Building an Java Console Application with File

Requirements

Create a java console applicationtion to manage the employee and customer infomations includes functions such as: create, find , search, remove, update, and export data to the file...

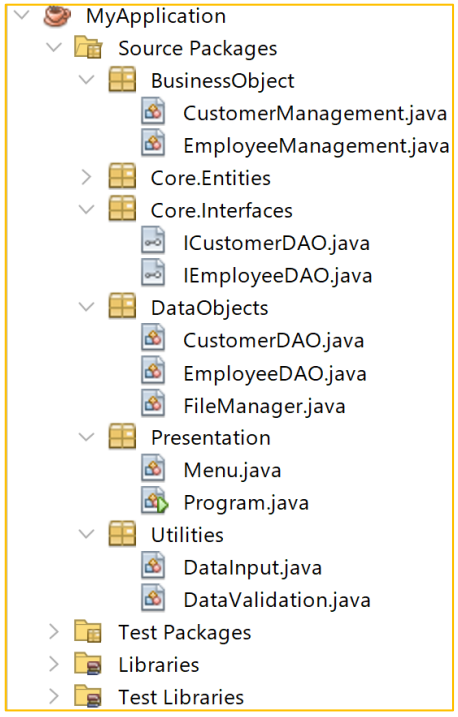
```
***************Main Menu***************
1.Employee Management
2.Customer Management
3.Exit
Select:
1
******Employee Management******
1.Add Employee
2.Update Employee
3.Remove Employee
4.Search Employees
5.Print Employee List
6.Export to file
7.Back to main menu
Select :
5
>>Employee List:
Employee Id | Employee Name        | DateOfBirth          | Email               | Salary
-------------------------------------------------------------------------------------------
E001        | Tran Gia Hong        | 01/01/2003           |hong@gmail.com       |1500.00
E002        | Do Anh Khoa          | 13/04/2002           |khoa@hotmail.com     |2500.00
E003        | Dinh Thu Thuy        | 21/08/2001           |thuy@yahoo.com       |2000.00
E004        | Tran Van An          | 29/12/2002           |antv@live.com        |2891.00
E005        | Pham Hoang Anh       | 12/12/2001           |anhdh@gmail.com      |1000.00
E006        | Tran Gia Hong        | 12/12/2001           |hong@gmail.com       |1000.00
E007        | Dinh The Luan        | 01/01/2002           |luan@gamail.com      |8300.00
-------------------------------------------------------------------------------------------
******Employee Management******
1.Add Employee
2.Update Employee
3.Remove Employee
4.Search Employees
5.Print Employee List
6.Export to file
7.Back to main menu
Select :
```

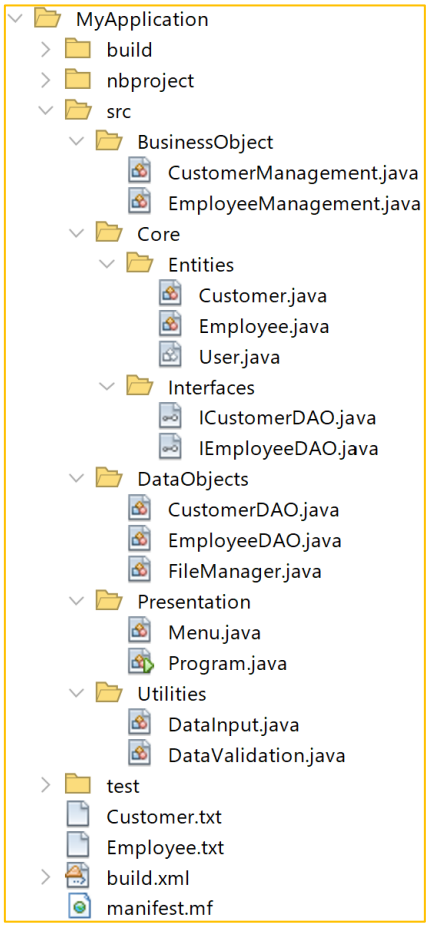
Lab Objectives

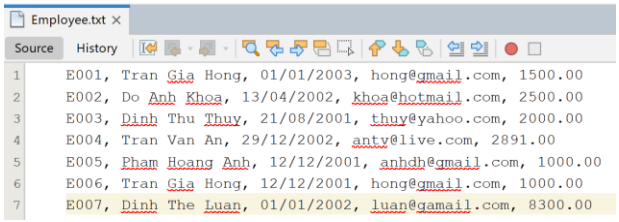
In this lab, you will be:

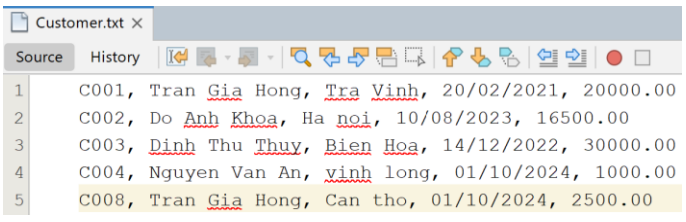
- Create Java console project using the NetBean IDE
- Create the classes and interfaces to perform the functions
- Apply the Java Dependency Injection in the project
- Apply the layered architecture to develop the project
- Run the project and test the functions of the applicaiton

Step 01. Create a Java Web application named **MyApplication** is structured as follows:

