in, is first item out. This is usually referred to as First In, First Out (FIFO).

For example, consider a car wash where there is a lineup of vehicles waiting to be served.



Car A arrives first at the car wash, and so it is served first. In this sense the car wash can only access (serve) the vehicle at the front of the queue.



After serving Car A, the car wash is done with that vehicle, and so it removes it from the queue.

The order is continued, where the next car in the queue is served. The last car to be serve is Car C as it is at the end of the queue and is therefore accessed last.

Display the **Queue** at the **marked lines**.

```
Queue<Integer> q = new LinkedList<>();
for (int i = 3; i <= 8; i++) {
    q.add(i);
}
q.remove(q.size() - 1);                  // 1
q.offer(q.element() + 3);                // 2
System.out.println(q.contains(9));   // 3
q.add(q.remove());                       // 4
q.offer(q.peek());                       // 5
```

Working: starts with 3 4 5 6 7 8 (size=6)

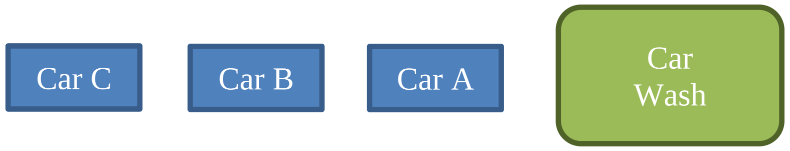
FRONT	
1.	**3 4 6 7 8**
2.	**3 4 6 7 8 6**
3.	*Output: false*
4.	**4 6 7 8 6 3**
5.	**4 6 7 8 6 3 4**

Step B	
Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *Tested the case in an IDE and works as expected.* *Reference: Lecture slide on ADTs.*
Room for improvement:	*n/a*

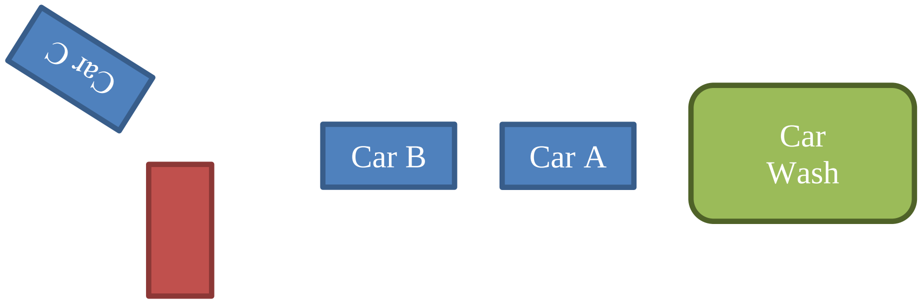
B.2 - 10 Points
What is a **deque**? How does it work? Use an example and diagrams to explain its operations.

A deque is similar to a queue. Deque stands for double-ended queue. As the name suggests, instead of just being able to access elements from either ends, a deque allows for all of the operations permitted on either ends of the queue on both sides of the queue. Hence, a "double-ended queue" – i.e., deque.

Continuing from the car wash example. With queue, one may not remove Car C until it gets to the car wash (front of queue).



However, with deque, since removal (among other operations) are permitted on either end, Car C can be removed. For instance, another vehicle may T-bone Car C, removing it from the queue.



Then, the queue (now "deque") now consists of only Car B and Car A, where Car A remains at the front of the deque.

Display the **Deque** at the **marked lines**.

```
Deque d = new LinkedList<>();
d.add("H");
d.addFirst("O");
d.addLast(d.contains("P"));            // 1
d.offer(d.element());                  // 2
d.offerLast(d.remove());               // 3
d.offerFirst(d.remove(d.element()));   // 4
d.push(d.remove("E"));                 // 5
```

Working: before 1, d is O H

FRONT	
1.	**O H false**
2.	**O H false O**
3.	**H false O O**
4.	**true false O O**

5.	**false true false O O**

Step B	
Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *Tested the case in an IDE and works as expected.* *Reference: Lecture slide on ADTs.*
Room for improvement:	• *Perhaps a more robust and less violent example may better explain the ADT, but I was running out of time so I had to use existing explanations to elaborate.*

B.3 - 10 Points
What is a **priority queue**? How does it work? Use an example and diagrams to explain its operations.

