1. Section, Date and Time:
   CSC 220.02 + .03 - Due **05-16-2020 at 11:55 PM**

2. Final Exam (1 exam, 0 dropped): 100 points

Full Name in **CAPITAL LETTERS** | **SFSU ID**

KULLATHON SITTHISARNWATTANACHAI	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 (zoom.ducta.net) or via email during the exam time.
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 – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

How does **`compareTo()`** work differently when comparing 2 **`String`** objects versus comparing 2 **`enum`** objects?

- compareTo() in String compares the lexicographical values of each String, whereas compareTo() in enum compares the order in which they are declared.
- In Strings, compareTo() will compare the ASCII values of the first difference in the Strings.
  - If two Strings differ in length, where they are exactly identical before the length differ, then compareTo() will return the difference in length.
  - If the content of the Strings are exactly identical, then compareTo will return zero.
- In enum objects, compareTo compares the position in which they are declared.
  - Their positions are similar to the index values of those in an array, where their values are dictated by the order of declaration. Instances declared earlier have a lower value.
  - Comparing two identical enum objects yields zero.

A.2 - 5 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

The SFSU One Stop office asks you to recommend a data structure to manage the line of students they serve. Which one will you use? Please explain why. [ ] **Stack** [X] **Queue** [ ] **Deque** [ ] **Priority Queue**

- Queues would be a suitable data structure to manage a queue at a service center.
- **First come, first served:** To maintain order, the office would serve the students in the order of arrival.

- **First in, first out:** The queue data structure behaves in the same manner as, well… a queue. First item in is the first item out.
- Since the behavior of queues mimics the behavior of a real-life queue, it would be a suitable data structure to represent the line at the service center.
    - New students are added to the back of the queue.
    - Students that arrived earlier will be in the front of the queue and will be served first (effectively *removing* them from the queue, once called).
- One of the ways to implement the solution would be to store the queue with a custom data type that represents a student. This will allow the office to utilize the queue data structure to call up the next student in order and obtain all of the necessary information associated with the student to assist them.

A.3 - 5 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

What is the output? And **why**?

```
String a, b;
String c = new String("Google");

a = "GooglePixel4XL";
b = "GooglePixel4a";
System.out.println(a.compareTo(b));

OUTPUT: -9

WHY: 88 - 97 = -9. Comparison of characters' ASCII values at first difference, "X" and "a".


a = "AndroidStudioOreo";
b = "AndroidStudioPie";
System.out.println(b.compareTo(a));

OUTPUT: 1

WHY: Comparison of characters' ASCII values at first difference, "P" and "O". Same character case
and exactly one character apart.


a = "TESLAx";
b = "TESLAy";
System.out.println(b.compareTo(a));

OUTPUT: 1

WHY: Comparison of characters' ASCII values at first difference, "y" and "x". Same character case
and exactly one character apart.


a = "Kotlin";
b = "KotlinX";
System.out.println(a.compareTo(b));

OUTPUT: -1

WHY: Length difference. String a and b are exactly identical until index 5, where b has one more
character.


System.out.println(a.compareTo(c));

OUTPUT: 4

WHY: 75 – 71 = 4. characters' ASCII values at first difference, "K" and "G". Same character case
and exactly four characters apart.
```

A.4 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

Explain the 3 methods used to make **Deep Copies**.

1. One could use a *copy constructor* to create a deep copy of an object. A copy constructor is a constructor that accepts another instance of the class to create a (deep) copy of said object.
   A basic copy constructor requires that it sets the values of the fields according to those of the provided instance. This is done by assigning the fields of the new instance to the value of the field of the provided instance using their respective getters. It should also be noted that those fields will require a way for outside members to access its value (i.e., getters). In so doing, the values of the fields in the new instances are dereferenced from the provided instance, thus creating a deep copy of provided instance while ensuring that changes made in the new instance are not reflected in the one provided.
2. A (static) *copy method* may also be provided to create a deep copy of an object. The method should be static in order for it to be accessed by referencing the class itself and not its instance. A copy method accepts a parameter that is an instance of the class, from which it may invoke the *copy constructor* mentioned earlier using the *new* keyword and returning its result.
3. Lastly, the *Cloneable.clone()* method may be utilized in order to "clone" (create a deep copy) of the object. Do note however, that the parent method do not create deep copies of nonprimitive fields. Instead, the pointers for those fields will still reference the instance that the original field point to. As such, the method must be overridden and implemented properly in order to clone and dereference said fields.
   To utilize *Clonanble.clone()*, one may override its implementation and invoke the parent method to clone the primitive fields. For all other nonprimitive fields, the programmer must manually create a deep copy for each of those fields by using a method similar to the implementation to those of the *copy constructor*.