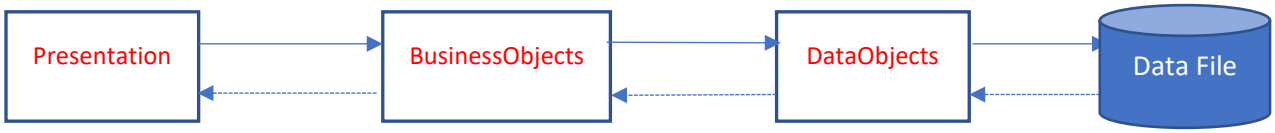








Design Architecture



Step 02. Write codes for the classes in the **Utilities** package as the follows:

- **DataValidation.java**

DataValidation.java

```
Source  History
1  package Utilities;
2  /**
3   * @author SwordLake
4   */
5  public final class DataValidation {
6      //---------------------------------------------------
7      public static boolean checkNumberInMinMax(int number,int min, int max){
8          boolean result = true;
9          if(number < min || number > max){
10             result = false;
11         }
12         return result;
13     }
14     //---------------------------------------------------
15     public static boolean checkStringEmpty(String value) {
16         boolean result = true;
17         if(value.isEmpty()){
18             result = false;
19         }
20         return result;
21     }
22     //---------------------------------------------------
23     public static boolean checkStringLengthInRange(String value, int min, int max) {
24         boolean result = true;
25         int length;
26         if (!checkStringEmpty(value)) {
27             result = false;
28         } else {
29             length = value.length();
30             if (length < min || length > max) {
31                 result = false;
32             }
33         }
34         return result;
35     }
36     //---------------------------------------------------
37     public static boolean checkStringWithFormat(String value,String pattern){
38         boolean result = false;
39         if(value.matches(pattern)){
40             result = true;
41         }
42         return result;
43     }
44     //---------------------------------------------------
45     //More the methods here.........
46 }
```

- **DataInput.java**

DataInput.java

```java
package Utilities;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;
/**
 * @author SwordLake
 */
public class DataInput {
    //-----------------------------------------------
    public static int getIntegerNumber(String displayMessage)
            throws Exception {
        int number = 0;
        System.out.print(displayMessage);
        number = getIntegerNumber();
        return number;
    }
    //-----------------------------------------------
    public static int getIntegerNumber() throws Exception {
        int number = 0;
        String strInput;
        strInput = getString();
        //if (s.matches("\\d{1,10}") == false) {
        if (!DataValidation.checkStringWithFormat(strInput, "\\d{1,10}")) {
            throw new Exception("Data invalid.");
        } else {
            number = Integer.parseInt(strInput);
        }
        return number;
    }
    //-----------------------------------------------
    public static double getDoubleNumber(String displayMessage) throws Exception {
        double number = 0;
        String strInput = getString(displayMessage);
        if (DataValidation.checkStringEmpty(strInput)) {
            if (!DataValidation.checkStringWithFormat(strInput, "[0-9]+[.]?[0-9]+")) {
                throw new Exception("Data invalid.");
            } else {
                number = Double.parseDouble(strInput);
            }
        }
        return number;
    }
```

```
44          //-------------------------------------------------
45          public static String getString(String displayMessage) {
46              String strInput;
47              System.out.print(displayMessage);
48              strInput = getString();
49              return strInput;
50          }
51          //-------------------------------------------------
52          public static String getString() {
53              String strInput;
54              Scanner sc = new Scanner(System.in);
55              strInput = sc.nextLine();
56              return strInput;
57          }
58          //-------------------------------------------------
59          public static LocalDate getDate(String displayMessage) throws Exception {
60              String strInput;
61              LocalDate date = null;
62              strInput = getString(displayMessage);
63              if (DataValidation.checkStringEmpty(strInput)) {
64                  try {
65                      date = LocalDate.parse(strInput, DateTimeFormatter.ofPattern("dd/MM/yyyy"));
66                  } catch (Exception ex) {
67                      throw new Exception("Date invalid.");
68                  }
69              }
70              return date;
71          }
72          //-------------------------------------------------
73          //To do here.....
74      }
```

Step 03. Write codes for the classes in the **Core.Entities** package as the follows:

- **User.java**

User.java ×

Source | History

```
1   package Core.Entities;
2
3   import Utilities.DataValidation;
4
    public abstract class User {
6       //Fields
7       protected String id;
8       protected String name;
9       public User() {
10          id="U000";
11          name="New User";
12      }
13      public User(String id, String name) {
14          this.id = id;
15          this.name = name;
16      }
```

```
17      public String getId() {
18          return id.toUpperCase();
19      }
        public void setId(String id) throws Exception{
21          this.id = id;
22      }
23      public String getName() {
24          return toTiteCase(name);
25      }
26      public void setName(String name) throws Exception{
27          if(!DataValidation.checkStringWithFormat(name,"[A-Za-z|\\s]{3,50}")){
28              throw new Exception("Name must be from 3 to 50 characters.");
29          }
30          this.name = toTiteCase(name);
31      }
32      public String toTiteCase(String value) {
33          String s = "";
34          value = value.trim().replaceAll("\\s+", " ").toLowerCase();
35          String[] words = value.split(" ");
36          for (String word : words) {
37              char[] arr = word.toCharArray();
38              arr[0] = Character.toUpperCase(arr[0]);
39              s = s + new String(arr) + " ";
40          }
41          return s.trim();
42      }
43      @Override
        public String toString() {
45          return String.format("%s, %s", id, name);
46      }
47  }
```