Display the **Priority Queue** at the **marked lines**.

```
PriorityQueue<String> pq = new PriorityQueue<>();
pq.offer("C");
pq.add("O");
pq.offer("M");
pq.add("P");
pq.offer(String.valueOf(pq.remove(pq.peek())));      // 1
pq.add(String.valueOf(pq.contains("X")));            // 2
pq.add(String.valueOf(pq.contains(pq.remove())));    // 3
pq.offer(pq.remove());                               // 4
pq.add(pq.peek());                                   // 5
```

FRONT	
1.	
2.	
3.	
4.	
5.	

Step B	
Original score:	*0/10*
Reasoning and corrections:	*No points. Did not attempt question; ran out of time.*

Correction:
Reference: Lecture slide on ADTs.

- *Priority queues (PQs) places objects by their "priority value."*
- *"**Priority Queue** is an ADT which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it."*

Using the given code yields the following for the second part:

FRONT	
1.	*[M, O, P, true]*
2.	*[M, O, P, true, false]*
3.	*[O, false, P, true, false]*
4.	*[O, P, false, true, false]*
5.	*[O, P, O, true, false, false]*

Room for improvement:

- *I could have at least copied the explanation part from an earlier question to get partial credit for this question.*

B.4 - 10 Points
What is a **list**? How does it work? Use an example and diagrams to explain its operations.

Display the **List** at the **marked lines**.

```
LinkedList li = new LinkedList();
for (int i = 3; i <= 8; i++) {
   li.add(i);
}
li.remove(li.get(2));
li.set(4,li.peekLast());
li.add(li.size()+1);                // 1
li.addFirst(li.get(li.size()-2));   // 2
Collections.sort(li);               // 3
li.add(li.indexOf(3));              // 4
li.addFirst(li.peekFirst());        // 5
```

FRONT	
1.	
2.	

3.	
4.	
5.	

Step B

Original score:	*0/10*
Reasoning and corrections:	*No points. Did not attempt question; ran out of time.*

Correction:
Reference: Package 13.

- *"An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list."*

Using the given code yields the following for the second part:

FRONT	
1.	*[3, 4, 6, 7, 8, 6]*
2.	*[8, 3, 4, 6, 7, 8, 6]*
3.	*[3, 4, 6, 6, 7, 8, 8]*
4.	*[3, 4, 6, 6, 7, 8, 8, 0]*
5.	*[3, 3, 4, 6, 6, 7, 8, 8, 0]*

Room for improvement:

- *At the very least, I could have explained that they were an extension of arrays.*
- *This would have been one of the easier code to trace. It may have been wiser to skip other ones in order to score these 10 points first.*

B.5 - 5 Points
Given a **queue q** of Integer elements, please write code to check if the elements are:

- Not Sorted
- Sorted in Ascending Order
- Sorted in Descending Order

Step B	
Original score:	*0/5*

Reasoning and corrections:	*No points. Did not attempt question; ran out of time.*
	
	Correction:
	It's an awful way to solve it, but it certainly works.
	
	• *ArrayList original = new ArrayList(q);*
	ArrayList temp = new ArrayList(q);
	
	Collections.sort(temp);
	if (original.equals(temp)) {
	return "Sorted in Ascending Order";
	}
	Collections.sort(temp, Collections.reverseOrder());
	if (original.equals(temp)) {
	return "Sorted in Descending Order";
	}
	return "Not Sorted";

Room for improvement:	• *Better time management.*

B.6 - 5 Points
Given 2 **lists** of **comparable `String`** elements and of different lengths, please write code to output:

- *"The tail of A is the reverse of B."* if
  listA: x, y, z, f, a, b, d, **A, B, C, D, E, F**
  listB: **F, E, D, C, B, A**

- *"The tail of A is NOT the reverse of B."* if not

Step B	
Original score:	*0/5*
Reasoning and corrections:	*No points. Did not attempt question; ran out of time.* • *Looking back, I still do not understand what the question is asking for.*
Room for improvement:	• *Better time management.*

Points Breakdown

Question	Scored	Total
A.1	5.0	5.0
A.2	5.0	5.0
A.3	10.0	10.0
A.4	10.0	10.0
A.5	10.0	10.0
A.6	10.0	10.0
B.1	10.0	10.0
B.2	10.0	10.0
B.3	0.0	10.0
B.4	0.0	10.0
B.5	0.0	5.0
B.6	0.0	5.0
Total	70.00	100.00

DO NOT DETACH THIS PAGE FROM YOUR EXAM.
You may get partial points for what you write on this page.