**CSE 220 Homework Assignment 3 (Due 3/10/23)**

**1) (20 pts)** Explain the difference between the major types of relationships among classes and other data types in OOP (dependency, aggregation, inheritance). Cite at least one example (covered in our course or otherwise) of each relationship.

**2) (20 pts)** Replace the following code with a version that uses a **for-loop**. **For full credit, you should have the for loop, and no more than *4 other statements*.  
 *String mystr = "My\_apple\_is\_red.";  
 String eostr = "";  
 int curindex = -2;   
 while (curindex < mystr.length()-2) {  
 curindex += 2;  
 eostr = eostr + mystr.charAt(curindex);  
 }  
 System.out.println(eostr);***

**3) (60 pts)** *(Classes and Methods)* For this problem you are going to build onto the existing hierarchy of **Student** and **Address** classes (included with this assignment – note that these are modified versions of the ones included in chapter 7 code) .

In the academic world, a *cohort* is a group of students who work on a project, objective, or task together. A cohort can simply include students in the same major at a university, for example. For this problem, you are to create class **Cohort**, which includes a *collection* of Students. You are free to store the collection using any means you wish, but an ArrayList is likely the simplest. **Note that any variables in Cohort must be private, to protect encapsulation**. You are to write the following 5 methods for Cohort:   
i. **public Cohort()** : a standard, default constructor – it should initialize the collection of students to be empty.

ii. **public void addStudent(Student newStudent)** : this method is a relatively simple one, and should simply add *newStudent* to the collection of students in the cohort.

iii. **public Student getStudent(String firstname, String lastname)** : this method returns the first Student in the cohort whose first and last names match the input values (firstname, lastname). If no Students match, the method should return **null**.

iv. **public Student getStudentEarliestName()** : this method should return the Student with the alphabetically earliest (i.e. A before B before C, etc.) *last name*. If two students have the same last name, the student with the alphabetically earliest *first name* should then be chosen to break the tie. (You can assume students in the Cohort won’t have identical first *and* last names.)

v. **public String toString()** : the standard toString method: in this case, it should produce a sequence of the students and their info according to the order in which they were added to the collection, separated by a blank/empty line per student.

Remember that the method **equals** should be used when checking to see if two strings match for method iii, rather than ==. Likewise, you will almost certainly want to make use of the **compareTo** method for method iv.

**While not mandatory, it is highly recommended that you test your Cohort class using the included CohortDriver.java file. Your output on screen should match (or come close to matching) what is provided in the file *CohortDriver\_output.txt*.**

**Responses to Problems #1 and #2 should be in .doc(x) or .pdf format. Upload this file and Cohort.java in *.zip file* format to Blackboard. The zip file should have the format “LN\_FN\_3.zip” where LN is your last name and FN is your first name.**

**1)**  **Dependency** unables to turn regular java classes into managed objects and to inject them into any other managed object. Dependecy is a relationship between one class depends on the other class perform a specific operation. One of the example professor gaves to us was a if a Car class has method called 'startEngine' and it may depend on a battery class to provide the gas to the car and make it move. If the battery class changes, the Car class will be affected.

**Aggregation** is a relationship between two classes where one class contains another class as a part of its state. Aggregation is also known as a has-a relationship. The main reason why we need Aggregation is to maintain code reusability, for example a Car class may contain a Wheel class as a part of its state. The wheel class can exist independently of the car class.

**Inheritance** is a relationship between two classes where one class is a subtype of other class. The subclass inherits the propeties and methods of superclass. For example, a Car class may be a subclass of a vehicle class. The Car class inherits the properties and methods of the Vehicle class.

**2)** String mystr = "My\_apple\_is\_red.";

String eostr = " ";

for (int curindex = 0; curindex < mystr.length(); curindex += 2) {

eostr += mystr.charAt(curindex);

}

**System.out.println(eostr);**