The SFSU University Police Department has been sending out live updates when there are incidents happening on campus. Would you use **Deep** or **Shallow** copies to implement this communication? Please explain in detail.

- Shallow copies would be appropriate for implementing this type of communication.
- Shallow copies ensure that the information utilized by all parties are from a single source and thus maintaining the consistency of the information.
- In this case, the source of the information is the police department, and the copies made by different clients are the general public.
- Since the nature of the communication of status updates are one-way, the clients (general public, in this case) have no need to mutate the information sent.
- Ensuring a single, authoritative source of information ensures that any data broadcasted are up to date for all clients and may allow for retractions, depending on the information.
- This may be best analogous to a police department's Twitter account.
    - Information can be sent from the account to inform the general public, where they may "follow" the account.
    - Further (deep) copies may be made by the public by "re-Tweeting," which ensures that the data shared comes from the authorities.
    - Should any corrections or retractions be made, the police department may send out another Tweet, which will always be updated. Or delete them, preventing the public from accessing obsolete information.

A.5 - 5 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

How is **enum** a special type? Please briefly explain 3 characteristics of **enum.**

1. The purpose of enum is to represent a set of constants. As such, it is a special class in the way that its constructors are always private, meaning that its instances may only be created inside of the enum. Any value associated with an enum is therefore predefined and cannot be changed.
2. Unlike a class, enum fields (which are their instances) may be declared without the use of a semicolon or the otherwise-redundant declaration of its type. Note that the former only applies to the simplest types of enum, where they do not require the use of a constructor (and will therefore implicitly invoke the default).
3. Enum constants are final and cannot be changed. This means that once they are defined, they may not be changed later. Although, it should be noted that this do not necessarily apply to its field, wherein a public setter may allow for its value to be changed. However, the number of enums will remain constant as its constructor are always private and cannot be invoked outside. This guarantees that the usage of the enum outside the class will be guaranteed to be one of those declared at the start.

A.6 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

How does **Insertion Sort** work? What are the steps?

1. Consider the first element to be a (sorted) sublist of the list.
2. Then, add the element adjacent to the end of the sublist to the sublist.
3. Sort the sublist, if needed.
4. Repeat (2) and (3) until the sublist contains all elements of the list.

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

8	**2**	**9**	**7**	**6**	**5**	**1**	**3**	**4**	**ORIGINAL LIST; SINGLETON SUBLIST [8] ALREADY SORTED**
2	**8**								
2	**8**	**9**							
2	**7**	**8**	**9**						
2	**6**	**7**	**8**	**9**					
2	**5**	**6**	**7**	**8**	**9**				
1	**2**	**5**	**6**	**7**	**8**	**9**			
1	**2**	**3**	**5**	**6**	**7**	**8**	**9**		
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	

A.7 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

You are hired to create a simple Dictionary application. Your dictionary can take a search key from users then returns value(s) associated with the key.
- Please explain **how** you would implement your dictionary.
- Please state clearly **which data structure(s)** you will use and explain your decision.
- Please state clearly **at least 3 methods** you will use and explain your decision.

- We can use Google Guava's implementation of ArrayListMultimap to represent the dictionary, where each key of the map represents the word, and its corresponding value is the list of definitions (Strings).
- ArrayListMultimap's structure fulfills the requirement of a "simple dictionary" as defined above, where each definition is linked to a word, and where each word may contain multiple definitions.

