NORTH SOUTH UNIVERSITY
Department of Electrical and Computer Engineering
B.Sc. in Computer Science and Engineering Program
Mid Term II Examination, Summer 2019 Semester

Course: CSE 215 Programming Language II, Section-13
Instructor: Mohammad Rezwanul Huq (MRH1), PhD, Assistant Professor (Part-time)
Full Marks: 40 (20 will be counted for final grading)
Time: 1 Hour and 20 Minutes

Note: There are **5** (**FIVE**) questions, answer ALL of them. Marks of each question is mentioned at the right margin.

---

1. **Define** a class named **Weight** whose objects represent weights in English unit. The class should have two private instance variables of type int for the *pounds* and *ounces*. The Weight class must have the following methods. 8

- Include appropriate *constructors*, *setters* and *getters*.
- Include *toString* method that returns a string as per the following format-
  The Weight object has the value *x* pounds and *y* ounces. [where x and y are the actual values of pounds and ounces fields]
- Include *compareTo* method which must be invoked in associated with a caller object and takes an argument which is the callee object. The method returns 1 if the caller object has greater value than the callee object; returns 0 if both objects have the equal values; otherwise, returns -1. The following line shows the sample method definition.
  `public int compareTo(Weight w){ … }`
- Include the methods *add* for addition of amounts of weight. These methods should be invoked in associated with a caller object and takes an argument which is the callee object. The added value must be returned as another object of Weight type. The following two lines show the sample method definitions.
  `public Weight add (Weight m){ … }`

Further, **define** a main method within the **Main** class and demonstrates the functionalities of Weight class. You must create a couple of Weight type objects and the must call the methods – *add*, *compareTo* and *toString* accordingly.

2. Assume that you are one of the programmers developing an application for NSU Medical Center. Your job is to build a module through which the doctors can keep track of appointments made by the patients. For each day, there are ten appointment slots, starting from 1 to 10 (int type). Please note that, a patient can have many appointments whereas an appointment in a slot would be made for one patient only. You need to do the followings to accomplish that. 10

Define a class **Appointment** that has four instance variables: *appointmentId*, *date, slotId and isAvailable* (a boolean type; true if the appointment slot is available to book; false otherwise). Include appropriate constructors. Add *toString()* method that returns a string containing appointment's information.

Define another class **Patient** that has two instance variables: *patientId* and *patientName*. Include appropriate constructors. Also include setters and getters if required. Add *toString()* method that returns a string containing patient's information. Inside the patient class, you should also hold the appointment information which are made by a patient. Moreover, the Patient class must have an

`addAppointment(Appointment a)` method which is invoked when a patient wants to make an appointment. Please note that, when an appointment is made, *isAvailable* field would be set to false.

Write a **Main** class that includes a *main* method for testing the functionalities of these two classes: **Appointment** and **Patient**. Create a couple of relevant objects of both classes and invoke *addAppointment* method of the *Patient* class appropriately.

3. A hospital has mainly two types of patients: Outdoor patients are coming to visit a doctor and pays a visiting fee and Indoor patients are admitted into the hospital and pays fee in different categories such as bed charge, medicine charge, lab test charge etc. The hospital wants you to write a Java application that performs its bill calculations polymorphically considering the following specifications. 5

Patient class has two instance variables: *id* and *name*. **OutdoorPatient** class has one instance variable of its own, which is *visitingFee*. **IndoorPatient** has three instance variables: *bedFee, medicineFee* and *labTestFee*. All classes include appropriate constructors and *toString()* method. Patient class also include a method *bill()*.

Draw the corresponding class diagram. Use appropriate access specifier symbol, data type and return type of methods.

4. As per the use case specified in Question 3, *bill()* and *toString()* methods return the followings: 12

Class	*bill()* Method	*toString()* Method
Patient	returns 0	*id name*
OutdoorPatient	returns *visitingFee*	*id name* visiting fee: *visitingFee*
IndoorPatient	returns the sum of *bedFee*, *medicineFee* and *labTestFee*	*id name* bed fee: *bedFee* medicine fee: *medicineFee* lab test fee: *labTestFee*

Define *Patient* class. Include constructor(s), toString() and a method *bill()*.

Define *OutdoorPatient* class. Include constructor(s) and override bill() and toString() methods as per the specification shown in the table above.

Define *IndoorPatient* class. Include constructor(s) and override bill() and toString() methods as per the specification shown in the table above.

Write a *Main* class that has *main()* method. Within the *main()*, define an ArrayList of appropriate type. Instantiate two objects each from *IndoorPatient* and *OutdoorPatient* class and add those four objects into that array list. Invoke *bill()* and *toString()* methods polymorphically and print these values. At last, print the average bill of *IndoorPatient* type patients.

5. **Match** keywords in column A with appropriate statement in column B. 5

Keywords (Column A)	*Statements (Column B)*
A. Inheritance **B.** Class **C.** Association **D.** Method Overloading **E.** Instance Variable	**i.** methods having the same name with same method signatures in a superclass-subclass relationship **ii.** relationships among non-related classes **iii.** fields of a class **iv.** a blueprint of same type of objects **v.** variable belongs to the object **vi.** methods having the same name but different method signatures **vii.** 'is a' relationship **viii.** a general binary relationship between classes **ix.** variable belongs to the class **x.** 'has' relationship