Middle East Technical University
Northern Cyprus Campus

CNG 443: Intr. to Object-Oriented Programming Languages and Systems
Assignment 2: CyprusDryClean: A Dry-Cleaning Management System

Date handed-out: 8 November 2024, Friday
Date submission due: 22 November 2024, Friday 23:55 (Cyprus time)

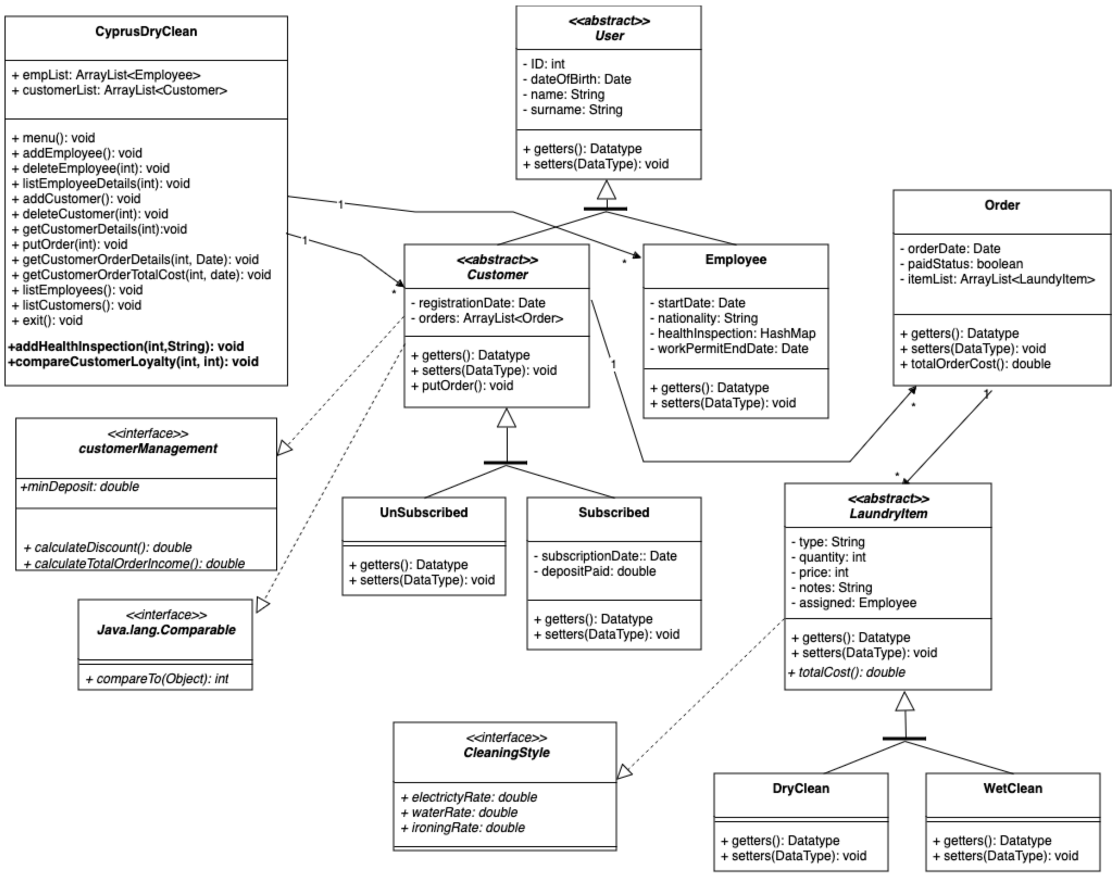
Learning Outcomes

On successful completion of this assignment, a student will:

- Have used an UML class diagram to implement an application.
- Have practiced class hierarchy and the relevant design and implementation decisions.
- Have learnt how to maintain different types of objects.
- Have practiced and used abstract classes, and interfaces.
- Have learnt how to create a package for an application.
- Have also practiced Exception handling in Java.

Requirements

This assignment is about creating a small Java application for a dry-cleaning company called CyprusDryClean. This application will help the company maintain their customers, their orders and also manage their employees and their duties.



