CSE2DES/CSE5DES - Assignment Part 2

Due Date: 6 pm Tuesday 13th October 2020

Assessment: This assignment Part 2 is worth 15% of the final mark for CSE2DES/CSE5DES.

This is an individual assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Computer Engineering treats plagiarism very seriously. When it is detected, penalties are strictly imposed. Students are referred to the Department of Computer Science and Computer Engineering's Handbook and policy documents regarding plagiarism.

No extensions will be given: Penalties are applied to late assignments (5% of total assignment mark given is deducted per day, accepted up to 7 days after the due date only). If there are circumstances that prevent the assignment being submitted on time, an application for special consideration may be made. See the departmental Student Handbook for details. Note that delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime.

Assignments submitted more than 7 days late (i.e. after 6 PM on Tuesday 20 October 2020) will receive the mark of 0.

Please note while calculating the penalty for late assignment, Saturday and Sunday are excluded from day count.

Objectives: To learn to represent navigation and the detail required on a design class diagram, specify atomic use cases, and implement and test a prototype of the system.

For Assignment Part 2, you are to continue with the **Standing Order Management System** described in Assignment Part 1. While Assignment Part 1 is concerned with the analysis phase, Assignment Part 2 will be concerned with design, prototyping and testing.

As the starting point for Assignment Part 2, assume that the structural design model, given in Figures 1 and 2, has been adopted. Figure 1 shows the classes of domain objects, their attributes, relationships, and the chosen navigation directions. Figure 2 shows the system class.