- **Employee.java**

Employee.java ×

Source | History

```
1   package Core.Entities;
2
3   import Utilities.DataValidation;
4   import java.time.LocalDate;
5   import java.time.format.DateTimeFormatter;
6
7
8   public class Employee extends User {
9       private LocalDate dateOfBirth;
10      private double salary;
11      private String email;
```

```
12          //Constructors
13          public Employee(String id, String name,
14                  LocalDate dateOfBirth,String email,double salary) throws Exception {
15              super(id, name);
                setId(id);
                setEmail(email);
18              setName(name);
                setDateOfBirth(dateOfBirth);
                setSalary(salary);
21          }
22          @Override
            public void setId(String value) throws Exception{
24              if(!DataValidation.checkStringWithFormat(value.toUpperCase(),"E\\d{3}")){
25                  throw new Exception("Id is invalid. The correct format:Exxx, with x is digits");
26              }
27              this.id = value;
28          }
29          public String getEmail() {
30              return email;
31          }
32
33          public void setEmail(String email)  throws Exception {
34              if(!DataValidation.checkStringWithFormat(email,"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]
35                  throw new Exception("Email is invalid.");
36              }
37              this.email = email;
38          }
39
40          public LocalDate getDateOfBirth() {
41              return dateOfBirth;
42          }
```

```
44          public void setDateOfBirth(LocalDate dateOfBirth) throws Exception {
45              if(dateOfBirth == null){
46                  throw new Exception("DateOfBirth is invalid.");
47              }
48              this.dateOfBirth = dateOfBirth;
49          }
50
51          public double getSalary() {
52              return salary;
53          }
54
55          public void setSalary(double salary) {
56              this.salary = salary;
57          }
58
59          //Methods....
60           @Override
            public String toString(){
62              DateTimeFormatter formatters = DateTimeFormatter.ofPattern("dd/MM/yyyy");
63              return String.format("%s, %s, %s, %s, %.2f",getId(),
64                      getName(), dateOfBirth.format(formatters),email,salary);
65          }
66      }
```

Hints: Regex for checking the email as follows:

^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$

- **Customer.java**

Customer.java

Source | History

```java
package Core.Entities;

import Utilities.DataValidation;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Customer extends User {
    private String address;
    private LocalDate lastOrderDate;
    private double amount;
    //Constructors
    public Customer(String id, String name, String address,
            LocalDate lastOrderDate, double amount) throws Exception {
        super(id, name);
        setId(id);
        setName(name);
        setAddress(address);
        setLastOrderDate(lastOrderDate);
        setAmount(amount);
    }
    //-------------------------------------------------
    @Override
    public void setId(String value) throws Exception {
        if (!DataValidation.checkStringWithFormat(value.toUpperCase(), "C\\d{3}")) {
            throw new Exception("Id invalid. The correct format:Cxxx, with x is digits");
        }
        this.id = value;
    }
    //-------------------------------------------------
    public String getAddress() {
        return address;
    }
    //-------------------------------------------------
    public void setAddress(String address) {
        this.address = address;
    }
    //-------------------------------------------------
    public LocalDate getLastOrderDate() {
        return lastOrderDate;
    }
    //-------------------------------------------------
    public void setLastOrderDate(LocalDate lastOrderDate) throws Exception{
        if(lastOrderDate == null){
            throw new Exception("LastOrderDate is invalid.");
        }
        this.lastOrderDate = lastOrderDate;
    }
    //-------------------------------------------------
    public void setAmount(double amount) {
        this.amount = amount;
    }
```

```
52          //--------------------------------------------------
53          public double getAmount() {
54              return amount;
55          }
56          //--------------------------------------------------
57          @Override
            public String toString() {
59              DateTimeFormatter formatters = DateTimeFormatter.ofPattern("dd/MM/yyyy");
60              return String.format("%s, %s, %s, %s, %.2f",getId(),
61                      getName(),getAddress(), lastOrderDate.format(formatters),amount);
62          }
63      }
```

Step 03. Write codes for the interfaces of the **Core.Interfaces** package as follows:

- **IEmployeeDAO.java :**

IEmployeeDAO.java

Source | History

```
1   package Core.Interfaces;
2
3   import Core.Entities.Employee;
4   import java.util.List;
5   import java.util.function.Predicate;
6
7   /**
8    * @author SwordLake
9    */
    public interface IEmployeeDAO{
        List<Employee>  getEmployees() throws Exception ;
        Employee  getEmployeeById(String id) throws Exception ;
        void addEmployee(Employee employee) throws Exception ;
        void updateEmployee(Employee employee) throws Exception ;
        void removeEmployee(Employee employee) throws Exception ;
        void saveEmployeeListToFile() throws Exception ;
        List<Employee> search(Predicate<Employee> predicate) throws Exception ;
18      //--------------------------------------------------
19      //More the method here.........
20  }
```

- **ICustomerDAO.java :**

ICustomerDAO.java

```java
package Core.Interfaces;
/**
 * @author Sword Lake
 */
import Core.Entities.Customer;
import java.util.List;
import java.util.function.Predicate;

public interface ICustomerDAO{
    List<Customer> getCustomers() throws Exception ;
    Customer  getCustomerById(String id) throws Exception ;
    void addCustomer(Customer customer) throws Exception ;
    void updateCustomer(Customer customer) throws Exception ;
    void removeCustomer(Customer customer) throws Exception ;
    void saveCustomerListToFile() throws Exception ;
    List<Customer> search(Predicate<Customer> predicate) throws Exception ;
    //More the mehthods here
}
```