CyprusDryClean
+ empList: ArrayList<Employee>
+ customerList: ArrayList<Customer>
+ menu(): void
+ addEmployee(): void
+ deleteEmployee(int): void
+ listEmployeeDetails(int): void
+ addCustomer(): void
+ deleteCustomer(int): void
+ getCustomerDetails(int):void
+ putOrder(int): void
+ getCustomerOrderDetails(int, Date): void
+ getCustomerOrderTotalCost(int, date): void
+ listEmployees(): void
+ listCustomers(): void
+ exit(): void
+addHealthInspection(int,String): void
+compareCustomerLoyalty(int, int): void
<<abstract>> User
- ID: int
- dateOfBirth: Date
- name: String
- surname: String
+ getters(): Datatype
+ setters(DataType): void
<<abstract>> Customer
- registrationDate: Date
- orders: ArrayList<Order>
+ getters(): Datatype
+ setters(DataType): void
+ putOrder(): void
Employee
- startDate: Date
- nationality: String
- healthInspection: HashMap
- workPermitEndDate: Date
+ getters(): Datatype
+ setters(DataType): void
Order
- orderDate: Date
- paidStatus: boolean
- itemList: ArrayList<LaundyItem>
+ getters(): Datatype
+ setters(DataType): void
+ totalOrderCost(): double
<<interface>> customerManagement
+minDeposit: double
+ calculateDiscount(): double
+ calculateTotalOrderIncome(): double
<<interface>> Java.lang.Comparable
+ compareTo(Object): int
UnSubscribed
+ getters(): Datatype
+ setters(DataType): void
Subscribed
- subscriptionDate:: Date
- depositPaid: double
+ getters(): Datatype
+ setters(DataType): void
<<abstract>> LaundryItem
- type: String
- quantity: int
- price: int
- notes: String
- assigned: Employee
+ getters(): Datatype
+ setters(DataType): void
+ totalCost(): double
<<interface>> CleaningStyle
+ electrictyRate: double
+ waterRate: double
+ ironingRate: double
DryClean
+ getters(): Datatype
+ setters(DataType): void
WetClean
+ getters(): Datatype
+ setters(DataType): void
1
1
*
*
1
*
1
*

Figure 1 CyprusDryClean -- Class Diagram

The overall requirements are based on the class diagram above. *Please note that the arrows show the direction of objects that need to be included. For example, a customer object will have references to multiple order objects but not the other way around.* Further requirements are also summarised below:

- The main application called CyprusDryClean will be used to maintain information about employees and also customers. CyprusDryClean will also have the main method and will provide the overall interaction with the application. **The UML diagram given above includes arrows which shows in which way the classes interact**. For example, the CyprusDryClean application which will have the main will maintain employees and customers but not orders. For instance, Order is associated to a customer and laundry items. Therefore, CyprusDryClean class should include the static main method where an instance of this class is constructed and the menu of commands is displayed to the user. Since we have not yet covered Graphical User Interfaces (GUI) in this course, you need to implement it as a command-line application. The required methods are as follows:
    - ***void menu()***: This method will display the interaction menu to the user. The menu should include an option for each of the following functionalities below.
    - ***void addEmployee()***: CyprusDryClean application maintains a list of employees that work in the company. This method will add new employee to this list. Each employee needs to have unique ID number.
    - ***void deleteUser(int empId)***: This method will read an ID number of an employee, and delete the corresponding employee object. If the employee ID number does not exist, the program should provide an appropriate error message.
    - ***void listEmployeeDetails(int empId)***: Given an employee ID number, this method will display the employee details. Please note that you should show the employee ID, name, surname, date of birth and the start date. If the ID number does not exist, the program should provide an appropriate error message.
    - ***void addCustomer()***: This method will add a new customer. Each customer has a unique ID, name, surname, the date of the registration and a list of orders. When a customer is added you need to get the details from the customer and record them. However, regarding the registration date you can directly get the date from the system and no need to manually ask it to the user. **Customers can also subscribe if they like to and they can bring their laundry items regularly for cleaning. Therefore, when you add a customer you need to ask them if they would like to become a subscribed customer then you need to record them as subscribed customer. Otherwise they will be unsubscribed customers. When they become a subscribed customer, they need to pay a deposit and you take this deposit and also record it. minDeposit value is given in the interface customerManagement so that can be used by default but ofcourse customers can also give more. You need to ask the customer and if**

they want to add different deposit then you record what they give (which cannot be less than minDeposit). Unsubscribed customers can also become subscribed customers later therefore we also record the date they become a subscribed customer.