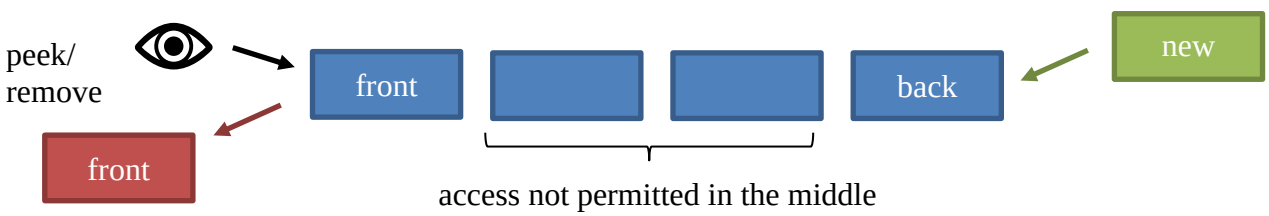
- The ArrayListMultimap data structure is Google's implementation of the Map, where it utilizes Lists to store the value associated with each keys.
- The definitions to be used for the dictionary can be stored in an enum, with each instance containing the word and an array of definitions.
    - For a minimal implementation of this enum class, it should have at least two (private) fields: a String field for the word and a String array field for the definitions.
    - A **getter and setter methods** for those respective fields.
    - At least one constructor accepting a String for the word field, and a vararg to accept an arbitrary amount of definitions for each instance.
- To create the dictionary, we first instantiate the ArrayListMultimap using the static **create() method** which we call ***myDict*.**
    - ***myDict*** will be used to store the word and definition in a key-value pair, where the definition(s) of each word will be accessed through its word.
- Then, to retrieve the definitions stored in our enum class, we use the **values() method** to access all of the defined instances and iterate through them.
    - As we iterate through our enum database, we invoke the **put() method** on the ArrayListMultimap instance in order to store the word and its definition as a key-value pair.
    - To store the definition in *myDict*, use the **getters** of the enum to access the word and definition, and invoke the **put()** method for each definition in field.
- To look up the dictionary, simply invoke the **get() method** on ***myDict*** to retrieve the definition for a given word.
    - The argument should be a String that is a sanitized version of the user input.
    - The returned value is a list. A **forEach() method** may be used to invoke the **println() method** using a lambda expression.
    - If the returned list is empty, then display an error message to the user.
- It may be wise to separate the implementation of the user's interface from the dictionary itself.
    - As mentioned, input sanitation (at the very least, stripping leading and trailing whitespaces) should be implemented. Along with messages to display to the user when a definition cannot be found.

PART B – 50 Points

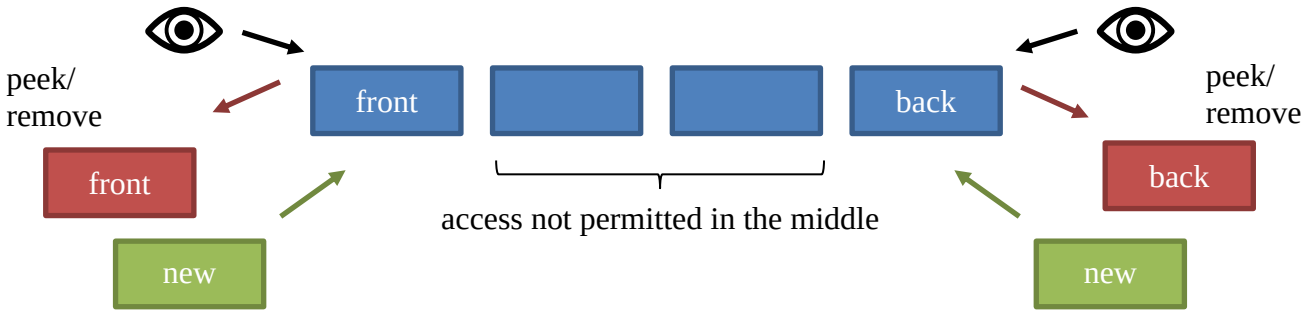
B.1 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