Step 04. Write codes for the interfaces of the **Core.Interfaces** package as follows:

- **FileManager.java :**

FileManager.java

```java
package DataObjects;

import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.util.List;

/**
 * @author SwordLake
 */
public class FileManager{
    private String fileName;

    public FileManager() {
    }
    //-------------------------------------------------
    public FileManager(String fileName) {
        this.fileName = fileName;
    }
```

```
21      //-----------------------------------------------------
22      public List<String> readDataFromFile() throws IOException {
23          List<String> result;
24          result = Files.readAllLines(new File(fileName).toPath(),
25                  Charset.forName("utf-8"));
26          return result;
27      }
28      //-----------------------------------------------------
29      public void saveDataToFile(String data) throws IOException {
30          Files.writeString(new File(fileName).toPath(), data,
31                  Charset.forName("utf-8"));
32      }
33      //-----------------------------------------------------
34      //More the methods here
35  }
```

- **EmployeeDAO.java :**

EmployeeDAO.java ×

Source | History

```
1   package DataObjects;
2   import Core.Entities.Employee;
3   import Core.Interfaces.IEmployeeDAO;
4   import java.time.LocalDate;
5   import java.time.format.DateTimeFormatter;
6   import java.util.ArrayList;
7   import java.util.Arrays;
8   import java.util.Collections;
9   import java.util.List;
10  import java.util.function.Predicate;
11  import java.util.stream.Collectors;
12
13  public final class EmployeeDAO implements IEmployeeDAO {
14      private final List<Employee> employeeList = new ArrayList();
15      private final  FileManager fileManager;
16      public EmployeeDAO(String fileName)throws Exception {
17          this.fileManager = new FileManager(fileName);
18          loadDataFromFile();
19      }
```

```
20      //-----------------------------------------------
21      public  void loadDataFromFile() throws Exception {
22          String id, name, email;
23          LocalDate dateOfBirth;
24          double salary;
25          try {
26              employeeList.clear();
27              List<String> empData = fileManager.readDataFromFile();
28              for (String e : empData) {
29                  List<String> emp = Arrays.asList(e.split(","));
30                  id = emp.get(0).trim();
31                  name = emp.get(1).trim();
32                  dateOfBirth = LocalDate.parse(emp.get(2).trim(),
33                          DateTimeFormatter.ofPattern("dd/MM/yyyy"));
34                  email = emp.get(3).trim();
35                  salary = Double.parseDouble(emp.get(4).trim());
36                  Employee newEmp = new Employee(id, name, dateOfBirth, email, salary);
37                  employeeList.add(newEmp);
38                  if (employeeList.isEmpty()) {
39                      throw new Exception("Employee list is empty.");
40                  }
41              }
42          } catch (Exception ex) {
43              throw new Exception("Can not read data from file.Please check file again.");
44          }
45      }
```

```
46      //-----------------------------------------------
47      @Override
48      public  List<Employee> getEmployees() throws Exception {
49          Collections.sort(employeeList,(e1,e2)->e1.getId().compareTo(e2.getId()));
50          return employeeList;
51      }
52      //-----------------------------------------------
53      @Override
54      public void addEmployee(Employee employee) throws Exception  {
55          employeeList.add(employee);
56      }
57      //-----------------------------------------------
58      @Override
59      public void updateEmployee(Employee employee) throws Exception {
60           Employee emp  = getEmployeeById(employee.getId());
61           if(emp!= null){
62               emp.setName(employee.getName());
63               emp.setDateOfBirth(employee.getDateOfBirth());
64               emp.setEmail(employee.getEmail());
65               emp.setSalary(employee.getSalary());
66           }
67      }
68      //-----------------------------------------------
```

```
69      @Override
        public void removeEmployee(Employee employee) throws Exception {
71          Employee emp  = getEmployeeById(employee.getId());
72          if(emp!= null){
73              employeeList.remove(emp);
74          }
75      }
76       //-----------------------------------------------------
77      @Override
        public Employee getEmployeeById(String id) throws Exception {
79          if(employeeList.isEmpty()){
80              getEmployees();
81          }
82          Employee employee = employeeList.stream()
83                  .filter(e->e.getId()
84                  .equalsIgnoreCase(id)).findAny().orElse(null);
85          return employee;
86      }
87       //-----------------------------------------------------
88      @Override
        public List<Employee> search(Predicate<Employee> predicate) throws Exception {
90          return  employeeList.stream().
91                  filter(employee->predicate.test(employee)).toList();
92      }
93      //-----------------------------------------------------
```

```
95      @Override
        public void saveEmployeeListToFile() throws Exception {
            List<String> stringObjects = employeeList.stream()
                    .map(String::valueOf).collect(Collectors.toList());
99          String data = String.join("\n", stringObjects);
100         fileManager.saveDataToFile(data);
101     }
102     //-------------------------------------------------------------
103     //More the methods here.........
```