- ***void deleteCustomer(int customerId)***: The main application maintains a list of customers. Given a customer ID, this method will delete the given customer from the system. If the given customer ID does not exist, the program should provide an appropriate error message.
- ***void getCustomerDetails(int customerID)***: Given a customer ID, this method will display the customer details. It should show the customer ID, name, surname, the date of registration and the total number of orders made so far. If the customer ID does not exist, the method should provide an appropriate error message.
- ***void putOrder(int customerId)***: When a customer brings their cloths for cleaning, an order is going to be created. This method will be used to create such order. If the method is invoked with a customer ID that does not exist then the method should give an error, otherwise it should ask all the details about an order. This method should create an order object and also record all the laundry items that the customer brought. An order should record the date of the order and it should also record whether it is paid (i.e., the customer can pay when they order or when they collect), and can have many laundry items as customer can bring many cloths to be cleaned. For example, if a customer brings two shirts and one skirt then the system needs to create two LaundryItem objects. One for shirts and one for skirt. For each laundry item, the system will record the type (e.g., skirt, shirt, etc), quantity (e.g., 2), price per item, notes which can record special notes about the item (e.g., "it is already damaged") and will also assign it to an employee. This assignment needs to be done randomly. The system will assign it randomly to an existing employee. Please note that more than one lanudryItem can be assigned to an employee.

When a laundry item is recorded in the system, it should also be recorded whether it should be wet cleaned or dry cleaned. You need to ask the user if they want wet or dry cleaning for their cloths. Wet and Dry clean total costs will be calculated differently as follows:

- **If it is dry clean then the total cost will be quantity*price*electricityRate. Please note that electricityRate is stored in the cleaningStyle interface.**
- **If it is wet clean then the total cost will be quantity*price*waterRate + ironingRate*quantity. Please note that the waterRate and ironingRate is stored in the CleaningStyle interface.**

Rates can be as follows in your interfaces:

- **electricityRate: 1.05**
- **waterRate: 1.01**

- **ironingRate: 10**

- ***void getCustomerOrderDetails(int customerId, Date orderDate)***: This method will display the orders done by a customer on a given date. A customer can have multiple orders in one day, so it should display all of them. It should display the orderDate, the number of orders and also for each order, whether it was paid or not. For each order, it should have the clots included in the order – it should show the type of laundry and quantity for each type.
- ***void getCustomerOrderTotalCost(int customerId, Date orderDate)***: This method will call to totalOrderCost method on the Order object. The method should first identify the order object for a given customer for a given date. If there are multiple order on a given date by the given customer, then you will need to display the total cost of each of those orders.

  Depending on the customer types, they can also get different discounts. If they are subscribed customers they will receive 10% off and if they are unsubscribed but they have been our customer more than 10 years then they can get %5 discount. Therefore, you need to make use of calculateDiscount on the customer (i.e., comes from the interface) to also give them the discount when you calculate the order total reported. Discount will be applied to the calculated total order cost.

- ***void listEmployees()***: This method will list all the employee details. All employee details should be displayed.
- ***void listCustomers()***: This method will list all the customer details. All customer details should be displayed.
- ***void exit()***: This method should terminate the program.
- ***addHealthInspection(int empID, String)*****: Government requires that employees working in this sector has regular health inspection and the company is responsible for recording the details. Therefore, for each employee will have regular inspections and their reports as a simple String will be recorded. To record the inspections, you will be using a Hashmap. When an inspection is added then you will take the current system date and use it as hash key (There can only be one inspection per day), and you will record the inspection text (i.e., String). That means inspections will be as follows:**
  - **1 January 2024 – “No health issues are observed”;**
  - **10 February 2024 – “Employee has influenza”.**
- ***compareCustomerLoyalty(int customerID, int customerID):*** **This method will take two customer IDs and show which one brings more income. It should use compareTo() method on the customer class which compares customers with calculateTotalOrderIncome() method on each customer. This method will mainly calculate how much income has been**

generated from a customer by calculating the total costs of all their orders. Please note that we ignore the discounts made in this calculation as we want to roughly compare our customers.