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

- A deque is a **d**ouble-**e**nded **que**ue. Essentially, it is a queue where the operations on either sides of the queue are permitted on both sides.
- Since deque is an extension of a queue, let us explore what a queue is.
    - A queue is a first-in, first-out data structure where the elements are ordered by the order of insertion. Elements that are added to queue first are at the front of the list, and elements that are added to the end of a queue are at the end.
    - Much like an actual queue, elements that arrive in the queue first are removed first. And elements that arrive later are removed last.
        - At the front, queues allow for elements to be removed (retrieved) or peeked.
        - At the back, queues allow for elements to be added.
        - Operations are only permitted at the front and back of the queue.



- With the characteristics of a queue in mind, we expand that to the *double-ended queue*, or "deque," where the permitted operations are extended to both sides.
    - Operations are still restricted to either end of the ADT.
    - For the sake of completeness: in a deque, you can:
        - Peek, remove, and add new elements to the front.
        - Peek, remove, and add new elements to the back.



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

```
Deque d = new LinkedList<>();
d.addLast("T");
d.addFirst("e");
d.add("s");
d.addFirst("L");
d.addLast("a");
d.offer(d.remove(d.contains("T")));        // 1
d.push(d.remove("S".toLowerCase())); // 2
d.addLast(d.remove(d.contains("Z")));  // 3
d.offer(d.remove());                       // 4
d.offerFirst(d.remove(d.element()));      // 5
```

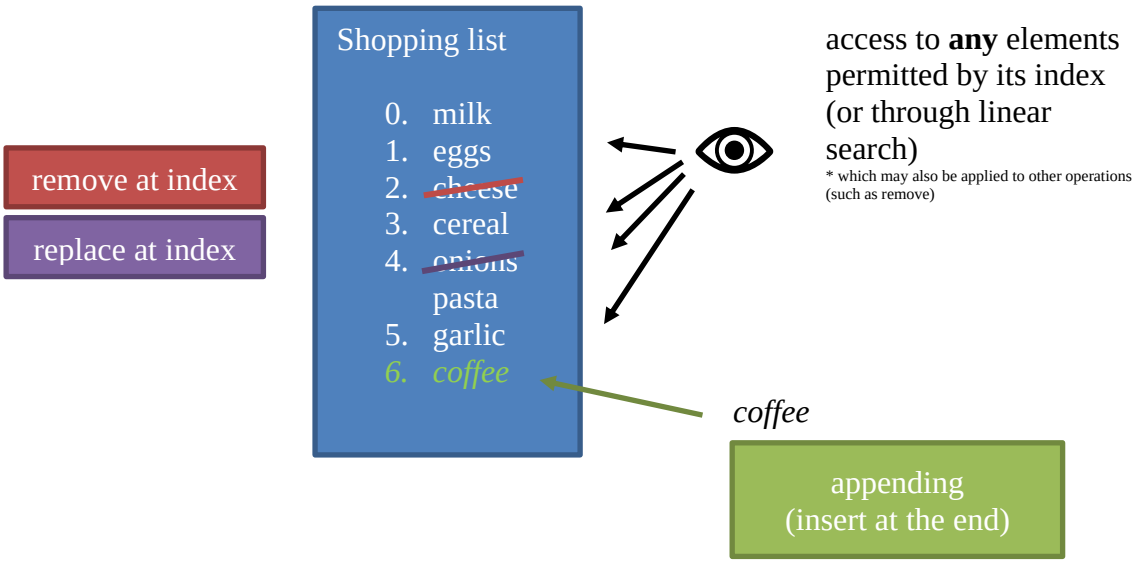
Working: before 1, d contains: L e T s a

FRONT	
1.	**L e T s a false**
2.	**true L e T a false**
3.	**true L e T a true**
4.	**L e T a true true**
5.	**true e T a true true**

B.2 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

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

- A list is an ordered collection of items.
- It is perhaps the simplest and most flexible of all the data structure in terms of usage. Like queues and deques, it is an ordered collection. It also allows for all operations that deques permits. Additionally, those operations can be applied to any elements within the list (unlike deques, where those operations are restricted to either ends of the ADT).
- Lists allow for indexed access. This means that elements can be inserted from the front, back, or at a particular index (replace). Similarly, they can also be removed from the front, back, or at a specified index.
- A list can be compared to those of a shopping list, where items are usually appended to the end of the list. But can also be accessed by its indexes.