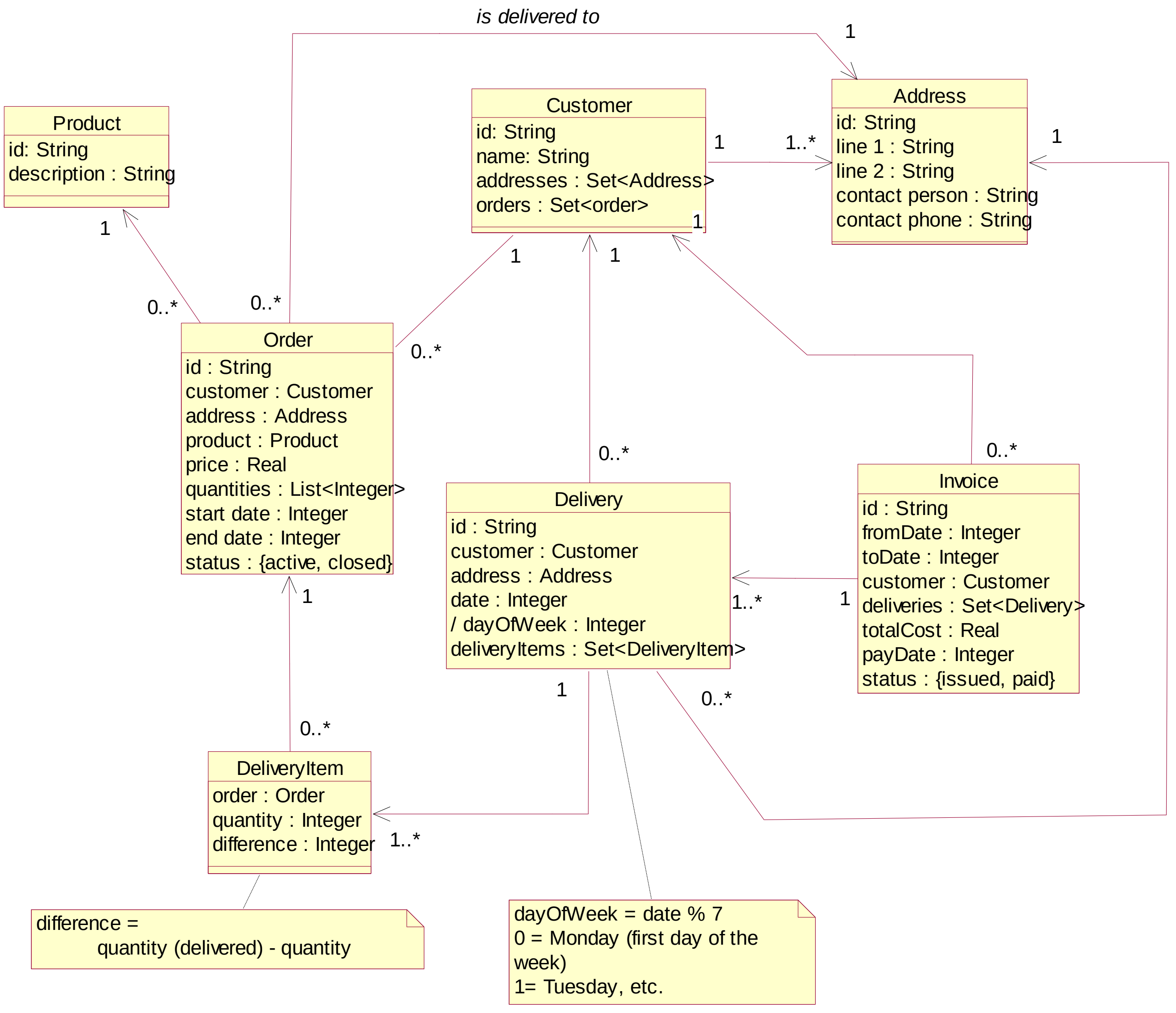

Figure 1 – Design Class Diagram Showing Classes for Domain Objects

NOTE: Though the associations from Delivery to Customer and Address can be derived, they are shown in the diagram. These shown associations serve to express the constraint that each delivery is for one customer to one address only.

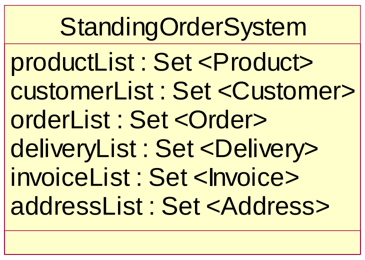
StandingOrderSystem
productList : Set <Product>
customerList : Set <Customer>
orderList : Set <Order>
deliveryList : Set <Delivery>
invoiceList : Set <Invoice>
addressList : Set <Address>

Figure 2 – The System Class – With Attributes Only

In Part 1, the following use cases have been identified as use cases that need to be supported by the system.

1. Add a new product ✓
2. Remove a product - provided there are no standing orders for this product.
3. Add a new customer. At least one delivery address and one standing order need to be added for the new customer. ✓
4. Add a delivery address to an existing customer. ✓
5. Remove a delivery address, provided no delivery has been made to this address.
6. Add a standing order to an existing customer. ✓
7. Close a standing order – when the end-date is reached or when the customer cancels the order.
8. List standing orders that need to be filled for a particular date, sorted by customer name and, within a name, by the customer id. ✓
9. Add a delivery. ✓
10. Generate a delivery docket.
11. List all the customers who have any deliveries for a particular week, given the ending date of the week.
12. Add an invoice for a customer.

    The system is capable of automatically generating all the invoices for deliveries made in a given week. This capability is envisaged as part of the functionality of the final system. However, for the initial development of the system, this capability is put on hold. Instead, as part of the core functionality provided by the system, a simpler use case is considered: it is the use case which adds the details of an invoice (for a customer for deliveries in a particular week) to the system's information base. The invoice number is generated by the system.
13. Generate an invoice document, given the invoice number.
14. Record the payment for an invoice.

We are going to build a prototype. The purpose of building the prototype is to evaluate the design model. As a significant part of the prototype, we choose to consider a small selected set of use cases. Those are the ones that are marked with a tick in the list above.

Selected Use Cases

For convenience, the selected use cases are listed and re-numbered below:

1. Add a new product
2. Add a new customer.

   According to the original requirement, at least one delivery address and one standing order need to be added for the new customer. *For the sake of simplicity, we will relax this condition, and require only that one delivery address is to be added together with a new customer. In other words, the new customer must have a delivery address, but does not (yet) have a standing order.*
3. Add a delivery address to an existing customer.
4. Add a standing order to an existing customer.
5. List standing orders that need to be filled for a particular date, sorted by customer name and, within a name, by the customer id
6. Add a delivery.

   In an operational system, we should generate all the deliveries for a particular data and then, if necessary, adjust the actual quantity delivered. In this prototype, to keep it simple, we simply "mimic" part of this process by entering a delivery manually.

Your Tasks

Task 1 – Atomic Use Case Specifications (30 marks)

Atomic Use Cases

To support the selected use cases, and for the purpose of this prototype, we have identified the following atomic use cases. Some important decisions regarding the atomic use cases are given as well.

1. Add a new product.
2. Add a new customer with one delivery address.
3. Add a delivery address to an existing customer.
4. Add a standing order to an existing customer.

