MIDTERM EXAM INSTRUCTIONS

1. Midterm Exam: **50 points w/ 0 E.C. points**
2. Due Date & Time: **07-12-2020 at 6:00 PM**

WHAT TO SUBMIT

1. Take-home Exam Report (instructions provided below)

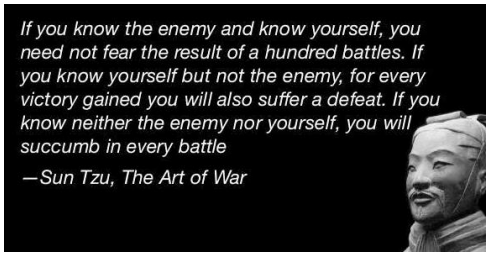
HOW TO SUBMIT AND THE RULES TO FOLLOW

- Submit via iLearn, the Assignment 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	55	
03	55	
MIDTERM EXAM	50	
TOTAL	185	

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 to know what we do not know.

Unfortunately, the campus closure does not allow us to host Midterm exam in person. However, we do not have to give up the goal of the exam which is to have an opportunity to evaluate our knowledge of the topics which we covered.

Let us self-evaluate. Then we will go through the questions and the answers together.

We should be able to find an answer to every exam question from the packages, from the in-class discussions, from our assignments, and from the other course materials. It is important that we take this exam no less seriously than how we take an actual exam. We need to know what we do not know. Learn that again to gain a strong foundation for the second half of the semester. Thank you.

PART A – Take the Exam, **20 points**

- Please allocate 115 quiet minutes to take the exam on page 2 to page 9 as seriously as how you take an actual exam.
- Stop right at minute 116 and grade your answers yourself. Please:
  1. Record the date and time when you started and stopped taking the exam.
  2. (*Do part B then get back to record*) Your score for each question and your total score out of 100.
  3. For each answer, list all the mistakes then explain why, you think, you made the mistakes. If you did not make any mistakes, please document how you could have done better.
  4. Include your graded exam in this part.
  5. **Problem Solving**: Reflect if you managed the exam time efficiently and if you strategized your test-taking successfully.

PART B – Correct Your Answers, **20 points**

- Please review the related course materials and please write your own code to find a correct answer for each question.
- Please document the correct answers, document how you found the correct answers, and document where you found the materials which support the answers.

PART C – Retake the Exam, **10 points**

- Please do all the steps in part A again.
- Repeat part A to part C again if necessary.
- Please think if the same topics will be tested again in 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 Final exams. And this is also the best way to prepare ourselves to succeed in the second half of the semester. Thank you.

1. Section, Date and Time:
   CSC 220.01, due 07-12-2020 at 6:00 PM

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

KAYVAUN KHOSHKHOU	920357344

2. Midterm Exam (1 exam, 0 dropped): 100 points
3. To prepare for this exam, please review all the related materials including WEEK 01-06 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 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

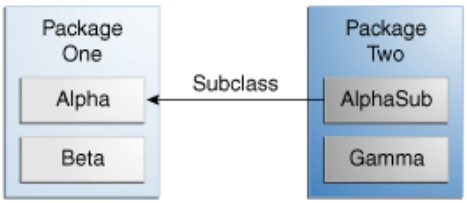
---

STARTING TIME: 07/11/20 4:10 PM (NOTED LATER - RAN OUT OF TIME AT END OF PROBLEM B.5)
HIGHLIGHTED TEXT ARE MY CORRECTIONS (PART B OF INSTRUCTIONS)

PART A – 40 Points

A.1 - 5 Points
Please complete the "where the members of the Alpha class are visible" table by filling in Y or N.

Modifier	Alpha	Beta	AlphaSub	Gamma
public	Y			
protected	Y			N
no modifier	Y		N	N
private	Y	N	N	N



If a member of the class **Beta** is **protected**, is this member visible to **Gamma**? If yes, please explain why. If not, please explain what we should do make it possible.

If a member of class Beta is protected then the member is not visible to Gamma. In order to make it visible to Gamma we need to make its access modifier public. This will allow other classes to see the member.