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

```
LinkedList li = new LinkedList();
for (int i = 9; i >= 2; i--) {
   li.addLast(i);
}
li.remove(li.get(4));
li.set(6, li.size() % 10);
li.add(li.size());                          // 1
li.addFirst(li.get(li.size() - 4));         // 2
li.add(li.indexOf(5));                      // 3
li.addFirst(li.remove(li.indexOf(6)));      // 4
li.addFirst(li.peekLast());                 // 5
System.out.println(li);
```

Working: before 1, li contains: 9 8 7 6 4 3 7

FRONT	
1.	**9 8 7 6 4 3 7 7**
2.	**4 9 8 7 6 4 3 7 7**
3.	**4 9 8 7 6 4 3 7 7 -1**
4.	**6 4 9 8 7 6 4 3 7 7 -1**
5.	**-1 6 4 9 8 7 6 4 3 7 7 -1**

B.3 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

Write a statement to create a **Google Guava Multimap** instance.

Notes:

- *Assuming "**Google Guava Multimap** instance" refers to any instance of a class that implements Multimap.*
- *Since the instruction did not specify the datatype associated with the instance, the below examples will use a key-value pair of type String.*
- *The below examples will use the ArrayListMultimap class, which implements Multimap.*

- ArrayListMultimap<String, String> numbers = ArrayListMultimap.create();

And write code to input 5 entries into this data structure.

- numbers.put("five", "5");
- numbers.put("four", "4");
- numbers.put("three", "3");
- numbers.put("two", "2");
- numbers.put("one", "1");

And use **keyset()** to display all the contents stored in this data structure.

The following will display a space-delimited list of keys.

- numbers.keySet().forEach(k -> System.out.printf("%s = %s%n", k, numbers.get(k));

B.4 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

Given a **queue q** of String elements, please write code to output:

- "*Every other.*" if the elements are organized in these patterns:
  a, OK, b, OK, c, OK, d
  a, OK, b, OK, c, OK, d, OK
  OK, a, OK, b, OK, c, OK, d
  OK, a, OK, b, OK, c, OK, d, OK

- "*NOT every other.*" if other patterns.

```
// first two elements, and flags to check whether they are alternating
String a = q.remove(), b = q.remove();
boolean aEa = false, bEa = false;

for (int i=1; q.size()>0; i++) {
    if (i%2==0) bEa = b.equals(q.remove());
    else aEa = a.equals(q.remove());
}

If (bEa||aEa) System.out.println("Every other.")    // at least one is alternating
else System.out.println("NOT every other.")
```

B.5 - 10 Points – *Your answer must be in your own words, be in complete sentences, and provide very specific details to earn credit.*

Given 2 Priority Queues, please write code to detect if one queue is the reverse of the other. Our program should output meaningful message(s).

- Given two priority queues, pqA and pqB, we assume that the size of pqA and pqB are the same and contains elements of the same type.
- If pqA and pqB are not of the same size, then they cannot be the reverse of each other.

```
Object[] a = pqA.toArray(), b = pqB.toArray();
boolean reversed = true;

for (int i = 0; i < a.length; i++) {
    if (!a[i].equals(b[a.length-1-i])) reversed = false; break;
}

if (reversed) System.out.println("A is a reverse of B");
else System.out.println("A is NOT a reverse of B");
```

PART C – 5 Extra Credit Points

Class **Entry** has 1 property: String **finalExam**. Please write the **hashCode** method for this class using **Joshua Bloch's** recommendation.

- Unfortunately, I do not remember what the formula was. :(

And write the **euqals()** method to compare 2 objects of this class.