- **CustomerDAO.java :**

CustomerDAO.java

Source | History

```
1   package DataObjects;
2
3   import Core.Interfaces.ICustomerDAO;
4   import Core.Entities.Customer;
5   import java.time.LocalDate;
6   import java.time.format.DateTimeFormatter;
7   import java.util.ArrayList;
8   import java.util.Arrays;
9   import java.util.Collections;
10  import java.util.List;
11  import java.util.function.Predicate;
12  import java.util.stream.Collectors;
```

```
14  public final class CustomerDAO implements ICustomerDAO {
        private final List<Customer> customerList = new ArrayList<>();
        private final  FileManager fileManager;
17      public CustomerDAO(String fileName) throws Exception{
18          this.fileManager = new FileManager(fileName);
19          loadDataFromFile();
20      }
21      //-------------------------------------------------
22      public  void loadDataFromFile() throws Exception {
23          //Write codes to load customer list into the customerList
24          throw new UnsupportedOperationException("Not supported yet.");
25      }
26      //-------------------------------------------------
27      @Override
        public  List<Customer> getCustomers() throws Exception {
29          //Write codes to return the customerList sorted in ascending order by Id
30          throw new UnsupportedOperationException("Not supported yet.");
31      }
32      //-------------------------------------------------
33      @Override
        public Customer getCustomerById(String id) throws Exception {
35         //Write codes to get the customer by Id
36         throw new UnsupportedOperationException("Not supported yet.");
37      }
38      //-------------------------------------------------
39      @Override
        public void addCustomer(Customer customer) throws Exception {
41          //Write codes to add  the customer to customerList
42          throw new UnsupportedOperationException("Not supported yet.");
43      }
44      //-------------------------------------------------
45      @Override
        public void updateCustomer(Customer customer) throws Exception {
47          //Write codes to update  the customer of the customerList
48          throw new UnsupportedOperationException("Not supported yet.");
49      }
50      //-------------------------------------------------
51      @Override
        public void removeCustomer(Customer customer) throws Exception {
53          //Write codes to remove the customer of the customerList
54          throw new UnsupportedOperationException("Not supported yet.");
55      }
56      //-------------------------------------------------
57      @Override
        public void saveCustomerListToFile() throws Exception {
59          //Write codes to save the customerList to file
60          throw new UnsupportedOperationException("Not supported yet.");
61      }
62      //-------------------------------------------------
```

```
63      @Override
64      public List<Customer> search(Predicate<Customer> predicate) throws Exception {
65          //Write codes to search the customer by predicate
66          throw new UnsupportedOperationException("Not supported yet.");
67      }
68      //-------------------------------------------------
69      //Write more the methods here
70  }
```

Step 05. Write codes for the classes of the **BusinessObject** package as follows:

- **EmployeeManagement.java:**

EmployeeManagement.java

Source | History

```
1   package BusinessObject;
2
3   import Utilities.DataInput;
4   import Core.Entities.Employee;
5   import java.util.Collections;
6   import java.util.List;
7   import java.util.function.Predicate;
8   import Core.Interfaces.IEmployeeDAO;
9   import Presentation.Menu;
10  import java.time.LocalDate;
11  import java.time.format.DateTimeFormatter;
12  /**
13   * @author SwordLake
14   */
15  public class EmployeeManagement {
16      IEmployeeDAO employeeDAO;
17      //-------------------------------------------------
18      //Inject the EmployeeDAO object by IEmployeeDAO interface
19      public EmployeeManagement(IEmployeeDAO employeeDAO) {
20          this.employeeDAO = employeeDAO;
21      }
22      //-------------------------------------------------
```

```
23      public void processMenuForEmployee() {
24          boolean stop = true;
25          try {
26              do {
27                  Menu.print("******Employee Management******|1.Add Employee|"
28                          + "2.Update Employee|3.Remove Employee"
29                          + "|4.Search Employees|5.Print Employee List|"
30                          + "6.Export to file|7.Back to main menu|Select :");
31                  int choice = Menu.getUserChoice();
32                  switch (choice) {
33                      case 1 -> {
34                          addNewEmployee();
35                      }
36                      case 2 -> {
37                          updateEmployee();
38                      }
39                      case 3 -> {
40                          deleteEmployee();
41                      }
42                      case 4 -> {
43                          searchEmployees();
44                      }
45                      case 5 -> {
46                          System.out.println(">>Employee List:");
47                          printList(employeeDAO.getEmployees());
48                      }
49                      case 6 -> {
50                          exportToFile();
51                          System.out.println(">>The employee list has "
                                    + "successfully exported.");
53                      }
54                      case 7 ->
55                          stop = false;
56                      default ->
57                          System.out.println(">>Choice invalid");
58                  }
59              } while (stop);
60          } catch (Exception e) {
61              System.out.println(e.getMessage());
62          }
63      }
64
65      //-------------------------------------------------
```

```
65      //-----------------------------------------------------
66      public Employee inputEmployee() throws Exception {
67          String id = DataInput.getString("Enter the id:");
68          String name = DataInput.getString("Enter the name:");
69          LocalDate dateOfBirth = DataInput.getDate("Enter the date of birth:");
70          String email = DataInput.getString("Enter the email:");
71          double salary = DataInput.getDoubleNumber("Enter the salary:");
72          return new Employee(id, name, dateOfBirth, email, salary);
73      }
74      //-----------------------------------------------------
75      public void setNewEmployeeInfo(Employee employee) throws Exception {
76          String name = DataInput.getString("Enter new name:");
77          if (!name.isEmpty()) {
78              employee.setName(name);
79          }
80          LocalDate dateOfBirth = DataInput.getDate("Enter new date of birth:");
81          if (dateOfBirth != null) {
82              employee.setDateOfBirth(dateOfBirth);
83          }
84          String email = DataInput.getString("Enter new email:");
85          if (!email.isEmpty()) {
86              employee.setEmail(email);
87          }
88          double salary = DataInput.getDoubleNumber("Enter new salary:");
89          if (salary > 0) {
90              employee.setSalary(salary);
91          }
92      }
93      //-----------------------------------------------------
```

```
94      public void addNewEmployee() {
95          try {
96              Employee employee = inputEmployee();
97              if (employeeDAO.getEmployeeById(employee.getId()) != null) {
98                  System.out.println(">>The employee already exists.");
99                  return;
100             }
101             employeeDAO.addEmployee(employee);
102             System.out.println(">>The employee has added successfully.");
103         } catch (Exception e) {
104             System.out.println(">>Error:" + e.getMessage());
105         }
106     }
107     //-----------------------------------------------------
```

```
108     public void updateEmployee() {
109         try {
110             String id = DataInput.getString("Enter employee id:");
111             Employee employee = employeeDAO.getEmployeeById(id);
112             if (employee == null) {
113                 System.out.println(">>The employee not found.");
114                 return;
115             }
116             setNewEmployeeInfo(employee);
117             employeeDAO.updateEmployee(employee);
118             System.out.println(">>The employee has updated successfully.");
119 
120         } catch (Exception e) {
121             System.out.println(">>Error:" + e.getMessage());
122         }
123     }
124     //-------------------------------------------------
125     public void deleteEmployee() {
126         try {
127             String id = DataInput.getString("Enter employee id:");
128             Employee employee = employeeDAO.getEmployeeById(id);
129             if (employee == null) {
130                 System.out.println(">>The employee not found.");
131                 return;
132             }
133             employeeDAO.removeEmployee(employee);
134             System.out.println(">>The employee has deleted successfully.");
135         } catch (Exception e) {
136             System.out.println(">>Error:" + e.getMessage());
137         }
138     }
```

```
139     //-------------------------------------------------
140     public void findEmployeeById() {
141         Employee employee;
142         try {
143             String id = DataInput.getString("Enter employee id:");
144             employee = employeeDAO.getEmployeeById(id);
145             if (employee != null) {
146                 System.out.println(employee);
147             } else {
148                 System.out.format("The employee id : %s not found.%n", id);
149             }
150         } catch (Exception e) {
151             System.out.println(">>Error:" + e.getMessage());
152         }
153     }
```

```
154     //-----------------------------------------------------
155     public void searchEmployees() throws Exception {
156         int choice;
157         boolean stop = true;
158         String value;
159         try {
160             do {
161                 Menu.print("1.Search by name|2.Search by salary|3.Back|Select:");
162                 choice = DataInput.getIntegerNumber();
163                 switch (choice) {
164                     case 1 -> {
165                         value = DataInput.getString("Enter the name:");
166                         List<Employee> employees = searchEmployeeByName(value);
167                         if (!employees.isEmpty()) {
168                             printList(employees);
169                         } else {
170                             System.out.format("The employees with name:%s "
171                                     + "not found.%n",value);
172                         }
173                     }
174                     case 2 -> {
175                         double salary = DataInput.getDoubleNumber("Enter the salary:");
176                         List<Employee> employees = searchEmployeeBySalary(salary);
177                         if (!employees.isEmpty()) {
178                             printList(employees);
179                         } else {
180                             System.out.format("The employees with salary "
181                                     + "are greater than or equal to :%s not found.%n",
182                                     String.valueOf(salary));
183                         }
184                     }
185                     case 3 -> {
186                         stop = false;
187                     }
188                     default -> {
189                         System.out.println(">>Choice invalid");
190                     }
191                 }
192             } while (stop);
193         } catch (Exception e) {
194             System.out.println(">>Error:" + e.getMessage());
195         }
196     }
197     //-----------------------------------------------------
198     public void printList(List<Employee> employees) throws Exception {
199         DateTimeFormatter formatters = DateTimeFormatter.ofPattern("dd/MM/yyyy");
200         System.out.format("%-10s | %-20s | %-20s | %-19s | %s%n",
201                 "Employee Id", "Employee Name", "DateOfBirth",
202                 "Email", "Salary");
203         System.out.println(String.join("", Collections.nCopies(95, "-")));
204         for (Employee employee : employees) {
205             System.out.format("%-11s | %-20s | %-20s |%-20s |%.2f%n",
206                     employee.getId(), employee.getName(),
207                     employee.getDateOfBirth().format(formatters),
208                     employee.getEmail(), employee.getSalary());
209         }
210         System.out.println(String.join("", Collections.nCopies(95, "-")));
211     }
212     //-----------------------------------------------------
```

```
213     public void exportToFile() throws Exception {
214         employeeDAO.saveEmployeeListToFile();
215     }
216     //-------------------------------------------------
217     public List<Employee> searchEmployeeByName(String value) throws Exception {
218         Predicate<Employee> predicate = p -> p.getName().toLowerCase().
219                 contains(value.toLowerCase());
220          List<Employee> employees = employeeDAO.search(predicate);
221         return employees;
222     }
223     //-------------------------------------------------
224     public List<Employee> searchEmployeeBySalary(double value) throws Exception {
225         Predicate<Employee> predicate = (e -> e.getSalary() >= value);
226         List<Employee> employees = employeeDAO.search(predicate);
227         return employees;
228     }
229     //-------------------------------------------------
230     //More the methods here.........
231 }
```