I believe my answer here is correct to the fullest extent. An example of private and public access modifiers being used can be seen in Package 01. On line 821 we have abstract class Animal. The variable weight is classified as a double and given private access. It is accessible within the protected constructor Animal. Outside of this class, for example in the code starting on line 841, Tiger class, we would not be able to access "weight" because it is private in the Animal class. **(5/5)**

A.2 - 10 Points
- What are the 2 ways to use the keyword "**super**" in a class? - Give code examples to demonstrate your answers.

"**super**" can be used to invoke parent class methods, as well as to refer parent class instance variables.

We can use "**super**" in a case like this to refer parent class instance variables:

```
class Student{
String favoriteFood = "Pizza";
}
class Professor extends Student{
String favoriteFood = "Pasta";
void craving(){

// This will print out "Pasta" since the current class idenitifies the variable "favoriteFood" as such.
System.out.println(favoriteFood);

// With "super" this will print out "Pasta" because it is accessing the Student class to print the data for favoriteFood from that class
System.out.println(super.favoriteFood);
}}
class test{
public static void main(String args[]){
Professor p = new Professor();
p.craving();
}}
```

OUTPUT:
Pasta
Pizza

We can also use "**super**" in a case like this to invoke parent class methods:

```
class Student{
void lunch(){
System.out.println("Time for Pizza!");
}}
class Professor extends Student{
void lunch(){
System.out.println("Pasta time!");
}
void snack(){
System.out.println("Maybe I'll just have an apple instead…");
}
void print(){
super.lunch();
snack();
}}
class test{
public static void main (String args[]){
Professor p = new Professor();
p.print();
}}
```

OUTPUT:
Time for Pizza!
Maybe I'll just have an apple instead…

Within Package 01 on line 350 we begin a class Person, and on line 406 we begin a class Student that extends Person. We use super on line 418 to refer to the parent class variables from Person. We also use super on line 451 to invoke the parent class method sayHi from Person. **(10/10)**

A.3 - 10 Points
When using a linked list for stack implementation, we use the first Node as the top entry of stack. Explain, in this linked list implementation, how we **add** and how we **remove** an entry from a stack. *Use linked nodes diagrams to save your time.*

add
if the stack is empty then it will start with null in the first position
stack: [null]
if we push or add a new node to the stack, it will go to the very top and will have a pointer pointing to it. Say we add node with 1
stack: [top of the stack] -> [1] -> [null]
then if we add a second node with 2 it will be inserted on top of 1, and point to 1 as well.
stack: [top of the stack] -> [2] -> [1] -> [null]
This will continue in this fashion until you remove a node

remove
If we decide to remove a node, we will remove the node that is being pointed to by the top of the stack.
stack: [top of the stack] -> [2] -> [1] -> [null]
Here is our stack from earlier
stack: [top of the stack] -> [1] -> [null]
That would be our new stack after removing the top node.

Explain one major difference between the behaviors of the LinkedBag and the Stack which we implemented.

A Stack's behavior resembles a system where the last item inserted can only be popped out first. This to me closely resembles those PEZ candy dispensers. Whereas the LinkedBag's behavior acts like a container for objects and methods to handle objects. There is no order in which you add or remove objects (etc.).

Within Package 06, starting on line 05 with StackIntro class, we can find many examples of push and pop. There is no diagram to reference, but I believe my diagrams accurately reflect what is being done in the abstract side of computer science. On line 15-18 we have stack.push(1001); stack.push(1002); stack.push(1003); stack.push(1004);. These values are pushed onto the top of the stack in that order. I believe this reference from Package 06 justifies my answer. **(10/10)**

A.4 - 5 Points
What are stored in each activation record and Why? Which method is pushed in the Program Stack first? Which method is popped out of the Program Stack last?

The storage of activation record follows; parameters, return value, local data, temporaries, control links, and access links.

Upon initial start up or execution, the main method is always the first method to be pushed in the stack. Everything else follows *after* the main method.

Upon closing or termination, the main method is also the last method to be popped out from the program stack.

I believe the comments within the first program in Package 06 justify my answer. Specifically from lines 43 – 49**.(5/5)**

A.5 - 10 Points - Part A of Assignment 02:

What is the ouput (what are in the Bag) when the 2 below lines are executed? **Please show the steps.**
String[] items = {"Z", "Y", "Y", "X", "S", "C", "A", "E", "M"};
testAdd(aBag, items);

When these two lines are executed, then all the values within items will be dumped into the bag and the output will resemble exactly what was entered as the string value for items.

