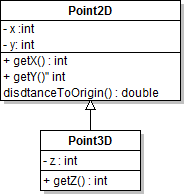
**Midterm Exam** (CSC341)

1. Assume the following:
   1. *IA* is an interface,
   2. *B is an abstract* *class implementing* *IA*
   3. *C is a concrete subclass of B*
   4. *D is a concrete subclass of C*

Declare and instantiate **all possible** objects using these types and their default constructors (for example, *C obj = new C();* is one of the possibilities).

1. In the following class hierarchy, subclass *Point3D* has an instance variable z to add another dimension of a point with one additional getter method.



* 1. Given only the information above, list all issues you can identify in this hierarchy.
  2. What would you do to design Point2D and Point3D? Show your design with a class diagram.

1. An education software system must model college students of different kinds: regular full time, part time, undergrads (first-year, sophomore, junior, senior), grads, students who are employees of the school, students from the community (who take enrichment classes), online students*.* There are attributes applicable to students of all kinds such as name, student id, contact info, classes they enrolled in, GPA, etc. There are also aspects that are different depending on a type of student the person is such as whether the person is eligible to live in a dorm, financial aid, how they get paid (as student workers), etc. All information about a student is represented with instance variables and methods (for example, method *getFinancialAids()* would display all information about student’s financial aid (whether or not they receive one).
2. Design a hierarchy of types to be used to represent students of different kinds and express your design in a class diagram. (It’s appropriate to make any additional assumptions from your perspectives as a student)
3. Briefly justify your design.
4. Design and implement a (static) method that computes and returns both maximum “value” and minimum “value” of a list of elements of any object type based on any given comparison criterion. (It’s appropriate to create new types at your discretion.)
5. Consider the following method(with unimportant code omitted, particularly the constructors):

*class OnlineStoreManager {*

*List<Branch> branches;*

*public double computeTotalSales(){*

*double total = 0.0;*

*for(Branch b : branches){*

*List<Order> orders = b.getOrders();*

*for(Order o : orders){*

*total += o.getOrderValue();*

*}*

*}*

*return total;*

*}*

*}*

* 1. This code violates the Law of Demeter, why?
  2. How to correct the problem? (Reimplement the method to demonstrate the improvement.)

1. The following class for hotel management has two methods to compute total bill of a guest as follows:

*class HotelManager*

*//only the relevant code is shown*

*public double getTotalBill(){*

*return chargeFor(“room”) + chargeFor(“meals”) + chargeFor(“movies”)*

*}*

*private double chargeFor(String type){*

*if( type.equals(“room”) ) { /\* ….. compute room charge …. \*/ }*

*else if( type.equals(“meals”) ) { /\* ….. compute meal charges …. \*/ }*

*else if( type.equals(“movies”) ) { /\* ….. compute movie charges …. \*/ }*

*}*

*}*

However, this code is vulnerable to changes when details of a service change (so would billing details), or services are added or removed. It is desirable that total service charge could be computed like:

*for( …. ){ totalBill += listOfServices.get(i).charge(); }*

What does it take to do that (elaborate with all necessary details)? (Alternatively, make your own design to improve the original code.)

1. The following instance method of class *GradeBook* computes the average score of all the assignments of a student, where assignments are weekly homework assignments, projects, presentations, etc., which require different ways to compute scores, thus the implementation of the method *computeScore()* below is polymorphic. Besides, variable *classAssignmentRecords* is an instance of *Map* type.

*public double getAverageScore(int studentId){*

*List<Assignment> assignments = classAssignmentRecords.get(studentId);*

*double total = 0;*

*for(int i = 0; i < assignments.size(); i++ ) total += assignments.get(i).computeScore();*

*return total / assignemnts.size();*

*}*

Based on your understanding of the code above, draw a class diagram that captures the design behind the code. The diagram should include the classes/interfaces with their relations, instance variables and methods identifiable in the code.

1. Assume the following class (only relevant code is shown, however):

*class OnlineOrderProcessor {*

*private Map<String, Order> orders;*

*private PaymentProcessor paymentHandler;*

*private ShippingService delivery;*

*public OnlineOrderProcessor(){*

*paymentHandler = new PaymentProcessor();*

*delivery = new ShippingService();*

*orders = new HashMap<String, Order>();*

*}*

*public boolean* *processOrder(String orderID){*

*Order order = orders.get(orderID);*

*If( order != null ){*

*paymentHandler.processPayment(order);*

*if( order.isExpedited ) delivery.exprShipping(order).*

*else delivery.regShipping(order).*

*return true;*

*}*

*return false;*

*}*

*public void processReturn(String orderID, String kind){*

*Order order = orders.get(orderID);*

*if(order != null){*

*if( kind.equals(“damaged”) ) processNoRefund(order);*

*else if( kind.equals(“past\_return\_window”) ) processPartialRefund(order);*

*else processFullRefund(order);*

*}*

*}*

*} /\*end of class\*/*

* 1. What would be an appropriate level of cohesion for the constructor and each method? (Justify your answers).
  2. For the method *processOrder*, draw (1) a sequence diagram, and (2) an activity diagram. (Readable hand-drawn diagrams are ok.)

1. An electric switch turns a lamp on and off. Suppose type *Switch* simulates operations of an electric switch and type *Lamp* simulates the behavior of a real-world lamp.
   1. Design and implement types *Switch* and *Lamp*.
   2. Write some code to show how an object of Switch would turn a lamp on and off.