- **CustomerManagement.java:**

CustomerManagement.java ×

Source | History

```
1   package BusinessObject;
2
3   import Utilities.DataInput;
4   import Core.Entities.Customer;
5   import java.util.Collections;
6   import java.util.List;
7   import java.util.function.Predicate;
8   import Core.Interfaces.ICustomerDAO;
9   import Presentation.Menu;
10  import java.time.LocalDate;
11  import java.time.format.DateTimeFormatter;
12  /**
13   * @author SwordLake
14   */
15  public class CustomerManagement {
16      ICustomerDAO customerDAO;
17      //-------------------------------------------------
18      public CustomerManagement(ICustomerDAO customerDAO) {
19          this.customerDAO = customerDAO;
20      }
21      //-------------------------------------------------
```

```
22      public void processMenuForCustomer() {
23          boolean stop = true;
24          try {
25              do {
26                  Menu.print("******Customer Management******|"
27                          + "1.Add Customer|2.Update Customer|"
28                          + "3.Remove Customer|4.Search Customers|"
29                          + "5.Print Customer List|6.Export to file|"
30                          + "7.Back to main menu|Select :");
31                  int choice = Menu.getUserChoice();
32                  switch (choice) {
33                      case 1 -> {
34                          addNewCustomer();
35                      }
36                      case 2 -> {
37                          updateCustomer();
38                      }
39                      case 3 -> {
40                          deleteCustomer();
41                      }
42                      case 4 -> {
43                          searchCustomers();
44                      }
45                      case 5 -> {
46                          System.out.println(">>Customer List:");
47                          printList(customerDAO.getCustomers());
48                      }
49                      case 6 -> {
50                          exportToFile();
51                          System.out.println(">>The customer list "
52                                  + "has successfully exported.");
53                      }
54                      case 7 ->
55                          stop = false;
56                      default ->
57                          System.out.println(">>Choice invalid");
58                  }
59              } while (stop);
60          } catch (Exception e) {
61              System.out.println(e.getMessage());
62          }
63      }
64      //-------------------------------------------------
```

```
65      public Customer inputCustomer() throws Exception {
66          String id = DataInput.getString("Enter the id:");
67          String name = DataInput.getString("Enter the name:");
68          String address = DataInput.getString("Enter the address:");
69          LocalDate lastOrderDate = DataInput.getDate("Enten the last order date:");
70          double amount = DataInput.getDoubleNumber("Enter the amount:");
71          return new Customer(id, name, address, lastOrderDate, amount);
72      }
73
74      //-------------------------------------------------
75      public void setNewCustomerInfo(Customer customer) throws Exception {
76          String name = DataInput.getString("Enter new name:");
77          if (!name.isEmpty()) {
78              customer.setName(name);
79          }
80          String address = DataInput.getString("Enter new address:");
81          if (!address.isEmpty()) {
82              customer.setAddress(address);
83          }
84          LocalDate lastOrderDate = DataInput.getDate("Enter new last order date:");
85          if (lastOrderDate != null) {
86              customer.setLastOrderDate(lastOrderDate);
87          }
88          double amount = DataInput.getDoubleNumber("Enter new amount:");
89          if (amount > 0) {
90              customer.setAmount(amount);
91          }
92      }
93      //-------------------------------------------------
94      public void addNewCustomer() {
95          //Write codes to add the new employee
96          throw new UnsupportedOperationException("Not supported yet.");
97      }
98      //-------------------------------------------------
99      public void updateCustomer() {
100         //Write codes to update the employee
101         throw new UnsupportedOperationException("Not supported yet.");
102     }
103     //-------------------------------------------------
104     public void deleteCustomer() {
105         //Write codes to remove the employee
106         throw new UnsupportedOperationException("Not supported yet.");
107     }
108     //-------------------------------------------------
109     public void findCustomerById() {
110         //Write codes to get the employee by Id
111         throw new UnsupportedOperationException("Not supported yet.");
112     }
113     //-------------------------------------------------
```

```java
    public void searchCustomers() throws Exception {
        int choice;
        boolean stop = true;
        String value;
        try {
            do {
                Menu.print("1.Search by Name|2.Search by last order date|3.Back|Select:");
                choice = DataInput.getIntegerNumber();
                switch (choice) {
                    case 1 -> {
                        value = DataInput.getString("Enter customer name:");
                        List<Customer> customers = searchCustomerByName(value);
                        if (!customers.isEmpty()) {
                            printList(customers);
                        } else {
                            System.out.format("The customers with name:%s not found.%n", value);
                        }
                    }
                    case 2 -> {
                        LocalDate lastOrderDate = DataInput.getDate("Enter last order date:");
                        List<Customer> customers = searchCustomerByLastOrderDate(lastOrderDate);
                        if (!customers.isEmpty()) {
                            printList(customers);
                        } else {
                            DateTimeFormatter formatters = DateTimeFormatter.ofPattern("dd/MM/yyyy");
                            System.out.format("The customers with last order date:%s not found.%n",
                                    lastOrderDate.format(formatters));
                        }
                    }
                    case 3 -> {
                        stop = false;
                    }
                    default -> {
                        System.out.println(">>Choice invalid");
                    }
                }
            } while (stop);
        } catch(Exception e) {
            System.out.println(">>Error:" + e.getMessage());
        }
    }

    //-------------------------------------------------
    public void printList(List<Customer> customers) throws Exception {
        DateTimeFormatter formatters = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        System.out.format("%-10s | %-20s | %-20s | %-19s | %s%n",
                "Customer Id", "Customer Name",
                "Address", "LastOrderDate", "Amount");
        System.out.println(String.join("", Collections.nCopies(95, "-")));
        for (Customer customer : customers) {
            System.out.format("%-11s | %-20s | %-20s |%-20s |%.2f%n",
                    customer.getId(), customer.getName(),
                    customer.getAddress(),
                    customer.getLastOrderDate().format(formatters)
                    ,customer.getAmount());
        }
        System.out.println(String.join("", Collections.nCopies(95, "-")));
    }
```

```
172         //-----------------------------------------------
173         public void exportToFile() throws Exception {
174             //Write codes to export the employee list to tthe file
175             throw new UnsupportedOperationException("Not supported yet.");
176         }
177        //-----------------------------------------------
178         public List<Customer> searchCustomerByName(String value) throws Exception {
179            //Write codes to search the employee by keyword of the name and
180            //result sorted by name ascending (case insensitive)
181            //if two customers have the same name then sort by amount descending
182            throw new UnsupportedOperationException("Not supported yet.");
183         }
184         //-----------------------------------------------
185         public List<Customer> searchCustomerByLastOrderDate(LocalDate value) throws Exception {
186             //Write codes to search the employee by LastOrderDate and
187             //result sorted by LastOrderDate ascending
188             //if two customers have the same LastOrderDate then sort by amount descending
189             throw new UnsupportedOperationException("Not supported yet.");
190         }
191         //-----------------------------------------------
192         //Write more the methods here.........
193     }
```