OUTPUT:
ZYYXSCAEM

What is the output (what are in the Bag) when the 3 below lines are executed? **Please show the steps.**
String[] testString = { "X", "Y", "Z" };
aBag.removeAllOccurences(testString);
displayBag(aBag);

When the 3 lines are executed, then all the string values within the bag that resemble { "X", "Y", "Z" } will be removed from the bag, and the bag will print out its remaining contents similar to the first output, but with those 3 string values X, Y, Z excluded.

OUTPUT:
SCAEM

Line 236 from Package 04 is a good reference for this problem. Using remove to take out a string from the bag as it says on line 241 where in this case can be reference because we are removing the occurences of XYZ string in our bag. **(10/10)**

PART B – 60 Points

B.1 - 5 Points
Which of the statement(s) are erroneous and why?

```
MidtermExam midtermExam = new Exam();                // A
SFSUStaff person = new Person();                     // B
StackInterface<String> s = new ArrayStack<>();       // C
```

The way we work with these statements involves a certain syntax, such that "Test1 object = new Test2();" , this statement is only verified when Test2 **is-a** Test1. In the case of A, MidtermExam is-a Exam, but it doesn't quite work that well the other way around. The statement is not valid when reading in the format Test1 is-a Test2. Here we have MidtermExam as Test2 and Exam as Test1, hence the invalidity of the statement.

It looks like here my answer is correct but not complete, as under this logic B would also qualify as invalid. Person is-a SFSUStaff but SFSUStaff cannot be Person. Line 661 from Package 01 begins class "PersonTester" and in line 682 we can reference the same syntax that we used here. **(3/5)**

B.2 - 5 Points
What is the output if any?

```
abstract class Person {

    private final String name;

    protected Person(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

class Student extends Person {

    public Student(String name) {
        super(name);
    }

    public static void main(String[] args) {
        Student stu = new Student("Mickey");
        System.out.println(stu.getName());
        stu.setName("Super Mouse");
        System.out.println(stu.getName());
    }
}
```

*Attention to the keyword "**final**"*

The hint attention to keyword final signals to me that there is an imminent error, because final makes the value of the variable "name" permanent. After assigning a value for the first time to a variable with *final* attached to it, you cannot reassign the value of that said variable. On the line "this.name = name;" I believe there will be an error where the compiler will leave you with a message saying cannot overwrite value to final variable "name".

Line 870 from Package 01 we have final String EDIBLE, here in the comments we see it being called a final data field, which in turn references similarly to this problem where we CANNOT change the value of a final variable. Once it is initialized at a value we cannot change that value. **(5/5)**

B.3 - 10 Points
Anything wrong with the code? If yes, how to fix?

```
class CSC220 {
    private CSC220(){}
    private CSC220(int x){}
    private CSC220(int x, int y){}
}
```

Yes, instead of private, public should be used because if private is used then the constructors will not be accessible outside of the class CSC220.

And is it possible to create a sub class to this class? Why?

It is not possible to create a sub class to this class in its current state because the constructors are all private. The classes cannot be inherited and the base class object cannot be created for the creation of the child classes. If the class had its constructors set to public then I believe sub classes could be created.

From our class design guidelines Section 8 Interfaces vs. Abstract Classes, we know that classes follow a parent-child relationship. Meaning that the child class is dependent on the parent class. If the parent is hiding something or has a private variable, the child class will not be able to access it. Thus, a sub class can only have access to the main class if it has its access modifies set to public. **(10/10)**

B.4 - 10 Points
What is the output if any?

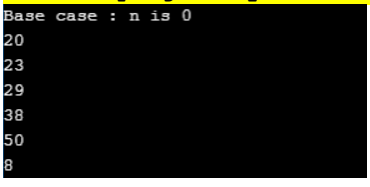
```
    public static void main(String[] args) {
        int x = 12;
        System.out.println(sumOf(x) - 42);
    }

    public static int sumOf(int n) {
        int sum = 15;
        if (n == 0) {
            System.out.println("Base case: n is " + n);
            sum += 5 + n % 2;
        } else {
            sum = sumOf(n - 3) + n;
        }
        System.out.println(sum);
        return sum;
    }
```