- The given class diagram has all the fields and methods needed, so please follow the diagram. If you need extra fields, you can but please make sure that you update your class diagram.
- In this assignment, you need to do exception handling. Please make sure that all the **checked exception types are handled** in your methods.
- In this assignment, you need to organise your code into a package called "dryclean". You can also have sub-packages to organise your code if you like.
- Since you did not learn how to make your class persistent or use a database, you will lose data every time you run your application. Therefore, you need to create some objects before you start your application. Your application needs to start with 4 customer objects (2 subscribed and 2 unsubscribed), 3 employee objects, with each customer having one order and each order having 2 laundaryItem (one dry clean and one wet clean). To create this data, you need to create a class which is called *PopulateData* that can be used to populate your application with these initial data. Please note that this is not given in the UML diagram but it is an extra class that needs to be created. You do not need to write specific constructors for this class.
- Once you complete your implementation, fully update the UML class diagram, in case any changes are needed and submit it as well. Original UML diagram was created with Draw.io. You can use that or any other tool to create your updated UML diagram (e.g. Draw.io (www.draw.io), LucidChart (www.lucidchart.com/), Visio, etc.). This assignment also has an attachment that is the Visio version of this diagram so that you can import it to a tool and edit it.

Environment: As a development environment, you can use any IDE you like but you are strongly recommended to use **IntelliJ** (https://www.jetbrains.com/idea/).

Submission: Please organise your submission as a *single ZIP file* that includes the following:

- **[Jar file]**: A JAR file that can be executed on a command line. Mark sure that CyprusDryClean is the main class.
- **[doc folder]:** This should include the full Javadocs generated.
- **[source folder]:** This should include your full source code. Please note that if you do not include *.java files, we cannot grade your work and you will automatically receive **zero**.
- [**diagram folder**]: This should include the updated UML in PNG format.

If you are not following this submission structure, you will not receive marks from the Package item in the grading policy.

Extra Requirements:

Some additional requirements are listed below:

- We have not yet covered how to use a Database or make objects persistent in this course. Therefore, this assignment maintains objects such as employee and customers in arrayLists.
- We have not yet covered Graphical User Interfaces (GUI) in this course. So please provide a command-line interaction (CLI).
- For each class, please decide what kind of constructors are required, the access types of methods and fields. If you use private fields, make sure that you provide accessor and mutators. For each class, you need to do constructor overloading and provide at least **two constructors**. Please note that the default constructor does not count so your code should include default plus two different versions.
- You can use the Date class provided in java.utils in order to read the date from the user, read a string with the "dd/mm/yyyy" format, in which dd, mm and yyyy represent the day, month and the year, respectively. Study the "Parsing Strings into Dates" section provided in:
    - https://www.tutorialspoint.com/java/java_date_time.htm

  In this assignment, you are allowed to use alternative Date classes.
- Pay attention to the overall design, layout and presentation of your code.
- You need to submit your Java code with proper JavaDoc comments. For each class, you need to have used at least @author and @version, and for each method, you need at least @param and @return (no need to use them for getters and setters).

Assessment Criteria

This assignment will be marked as follows:

Aspect	**Marks (Total 100)**
All classes are implemented	10
All class hierarchies are implemented	10
All interfaces are implemented and used	10
For all classes constructors are properly implemented	10
For all classes all required data fields are implemented	10
For all classes all required methods are implemented	10
All methods in the main application are implemented	30
Package Structure and Jar for Invoking the application	5
Exception Handling is done	5

For each of the items above, we will also use the following grading criteria:

Half working	%20
Fully working	%20
Appropriate reuse of other code	%10
Good Javadoc comments	%10
Good quality code[1]	%40

[1] 15 principles will be considered during grading: https://www.informit.com/articles/article.aspx?p=2223710