Step 06. Write codes for the classes of the **Presentation** package as follows:

- **Menu.java:**

Menu.java ×

Source | History

```
1   package Presentation;
2
3   import BusinessObject.EmployeeManagement;
4   import BusinessObject.CustomerManagement;
5   import Utilities.DataInput;
6   import java.util.Arrays;
7   import Core.Interfaces.ICustomerDAO;
8   import Core.Interfaces.IEmployeeDAO;
9   /**
10   * @author SwordLake
11   */
12  public class Menu {
13      public static void print(String str) {
14          var menuList = Arrays.asList(str.split("\\|"));
15          menuList.forEach(menuItem -> {
16              if (menuItem.equalsIgnoreCase("Select")) {
17                  System.out.print(menuItem);
18              } else {
19                  System.out.println(menuItem);
20              }
21
22          });
23      }
24      //-------------------------------------------------------
```

```
25      public static int getUserChoice() {
26          int number = 0;
27          try {
28              number = DataInput.getIntegerNumber();
29          } catch (Exception e) {
30              System.out.println(e.getMessage());
31          }
32          return number;
33      }
34       //-------------------------------------------------------
35      public static void manageEmployee(IEmployeeDAO service) {
36          EmployeeManagement empMenu = new EmployeeManagement(service);
37          empMenu.processMenuForEmployee();
38      }
39       //-------------------------------------------------------
40      public static void manageCustomer(ICustomerDAO service) {
41          CustomerManagement cusMenu = new CustomerManagement(service);
42          cusMenu.processMenuForCustomer();
43      }
44       //-------------------------------------------------------
45  }
```

- **Program.java:**

Program.java

Source | History

```
1   package Presentation;
2
3   import Utilities.DataInput;
4   import DataObjects.CustomerDAO;
5   import DataObjects.EmployeeDAO;
6   import Core.Interfaces.ICustomerDAO;
7   import Core.Interfaces.IEmployeeDAO;
8
9   public class Program {
10      public static void main(String[] args) {
11          int choice;
12          String employeeDataFile = "Employee.txt";
13          String customerDataFile = "Customers.txt";
14          try {
15              do {
16                  System.out.println("***************Main Menu***************");
17                  Menu.print("1.Employee Management|2.Customer Management|3.Exit|Select:");
18                  choice = DataInput.getIntegerNumber();
19                  switch (choice) {
20                      case 1 -> {
21                          IEmployeeDAO employeeService = new EmployeeDAO(employeeDataFile);
22                          Menu.manageEmployee(employeeService);
23                      }
24                      case 2 -> {
25                          ICustomerDAO customerService = new CustomerDAO(customerDataFile);
26                          Menu.manageCustomer(customerService);
27                      }
```

```
28                      default -> {
29                          System.out.println("Good bye !");
30                          System.exit(0);
31                      }
32                  }
33              } while (true);
34
35          } catch (Exception ex) {
36              System.out.println(ex.getMessage());
37          }
38      }
39  }
```

Step 07. Run the project and test all the functions.

```
Output - MyApplication (run)
******Employee Management******
1.Add Employee
2.Update Employee
3.Remove Employee
4.Search Employees
5.Print Employee List
6.Export to file
7.Back to main menu
Select :
4
1.Search by name
2.Search by salary
3.Back
Select:
1
Enter the name:an
Employee Id | Employee Name        | DateOfBirth          | Email                | Salary
-------------------------------------------------------------------------------------------
E001        | Tran Gia Hong        | 01/01/2003           |hong@gmail.com        |1500.00
E002        | Do Anh Khoa          | 13/04/2002           |khoa@hotmail.com      |2500.00
E004        | Tran Van An          | 29/12/2002           |antv@live.com         |2891.00
E005        | Pham Hoang Anh       | 12/12/2001           |anhdh@gmail.com       |1000.00
E006        | Tran Gia Hong        | 12/12/2001           |hong@gmail.com        |1000.00
E007        | Dinh The Luan        | 01/01/2002           |luan@gamail.com       |8300.00
-------------------------------------------------------------------------------------------
1.Search by name
2.Search by salary
3.Back
Select:
```