OUTPUT:

```
Base case: n is 0
20
23
29
38
50
8
```

Ran the program myself and got the same output.(10/10)



B.5 - 10 Points
This program outputs 10 lines. What are they?

```
Stack<String> resume = new Stack<>();
resume.push("JavaScript");
System.out.println("Is empty: \t" + resume.isEmpty());
resume.push("Scala");
resume.push("C++");
resume.push("Dart");
resume.push("Go");

resume.pop();
System.out.println("Stack : \t" + resume);
resume.push("Python");
System.out.println("search() : \t" + resume.search("Scala"));
System.out.println("pop() : \t" + resume.pop());
System.out.println("pop() : \t" + resume.pop());
System.out.println("search() : \t" + resume.search("Dart"));
System.out.println("After pop() : \t" + resume);
System.out.println("pop() : \t" + resume.pop());
System.out.println("Is empty : \t" + resume.isEmpty());
System.out.println("Stack:  \t" + resume);
```

1. `Is empty:  false`
2. `Stack:  [JavaScript, Scala, C++, Dart]`
3. `search():  4`
4. `pop():  Python`
5. `pop():  Dart`
6. `search():  -1`
7. `After pop():  [JavaScript, Scala, C++]`
8. `pop():  C++`
9. `Is empty:  False`
10. `Stack:  [JavaScript, Scala]`

FROM PACKAGE 06, WE HAVE MANY EXAMPLES OF IS EMPTY, POP, SEARCH, PUSH, ETC. STARTING WITH STACKINTRO WE ALREADY HAVE EXAMPLES OF SEARCH, POP, AND ISEMPTY. THROUGH LINES 20-25. FROM THIS REFERENCE AFTER READING THROUGH THE QUESTION AGAIN I WOULD NOT CHANGE ANYTHING ABOUT THE OUTPUT AS I BELIEVE IT IS CORRECT. **(10/10)**

TIME UP: 07/11/20 6:05 PM

B.6 - 10 Points
Implement *findTheThird* method in linked list that searches the bag for a given *entry*. If found,
- removes the first occurrence
- leave the second occurrence intact
- then replace third occurrence with the string "Found3rd"
- remove the rest of the occurrences

Return *false* if no replacement happened. Otherwise, *true*.

```
public boolean findTheThird (T entry)
```

Note: You may assume that *firstNode* is a private data in list
which references to first node.

N/A

B.7 - 10 Points
Assume that you have the following Bag object, myBag, with n String data:

BagInterface <String> myBag = new ArrayBag< >();

Write Java statements that create a newBag object which
contains non-duplicate data in myBag and marks the duplicate data.

Example:
if myBag contains data:
"A", "A", "A", "B", "B", "C", "D", " "
newBag object should contain:
"A", "DUP.1.A", "DUP.2.A", "B", "DUP.3.B", "C", "D", " "

Hint: You can use the Bag's methods:
int getCurrentSize (); boolean isFull (); boolean isEmpty (); boolean add (T newEntry); T remove (); boolean remove (T anEntry); void clear (); int getFrequencyOf (T anEntry);
boolean contains (T anEntry); T [] toArray ();

N/A

TOTALS:

Part A: 40 POINTS
Part B: 38 POINTS
TOTAL 78 POINTS

PART C – 10 Points
RETAKE

B.1 - 5 Points
Which of the statement(s) are erroneous and why?

```
MidtermExam midtermExam = new Exam();                 // A
SFSUStaff person = new Person();                      // B
StackInterface<String> s = new ArrayStack<>();        // C
```

The way we work with these statements involves a certain syntax, such that "Test1 object = new Test2();" , this statement is only verified when Test2 **is-a** Test1. In the case of A, MidtermExam is-a Exam, but it doesn't quite work that well the other way around. The statement is not valid when reading in the format Test1 is-a Test2. Here we have MidtermExam as Test2 and Exam as Test1, hence the invalidity of the statement. The same goes for B, since Person is-a SFSUStaff but SFSUStaff cannot be Person. So my final answer for B.1 is [A & B] are both erroneous.

Corrected using "PersonTester" from Package 01 line 682. **(5/5)**

FINAL TOTALS:

Part A: 40 POINTS (All correct, Great job!)
Part B: 40 POINTS (-20 POINTS from B.6 & B.7 incomplete)

TOTAL 80 POINTS