   Date is implemented as an integer. The quantities, as can be seen by the design model, are maintained as a list of integers (which can be implemented in Java as an array or a List). The first number in the list is the quantity for Mondays; the second is that for Tuesdays, and so on. We will also assume that the day of the week for a date can be determined as the remainder obtained when we divide date by 7, with value 0 signifying Monday, 1 Tuesday, and so on.

5. List standing orders that need to be filled for a particular date, sorted by customer name and, within a name, by the customer id
6. Add a delivery.

Your task is to specify the atomic use cases listed above, using the specification language introduced in the course.

Note the following points:

- Your specifications must be based on the given structural design model.
- Your specification must also be based on the *problem description given in Assignment Part1, except where we make explicit changes for the prototype* (e.g. we treat dates as integers)

State any assumption you make.

You are required to submit your answers to Task 1 questions (6 atomic use cases) through the LMS.

Task 2 – Prototyping and Testing the Prototype (60 marks)

You are required to develop and test a prototype of the Standing Order Management System in Java. Your implementation of the prototype must be done in a systematic manner. In particular, for atomic use cases (1 and 2) listed for Task 1, the pre-conditions should be checked first, and the post-conditions should then be satisfied. For atomic use cases (1 and 2), design the test cases and include them in a Java program, called StandingOrderSystemTester, to carry out the testing for both valid and invalid requests. The test program should be appropriately commented so that we can see the purpose of each of the test case.

You must submit your answer to Task 2 through the LMS. You are required to attach the following:

1. A compressed (zip) file containing Java programs for (a) "control" class i.e., StandingOrderSystem; (b) all "domain" classes including Address, Customer, Delivery, DeliveryItem, Invoice, Order and Product; (c) enumerated class, if any; and (d) test program i.e., StandingOrderSystemTester.
2. A MS-Word document containing code listing of your Java programs. In the word document arrange your classes as follows. The first class in the listing is the "control" class StandingOrderSystem, followed by the "domain" classes and enumerated classes (if any) in *alphabetical order* of the class names. Then, the test program StandingOrderSystemTester, followed by the results of the tests.

NOTE:

1. In "control" class i.e., StandingOrderSystem, you are required to implement only atomic use cases 1 and 2 listed in Task 1.

2. In test program i.e., StandingOrderSystemTester, you are required to test only atomic use cases 1 and 2 listed in Task 1. For each test, you are to test for both valid and invalid requests.
3. Even if your prototype is not complete, you still need to provide the test program for the parts that you have completed.

Task 3 – For CSE5DES Students Only (10 marks)

Suppose we want to verify the structural design model, presented in Figure 1 and Figure 2, completely.
(a) Is the set of six use cases that we have selected (see page 4) adequate for this purpose?
(b) If not, which other use cases listed on page 3 would you add to the six use cases that we have selected?
For both parts (a) and (b), you must give reasons to support your answers.

You must submit your answer to Task 3 through the LMS.

Appendix - A Sample Test Program (to illustrate the style only)

The following program illustrates how you should organize your test cases. You should include comments to explain the purpose of each test case.

```
public class StandingOrderSystemTester
{
    static int testCount = 0;
    static String test;

    public static void main(String [] args) throws Exception
    {
        testInit();
        testUC1();
        testUC2();

        // for a complicated use case, such as use case 4, you should consider
        // having more than one test method
    }


    // test creation of new system
    public static void testInit() throws Exception
    {
        test = "TEST UC0: Create new system";
        StandingOrderSystem sos = new StandingOrderSystem();
        System.out.println("\n" + test + "\n" + sos);
    }
    // test add product
    public static void testUC1()throws Exception
    {
        test = "TEST UC1: Add product";

        // create a new system object
        StandingOrderSystem sos = new StandingOrderSystem();

        // then add a product
        sos.addProduct("P1", "Coke");
        System.out.println("\n" + test + "\n" + sos);

        // should have one or two more valid cases

        // then some invalid cases, i.e. preconditions are not satisfied
    }


    // test add customer
    public static void testUC2()throws Exception
    {
        test = "TEST UC2: Add customer";

        // create a system object
        // then add enough data for your testing purpose

        StandingOrderSystem sos = new StandingOrderSystem();
        sos.addProduct("P1", "Coke");

        // then test some valid cases

        int [] quantities = { 20, 20, 20, 20, 20, 10, 0};
        sos.addCustomer("C1", "Smith",
            "A1", "1 Street-1", "Suburb-1","John","1111",
            "ORD1", "P1", "Coke", 1.5, quantities, 1, 100);

        System.out.println("\n" + test + "\n" + sos);

        // then some invalid cases
    }
}
```

■